2014-08-20

Performance Assessment of Model-Driven FPGA-based Software-Defined Radio Development

Matthew S. Allen
Worcester Polytechnic Institute

Performance Assessment of Model-Driven FPGA-Based Software-Defined Radio Development

by

Matthew S. Allen

A Thesis
Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Master of Science
in
Electrical and Computer Engineering
by

August 2014

APPROVED:

Professor Alexander Wyglinski, Research Advisor

Professor R. James Duckworth

Professor Susan Jarvis

Abstract

This thesis presents technologies that integrate field programmable gate arrays (FPGAs), model-driven design tools, and software-defined radios (SDRs). Specifically, an assessment of current state-of-the-art practices applying model-driven development techniques targeting SDR systems is conducted. FPGAs have become increasingly versatile computing devices due to their size and resource enhancements, advanced core generation, partial reconfigurability, and system-on-a-chip (SoC) implementations. Although FPGAs possess relatively better performance per watt when compared to central processing units (CPUs) or graphics processing units (GPUs), FPGAs have been avoided due to long development cycles and higher implementation costs due to significant learning curves and low levels of abstraction associated with the hardware description languages (HDLs). This thesis conducts a performance assessment of SDR designs using both a model-driven design approach developed with Mathworks HDL Coder and a hand-optimized design approach created from the model-driven VHDL. Each design was implemented on the FPGA fabric of a Zynq-7000 SoC, using a Zedboard evaluation platform for hardware verification. Furthermore, a set of guidelines and best practices for applying model-driven design techniques toward the development of SDR systems using HDL Coder is presented.

Acknowledgements

I would first like to acknowledge my advisor Professor Wyglinski for guidance and support durin..g the thesis research. I would like to acknowledge and thank Raytheon for their financial support, for which this thesis would not be possible. I would like to thank Stephen Freitag for the guidance as the Raytheon liaison. I would also like to thank members of the thesis defense committee to include Professors Jim Duckworth and Susan Jarvis. Finally, I would like to thank Mathworks for the license to allow testing of HDL Coder.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Over the last century technological advancements in the area of electrical and computer engineering have improved at an ever increasing rate. Transistors have decreased in size each year following Moore's Law, processors have included more advanced instruction sets and added multiple cores, programming languages and their compilers have become more advanced, and HDL design suites are capable of generating advanced cores. Each improvement in hardware and software adds to the productivity of engineers, enabling them to accomplish more complex tasks at a much faster rate.

Although core generation tools have helped decrease development cycles, the HDL languages such as VHDL and Verilog have remained the primary method for programming FPGAs for many reasons. One of the most common reasons is the requirement for efficient design that demand low data latency and minimal hardware utilization, which until recently model-driven tools had not been able to compare with the HDL languages. Although there are many benefits associated with HDL languages, there are several disadvantages as well. The disadvantages of either VHDL or Verilog are: steep learning curves, long development cycles, and portability issues between FPGA architectures.

With a recent increased level of interest in model-driven design, the arrival of High Level Synthesis (HLS) has added another level of abstraction by allowing engineers to program in languages such as C, C++, or even MATLAB. By deemphasizing timing details and focusing on algorithmic details, this abstraction helps lower development time and improves compatibility between platforms, thus increasing productivity. These increases are due to writing higher level code that will later be converted to Register-Transfer Level (RTL) implementations and tested.

Academia and industry could both benefit from model-driven design or high level synthesis tools for FPGA-based development. Reduction in development time for prototypes is one area that each would benefit from the use of these tools. Potentially, allowing prototypes to be developed at a faster rate for use as demonstrations to sponsors or customers. Model-driven tools could also provide an opportunity for hardware acceleration of complex tasks, which are normally executed on computers or servers. These tasks could be transferred onto FPGA hardware, potentially allowing faster computation by taking advantage of reconfigurability and the ability to create parallel architectures. Software-defined radio development cycles would benefit as well by allowing easier access to the FPGA fabric onboard most commercial FPGA platforms. This access would be perfect for the physical layer implementation, allowing the CPU to handle higher levels of the radio with less latency in the design.

The goal of this project was to investigate, analyze, and document the differences between a model-driven design approach and a hand-optimized approach of the same design, as applied to a software-defined radio built on an FPGA fabric. The model-driven design was accomplished by the use of the Mathworks HDL Coder software package to create VHDL code. The hand-optimized design made improvements of the model-driven VHDL code by removing excesses in hardware usage and delays. Both designs were then analyzed by comparing many subject areas including hardware utilization, timing constraints and delays, and implementation time, among others.

1.2 State-of-the-Art of Model-Driven Design and High Level Synthesis

Although model-driven design and high-level synthesis techniques for FPGA design have failed to gain wide adoption in the past, there have been many lessons learned in the process. Kong et al. [1], discusses these lessons learned and attributes the failings of wide adoption of HLS tools for several reasons to include the lack of:

- Comprehensive design language support
- Reusable and portable design specification
- Narrow focus on data path synthesis
- Satisfactory quality of results (QoR)
- Compelling reason or event to adopt a new design methodology.

Starting in 2000, a new generation of HLS tools appeared addressed many of the issues that had previously plagued the FPGA development community.

HLS tools using the C or C++ programming languages have emerged as top competitors in the modern HLS tools market, adopting language extensions and constructs to mitigate the problems with timing and dynamic constructs [1]. These tools typically employ a systems level approach to hardware design by allowing the compiler to interpret basic composition, adding a level of abstraction and much faster time to market. Another approach to HLS has been model-driven design tools that use a graphical user interface and allow predefined complex algorithms to be available as drag and drop blocks. Several model-driven tools offer many options for communications and DSP functional blocks due to their generic implementation structures, such as a basic FIR filter. Additional settings allow each block to be customized in a GUI environment. Although many model-driven tools provide solutions for communications and DSP structures, they do not necessarily address unique design applications for hardware acceleration or massively parallel programming efforts. Some of the HLS tools have been designed to work directly with the design suites of Xilinx and

Altera by adding many benefits such as hardware-in-the-loop cosimulation citeHDLCoder. Many of the modern HLS tools incorporate fixed-point and floating-point data types for most of the common algorithmic operations [1] [2] [3].

The current HLS tools that offer the most efficient FPGA prototyping/development include MATLAB's HDL Coder [2], Xilinx's HLS tool [4], Altera's SDK for OpenCL [5], Agilent SystemVue [3], Xilinx's System Generator for DSP [6], and Altera's DSP Builder [7]. These HLS tools not only offer efficient and quicker development cycles that many other HLS tools have as well, but they also offer many tools to aid in testing and verification that make them ideal choices. Xilinx's HLS tool and Altera's SDK for OpenCL are both C/C++ based HLS tools. However, each option targets a different design community and intended market. Xilinx's HLS tool is directed toward the hardware engineering community, while Altera's SDK for OpenCL is directed toward the software programming community [8]. Xilinx's System Generator for DSP and Altera's DSP builder use the Simulink environment with specialized blocksets that reduce communications and DSP applications development cycles tremendously. MATLAB's HDL Coder can use MATLAB code, Simulink models, or Stateflow models to create efficient high-level synthesis designs, and can also be used with System Generator for DSP and DSP builder. Finally, Agilent SystemVue specializes in the design of communications physical layer design with a GUI environment that supports specialized blocks, C++, MATLAB code, or HDLs, and can provide simulation references for verification of some of the most recent communications standards such as LTE/LTE-A, IEEE 802.11 AC/AD, or custom OFDM.

1.3 Approach

The primary objectives and approach of this thesis included:

- Identify software solutions for converting high level synthesis or model-driven designs into HDL code, and investigate the pros and cons of each.
- Utilize an off-the-shelf software suite to convert high level synthesis or a model-driven

design into HDL for targeting an FPGA as a component of a software-defined radio system.

- Quantitatively assess the performance of a system developed with both model-driven and hand-coded techniques.

1.4 Thesis Contributions

This research contributes the following to the communications system engineering community:

- An assessment of current state-of-the-art practices in applying model-driven development techniques toward the development of SDR functional components in FPGAs.
- A thorough analysis of implementations of components of a SDR in FPGA fabric using both model-driven development and traditional hand-coded methods.
- A set of guidelines and best practices for applying model-driven development techniques toward the development of SDR functional components in FPGAs using HDL Coder.

1.5 Document Organization

The organization of this document is split into five chapters. Chapter 2 first discusses FPGAs, how they are designed, and their modern advancements. The second subsection discusses model-driven development tools, the HDL design flow, framework languages, descriptions of current model-driven development tools, and comparison of model-driven development tools. Chapter 3 describes the design approach for developing and testing an SDR platform consisting of a Zedboard with Zynq-7000 FPGA using MATLAB's HDL coder. Chapter 4 describes the implementation of BPSK model-driven and hand-optimized designs, a QPSK model-driven design, and the integration of MATLAB Coded blocks as well as existing HDL into a model-driven design. Chapter 5 discusses the performance assessment of a model-driven design as compared with a hand-optimized design, as well as

the best practices for using HDL Coder for model-driven design. Chapter 6 discusses the conclusions obtained from the performance assessment and project, as well as future work to be performed on the subject.

Chapter 2

Overview of Field Programmable Gate Array Technology

2.1 Field Programmable Gate Arrays (FPGAs)

A Field Programmable Gate Array (FPGA) is one type of Programmable Logic Device (PLD) that was first invented by Xilinx in the mid 1980s [9]. Since then, FPGAs have become one of the most prominent PLDs available and have significantly cut into the Application-Specific Integrated Circuit (ASIC) and General-Purpose Processors (GPP) markets. Much like an ASIC, an FPGA can implement specific digital circuits to meet timing requirements of very high bandwidth applications that a typical processor cannot handle. Similar to a processor, an FPGA can be reprogrammed or reconfigured to update the desired functionality, which is not possible with an ASIC. Although FPGAs previously were less likely to be used in industry, there are many reasons that FPGAs have become more desirable and widely used today. Although an FPGA will not compare to the power consumption of an ASIC with the same design, the performance per watt of an FPGA today is better than that of a CPU [10]. Compared again with both an ASIC and a processor, FPGAs are able to be fully reconfigurable without any additional expense (as compared with an ASIC), and are now capable of reconfiguration on-the-fly (as compared with a processor). Additionally, todays FPGAs now contain core generators that can replicate advanced logic

circuits without needing to take time to reproduce it by hand. For all of these reasons, an FPGA is an ideal choice for projects with reconfigurability and circuit timing requirements in mind.

2.1.1 Basic FPGA Structure

Currently there are many different choices for resources and specifications of FPGAs. However, they all have a basic elements in common. Most FPGAs are composed of input/output blocks, configurable logic blocks, programmable interconnects, block RAM, DSP slices, and digital clock management components. Although most FPGAs consist of these basic elements, each element uses different structures between different FPGAs. These structures have evolved, adapting to the needs of industry or application of each FPGA. Typically, these basic structures are more abstract during the RTL generation process in order to focus on the descriptions of hardware. However, during the implementation step, resource utilization becomes apparent.

Configurable logic blocks (CLBs), also called Logic Array Blocks (LABs) for Altera FPGAs, are used to create logic, arithmetic, or ROM functionality. Within a CLB are multiple logic slices, and within each logic slice are multiple logic cells. A logic cell typically consists of Look Up Tables (LUTs), multiplexers, full adders, and registers (flip-flops) to perform the operations of the intended logic design. This structure has been established as a layered approach to allow the CLB level more flexibility in programmable interconnects but slower connections, while the logic cell level has restricted programmable interconnects but faster connections [11]. Within the logic cells, LUTs are used for combinational logic or ROM by using the inputs as addressable lookups into a table of stored outputs. This structure can replace several layers of logic gates while reducing the propagation delay to a single LUT. Modern LUTs can be considered "fracturable" as well, meaning that they can be used to break a 6 input LUT with one output into two 5 input LUTs with two outputs. However, this requires the use of common addressing due to hardware constraints. Modern design suites can also piece together several LUTs to be used for logic requiring more inputs than

a single LUT can provide. Typically the inputs to a logic cell starts at the LUTs (with the exception of the carry in to the full adders). After the LUTs the outputs will then enter or bypass the full adders using the multiplexers. The outputs of the logic cell are then fed directly to a combinational output or through a register for a registered or clocked output, allowing the design to use either.

DSP slices are an important aspect to many complex signal processing designs and are an essential element of modern FPGAs to perform pipelined parallel algorithms, such as multiply and accumulate (MAC) operations used for filtering and FFTs. DSP slices can also be used for operations other than DSP, such as wide dynamic bus shifters, memory address generators, wide bus multiplexers, and memory-mapped I/O registers [12]. Each manufacturer has a slightly different build, but a Xilinx DSP slice contains a pre-adder, multiplier, a Single-Instruction-Multiple-Data (SIMD) arithmetic unit, a bitwise logic unit, and a pattern detector. Multiple DSP slices are contained in a unit called a DSP tile and share an interconnect allowing the slices to be cascaded to implement larger fixed point DSP algorithms with faster connections. Using high level design programs such as Xilinx's System Generator allows the DSP slice to accommodate fixed point, single-precision floating point, double-precision floating point, or even customized precision operations, depending on the dynamic range and precision necessary for each design.

The Input/Output Blocks (IOBs) provide programmable placement for inputs and outputs of the FPGA via pins to fit both physical PCB layout and electrical voltage level design constraints. Block RAM typically are synchronous true dual port memories able to act as RAM, ROM, FIFOs, and shift registers depending on design. Data within the block RAMs can be stored or erased during the bitstream programming for these devices. Block RAM write mode can accommodate either reading or writing before both a simultaneous read and write operation allowing a highly customizable usage. One of the most important parts of an FPGA is the programmable interconnect, which allows the device to connect all of the internal elements together.

2.1.2 Modern FPGA Advancements

The first production FPGA was the Xilinx XC2064, originally called a "Logic Cell Array", that contained 3 types of configurable elements: input/output blocks, logic blocks, and interconnect. The specifications of this device included 1200 logic gates, 64 Configurable Logic Blocks (CLBs), 58 user inputs and outputs, and the claim "fully field-programmable" [13]. This original Logic Array Cell required the programming by Xilinx's XACT Development System, which could accomplish schematic entry, automatic place and route, and interactive timing among other necessary functions. Since the mid-eighties when the FPGA was first introduced, many variations have been developed that have improved upon the original design, from several different companies. Two distinct advancements that show promise and further improvement of FPGAs include the integration of System on a Chip (SoC) and partial reconfigurability.

System on a Chip (SoC)

SoC, or system on a chip, is the inclusion of a processor, peripherals, memory, and other hardware to create a single chip system. FPGAs have had SoC for many years. However, recent decreases in cost, increases in use of FPGAs in embedded systems, and the manufacturing costs associated with following Moore's law have created conditions for SoCs to be more widely used. A shift toward parallel computing has recently gained attention for FPGAs to be used as hardware accelerators for CPUs. With the exceptions of cell phones, PCs, and tablets, producing fixed function semiconductors is becoming no longer economically justifiable for their return on investment. With the decreases in cost of FPGAs, the emergence of SoC FPGAs, and SoCs potential for markets, SoC FPGAs will likely be a target for an increase of investment [14].

Typical SoCs contain a processing system and programmable logic. Processing systems typically include features like DDR3/NAND/NOR memory support, UART, I^2C, SPI, USB, Gigabit Ethernet, Secure Digital (SD), encryption schemes (i.e., AES, RSA, and SHA-256),

among many others. Using a high bandwidth interconnect, these functions along with the processor can be used with the FPGA fabric programmable logic to create a high functioning system. Allowing a processor to utilize the inherent parallel nature of FPGA fabric for hardware acceleration.

Several examples of modern SoCs include the following:

- **Xilinx Zynq-7000 Family:** 6 devices each containing dual ARM cortex-A9 MPCore processors with clock frequencies upto 1 GHz. 3 devices contain Kintex-7 fabric and 3 devices with Artix-7 fabric [15].
- **Altera Cyclone V SoC Family:** Low-range SoC with 700 MHz or 925 MHz, single (SE variant) or dual (SE, SX, ST variant) ARM Cortex-A9 MPCore processors, and 10 configurations in 3 variants (SE, SX, ST) [16].
- **Altera Arria V SoC Family:** Mid-range SoC with 1.05 GHz dual-core ARM Cortex-A9 MPCore processor, and 4 configurations in 2 variants (ST, SX) [17].
- **Altera Arria 10 SoC Family:** Mid-range SoC with 1.5 GHz dual-core ARM Cortex-A9 MPCore processor, and 7 configurations [18].
- **Altera Stratix 10 SoC Family:** High-range SoC with 64 bit quad-core ARM Cortex-A53 processor [19].

Partial Reconfigurability

Partial reconfiguration is the ability to dynamically reprogram sections of an FPGA without interruption of normal operation of the main design. This is accomplished by the ability to download partial bitstreams on-the-fly. Although FPGAs are increasing in size and can accommodate most designs, there are many practical reasons for using partial reconfigurability. A few of the major reasons for partial reconfigurability are cost, reducing power consumption, or mission critical applications [20]. Choosing a less expensive FPGA for a mass production design can benefit from reconfigurability by allowing hardware only

used for a short duration to be overwritten or replaced after operation. In some applications power consumption can be very important. Choosing a smaller FPGA would be ideal for situations where power consumption and reprogrammability are important design constraints (without reprogrammability an ASIC would be the better choice). Applications where time-sharing of resources is desired and disruption of operation is not an option. Thus, partial reconfiguration enables an engineer to replace hardware that is only necessary for a short duration, on-the-fly. This method can possibly achieve maximum usage of overall FPGA hardware and minimum usage of time dependent functionality. Another benefit from partial reconfigurability for large applications and FPGAs is faster system startups for systems that cannot tolerate slower start times. An example of a use of partial reconfigurability would be software-defined radios, where switching between many different modulation schemes may be important and power consumption is an issue.

With the Altera Quartus II software suite, partial reconfiguration is implemented by either flash parallel X16 interface, internal processor, or through external interface [21]. Partial reconfiguration is currently supported by these devices:

- **Xilinx Products:** Virtex-4, Virtex-5, Virtex-6, Virtex-7, Kintex-7, Artix-7, and Zynq 7000 families [22].
- **Altera Products:** Cyclone V and Stratix V families [21].

2.2 FPGA Model-Driven Development Tools

Since the late 1980s, the HDLs of VHDL and Verilog have remained dominant languages for programming FPGAs. These two hardware description languages rely on a low register transfer level descriptions of the processes desired. Difficult learning curves and long implementation times have led programmers away from these languages and FPGAs in general. However, at the same time the number of programming languages for processors have drastically increased. They have become increasingly abstract, such as memory management, allowing programmers to focus on higher level design. The compilers have adopted many

standards and have become much more versatile, as advanced instruction set architectures have led to less lower level programming. These vast improvements and higher level of abstraction has led to increased productivity, less errors in design cycle, and have decreased time to market or for prototyping.

Since the first FPGA, FPGA hardware has improved significantly with the volume of gates, reduction in power consumption, inclusion of advanced slices (DSP, etc), core generation, and hard core processors. With drastic increases in hardware, there has been a need to develop a set of more productive model-driven tools that increase the level of abstraction to essentially allow an increase in productivity. While this need has been filled by the advent of several different model-driven tools, there has not been many standardization efforts with respect to these tools. These tools come with many different programming styles, the most popular being the C programming language. Due to the hardware development improvements, new SoC designs, and overall specifications of modern FPGAs, industry and academia have many reasons for choosing an FPGA over processors or ASICs. Now would be the best time for industry and academia to shift into high level synthesis and model-driven design tools. Before this is possible, there needs to be a review of current model-driven tools. With a shift to model-driven tools will lead innovation in the way of standardization and fast prototyping with FPGAs.

2.2.1 Design Flow of Hardware Description Languages

Before comparing current FPGA model-driven development tools, an explanation of the current design flow process should be described for comparison later with newer approaches. With hardware description languages, there exists a design flow with several steps that are necessary to take VHDL, Verilog, and user constraint files and convert them to bitstreams that can program an FPGA. For Xilinx design suites, these steps include the synthesis, translate, map, place and route, and programming file generation.

The synthesis step will analyze the VHDL, Verilog, and user constraint file, optimizes

the design based on the hardware of the device, and creates netlist files. The translate step takes the netlists and merges them with the constraints into a design file that describes the design as basic FGPA elements. The map step maps the basic FPGA elements design to physical hardware components included on the device. The place and route step actually determines placement and routes for the design. The programming file generation step produces the bitstream that can be downloaded onto the device [23].

The processes mentioned above are necessary for any design to be implemented on an FPGA, including high-level synthesis tools. However, the synthesis step is uniquely designed for the HDLs because it determines the underlying logic of the design as well as estimates timing. Some high-level synthesis tools convert higher level code down to HDL, requiring them to fit the design syntax for the basic hardware components, which is a difficult task.

2.2.2 Framework Languages of Model-Driven Tools

There are many different types of model-driven development tools to program FPGAs, both open sourced and proprietary, that add a higher level of abstraction and increase productivity. Bacon has categorized these frameworks into categories based on language/design flow. These categories include: HDL-like languages (SystemVerilog), C-based frameworks (C/C++/SystemC), CUDA (Compute Unified Device Architecture)/OpenCL (Open Computing Language)-based frameworks, high-level language based frameworks (MATLAB or Python), and model based frameworks (MATLAB's Simulink and Stateflow charts) [10]. Below are brief descriptions of several of the high-level languages or constructs that are used in modern model-driven design tools, including the following:

- **SystemVerilog:** SystemVerilog is built on top of the existing Verilog hardware language and was created to improve productivity, readability, and reuse. SystemVerilog has extensions to data types, operators, and procedural statements, as well as enhanced process control, tasks and functions, and classes. SystemVerilog incorporates

a substantial amount of C functionality and dynamic data types to create more compact higher functioning code [24].

- **SystemC:** Contains a subset of C++ library functions that acts as a system level design language for transaction level modeling of hardware. SystemC was created by the Open SystemC Initiative (OSCI) and has been approved by the IEEE Standards association as IEEE 1666-2005. SystemC can be compiled into an executable. However, it is not as efficient as Verilog or VHDL [25].

- **OpenCL:** Open Computing Language is an open source royalty-free standard based on the C programming language (C99) that has been designed for parallel programming of heterogeneous systems that can target CPUs, GPUs, DSPs, or FPGAs. The OpenCL construct allows for efficient and portable code that has a low-level hardware abstraction and a framework to support programming [26].

- **MATLAB:** Matrix Laboratory (MATLAB), is a high-level scripted programming language designed around computation and algorithm development for engineers and scientists. MATLAB has many built in libraries and tools for communications and signal processing that enable programming complex algorithms without worrying about memory management, and usually without data type [27].

- **Python:** A very powerful high-level scripting language that contains many functions similar to MATLAB. Python is considered an intuitive object object oriented language with high-level dynamic data types. Python has an extensive library and allows matrix computation and indexing without the need of loops [28].

- **Simulink:** A tool to develop models of algorithms using predefined blocks that adds a layer of abstraction. Simulink can also simulate designs with ease and allows for

exporting data for analysis. Using inputs to models and custom design, outputs can be viewed by many different methods and scopes. Simulink also allows for customized MATLAB code to be implemented using M blocks [29].

- **Stateflow Charts:** A tool for modeling and simulation of state machine functionality and logic based decision flow charts. The tool allows graphical representations to be incorporated into designs such as state transition diagrams, flow charts, state transition tables, and truth tables. These representations describe how the system should respond to inputs signals, events, and time-based conditions. The models can then be simulated and analyzed [30].

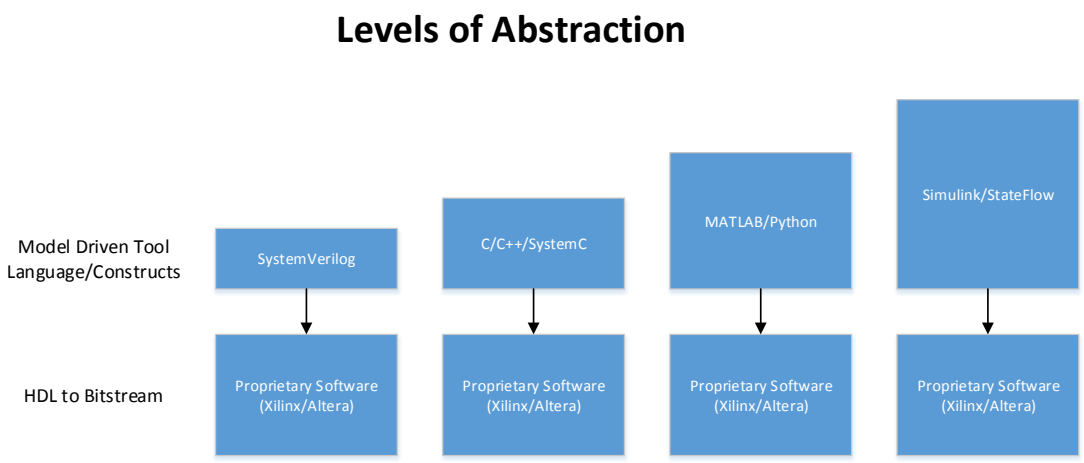

Figure 2.1: Diagram depicting the various levels of abstraction between the framework languages. Each different tool can then interface with proprietary software applications such as Xilinx ISE or Altera Quartus II to generate bitstreams that target specific FPGA devices.

2.2.3 Descriptions of Current FPGA Model-Driven Development Tools

Each model-driven tool has its own benefits, such as levels of abstraction, design structures, and engineering fields that would benefit from each tool. Therefore, an understanding

of each tool must be taken into account when deciding which tool is better for an individual application. Below are descriptions of current major model-driven design tools available.

Xilinx: Vivado HLS Tool and System Generator for DSP

The Xilinx Vivado Suite is a development environment designed with two tools for model-driven design that include the Vivado HLS (High Level Synthesis) Tool and the System Generator for DSP. Xilinx acquired AutoESL in 2010, and turned their product AutoPilot into their new HLS tool [31]. The HLS tool uses a system representation using the C, C++, and SystemC languages and converts it to hardware description languages (VHDL or Verilog). The HDL code can then be put into the Vivado Design suites physical-implementation flow to program Xilinx specific devices.

System Generator for DSP is a high-level tool in the Xilinix Vivado Suite that was developed for creating DSP systems for FPGAs. System Generator was also built so that a developer can build a DSP system in Simulink using the Xilinx Blockset that boasts bit and cycle accuracy. The System Generator will also automatically generate HDL code directly from Simulink, and has a power analyzer (XPA) integration that can allow for optimizing based on power constraints. One of the more impressive functions of the system generator is that it allows for co-simulation between hardware and Simulink using Ethernet or JTAG connections to ensure accuracy of the design. Documentation for the System Generator can be found in the System Generator User Guide [32].

The Xilinx Blockset is a group of predefined libraries of Simulink blocks containing many of the common DSP functions that are used today. Some of these blocks include: communications and DSP blocks (FFT, FIR filter, convolutional encoder, Viterbi encoder/decoder, Reed-Solomon encoder/decoder, CIC filter compiler, etc.), basic element blocks, control logic blocks, data type blocks, index block, math blocks, memory blocks, and tool blocks. All of these blocks are drag and drop in the Simulink environment with slight customization, that allow for quick design and HDL generation. The tool blocks included provide essential testing and verification to ensure a design meets the specifications that the designer

intended including a resource estimator, single-step simulation, ChipScope, FDATool (filter design tool), among several others. With the blocks and tools, the combination of MATLAB's Simulink allows the designer to easily test while building a high-level design at a faster rate [33].

Altera: SDK for OpenCL and DSP Builder

Altera's SDK for OpenCL (Open Computing Language) allows developers to write high level OpenCL C code that compiles kernels that can be implemented on an FPGA. In fact, the SDK allows for running the kernals on multiple FPGAs or even multiple boards to meet powerful computational needs. Altera's motivation was to use a new standard for writing code that allows FPGAs to work there way into mainstream computing by utilizing the powerful parallel construct of FPGAs. Using OpenCL also introduced a higher level programming language that can reduce the long development cycle of traditional coding in the HDLs for FPGAs.

OpenCL was ratified by the Khronos Group and has been contributed by many companies that all stand to benefit from a working standard and include: Apple, NVidia, AMD, Intel, IBM, Texas Instruments, ARM, Altera, Xilinx, Broadcom, Qualcomm, Ericson among many others [34]. A detailed explanation of OpenCL can be found in the OpenCL specification or at [26].

The OpenCL model uses a host that runs a host program, typically a CPU, and OpenCL devices that contain compute units and execute kernels. Each compute unit contains processing elements where the computation occurs. The OpenCL program running on the host issues commands to the processing elements as a single stream of instructions of Single Instruction Multiple Data (SIMD; identical set of instructions for each element) units or Single Program Multiple Data (SPMD ; possible different instruction execution for each element) units. The execution model can be described as having work-groups of work-items that operate across an N-dimensional index space called NDRange [26].

OpenCL is C code written in a highly parallelized fashion described similar to the program flow of CUDA. For FPGA implementation, the compiler of the FPGA would specify how to implement the kernel written for it. This removes the limitations of platform or vendor specific issues, as well as creates a highly parallel portable code perfect for implementation on an FPGA [26].

With HDLs, the developer must create cycle-by-cycle descriptions of hardware that is to be implemented on the FPGAs. With OpenCL, it is the job of the compiler to understand timing constraints of the external interfaces and design, and it is the job of the developer to design the algorithm itself. Using the C language with OpenCL constructs, Altera has created an environment for software engineers to enter a field traditionally held by hardware engineers allowing them to utilize the massively parallelized architecture of the FPGA [26].

Altera's DSP builder uses Simulink to create system level models of a design which can be converted to VHDL and TCL scripts for synthesis using the DSP Builder Signal Compiler block [7]. The Simulink models are built from Altera's DSP Builder and MegaCore blocks, and generates timing optimized HDL code. This method is similar to Xilinx's System Generator, and allows for modeling, simulation, and conversion to HDL using Simulink's model-driven approach for fast development cycles. IPs can be used with DSP builder as well, and many are available from the MegaStore website that work with DSP Builder. The DSP Builder Advanced Blockset contains many of the necessary algorithms to create high performance DSP functionality. Some highlights of the DSP builder include generation of resource utilization tables, fixed and floating point DSP, pipelining/TDM/close timing, and access to math.h functions [35].

Agilent SystemVue ESL

SystemVue ESL software is a systems-level software suite that specializes in the design of communications systems at the physical layer with a graphical user interface modeling environment similar to Simulink. SystemVue is sold as several software tools each specializing

in different areas. The ESL tools that apply to communications systems and FPGA development include: W1465 SystemVue System Architect [36], W1462BP SystemVue FPGA Architect [37], and W1461BP SystemVue Comms Architect [38]. SystemVue System Architect and SystemVue FPGA Architect both include SystemVue Comms Architect.

SystemVue is able to use multiple languages, such as their built-in graphical blocks, MATLAB code, C++, and fixed-point HDL, and generates HDL code or C++ for embedded DSP or HLS tools. SystemVue can also link to additional software such as Spectrasys to simulate RF systems and model non-linearities or RF effects that allows early detection of issues in the SystemVue environment. SystemVue can be used to implement floating-point or fixed-point models and has an drag-and-drop filter block that incorporates a digital filter design tool to create filters relatively fast. With SystemVue tools, designs can use co-simulation with the W1717 Hardware Design Kit, as well as Agilent test equipment can be directly programmed.

SystemVue System Architect in combination with SystemVue FPGA Architect can provide a powerful analytical design environment for the creation of powerful communications systems. The software suite provides many opportunities for testing during the development cycle. SystemVue provides HDL generation that is not device specific, however offers direct flow support for specific FPGA families.

Mathworks HDL Coder

The HDL Coder generates portable and synthesizable HDL code written with MATLAB functions, Simulink blocks, and Stateflow charts. First designs are modeled using MATLAB functions, Simulink blocks, and Stateflow charts. The models can then be optimized to meet area-speed requirements. Using the HDL Workflow Advisor, the models will then be converted to HDL code and can be verified using MATLAB's HDL Verifier.

Using MATLAB, the HDL Coder allows the developer to use MATLAB constructs and

system objects, as well as a library of basic logic elements to use, including counters and timers. The HDL Workflow Advisor also will automatically convert from floating point to fixed point during the process. The fixed point type can also be verified and adjusted to meet the desired accuracy of the fixed point design by simulation of both the floating point, fixed point, and error results. Using Simulink, the HDL coder allows the developer to create a model from a library of over 200 Simulink blocks that contain many complex operations for quick and easy integration into a design. The developer can change settings for optimizing HDL code, such as persistent array and matrix variables can be mapped to RAM, loops can be unrolled or streamlined, code can be pipelined with input, output, and distributed registers, and area optimizations such as resource sharing can be made. When the design is ready for FGPA integration, the HDL Workflow Advisor checks the Simulink model for compatibility. If it is compatible, it can generate HDL code or an HDL test bench.

Using HDL Verifier, the developer can automate two different types of co-simulations models. The first type co-simulation model is FPGA-in-the-loop that allows the developer to verify the correct functionality of a design in hardware by running test scenarios using Simulink over the gigabit ethernet interface of the target Xilinx or Altera FPGA board. The second type of co-simulation model uses Simulink to run test scenarios and then either Cadence Incisive or Mentor Graphics ModelSim and Questa for HDL simulation to test HDL code.

MATLAB has been integrated with Xilinx ISE and Altera Quartus II to allow the HDL Workflow Advisor to implement the designs directly onto the FPGA boards. Workflow Advisor can be used for synthesis and timing analysis, estimating usage of resources, and annotating critical path timing information. HDL Workflow Advisor will highlight possible bottlenecks in the design and will also show timing information that can be used for improvements in the design. During the automation process, HDL coder generates an HTML document of the HDL code with hyperlinks that can direct the developer between the HDL code and the original MATLAB code or Simulink blocks that it was generated from. At this stage, the user defined comments can be input for readability in the future. Lastly,

HDL Coder can be used to generate IP cores to use the generated HDL code at a later date. By specifying whether each input or output should be an external port or AXI4-Lite, the HDL code can be be generated to work with a processor or other hardware within the FPGA. MATLAB's HDL Coder can significant reduce design cycle time by allowing access to MATLAB's libraries of complex operations, performing critical design estimates, and allowing more control by the developer with ease.

Additional Model-Driven Tools Not Previously Discussed

Model-driven design or high level synthesis has been desired for many years, there have been a number of other tools created to accomplish these tasks. Some of these tools may specialize in areas that make them attractive to generate portions of a design using them, such as utilizing Pythons math functions for a Python based model-driven tool.

- **IBM Liquid Metal/Lime:** Heterogeneous programming language for programming hybrid computers consisting of multi-core processors, GPUs, and FPGAs [39].
- **myHDL:** Open source Python package for modeling and simulation/verification of designs in Python, as well as conversion from Python to HDL (Verilog/VHDL) [40]. myHDL is subject to limitations with using IP cores.
- **Dillon Engineering ParaCore Architect:** Paracore Architect's modeling language can be used to create highly parameterized IP cores. Dillon Engineering boasts fastest FFT/IFFT IP cores in the world [41].

2.2.4 Comparisons of Current FPGA Model-Driven Development Tools

Over several years many companies have created solutions for the next step of model-driven design for FPGAs. Many solutions are not well supported or funded to create market-wide solutions to the issues plaguing FPGA development. Most tools provide only a few solutions to the development cycle or functionally are inconsistent. Recent adoptions by the FPGA giants have created a path for future development. However, these paths are

new and pose the question of which is the correct one to adopt. This section will discuss the advantages of each, how they have been used, and suggestions for uses of model-driven tools.

MATLAB HDL Coder

Many engineers develop algorithms in MATLAB for verification because of its apparent mathematical computational strengths. Being a very powerful scripting language, MATLAB can clearly be used to develop algorithms very efficiently with integral step through verification of functionality. Utilizing one of the most widely used engineering test beds for algorithm development, taking one step further by designing algorithms to fit the HDL coder design parameters could save many hours of conversion to a different language. For one of the most economical moves toward model-driven design, using MATLAB's HDL coder would be a great option for engineers that already have the experience using MATLAB.

Altogether HDL coder is a very proficient and powerful tool that can reduce development cycles with engineers previously exposed or familiar with MATLAB, Simulink, or Stateflow. The advantages of MATLAB's extensive libraries of powerful functions could be a desirable resource to many engineers that would rather worry about higher level design.

Xilinx HLS Tool

The HLS tool originally debuted as the AutoPilot by AutoESL and was received very well by researchers and industry. It was not until recently when Xilinx purchased AutoESL that it will become a widely used and powerful tool for hardware engineers to utilize. An evaluation of the AutoPilot tool has shown that AutoPilot has been able to achieve high quality of results equivalent to hand-written RTL code, which has been a weakness for HLS tools in the past. [42] Cong found that the HLS tool produced a design of a sphere decoder that used between 11 percent and 31 percent less resources than a hand-coded design as well as improved productivity [1]. In an XCell Journal, the author Nathan Jachimiec highlights the ease of use and comparing situations for making changes to code that would take days

for Verilog, but takes only minutes in C code and did not require major modifications as it would in Verilog. More specifically, he mentions the ease of changing bit widths of FIFOs and RAMs, partitioning RAMs into smaller memories, reordering arithmetic operations, and utilizing dual-port instead of single-port RAM [43]. For all of these reasons, the HLS tool will be a valuable tool for industry.

OpenCL

With Altera's adoption, and Xilinx's future adoption, of OpenCL as a model-driven design component, along with the support and development from industry, OpenCL has become the clearest competitor as a widely-used standard for model-driven design of FPGAs. The two major feats of OpenCL being introduced into the FPGA development cycle are that it allows software engineers to become part of the design process and accelerates development of up to 15 times over HDL-based design flows [44]. With these achievements, design teams can use one of the most commonly used languages with a diverse team of hardware and software engineers to develop FPGA solutions at much faster paces. OpenCL also will likely solve the issues with FPGAs typically being avoided for faster design and prototyping needs, allowing industry to harness the benefits of the parallel architectures of FPGAs. Many advanced systems already reliant on FPGA architecture will benefit tremendously by reducing development cycles, which in turn reduces cost to companies, as well as adopting a more diverse design team that does not require the knowledge and design practices of the HDL languages, making it easier to hire and train within.

In an Altera white paper entitled "Implementing FPGA Design with the OpenCL Standard" [45], they show that an algorithm implemented using OpenCL across a CPU, GPU, and FPGA, was much faster and power efficient on the FPGA. The testing showed that while consuming one third the power of a CPU and one fifth the power of a GPU, the FPGA was able to perform 375 times the computation of a CPU, and 1.14 times the computation of the GPU. Ultimately the efficiency of the FPGA was 266 Million simulations per second per Watt, as compared to 48 for the GPU, and 0.0025 for the CPU [45]. Many papers have

shown similar results that high-end FPGA implementations of OpenCL code have better performance per watt and are faster than high-end CPUs and GPUs implemented with OpenCL [46].

MATLAB Simulink

For engineers primarily focused on DSP design such as the fields in communications, radar, video processing, among many others, Simulink would be a useful tool to learn. Being supported by both Xilinx's System Generator and Altera's DSP builder, Simulink can achieve a fast model-driven approach to creating complex signal processing algorithms with ease. Simulink contains so many advanced and complex functions, drag and drop makes development incredibly efficient. The integration and collaboration between Mathwork's Simulink, Xilinx, and Altera has made not only made Simulink one of the best approaches, but also one of the most well supported and funded DSP/FPGA solutions as well.

Portability and Interoperability of Different Model-Driven Development Tools

Many of the tools from the previous sections can be used in combination or have some interoperability to build a design for an FGPA. This section will discuss the portability and interoperability of the model-driven development tools.

With OpenCL designs will be portable between Xilinx and Altera FPGAs, as well as other OpenCL devices such as GPUs. The portability of code is not limited with OpenCL as long as there is a compiler that can target the individual FPGA vendor that can interpret the kernels. Since Xilinx and Altera make up 85 percent of the FPGA market and currently have the most cutting edge FPGA technologies, designs implemented in this new standard will most likely be able to move between vendors as the technologies improve. This allows for quicker product improvements and faster product development times. The design construct of OpenCL requires a host CPU to run the host program, so this is very efficient for SoC FPGA applications.

Xilinx's HLS tool will likely be one of the most target independent tools because it directly converts the C/C++/SystemC code to HDL code, allowing easy instantiation into different projects of different vendors. However, the original code is not portable between EDA environments because it is locked to Xilinx's HLS tool. The power of this tool may make this ideal for some applications where generating an IP core for lasting code may be necessary. For hardware engineers that typically do not have as much software experience may find the HLS tool the easiest and quickest transition compared with the OpenCL.

Although Altera and Xilinx each use Simulink to implement the DSP functions for their FPGAs, they each use different Simulink blocks to create them. Simulink models can be developed very fast and with ease, however if it is necessary to convert between these two vendors due to a new FPGA technology the designs must be converted by hand. This entails switching from the Xilinx blockset to the Altera blockset. There may be issues with this due to differences within the blocks.

Xilinx System Generator and HDL Coder can efficiently be used together in order to use Simulink blocks and Xilinx specific blocks. MATLAB has specialty Simulink blocks from the Communications System Toolbox and DSP System Toolbox that may not be accessible in the Xilinx blockset, therefore creating a Simulink model and converting it to HDL using the HDL Coder can be used in combination to Xilinx's System Generator or Altera's DSP Builder.

Possible Suggestions

Several possible suggestions and recommendations for uses of model-driven design tools are listed as follows:

- Due to the increased use of Simulink in HDL coder, Xilinx's System Generator, and Altera's DSP builder, it would be recommended that DSP engineers begin to learn

Simulink and transition to this design process. With such a strong DSP and communications blocksets, creating and testing advanced algorithms with Simulink will save time and money in the development cycle.

- Development teams with projects requiring strict timing constraints that would like to reduce development cycles should look to MATLAB HDL coder or Xilinx HLS tool. Both of these options are high-level synthesis tools, but require more understanding of the underlying hardware as compared with the OpenCL approach. This would be a smoother transition for hardware engineers who have the hardware expertise to evaluate the implementation and perform optimizations.

- Development teams focused on utilizing FPGAs for hardware acceleration and massively parallel processor architectures for complex algorithms should invest in OpenCL approaches with Altera's SDK for OpenCL and Xilinx's future OpenCL tool. OpenCL was designed from a software engineers perspective, and although derived from the C language, it requires a software design style.

- Agilents SystemVue would be a practical choice for engineering firms specializing in physical layer design of radio systems due to the strong RF simulation environments and simulation references of wireless standards.

2.3 Chapter Summary

Chapter 2 discussed FPGAs and model-driven development tools. Much like a tutorial, this chapter discussed the modern day structures of FPGAs, the modern FPGA advancements such as SoC and partial reconfigurability, and current FPGA technologies by comparing FPGAs and applications. Model-driven development tools were also discussed by outlining the design flow of the hardware description languages and describing the various framework languages that they are composed of. The FPGA model-driven development tools were described and included MATLAB's HDL Coder, Xilinx's HLS Tool and System Generator, Altera's SDK for OpenCL and DSP Builder, and Agilent's SystemVue software. Finally, the tools were compared showing that each had there own benefits, and descriptions

of the portability and interoperability were made as well as possible suggestions for uses of each.

Chapter 3

Design Approach

The primary goal of the design aspect of this project was to evaluate model-driven development tools as applied to the development of software-defined radios. This was accomplished by a detailed performance assessment of both a model-driven design and a hand-optimized design of a basic SDR system implemented on an FPGA-based SDR platform. An evaluation was then conducted to determine the performance, efficiency, and differences of a model-driven design as compared with the hand-optimized design. The sections of this chapter describe how the designs were implemented and how the performance assessment was conducted.

3.1 SDR Design Specifications and Implementation

The architecture and framework of the project was based on the hardware of a ZedBoard academic edition with a Zynq-7000 SoC. The Zedboard was chosen due to the increased use in academia and projects, support and forums, and its integration with the Analog Devices FMComms1 RF front-end. An experimental design study was conducted to determine the feasibility of including the Analog Devices FMComms1 board as a RF front-end in the project, which would allow the signal to transmitted and received over-the-air. However, it was determined that this inclusion would not be practical given the project time constraints due to a lack of development in the area that it would be applied for this project.

The software that was used to implement the SDR were Mathworks Simulink, Mathworks HDL Coder, Xilinx ISE 14.4, and Xilinx iMPACT. Simulink and HDL Coder was used to generate the VHDL code for the model-driven designed SDR system. The design used Simulink models, as opposed to MATLAB code, due to the higher level of abstraction that is more desirable for both industry and academia. The VHDL code of the model-driven designed SDR system generated by HDL Coder was then be rewritten and optimized to create the hand-optimized SDR system. Xilinx ISE was used with the VHDL coded designs to create testbenches for design performance assessments. Xilinx ISE was also used to integrate each model into a project and bitstream generation for programming the FPGA on the Zedboard. Each design was implemented in the fabric of the FPGA and verified for correct hardware operation.

SDR Design Components

The basic SDR design was implemented on the FPGA-based SDR platform and consisted of a digital data source, a transmit data chain, and receive data chain, as shown in Figure 3.1. The digital data source provided the data to the transmit data and receive

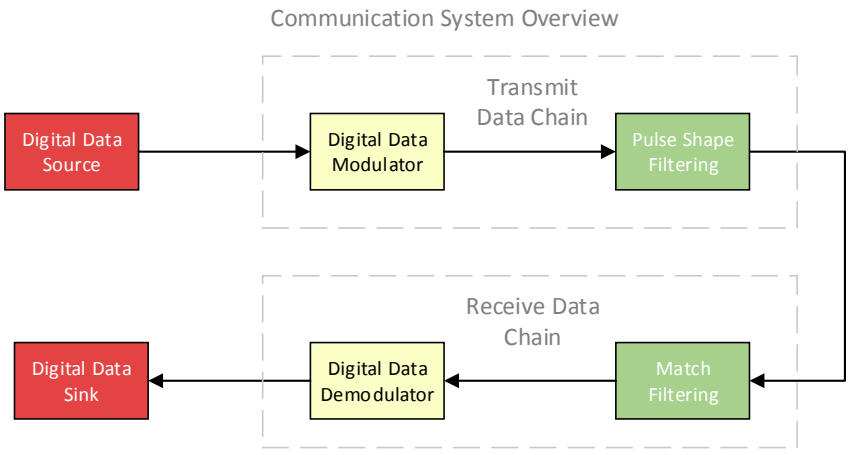

Figure 3.1: Overview of a basic communications system.

data chains. The transmit data chain consisted of a digital data modulator and a pulse shaping filter. The receive data chain consisted of a matched filter and a digital data demodulator. These SDR design components were chosen to implement a basic SDR system because they accomplish most of the necessary aspects of a basic communications system. Carrier recovery, timing recovery, equalization, and many other receive chain blocks were not included due to the complexities involved. The second reason they were not included was because they did not apply directly to the goal of this project. The modulation scheme chosen was Binary Phase-Shift Keying (BPSK), due to its simplicity and its availability as a Simulink block in HDL Coder. The pulse shaping and matched filters were chosen to be a square root raised cosine (SRRC) filter due to its availability as an FIR filter block in HDL coder, as well as meeting the desired Nyquist inter-symbol interference (ISI) criteria. Both of these choices were determined for their ease of implementation allowing more time for performance assessment or hand-coded optimizations. Table 3.1 specifies the choices for each component of the system.

Table 3.1: Table of communications system components.

Communications System Components	
Component	**Selection**
Digital Data Source	PN Sequence Generator
Digital Data Modulation	BPSK Modulator
Pulse Shape Filtering	FIR Interpolation Filter
Match Filtering	FIR Decimation Filter
Digital Data Demodulation	BPSK Demodulator

The HDL Coder does not automatically add registers within a block without declarations. Therefore, the placement of the registers was needed. A series of input pipeline registers were placed around each block for the purpose of preventing timing errors between blocks, with the exception of between the digital data source and the digital data modulator. Each filter required many adders and multipliers. Therefore, a register was placed between each adder and/or multiplier, effectively implementing a pipeline and adder trees. Both of these actions helped prevent timing errors and ensured setup times of the follow on registers was acceptable. The hand-optimized version did not result in exceptions to these rules, but

found excessive components and removed them for the leanest possible operations. Rollover of data values during the multiplication or addition stages was prevented in the design by allowing saturation instead. The purpose of this process was to preserve accuracy of the data at the expense of extra hardware. The register placement guidelines was maintained for the hand-optimized design as well.

During the project there were several different models created. The two most important models were the BPSK model-driven and BPSK hand-optimized designs. These were the primary models used for the performance assessment aspect of the project as described above. However, there were three additional Simulink models created. The first additional model converted the BPSK model-driven design to a QPSK system, specifically to show the ease of swapping components using Simulink. The second additional model determined a method of how to integrate MATLAB code into a Simulink model. The third additional Simulink model determined a method for how to integrate existing VHDL code into a design. These three models were produced and chosen specifically because development cycles for academia or industry may require using HDL Coder in several different ways. Sometimes there will be a need to integrate existing code or implement MATLAB code when there are no available blocks for the system. A description of how to best do both of these tasks was determined and described in Chapter 4.

Finally, in order to ensure correct operation of each of the two primary designs, both testbenches were written and simulated, and hardware verification was performed. A baseline for correct data was necessary to ensure testbenches were designed correctly, in order to verify the correct output sequences of the system.Therefore, the Simulink model that produced the model-driven VHDL design was used as the primary baseline because it simulated the correct operations of the system. Although the Simulink model was used as a baseline, it only produced the output of the system. Therefore a hand-coded MATLAB model of the system was created in order to allow for more dynamic testing situations. These situations included applying alternative inputs and testing single components for correct output. The hand-coded MATLAB system model was verified for correctness by comparison with the

Simulink model. Table 3.2 shows all of the models that were created, their languages and environments, and their purposes.

Table 3.2: Table of model design types created during the course of the project.

Model Design Types		
Model	**Language(s)/ Environment Used**	**Purpose**
BPSK Model-Driven Design	Simulink HDL Coder VHDL	Performance Assessment
BPSK Hand-Optimized Design	VHDL	Performance Assessment
QPSK Simulink Design	Simulink	Show Ease of Conversion
BPSK Simulink Design w/MATLAB Code	Simulink	Show Integration of MATLAB Code
BPSK Simulink Design w/Exisiting Code	Simulink	Show Integration of Existing Code
Hand-Coded MATLAB BPSK System Model	MATLAB	Testing/Performance Assessment

3.2 Performance Assessment

In order to determine the performance of the model-driven design, it was compared with a hand-optimized design. The following descriptions outline how the performance assessment was conducted.

The hand-optimized design did not use currently available core generation tools. Xilinx's Filter Compiler can produce both filters used in this project. However, this tool was not used for several reasons. Filter Compiler produces many excess controls that would have to be added to the BPSK Simulink model. These controls were difficult to determine due to the inability to see the base HDL code produced by the compiler, making verification difficult. The more important reason that the filter compiler was not used was that the primary purpose of the project was to determine the performance of HDL Coder with a hand-optimized approach, allowing a clear understanding of the excesses of the HDL Coder design. Using the filter compiler would be a direct comparison with other tools.

After development of each SDR system, hardware verification was performed with an error counter that tracked the number of errors that occurred in each system. The error counter was built to compare the output of the system with a delayed output of digital data source. The length of the shift register needed to implement the delayed data source was determined by simulations of testbenches, and was be applied to each error counter for their respective designs. The error counter was incremented when the values entering and exiting the system were different. The number of errors was displayed on the LEDs of the Zedboard.

Two separate resets were implemented for each designed system. The first reset was used for the SDR system, while the second reset was used for the error counter. On an error counter reset, several errors occurred until the first value from the data source propagated through both the error counter shift register and the SDR system. Once errors stopped occurring, the error counter was no longer enabled, which was reflected by the LEDs holding steady on a single value. An operational length of 3 seconds without any new errors observed on the LEDs was considered acceptable for hardware verification. Three seconds was justified due to a short maximum length of the sequence being generated (2.52 μS) and the continual stream of correct results after significant computation.

Resource utilization is a factor for model-driven design due to the expected lack of optimizations by model-driven tools. Therefore, resource utilization tables outlining the number of flip-flops, registers, DSP slices, and other hardware were generated for both designs, with the exclusion of the hardware verification error counter module. The resource utilization table was then compared and described for each model. Tables were also generated outlining the most widely used elements for each design such as the number and size of adders, multipliers, and bit reductions/saturation checks. These tables provide more insight to elements of each design and resource consumption. Finally, an explanation of design contributions to the differences of hardware utilization between the two models will be presented as well.

Timing analysis was performed in several ways such as the simulation of testbenches,

visual code inspection, modeling of system in MATLAB, and noting maximum allowable frequencies for each design. Simulations of created testbenches provided insight into the delays involved between each block, within each block, and for each system as a whole, and were be recorded. These checks were performed, compared, and described for both the model-driven and hand-optimized designs. Tables outlining the delays for each system and their components were created and presented as well. Finally, an explanation of design contributions to the differences of timing between the two models was presented as well.

Several testbenches were created to determine the correct operations, as well as the timing analysis of the system, and applied to each design. A testbench for the overall system was created to test the output of the system. This testing also included counting the number of delays induced by registers in the system by reading VHDL code and running simulations on ISim. The output was verified such that it was identical to the input of the system by means of assert statements, using the known input and delay. Testbenches were produced for the BPSK modulators and demodulators, for both designs. These testbenches applied a range of inputs and were verified, with the use of assert statements, for correct operation. The BPSK modulation has a binary 1-bit input, therefore only two values were checked. However, the demodulator had a larger input range and thus the full range was verified by applying numbers at the extremes and close to the decision regions. Two testbenches were developed to verify operation for the FIR Transmit filter. The first testbench applied an impulse to the filter to show the impulse response, which produced the output of the coefficients in order. This test used an assert statement to verify that each of the coefficients made it through the filter correctly. The second testbench created for the FIR transmit filter was a maximum value test. This test was developed to test a unique optimization made to the hand-optimized design, which is explained further detail in Chapter 4. The purpose of this test was to verify that the filter could correctly produce its maximum possible output after applying an input sequence that would cause this condition. The hand-coded MATLAB system model was used to determine the correct output, which was applied to the testbench in the form of assert statements. The last testbench created was for the FIR receive filter, which is very difficult to determine how to test. Therefore,

the input sequence was chosen to be the output of the FIR interpolation filter after reset. This input sequence was determined using the hand-coded MATLAB system model, which also produced the expected output used for comparison as well. This test ensured correct operation of the convolution produced by the filter and was verified using the assert statement. All test benches were simulated by ISim and visually inspected for correctness as well. Each testbench was adjusted for known pipeline register delays and applied to both the model-driven design and the hand-optimized design.

Implementation times and difficulty of development was recorded, compared, and assessed as well. The values determined were difficult to apply industry wide. However, they provide insight to the feasibility of using the current version of HDL Coder as a FPGA development tool for academia and industry. A series of best practices for model-driven design using HDL Coder was established and described that were derived from experience with HDL Coder over the course of the project. The primary purpose of established best practices will be to aid future FPGA development in academia and industry for better practices with design and productivity.

3.3 Chapter Summary

Chapter 3 discussed software-defined radio specifications and implementation objectives and how the performance assessment will be conducted. The SDR design specification and implementation section discussed the hardware components and software tools used to create the designs. This section also presented the SDR design components of the radio designs, the register placement for each design, and all of the model design types to be created. The performance assessment section discussed how hardware verification would be performed for the designs implemented on the hardware components. This section also detailed how the timing analysis would be conducted as well as outlines the testbenches to be created to analyze the timing information as well as simulate correct operations of each system and system components. Finally, this section described the analysis of implementa-

tion time and difficulty of development between the two designs, as well as an establishment of best practices to aid in future uses of HDL Coder.

Chapter 4

Implementation

In order to establish the effectiveness of a model-driven tool, such as HDL Coder, the tool must be compared with the method it must replace. One method of accomplishing this is developing two designs that produce the same results, one with the tool, and one without. This chapter discusses the implementation phase of developing two designs to be used for comparison. The first design used MathWorks HDL Coder tool to create a Simulink model of a BPSK model-driven communications system, which was then converted to VHDL using the HDL Coder Workflow Advisor. The process of creating the BPSK model-driven system, VHDL code generation, and the resulting VHDL code is presented. After the model-driven VHDL code was generated, a hand-optimized design was created. This BPSK hand-optimized design removed the inefficiencies of the HDL Coder design. Three more designs were created to show alternative methods for using the HDL Coder tool. These models were not used in the performance assessment. The first of these models presented shows the ease of block replacement by converting the BPSK Simulink model into a QPSK model. The second of these models presented shows the ease of developing a block using MATLAB code when a desired block is not available, or to replace developing HDL code by hand. The third and final of these models presented shows how to integrate existing HDL Code into a design developed with HDL Coder.

4.1 BPSK Model-Driven Design

The BPSK model-driven design was generated in several phases using Simulink, HDL coder, and Xilinx ISE. The first phase consisted of developing a communications model in Simulink to reflect the major functionality of the design using blocks that were supported by the HDL coder. The second phase consisted of using the HDL coder Workflow Advisor to generate VHDL code from the model. The third phase consisted of integrating the HDL coder generated VHDL code into Xilinx ISE project to generate the bitstream. The last phase involved downloading the bitstream to hardware for verification of expected functionality.

4.1.1 Model-Driven Simulink Design Description

The BPSK model that was designed contained a digital data source, transmitter chain, receiver chain with an output. The transmit chain consisted of a data source, a modulation block, and transmit pulse shaping filter. The receive chain consisted of a receive matched filter, and demodulation block. Each component of the Simulink design is described in detail as well as an explanation of the function of each component. Figure 4.1 shows the Simulink model created for this design.

Figure 4.1 shows the transmit chain consisting of the PN sequence generator as the data source, a BPSK modulator, and the FIR interpolation filter. The output of the transmit side of the communications model is labeled as "To_FMComms". This output represents the signal to be sent to a potential RF front-end for transmission, however was only used for testing. This signal is then looped back to the receive chain to be match filtered by the FIR decimation filter and demodulated by the BPSK demodulator. The output of the Simulink model labeled "To_DataSink" is the received data and is used for both HDL generation and hardware verification.

The data source was produced by selecting a PN sequence generator block from the

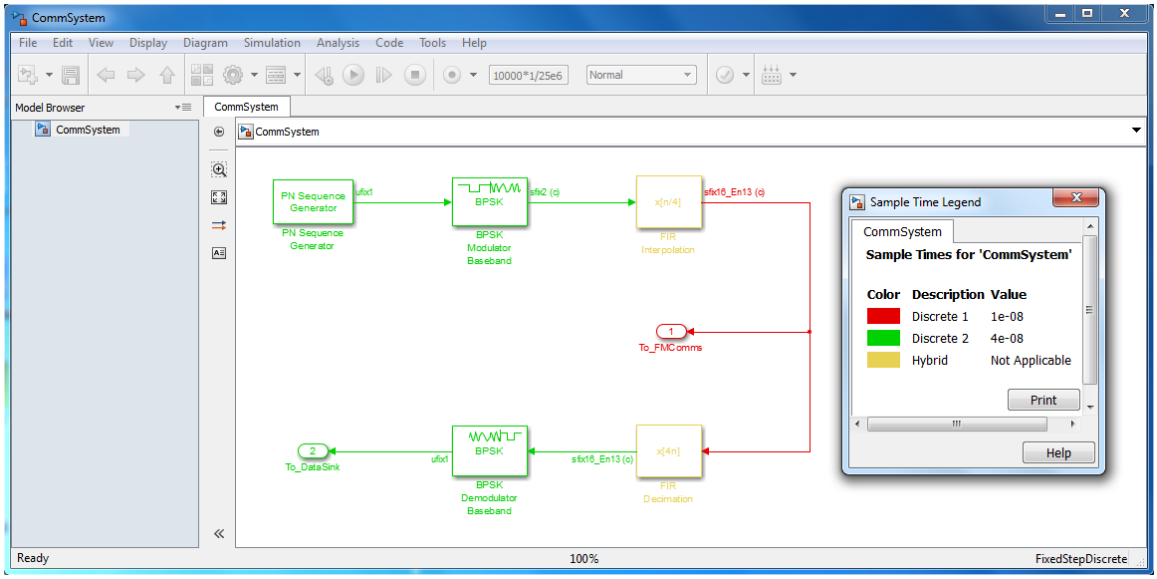

Figure 4.1: BPSK communications system model using Simulink. The model includes a PN sequence generator block, BPSK modulation and demodulation blocks, and FIR interpolation and decimation filter blocks. The legend shows the sample times of the model, which are each represented by different colors for the blocks and connections between them.

Simulink block library, using the default settings. The PN sequence generator is implemented using a linear feedback shift register (LFSR) with a 7-bit polynomial set to the binary vector [1 0 0 0 0 1 1] (read in descending order), where the 1's represent the shift register elements to be XORed together. Note that the top bit is excluded in the XOR process. The LFSR was also set to be initiated with the binary vector [0 0 0 0 0 1] (read in descending order). These values will produce an output sequence of length 63. The PN sequence generator sets the sampling time of the Simulink model because it is a digital source, which affects the sampling times of subsequent blocks. A sampling time of 40 ns was chosen and entered into the settings of the block, due to the multirate design and a 100 MHz clock source from the Zedboard. The LFSR will produce a new bit at a frequency of 25 MHz. The output of the LFSR was set to produce the "smallest unsigned integer" which is converted in HDL Coder to an unsigned 1-bit integer.

The BPSK modulator block was chosen from the Simulink library to perform the modulation of the data. Modulation This block will take in 0's and 1's to produce 1's and -1's

respectively, with the option for a phase offset to adjust the resulting output values. A phase offset of 0 was chosen for this design. However, with a phase offset of π, the output would produce a -1 and 1 for inputs of 0 and 1, respectively. As can be seen, this block could potentially produce imaginary values and therefore becomes the first block to output both real and imaginary output values. The user defined outputs were set to produce signed 2-bit integer values. Although the BPSK block may not produce an imaginary output, such as the chosen phase offset of 0, the HDL Coder will automatically create an imaginary output and set this value to zero.

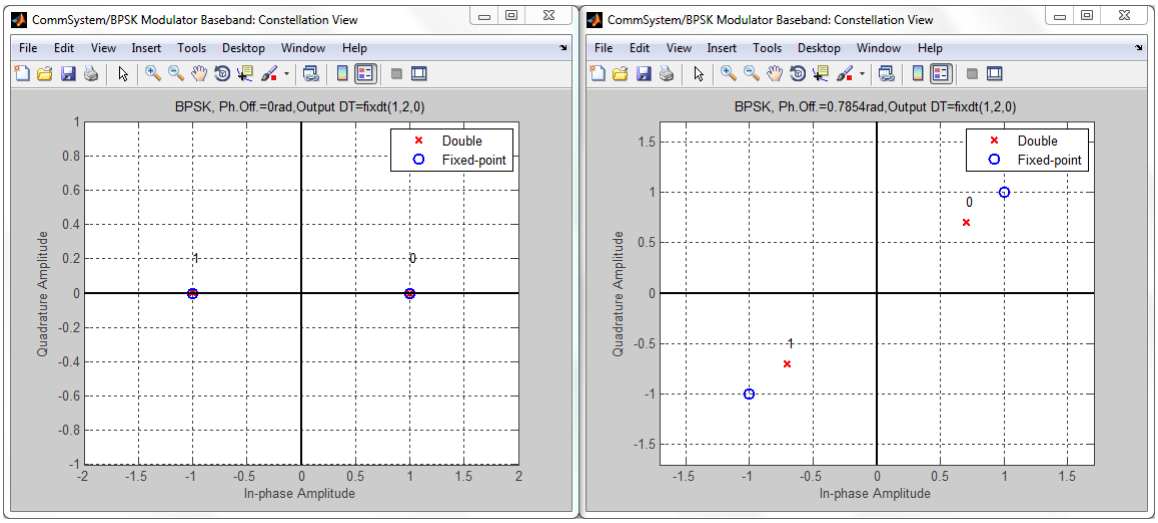

Figure 4.2: Plots of BPSK constellation diagrams from the BPSK modulator Simulink block. The red Xs show the desired locations for the BPSK symbol. The blue circles shows the actual location of the symbol based on the user defined output data type. The plot on the left used a 0 radian offset and a signed 2-bit integer for the user defined output data type. The plot on the right used an offset of $\frac{\pi}{4}$ radians and a signed 2-bit integer for the user defined output data type. Notice that a phase offset of $\frac{\pi}{4}$ and a signed 2-bit integer output does not produce a symbol with a magnitude of 1.

The plots in Figure 4.2 are the output constellations of the BPSK modulator block and show that special attention must be paid to the output data types. A signed 2-bit integer cannot represent $\frac{\sqrt{2}}{2}$ properly for both the real and imaginary parts, and will assign ones to both. Ones for both the real and imaginary parts of the output cause the magnitude to equal $\sqrt{2}$, which may not be desired in the system.

The filters for both the transmit pulse shaping and matched filters are identical and were designed with MATLAB's filterbuilder tool. The filter was designed as a square root raised cosine filter, with 4 samples per symbol, an order of 80, and a roll-off factor of 0.25. The combined response of the transmit pulse shaping filter and matched filter using two SRRC filters should reproduce the original data after downsampling, therefore the coefficients were normalized by the square root of the energy of the coefficients. The filter coefficient plot and magnitude response plot of the SRRC transmit pulse shaping filter and matched filter is shown in Figure 4.3.

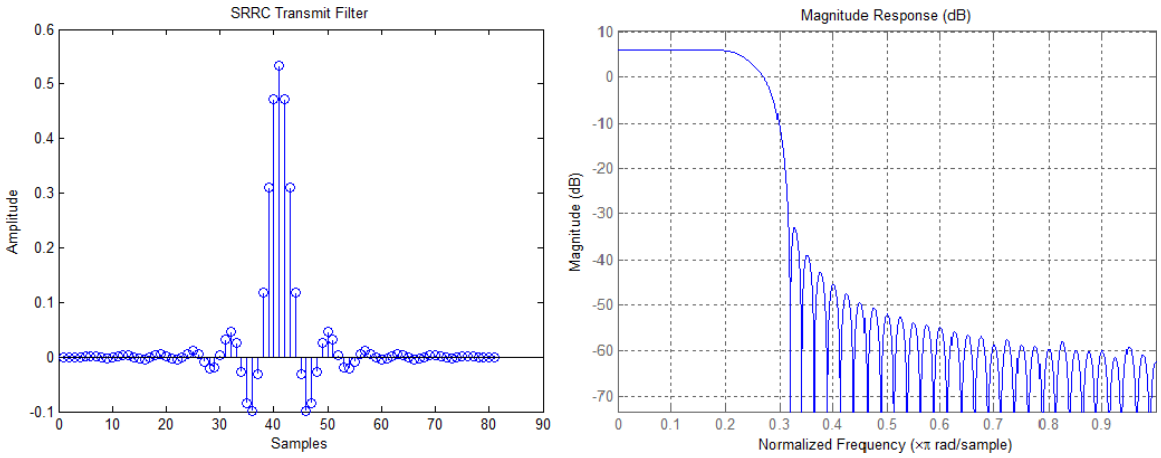

Figure 4.3: Transmit pulse shaping and matched filter plots. Both are a square root raised cosine filter with 4 samples per symbol, an order of 80, and a rolloff factor of 0.25. On the left is a plot of the filter coefficients and on the right is magnitude response of the filter. Passband is measured at 6dB and the stopband attenuation is measured at -33dB.

The transmit pulse shaping was implemented with a FIR Interpolation block, which interpolates the signal by some factor and then applies a FIR filter specified by the user. This block will take in signed 2-bit integers at a rate of 25 MHz and produces a pulse shaped signed 16-bit output signal at a rate of 100 MHz. The block in Simulink is implemented by scaling the coefficients by the interpolation factor and then applying the filter with a polyphase structure. The coefficients and the output data type were user specified to be signed 16-bit integers with 15 fractional bits.

A 16-bit output was chosen due to initial plans to integrate the FMComms1 board, which accepts a signed 16-bit integer to the DAC. However, this device was not used in the design. The largest producible output of the filter occurs when the all of the signs match between the input symbols (1's and -1's) and the filter coefficients in the correct order. When this occurs, the peak output value comes from the third subfilter, h(4n+2), at a value of 0.9028. Since the peak value is less than 1, the output does not require any non-fractional bits. Therefore, 15 fractional bits and 1 signed bit is acceptable to maintain precision.

Figure 4.4 shows a basic interpolation (polyphase) filter structure. The filter coefficients will be broken in 4 subfilters, by selecting every 4th coefficient. The input symbol from the modulator will be shifted into the shift register at a 25 MHz rate, which holds the current and previous 20 symbols. During the 40 ns period of the symbol, each subfilter will be selected, in order, for 10 ns and apply a convolution. Each coefficient of the selected subfilter is multiplied by a symbol in the shift register (in a time reversed order). The results of the multiplications are then all added together, and the result is output by a 100 MHz output register at the end of the 10 ns period. For every symbol that is shifted in, 4 samples are output at a 100 MHz rate.

The matched filter was implemented using the corresponding FIR Decimation block. This block implements the filter with a polyphase filter structure to downsample and filter the signal. The structure uses a decimation factor of 4 and filter order of 80. The input is a signed 16-bit integer with 15 fractional bits. The coefficients are stored as signed 16-bit integers with 15 fractional bits as well. The output data type of this block is a signed 16-bit integer with 14 fractional bits. The fractional bits are lower because the result of this receive filter can have an amplitude as high as 1.8, which requires at least 1 non-fractional bit. Therefore, there is 1 signed bit, 1 non-fractional bit, and 14 fractional bits.

Figure 4.5 shows the basic decimation filter structure. Each sample is input to a commutator that feeds a different shift register at a 100 MHz rate. Each shift register is then

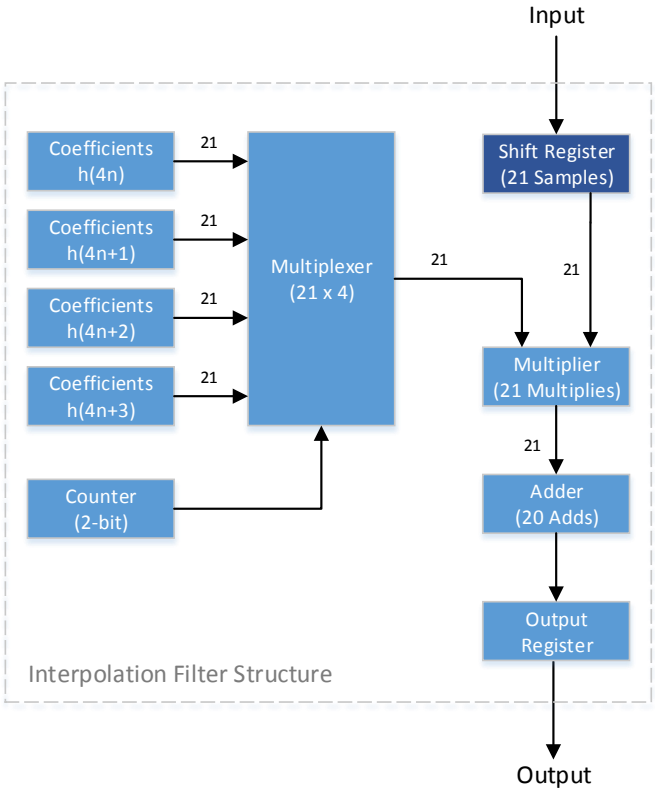

Figure 4.4: FIR Interpolation block with a polyphase filter structure, interpolation factor of 4, and a filter order of 80. Light blue blocks function at a clock rate 4 times greater than the dark blue blocks. This polyphase filter structure will take in a new input sample at a 1/4th the clock rate of the output samples and filter with each of 4 filter coefficient sets. Three zeros are appended to the filter h to make each to make the same number of multiplies for each phase.

multiplied by the time inverse of each corresponding subfilter. The results of each of the 81 multiplies are added together and fed to a output register that only takes in every 4th result by using a 25 MHz rate. In the implementation of the decimation filter, there are alignments needed to ensure the correct result such as commutator order. However, this is a simple explanation for the structure chosen in the Simulink model.

The BPSK demodulator block was chosen to perform the demodulation. This block allows for a phase offset similar to the BPSK modulator block. This block receives a new symbol at a 25 MHz rate. The decision type chosen was a hard decision, which will output

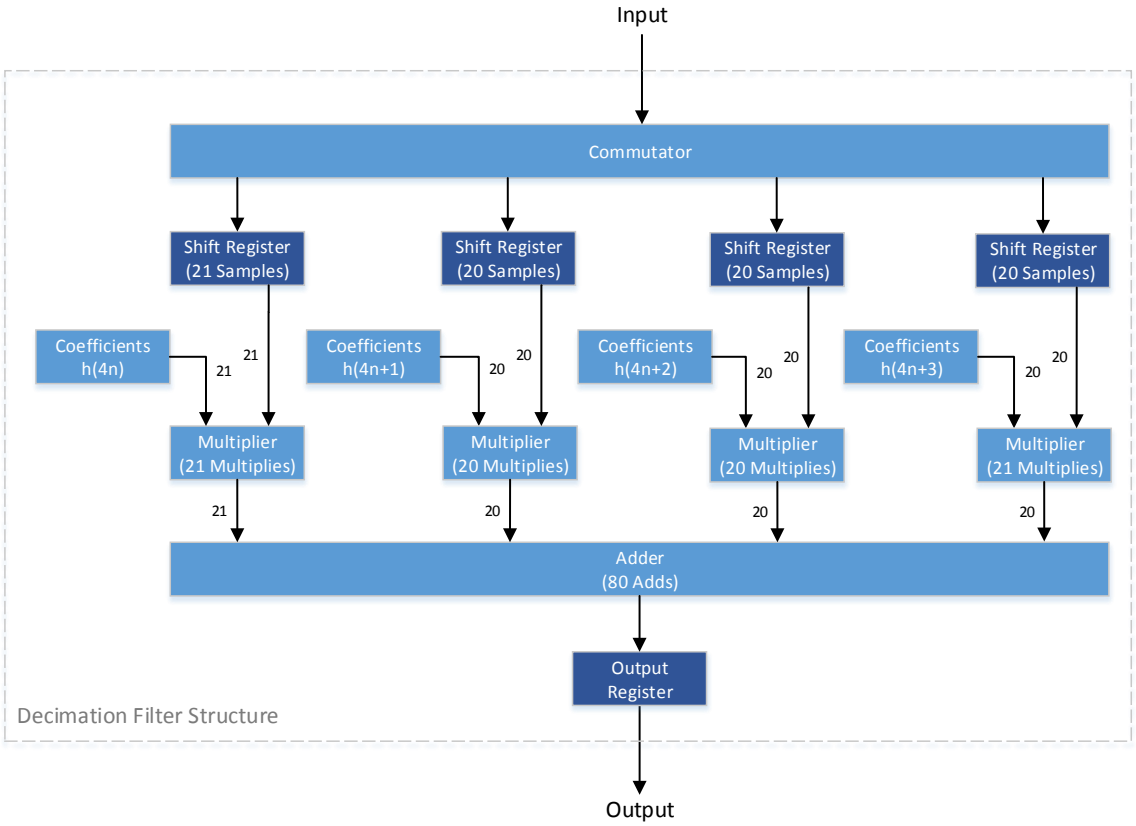

Figure 4.5: FIR Decimation block structure with a polyphase filter structure, downsampling factor of 4, and a filter order of 80. Light blue blocks function at a clock rate 4 times greater than the dark blue blocks. The input samples go through a commutator to be fed to 4 separate shift registers. The results of the multipliers and adders propagate to the output register which accepts every 4th sample to output.

a 1 for an input value equal to or greater than zero, and output a -1 for an input value less than 0. The output of this block will produce an unsigned 1-bit value to match the 1-bit value created by the PN sequence generator. However, this will include a delay due to the convolutions and memory elements of the filters. The result of this block is the output of the system labeled "To_DataSink".

Figure 4.6 shows two plots. The plot on the left shows the output of the transmit side of the model that could be sent to an RF front-end such as the FMComms1 board. The plot on the right shows the received data after demodulation. The values in these two plots are not correlated, but were chosen to show the testing environment in Simulink. There is

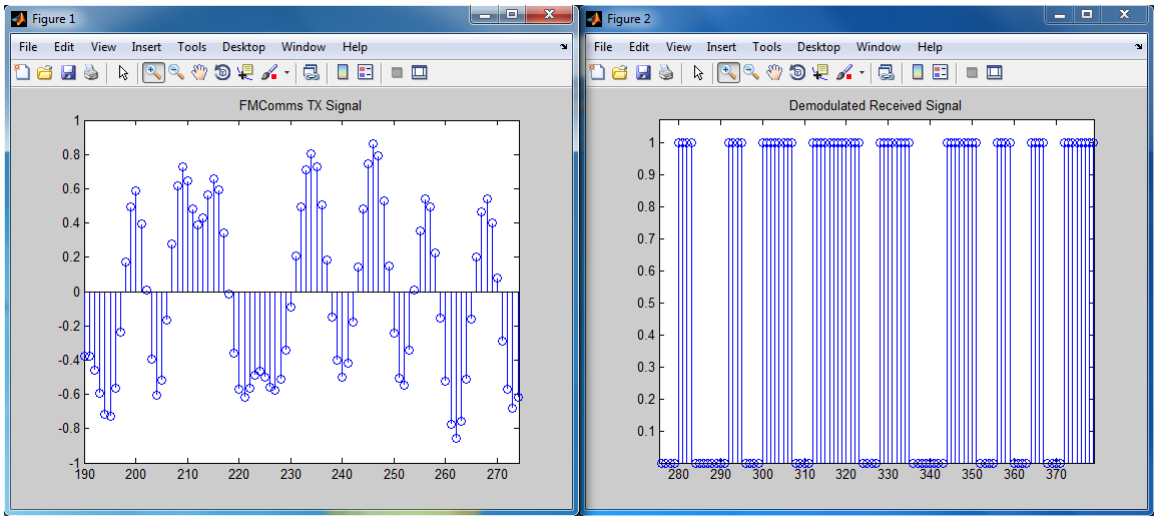

Figure 4.6: Plots produced during testing of the Simulink communications system model. The figure on the left shows the output of the interpolation filter. This signal could be sent to an RF front-end. The plot on the right shows the demodulated received signal.

a difference in time between the input of the system and the output of the system due to the nature of a convolution which skews data in time. Using Simulink for model verification made checking results very simple.

4.1.2 HDL Coder Conversion

A hardware engineer that builds systems using FPGAs must understand the timing and basic components of each design, where data types and registers matter for resource usage and timing. These are typical considerations that are abstracted from software engineers, who do not work with FPGAs. The HDL coder allows for a higher level of abstraction, but these considerations must be understood to use the tool effectively. The process of converting the model to work with HDL coder and to be converted to working VHDL is described in detail in this section.

The first step in the conversion process involves opening the HDL Coder Workflow Advisor. Attempting to convert the model described above results in timing issues and will not allow the VHDL code to function on a board. The reasons for the VHDL code not

working on the board are simple. The HDL coder does not add unnecessary registers unless specified by the user. This allows the user to select the positions within the component to apply pipeline registers. In the FIR decimation filter of the Simulink model, the block will need to accomplish 1 stage of 81 multiplies in parallel, plus 7 stages of 81 adds (by means of an adder tree), in one clock period. The propagation delays between all 8 stages of hardware elements was greater than the required time for the setup time of the output register. Initial testing on the Zedboard verified this to be true. To rectify this issue, either the clock period must be lengthened or pipeline registers must be added in the path to accommodate the propagation delays. The second option was chosen.

After evaluation of the design, there were input registers added to each block except the PN sequence generator and the BPSK modulator. This exception was justified because the output of the PN sequence generator was a register itself. This was accomplished by right clicking on the block and selecting "HDL Code" and then "HDL Block Properties". After selecting this menu option there was a window that defined how to implement the selected block, and had many options for pipelining within the block or adding input/output registers. The options vary depending on the block. Both the BPSK modulator and demodulator blocks only allowed a default architecture with constrained output pipeline, input pipeline, and output pipeline settings. However, the user can specify how many registers are desired. For the demodulator, an input register was added. For both the FIR interpolation and decimation blocks, many more options were available. Each had several types of architectures available, including: distributed arithmetic, fully parallel, fully serial, and partly serial. For each choice of architecture there were many different implementation parameters. Fully parallel architecture was chosen for both filters. The implementation parameters for a fully parallel architecture include: add pipeline registers (on/off), coefficient multipliers (multiplier/canonical signed digit/factored-CSD), constrained output pipeline, input pipeline, multiplier input pipeline, multiplier output pipeline, and output pipeline. For both filters, the add pipeline registers was turned on and an input pipeline register was added. For the interpolation filter, one multiplier output pipeline was added and was not needed for the decimation filter. By default, the decimation filter already had one. These

settings for the model allowed for the design approach for pipeline registers from Chapter 3, and the timing constraints, to be met for operation on hardware. The HDL block properties window is shown in Figure 4.7.

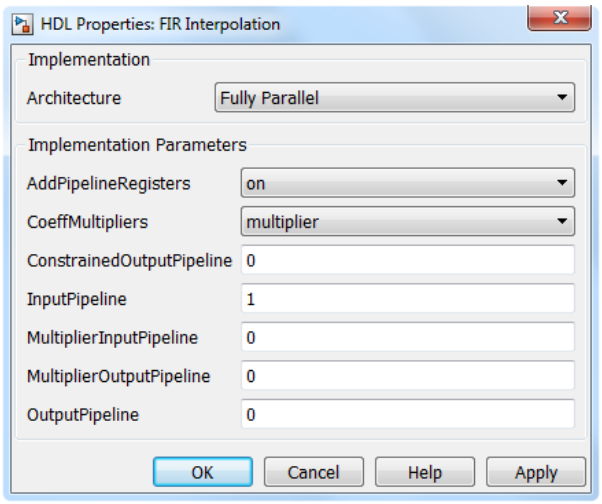

Figure 4.7: HDL Coder HDL block properties for Simulink FIR interpolation block. This window shows the options available for architecture type and parameters related to the architecture type in order to customize the HDL implementation. Each block in Simulink that is compatible with HDL coder will have a HDL properties block similar to this one.

After completing the model for HDL Coder block properties, that included adding necessary pipeline registers, the next step used the HDL Coder Workflow Advisor to convert the model to VHDL. If the model has been setup properly, the Workflow Advisor is very simple to operate and generates code quickly. There are 4 primary steps including: "Set Target", "Prepare Model for HDL Code Generation", "HDL Code Generation", and "FPGA Synthesis and Analysis". In the first step, the user sets the target device and synthesis tool. The first option is target workflow which can be "Generic ASIC/FPGA", "FPGA-in-the-loop" (requires HDL Verifier), "FPGA Turnkey" (applies to evaluation boards), "xPC Target FPGA I/O", and "IP Core Generation". The Generic ASIC/FPGA target workflow was chosen to apply toward the Zynq-7000 SoC of the Zedboard. Xilinx ISE was chosen for the synthesis tool, the specific Zynq-7000 SoC was chosen based on specifications, and the project folder location for the generated VHDL code was set.

The next step was "Prepare Model for HDL Code Generation", which did not require adjusting settings, but provided necessary checks for proper code generation. The checks involved include: Global settings, algebraic loops, block compatibility, and sample times. Assuming there are no problems during these checks, the advisor can advance to the "HDL Code Generation" step. Within this step are 3 important substeps that allow the user to define many parameters for code generation. On the "Set Basic Options" substep, the user can select whether VHDL or Verilog will be generated and which reports to generate. In the substep entitled "Set Advanced Options", there are many options from setting the names or postfixes for different signals or components to the design, to optimizations. The user can define the data type of inputs and outputs as well as specifics about coding style. For the optimizations tab, the user can optimize the timing controller, minimize clock enables, hierarchical distributed pipelining, and balance delays. Balance delays will analyze the model and add delay registers along specific data paths to match delays induced by hardware so that the system retains functional integrity. With Simulink, this may become an issue due to the way Simulink works. Simulink solves the model at discrete instances which does not account for hardware delays. Therefore, a delay in one block may take longer than a delay in different block. If the output of both blocks are required at the input of a third block, there will be a need for delay balancing to correct this hardware timing offset. Delay balancing was selected to ensure that the FIR decimation filter would receive the correct alignment of the input data, otherwise it would not operate correctly. The third substep contains options to set the testbench parameters such as clock timing and hold/setup times. Finally, the user can choose to generate the HDL code, testbench, cosimulation model, and validation model.

The fourth and final step is "FPGA Synthesis and Analysis", which allows the user to use third-party synthesis software from Xilinx or Altera to perform synthesis, mapping, and place and route functions. This step will create a project and perform these steps for the target hardware without having to start a project in the third-party IDE. The user has the ability to review each step to determine important information. Assuming these steps pass, HDL coder will annotate the model with synthesis and timing information that allows a

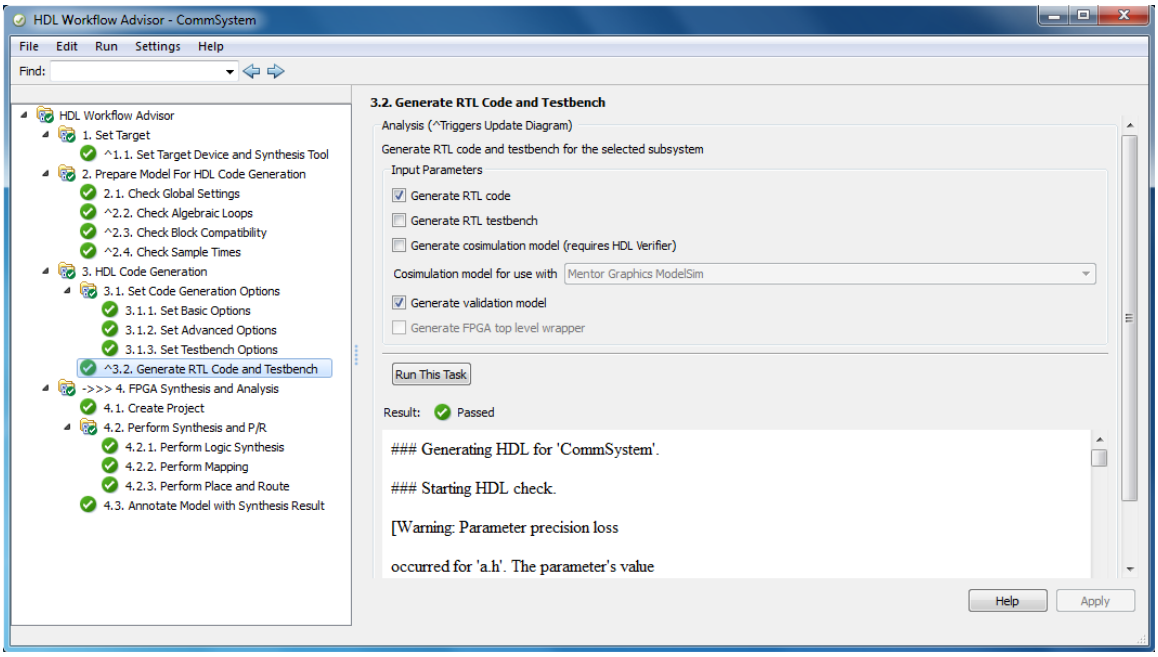

Figure 4.8: Example of HDL Coder's Workflow Advisor.

visual analysis on the Simulink model itself, highlighting critical paths. Using the Xilinx or Altera IDE and opening the project allows a quick start to bitstream generation.

After the build of the Simulink model was complete, HDL code was generated using the Workflow Advisor. Careful analysis was made to the structures the tool created, as well as the deficiencies, and required an iterative process between Simulink model adjustments and HDL Coder VHDL code generation to determine the extent of what the tool produced. The HDL Coder produced VHDL code that had very little registers unless it was necessary for the components function, such as the LFSR of the PN sequence generator, and the FIR Interpolation/decimation filter's input shift register and data hold registers. Due to this issue, the FIR interpolation filter, the FIR decimation filter, and BPSK demodulator were given input registers. The BPSK demodulator was given a output register to allow a constant value for the output. Without these delays, the hardware may not have time to implement all of the adds and multiplies between the existing registers. Starting at the output of the BPSK modulator with no phase shift, the model produced a complex result

with constant zeros for the imaginary. This complex value, although non-changing, caused the FIR Interpolation filter, FIR decimation filter, and BPSK demodulator to create code for the complex results as well, significantly adding unnecessary hardware.

4.1.3 Model-Driven VHDL Design Description

Through the process of building a Simulink model, using HDL Coder to create VHDL code, and the iterative process of making slight changes to get the desired design, a functional design was created. The resultant code produced was used for the model-driven design to compare with the hand-optimized design. This section describes the design produced by the HDL Coder, with the exception of the unnecessary complex components created by the HDL Coder.

The main top module called CommSystem will be described first. Figure 4.9 shows the configuration of the module. Contained in this module is the PN sequence generator, a timing control module, BPSK modulator/demodulator modules, and the FIR interpolation/decimation filter modules. The PN sequence was created in the top module using a shift register operating at a 25 MHz clock rate and an XOR. The 5th and 6th elements of the register were fed back into the input, or 1st element. The output of the PN sequence generator was a 1-bit signal from the 6th element of the shift register and fed into the BPSK modulator. The modulator contained a simple conditional signal assignment that output a signed 2-bit 1 for an input of 0 and a -1 for an input of 1. The output of the modulator was fed into the pipeline register of the FIR interpolation filter, that was specified in HDL Coder. The interpolation filter produced a 16-bit signed output operating at a 100 MHz clock rate, which was fed into 3 serial pipeline registers of the same frequency. One of these registers was the specified input pipeline register for the FIR decimation filter, while the other two were created by HDL Coder to match the delays needed for the FIR decimation filter to operate correctly. The signed 16-bit signal was input to the FIR decimation filter and output at a 25 MHz clock rate. The signed 16-bit output of the filter was sent to the specified input pipeline register of the BPSK demodulator, which used 3 simple conditional

signal assignments and a LUT to convert values between 0 and 32767 to a 1-bit 0, and values between -32768 and -1 to a 1-bit 1. The output of the BPSK demodulator was then input to the specified output pipeline register. The output of this pipeline was the output of the system.

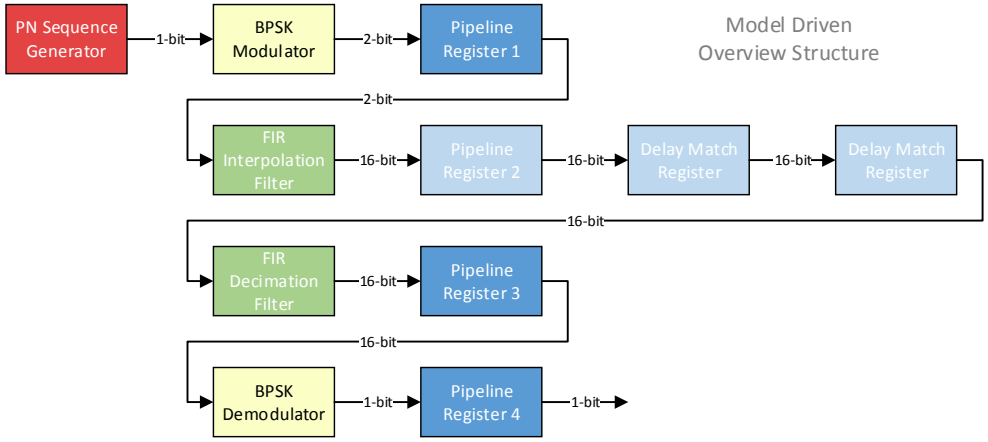

Figure 4.9: Model-driven communications system hardware structure. Dark blue registers operate at 25 MHz, while the light blue registers operate at 100 MHz. The PN sequence generator generates data at 25 MHz, which changes to 100 MHz in the FIR interpolation module. The data rate then changes back to 25 MHz in the decimation filter.

The timing control module produced 4 different signals for the purpose of handling the clock enables for the multi-rate design, in which the data moved at both the 100 MHz rate and the 25 MHz rate. Two of the enable signals, enb and enb_1_1_1 were also used to enable the 100 MHz components. They were created by, and reproduce, the clock enable of the system and are used to prevent registers from latching new data when the system should be temporarily halted. These two enables were also used to enable the 100 MHz components. One enable signal, enb_1_4_0, went high when a 2-bit counter reached a count of 0, to allow the pipeline registers within the CommSystem to operate at the 25 MHz rate. The last enable, enb_1_4_1, signal went high when the counter reached 1, and was used specifically for the clock enable output of the system. This enable allows CommSystem to be integrated with a larger system, which was not used. Table 4.1 shows the configuration of enables.

Table 4.1: Table of model-driven top module clock inputs.

Enable Inputs		
Module	**Enable**	**Data Rate**
PN Sequence Generator	Enb_1_4_0	25 MHz
BPSK Modulator	None	N/A
Pipeline Register 1	Enb_1_4_0	25 MHz
FIR Interpolation Filter	Enb_1_1_1	100 MHz
Pipeline Register 2	Enb	100 MHz
Delay Match Register 1	Enb	100 MHz
Delay Match Register 2	Enb	100 MHz
FIR Decimation Filter	Enb_1_1_1	100 MHz
Pipeline Register 3	Enb_1_4_0	25 MHz
BPSK Demodulator	None	N/A
Pipeline Register 4	Enb_1_4_0	25 MHz
Clock Enable Out	Enb_1_4_1	25 MHz

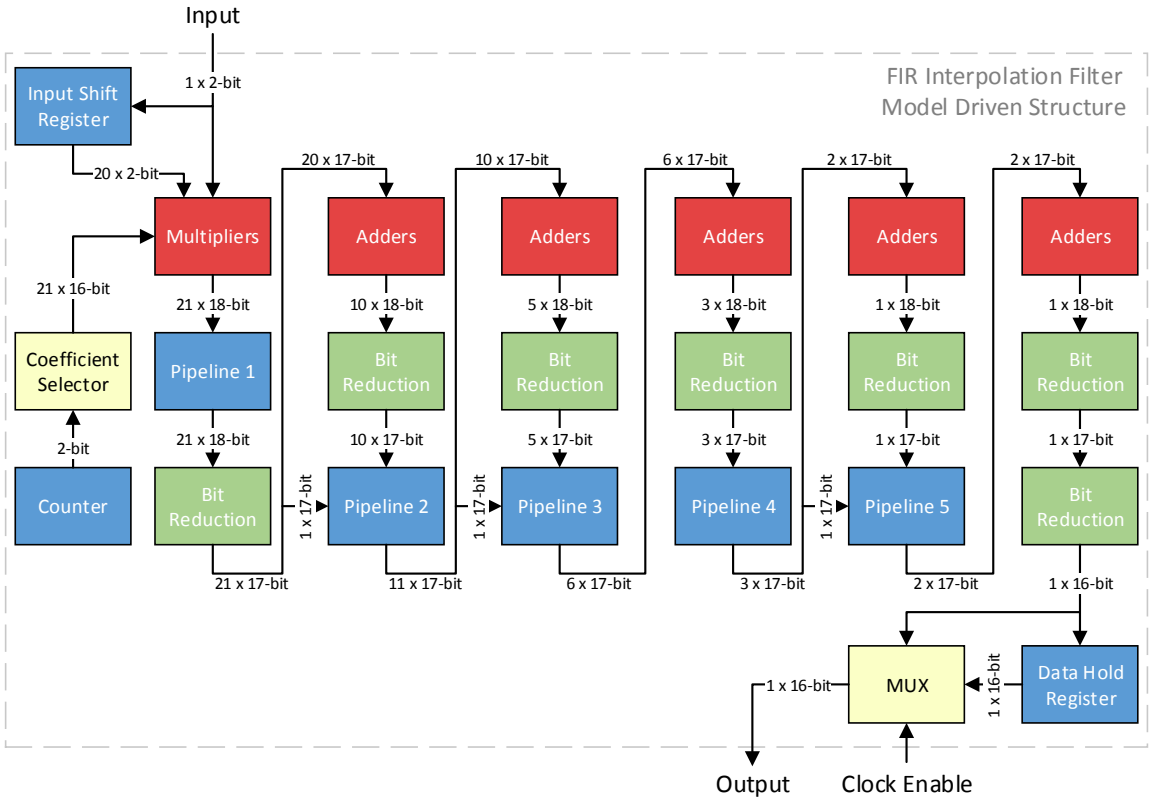

Figure 4.10: Model-driven FIR Interpolation Filter Hardware Structure.

The FIR interpolation filter structure is shown in Figure 4.10. The filter received a signed 2-bit input at a 25 MHz rate. Using a 2-bit counter operating at a 100 MHz rate, this filter selected 1 of 4 subsets of the signed 16-bit coefficients. When the next input arrived at the filter it selected the first subset for the first 10 ns, the second subset for the next 10 ns, the third subset for the next 10 ns, and finally the fourth subset during the last 10 ns of the 40 ns input data period. The current input was fed directly into a multiplier as well as a shift register that held the previous 20 inputs. In the first 10 ns of the input data's 40 ns period, the first subset of coefficients were multiplied by the current input and the previous 20 inputs. The current input was multiplied by the first coefficient of the subset, while the oldest input data was multiplied by the 21^{st} coefficient. This process was repeated with the second subset of coefficients with the same inputs in the next 10 ns, and so on. The results of these multiplies produced 21 signed 18-bit values per 10 ns period. Each of the 21 signed 18-bit values were latched by the specified multiplier output pipeline at a 100 MHz rate.

The pipeline values were then fed into bit reduction units with saturation checks that reduced all 21 signed 18-bit values to signed 17-bit values. Using a conditional signal assignment, the highest two bits were checked. If the MSB was a 0 (meaning positive number) and the next bit was a 1, then the result of the bit reduction was saturated. If the result was saturated in this case, the output produced all ones with an MSB of zero. If the MSB was a 1 (meaning negative number) and the next bit was a 0, then the result of the bit reduction was saturated. If the result was saturated in this case, the output produced all zeros with an MSB of one. If a saturation did not occur, then it simply selected the lowest 17 bits of the signal (which carried the same sign). The outputs of the bit reductions were then fed into the first stage of an adder tree.

The first adder stage took 20 of the 21 17-bit values and added them in pairs. The remaining 17-bit value was bypassed, which happened in subsequent stages when the number of values was odd. The output of this adder stage was 10 18-bit values, which was then passed into bit reductions to create 10 17-bit values. After the bit reduction, the 10

values were recombined with the bypassed 17-bit value and were then passed into a specified pipeline stage. These three stages (adder, bit reductions, pipeline) were repeated several times, reducing the number of values from 11, to 6, to 3, and then to 2. After the 2 remaining 17-bit values exited the last pipeline stage, they went through an adder, then two bit reductions until there was only 1 16-bit value. This 16-bit value went to two places, a data hold register and a multiplexer. The multiplexer allowed the result of the filter to pass through to the output when the clock enable is high, else output the value held in the data hold register.

Table 4.2: FIR Interpolation Filter Input Data.

FIR Interpolation Filter Input Data			
Data	**Number**	**Data Type**	**Data Rate**
Filter Input	1	2-bit	25 MHz
Input Shift Register	20	2-bit	25 MHz
Subfilter 1-4 Coefficients	21	16-bit	100 MHz

Table 4.2 shows the input values of the FIR interpolation filter up to the multiplier. The filter input was held for 4 clock cycles, while the coefficients cycled from subfilter 1 to subfilter 4 on each clock cycle. Table 4.3 shows the internal stages of the interpolation filter. These tables aid in understanding the internals of the interpolation filter.

The FIR decimation filter structure is shown in Figure 4.11. The FIR decimation filter received 16-bit signed values at a 100 MHz rate after they exited the FIR interpolation filter and 3 pipeline registers (used for timing alignment). The filter had 4 phases similar to the interpolation filter. However, instead of changing coefficients, it loaded the 100 MHz input into 1 of 4 different 20 x 16-bit input shift registers. Each shift register was enabled by a 4-bit ring counter. The order that they were loaded was [1 4 3 2]. Each register individually shifted in a value at a 25 MHz rate due to the enables generated by the ring counter. If an input of [1 2 3 4 5 6 7 8 9 10 11 12] entered the filter, registers 1 through 4 would contain [1 5 9], [4 8 12], [3 7 11], and [2 6 10], respectively (if shifted left). The input and input shift registers 1-4 were fed to 81 multipliers, and were then multiplied

Table 4.3: FIR Interpolation Filter Internal Stages.

FIR Interpolation Filter Internal Stages			
Block	**Input(s)**	**Data Bypassed**	**Output**
Multipliers	(21 x 2-bit) (21 x 16-bit)		21 x 18-bit
Pipeline Stage 1	**21 x 18-bit**		**21 x 18-bit**
Bit Reductions	21x 18-bit		21 x 17-bit
Adder Stage 1	20 x 17-bit	1 x 17-bit	10 x 18-bit
Bit Reductions	10 x 18-bit	Same as above	10 x 17-bit
Pipeline Stage 2	**11 x 17-bit**	**Recombined**	**11 x 17-bit**
Adder Stage 2	10 x 17-bit	1 x 17-bit	5 x 18-bit
Bit Reductions	5x 18-bit	Same as above	5 x 17-bit
Pipeline Stage 3	**6 x 17-bit**	**Recombined**	**6 x 17-bit**
Adder Stage 3	6 x 17-bit		3 x 18-bit
Bit Reductions	3 x 18-bit		3 x 17-bit
Pipeline Stage 4	**3 x 17-bit**		**3 x 17-bit**
Adder Stage 4	2 x 17-bit	1 x 17-bit	1 x 18-bit
Bit Reductions	1 x 18-bit	Same as above	1 x 17-bit
Pipeline Stage 5	**2 x 17-bit**	**Recombined**	**2 x 17-bit**
Adder Stage 5	2 x 17-bit		1 x 18-bit
Bit Reductions	1 x 18-bit		1 x 17-bit
Bit Reductions	1 x 17-bit		1 x 16-bit
Multiplexer	1 x 16-bit		1 x 16-bit

with all 81 16-bit stored coefficients. The input and input shift registers were multiplied according to Table 4.4. The multiplies not listed follow the convention of next newest input sample in respective input shift register to increasing coefficient of the respective subset.

The multiplier stage produced 81 32-bit values at a rate of 100 MHz. The outputs of the multipliers were then fed into 81 bit reductions with saturation checks, following the same conventions as previously described. The outputs of the bit reductions were 81 31-bit values, which were then fed into the first stage of specified pipeline registers. The pipeline operates at a 25 MHz rate using the same enable fed to input shift register 1. This enable caused an unusual behavior as depicted and explained in Figure 4.12.

The results of the multiplies of each subfilter coefficient with their respective input pipeline register, as well as the current input, were latched into the pipeline as per descrip-

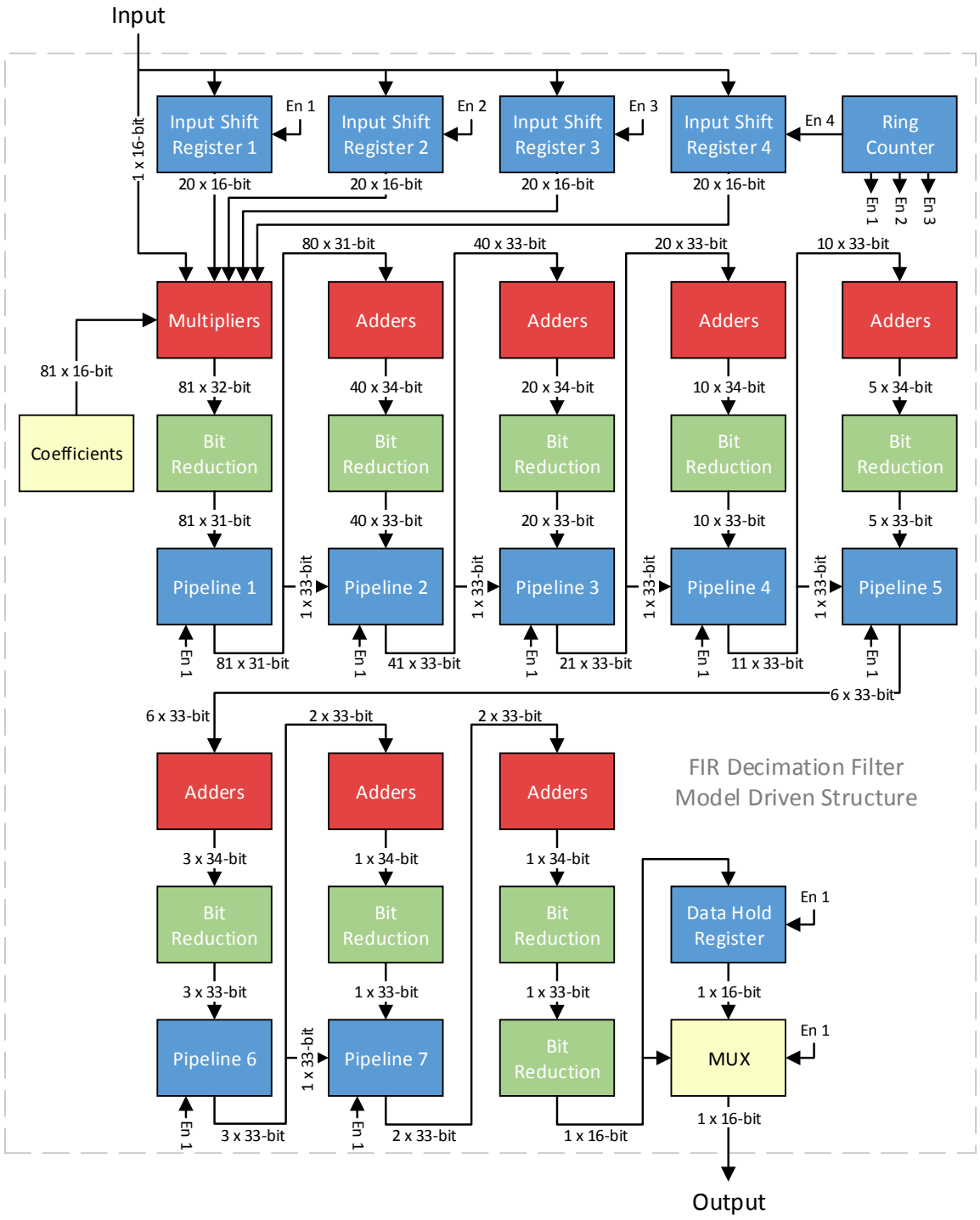

Figure 4.11: Model-driven FIR Decimation Filter Hardware Structure.

Table 4.4: FIR Decimation Filter Multiply Pairs.

FIR Decimation Filter Multiply Pairs			
Multiplicand 1		**Multiplicand 2**	
Input Shift Register	**Register Position**	**Subfilter**	**Coefficient**
Current Input		1	1
1	0	1	2
1	19	1	20
2	0	2	1
2	19	2	20
3	0	3	1
3	19	3	20
4	0	4	1
4	19	4	20

tion in Figure 4.12. Eighty of the values from the output of the pipeline then entered the first adder stage, while the remaining 31-bit value bypassed. The adder stage produced 40 34-bit values, which was then passed into a bit reduction stage. The bit reduction stage reduced these values to 40 33-bit values. The 31-bit value that bypassed the adder stage and bit reductions was then resized to a 33-bit value and input to the second stage of pipelines with the other 40 33-bit values. The 41 33-bits values then repeated 5 cycles of adder stages, bit reductions, and pipeline stages. The number of values was reduced from 41, to 21, to 11, to 6, to 3, and then to 2. After pipeline stage 7, the remaining 2 33-bit values were passed through an adder and bit reduction stages to produce a final 33-bit value. The final value went through a bit reduction with saturation check step that differed slightly from the previously mentioned ones. The reduction step checked for saturation similar to the previously mentioned ones. However, if no saturation had occurred, it output a 16-bit value from selected bits 32 down to 17. The output of this bit reduction entered both a multiplexer and a data hold register. The multiplexer allowed the output of the final bit reduction to be fed to the output of the filter when the ring counter enable for input shift register 1 was high. While this enable was high, the data hold register loaded the output of the bit reduction step. When the enable went low, the multiplexer would pass through the output of the data

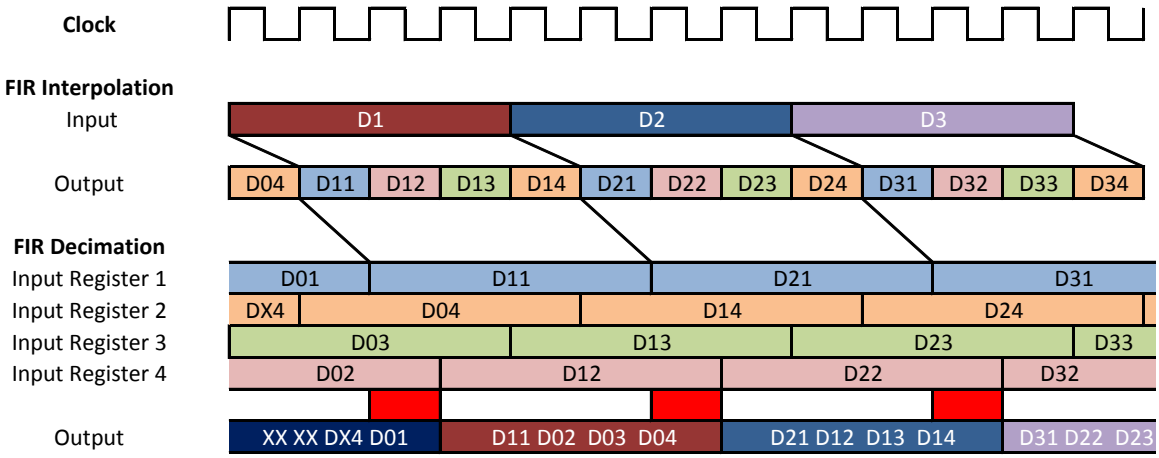

Figure 4.12: Propagation of input data through FIR interpolation filter and FIR decimation filter to show data origins and groupings for decimation output. Figure neglects actual delays and results, and assumes one clock cycle delay between each stage (input/output) for ease of understanding. Notice in the decimation output (D11, D02, D03, D04) that the first output value (D11) of the interpolation filter input (D1) is grouped with the previous inputs (D02, D03, D04). The red boxes represent the transitional grouping of the values above them to the decimation filter output. In reality, D11 would be the convolution of all 20 of the previous and the D1 input with the subfilter 1 coefficients. The input registers of the decimation filter only show the current input, however would actually hold 19 of the previous values for each (every 4^{th} input). The output of the decimation filter that contains [D11 D02 D03 D04] would be D11, and the previous 19 inputs of the input register, multiplied by their respective subfilter coefficients, plus the other registers (each containing 20 values) multiplied by their respective coefficients. Note that the action of the convolution induces a delay as well, therefore the input of the interpolation filter is not equivalent to the output of the decimation filter until many clock cycles later.

hold register. Table 4.5 shows all of the internal stages with inputs, outputs, and data rates.

After the VHDL code was created, it was integrated with the Error Counter module and tested on the Zedboard, as specified per Chapter 3. The model-driven design VHDL code proved to create a working design by not receiving errors for a period of 3 seconds, and was considered ready for the performance assessment.

Table 4.5: FIR Decimation Filter Internal Stages.

FIR Decimation Filter Internal Stages			
Block	**Input(s)**	**Data Bypassed**	**Output**
Multipliers	(81 x 16-bit) (81 x 16-bit)		81 x 32-bit
Bit Reductions	81 x 32-bit		81 x 31-bit
Pipeline Stage 1	**81 x 31-bit**		**81 x 31-bit**
Adder Stage 1	80 x 31-bit	1 x 31-bit	40 x 34-bit
Bit Reductions	40 x 34-bit	1 x 31-bit	40 x 33-bit
Pipeline Stage 2	**41 x 33-bit**	**Resized Recombined**	**41 x 33-bit**
Adder Stage 2	40 x 33-bit	1 x 33-bit	20 x 34-bit
Bit Reductions	20 x 34-bit	1 x 33-bit	20 x 33-bit
Pipeline Stage 3	**21 x 33-bit**	**Recombined**	**21 x 33-bit**
Adder Stage 3	20 x 33-bit	1 x 33-bit	10 x 34-bit
Bit Reductions	10 x 34-bit	1 x 33-bit	10 x 33-bit
Pipeline Stage 4	**11 x 33-bit**	**Recombined**	**11 x 33-bit**
Adder Stage 4	10 x 33-bit	1 x 33-bit	5 x 34-bit
Bit Reductions	5 x 34-bit	1 x 33-bit	5 x 33-bit
Pipeline Stage 5	**6 x 33-bit**	**Recombined**	**6 x 33-bit**
Adder Stage 5	6 x 33-bit		3 x 34-bit
Bit Reductions	3 x 34-bit		3 x 33-bit
Pipeline Stage 6	**3 x 33-bit**		**3 x 33-bit**
Adder Stage 6	2 x 33-bit	1 x 33-bit	1 x 34-bit
Bit Reductions	1 x 34-bit	1 x 33-bit	1 x 33-bit
Pipeline Stage 7	**2 x 33-bit**	**Recombined**	**2 x 33-bit**
Adder Stage 7	2 x 33-bit		1 x 34-bit
Bit Reductions	1 x 34-bit		1 x 33-bit
Bit Reductions	1 x 33-bit		1 x 16-bit
Multiplexer	1 x 16-bit		1 x 16-bit

4.2 BPSK Hand-Optimized Design

Using the resulting VHDL code produced by HDL Coder model-driven design as a template of the design, a hand-optimized version was produced by optimizing the code. The purpose of the hand-optimized VHDL coder was to compare timing and hardware utilization with the model-driven design. Optimizing the HDL Coder generated code was justified because the product of handwritten code would be very close in structure. The first major change between the designs was the removal of all the imaginary components. The imaginary values were held constant and propagated through most of the design, which had little

affect to hardware utilization due to optimizations of Xilinx's ISE suite. The imaginary components started in the BPSK modulator, which was changed to produce only 1's and -1's for only the real component. Figure 4.13 shows the changes to the top module, with the enables shown in Table 4.6. The changes to the top module are related to the changes made within the filters, and will be discussed when the topics occur.

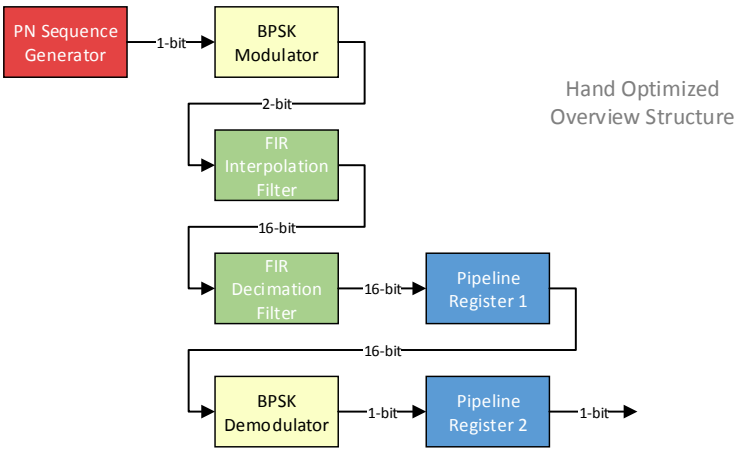

Figure 4.13: Hand-optimized communications system hardware structure. Differences from BPSK model-driven design include: Removal and replacement of input pipeline registers of the FIR interpolation and decimation filters inside of the structures, and removal of match delay registers.

Table 4.6: Table of hand-optimized top module clock inputs.

Enable Inputs		
Module	**Enable**	**Data Rate**
PN Sequence Generator	Enb_1_4_0	25 MHz
BPSK Modulator	None	N/A
FIR Interpolation Filter	Enb_1_1_1	100 MHz
FIR Decimation Filter	Enb_1_1_1	100 MHz
Pipeline Register 1	Enb_1_4_0	25 MHz
BPSK Demodulator	None	N/A
Pipeline Register 2	Enb_1_4_0	25 MHz
Clock Enable Out	Enb_1_4_1	25 MHz

In the hand-optimized code, the inputs of the interpolation and decimation filters were fed directly to the shift registers. In the model-driven design, the inputs were fed to the

multipliers first. For implementation of this adjustment, each of the input shift registers were increased by one register value. The input pipeline registers were able to be removed in the top module due to this replacement. Removal of these registers in the top module was justified because of the addition of the input pipeline registers in each filter, effectively still acting as a pipeline between the BPSK modulator and the multipliers of each filter. The register placement description in the design approach chapter was also met in the process as well. By moving the input pipeline register of the interpolation filter, the reset value of the 2-bit counter that controlled the enables of the input shift register was able to be adjusted. This adjustment allowed the input value to be latched two clock cycles earlier, in the center of the input data period, instead of at the end of the 4 cycle input period.

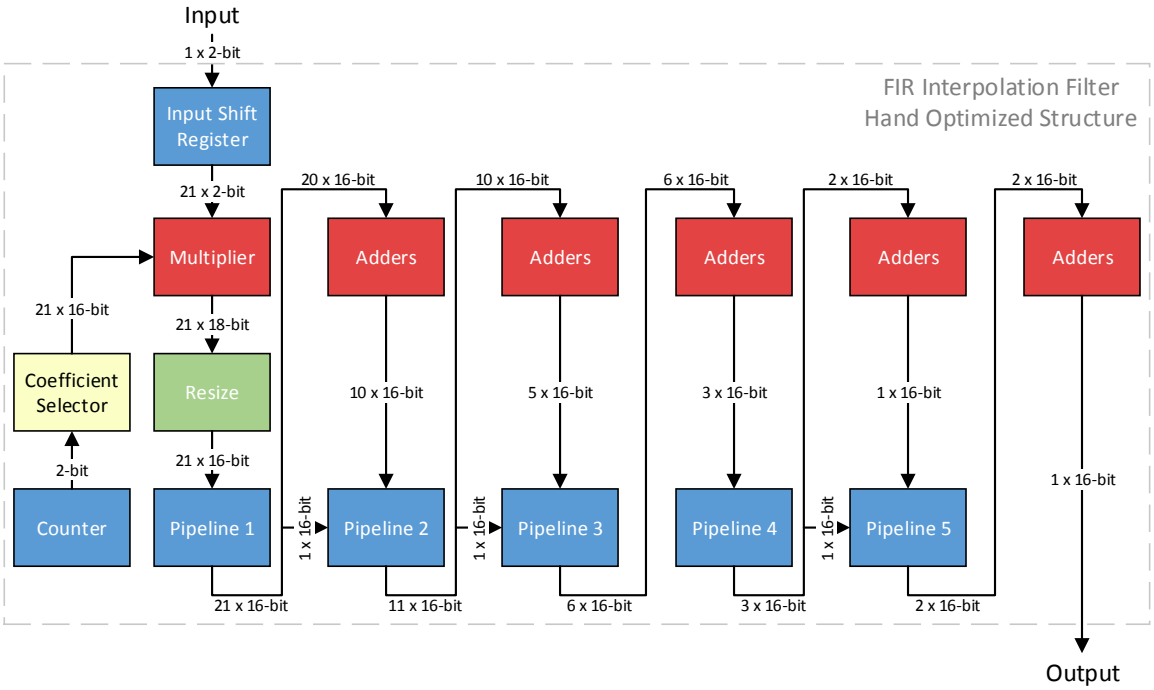

Figure 4.14: Hand-optimized FIR Interpolation Filter Hardware Structure. Differences from BPSK model-driven design include: extra input pipeline value, removal of all bit reduction with saturation checks, and bit sizes are fixed at 16-bit after multiplier.

The largest hardware optimizations came from the FIR interpolation filter, which is shown in Figure 4.14. Many of the saturation checks and resizing produced unnecessary

hardware and were removed from the filter. The removal of these checks was justified due to the maximum possible output of the filter. The values in the input shift register were either 1's or -1's due to the modulator output. The maximum possible output, or worst case scenario, occurred when the signs of the values contained in the input shift register matched the signs of the coefficients. Therefore, the sum of the absolute values of each of the subfilter's coefficients was the maximum possible output for a given filter. For subfilters 1 through 4, these values were 0.7324, 0.8250, 0.9028, and 0.8250, respectively. These values translate to the 16-bit values, with 15 fractional bits, of 24002, 27034, 29584, and 27034. Each of these outputs fell short of 32767, which was the maximum value before saturation and rollover occurred from the adders. Therefore, the removal of the resizes and saturation checks was possible and allowed for reduced hardware utilization. Table 4.7 shows the hand-optimized interpolation filter's internal stages. By comparison with Table 4.3, it was evident that there should be a significant reduction in hardware.

Table 4.7: FIR Interpolation Filter Internal Stages.

FIR Interpolation Filter Internal Stages				
Block	**Input(s)**	**Data Bypassed**	**Output**	**Data Rate**
Multipliers	21 x 2-bit 21 x 16-bit		21 x 18-bit	100 MHz
Pipeline Stage 1	**21 x 16-bit**		**21 x 16-bit**	**100 MHz**
Adder Stage 1	20 x 16-bit	1 x 16-bit	10 x 16-bit	100 MHz
Pipeline Stage 2	**11 x 16-bit**	**Recombined**	**11 x 16-bit**	**100 MHz**
Adder Stage 2	10 x 16-bit	1 x 16-bit	5 x 16-bit	100 MHz
Pipeline Stage 3	**6 x 16-bit**	**Recombined**	**6 x 16-bit**	**100 MHz**
Adder Stage 3	6 x 16-bit		3 x 16-bit	100 MHz
Pipeline Stage 4	**3 x 16-bit**	**Recombined**	**3 x 16-bit**	**100 MHz**
Adder Stage 4	2 x 16-bit	1 x 16-bit	1 x 16-bit	100 MHz
Pipeline Stage 5	**2 x 16-bit**	**Recombined**	**2 x 16-bit**	**100 MHz**
Adder Stage 5	2 x 16-bit		1 x 16-bit	100 MHz

The result of the multiplier, which produced a 18-bit value, was shortened to using the lowest 16-bits. The largest coefficient was 0.5342, or a 16-bit value with 15 fractional bits of 17504, which required only the lowest 15 bits plus the signed bit. Due to the worst case scenario, all of the adders functioned without a requirement to check for saturation

using only 15 bits plus a signed bit. Therefore, the saturation checks were removed after 21 multipliers, 20 adders (implemented with an adder tree), and an output type conversion.

In the model-driven design there were 3 delays between the interpolation filter and the decimation filter. One was the input pipeline of the decimation filter, while the other two were "delay matching" pipelines meant to keep timing correct for the decimation filter. The input pipeline for the decimation filter was moved inside the decimation filter by adding one more register value to input shift register 1. In addition to the pipeline register move, both of the delay matching filters were removed. To ensure correct alignment of the data for the decimation filter, the reset value of the ring counter was adjusted. The 3 pipelines combined consumed 4 clock cycles from the output of the interpolation filter to the first pipeline register of the decimation filter (after the multipliers). By adding the input register, removing the delay match registers, and adjusting the ring counter reset value, this delay was reduced to 2 clock cycles.

The hand-optimized decimation filter is shown in Figure 4.15. The FIR decimation filter did require using some of the saturation checks, thus these were not removed as they were in the interpolation filter. The FIR decimation filter proved much more difficult due to a varying 16-bit input with potential added noise from a RF front-end and required careful attention to saturation. Although there were no significant reductions of hardware in the decimation filter, there were significant reductions to the number of clock cycle delays between the input and output. The clock cycle delays were reduced heavily within the decimation block by changing the pipeline stage enables. In the model-driven design, each pipeline was enabled by the same 25 MHz enable signal, causing 4 clock cycle delays per pipeline register. The overall delay of the filter was reduced from 28 clock cycles to 8 clock cycles, measured from propagation of the input data to the output, by using different ring counter enables for each pipeline (shown in Figure 4.15).

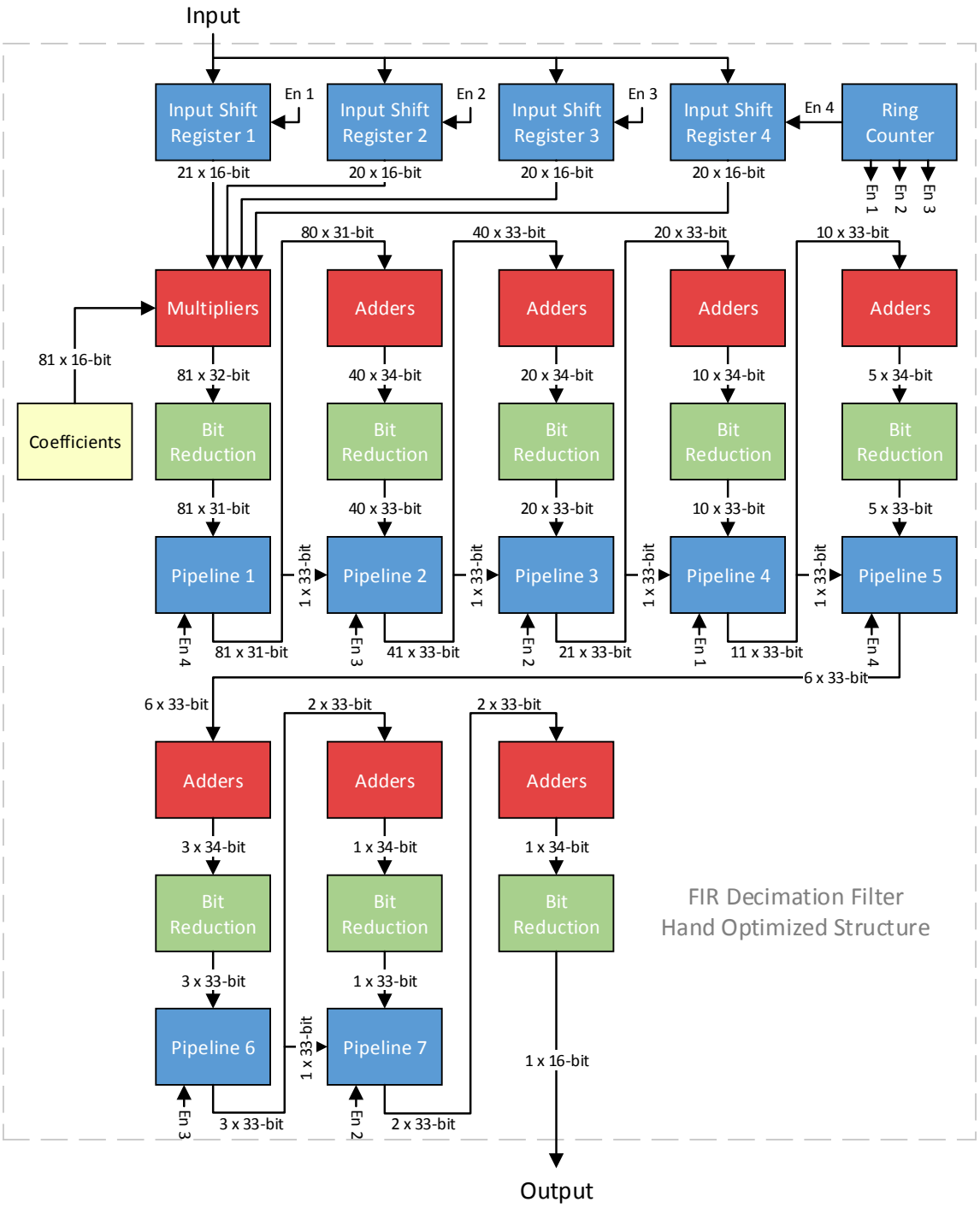

Figure 4.15: Hand-optimized FIR Decimation Filter Hardware Structure. Differences from BPSK model-driven design include: extra input pipeline value (subfilter 1), pipeline register enables, and removal of data holder register and multiplexer due to existing pipeline register enables.

The BPSK demodulator input and output pipeline registers were the only remaining registers that needed timing adjustments. The input pipeline register was actually given the correct enable signal already, thus the BPSK output register was given the clock enable out signal, which removed 3 more delays. The model-driven design version of the BPSK demodulator took in a complex structure and made a hard decision using a look up table (LUT) based on if the real or imaginary parts were less than or equal to zero. Instead of using a large LUT for both real and imaginary parts, this process was simplified in the hand-optimized design by only using the signed bit (of the signed 16-bit input) to specify the demodulated bit. A negative value would translate to a one and a positive would translate to a zero.

After the VHDL code was hand-optimized, it was integrated with the Error Counter module and tested on the Zedboard, as specified per Chapter 3. This model proved to be a working model by not receiving errors for a period of 3 seconds, and was considered ready for the performance assessment.

4.3 QPSK Model-Driven Design

A QPSK model was produced to show simple conversion and versatility of the HDL coder. Very quickly and efficiently the BPSK model was converted to a QPSK modulation scheme by simply removing the BPSK blocks and replacing them with QPSK modulation and demodulation blocks. The QPSK modulation scheme would utilize the imaginary components of the FIR interpolation and decimation filters.

Another PN Sequence generator was added, as well as a vector concatenate block, to allow the complex data into the QPSK modulation block as shown in Figure 4.16. The modulation block was given a phase offset of $\frac{\pi}{4}$, used gray constellation ordering, and accepted a bit input. The output data type was a user defined signed 2-bit integer which actually produced two outputs, one for real and one for imaginary parts. The QPSK demodulator had a phase offset of $\frac{\pi}{4}$, gray coded constellation ordering, and used a hard decision for

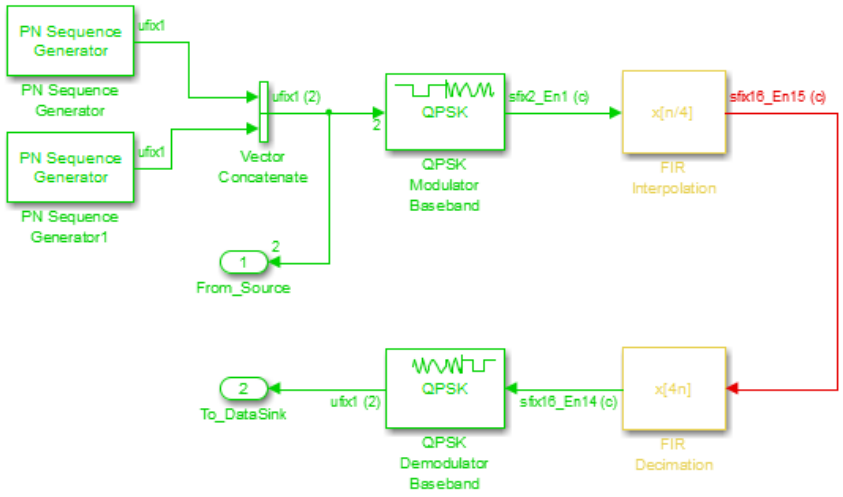

Figure 4.16: QPSK Simulink Model.

decision type. The output of this block was set to produce the smallest unsigned integer, which became a unsigned 1-bit integer for both real and imaginary parts. The model was then converted to VHDL code, but was not implemented on the Zedboard due to lack of resources available. This model was created to show that converting to a different modulation scheme was a simple process that consumed very little time.

4.4 Integration of MATLAB coded blocks

HDL Coder can be used to create many types of systems, however is limited to the blocks that are supported by HDL Coder. If a block is not supported by HDL Coder it can possibly be implemented with a MATLAB Function block. The MATLAB Function block allows the user to create algorithms using MATLAB code. This process allows expanded usage and easy implementation using MATLAB code instead HDL code. The MATLAB function block supports all of the standard data types as well as fixed point. The operations supported include arithmetic, rational, and logical, as well as bitwise or matrix operations. The fixed point runtime library contains many of the functions that can be converted to HDL as well. Many features such as these make this block ideal for ease of writing code

and generating HDL code to integrate into an existing design.

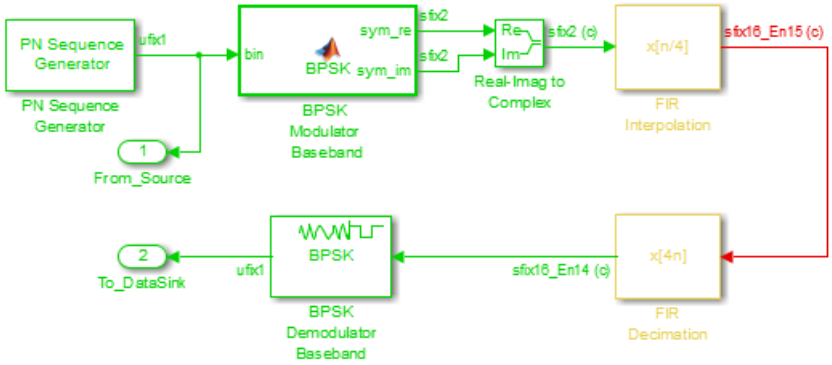

Figure 4.17: BPSK Simulink Model with MATLAB code block implementing a BPSK modulator.

BPSK MATLAB Coded Block

```
function [sym_re, sym_im] = BPSK(bin)

if bin == 1
    sym_re = sfi(-1,2,0);
else
    sym_re = sfi(1,2,0);
end

sym_im = sfi(0,2,0);

end
```

Figure 4.18: MATLAB Function block code for BPSK modulator.

The BPSK Simulink model was altered to include the MATLAB Function block in place of the BPSK Modulator block. MATLAB code was written in the block to perform the same function that the BPSK Modulator block performed. The function name was given to be BPSK(), the input was named “bin”, and the outputs were named “sym_re” and “sym_im”. This setup can be seen in Figure 4.17. Inside the function there was an if statement that checked if bin was equal to 1. If it was true, it would output a -1 for sym_re. If it was false, it would output a 1 for sym_re. The output sym_im was always given a 0. In order to

prevent MATLAB from interpreting the output as a double, two functions named fi() and sfi() can be used to specify the output as fixed point or signed fixed point. By setting all the outputs to sfi(x,2,0), with x being their respective values, the output will be interpreted by the next block as a 2-bit signed integer with 0 fractional bits. After code generation, the BPSK modulator MATLAB code was translated to VHDL and placed as an if statement inside a process with the correct data types produced.

4.5 Integration of Existing HDL Code

When using HDL Coder, there are limitations of the blocks that can be implemented. Many functions can be created using MATLAB code and be converted to allow more versatility. However, sometimes there is a need to integrate existing VHDL code into designs that utilize the benefits of model-driven tools such as HDL Coder. The process of integration of VHDL into a model is not direct without the use of HDL verifier.

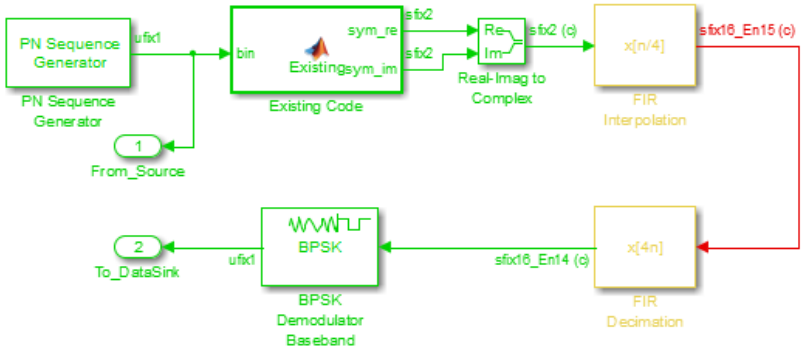

Figure 4.19: BPSK Simulink Model with MATLAB Function block as a place holder for existing code to be inserted after generation.

Existing code must be integrated after code generation of the Simulink model. There are several ways to create the model for integrating existing code. The first method is used when the data type will change when passing through the existing code. This method involves using a MATLAB Function block as a place holder. Within the MATLAB Function

block, the input should be directly connected to the output unless the data type changes between input and output. If the data type does change, the input should be disconnected and the functions fi() or sfi() should be used with constants to set the output data type to allow correct HDL code generation. As an example, this process can be done by setting the output equal to fi(constant,16,5). This will set the output to a 16-bit unsigned integer the same value as "constant" with 5 fractional bits.

Once HDL code is generated, this segment must be found and replaced by an instantiation of the existing code desired to integrate. If there are no changes in data type, the model can be generated and the existing code can be instantiated in the model, inserted where desired. If any changes occur in the model, the model can be regenerated and individual blocks of the new HDL code can be added to the project without changing the top module. Lastly, individual components or partial modules including several blocks can be created and integrated into larger projects, the same way specific IP cores can be generated but with more possible operations.

BPSK MATLAB Coded Block

```
function [sym_re, sym_im] = Existing(bin)

sym_re = sfi(1,2,0);
sym_im = sfi(0,2,0);

end
```

Figure 4.20: Code used as placeholder for existing VHDL after generation.

The BPSK model was altered and HDL code generated to show the integration of existing code into the model in the place of the BPSK modulator. Figure 4.19 shows an example of the model, while Figure 4.5 shows the code placed inside the MATLAB function block. The MATLAB function block was added and the name, the inputs, and the outputs of the function were declared. Since the input and output data types of the BPSK modulator differed, the input named "bin" was disconnected. Since the outputs were a signed 2-bit

fixed point, the sfi() function was used with a constant value. These steps ensured that the inputs and outputs are handled properly during code generation. The HDL code that was produced contained a file with the correct inputs and outputs, of which can be removed in Xilinx ISE and replaced with the actual desired code. If the name given underneath the block is the same as the existing HDL coded module, the instantiations will not need to change in the top module. Only the HDL generated place holder module will need to be removed from the project, while adding the existing HDL coded module to the project. This method creates the quickest integration cycle.

4.6 Chapter Summary

Chapter 4 discussed the implementation of several different models and implementations in detail. The first model discussed was the BPSK model-driven design, which was created by Simulink, converted to VHDL with HDL Coder Workflow Advisor, and verified operational by hardware. The second model described was the BPSK hand-optimized design, which took the BPSK model-driven design and made optimizations, verified operational by hardware, for the purpose of determining the efficiency of the HDL Coder tool. The third model is a Simulink model that converted the BPSK model-driven design to a QPSK model for the purpose of describing the ease at which designs can be changed and converted to a HDL. The next model presented how MATLAB blocks that are not available in Simulink can be created using MATLAB code in a MATLAB function block. The model used the BPSK model-driven design to create a BPSK modulator with MATLAB code. The final model presented showed the ease of creating a model to allow easy integration of existing VHDL code using HDL Coder. This model created a shell or placeholder for the BPSK modulator that the existing VHDL would replace.

Chapter 5

Results

Two models were developed for the purpose of determining the effectiveness of the HDL Coder as a viable tool for academia and industry. A model-driven approach was taken to build a BPSK communications system using a Simulink model, and then converted to VHDL code using HDL Coder. The second design took the model-driven version and hand-optimized it to create a hand-optimized BPSK communications system. In order to determine the performance of HDL Coder as a viable tool, these two designs were compared in various areas. This chapter will first discuss the hardware verification process that verified that both models could actually target and operate on the Zedboard containing an Zynq-7000 FPGA. Next will be an overview of the hardware utilization followed by the timing analysis and constraints to show and compare these two designs for their implementation on hardware. After a review of the hardware will be a discussion of the design process of MATLAB HDL Coder. Finally, there will be a section for the determined best practices using HDL Coder over the course of this project.

5.1 Performance Assessment

The performance assessment section compares the two designs, one model-driven and one hand-optimized. First, will be a detailed description of the process of hardware verification. Next, will be a discussion of hardware utilization between the two designs. Finally, the

timing information and testbenches will be presented.

5.1.1 Hardware Verification

Part of the requirement for developing any design for use on an FPGA is that the HDL code actually works on the target platform. Sometimes simulation will work for testbenches when the design will not work on hardware. As discussed in Chapter 3, each design was implemented on the board by using an error counter to perform the hardware verification test. The error counter delayed the output of the data source to match the known delay of the data path of the system using a shift register. If the two signals were different, an enable was produced to increment the counter that kept track of the number of errors. The number of errors was then displayed on the LEDs of the Zedboard. To verify that a design was working correctly, the design was loaded to the FPGA and reset to ensure correct reset states for all of the registers and counters of the system. Next the error counter was reset to remove all the errors induced by the system before the system was reset. After reset of the error counter, errors continued to occur due to the reset itself, which filled the data source delay registers with zeros. Once data from the data source was shifted out, the sequences were aligned and the LEDs held at a fixed number. The absence of new errors showed that the system was working.

To verify that the error counter operated correctly, a testbench was created to test the conditions of a count (error), the condition of no count (no error), and the reset, as shown in Figure 5.1. After verifying the testbench, the hardware verification test was performed on both the model-driven design and the hand-optimized design. Knowing that the PN sequence repeats after 2.52 micro seconds, a proposed time requirement of 3 seconds without errors was used to verify correct operation, as defined in Chapter 3. Each passed the hardware verification time requirement shown in Chapter **??** and therefore are considered operational designs. To verify that the error counter was working correctly on hardware, an incorrect delay was added to both designs as well. By doing so, this caused the counter to continually count, which meant errors were occurring. The culmination of the testbench showing correct results, the error counter operating as expected for the correct delays (on

both designs), and the error counter operating as expected for the incorrect delays (on both designs) was strong enough evidence that the error counter and each design were operating correctly.

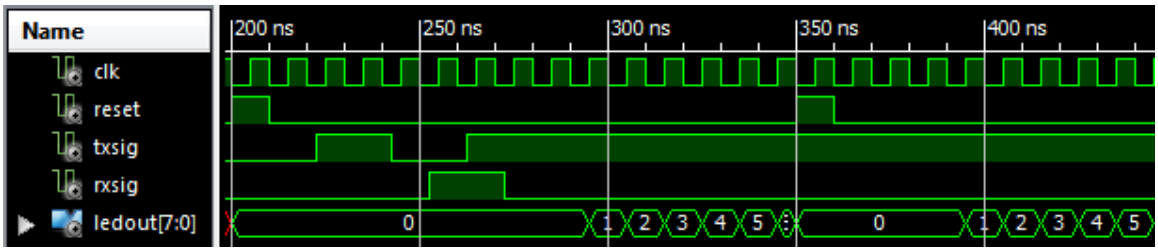

Figure 5.1: Simulation of testbench for error counter. The error counter was given a delay of 3 (to fit page width), which created a 3-bit shift register for the input signal "txsig". The signals started off identical with a delay until txsig went high and the rxsig stayed low. On the 3rd clock cycle after txsig went high, the output of the delay register went high as well causing txsig and rxsig to be different. This caused the counter enable to go high and, on the 4th clock cycle, the counter started counting. After several clock cycles, the reset was enabled which caused the counter and register to reset to all zeros. After 4 more clock cycles, the counter resumed counting again.

5.1.2 Hardware Utilization

The HDL languages and FPGAs have been the desired approach for many types of applications, however resources are physically limited without using modern reconfigurability approaches. Until recently, model-driven tools have been thought of as underdeveloped and underutilized for many reasons including a consensus that they will not be as efficient as handwritten methods. The findings for hardware utilization, discussed in this section, agree with this statement. However, experimentation on individual components highlighted places for improvement in the future of HDL conversion and model-driven design.

Solely based on visual inspection between the HDL coder generated VHDL of the BPSK model-driven design and the hand-optimized BSPK design, it was clear that the hand-optimized version had saved hardware resources due to less combinational logic statements, primarily in the FIR interpolation filter. The main differences in hardware reduction are

shown in Table 5.1. Each of these differences contributed to a noticeable reduction in hardware usage between the two designs.

Table 5.1: Table of all major hardware differences between the model-driven design and the hand-optimized design.

Hardware Differences			
Module	**Component**	**Model-Driven**	**Hand-Optimized**
CommSystem Top Module	Delay Match Pipelines	2 x 16-bit	Removed
FIR Interpolation Filter	Bit Reductions w/Saturation Checks	42 x 18-bit 1 x 17-bit	Removed Removed
	Pipeline Registers	21 x 18-bit 22 x 17-bit	21x 16-bit 22 x 16-bit
	Adders	40 x 18-bit	40 x 16-bit
	Data Hold Register	1 x 16-bit	Removed
	Multiplexer (2 Input)	1 x 16-bit	Removed
FIR Decimation Filter	Bit Reductions w/Saturation Checks	1 x 33-bit	Removed
	Data Hold Register	1 x 16-bit	Removed
	Multiplexer (2 Input)	1 x 16-bit	Removed

Hardware utilization summaries were generated for both systems after removing the error counter, as shown in Table 5.2. The hardware was targeting the Zynq-7000 SoC, device XC7Z020, package CLG484, with speed grade -1. The table in Figure 5.2 highlighted some of the important numbers. The most apparent difference is the number of DSP48E1s that were consumed under the BPSK model-driven design. The model-driven design used 158 of the 220 DSP48E1s (71%), whereas the hand-optimized version used only 79 of the 200 (35%), representing a reduction by half of the DSPs.

The hand-optimized design reduced both the number of occupied slices and the number of slice LUTs used almost in half, and of the total, by 9% and 7% respectively. The number of slice registers was reduced almost in half, from 10,897 to 6,428 or 4% of the total 106,400 slice registers. There were 3% more unused LUTs of LUT Flip Flop pairs in the hand-optimized design.

Table 5.2: Hardware utilization report comparison between model-driven BPSK system and hand-optimized BPSK system on the Zynq-7000 SoC XC7Z020. Bold text represents the most significant differences.

Device Utilization Summary					
		Model Driven		Hand Optimized	
Slice Logic Utilization	Available	Used	Utilization	Used	Utilization
Number of Slice Registers	**106,400**	**10,897**	**10%**	**6,428**	**6%**
Number used as Flip Flops		10,897		6,428	
Number of Slice LUTs	**53,200**	**8,154**	**15%**	**4,557**	**8%**
Number used as logic	53,200	7,136	13%	3,820	7%
Number using O6 output only		5,553		3,023	
Number using O5 output only		6		6	
Number using O5 and O6		1,577		791	
Number used as Memory	17,400	19	1%	0	0%
Number used as Shift Register		19		0	
Number using O6 output only		19		0	
Number used exclusively as route-thrus		999		737	
Number with same-slice register load		942		737	
Number with same-slice carry load		57		0	
Number of occupied Slices	**13,300**	**3,055**	**22%**	**1,792**	**13%**
Number of LUT Flip Flop pairs used		10,965		6,336	
Number with an unused Flip Flop	10,965	2,778	25%	1,353	21%
Number with an unused LUT	10,965	2,811	25%	1,779	28%
Number of fully used LUT-FF pairs	10,965	5,376	49%	3,204	50%
Number of unique control sets		9		8	
Number of slice register sites lost to control set restrictions	106,400	20	1%	28	1%
Number of bonded IOBs	200	6	3%	6	3%
Number of BUFG/BUFGCTRLs	32	2	6%	2	6%
Number used as BUFGs		2		2	
Number of DSP48E1s	**220**	**158**	**71%**	**79**	**35%**
Average Fanout of Non-Clock Nets		2.74		2.9	

These differences in hardware occur from the removal/reduction of many pipeline registers, the reduction in data size of adders, and the removal of many bit reductions. Overall, it is clear that the hand-optimized design used almost half of the resources that the model-driven design used. On a larger scale project, using the same target hardware, this extra usage might be of concern.

5.1.3 Timing Analysis and Constraints

The structure of the communications system design, including two filters with many layers of multipliers and adders, poses as a good candidate for analyzing the timing and constraints. Evaluation of the delays involved can be important for design concerns of someone thinking about using a tool such as HDL coder. Adding pipeline, input, and output registers allows for keeping within timing constraints. There are trade offs between faster clock frequency and hardware usage that affect the timing constraints in a design.

The BPSK model-driven design included input registers between the BPSK modulator and the FIR interpolation filter, between the FIR decimation filter and the BPSK demodulator, after the BPSK demodulator, and 3 between the FIR interpolation filter and FIR decimation filter. The FIR interpolation filter contained 5 pipeline registers and the decimation filter contained 7 pipeline registers. All of these registers, including the delays produced by the convolutions, caused a total delay of 128 clock cycles between corresponding input and output. The hand-optimized version reduced the clock cycle delays between corresponding input and output to 97 clock cycles. There was a reduction of 31 clock cycles throughout the system. The hand-optimized design was shown to reduce total delay in the system by about 25% while still operating correctly on hardware. This value is significant given that 80 clock cycles of the delay are used by the convolution itself. The model-driven design contained 48 register delays, while the hand-optimized design contained 17 register delays, which is almost a 65% register delay reduction.

Timing constraints of elements between registers must be performed for analysis. In this design, register placement was defined to ensure the hardware utilization comparison was fair between the model-driven design and the hand-optimized design. Through the hand-optimization process, the maximum timing constraints were not changed. Therefore, the maximum clock frequency, or minimum period, was unaltered between designs. The maximum clock frequency for both designs was 197.694 MHz and the corresponding minimum period was 5.058 ns. These values were well within the 100 MHz operating clock

frequency of the design.

Figure 5.2 shows the simulation of the model-driven design system and Figure 5.3 shows the simulation of the hand-optimized system. These figures show all of the important internal signals. Notice the propagation of the first input value through each system by the data transitions from zero to a value other than zero. Due to there being many filter pipeline registers, the inputs of the interpolation and decimation filters were always followed through the multipliers/adder trees. Notice the large reduction of register delays of the decimation filter for the hand-optimized design. The model-driven design uses the same 25 MHz enable signal for each pipeline within the filter causing a value to take 4 clock cycles per pipeline register, whereas the hand-optimized design has different enables for each pipeline allowing the value to propagate through to the next pipeline register on each clock cycle. Table 5.3 shows the delays involved in the system.

Table 5.3: Table of register delays for the model-driven and hand-optimized designs.

CommSystem Delays			
Module	**Internal Components**	**Model Driven**	**Hand Optimized**
PN Sequence	Data Source		
BPSK Modulator		0 Cycles	0 Cycles
Input Pipeline 1		4 Cycles	Removed
FIR Interpolation Filter	Input Shift Register	N/A	2 Cycles
	Pipeline Registers 1-5	5 Cycles	5 Cycles
Input Pipeline 2		1 Cycle	Removed
Delay Match Register 1		1 Cycle	Removed
Delay Match Register 2		1 Cycle	Removed
FIR Decimation Filter	Input Shift Register 1	N/A	1 Cycle
	Pipeline Registers 1-7	25 Cycles	7 Cycles
	Data Hold/MUX	3 Cycles	Removed
Input Pipeline 3		4 Cycles	1 Cycle
BPSK Demodulator		0 Cycles	0 Cycles
Output Pipeline 1		4 Cycles	1 Cycle
	Total Delays	**48 Cycles**	**17 Cycles**

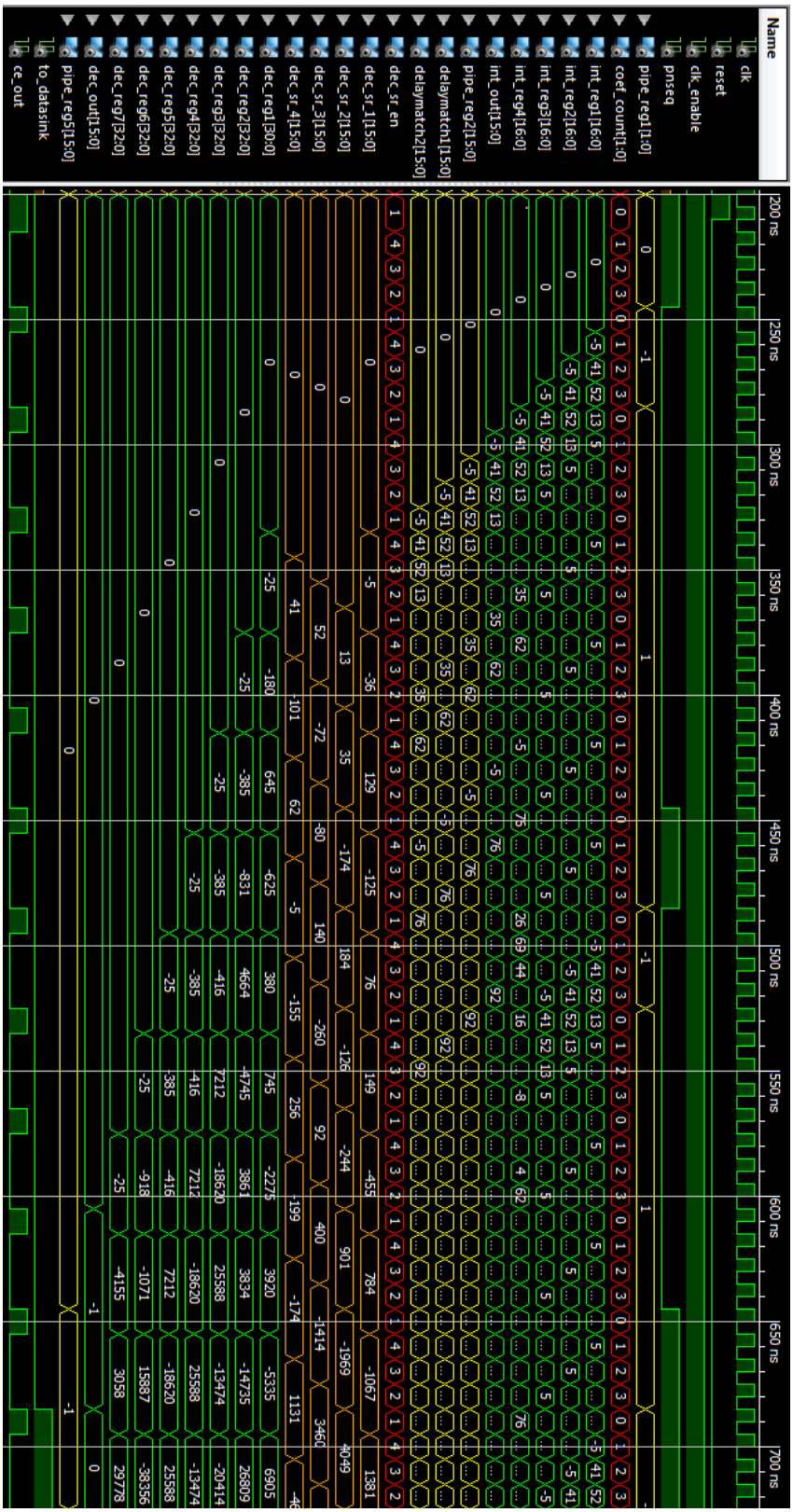

Figure 5.2: Simulation of testbench of model-driven BPSK communications system.

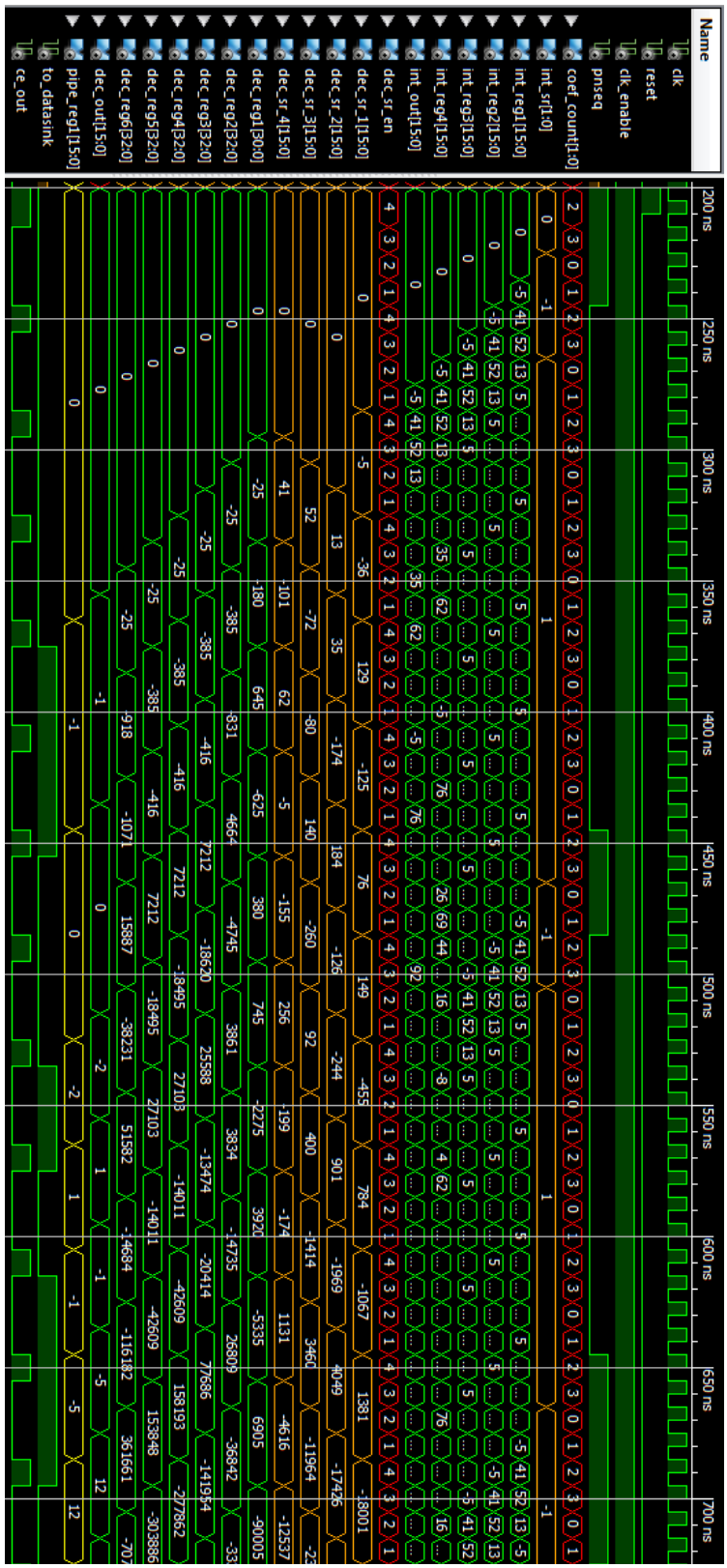

Figure 5.3: Simulation of testbench of hand-optimized BPSK communications system.

Each component will be briefly discussed for comparison between designs. The first component is the BPSK modulator, which did not require a clock. However, testbenches were created to show correct operation between the model-driven and hand-optimized designs. Figure 5.4 shows the model-driven version and Figure 5.5 shows the hand-optimized version, performing the desired operations. Each testbench was written to verify the results using the ASSERT statement.

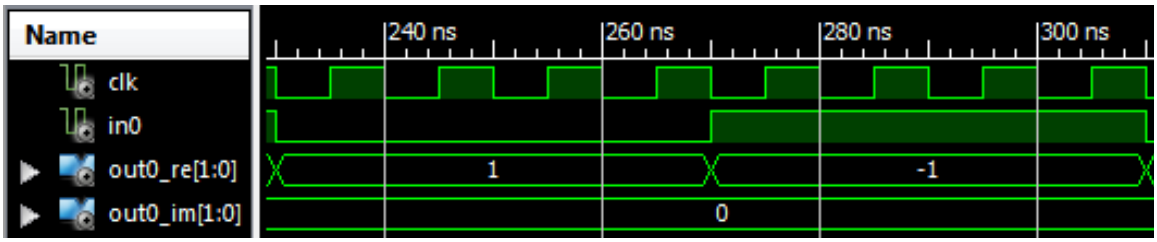

Figure 5.4: Simulation of testbench for model-driven BPSK modulator.

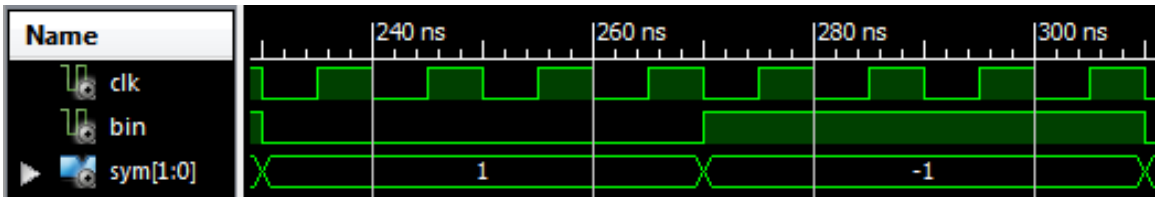

Figure 5.5: Simulation of testbench for hand-optimized BPSK modulator.

Testbenches were also created to test the outputs of the BPSK demodulator for each design. Figure 5.6 shows the model-driven design and Figure 5.7 shows the hand-optimized design. Due to the model-driven design including imaginary values, this input was tested as well. The model-driven demodulator used 3 conditional signal assignments from the real and imaginary inputs and fed them to a 3 input LUT which produced the corresponding output. One of the signal assignments checked the imaginary input that was equal to zero, and reflected by the input alternating from 0 to 1. However, the imaginary input was a constant zero in the design. Therefore, the conditions when the imaginary input equaled one never actually occurred in the design. The hand-optimized design only checks the sign

bit of the input signal to determine the output. Both designs performed the correct operations with inputs that could actually occur. Each testbench was written to verify the results using the ASSERT statement as well.

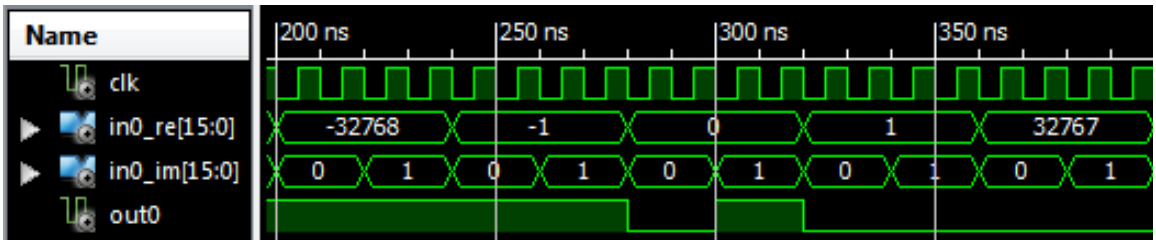

Figure 5.6: Simulation of testbench for model-driven BPSK demodulator. Imaginary input was a constant zero in the design. Therefore, the conditions that the imaginary input equals one never occurred and had no effect on the performance of the design.

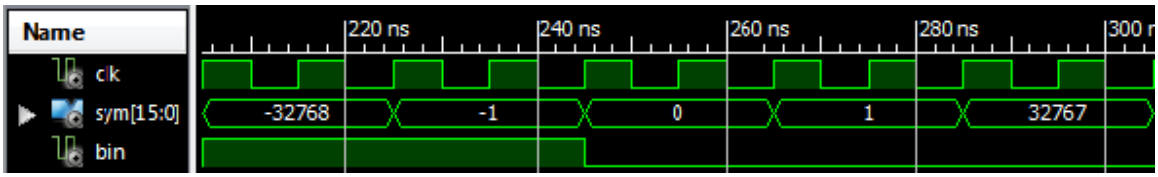

Figure 5.7: Simulation of testbench for hand-optimized BPSK demodulator.

The interpolation filter received two distinct types of testing for both designs. The first sent an impulse through the filter to verify the impulse response. An impulse as an input showed all of the coefficients were correct and also showed the data propagation delays within filter. Figure 5.8 shows the model-driven design and Figure 5.9 shows the hand-optimized design. The 2-bit counter, labeled “phase”, shown in both simulations, was used to select the subfilter for multiplication with the input value and input shift register values in the design. For the model-driven design, the first pipeline register is shown, labeled “int_reg1”. This pipeline register contained the result of the multiplication of the input with the first coefficient of each subfilter. For the hand-optimized design, the signal denoted ”int_reg1” is the most recent input of input shift register, which was moved inside the filter as discussed in Chapter 4. Notice that this register shows the input of the filter with a delay. The signal denoted ”int_reg2” contained the result of the multiplication of

the input shift register, “int_reg1”, with the first coefficient of each subfilter. Notice that “int_reg1” of the model-driven filter and “int_reg2” of the hand-optimized filter correspond with each other.

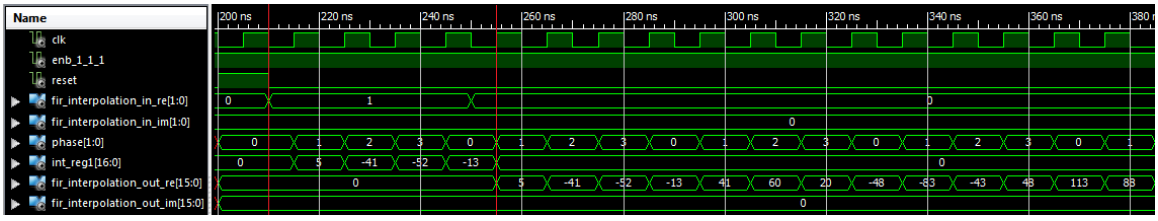

Figure 5.8: Simulation of testbench of model-driven FIR interpolation filter with an impulse as an input. Output produced coefficients of filter. The first red line shows the transition of the input of the filter, which is multiplied and latched by the first pipeline register “int_reg1” on the next positive clock edge. The output is produced 4 clock cycles later after data propagation through the remaining 4 pipeline registers. The second red line denotes the output of the first input value. The difference between the two red lines show the propagation delay between the filters input and output.

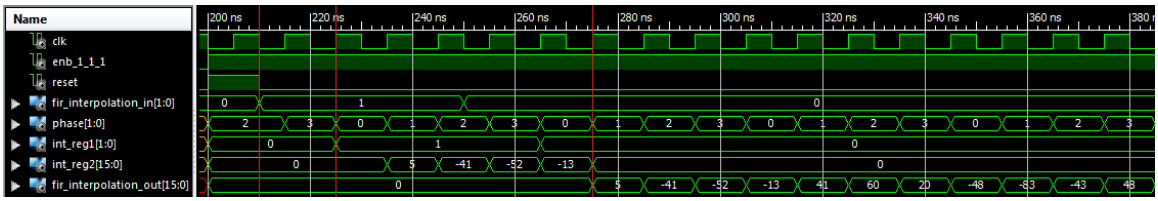

Figure 5.9: Simulation of testbench of hand-optimized FIR interpolation filter with an impulse as an input. Output produced coefficients of filter. The first red line shows the transition of the input of the filter, which is passed to the input pipeline register “int_reg1” and latched after a count of 3 of the 2-bit counter “phase”. The second red line shows the output of the input pipeline register. The output is produced 5 clock cycles later after data propagation through the 5 pipeline registers of the filter. The third red line denotes the output of the first input value. The difference between the first and third red lines show the propagation delay between the filters input and output. The difference between the second and third red lines show the propagation delay of operations similar to the operations of the the model-driven filter.

As can be seen in both simulations, the input goes from zero to one for 4 clock cycles to allow all four subfilters to multiply the first coefficient with the input. The input goes to zero for the remainder of the test. The model-driven filter’s first register shows the result of these multiplies, and goes to zero after the input does. The output of the model-driven

filter shows the results of the impulse as the input is shifted in the input shift register to be multiplied with alternate coefficients. The output produced all 81 coefficients in order. The hand-optimized filter simulation performs in a similar manner with the exception of the input pipeline register "int_reg1", which contained the delayed input.

The test benches for each design were written to incorporate the individual delays of each version of the filter. However, each was tested such that the coefficients were output properly. Each testbench was written to verify the results using the ASSERT statement, and both were correct for all the outputs. The model-driven design by visual inspection has a delay of 4.5 clock cycles which correspond to the 5 delays of the pipeline registers in the design. During normal operation the input would change one full cycle before being latched by the first pipeline register. The hand-optimized design by visual inspection has delay of 7.5 clock cycles, which correspond to the delay of the input pipeline register (enabled on count 3) and the 5 delays of the pipeline registers. Although the filter delay is longer for the hand-optimized filter, adjustments made to the system actually make propagation of data from the data source through the filter shorter, as described in Chapter 4. Ultimately, adjustments reduced 2 clock cycles worth of delays in the system through output of the interpolation filter.

The next test performed on the interpolation filters was the maximum output test, which was to verify that by removing all of the bit reductions with saturation checks the output of the filter would produced correct results without rollover. Calculations showed that the removal of these bit reductions would not affect data accuracy, however this was important to verify by testbench. The input sequence that would cause the maximum output possible for the system, which was the sum of the absolute value of subfilter 3's coefficients, the output of the system did not rollover. The maximum value possible was 29584 for subfilter 3, while the other values output during the 4 cycle input data period were 16637, 26106, and 26106, respectively. The correct output for this test was determined by a MATLAB Interpolation filter model and verified with testbench using ASSERT statements. Both testbenches reported correct results for 80 data outputs. Figure 5.10 shows the model-driven design and

Figure 5.11 shows the hand-optimized design. The red line denotes a lapse in time. The second to last output shows the correct maximum value for both filter designs. These results show that removing all of the saturation checks did not affect the accuracy of the filter.

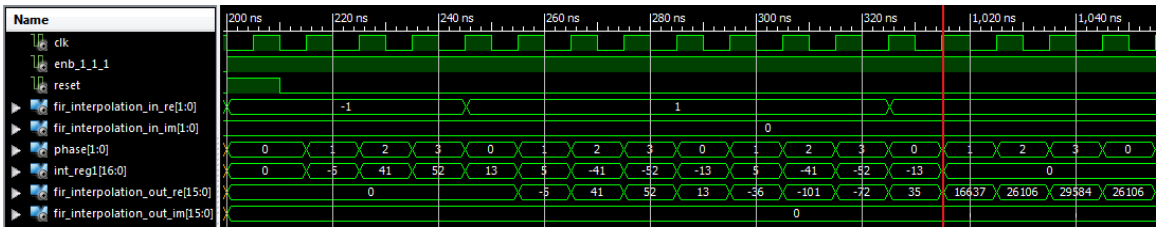

Figure 5.10: Simulation of testbench of model-driven FIR interpolation filter with an input sequence causing maximum output to check for saturation. The red line denotes time lapse. The second to last output represents the maximum value possible for the filter output.

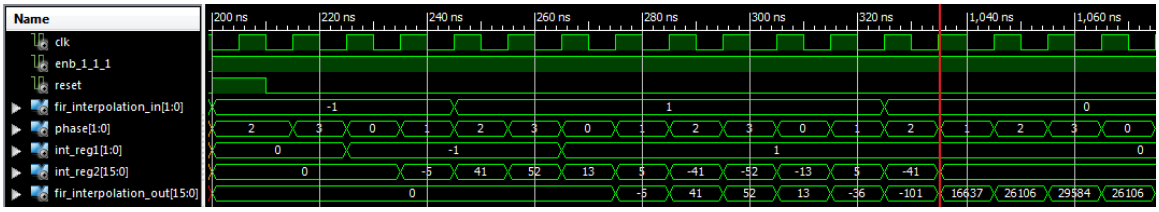

Figure 5.11: Simulation of testbench of hand-optimized FIR Interpolation Filter with an input sequence causing maximum output to check for saturation. The red line denotes time lapse. The second to last output represents the maximum value possible for the filter output.

The FIR decimation filter only received one test because of its structure. A MATLAB model was used to determine the first 100 values of the system starting with the PN sequence and up to the decimation filter input, to be used as input for the testbenches. The model was also used to compute the expected output of the filter. These values were then used to create a testbench for each design that tested the correct functionality. Each testbench was written to verify the results using the ASSERT statement, and both were correct for all the outputs. Figure 5.12 shows the model-driven design and Figure 5.13 shows the hand-optimized design. The input sequences were aligned by hand due to the decimation filters requirement that they be aligned to function properly. Each testbench shows the most recent inputs in the input shift registers of the filter. Notice that by visual inspection

that the output of the model-driven design has a 28 clock cycle delay, while the output of the hand-optimized design has a 8 clock cycle delay. This difference is due to adjustments of enables between the pipeline registers that use a 25 MHz clock. Each subsequent register was given an enable one clock cycle later than the previous pipeline register. A total savings of 20 clock cycle register delays was seen in the decimation filter. Much like the significant reduction in hardware usage of the interpolation filter, the decimation filter significantly reduced register delays.

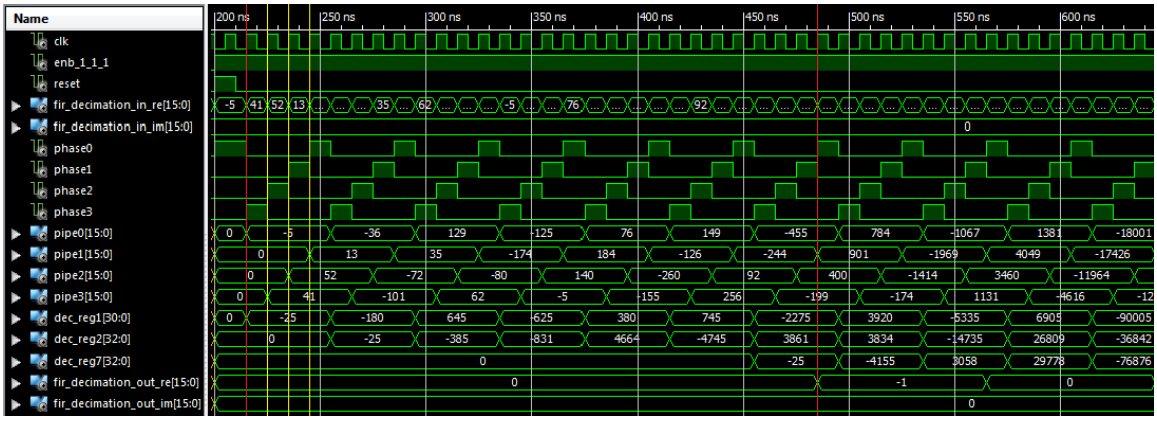

Figure 5.12: Simulation of testbench of model-driven FIR decimation filter with an input of the PN generator after output from the FIR decimation filter. The first red line shows the first input (-5) propagating into shift register 1 "pipe0" and the first pipeline register "dec_reg1". Each yellow line shows inputs (41, 52, 13) propagating into input shift registers (4, 3, 2) respectively. The second red line shows the data propagation out of the filter.

In total the components built by MATLAB HDL Coder were not very efficient for timing, there were not enough simple optimizations produced, especially in the decimation filter. Even though the user must specify the register placement, it is obvious that HDL Coder does not try to place the pipeline register enables better. All of the enables should be generated in the timing controller module and fed into an already optimized block. This would have removed the need for the delay match registers required between the interpolation filter and the decimation filter to allow the decimation filter to work properly. A thorough understanding of the resultant hardware design was needed before any of the improvements were made with the hand optimizations, which is not ideal for a user of a

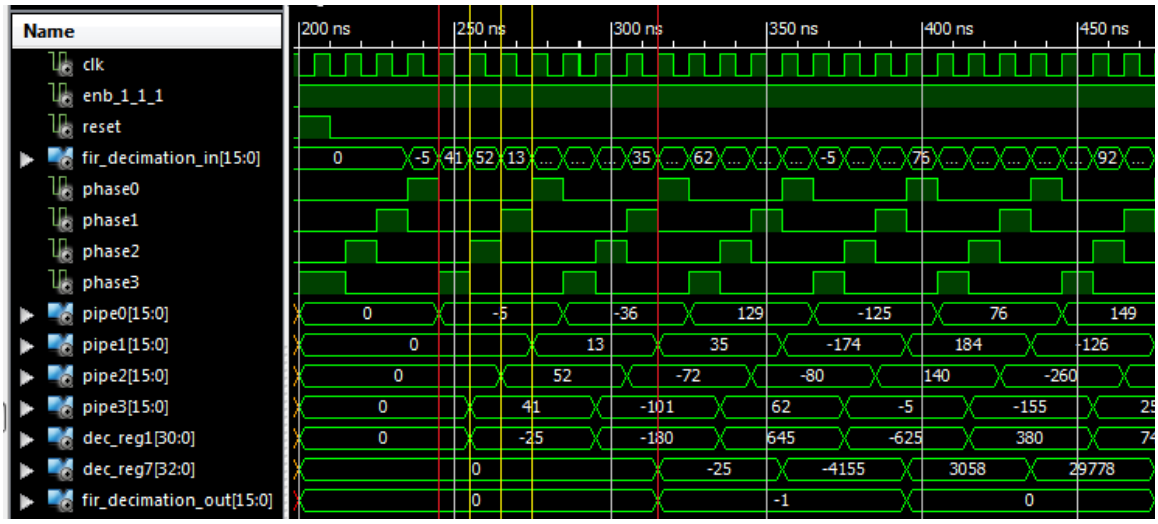

Figure 5.13: Simulation of testbench of hand-optimized FIR decimation filter with an input of the PN generator after output from the FIR decimation filter. The first red line shows the first input (-5) propagating into shift register 1 "pipe0". Notice that the data did not propagate into the first pipeline register "dec_reg1" until one clock cycle later than in the model-driven design, due to the input pipeline register being moved inside the filter. Each yellow line shows inputs (41, 52, 13) propagating into input shift registers (4, 3, 2) respectively. The second red line shows the data propagation out of the filter.

model-driven tool to have to do manually. In a much larger project, if excess register delays of 2.8 times is acceptable, not including the hardware utilization losses, then HDL Coder could present an acceptable tool to a hardware engineer, who has the knowledge and experience to understand where to place registers (since this is done almost exclusively by hand).

5.1.4 Development Time Requirements

Development time is one of the main motivations for using model-driven tools. Reducing complexity while increasing abstraction allows the designer to implement designs at faster rates. The ease created by model-driven tools like HDL Coder allow design teams to test algorithms before the lengthy HDL code design cycle. A model created in HDL Coder may take on the order of several minutes to a couple hours, while writing HDL from scratch can take several hours to days. Any changes that occur in the design of handwritten HDL code can take hours to days to correct as well. Although a tool like HDL coder may not have

the advantage with hardware utilization, it has an advantage over handwritten HDL code for development time requirements.

During the process of model creation for the BPSK model-driven design, there were many hours of testing and HDL code generation. Part of this was due to learning the Simulink requirements for HDL Coder and how it worked. After learning the process of learning HDL Coder, it would take 10 minutes to select the blocks, 20 minutes to select the data types, set the block properties, and testing in MATLAB, 5 minutes for HDL code generation, 1 hour for review of HDL code for determining additional register placement, and 10 minutes to set the HDL block properties and code regeneration. Some of these values may vary, but from beginning to end it should take less than 2 hours for someone with HDL Coder experience to redesign the BPSK model-driven design.

Hand coding of the BPSK model from scratch would take on the order of a couple days. The FIR Interpolation filter produced by the HDL coder was over 1300 lines of code, while the FIR decimation filter was over 3600 lines of code. These values do not include the short simple BPSK modulator and demodulator, or the longer top module with the timing controller and pipeline registers. Two hand-optimized versions were created during this project. One was not mentioned in Chapter 4 because the model was changed, and it was easier to rework the entire hand-optimized design than to convert it. Both instances of hand-optimized code used MATLAB's sprintf function to print HDL code to screen that could be copy and pasted into the models. This method reduced time for hand-optimized code, but it still took over 8 hours for both because of mistakes during development. The FIR decimation filter contained over 00 signals, many necessary to perform all of the resizes and saturation checks. Writing all of these without using MATLAB's sprintf function to auto-generate a bulk of them could take hours by itself, let alone writing counters, registers, or other operations. The thought process involved in determining the optimum data sizes for large operations can be a time consuming process as well.

Converting a model to include different components or integrating existing VHDL code

will take advantage of the model-driven design development time requirements of HDL Coder as well. Integrating or instantiating existing code into a model is very direct and fast, thus the time requirements to build the model around it can generally be as fast as mentioned above. A simple model created of only the FIR decimation filter could generate 3600 lines of code in 5 minutes. Companies and universities could use model-driven tools such as HDL coder to aid in the design of prototypes to show functionality before applying optimizations to the design later.

5.1.5 Design Process

The design process for Simulink combined with HDL Coder is a very simple process that is very powerful as well. The advantages for the design process using HDL Coder outweigh the complications with handwritten code. Once the model is setup for HDL Coder, the design process is implemented by selecting blocks to perform system functions of the design. This process can be done with Simulink in a GUI interface, or as MATLAB code functions. The advantages really lay with integrating blocks that implement complicated functions (such as filters, FFTs, and convolutional encoders), testing design for bit accuracy in simple environment, and making adjustments to model that change generated HDL code. Each of these provides benefits that hand coding does not offer, with the exception of IP core usage, which does not generate the top module.

A small change to data sizes or types as well as adding pipeline registers can be as simple as double clicking on a block and editing numbers. The model can then be tested for accuracy inside Simulink, converted to HDL, and tested for functionality related to timing. Changes that would be complicated with handwritten HDL are quick and simple with HDL Coder. Integration of existing code becomes straightforward and easy to integrate once the model is converted to HDL.

Once the model is proven accurate in the Simulink environment, HDL Coder will generate code and a testbench, create the project in the targeted third-party IDE, perform logic

synthesis, map, and place and route to verify code operates without errors. A quick transition to the IDE such as Xilinx ISE allows for running testbench and bitstream generation to download to a target device. HDL Coder provides easy model to target device flow while providing the tools for testing and quick editing of design.

5.2 Best Practices

This section will provide some insight for the best practices for using Mathworks HDL Coder tool in the Simulink environment, and are as follows:

- Due to the settings adjustments to allow Simulink to work with the HDL Coder tool, a user should always save a blank template for future use. This process allows the user to start a new model without the hassle of changing Simulink settings to work with HDL Coder and can move on to the design aspect immediately.

- The user should always convert to fixed point data types or select smallest data type when possible. Many blocks may not be able to be generated without a fixed point data type and it provides an easier process when trying to match data types between blocks. Also, by specifying a fixed point or smallest data type, the design will not generate extraneous code.

- The user should save the list of compatible blocks for HDL Coder. This will allow easy determination of the possible blocks that can be selected from when creating a project. Since HDL Coder will only accept certain blocks for HDL code generation, this list will be essential.

- The user should consider building subsystems or modules from the basic components, such as adders, multipliers, and registers. These designs generate more desirable results due to the design being more explicit, which would alleviate an issue of hardware utilization and timing constraints seen during this project. This method can also allow previous designed subsystems to be reused by a simple copy and paste between Simulink models.

- Lastly, use the methods described in Chapter 4 to integrate existing code or to generate MATLAB coded blocks. These methods will allow HDL Coder to be used beyond the scope of its limits with generation, and allows faster development of projects as well.

5.3 Chapter Summary

Chapter 5 discussed several topics related to the performance and potential of HDL Coder. A detailed performance assessment was reported to include hardware verification, hardware utilization, timing constraints, and development time requirements. The hand-optimized design outperformed the model-driven design significantly, for both hardware utilization and timing constraints. However, for development time requirements, the model-driven design created by HDL Coder significantly outperformed the hand-optimized model. The chapter also discussed the typical design process behind using HDL Coder, as well as best practices that were learned by using the tool.

Chapter 6

Conclusions and Recommendations

Academia and industry are both looking for methods to increase productivity, lower development costs, and reduce learning curves, while still developing a high quality product. In the area of FPGA-based design, high-level synthesis and model-driven design tools have been the candidates to alleviate many of these issues that have plagued development of FPGA-based systems. There have been many model-driven design tools being developed and utilized in recent years to accomplish these goals, including:

- Xilinx HLS tool
- Altera OpenCL
- Xilinx System Generator for DSP
- Altera DSP Builder
- Mathworks HDL Coder

In order to determine the effectiveness of these tools to alleviate the issues in FPGA design, evaluations on their performance must be conducted. This thesis performed an evaluation of the model-driven design tool Mathworks HDL Coder by developing an FPGA-based SDR system.

This evaluation was accomplished by developing two designs to conduct performance assessments for comparison. In the first design, a Simulink model of a BPSK model-driven design was developed, converted to VHDL using HDL Coder, and implemented on an FPGA. The second design was a hand-optimized version of the BPSK model-driven design's VHDL code, which was implemented on the same FPGA and used for comparison. The performance assessment included hardware verification, hardware utilization, timing constraints, and development time requirements. The processes involved from implementation to the performance assessment gave insight and conclusions about the capabilities of the tool, with respect to the design implemented.

The first major insight to HDL Coder was that it could be used to create a design, from creation of the model to HDL conversion, much faster than a hand-optimized approach. With this knowledge, a design could be developed, including the integration of existing code, in a fraction of the time. However, there are some caveats to HDL Coder. HDL Coder does not remove all of the learning curves of hardware design and HDL code. In order to use HDL Coder it was necessary to understand the structure of the generated HDL code in order to create a design that could actually function on an FPGA. HDL Coder produces cycle accurate code, however it does not include important timing analysis of the hardware itself, requiring the user to add important registers or pipelines where needed. Knowledge of the design is necessary in order to accomplish this step, which requires knowledge of both hardware design and HDL code.

As mentioned, HDL Coder requires a learning curve to understand the tool. Once the tool is understood, it can be utilized to create designs at a much faster rate and used to it's fullest potential. Designs remain in a visual representation in the Simulink environment allowing testing and easy understanding of the design. In an IDE with VHDL code, testbenches must be created to test the design and the user must read the code to understand the design. Simulink and HDL Coder enable easier code reuse by a simple copy and paste between Simulink models. They also allow changes in data sizes by simply adjusting the settings of a single block, that can automatically propagate changes to follow on blocks. In

a handwritten design, this change requires much more detailed HDL code to be written in order to replicate the same effect. For reasons that were determined during the project, HDL Coder can potentially be used to increase productivity once the learning curve has been bridged. Prototype development could also benefit with HDL Coder by enabling faster development times of working models, by utilizing a simple testing environment that allows accuracy checks before and after HDL code generation. Although, optimizations may need to be made for the final product.

Although HDL Coder may be used to increase productivity, it was proven to not be as efficient for hardware utilization or timing constraints. The model-driven design consumed almost twice as much resources in each category than the hand-optimized design did. This issue stemmed directly from a lack of optimizations by HDL Coder. One issue was created by HDL Coder producing a signal chain for the imaginary components, which propagated throughout the design, which should not have been created in the first place. This was optimized out by Xilinx ISE during the synthesis stage and was not included against the hardware utilization. The second issue arose due to a polyphase interpolation filter that could be exploited in the hand-optimized design. By knowing the inputs, the maximum value that could be produced by the output was determined, which allowed the size of adders and various other elements to be reduced, without sacrificing any accuracy of the data. This filter could even be created in Simulink using basic blocks and use the same resources as a handwritten approach would. Therefore, an understanding of the tool, the hardware implemented by each block, and a knowledge of hardware design are required to produce resource efficient designs to meet the needs of developing a quality product with this tool.

Timing and delays were an issue as well with the design implemented using HDL Coder. Register and pipeline placement was required to produce a functional design for an FPGA, however the resultant design added extra delays that were not needed. The model-driven design contained 2.8 times the amount of delay that the hand-optimized design contained. This was discovered by inefficiencies of the design, most prominent in the polyphase dec-

imation filter. These delays could have been avoided if HDL Coder optimized the design, or if the block was designed more efficiently. There is a need for timing to be analyzed and registers to be placed automatically for model-driven tools, primarily due to timing being one of the most difficult issues with HDL design. Although the data accuracy of the design can be tested very easily in Simulink, timing simulations cannot be performed until HDL code generation, making this process more difficult to correct for, and possibly implemented by hand. Timing is one of the larger concerns when considering this tool, especially if the application of the design requires minimal delays.

The primary concern of the designs being able to target an FPGA was met when both designs were operational on the board. This showed that HDL Coder could produce HDL code that was functional on the targeted device with relative ease, through an iterative process of model adjustments, Simulink testing, and hardware verification. However, there are still many enhancements to the tool that could improve the performance and design cycle for the hardware engineer. Documentation with better descriptions of the hardware structures, and settings that affect these structures, created by HDL Coder would greatly improve a hardware engineers knowledge of each block, which would also aid in register placement and understanding of timing. In the case of this project, HDL Coder did not create a design that was optimized for input data ranges otherwise imaginary components would be removed and hardware would be reduced in the interpolation filter. Therefore, optimizations should always be performed for hardware utilization and timing to create efficient designs that meet the needs of their targeted audience. With this ability, HDL Coder would be a tool that could truly perform the necessary tasks of model-driven design by allowing more abstraction.

Overall, HDL Coder has advantages for the users with a direct knowledge of the hardware design process and HDL languages. However, as with any tool, there is room for improvement with the end goal of a systems level approach to generation of designs. The primary difficulties with model-driven design of FPGA-based systems have not been totally solved with HDL Coder. This tool still requires a few more optimizations and added tim-

ing analysis to be truly an impressive model-driven tool. However, it still provides a real advantage for hardware engineers who may desire a tool to aid in the faster development cycles, similar to IP core generation with less options and optimizations. HDL Coder and Simulink could be much more valuable for visual models, third-party blocksets, and testing when combined with Xilinx System Generator or Altera DSP builder.

6.1 Future Work

There are several areas to the future work with this project. The first would be to integrate the VHDL code produced by the HDL Coder with the ARM processors and the FMComms1 RF front-end. Incorporating these components adds a real communications system element that can not be achieved for an SDR solely implemented on the FPGA fabric. This was investigated during this project and seen as a very ambitious task due to the maturity of the tools and hardware at the time, as well as necessary addition of extra hardware for equalization and various synchronization issues that arise. While the main components of the design would remain on the FPGA, the addition of a processor and FMComms board would alter the data path and would remove the data source from the FPGA. Data generation could occur in the processor, sent through the SDR transmit path in the FPGA fabric, and sent over-the-air through the FMComms1 board. The FMComms1 board would then receive this signal, send it through the SDR receive path in the FPGA fabric, and back into the processor for analysis and output to a monitor. Although this design structure may be difficult, it would be ideal for utilizing the full capabilities of the hardware.

The second approach would be to investigate other model-driven tools and design environments. Some of these include Xilinx's Vivado Suite and its HLS (High Level Synthesis) tool, Altera's SDK for OpenCL and hardware acceleration, Simulink for Xilinx System Generator, and Altera's DSP builder. These tools focus better on timing and scheduling that HDL Coder did not, allowing the higher level of abstraction desired by model-driven design. These tools will allow for model-driven solutions to the problems plaguing FPGA

development times and design costs.

These appendices contain the source code from the project, the testbenches, and MATLAB code. The first appendix contains the source code of the BPSK model-driven design, the BPSK hand-optimized design, the error counter, and the hardware verification integration modules. The second appendix contains all testbenches of the project. The final appendix contains the MATLAB system model used for developing the testbenches.

Appendix A

Source Code

A.1 BPSK Model-Driven Design

A.1.1 Top Module

```
-- -------------------------------------------------------------
--
-- File Name: CommSystem.vhd
-- Created: 2014-03-16 01:42:40
--
-- Generated by MATLAB 8.2 and HDL Coder 3.3
--
--
-- -------------------------------------------------------------
-- Rate and Clocking Details
-- -------------------------------------------------------------
-- Model base rate: 1e-08
-- Target subsystem base rate: 1e-08
--
--
-- Clock Enable  Sample Time
-- -------------------------------------------------------------
-- ce_out        4e-08
```

```
-- -------------------------------------------------------------
--
--
-- Output Signal                 Clock Enable  Sample Time
-- -------------------------------------------------------------
-- To_DataSink                   ce_out        4e-08
-- -------------------------------------------------------------
--
-- -------------------------------------------------------------



-- -------------------------------------------------------------
--
-- Module: CommSystem
-- Source Path: CommSystem
-- Hierarchy Level: 0
--
-- -------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE work.CommSystem_pkg.ALL;

ENTITY CommSystem IS
  PORT( clk                               :   IN    std_logic;
        reset                             :   IN    std_logic;
        clk_enable                        :   IN    std_logic;
        ce_out                            :   OUT   std_logic;
        To_DataSink                       :   OUT   std_logic  -- ufix1
        );
END CommSystem;
```

```
ARCHITECTURE rtl OF CommSystem IS

  -- Component Declarations
  COMPONENT CommSystem_tc
    PORT( clk                             :   IN    std_logic;
          reset                           :   IN    std_logic;
          clk_enable                      :   IN    std_logic;
          enb                             :   OUT   std_logic;
          enb_1_1_1                       :   OUT   std_logic;
          enb_1_4_0                       :   OUT   std_logic;
          enb_1_4_1                       :   OUT   std_logic
          );
  END COMPONENT;

  COMPONENT BPSK_Modulator_Baseband
    PORT( in0                             :   IN    std_logic;  -- ufix1
          out0_re                         :   OUT   std_logic_vector(1
            DOWNTO 0);  -- ufix2
          out0_im                         :   OUT   std_logic_vector(1
            DOWNTO 0)  -- ufix2
          );
  END COMPONENT;

  COMPONENT FIR_Interpolation
    PORT( clk                             :   IN    std_logic;
          enb_1_1_1                       :   IN    std_logic;
          reset                           :   IN    std_logic;
          FIR_Interpolation_in_re         :   IN    std_logic_vector(1
            DOWNTO 0);  -- ufix2
          FIR_Interpolation_in_im         :   IN    std_logic_vector(1
            DOWNTO 0);  -- ufix2
          FIR_Interpolation_out_re        :   OUT   std_logic_vector(15
            DOWNTO 0);  -- sfix16_En15
```

```
        FIR_Interpolation_out_im        :   OUT   std_logic_vector(15
          DOWNTO 0)  -- sfix16_En15
        );
END COMPONENT;

COMPONENT FIR_Decimation
  PORT( clk                             :   IN    std_logic;
        enb_1_1_1                       :   IN    std_logic;
        reset                           :   IN    std_logic;
        FIR_Decimation_in_re            :   IN    std_logic_vector(15
          DOWNTO 0);  -- ufix16
        FIR_Decimation_in_im            :   IN    std_logic_vector(15
          DOWNTO 0);  -- ufix16
        FIR_Decimation_out_re           :   OUT   std_logic_vector(15
          DOWNTO 0);  -- sfix16_En14
        FIR_Decimation_out_im           :   OUT   std_logic_vector(15
          DOWNTO 0)  -- sfix16_En14
        );
END COMPONENT;

COMPONENT BPSK_Demodulator_Baseband
  PORT( in0_re                          :   IN    std_logic_vector(15
    DOWNTO 0);  -- ufix16
        in0_im                          :   IN    std_logic_vector(15
          DOWNTO 0);  -- ufix16
        out0                            :   OUT   std_logic  -- ufix1
        );
END COMPONENT;

-- Component Configuration Statements
FOR ALL : CommSystem_tc
  USE ENTITY work.CommSystem_tc(rtl);
```

```
FOR ALL : BPSK_Modulator_Baseband
  USE ENTITY work.BPSK_Modulator_Baseband(rtl);

FOR ALL : FIR_Interpolation
  USE ENTITY work.FIR_Interpolation(rtl);

FOR ALL : FIR_Decimation
  USE ENTITY work.FIR_Decimation(rtl);

FOR ALL : BPSK_Demodulator_Baseband
  USE ENTITY work.BPSK_Demodulator_Baseband(rtl);

-- Signals
SIGNAL enb_1_4_0                          : std_logic;
SIGNAL enb_1_1_1                          : std_logic;
SIGNAL enb_1_4_1                          : std_logic;
SIGNAL enb                                : std_logic;
SIGNAL PN_Sequence_Generator_out1         : std_logic;  -- ufix1
SIGNAL BPSK_Modulator_Baseband_out1_re    : std_logic_vector(1 DOWNTO 0)
    ;  -- ufix2
SIGNAL BPSK_Modulator_Baseband_out1_im    : std_logic_vector(1 DOWNTO 0)
    ;  -- ufix2
SIGNAL BPSK_Modulator_Baseband_out1_re_signed : signed(1 DOWNTO 0);
   -- sfix2
SIGNAL BPSK_Modulator_Baseband_out1_im_signed : signed(1 DOWNTO 0);
   -- sfix2
SIGNAL BPSK_Modulator_Baseband_out1_re_1 : signed(1 DOWNTO 0);  --
    sfix2
SIGNAL BPSK_Modulator_Baseband_out1_im_1 : signed(1 DOWNTO 0);  --
    sfix2
SIGNAL BPSK_Modulator_Baseband_out1_re_2 : std_logic_vector(1 DOWNTO
    0);  -- ufix2
SIGNAL BPSK_Modulator_Baseband_out1_im_2 : std_logic_vector(1 DOWNTO
```

```
    0);  -- ufix2
SIGNAL FIR_Interpolation_out1_re         : std_logic_vector(15 DOWNTO
    0);  -- ufix16
SIGNAL FIR_Interpolation_out1_im         : std_logic_vector(15 DOWNTO
    0);  -- ufix16
SIGNAL FIR_Interpolation_out1_re_signed : signed(15 DOWNTO 0);  --
    sfix16_En15
SIGNAL FIR_Interpolation_out1_im_signed : signed(15 DOWNTO 0);  --
    sfix16_En15
SIGNAL FIR_Interpolation_out1_re_1       : signed(15 DOWNTO 0);  --
    sfix16_En15
SIGNAL FIR_Interpolation_out1_im_1       : signed(15 DOWNTO 0);  --
    sfix16_En15
SIGNAL delayMatch_reg_re                 : vector_of_signed16(0 TO 1);
     -- sfix16_En15 [2]
SIGNAL delayMatch_reg_im                 : vector_of_signed16(0 TO 1);
     -- sfix16_En15 [2]
SIGNAL FIR_Interpolation_out1_re_2       : signed(15 DOWNTO 0);  --
    sfix16_En15
SIGNAL FIR_Interpolation_out1_im_2       : signed(15 DOWNTO 0);  --
    sfix16_En15
SIGNAL FIR_Interpolation_out1_re_3       : std_logic_vector(15 DOWNTO
    0);  -- ufix16
SIGNAL FIR_Interpolation_out1_im_3       : std_logic_vector(15 DOWNTO
    0);  -- ufix16
SIGNAL FIR_Decimation_out1_re            : std_logic_vector(15 DOWNTO
    0);  -- ufix16
SIGNAL FIR_Decimation_out1_im            : std_logic_vector(15 DOWNTO
    0);  -- ufix16
SIGNAL FIR_Decimation_out1_re_signed     : signed(15 DOWNTO 0);  --
    sfix16_En14
SIGNAL FIR_Decimation_out1_im_signed     : signed(15 DOWNTO 0);  --
    sfix16_En14
```

```
  SIGNAL FIR_Decimation_out1_re_1          : signed(15 DOWNTO 0);  --
      sfix16_En14
  SIGNAL FIR_Decimation_out1_im_1          : signed(15 DOWNTO 0);  --
      sfix16_En14
  SIGNAL BPSK_Demodulator_Baseband_out1    : std_logic;  -- ufix1
  SIGNAL BPSK_Demodulator_Baseband_out1_1  : std_logic;  -- ufix1
  SIGNAL pn_reg                            : unsigned(5 DOWNTO 0);  --
      ufix6
  SIGNAL pn_out                            : std_logic;
  SIGNAL pn_xorout                         : std_logic;
  SIGNAL pn_newvalue                       : vector_of_unsigned6(0 TO 1);
      -- ufix6 [2]
  SIGNAL pn_value_shifted                  : unsigned(4 DOWNTO 0);  --
      ufix5_E1

BEGIN
  u_CommSystem_tc : CommSystem_tc
    PORT MAP( clk => clk,
              reset => reset,
              clk_enable => clk_enable,
              enb => enb,
              enb_1_1_1 => enb_1_1_1,
              enb_1_4_0 => enb_1_4_0,
              enb_1_4_1 => enb_1_4_1
              );

  -- <Root>/BPSK Modulator Baseband
  u_BPSK_Modulator_Baseband : BPSK_Modulator_Baseband
    PORT MAP( in0 => PN_Sequence_Generator_out1,  -- ufix1
              out0_re => BPSK_Modulator_Baseband_out1_re,  -- ufix2
              out0_im => BPSK_Modulator_Baseband_out1_im  -- ufix2
              );
```

```
-- <Root>/FIR Interpolation
u_FIR_Interpolation : FIR_Interpolation
  PORT MAP( clk => clk,
            enb_1_1_1 => enb_1_1_1,
            reset => reset,
            FIR_Interpolation_in_re =>
                BPSK_Modulator_Baseband_out1_re_2,  -- ufix2
            FIR_Interpolation_in_im =>
                BPSK_Modulator_Baseband_out1_im_2,  -- ufix2
            FIR_Interpolation_out_re => FIR_Interpolation_out1_re,  --
                sfix16_En15
            FIR_Interpolation_out_im => FIR_Interpolation_out1_im  --
                sfix16_En15
            );

-- <Root>/FIR Decimation
u_FIR_Decimation : FIR_Decimation
  PORT MAP( clk => clk,
            enb_1_1_1 => enb_1_1_1,
            reset => reset,
            FIR_Decimation_in_re => FIR_Interpolation_out1_re_3,  --
                ufix16
            FIR_Decimation_in_im => FIR_Interpolation_out1_im_3,  --
                ufix16
            FIR_Decimation_out_re => FIR_Decimation_out1_re,  --
                sfix16_En14
            FIR_Decimation_out_im => FIR_Decimation_out1_im  --
                sfix16_En14
            );

-- <Root>/BPSK Demodulator Baseband
u_BPSK_Demodulator_Baseband : BPSK_Demodulator_Baseband
  PORT MAP( in0_re => std_logic_vector(FIR_Decimation_out1_re_1),  --
```

```
      ufix16
            in0_im => std_logic_vector(FIR_Decimation_out1_im_1), --
               ufix16
            out0 => BPSK_Demodulator_Baseband_out1  -- ufix1
            );

-- <Root>/PN Sequence Generator
pn_newvalue(0) <= pn_reg;

pn_xorout <= pn_newvalue(0)(0) XOR pn_newvalue(0)(1);

pn_value_shifted <= pn_newvalue(0)(5 DOWNTO 1);

pn_newvalue(1) <= pn_xorout & pn_value_shifted;

pn_out <= pn_newvalue(0)(0);

PN_generation_temp_process3 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    pn_reg <= to_unsigned(1, 6);
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_4_0 = '1' THEN
      pn_reg <= pn_newvalue(1);
    END IF;
  END IF;
END PROCESS PN_generation_temp_process3;

PN_Sequence_Generator_out1 <= pn_out;


BPSK_Modulator_Baseband_out1_re_signed <= signed(
   BPSK_Modulator_Baseband_out1_re);
```

```
BPSK_Modulator_Baseband_out1_im_signed <= signed(
    BPSK_Modulator_Baseband_out1_im);

FIR_Interpolation_in_pipe_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    BPSK_Modulator_Baseband_out1_re_1 <= to_signed(2#00#, 2);
    BPSK_Modulator_Baseband_out1_im_1 <= to_signed(2#00#, 2);
  ELSIF clk'EVENT AND clk = '1' THEN
    IF enb_1_4_0 = '1' THEN
      BPSK_Modulator_Baseband_out1_re_1 <=
          BPSK_Modulator_Baseband_out1_re_signed;
      BPSK_Modulator_Baseband_out1_im_1 <=
          BPSK_Modulator_Baseband_out1_im_signed;
    END IF;
  END IF;
END PROCESS FIR_Interpolation_in_pipe_process;


BPSK_Modulator_Baseband_out1_re_2 <= std_logic_vector(
    BPSK_Modulator_Baseband_out1_re_1);

BPSK_Modulator_Baseband_out1_im_2 <= std_logic_vector(
    BPSK_Modulator_Baseband_out1_im_1);

FIR_Interpolation_out1_re_signed <= signed(FIR_Interpolation_out1_re);

FIR_Interpolation_out1_im_signed <= signed(FIR_Interpolation_out1_im);

FIR_Decimation_in_pipe_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
```

```
      FIR_Interpolation_out1_re_1 <= to_signed(16#0000#, 16);
      FIR_Interpolation_out1_im_1 <= to_signed(16#0000#, 16);
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
        FIR_Interpolation_out1_re_1 <= FIR_Interpolation_out1_re_signed;
        FIR_Interpolation_out1_im_1 <= FIR_Interpolation_out1_im_signed;
      END IF;
    END IF;
  END PROCESS FIR_Decimation_in_pipe_process;


  delayMatch_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      delayMatch_reg_re <= (OTHERS => to_signed(16#0000#, 16));
      delayMatch_reg_im <= (OTHERS => to_signed(16#0000#, 16));
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
        delayMatch_reg_im(0) <= FIR_Interpolation_out1_im_1;
        delayMatch_reg_im(1) <= delayMatch_reg_im(0);
        delayMatch_reg_re(0) <= FIR_Interpolation_out1_re_1;
        delayMatch_reg_re(1) <= delayMatch_reg_re(0);
      END IF;
    END IF;
  END PROCESS delayMatch_process;

  FIR_Interpolation_out1_re_2 <= delayMatch_reg_re(1);
  FIR_Interpolation_out1_im_2 <= delayMatch_reg_im(1);

  FIR_Interpolation_out1_re_3 <= std_logic_vector(
     FIR_Interpolation_out1_re_2);

  FIR_Interpolation_out1_im_3 <= std_logic_vector(
```

```
    FIR_Interpolation_out1_im_2);

FIR_Decimation_out1_re_signed <= signed(FIR_Decimation_out1_re);

FIR_Decimation_out1_im_signed <= signed(FIR_Decimation_out1_im);

BPSK_Demodulator_Baseband_in_pipe_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    FIR_Decimation_out1_re_1 <= to_signed(16#0000#, 16);
    FIR_Decimation_out1_im_1 <= to_signed(16#0000#, 16);
  ELSIF clk'EVENT AND clk = '1' THEN
    IF enb_1_4_0 = '1' THEN
      FIR_Decimation_out1_re_1 <= FIR_Decimation_out1_re_signed;
      FIR_Decimation_out1_im_1 <= FIR_Decimation_out1_im_signed;
    END IF;
  END IF;
END PROCESS BPSK_Demodulator_Baseband_in_pipe_process;


BPSK_Demodulator_Baseband_out_pipe_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    BPSK_Demodulator_Baseband_out1_1 <= '0';
  ELSIF clk'EVENT AND clk = '1' THEN
    IF enb_1_4_0 = '1' THEN
      BPSK_Demodulator_Baseband_out1_1 <=
          BPSK_Demodulator_Baseband_out1;
    END IF;
  END IF;
END PROCESS BPSK_Demodulator_Baseband_out_pipe_process;
```

```
    ce_out <= enb_1_4_1;

    To_DataSink <= BPSK_Demodulator_Baseband_out1_1;

END rtl;
```

A.1.2 Package

```
-- -------------------------------------------------------------
--
-- File Name: CommSystem_pkg.vhd
-- Created: 2014-03-16 01:42:40
--
-- Generated by MATLAB 8.2 and HDL Coder 3.3
--
-- -------------------------------------------------------------


LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

PACKAGE CommSystem_pkg IS
  TYPE vector_of_signed16 IS ARRAY (NATURAL RANGE <>) OF signed(15
      DOWNTO 0);
  TYPE vector_of_unsigned6 IS ARRAY (NATURAL RANGE <>) OF unsigned(5
      DOWNTO 0);
END CommSystem_pkg;
```

A.1.3 Timing Controller

```
-- -------------------------------------------------------------
--
-- File Name: CommSystem_tc.vhd
-- Created: 2014-03-16 01:42:10
--
-- Generated by MATLAB 8.2 and HDL Coder 3.3
--
-- -------------------------------------------------------------



-- -------------------------------------------------------------
--
-- Module: CommSystem_tc
-- Source Path: CommSystem_tc
-- Hierarchy Level: 1
--
-- Master clock enable input: clk_enable
--
-- enb         : identical to clk_enable
-- enb_1_1_1   : identical to clk_enable
-- enb_1_4_0   : 4x slower than clk_enable with last phase
-- enb_1_4_1   : 4x slower than clk_enable with phase 1
--
-- -------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY CommSystem_tc IS
  PORT( clk                               :   IN    std_logic;
        reset                             :   IN    std_logic;
        clk_enable                        :   IN    std_logic;
```

```
        enb                                 :   OUT   std_logic;
        enb_1_1_1                           :   OUT   std_logic;
        enb_1_4_0                           :   OUT   std_logic;
        enb_1_4_1                           :   OUT   std_logic
        );
END CommSystem_tc;


ARCHITECTURE rtl OF CommSystem_tc IS

  -- Signals
  SIGNAL count4                             : unsigned(1 DOWNTO 0); --
      ufix2
  SIGNAL phase_all                          : std_logic;
  SIGNAL phase_0                            : std_logic;
  SIGNAL phase_0_tmp                        : std_logic;
  SIGNAL phase_1                            : std_logic;
  SIGNAL phase_1_tmp                        : std_logic;

BEGIN
  Counter4 : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      count4 <= to_unsigned(1, 2);
    ELSIF clk'event AND clk = '1' THEN
      IF clk_enable = '1' THEN
        IF count4 = to_unsigned(3, 2) THEN
          count4 <= to_unsigned(0, 2);
        ELSE
          count4 <= count4 + 1;
        END IF;
      END IF;
    END IF;
```

```
END PROCESS Counter4;

phase_all <= '1' WHEN clk_enable = '1' ELSE '0';

temp_process1 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    phase_0 <= '0';
  ELSIF clk'event AND clk = '1' THEN
    IF clk_enable = '1' THEN
      phase_0 <= phase_0_tmp;
    END IF;
  END IF;
END PROCESS temp_process1;

phase_0_tmp <= '1' WHEN count4 = to_unsigned(3, 2) AND clk_enable =
    '1' ELSE '0';

temp_process2 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    phase_1 <= '1';
  ELSIF clk'event AND clk = '1' THEN
    IF clk_enable = '1' THEN
      phase_1 <= phase_1_tmp;
    END IF;
  END IF;
END PROCESS temp_process2;

phase_1_tmp <= '1' WHEN count4 = to_unsigned(0, 2) AND clk_enable =
    '1' ELSE '0';

enb <=  phase_all AND clk_enable;
```

```
  enb_1_1_1 <=  phase_all AND clk_enable;

  enb_1_4_0 <=  phase_0 AND clk_enable;

  enb_1_4_1 <=  phase_1 AND clk_enable;


END rtl;
```

A.1.4 BPSK Modulator

```
-- -------------------------------------------------------------
--
-- File Name: BPSK_Modulator_Baseband.vhd
-- Created: 2014-03-16 01:42:09
--
-- Generated by MATLAB 8.2 and HDL Coder 3.3
--
-- -------------------------------------------------------------



-- -------------------------------------------------------------
--
-- Module: BPSK_Modulator_Baseband
-- Source Path: CommSystem/BPSK Modulator Baseband
-- Hierarchy Level: 1
--
-- -------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY BPSK_Modulator_Baseband IS
  PORT( in0                               :   IN    std_logic;  -- ufix1
        out0_re                           :   OUT   std_logic_vector(1
          DOWNTO 0);  -- sfix2
        out0_im                           :   OUT   std_logic_vector(1
          DOWNTO 0)  -- sfix2
        );
END BPSK_Modulator_Baseband;


ARCHITECTURE rtl OF BPSK_Modulator_Baseband IS
```

```
  -- Signals
  SIGNAL inphase                          : signed(1 DOWNTO 0);  --
      sfix2
  SIGNAL quadrature                       : signed(1 DOWNTO 0);  --
      sfix2

BEGIN

  inphase <= to_signed(2#01#, 2) WHEN in0 = '0' ELSE
       to_signed(2#11#, 2);

  out0_re <= std_logic_vector(inphase);

  quadrature <= to_signed(2#00#, 2);

  out0_im <= std_logic_vector(quadrature);

END rtl;
```

A.1.5 FIR Interpolation Filter

```
-- -------------------------------------------------------------
--
-- File Name: FIR_Interpolation
-- Created: 2014-03-16 01:42:10
-- Generated by MATLAB 8.2 and HDL Coder 3.3
--
-- -------------------------------------------------------------
--
--
-- -------------------------------------------------------------
--
-- Module: FIR_Interpolation
-- Source Path: /FIR_Interpolation
--
-- -------------------------------------------------------------
--
-- HDL Implementation    : Fully parallel
-- Multipliers           : 20
-- Folding Factor        : 1


LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.ALL;

ENTITY FIR_Interpolation IS
  PORT( clk                             :   IN    std_logic;
        enb_1_1_1                       :   IN    std_logic;
        reset                           :   IN    std_logic;
        FIR_Interpolation_in_re         :   IN    std_logic_vector(1
          DOWNTO 0); -- sfix2
        FIR_Interpolation_in_im         :   IN    std_logic_vector(1
```

```
            DOWNTO 0); -- sfix2
        FIR_Interpolation_out_re        :   OUT   std_logic_vector(15
            DOWNTO 0); -- sfix16_En15
        FIR_Interpolation_out_im        :   OUT   std_logic_vector(15
            DOWNTO 0)  -- sfix16_En15
        );

END FIR_Interpolation;


-------------------------------------------------------------------
--Module Architecture: FIR_Interpolation
-------------------------------------------------------------------
ARCHITECTURE rtl OF FIR_Interpolation IS
  -- Local Functions
  -- Type Definitions
  TYPE delay_pipeline_type IS ARRAY (NATURAL range <>) OF signed(1
      DOWNTO 0); -- sfix2
  TYPE sumdelay_pipeline_type IS ARRAY (NATURAL range <>) OF signed(16
      DOWNTO 0); -- sfix17_En15
  -- Constants
  CONSTANT coeffphase1_1                : signed(15 DOWNTO 0) :=
      to_signed(5, 16); -- sfix16_En15
  CONSTANT coeffphase1_2                : signed(15 DOWNTO 0) :=
      to_signed(41, 16); -- sfix16_En15
  CONSTANT coeffphase1_3                : signed(15 DOWNTO 0) :=
      to_signed(-83, 16); -- sfix16_En15
  CONSTANT coeffphase1_4                : signed(15 DOWNTO 0) :=
      to_signed(88, 16); -- sfix16_En15
  CONSTANT coeffphase1_5                : signed(15 DOWNTO 0) :=
      to_signed(-25, 16); -- sfix16_En15
  CONSTANT coeffphase1_6                : signed(15 DOWNTO 0) :=
      to_signed(-123, 16); -- sfix16_En15
```

```
CONSTANT coeffphase1_7                  : signed(15 DOWNTO 0) :=
    to_signed(348, 16); -- sfix16_En15
CONSTANT coeffphase1_8                  : signed(15 DOWNTO 0) :=
    to_signed(-615, 16); -- sfix16_En15
CONSTANT coeffphase1_9                  : signed(15 DOWNTO 0) :=
    to_signed(869, 16); -- sfix16_En15
CONSTANT coeffphase1_10                 : signed(15 DOWNTO 0) :=
    to_signed(-1052, 16); -- sfix16_En15
CONSTANT coeffphase1_11                 : signed(15 DOWNTO 0) :=
    to_signed(17504, 16); -- sfix16_En15
CONSTANT coeffphase1_12                 : signed(15 DOWNTO 0) :=
    to_signed(-1052, 16); -- sfix16_En15
CONSTANT coeffphase1_13                 : signed(15 DOWNTO 0) :=
    to_signed(869, 16); -- sfix16_En15
CONSTANT coeffphase1_14                 : signed(15 DOWNTO 0) :=
    to_signed(-615, 16); -- sfix16_En15
CONSTANT coeffphase1_15                 : signed(15 DOWNTO 0) :=
    to_signed(348, 16); -- sfix16_En15
CONSTANT coeffphase1_16                 : signed(15 DOWNTO 0) :=
    to_signed(-123, 16); -- sfix16_En15
CONSTANT coeffphase1_17                 : signed(15 DOWNTO 0) :=
    to_signed(-25, 16); -- sfix16_En15
CONSTANT coeffphase1_18                 : signed(15 DOWNTO 0) :=
    to_signed(88, 16); -- sfix16_En15
CONSTANT coeffphase1_19                 : signed(15 DOWNTO 0) :=
    to_signed(-83, 16); -- sfix16_En15
CONSTANT coeffphase1_20                 : signed(15 DOWNTO 0) :=
    to_signed(41, 16); -- sfix16_En15
CONSTANT coeffphase1_21                 : signed(15 DOWNTO 0) :=
    to_signed(5, 16); -- sfix16_En15
CONSTANT coeffphase2_1                  : signed(15 DOWNTO 0) :=
    to_signed(-41, 16); -- sfix16_En15
CONSTANT coeffphase2_2                  : signed(15 DOWNTO 0) :=
```

```
    to_signed(60, 16); -- sfix16_En15
CONSTANT coeffphase2_3                  : signed(15 DOWNTO 0) :=
    to_signed(-43, 16); -- sfix16_En15
CONSTANT coeffphase2_4                  : signed(15 DOWNTO 0) :=
    to_signed(-19, 16); -- sfix16_En15
CONSTANT coeffphase2_5                  : signed(15 DOWNTO 0) :=
    to_signed(112, 16); -- sfix16_En15
CONSTANT coeffphase2_6                  : signed(15 DOWNTO 0) :=
    to_signed(-187, 16); -- sfix16_En15
CONSTANT coeffphase2_7                  : signed(15 DOWNTO 0) :=
    to_signed(163, 16); -- sfix16_En15
CONSTANT coeffphase2_8                  : signed(15 DOWNTO 0) :=
    to_signed(99, 16); -- sfix16_En15
CONSTANT coeffphase2_9                  : signed(15 DOWNTO 0) :=
    to_signed(-901, 16); -- sfix16_En15
CONSTANT coeffphase2_10                 : signed(15 DOWNTO 0) :=
    to_signed(3897, 16); -- sfix16_En15
CONSTANT coeffphase2_11                 : signed(15 DOWNTO 0) :=
    to_signed(15453, 16); -- sfix16_En15
CONSTANT coeffphase2_12                 : signed(15 DOWNTO 0) :=
    to_signed(-3256, 16); -- sfix16_En15
CONSTANT coeffphase2_13                 : signed(15 DOWNTO 0) :=
    to_signed(1542, 16); -- sfix16_En15
CONSTANT coeffphase2_14                 : signed(15 DOWNTO 0) :=
    to_signed(-699, 16); -- sfix16_En15
CONSTANT coeffphase2_15                 : signed(15 DOWNTO 0) :=
    to_signed(210, 16); -- sfix16_En15
CONSTANT coeffphase2_16                 : signed(15 DOWNTO 0) :=
    to_signed(46, 16); -- sfix16_En15
CONSTANT coeffphase2_17                 : signed(15 DOWNTO 0) :=
    to_signed(-132, 16); -- sfix16_En15
CONSTANT coeffphase2_18                 : signed(15 DOWNTO 0) :=
    to_signed(113, 16); -- sfix16_En15
```

```
CONSTANT coeffphase2_19                 : signed(15 DOWNTO 0) :=
    to_signed(-48, 16); -- sfix16_En15
CONSTANT coeffphase2_20                 : signed(15 DOWNTO 0) :=
    to_signed(-13, 16); -- sfix16_En15
CONSTANT coeffphase2_21                 : signed(15 DOWNTO 0) :=
    to_signed(0, 16); -- sfix16_En15
CONSTANT coeffphase3_1                  : signed(15 DOWNTO 0) :=
    to_signed(-52, 16); -- sfix16_En15
CONSTANT coeffphase3_2                  : signed(15 DOWNTO 0) :=
    to_signed(20, 16); -- sfix16_En15
CONSTANT coeffphase3_3                  : signed(15 DOWNTO 0) :=
    to_signed(48, 16); -- sfix16_En15
CONSTANT coeffphase3_4                  : signed(15 DOWNTO 0) :=
    to_signed(-124, 16); -- sfix16_En15
CONSTANT coeffphase3_5                  : signed(15 DOWNTO 0) :=
    to_signed(152, 16); -- sfix16_En15
CONSTANT coeffphase3_6                  : signed(15 DOWNTO 0) :=
    to_signed(-48, 16); -- sfix16_En15
CONSTANT coeffphase3_7                  : signed(15 DOWNTO 0) :=
    to_signed(-300, 16); -- sfix16_En15
CONSTANT coeffphase3_8                  : signed(15 DOWNTO 0) :=
    to_signed(1070, 16); -- sfix16_En15
CONSTANT coeffphase3_9                  : signed(15 DOWNTO 0) :=
    to_signed(-2790, 16); -- sfix16_En15
CONSTANT coeffphase3_10                 : signed(15 DOWNTO 0) :=
    to_signed(10188, 16); -- sfix16_En15
CONSTANT coeffphase3_11                 : signed(15 DOWNTO 0) :=
    to_signed(10188, 16); -- sfix16_En15
CONSTANT coeffphase3_12                 : signed(15 DOWNTO 0) :=
    to_signed(-2790, 16); -- sfix16_En15
CONSTANT coeffphase3_13                 : signed(15 DOWNTO 0) :=
    to_signed(1070, 16); -- sfix16_En15
CONSTANT coeffphase3_14                 : signed(15 DOWNTO 0) :=
```

```
    to_signed(-300, 16); -- sfix16_En15
CONSTANT coeffphase3_15                 : signed(15 DOWNTO 0) :=
    to_signed(-48, 16); -- sfix16_En15
CONSTANT coeffphase3_16                 : signed(15 DOWNTO 0) :=
    to_signed(152, 16); -- sfix16_En15
CONSTANT coeffphase3_17                 : signed(15 DOWNTO 0) :=
    to_signed(-124, 16); -- sfix16_En15
CONSTANT coeffphase3_18                 : signed(15 DOWNTO 0) :=
    to_signed(48, 16); -- sfix16_En15
CONSTANT coeffphase3_19                 : signed(15 DOWNTO 0) :=
    to_signed(20, 16); -- sfix16_En15
CONSTANT coeffphase3_20                 : signed(15 DOWNTO 0) :=
    to_signed(-52, 16); -- sfix16_En15
CONSTANT coeffphase3_21                 : signed(15 DOWNTO 0) :=
    to_signed(0, 16); -- sfix16_En15
CONSTANT coeffphase4_1                  : signed(15 DOWNTO 0) :=
    to_signed(-13, 16); -- sfix16_En15
CONSTANT coeffphase4_2                  : signed(15 DOWNTO 0) :=
    to_signed(-48, 16); -- sfix16_En15
CONSTANT coeffphase4_3                  : signed(15 DOWNTO 0) :=
    to_signed(113, 16); -- sfix16_En15
CONSTANT coeffphase4_4                  : signed(15 DOWNTO 0) :=
    to_signed(-132, 16); -- sfix16_En15
CONSTANT coeffphase4_5                  : signed(15 DOWNTO 0) :=
    to_signed(46, 16); -- sfix16_En15
CONSTANT coeffphase4_6                  : signed(15 DOWNTO 0) :=
    to_signed(210, 16); -- sfix16_En15
CONSTANT coeffphase4_7                  : signed(15 DOWNTO 0) :=
    to_signed(-699, 16); -- sfix16_En15
CONSTANT coeffphase4_8                  : signed(15 DOWNTO 0) :=
    to_signed(1542, 16); -- sfix16_En15
CONSTANT coeffphase4_9                  : signed(15 DOWNTO 0) :=
    to_signed(-3256, 16); -- sfix16_En15
```

```
CONSTANT coeffphase4_10                 : signed(15 DOWNTO 0) :=
    to_signed(15453, 16); -- sfix16_En15
CONSTANT coeffphase4_11                 : signed(15 DOWNTO 0) :=
    to_signed(3897, 16); -- sfix16_En15
CONSTANT coeffphase4_12                 : signed(15 DOWNTO 0) :=
    to_signed(-901, 16); -- sfix16_En15
CONSTANT coeffphase4_13                 : signed(15 DOWNTO 0) :=
    to_signed(99, 16); -- sfix16_En15
CONSTANT coeffphase4_14                 : signed(15 DOWNTO 0) :=
    to_signed(163, 16); -- sfix16_En15
CONSTANT coeffphase4_15                 : signed(15 DOWNTO 0) :=
    to_signed(-187, 16); -- sfix16_En15
CONSTANT coeffphase4_16                 : signed(15 DOWNTO 0) :=
    to_signed(112, 16); -- sfix16_En15
CONSTANT coeffphase4_17                 : signed(15 DOWNTO 0) :=
    to_signed(-19, 16); -- sfix16_En15
CONSTANT coeffphase4_18                 : signed(15 DOWNTO 0) :=
    to_signed(-43, 16); -- sfix16_En15
CONSTANT coeffphase4_19                 : signed(15 DOWNTO 0) :=
    to_signed(60, 16); -- sfix16_En15
CONSTANT coeffphase4_20                 : signed(15 DOWNTO 0) :=
    to_signed(-41, 16); -- sfix16_En15
CONSTANT coeffphase4_21                 : signed(15 DOWNTO 0) :=
    to_signed(0, 16); -- sfix16_En15

-- Signals
SIGNAL cur_count                        : unsigned(1 DOWNTO 0); --
    ufix2
SIGNAL phase_3                          : std_logic; -- boolean
SIGNAL delay_pipeline_re                : delay_pipeline_type(0 TO 19)
    ; -- sfix2
SIGNAL delay_pipeline_im                : delay_pipeline_type(0 TO 19)
    ; -- sfix2
```

```
SIGNAL FIR_Interpolation_in_regtype_re  : signed(1 DOWNTO 0); -- sfix2
SIGNAL FIR_Interpolation_in_regtype_im  : signed(1 DOWNTO 0); -- sfix2
SIGNAL product_re                       : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_im                       : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_mux                      : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL product_re_pipe_re               : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_re_pipe_im               : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_1_re                     : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_1_im                     : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_mux_1                    : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL product_1_re_pipe_re             : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_1_re_pipe_im             : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_2_re                     : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_2_im                     : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_mux_2                    : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL product_2_re_pipe_re             : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_2_re_pipe_im             : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_3_re                     : signed(16 DOWNTO 0); --
```

```
    sfix17_En15
SIGNAL product_3_im                 : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_mux_3                : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL product_3_re_pipe_re         : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_3_re_pipe_im         : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_4_re                 : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_4_im                 : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_mux_4                : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL product_4_re_pipe_re         : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_4_re_pipe_im         : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_5_re                 : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_5_im                 : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_mux_5                : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL product_5_re_pipe_re         : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_5_re_pipe_im         : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_6_re                 : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_6_im                 : signed(16 DOWNTO 0); --
    sfix17_En15
```

```
SIGNAL product_mux_6                    : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL product_6_re_pipe_re             : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_6_re_pipe_im             : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_7_re                     : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_7_im                     : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_mux_7                    : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL product_7_re_pipe_re             : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_7_re_pipe_im             : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_8_re                     : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_8_im                     : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_mux_8                    : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL product_8_re_pipe_re             : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_8_re_pipe_im             : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_9_re                     : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_9_im                     : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_mux_9                    : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL product_9_re_pipe_re             : signed(17 DOWNTO 0); --
```

```
    sfix18_En15
SIGNAL product_9_re_pipe_im             : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_10_re                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_10_im                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_mux_10                   : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL product_10_re_pipe_re            : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_10_re_pipe_im            : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_11_re                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_11_im                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_mux_11                   : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL product_11_re_pipe_re            : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_11_re_pipe_im            : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_12_re                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_12_im                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_mux_12                   : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL product_12_re_pipe_re            : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_12_re_pipe_im            : signed(17 DOWNTO 0); --
    sfix18_En15
```

```
SIGNAL product_13_re                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_13_im                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_mux_13                   : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL product_13_re_pipe_re            : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_13_re_pipe_im            : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_14_re                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_14_im                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_mux_14                   : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL product_14_re_pipe_re            : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_14_re_pipe_im            : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_15_re                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_15_im                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_mux_15                   : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL product_15_re_pipe_re            : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_15_re_pipe_im            : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_16_re                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_16_im                    : signed(16 DOWNTO 0); --
```

```
    sfix17_En15
SIGNAL product_mux_16                : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL product_16_re_pipe_re         : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_16_re_pipe_im         : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_17_re                 : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_17_im                 : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_mux_17                : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL product_17_re_pipe_re         : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_17_re_pipe_im         : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_18_re                 : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_18_im                 : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_mux_18                : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL product_18_re_pipe_re         : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_18_re_pipe_im         : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_19_re                 : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_19_im                 : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_mux_19                : signed(15 DOWNTO 0); --
    sfix16_En15
```

```
SIGNAL product_19_re_pipe_re          : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_19_re_pipe_im          : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_20_re                  : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_20_im                  : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL product_mux_20                 : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL product_20_re_pipe_re          : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL product_20_re_pipe_im          : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL sumvector1_re                  : sumdelay_pipeline_type(0 TO
    10); -- sfix17_En15
SIGNAL sumvector1_im                  : sumdelay_pipeline_type(0 TO
    10); -- sfix17_En15
SIGNAL add_cast                       : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_1                     : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp                       : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_2                     : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_3                     : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_1                     : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_4                     : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_5                     : signed(16 DOWNTO 0); --
```

```
    sfix17_En15
SIGNAL add_temp_2                       : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_6                       : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_7                       : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_3                       : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_8                       : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_9                       : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_4                       : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_10                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_11                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_5                       : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_12                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_13                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_6                       : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_14                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_15                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_7                       : signed(17 DOWNTO 0); --
    sfix18_En15
```

```
SIGNAL add_cast_16                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_17                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_8                       : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_18                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_19                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_9                       : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_20                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_21                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_10                      : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_22                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_23                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_11                      : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_24                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_25                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_12                      : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_26                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_27                      : signed(16 DOWNTO 0); --
```

```
    sfix17_En15
SIGNAL add_temp_13                    : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_28                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_29                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_14                    : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_30                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_31                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_15                    : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_32                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_33                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_16                    : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_34                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_35                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_17                    : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_36                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_37                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_18                    : signed(17 DOWNTO 0); --
    sfix18_En15
```

```
SIGNAL add_cast_38                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_39                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_19                      : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL sumdelay_pipeline1_re            : sumdelay_pipeline_type(0 TO
    10); -- sfix17_En15
SIGNAL sumdelay_pipeline1_im            : sumdelay_pipeline_type(0 TO
    10); -- sfix17_En15
SIGNAL sumvector2_re                    : sumdelay_pipeline_type(0 TO
    5); -- sfix17_En15
SIGNAL sumvector2_im                    : sumdelay_pipeline_type(0 TO
    5); -- sfix17_En15
SIGNAL add_cast_40                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_41                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_20                      : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_42                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_43                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_21                      : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_44                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_45                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_22                      : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_46                      : signed(16 DOWNTO 0); --
```

```
    sfix17_En15
SIGNAL add_cast_47                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_23                    : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_48                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_49                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_24                    : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_50                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_51                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_25                    : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_52                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_53                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_26                    : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_54                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_55                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_27                    : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_56                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_57                    : signed(16 DOWNTO 0); --
    sfix17_En15
```

```
SIGNAL add_temp_28                    : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_58                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_59                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_29                    : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL sumdelay_pipeline2_re          : sumdelay_pipeline_type(0 TO
    5); -- sfix17_En15
SIGNAL sumdelay_pipeline2_im          : sumdelay_pipeline_type(0 TO
    5); -- sfix17_En15
SIGNAL sumvector3_re                  : sumdelay_pipeline_type(0 TO
    2); -- sfix17_En15
SIGNAL sumvector3_im                  : sumdelay_pipeline_type(0 TO
    2); -- sfix17_En15
SIGNAL add_cast_60                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_61                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_30                    : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_62                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_63                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_31                    : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_64                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_65                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_32                    : signed(17 DOWNTO 0); --
```

```
    sfix18_En15
SIGNAL add_cast_66                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_67                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_33                    : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_68                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_69                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_34                    : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_70                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_71                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_35                    : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL sumdelay_pipeline3_re          : sumdelay_pipeline_type(0 TO
    2); -- sfix17_En15
SIGNAL sumdelay_pipeline3_im          : sumdelay_pipeline_type(0 TO
    2); -- sfix17_En15
SIGNAL sumvector4_re                  : sumdelay_pipeline_type(0 TO
    1); -- sfix17_En15
SIGNAL sumvector4_im                  : sumdelay_pipeline_type(0 TO
    1); -- sfix17_En15
SIGNAL add_cast_72                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_73                    : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_36                    : signed(17 DOWNTO 0); --
    sfix18_En15
```

```
SIGNAL add_cast_74                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_75                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_37                      : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL sumdelay_pipeline4_re            : sumdelay_pipeline_type(0 TO
    1); -- sfix17_En15
SIGNAL sumdelay_pipeline4_im            : sumdelay_pipeline_type(0 TO
    1); -- sfix17_En15
SIGNAL sum5_re                          : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL sum5_im                          : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_76                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_77                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_38                      : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL add_cast_78                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_cast_79                      : signed(16 DOWNTO 0); --
    sfix17_En15
SIGNAL add_temp_39                      : signed(17 DOWNTO 0); --
    sfix18_En15
SIGNAL output_typeconvert_re            : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL output_typeconvert_im            : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL regout_re                        : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL regout_im                        : signed(15 DOWNTO 0); --
```

```
    sfix16_En15
  SIGNAL muxout_re                        : signed(15 DOWNTO 0); --
    sfix16_En15
  SIGNAL muxout_im                        : signed(15 DOWNTO 0); --
    sfix16_En15


BEGIN

  -- Block Statements
  ce_output : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      cur_count <= to_unsigned(0, 2);
    ELSIF clk'event AND clk = '1' THEN
      IF enb_1_1_1 = '1' THEN
        IF cur_count = to_unsigned(3, 2) THEN
          cur_count <= to_unsigned(0, 2);
        ELSE
          cur_count <= cur_count + 1;
        END IF;
      END IF;
    END IF;
  END PROCESS ce_output;

  phase_3 <= '1' WHEN cur_count = to_unsigned(3, 2) AND enb_1_1_1 = '1'
      ELSE '0';

  --    ----------------- Delay Registers -----------------

  Delay_Pipeline_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
```

```
      delay_pipeline_re(0 TO 19) <= (OTHERS => (OTHERS => '0'));
      delay_pipeline_im(0 TO 19) <= (OTHERS => (OTHERS => '0'));
    ELSIF clk'event AND clk = '1' THEN
      IF phase_3 = '1' THEN
        delay_pipeline_re(0) <= signed(FIR_Interpolation_in_re);
        delay_pipeline_re(1 TO 19) <= delay_pipeline_re(0 TO 18);
        delay_pipeline_im(0) <= signed(FIR_Interpolation_in_im);
        delay_pipeline_im(1 TO 19) <= delay_pipeline_im(0 TO 18);
      END IF;
    END IF;
  END PROCESS Delay_Pipeline_process;

  FIR_Interpolation_in_regtype_re <= signed(FIR_Interpolation_in_re);
  FIR_Interpolation_in_regtype_im <= signed(FIR_Interpolation_in_im);

  product_mux <= coeffphase1_21 WHEN ( cur_count = to_unsigned(0, 2) )
      ELSE
                     coeffphase2_21 WHEN ( cur_count = to_unsigned(1,
                         2) ) ELSE
                     coeffphase3_21 WHEN ( cur_count = to_unsigned(2,
                         2) ) ELSE
                     coeffphase4_21;
  temp_process4 : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      product_re_pipe_re <= (OTHERS => '0');
      product_re_pipe_im <= (OTHERS => '0');
    ELSIF clk'event AND clk = '1' THEN
      IF enb_1_1_1 = '1' THEN

        product_re_pipe_re <= delay_pipeline_re(19) * product_mux;
        product_re_pipe_im <= delay_pipeline_im(19) * product_mux;
```

```
    END IF;
  END IF;
END PROCESS temp_process4;

product_re <= (16 => '0', OTHERS => '1') WHEN product_re_pipe_re(17) =
    '0' AND product_re_pipe_re(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_re_pipe_re(17) = '1'
       AND product_re_pipe_re(16) /= '1'
    ELSE (product_re_pipe_re(16 DOWNTO 0));
product_im <= (16 => '0', OTHERS => '1') WHEN product_re_pipe_im(17) =
    '0' AND product_re_pipe_im(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_re_pipe_im(17) = '1'
       AND product_re_pipe_im(16) /= '1'
    ELSE (product_re_pipe_im(16 DOWNTO 0));

product_mux_1 <= coeffphase1_20 WHEN ( cur_count = to_unsigned(0, 2) )
    ELSE
                     coeffphase2_20 WHEN ( cur_count = to_unsigned(1,
                         2) ) ELSE
                     coeffphase3_20 WHEN ( cur_count = to_unsigned(2,
                         2) ) ELSE
                     coeffphase4_20;
temp_process5 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    product_1_re_pipe_re <= (OTHERS => '0');
    product_1_re_pipe_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN

      product_1_re_pipe_re <= delay_pipeline_re(18) * product_mux_1;
      product_1_re_pipe_im <= delay_pipeline_im(18) * product_mux_1;
```

```
    END IF;
  END IF;
END PROCESS temp_process5;

product_1_re <= (16 => '0', OTHERS => '1') WHEN product_1_re_pipe_re
    (17) = '0' AND product_1_re_pipe_re(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_1_re_pipe_re(17) =
        '1' AND product_1_re_pipe_re(16) /= '1'
    ELSE (product_1_re_pipe_re(16 DOWNTO 0));
product_1_im <= (16 => '0', OTHERS => '1') WHEN product_1_re_pipe_im
    (17) = '0' AND product_1_re_pipe_im(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_1_re_pipe_im(17) =
        '1' AND product_1_re_pipe_im(16) /= '1'
    ELSE (product_1_re_pipe_im(16 DOWNTO 0));

product_mux_2 <= coeffphase1_19 WHEN ( cur_count = to_unsigned(0, 2) )
    ELSE
                     coeffphase2_19 WHEN ( cur_count = to_unsigned(1,
                         2) ) ELSE
                     coeffphase3_19 WHEN ( cur_count = to_unsigned(2,
                         2) ) ELSE
                     coeffphase4_19;
temp_process6 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    product_2_re_pipe_re <= (OTHERS => '0');
    product_2_re_pipe_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN

      product_2_re_pipe_re <= delay_pipeline_re(17) * product_mux_2;
      product_2_re_pipe_im <= delay_pipeline_im(17) * product_mux_2;
```

```
    END IF;
  END IF;
END PROCESS temp_process6;

product_2_re <= (16 => '0', OTHERS => '1') WHEN product_2_re_pipe_re
    (17) = '0' AND product_2_re_pipe_re(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_2_re_pipe_re(17) =
        '1' AND product_2_re_pipe_re(16) /= '1'
    ELSE (product_2_re_pipe_re(16 DOWNTO 0));
product_2_im <= (16 => '0', OTHERS => '1') WHEN product_2_re_pipe_im
    (17) = '0' AND product_2_re_pipe_im(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_2_re_pipe_im(17) =
        '1' AND product_2_re_pipe_im(16) /= '1'
    ELSE (product_2_re_pipe_im(16 DOWNTO 0));

product_mux_3 <= coeffphase1_18 WHEN ( cur_count = to_unsigned(0, 2) )
    ELSE
                     coeffphase2_18 WHEN ( cur_count = to_unsigned(1,
                         2) ) ELSE
                     coeffphase3_18 WHEN ( cur_count = to_unsigned(2,
                         2) ) ELSE
                     coeffphase4_18;
temp_process7 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    product_3_re_pipe_re <= (OTHERS => '0');
    product_3_re_pipe_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN

      product_3_re_pipe_re <= delay_pipeline_re(16) * product_mux_3;
      product_3_re_pipe_im <= delay_pipeline_im(16) * product_mux_3;
```

```
    END IF;
  END IF;
END PROCESS temp_process7;

product_3_re <= (16 => '0', OTHERS => '1') WHEN product_3_re_pipe_re
    (17) = '0' AND product_3_re_pipe_re(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_3_re_pipe_re(17) =
        '1' AND product_3_re_pipe_re(16) /= '1'
    ELSE (product_3_re_pipe_re(16 DOWNTO 0));
product_3_im <= (16 => '0', OTHERS => '1') WHEN product_3_re_pipe_im
    (17) = '0' AND product_3_re_pipe_im(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_3_re_pipe_im(17) =
        '1' AND product_3_re_pipe_im(16) /= '1'
    ELSE (product_3_re_pipe_im(16 DOWNTO 0));

product_mux_4 <= coeffphase1_17 WHEN ( cur_count = to_unsigned(0, 2) )
    ELSE
                     coeffphase2_17 WHEN ( cur_count = to_unsigned(1,
                         2) ) ELSE
                     coeffphase3_17 WHEN ( cur_count = to_unsigned(2,
                         2) ) ELSE
                     coeffphase4_17;
temp_process8 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    product_4_re_pipe_re <= (OTHERS => '0');
    product_4_re_pipe_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN

      product_4_re_pipe_re <= delay_pipeline_re(15) * product_mux_4;
      product_4_re_pipe_im <= delay_pipeline_im(15) * product_mux_4;
```

```
    END IF;
  END IF;
END PROCESS temp_process8;

product_4_re <= (16 => '0', OTHERS => '1') WHEN product_4_re_pipe_re
    (17) = '0' AND product_4_re_pipe_re(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_4_re_pipe_re(17) =
        '1' AND product_4_re_pipe_re(16) /= '1'
    ELSE (product_4_re_pipe_re(16 DOWNTO 0));
product_4_im <= (16 => '0', OTHERS => '1') WHEN product_4_re_pipe_im
    (17) = '0' AND product_4_re_pipe_im(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_4_re_pipe_im(17) =
        '1' AND product_4_re_pipe_im(16) /= '1'
    ELSE (product_4_re_pipe_im(16 DOWNTO 0));

product_mux_5 <= coeffphase1_16 WHEN ( cur_count = to_unsigned(0, 2) )
    ELSE
                     coeffphase2_16 WHEN ( cur_count = to_unsigned(1,
                         2) ) ELSE
                     coeffphase3_16 WHEN ( cur_count = to_unsigned(2,
                         2) ) ELSE
                     coeffphase4_16;
temp_process9 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    product_5_re_pipe_re <= (OTHERS => '0');
    product_5_re_pipe_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN

      product_5_re_pipe_re <= delay_pipeline_re(14) * product_mux_5;
      product_5_re_pipe_im <= delay_pipeline_im(14) * product_mux_5;
```

```
    END IF;
  END IF;
END PROCESS temp_process9;

product_5_re <= (16 => '0', OTHERS => '1') WHEN product_5_re_pipe_re
    (17) = '0' AND product_5_re_pipe_re(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_5_re_pipe_re(17) =
        '1' AND product_5_re_pipe_re(16) /= '1'
    ELSE (product_5_re_pipe_re(16 DOWNTO 0));
product_5_im <= (16 => '0', OTHERS => '1') WHEN product_5_re_pipe_im
    (17) = '0' AND product_5_re_pipe_im(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_5_re_pipe_im(17) =
        '1' AND product_5_re_pipe_im(16) /= '1'
    ELSE (product_5_re_pipe_im(16 DOWNTO 0));

product_mux_6 <= coeffphase1_15 WHEN ( cur_count = to_unsigned(0, 2) )
    ELSE
                     coeffphase2_15 WHEN ( cur_count = to_unsigned(1,
                         2) ) ELSE
                     coeffphase3_15 WHEN ( cur_count = to_unsigned(2,
                         2) ) ELSE
                     coeffphase4_15;
temp_process10 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    product_6_re_pipe_re <= (OTHERS => '0');
    product_6_re_pipe_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN

      product_6_re_pipe_re <= delay_pipeline_re(13) * product_mux_6;
      product_6_re_pipe_im <= delay_pipeline_im(13) * product_mux_6;
```

```
    END IF;
  END IF;
END PROCESS temp_process10;

product_6_re <= (16 => '0', OTHERS => '1') WHEN product_6_re_pipe_re
    (17) = '0' AND product_6_re_pipe_re(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_6_re_pipe_re(17) =
        '1' AND product_6_re_pipe_re(16) /= '1'
    ELSE (product_6_re_pipe_re(16 DOWNTO 0));
product_6_im <= (16 => '0', OTHERS => '1') WHEN product_6_re_pipe_im
    (17) = '0' AND product_6_re_pipe_im(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_6_re_pipe_im(17) =
        '1' AND product_6_re_pipe_im(16) /= '1'
    ELSE (product_6_re_pipe_im(16 DOWNTO 0));

product_mux_7 <= coeffphase1_14 WHEN ( cur_count = to_unsigned(0, 2) )
    ELSE
                     coeffphase2_14 WHEN ( cur_count = to_unsigned(1,
                         2) ) ELSE
                     coeffphase3_14 WHEN ( cur_count = to_unsigned(2,
                         2) ) ELSE
                     coeffphase4_14;
temp_process11 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    product_7_re_pipe_re <= (OTHERS => '0');
    product_7_re_pipe_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN

      product_7_re_pipe_re <= delay_pipeline_re(12) * product_mux_7;
      product_7_re_pipe_im <= delay_pipeline_im(12) * product_mux_7;
```

```
    END IF;
  END IF;
END PROCESS temp_process11;

product_7_re <= (16 => '0', OTHERS => '1') WHEN product_7_re_pipe_re
    (17) = '0' AND product_7_re_pipe_re(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_7_re_pipe_re(17) =
        '1' AND product_7_re_pipe_re(16) /= '1'
    ELSE (product_7_re_pipe_re(16 DOWNTO 0));
product_7_im <= (16 => '0', OTHERS => '1') WHEN product_7_re_pipe_im
    (17) = '0' AND product_7_re_pipe_im(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_7_re_pipe_im(17) =
        '1' AND product_7_re_pipe_im(16) /= '1'
    ELSE (product_7_re_pipe_im(16 DOWNTO 0));

product_mux_8 <= coeffphase1_13 WHEN ( cur_count = to_unsigned(0, 2) )
    ELSE
                     coeffphase2_13 WHEN ( cur_count = to_unsigned(1,
                         2) ) ELSE
                     coeffphase3_13 WHEN ( cur_count = to_unsigned(2,
                         2) ) ELSE
                     coeffphase4_13;
temp_process12 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    product_8_re_pipe_re <= (OTHERS => '0');
    product_8_re_pipe_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN

      product_8_re_pipe_re <= delay_pipeline_re(11) * product_mux_8;
      product_8_re_pipe_im <= delay_pipeline_im(11) * product_mux_8;
```

```
    END IF;
  END IF;
END PROCESS temp_process12;

product_8_re <= (16 => '0', OTHERS => '1') WHEN product_8_re_pipe_re
    (17) = '0' AND product_8_re_pipe_re(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_8_re_pipe_re(17) =
        '1' AND product_8_re_pipe_re(16) /= '1'
    ELSE (product_8_re_pipe_re(16 DOWNTO 0));
product_8_im <= (16 => '0', OTHERS => '1') WHEN product_8_re_pipe_im
    (17) = '0' AND product_8_re_pipe_im(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_8_re_pipe_im(17) =
        '1' AND product_8_re_pipe_im(16) /= '1'
    ELSE (product_8_re_pipe_im(16 DOWNTO 0));

product_mux_9 <= coeffphase1_12 WHEN ( cur_count = to_unsigned(0, 2) )
    ELSE
                     coeffphase2_12 WHEN ( cur_count = to_unsigned(1,
                         2) ) ELSE
                     coeffphase3_12 WHEN ( cur_count = to_unsigned(2,
                         2) ) ELSE
                     coeffphase4_12;
temp_process13 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    product_9_re_pipe_re <= (OTHERS => '0');
    product_9_re_pipe_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN

      product_9_re_pipe_re <= delay_pipeline_re(10) * product_mux_9;
      product_9_re_pipe_im <= delay_pipeline_im(10) * product_mux_9;
```

```
    END IF;
  END IF;
END PROCESS temp_process13;

product_9_re <= (16 => '0', OTHERS => '1') WHEN product_9_re_pipe_re
    (17) = '0' AND product_9_re_pipe_re(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_9_re_pipe_re(17) =
        '1' AND product_9_re_pipe_re(16) /= '1'
    ELSE (product_9_re_pipe_re(16 DOWNTO 0));
product_9_im <= (16 => '0', OTHERS => '1') WHEN product_9_re_pipe_im
    (17) = '0' AND product_9_re_pipe_im(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_9_re_pipe_im(17) =
        '1' AND product_9_re_pipe_im(16) /= '1'
    ELSE (product_9_re_pipe_im(16 DOWNTO 0));

product_mux_10 <= coeffphase1_11 WHEN ( cur_count = to_unsigned(0, 2)
    ) ELSE
                      coeffphase2_11 WHEN ( cur_count = to_unsigned
                          (1, 2) ) ELSE
                      coeffphase3_11 WHEN ( cur_count = to_unsigned
                          (2, 2) ) ELSE
                      coeffphase4_11;
temp_process14 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    product_10_re_pipe_re <= (OTHERS => '0');
    product_10_re_pipe_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN

      product_10_re_pipe_re <= delay_pipeline_re(9) * product_mux_10;
      product_10_re_pipe_im <= delay_pipeline_im(9) * product_mux_10;
```

```
    END IF;
  END IF;
END PROCESS temp_process14;

product_10_re <= (16 => '0', OTHERS => '1') WHEN product_10_re_pipe_re
    (17) = '0' AND product_10_re_pipe_re(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_10_re_pipe_re(17) =
        '1' AND product_10_re_pipe_re(16) /= '1'
    ELSE (product_10_re_pipe_re(16 DOWNTO 0));
product_10_im <= (16 => '0', OTHERS => '1') WHEN product_10_re_pipe_im
    (17) = '0' AND product_10_re_pipe_im(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_10_re_pipe_im(17) =
        '1' AND product_10_re_pipe_im(16) /= '1'
    ELSE (product_10_re_pipe_im(16 DOWNTO 0));

product_mux_11 <= coeffphase1_10 WHEN ( cur_count = to_unsigned(0, 2)
    ) ELSE
                      coeffphase2_10 WHEN ( cur_count = to_unsigned
                          (1, 2) ) ELSE
                      coeffphase3_10 WHEN ( cur_count = to_unsigned
                          (2, 2) ) ELSE
                      coeffphase4_10;
temp_process15 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    product_11_re_pipe_re <= (OTHERS => '0');
    product_11_re_pipe_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN

      product_11_re_pipe_re <= delay_pipeline_re(8) * product_mux_11;
      product_11_re_pipe_im <= delay_pipeline_im(8) * product_mux_11;
```

```
    END IF;
  END IF;
END PROCESS temp_process15;

product_11_re <= (16 => '0', OTHERS => '1') WHEN product_11_re_pipe_re
    (17) = '0' AND product_11_re_pipe_re(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_11_re_pipe_re(17) =
        '1' AND product_11_re_pipe_re(16) /= '1'
    ELSE (product_11_re_pipe_re(16 DOWNTO 0));
product_11_im <= (16 => '0', OTHERS => '1') WHEN product_11_re_pipe_im
    (17) = '0' AND product_11_re_pipe_im(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_11_re_pipe_im(17) =
        '1' AND product_11_re_pipe_im(16) /= '1'
    ELSE (product_11_re_pipe_im(16 DOWNTO 0));

product_mux_12 <= coeffphase1_9 WHEN ( cur_count = to_unsigned(0, 2) )
    ELSE
                      coeffphase2_9 WHEN ( cur_count = to_unsigned(1,
                          2) ) ELSE
                      coeffphase3_9 WHEN ( cur_count = to_unsigned(2,
                          2) ) ELSE
                      coeffphase4_9;
temp_process16 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    product_12_re_pipe_re <= (OTHERS => '0');
    product_12_re_pipe_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN

      product_12_re_pipe_re <= delay_pipeline_re(7) * product_mux_12;
      product_12_re_pipe_im <= delay_pipeline_im(7) * product_mux_12;
```

```
    END IF;
  END IF;
END PROCESS temp_process16;

product_12_re <= (16 => '0', OTHERS => '1') WHEN product_12_re_pipe_re
    (17) = '0' AND product_12_re_pipe_re(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_12_re_pipe_re(17) =
        '1' AND product_12_re_pipe_re(16) /= '1'
    ELSE (product_12_re_pipe_re(16 DOWNTO 0));
product_12_im <= (16 => '0', OTHERS => '1') WHEN product_12_re_pipe_im
    (17) = '0' AND product_12_re_pipe_im(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_12_re_pipe_im(17) =
        '1' AND product_12_re_pipe_im(16) /= '1'
    ELSE (product_12_re_pipe_im(16 DOWNTO 0));

product_mux_13 <= coeffphase1_8 WHEN ( cur_count = to_unsigned(0, 2) )
    ELSE
                      coeffphase2_8 WHEN ( cur_count = to_unsigned(1,
                          2) ) ELSE
                      coeffphase3_8 WHEN ( cur_count = to_unsigned(2,
                          2) ) ELSE
                      coeffphase4_8;
temp_process17 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    product_13_re_pipe_re <= (OTHERS => '0');
    product_13_re_pipe_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN

      product_13_re_pipe_re <= delay_pipeline_re(6) * product_mux_13;
      product_13_re_pipe_im <= delay_pipeline_im(6) * product_mux_13;
```

```
    END IF;
  END IF;
END PROCESS temp_process17;

product_13_re <= (16 => '0', OTHERS => '1') WHEN product_13_re_pipe_re
    (17) = '0' AND product_13_re_pipe_re(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_13_re_pipe_re(17) =
        '1' AND product_13_re_pipe_re(16) /= '1'
    ELSE (product_13_re_pipe_re(16 DOWNTO 0));
product_13_im <= (16 => '0', OTHERS => '1') WHEN product_13_re_pipe_im
    (17) = '0' AND product_13_re_pipe_im(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_13_re_pipe_im(17) =
        '1' AND product_13_re_pipe_im(16) /= '1'
    ELSE (product_13_re_pipe_im(16 DOWNTO 0));

product_mux_14 <= coeffphase1_7 WHEN ( cur_count = to_unsigned(0, 2) )
    ELSE
                      coeffphase2_7 WHEN ( cur_count = to_unsigned(1,
                          2) ) ELSE
                      coeffphase3_7 WHEN ( cur_count = to_unsigned(2,
                          2) ) ELSE
                      coeffphase4_7;
temp_process18 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    product_14_re_pipe_re <= (OTHERS => '0');
    product_14_re_pipe_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN

      product_14_re_pipe_re <= delay_pipeline_re(5) * product_mux_14;
      product_14_re_pipe_im <= delay_pipeline_im(5) * product_mux_14;
```

```
    END IF;
  END IF;
END PROCESS temp_process18;

product_14_re <= (16 => '0', OTHERS => '1') WHEN product_14_re_pipe_re
    (17) = '0' AND product_14_re_pipe_re(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_14_re_pipe_re(17) =
        '1' AND product_14_re_pipe_re(16) /= '1'
    ELSE (product_14_re_pipe_re(16 DOWNTO 0));
product_14_im <= (16 => '0', OTHERS => '1') WHEN product_14_re_pipe_im
    (17) = '0' AND product_14_re_pipe_im(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_14_re_pipe_im(17) =
        '1' AND product_14_re_pipe_im(16) /= '1'
    ELSE (product_14_re_pipe_im(16 DOWNTO 0));

product_mux_15 <= coeffphase1_6 WHEN ( cur_count = to_unsigned(0, 2) )
    ELSE
                      coeffphase2_6 WHEN ( cur_count = to_unsigned(1,
                          2) ) ELSE
                      coeffphase3_6 WHEN ( cur_count = to_unsigned(2,
                          2) ) ELSE
                      coeffphase4_6;
temp_process19 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    product_15_re_pipe_re <= (OTHERS => '0');
    product_15_re_pipe_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN

      product_15_re_pipe_re <= delay_pipeline_re(4) * product_mux_15;
      product_15_re_pipe_im <= delay_pipeline_im(4) * product_mux_15;
```

```
    END IF;
  END IF;
END PROCESS temp_process19;

product_15_re <= (16 => '0', OTHERS => '1') WHEN product_15_re_pipe_re
    (17) = '0' AND product_15_re_pipe_re(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_15_re_pipe_re(17) =
        '1' AND product_15_re_pipe_re(16) /= '1'
    ELSE (product_15_re_pipe_re(16 DOWNTO 0));
product_15_im <= (16 => '0', OTHERS => '1') WHEN product_15_re_pipe_im
    (17) = '0' AND product_15_re_pipe_im(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_15_re_pipe_im(17) =
        '1' AND product_15_re_pipe_im(16) /= '1'
    ELSE (product_15_re_pipe_im(16 DOWNTO 0));

product_mux_16 <= coeffphase1_5 WHEN ( cur_count = to_unsigned(0, 2) )
    ELSE
                      coeffphase2_5 WHEN ( cur_count = to_unsigned(1,
                          2) ) ELSE
                      coeffphase3_5 WHEN ( cur_count = to_unsigned(2,
                          2) ) ELSE
                      coeffphase4_5;
temp_process20 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    product_16_re_pipe_re <= (OTHERS => '0');
    product_16_re_pipe_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN

      product_16_re_pipe_re <= delay_pipeline_re(3) * product_mux_16;
      product_16_re_pipe_im <= delay_pipeline_im(3) * product_mux_16;
```

```
    END IF;
  END IF;
END PROCESS temp_process20;

product_16_re <= (16 => '0', OTHERS => '1') WHEN product_16_re_pipe_re
    (17) = '0' AND product_16_re_pipe_re(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_16_re_pipe_re(17) =
        '1' AND product_16_re_pipe_re(16) /= '1'
    ELSE (product_16_re_pipe_re(16 DOWNTO 0));
product_16_im <= (16 => '0', OTHERS => '1') WHEN product_16_re_pipe_im
    (17) = '0' AND product_16_re_pipe_im(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_16_re_pipe_im(17) =
        '1' AND product_16_re_pipe_im(16) /= '1'
    ELSE (product_16_re_pipe_im(16 DOWNTO 0));

product_mux_17 <= coeffphase1_4 WHEN ( cur_count = to_unsigned(0, 2) )
    ELSE
                      coeffphase2_4 WHEN ( cur_count = to_unsigned(1,
                          2) ) ELSE
                      coeffphase3_4 WHEN ( cur_count = to_unsigned(2,
                          2) ) ELSE
                      coeffphase4_4;
temp_process21 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    product_17_re_pipe_re <= (OTHERS => '0');
    product_17_re_pipe_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN

      product_17_re_pipe_re <= delay_pipeline_re(2) * product_mux_17;
      product_17_re_pipe_im <= delay_pipeline_im(2) * product_mux_17;
```

```
    END IF;
  END IF;
END PROCESS temp_process21;

product_17_re <= (16 => '0', OTHERS => '1') WHEN product_17_re_pipe_re
    (17) = '0' AND product_17_re_pipe_re(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_17_re_pipe_re(17) =
        '1' AND product_17_re_pipe_re(16) /= '1'
    ELSE (product_17_re_pipe_re(16 DOWNTO 0));
product_17_im <= (16 => '0', OTHERS => '1') WHEN product_17_re_pipe_im
    (17) = '0' AND product_17_re_pipe_im(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_17_re_pipe_im(17) =
        '1' AND product_17_re_pipe_im(16) /= '1'
    ELSE (product_17_re_pipe_im(16 DOWNTO 0));

product_mux_18 <= coeffphase1_3 WHEN ( cur_count = to_unsigned(0, 2) )
    ELSE
                      coeffphase2_3 WHEN ( cur_count = to_unsigned(1,
                          2) ) ELSE
                      coeffphase3_3 WHEN ( cur_count = to_unsigned(2,
                          2) ) ELSE
                      coeffphase4_3;
temp_process22 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    product_18_re_pipe_re <= (OTHERS => '0');
    product_18_re_pipe_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN

      product_18_re_pipe_re <= delay_pipeline_re(1) * product_mux_18;
      product_18_re_pipe_im <= delay_pipeline_im(1) * product_mux_18;
```

```
    END IF;
  END IF;
END PROCESS temp_process22;

product_18_re <= (16 => '0', OTHERS => '1') WHEN product_18_re_pipe_re
    (17) = '0' AND product_18_re_pipe_re(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_18_re_pipe_re(17) =
        '1' AND product_18_re_pipe_re(16) /= '1'
    ELSE (product_18_re_pipe_re(16 DOWNTO 0));
product_18_im <= (16 => '0', OTHERS => '1') WHEN product_18_re_pipe_im
    (17) = '0' AND product_18_re_pipe_im(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_18_re_pipe_im(17) =
        '1' AND product_18_re_pipe_im(16) /= '1'
    ELSE (product_18_re_pipe_im(16 DOWNTO 0));

product_mux_19 <= coeffphase1_2 WHEN ( cur_count = to_unsigned(0, 2) )
    ELSE
                      coeffphase2_2 WHEN ( cur_count = to_unsigned(1,
                          2) ) ELSE
                      coeffphase3_2 WHEN ( cur_count = to_unsigned(2,
                          2) ) ELSE
                      coeffphase4_2;
temp_process23 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    product_19_re_pipe_re <= (OTHERS => '0');
    product_19_re_pipe_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN

      product_19_re_pipe_re <= delay_pipeline_re(0) * product_mux_19;
      product_19_re_pipe_im <= delay_pipeline_im(0) * product_mux_19;
```

```
    END IF;
  END IF;
END PROCESS temp_process23;

product_19_re <= (16 => '0', OTHERS => '1') WHEN product_19_re_pipe_re
    (17) = '0' AND product_19_re_pipe_re(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_19_re_pipe_re(17) =
        '1' AND product_19_re_pipe_re(16) /= '1'
    ELSE (product_19_re_pipe_re(16 DOWNTO 0));
product_19_im <= (16 => '0', OTHERS => '1') WHEN product_19_re_pipe_im
    (17) = '0' AND product_19_re_pipe_im(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_19_re_pipe_im(17) =
        '1' AND product_19_re_pipe_im(16) /= '1'
    ELSE (product_19_re_pipe_im(16 DOWNTO 0));

product_mux_20 <= coeffphase1_1 WHEN ( cur_count = to_unsigned(0, 2) )
    ELSE
                      coeffphase2_1 WHEN ( cur_count = to_unsigned(1,
                          2) ) ELSE
                      coeffphase3_1 WHEN ( cur_count = to_unsigned(2,
                          2) ) ELSE
                      coeffphase4_1;
temp_process24 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    product_20_re_pipe_re <= (OTHERS => '0');
    product_20_re_pipe_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN

      product_20_re_pipe_re <= FIR_Interpolation_in_regtype_re *
          product_mux_20;
      product_20_re_pipe_im <= FIR_Interpolation_in_regtype_im *
```

```
            product_mux_20;

    END IF;
  END IF;
END PROCESS temp_process24;

product_20_re <= (16 => '0', OTHERS => '1') WHEN product_20_re_pipe_re
    (17) = '0' AND product_20_re_pipe_re(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_20_re_pipe_re(17) =
        '1' AND product_20_re_pipe_re(16) /= '1'
    ELSE (product_20_re_pipe_re(16 DOWNTO 0));
product_20_im <= (16 => '0', OTHERS => '1') WHEN product_20_re_pipe_im
    (17) = '0' AND product_20_re_pipe_im(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN product_20_re_pipe_im(17) =
        '1' AND product_20_re_pipe_im(16) /= '1'
    ELSE (product_20_re_pipe_im(16 DOWNTO 0));

add_cast <= product_re;
add_cast_1 <= product_1_re;
add_temp <= resize(add_cast, 18) + resize(add_cast_1, 18);
sumvector1_re(0) <= (16 => '0', OTHERS => '1') WHEN add_temp(17) = '0'
    AND add_temp(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp(17) = '1' AND
        add_temp(16) /= '1'
    ELSE (add_temp(16 DOWNTO 0));

add_cast_2 <= product_im;
add_cast_3 <= product_1_im;
add_temp_1 <= resize(add_cast_2, 18) + resize(add_cast_3, 18);
sumvector1_im(0) <= (16 => '0', OTHERS => '1') WHEN add_temp_1(17) =
    '0' AND add_temp_1(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_1(17) = '1' AND
        add_temp_1(16) /= '1'
```

```
    ELSE (add_temp_1(16 DOWNTO 0));

add_cast_4 <= product_2_re;
add_cast_5 <= product_3_re;
add_temp_2 <= resize(add_cast_4, 18) + resize(add_cast_5, 18);
sumvector1_re(1) <= (16 => '0', OTHERS => '1') WHEN add_temp_2(17) =
    '0' AND add_temp_2(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_2(17) = '1' AND
        add_temp_2(16) /= '1'
    ELSE (add_temp_2(16 DOWNTO 0));

add_cast_6 <= product_2_im;
add_cast_7 <= product_3_im;
add_temp_3 <= resize(add_cast_6, 18) + resize(add_cast_7, 18);
sumvector1_im(1) <= (16 => '0', OTHERS => '1') WHEN add_temp_3(17) =
    '0' AND add_temp_3(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_3(17) = '1' AND
        add_temp_3(16) /= '1'
    ELSE (add_temp_3(16 DOWNTO 0));

add_cast_8 <= product_4_re;
add_cast_9 <= product_5_re;
add_temp_4 <= resize(add_cast_8, 18) + resize(add_cast_9, 18);
sumvector1_re(2) <= (16 => '0', OTHERS => '1') WHEN add_temp_4(17) =
    '0' AND add_temp_4(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_4(17) = '1' AND
        add_temp_4(16) /= '1'
    ELSE (add_temp_4(16 DOWNTO 0));

add_cast_10 <= product_4_im;
add_cast_11 <= product_5_im;
add_temp_5 <= resize(add_cast_10, 18) + resize(add_cast_11, 18);
sumvector1_im(2) <= (16 => '0', OTHERS => '1') WHEN add_temp_5(17) =
```

```
    '0' AND add_temp_5(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_5(17) = '1' AND
        add_temp_5(16) /= '1'
    ELSE (add_temp_5(16 DOWNTO 0));

add_cast_12 <= product_6_re;
add_cast_13 <= product_7_re;
add_temp_6 <= resize(add_cast_12, 18) + resize(add_cast_13, 18);
sumvector1_re(3) <= (16 => '0', OTHERS => '1') WHEN add_temp_6(17) =
    '0' AND add_temp_6(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_6(17) = '1' AND
        add_temp_6(16) /= '1'
    ELSE (add_temp_6(16 DOWNTO 0));

add_cast_14 <= product_6_im;
add_cast_15 <= product_7_im;
add_temp_7 <= resize(add_cast_14, 18) + resize(add_cast_15, 18);
sumvector1_im(3) <= (16 => '0', OTHERS => '1') WHEN add_temp_7(17) =
    '0' AND add_temp_7(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_7(17) = '1' AND
        add_temp_7(16) /= '1'
    ELSE (add_temp_7(16 DOWNTO 0));

add_cast_16 <= product_8_re;
add_cast_17 <= product_9_re;
add_temp_8 <= resize(add_cast_16, 18) + resize(add_cast_17, 18);
sumvector1_re(4) <= (16 => '0', OTHERS => '1') WHEN add_temp_8(17) =
    '0' AND add_temp_8(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_8(17) = '1' AND
        add_temp_8(16) /= '1'
    ELSE (add_temp_8(16 DOWNTO 0));

add_cast_18 <= product_8_im;
```

```
add_cast_19 <= product_9_im;
add_temp_9 <= resize(add_cast_18, 18) + resize(add_cast_19, 18);
sumvector1_im(4) <= (16 => '0', OTHERS => '1') WHEN add_temp_9(17) =
    '0' AND add_temp_9(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_9(17) = '1' AND
        add_temp_9(16) /= '1'
    ELSE (add_temp_9(16 DOWNTO 0));

add_cast_20 <= product_10_re;
add_cast_21 <= product_11_re;
add_temp_10 <= resize(add_cast_20, 18) + resize(add_cast_21, 18);
sumvector1_re(5) <= (16 => '0', OTHERS => '1') WHEN add_temp_10(17) =
    '0' AND add_temp_10(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_10(17) = '1' AND
        add_temp_10(16) /= '1'
    ELSE (add_temp_10(16 DOWNTO 0));

add_cast_22 <= product_10_im;
add_cast_23 <= product_11_im;
add_temp_11 <= resize(add_cast_22, 18) + resize(add_cast_23, 18);
sumvector1_im(5) <= (16 => '0', OTHERS => '1') WHEN add_temp_11(17) =
    '0' AND add_temp_11(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_11(17) = '1' AND
        add_temp_11(16) /= '1'
    ELSE (add_temp_11(16 DOWNTO 0));

add_cast_24 <= product_12_re;
add_cast_25 <= product_13_re;
add_temp_12 <= resize(add_cast_24, 18) + resize(add_cast_25, 18);
sumvector1_re(6) <= (16 => '0', OTHERS => '1') WHEN add_temp_12(17) =
    '0' AND add_temp_12(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_12(17) = '1' AND
        add_temp_12(16) /= '1'
```

```
    ELSE (add_temp_12(16 DOWNTO 0));

add_cast_26 <= product_12_im;
add_cast_27 <= product_13_im;
add_temp_13 <= resize(add_cast_26, 18) + resize(add_cast_27, 18);
sumvector1_im(6) <= (16 => '0', OTHERS => '1') WHEN add_temp_13(17) =
    '0' AND add_temp_13(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_13(17) = '1' AND
        add_temp_13(16) /= '1'
    ELSE (add_temp_13(16 DOWNTO 0));

add_cast_28 <= product_14_re;
add_cast_29 <= product_15_re;
add_temp_14 <= resize(add_cast_28, 18) + resize(add_cast_29, 18);
sumvector1_re(7) <= (16 => '0', OTHERS => '1') WHEN add_temp_14(17) =
    '0' AND add_temp_14(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_14(17) = '1' AND
        add_temp_14(16) /= '1'
    ELSE (add_temp_14(16 DOWNTO 0));

add_cast_30 <= product_14_im;
add_cast_31 <= product_15_im;
add_temp_15 <= resize(add_cast_30, 18) + resize(add_cast_31, 18);
sumvector1_im(7) <= (16 => '0', OTHERS => '1') WHEN add_temp_15(17) =
    '0' AND add_temp_15(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_15(17) = '1' AND
        add_temp_15(16) /= '1'
    ELSE (add_temp_15(16 DOWNTO 0));

add_cast_32 <= product_16_re;
add_cast_33 <= product_17_re;
add_temp_16 <= resize(add_cast_32, 18) + resize(add_cast_33, 18);
sumvector1_re(8) <= (16 => '0', OTHERS => '1') WHEN add_temp_16(17) =
```

```
    '0' AND add_temp_16(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_16(17) = '1' AND
        add_temp_16(16) /= '1'
    ELSE (add_temp_16(16 DOWNTO 0));

add_cast_34 <= product_16_im;
add_cast_35 <= product_17_im;
add_temp_17 <= resize(add_cast_34, 18) + resize(add_cast_35, 18);
sumvector1_im(8) <= (16 => '0', OTHERS => '1') WHEN add_temp_17(17) =
    '0' AND add_temp_17(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_17(17) = '1' AND
        add_temp_17(16) /= '1'
    ELSE (add_temp_17(16 DOWNTO 0));

add_cast_36 <= product_18_re;
add_cast_37 <= product_19_re;
add_temp_18 <= resize(add_cast_36, 18) + resize(add_cast_37, 18);
sumvector1_re(9) <= (16 => '0', OTHERS => '1') WHEN add_temp_18(17) =
    '0' AND add_temp_18(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_18(17) = '1' AND
        add_temp_18(16) /= '1'
    ELSE (add_temp_18(16 DOWNTO 0));

add_cast_38 <= product_18_im;
add_cast_39 <= product_19_im;
add_temp_19 <= resize(add_cast_38, 18) + resize(add_cast_39, 18);
sumvector1_im(9) <= (16 => '0', OTHERS => '1') WHEN add_temp_19(17) =
    '0' AND add_temp_19(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_19(17) = '1' AND
        add_temp_19(16) /= '1'
    ELSE (add_temp_19(16 DOWNTO 0));

sumvector1_re(10) <= product_20_re;
```

```
sumvector1_im(10) <= product_20_im;

sumdelay_pipeline_process1 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    sumdelay_pipeline1_re <= (OTHERS => (OTHERS => '0'));
    sumdelay_pipeline1_im <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN
      sumdelay_pipeline1_re(0 TO 10) <= sumvector1_re(0 TO 10);
      sumdelay_pipeline1_im(0 TO 10) <= sumvector1_im(0 TO 10);
    END IF;
  END IF;
END PROCESS sumdelay_pipeline_process1;

add_cast_40 <= sumdelay_pipeline1_re(0);
add_cast_41 <= sumdelay_pipeline1_re(1);
add_temp_20 <= resize(add_cast_40, 18) + resize(add_cast_41, 18);
sumvector2_re(0) <= (16 => '0', OTHERS => '1') WHEN add_temp_20(17) =
    '0' AND add_temp_20(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_20(17) = '1' AND
        add_temp_20(16) /= '1'
    ELSE (add_temp_20(16 DOWNTO 0));

add_cast_42 <= sumdelay_pipeline1_im(0);
add_cast_43 <= sumdelay_pipeline1_im(1);
add_temp_21 <= resize(add_cast_42, 18) + resize(add_cast_43, 18);
sumvector2_im(0) <= (16 => '0', OTHERS => '1') WHEN add_temp_21(17) =
    '0' AND add_temp_21(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_21(17) = '1' AND
        add_temp_21(16) /= '1'
    ELSE (add_temp_21(16 DOWNTO 0));
```

```
add_cast_44 <= sumdelay_pipeline1_re(2);
add_cast_45 <= sumdelay_pipeline1_re(3);
add_temp_22 <= resize(add_cast_44, 18) + resize(add_cast_45, 18);
sumvector2_re(1) <= (16 => '0', OTHERS => '1') WHEN add_temp_22(17) =
    '0' AND add_temp_22(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_22(17) = '1' AND
        add_temp_22(16) /= '1'
    ELSE (add_temp_22(16 DOWNTO 0));

add_cast_46 <= sumdelay_pipeline1_im(2);
add_cast_47 <= sumdelay_pipeline1_im(3);
add_temp_23 <= resize(add_cast_46, 18) + resize(add_cast_47, 18);
sumvector2_im(1) <= (16 => '0', OTHERS => '1') WHEN add_temp_23(17) =
    '0' AND add_temp_23(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_23(17) = '1' AND
        add_temp_23(16) /= '1'
    ELSE (add_temp_23(16 DOWNTO 0));

add_cast_48 <= sumdelay_pipeline1_re(4);
add_cast_49 <= sumdelay_pipeline1_re(5);
add_temp_24 <= resize(add_cast_48, 18) + resize(add_cast_49, 18);
sumvector2_re(2) <= (16 => '0', OTHERS => '1') WHEN add_temp_24(17) =
    '0' AND add_temp_24(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_24(17) = '1' AND
        add_temp_24(16) /= '1'
    ELSE (add_temp_24(16 DOWNTO 0));

add_cast_50 <= sumdelay_pipeline1_im(4);
add_cast_51 <= sumdelay_pipeline1_im(5);
add_temp_25 <= resize(add_cast_50, 18) + resize(add_cast_51, 18);
sumvector2_im(2) <= (16 => '0', OTHERS => '1') WHEN add_temp_25(17) =
    '0' AND add_temp_25(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_25(17) = '1' AND
```

```
        add_temp_25(16) /= '1'
    ELSE (add_temp_25(16 DOWNTO 0));

add_cast_52 <= sumdelay_pipeline1_re(6);
add_cast_53 <= sumdelay_pipeline1_re(7);
add_temp_26 <= resize(add_cast_52, 18) + resize(add_cast_53, 18);
sumvector2_re(3) <= (16 => '0', OTHERS => '1') WHEN add_temp_26(17) =
    '0' AND add_temp_26(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_26(17) = '1' AND
        add_temp_26(16) /= '1'
    ELSE (add_temp_26(16 DOWNTO 0));

add_cast_54 <= sumdelay_pipeline1_im(6);
add_cast_55 <= sumdelay_pipeline1_im(7);
add_temp_27 <= resize(add_cast_54, 18) + resize(add_cast_55, 18);
sumvector2_im(3) <= (16 => '0', OTHERS => '1') WHEN add_temp_27(17) =
    '0' AND add_temp_27(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_27(17) = '1' AND
        add_temp_27(16) /= '1'
    ELSE (add_temp_27(16 DOWNTO 0));

add_cast_56 <= sumdelay_pipeline1_re(8);
add_cast_57 <= sumdelay_pipeline1_re(9);
add_temp_28 <= resize(add_cast_56, 18) + resize(add_cast_57, 18);
sumvector2_re(4) <= (16 => '0', OTHERS => '1') WHEN add_temp_28(17) =
    '0' AND add_temp_28(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_28(17) = '1' AND
        add_temp_28(16) /= '1'
    ELSE (add_temp_28(16 DOWNTO 0));

add_cast_58 <= sumdelay_pipeline1_im(8);
add_cast_59 <= sumdelay_pipeline1_im(9);
add_temp_29 <= resize(add_cast_58, 18) + resize(add_cast_59, 18);
```

```
sumvector2_im(4) <= (16 => '0', OTHERS => '1') WHEN add_temp_29(17) =
    '0' AND add_temp_29(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_29(17) = '1' AND
        add_temp_29(16) /= '1'
    ELSE (add_temp_29(16 DOWNTO 0));

sumvector2_re(5) <= sumdelay_pipeline1_re(10);
sumvector2_im(5) <= sumdelay_pipeline1_im(10);

sumdelay_pipeline_process2 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    sumdelay_pipeline2_re <= (OTHERS => (OTHERS => '0'));
    sumdelay_pipeline2_im <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN
      sumdelay_pipeline2_re(0 TO 5) <= sumvector2_re(0 TO 5);
      sumdelay_pipeline2_im(0 TO 5) <= sumvector2_im(0 TO 5);
    END IF;
  END IF;
END PROCESS sumdelay_pipeline_process2;

add_cast_60 <= sumdelay_pipeline2_re(0);
add_cast_61 <= sumdelay_pipeline2_re(1);
add_temp_30 <= resize(add_cast_60, 18) + resize(add_cast_61, 18);
sumvector3_re(0) <= (16 => '0', OTHERS => '1') WHEN add_temp_30(17) =
    '0' AND add_temp_30(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_30(17) = '1' AND
        add_temp_30(16) /= '1'
    ELSE (add_temp_30(16 DOWNTO 0));

add_cast_62 <= sumdelay_pipeline2_im(0);
add_cast_63 <= sumdelay_pipeline2_im(1);
```

```
add_temp_31 <= resize(add_cast_62, 18) + resize(add_cast_63, 18);
sumvector3_im(0) <= (16 => '0', OTHERS => '1') WHEN add_temp_31(17) =
    '0' AND add_temp_31(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_31(17) = '1' AND
        add_temp_31(16) /= '1'
    ELSE (add_temp_31(16 DOWNTO 0));

add_cast_64 <= sumdelay_pipeline2_re(2);
add_cast_65 <= sumdelay_pipeline2_re(3);
add_temp_32 <= resize(add_cast_64, 18) + resize(add_cast_65, 18);
sumvector3_re(1) <= (16 => '0', OTHERS => '1') WHEN add_temp_32(17) =
    '0' AND add_temp_32(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_32(17) = '1' AND
        add_temp_32(16) /= '1'
    ELSE (add_temp_32(16 DOWNTO 0));

add_cast_66 <= sumdelay_pipeline2_im(2);
add_cast_67 <= sumdelay_pipeline2_im(3);
add_temp_33 <= resize(add_cast_66, 18) + resize(add_cast_67, 18);
sumvector3_im(1) <= (16 => '0', OTHERS => '1') WHEN add_temp_33(17) =
    '0' AND add_temp_33(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_33(17) = '1' AND
        add_temp_33(16) /= '1'
    ELSE (add_temp_33(16 DOWNTO 0));

add_cast_68 <= sumdelay_pipeline2_re(4);
add_cast_69 <= sumdelay_pipeline2_re(5);
add_temp_34 <= resize(add_cast_68, 18) + resize(add_cast_69, 18);
sumvector3_re(2) <= (16 => '0', OTHERS => '1') WHEN add_temp_34(17) =
    '0' AND add_temp_34(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_34(17) = '1' AND
        add_temp_34(16) /= '1'
    ELSE (add_temp_34(16 DOWNTO 0));
```

```
add_cast_70 <= sumdelay_pipeline2_im(4);
add_cast_71 <= sumdelay_pipeline2_im(5);
add_temp_35 <= resize(add_cast_70, 18) + resize(add_cast_71, 18);
sumvector3_im(2) <= (16 => '0', OTHERS => '1') WHEN add_temp_35(17) =
    '0' AND add_temp_35(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_35(17) = '1' AND
        add_temp_35(16) /= '1'
    ELSE (add_temp_35(16 DOWNTO 0));

sumdelay_pipeline_process3 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    sumdelay_pipeline3_re <= (OTHERS => (OTHERS => '0'));
    sumdelay_pipeline3_im <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN
      sumdelay_pipeline3_re(0 TO 2) <= sumvector3_re(0 TO 2);
      sumdelay_pipeline3_im(0 TO 2) <= sumvector3_im(0 TO 2);
    END IF;
  END IF;
END PROCESS sumdelay_pipeline_process3;

add_cast_72 <= sumdelay_pipeline3_re(0);
add_cast_73 <= sumdelay_pipeline3_re(1);
add_temp_36 <= resize(add_cast_72, 18) + resize(add_cast_73, 18);
sumvector4_re(0) <= (16 => '0', OTHERS => '1') WHEN add_temp_36(17) =
    '0' AND add_temp_36(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_36(17) = '1' AND
        add_temp_36(16) /= '1'
    ELSE (add_temp_36(16 DOWNTO 0));

add_cast_74 <= sumdelay_pipeline3_im(0);
```

```
add_cast_75 <= sumdelay_pipeline3_im(1);
add_temp_37 <= resize(add_cast_74, 18) + resize(add_cast_75, 18);
sumvector4_im(0) <= (16 => '0', OTHERS => '1') WHEN add_temp_37(17) =
    '0' AND add_temp_37(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_37(17) = '1' AND
        add_temp_37(16) /= '1'
    ELSE (add_temp_37(16 DOWNTO 0));

sumvector4_re(1) <= sumdelay_pipeline3_re(2);
sumvector4_im(1) <= sumdelay_pipeline3_im(2);

sumdelay_pipeline_process4 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    sumdelay_pipeline4_re <= (OTHERS => (OTHERS => '0'));
    sumdelay_pipeline4_im <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN
      sumdelay_pipeline4_re(0 TO 1) <= sumvector4_re(0 TO 1);
      sumdelay_pipeline4_im(0 TO 1) <= sumvector4_im(0 TO 1);
    END IF;
  END IF;
END PROCESS sumdelay_pipeline_process4;

add_cast_76 <= sumdelay_pipeline4_re(0);
add_cast_77 <= sumdelay_pipeline4_re(1);
add_temp_38 <= resize(add_cast_76, 18) + resize(add_cast_77, 18);
sum5_re <= (16 => '0', OTHERS => '1') WHEN add_temp_38(17) = '0' AND
    add_temp_38(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_38(17) = '1' AND
        add_temp_38(16) /= '1'
    ELSE (add_temp_38(16 DOWNTO 0));
```

```
add_cast_78 <= sumdelay_pipeline4_im(0);
add_cast_79 <= sumdelay_pipeline4_im(1);
add_temp_39 <= resize(add_cast_78, 18) + resize(add_cast_79, 18);
sum5_im <= (16 => '0', OTHERS => '1') WHEN add_temp_39(17) = '0' AND
    add_temp_39(16) /= '0'
    ELSE (16 => '1', OTHERS => '0') WHEN add_temp_39(17) = '1' AND
        add_temp_39(16) /= '1'
    ELSE (add_temp_39(16 DOWNTO 0));

output_typeconvert_re <= (15 => '0', OTHERS => '1') WHEN sum5_re(16) =
     '0' AND sum5_re(15) /= '0'
    ELSE (15 => '1', OTHERS => '0') WHEN sum5_re(16) = '1' AND sum5_re
        (15) /= '1'
    ELSE (sum5_re(15 DOWNTO 0));
output_typeconvert_im <= (15 => '0', OTHERS => '1') WHEN sum5_im(16) =
     '0' AND sum5_im(15) /= '0'
    ELSE (15 => '1', OTHERS => '0') WHEN sum5_im(16) = '1' AND sum5_im
        (15) /= '1'
    ELSE (sum5_im(15 DOWNTO 0));

DataHoldRegister_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    regout_re <= (OTHERS => '0');
    regout_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN
      regout_re <= output_typeconvert_re;
      regout_im <= output_typeconvert_im;
    END IF;
  END IF;
END PROCESS DataHoldRegister_process;
```

```
  muxout_re <= output_typeconvert_re WHEN ( enb_1_1_1 = '1' ) ELSE
               regout_re;
  muxout_im <= output_typeconvert_im WHEN ( enb_1_1_1 = '1' ) ELSE
               regout_im;
  -- Assignment Statements
  FIR_Interpolation_out_re <= std_logic_vector(muxout_re);
  FIR_Interpolation_out_im <= std_logic_vector(muxout_im);
END rtl;
```

A.1.6 FIR Decimation Filter

```
-- -------------------------------------------------------------
--
-- File Name: FIR_Decimation
-- Created: 2014-03-16 01:42:17
-- Generated by MATLAB 8.2 and HDL Coder 3.3
--
-- -------------------------------------------------------------
--
--
-- -------------------------------------------------------------
--
-- Module: FIR_Decimation
-- Source Path: /FIR_Decimation
--
-- -------------------------------------------------------------
--
-- HDL Implementation    : Fully parallel
-- Multipliers           : 81
-- Folding Factor        : 1


LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.ALL;

ENTITY FIR_Decimation IS
  PORT( clk                               :   IN    std_logic;
        enb_1_1_1                         :   IN    std_logic;
        reset                             :   IN    std_logic;
        FIR_Decimation_in_re              :   IN    std_logic_vector(15
          DOWNTO 0); -- sfix16_En15
        FIR_Decimation_in_im              :   IN    std_logic_vector(15
```

```
        DOWNTO 0); -- sfix16_En15
    FIR_Decimation_out_re          :   OUT   std_logic_vector(15
        DOWNTO 0); -- sfix16_En14
    FIR_Decimation_out_im          :   OUT   std_logic_vector(15
        DOWNTO 0)  -- sfix16_En14
    );

END FIR_Decimation;


-----------------------------------------------------------------
--Module Architecture: FIR_Decimation
-----------------------------------------------------------------

ARCHITECTURE rtl OF FIR_Decimation IS
  -- Local Functions
  -- Type Definitions
  TYPE input_pipeline_type IS ARRAY (NATURAL range <>) OF signed(15
      DOWNTO 0); -- sfix16_En15
  TYPE sumdelay_pipeline_type IS ARRAY (NATURAL range <>) OF signed(32
      DOWNTO 0); -- sfix33_En30
  -- Constants
  CONSTANT coeffphase1_1                  : signed(15 DOWNTO 0) :=
      to_signed(5, 16); -- sfix16_En15
  CONSTANT coeffphase1_2                  : signed(15 DOWNTO 0) :=
      to_signed(41, 16); -- sfix16_En15
  CONSTANT coeffphase1_3                  : signed(15 DOWNTO 0) :=
      to_signed(-83, 16); -- sfix16_En15
  CONSTANT coeffphase1_4                  : signed(15 DOWNTO 0) :=
      to_signed(88, 16); -- sfix16_En15
  CONSTANT coeffphase1_5                  : signed(15 DOWNTO 0) :=
      to_signed(-25, 16); -- sfix16_En15
  CONSTANT coeffphase1_6                  : signed(15 DOWNTO 0) :=
      to_signed(-123, 16); -- sfix16_En15
```

```
CONSTANT coeffphase1_7                  : signed(15 DOWNTO 0) :=
    to_signed(348, 16); -- sfix16_En15
CONSTANT coeffphase1_8                  : signed(15 DOWNTO 0) :=
    to_signed(-615, 16); -- sfix16_En15
CONSTANT coeffphase1_9                  : signed(15 DOWNTO 0) :=
    to_signed(869, 16); -- sfix16_En15
CONSTANT coeffphase1_10                 : signed(15 DOWNTO 0) :=
    to_signed(-1052, 16); -- sfix16_En15
CONSTANT coeffphase1_11                 : signed(15 DOWNTO 0) :=
    to_signed(17504, 16); -- sfix16_En15
CONSTANT coeffphase1_12                 : signed(15 DOWNTO 0) :=
    to_signed(-1052, 16); -- sfix16_En15
CONSTANT coeffphase1_13                 : signed(15 DOWNTO 0) :=
    to_signed(869, 16); -- sfix16_En15
CONSTANT coeffphase1_14                 : signed(15 DOWNTO 0) :=
    to_signed(-615, 16); -- sfix16_En15
CONSTANT coeffphase1_15                 : signed(15 DOWNTO 0) :=
    to_signed(348, 16); -- sfix16_En15
CONSTANT coeffphase1_16                 : signed(15 DOWNTO 0) :=
    to_signed(-123, 16); -- sfix16_En15
CONSTANT coeffphase1_17                 : signed(15 DOWNTO 0) :=
    to_signed(-25, 16); -- sfix16_En15
CONSTANT coeffphase1_18                 : signed(15 DOWNTO 0) :=
    to_signed(88, 16); -- sfix16_En15
CONSTANT coeffphase1_19                 : signed(15 DOWNTO 0) :=
    to_signed(-83, 16); -- sfix16_En15
CONSTANT coeffphase1_20                 : signed(15 DOWNTO 0) :=
    to_signed(41, 16); -- sfix16_En15
CONSTANT coeffphase1_21                 : signed(15 DOWNTO 0) :=
    to_signed(5, 16); -- sfix16_En15
CONSTANT coeffphase2_1                  : signed(15 DOWNTO 0) :=
    to_signed(-41, 16); -- sfix16_En15
CONSTANT coeffphase2_2                  : signed(15 DOWNTO 0) :=
```

```
    to_signed(60, 16); -- sfix16_En15
CONSTANT coeffphase2_3                  : signed(15 DOWNTO 0) :=
    to_signed(-43, 16); -- sfix16_En15
CONSTANT coeffphase2_4                  : signed(15 DOWNTO 0) :=
    to_signed(-19, 16); -- sfix16_En15
CONSTANT coeffphase2_5                  : signed(15 DOWNTO 0) :=
    to_signed(112, 16); -- sfix16_En15
CONSTANT coeffphase2_6                  : signed(15 DOWNTO 0) :=
    to_signed(-187, 16); -- sfix16_En15
CONSTANT coeffphase2_7                  : signed(15 DOWNTO 0) :=
    to_signed(163, 16); -- sfix16_En15
CONSTANT coeffphase2_8                  : signed(15 DOWNTO 0) :=
    to_signed(99, 16); -- sfix16_En15
CONSTANT coeffphase2_9                  : signed(15 DOWNTO 0) :=
    to_signed(-901, 16); -- sfix16_En15
CONSTANT coeffphase2_10                 : signed(15 DOWNTO 0) :=
    to_signed(3897, 16); -- sfix16_En15
CONSTANT coeffphase2_11                 : signed(15 DOWNTO 0) :=
    to_signed(15453, 16); -- sfix16_En15
CONSTANT coeffphase2_12                 : signed(15 DOWNTO 0) :=
    to_signed(-3256, 16); -- sfix16_En15
CONSTANT coeffphase2_13                 : signed(15 DOWNTO 0) :=
    to_signed(1542, 16); -- sfix16_En15
CONSTANT coeffphase2_14                 : signed(15 DOWNTO 0) :=
    to_signed(-699, 16); -- sfix16_En15
CONSTANT coeffphase2_15                 : signed(15 DOWNTO 0) :=
    to_signed(210, 16); -- sfix16_En15
CONSTANT coeffphase2_16                 : signed(15 DOWNTO 0) :=
    to_signed(46, 16); -- sfix16_En15
CONSTANT coeffphase2_17                 : signed(15 DOWNTO 0) :=
    to_signed(-132, 16); -- sfix16_En15
CONSTANT coeffphase2_18                 : signed(15 DOWNTO 0) :=
    to_signed(113, 16); -- sfix16_En15
```

```
CONSTANT coeffphase2_19                 : signed(15 DOWNTO 0) :=
    to_signed(-48, 16); -- sfix16_En15
CONSTANT coeffphase2_20                 : signed(15 DOWNTO 0) :=
    to_signed(-13, 16); -- sfix16_En15
CONSTANT coeffphase2_21                 : signed(15 DOWNTO 0) :=
    to_signed(0, 16); -- sfix16_En15
CONSTANT coeffphase3_1                  : signed(15 DOWNTO 0) :=
    to_signed(-52, 16); -- sfix16_En15
CONSTANT coeffphase3_2                  : signed(15 DOWNTO 0) :=
    to_signed(20, 16); -- sfix16_En15
CONSTANT coeffphase3_3                  : signed(15 DOWNTO 0) :=
    to_signed(48, 16); -- sfix16_En15
CONSTANT coeffphase3_4                  : signed(15 DOWNTO 0) :=
    to_signed(-124, 16); -- sfix16_En15
CONSTANT coeffphase3_5                  : signed(15 DOWNTO 0) :=
    to_signed(152, 16); -- sfix16_En15
CONSTANT coeffphase3_6                  : signed(15 DOWNTO 0) :=
    to_signed(-48, 16); -- sfix16_En15
CONSTANT coeffphase3_7                  : signed(15 DOWNTO 0) :=
    to_signed(-300, 16); -- sfix16_En15
CONSTANT coeffphase3_8                  : signed(15 DOWNTO 0) :=
    to_signed(1070, 16); -- sfix16_En15
CONSTANT coeffphase3_9                  : signed(15 DOWNTO 0) :=
    to_signed(-2790, 16); -- sfix16_En15
CONSTANT coeffphase3_10                 : signed(15 DOWNTO 0) :=
    to_signed(10188, 16); -- sfix16_En15
CONSTANT coeffphase3_11                 : signed(15 DOWNTO 0) :=
    to_signed(10188, 16); -- sfix16_En15
CONSTANT coeffphase3_12                 : signed(15 DOWNTO 0) :=
    to_signed(-2790, 16); -- sfix16_En15
CONSTANT coeffphase3_13                 : signed(15 DOWNTO 0) :=
    to_signed(1070, 16); -- sfix16_En15
CONSTANT coeffphase3_14                 : signed(15 DOWNTO 0) :=
```

```
    to_signed(-300, 16); -- sfix16_En15
CONSTANT coeffphase3_15                 : signed(15 DOWNTO 0) :=
    to_signed(-48, 16); -- sfix16_En15
CONSTANT coeffphase3_16                 : signed(15 DOWNTO 0) :=
    to_signed(152, 16); -- sfix16_En15
CONSTANT coeffphase3_17                 : signed(15 DOWNTO 0) :=
    to_signed(-124, 16); -- sfix16_En15
CONSTANT coeffphase3_18                 : signed(15 DOWNTO 0) :=
    to_signed(48, 16); -- sfix16_En15
CONSTANT coeffphase3_19                 : signed(15 DOWNTO 0) :=
    to_signed(20, 16); -- sfix16_En15
CONSTANT coeffphase3_20                 : signed(15 DOWNTO 0) :=
    to_signed(-52, 16); -- sfix16_En15
CONSTANT coeffphase3_21                 : signed(15 DOWNTO 0) :=
    to_signed(0, 16); -- sfix16_En15
CONSTANT coeffphase4_1                  : signed(15 DOWNTO 0) :=
    to_signed(-13, 16); -- sfix16_En15
CONSTANT coeffphase4_2                  : signed(15 DOWNTO 0) :=
    to_signed(-48, 16); -- sfix16_En15
CONSTANT coeffphase4_3                  : signed(15 DOWNTO 0) :=
    to_signed(113, 16); -- sfix16_En15
CONSTANT coeffphase4_4                  : signed(15 DOWNTO 0) :=
    to_signed(-132, 16); -- sfix16_En15
CONSTANT coeffphase4_5                  : signed(15 DOWNTO 0) :=
    to_signed(46, 16); -- sfix16_En15
CONSTANT coeffphase4_6                  : signed(15 DOWNTO 0) :=
    to_signed(210, 16); -- sfix16_En15
CONSTANT coeffphase4_7                  : signed(15 DOWNTO 0) :=
    to_signed(-699, 16); -- sfix16_En15
CONSTANT coeffphase4_8                  : signed(15 DOWNTO 0) :=
    to_signed(1542, 16); -- sfix16_En15
CONSTANT coeffphase4_9                  : signed(15 DOWNTO 0) :=
    to_signed(-3256, 16); -- sfix16_En15
```

```
CONSTANT coeffphase4_10                 : signed(15 DOWNTO 0) :=
    to_signed(15453, 16); -- sfix16_En15
CONSTANT coeffphase4_11                 : signed(15 DOWNTO 0) :=
    to_signed(3897, 16); -- sfix16_En15
CONSTANT coeffphase4_12                 : signed(15 DOWNTO 0) :=
    to_signed(-901, 16); -- sfix16_En15
CONSTANT coeffphase4_13                 : signed(15 DOWNTO 0) :=
    to_signed(99, 16); -- sfix16_En15
CONSTANT coeffphase4_14                 : signed(15 DOWNTO 0) :=
    to_signed(163, 16); -- sfix16_En15
CONSTANT coeffphase4_15                 : signed(15 DOWNTO 0) :=
    to_signed(-187, 16); -- sfix16_En15
CONSTANT coeffphase4_16                 : signed(15 DOWNTO 0) :=
    to_signed(112, 16); -- sfix16_En15
CONSTANT coeffphase4_17                 : signed(15 DOWNTO 0) :=
    to_signed(-19, 16); -- sfix16_En15
CONSTANT coeffphase4_18                 : signed(15 DOWNTO 0) :=
    to_signed(-43, 16); -- sfix16_En15
CONSTANT coeffphase4_19                 : signed(15 DOWNTO 0) :=
    to_signed(60, 16); -- sfix16_En15
CONSTANT coeffphase4_20                 : signed(15 DOWNTO 0) :=
    to_signed(-41, 16); -- sfix16_En15
CONSTANT coeffphase4_21                 : signed(15 DOWNTO 0) :=
    to_signed(0, 16); -- sfix16_En15

-- Signals
SIGNAL ring_count                       : unsigned(3 DOWNTO 0); --
    ufix4
SIGNAL phase_0                          : std_logic; -- boolean
SIGNAL phase_1                          : std_logic; -- boolean
SIGNAL phase_2                          : std_logic; -- boolean
SIGNAL phase_3                          : std_logic; -- boolean
SIGNAL input_typeconvert_re             : signed(15 DOWNTO 0); --
```

```
    sfix16_En15
SIGNAL input_typeconvert_im             : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL input_pipeline_phase0_re         : input_pipeline_type(0 TO 19)
    ; -- sfix16_En15
SIGNAL input_pipeline_phase0_im         : input_pipeline_type(0 TO 19)
    ; -- sfix16_En15
SIGNAL input_pipeline_phase1_re         : input_pipeline_type(0 TO 19)
    ; -- sfix16_En15
SIGNAL input_pipeline_phase1_im         : input_pipeline_type(0 TO 19)
    ; -- sfix16_En15
SIGNAL input_pipeline_phase2_re         : input_pipeline_type(0 TO 19)
    ; -- sfix16_En15
SIGNAL input_pipeline_phase2_im         : input_pipeline_type(0 TO 19)
    ; -- sfix16_En15
SIGNAL input_pipeline_phase3_re         : input_pipeline_type(0 TO 19)
    ; -- sfix16_En15
SIGNAL input_pipeline_phase3_im         : input_pipeline_type(0 TO 19)
    ; -- sfix16_En15
SIGNAL product_phase0_1_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase0_1_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp                         : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_1                       : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase0_2_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase0_2_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_2                       : signed(31 DOWNTO 0); --
    sfix32_En30
```

```
SIGNAL mul_temp_3                       : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase0_3_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase0_3_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_4                       : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_5                       : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase0_4_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase0_4_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_6                       : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_7                       : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase0_5_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase0_5_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_8                       : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_9                       : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase0_6_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase0_6_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_10                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_11                      : signed(31 DOWNTO 0); --
```

```
    sfix32_En30
SIGNAL product_phase0_7_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase0_7_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_12                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_13                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase0_8_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase0_8_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_14                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_15                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase0_9_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase0_9_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_16                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_17                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase0_10_re             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase0_10_im             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_18                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_19                      : signed(31 DOWNTO 0); --
    sfix32_En30
```

```
SIGNAL product_phase0_11_re             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase0_11_im             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_20                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_21                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase0_12_re             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase0_12_im             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_22                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_23                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase0_13_re             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase0_13_im             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_24                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_25                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase0_14_re             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase0_14_im             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_26                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_27                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase0_15_re             : signed(30 DOWNTO 0); --
```

```
    sfix31_En30
SIGNAL product_phase0_15_im             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_28                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_29                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase0_16_re             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase0_16_im             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_30                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_31                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase0_17_re             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase0_17_im             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_32                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_33                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase0_18_re             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase0_18_im             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_34                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_35                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase0_19_re             : signed(30 DOWNTO 0); --
    sfix31_En30
```

```
SIGNAL product_phase0_19_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_36                       : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_37                       : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase0_20_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase0_20_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_38                       : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_39                       : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase0_21_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase0_21_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_40                       : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_41                       : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase1_1_re               : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase1_1_im               : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_42                       : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_43                       : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase1_2_re               : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase1_2_im               : signed(30 DOWNTO 0); --
```

```
    sfix31_En30
SIGNAL mul_temp_44                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_45                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase1_3_re          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase1_3_im          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_46                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_47                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase1_4_re          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase1_4_im          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_48                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_49                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase1_5_re          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase1_5_im          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_50                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_51                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase1_6_re          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase1_6_im          : signed(30 DOWNTO 0); --
    sfix31_En30
```

```
SIGNAL mul_temp_52                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_53                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase1_7_re          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase1_7_im          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_54                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_55                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase1_8_re          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase1_8_im          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_56                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_57                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase1_9_re          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase1_9_im          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_58                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_59                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase1_10_re         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase1_10_im         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_60                  : signed(31 DOWNTO 0); --
```

```
    sfix32_En30
SIGNAL mul_temp_61                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase1_11_re         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase1_11_im         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_62                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_63                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase1_12_re         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase1_12_im         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_64                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_65                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase1_13_re         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase1_13_im         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_66                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_67                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase1_14_re         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase1_14_im         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_68                  : signed(31 DOWNTO 0); --
    sfix32_En30
```

```
SIGNAL mul_temp_69                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase1_15_re         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase1_15_im         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_70                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_71                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase1_16_re         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase1_16_im         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_72                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_73                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase1_17_re         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase1_17_im         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_74                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_75                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase1_18_re         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase1_18_im         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_76                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_77                  : signed(31 DOWNTO 0); --
```

```
    sfix32_En30
SIGNAL product_phase1_19_re             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase1_19_im             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_78                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_79                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase1_20_re             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase1_20_im             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_80                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_81                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase2_1_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase2_1_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_82                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_83                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase2_2_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase2_2_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_84                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_85                      : signed(31 DOWNTO 0); --
    sfix32_En30
```

```
SIGNAL product_phase2_3_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase2_3_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_86                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_87                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase2_4_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase2_4_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_88                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_89                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase2_5_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase2_5_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_90                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_91                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase2_6_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase2_6_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_92                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_93                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase2_7_re              : signed(30 DOWNTO 0); --
```

```
    sfix31_En30
SIGNAL product_phase2_7_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_94                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_95                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase2_8_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase2_8_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_96                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_97                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase2_9_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase2_9_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_98                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_99                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase2_10_re             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase2_10_im             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_100                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_101                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase2_11_re             : signed(30 DOWNTO 0); --
    sfix31_En30
```

```
SIGNAL product_phase2_11_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_102                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_103                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase2_12_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase2_12_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_104                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_105                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase2_13_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase2_13_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_106                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_107                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase2_14_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase2_14_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_108                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_109                      : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase2_15_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase2_15_im              : signed(30 DOWNTO 0); --
```

```
    sfix31_En30
SIGNAL mul_temp_110                 : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_111                 : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase2_16_re         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase2_16_im         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_112                 : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_113                 : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase2_17_re         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase2_17_im         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_114                 : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_115                 : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase2_18_re         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase2_18_im         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_116                 : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_117                 : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase2_19_re         : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase2_19_im         : signed(30 DOWNTO 0); --
    sfix31_En30
```

```
SIGNAL mul_temp_118                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_119                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase2_20_re             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase2_20_im             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_120                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_121                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase3_1_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase3_1_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_122                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_123                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase3_2_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase3_2_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_124                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_125                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase3_3_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase3_3_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_126                     : signed(31 DOWNTO 0); --
```

```
    sfix32_En30
SIGNAL mul_temp_127                 : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase3_4_re          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase3_4_im          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_128                 : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_129                 : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase3_5_re          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase3_5_im          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_130                 : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_131                 : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase3_6_re          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase3_6_im          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_132                 : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_133                 : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase3_7_re          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase3_7_im          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_134                 : signed(31 DOWNTO 0); --
    sfix32_En30
```

```
SIGNAL mul_temp_135                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase3_8_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase3_8_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_136                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_137                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase3_9_re              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase3_9_im              : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_138                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_139                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase3_10_re             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase3_10_im             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_140                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_141                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase3_11_re             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase3_11_im             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_142                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_143                     : signed(31 DOWNTO 0); --
```

```
    sfix32_En30
SIGNAL product_phase3_12_re             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase3_12_im             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_144                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_145                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase3_13_re             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase3_13_im             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_146                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_147                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase3_14_re             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase3_14_im             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_148                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_149                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase3_15_re             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase3_15_im             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_150                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_151                     : signed(31 DOWNTO 0); --
    sfix32_En30
```

```
SIGNAL product_phase3_16_re          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase3_16_im          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_152                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_153                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase3_17_re          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase3_17_im          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_154                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_155                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase3_18_re          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase3_18_im          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_156                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_157                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase3_19_re          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_phase3_19_im          : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_158                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_159                  : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_phase3_20_re          : signed(30 DOWNTO 0); --
```

```
    sfix31_En30
SIGNAL product_phase3_20_im             : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL mul_temp_160                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL mul_temp_161                     : signed(31 DOWNTO 0); --
    sfix32_En30
SIGNAL product_pipeline_phase0_1_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_1_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_2_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_2_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_3_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_3_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_4_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_4_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_5_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_5_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_6_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_6_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_7_re     : signed(30 DOWNTO 0); --
    sfix31_En30
```

```
SIGNAL product_pipeline_phase0_7_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_8_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_8_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_9_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_9_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_10_re    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_10_im    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_11_re    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_11_im    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_12_re    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_12_im    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_13_re    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_13_im    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_14_re    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_14_im    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_15_re    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_15_im    : signed(30 DOWNTO 0); --
```

```
    sfix31_En30
SIGNAL product_pipeline_phase0_16_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_16_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_17_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_17_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_18_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_18_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_19_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_19_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_20_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_20_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_21_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase0_21_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_1_re      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_1_im      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_2_re      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_2_im      : signed(30 DOWNTO 0); --
    sfix31_En30
```

```
SIGNAL product_pipeline_phase1_3_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_3_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_4_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_4_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_5_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_5_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_6_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_6_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_7_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_7_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_8_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_8_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_9_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_9_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_10_re    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_10_im    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_11_re    : signed(30 DOWNTO 0); --
```

```
    sfix31_En30
SIGNAL product_pipeline_phase1_11_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_12_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_12_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_13_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_13_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_14_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_14_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_15_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_15_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_16_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_16_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_17_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_17_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_18_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_18_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_19_re     : signed(30 DOWNTO 0); --
    sfix31_En30
```

```
SIGNAL product_pipeline_phase1_19_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_20_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase1_20_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_1_re      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_1_im      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_2_re      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_2_im      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_3_re      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_3_im      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_4_re      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_4_im      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_5_re      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_5_im      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_6_re      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_6_im      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_7_re      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_7_im      : signed(30 DOWNTO 0); --
```

```
    sfix31_En30
SIGNAL product_pipeline_phase2_8_re      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_8_im      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_9_re      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_9_im      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_10_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_10_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_11_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_11_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_12_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_12_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_13_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_13_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_14_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_14_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_15_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_15_im     : signed(30 DOWNTO 0); --
    sfix31_En30
```

```
SIGNAL product_pipeline_phase2_16_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_16_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_17_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_17_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_18_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_18_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_19_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_19_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_20_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase2_20_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_1_re      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_1_im      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_2_re      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_2_im      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_3_re      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_3_im      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_4_re      : signed(30 DOWNTO 0); --
```

```
    sfix31_En30
SIGNAL product_pipeline_phase3_4_im      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_5_re      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_5_im      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_6_re      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_6_im      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_7_re      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_7_im      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_8_re      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_8_im      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_9_re      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_9_im      : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_10_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_10_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_11_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_11_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_12_re     : signed(30 DOWNTO 0); --
    sfix31_En30
```

```
SIGNAL product_pipeline_phase3_12_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_13_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_13_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_14_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_14_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_15_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_15_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_16_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_16_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_17_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_17_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_18_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_18_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_19_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_19_im     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_20_re     : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL product_pipeline_phase3_20_im     : signed(30 DOWNTO 0); --
```

```
    sfix31_En30
SIGNAL quantized_sum_re          : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL quantized_sum_im          : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL sumvector1_re             : sumdelay_pipeline_type(0 TO
    40); -- sfix33_En30
SIGNAL sumvector1_im             : sumdelay_pipeline_type(0 TO
    40); -- sfix33_En30
SIGNAL add_cast                  : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp                  : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_2                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_3                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_1                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_4                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_5                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_2                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_6                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_7                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_3                : signed(33 DOWNTO 0); --
    sfix34_En30
```

```
SIGNAL add_cast_8                       : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_9                       : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_4                       : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_10                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_11                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_5                       : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_12                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_13                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_6                       : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_14                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_15                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_7                       : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_16                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_17                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_8                       : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_18                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_19                      : signed(32 DOWNTO 0); --
```

```
    sfix33_En30
SIGNAL add_temp_9                       : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_20                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_21                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_10                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_22                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_23                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_11                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_24                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_25                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_12                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_26                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_27                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_13                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_28                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_29                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_14                      : signed(33 DOWNTO 0); --
    sfix34_En30
```

```
SIGNAL add_cast_30                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_31                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_15                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_32                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_33                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_16                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_34                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_35                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_17                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_36                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_37                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_18                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_38                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_39                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_19                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_40                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_41                      : signed(32 DOWNTO 0); --
```

```
    sfix33_En30
SIGNAL add_temp_20                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_42                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_43                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_21                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_44                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_45                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_22                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_46                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_47                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_23                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_48                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_49                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_24                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_50                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_51                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_25                    : signed(33 DOWNTO 0); --
    sfix34_En30
```

```
SIGNAL add_cast_52                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_53                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_26                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_54                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_55                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_27                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_56                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_57                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_28                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_58                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_59                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_29                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_60                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_61                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_30                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_62                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_63                      : signed(32 DOWNTO 0); --
```

```
    sfix33_En30
SIGNAL add_temp_31                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_64                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_65                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_32                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_66                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_67                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_33                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_68                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_69                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_34                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_70                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_71                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_35                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_72                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_73                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_36                    : signed(33 DOWNTO 0); --
    sfix34_En30
```

```
SIGNAL add_cast_74                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_75                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_37                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_76                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_77                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_38                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_78                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_79                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_39                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_80                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_81                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_40                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_82                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_83                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_41                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_84                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_85                      : signed(32 DOWNTO 0); --
```

```
    sfix33_En30
SIGNAL add_temp_42                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_86                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_87                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_43                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_88                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_89                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_44                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_90                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_91                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_45                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_92                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_93                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_46                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_94                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_95                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_47                    : signed(33 DOWNTO 0); --
    sfix34_En30
```

```
SIGNAL add_cast_96                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_97                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_48                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_98                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_99                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_49                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_100                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_101                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_50                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_102                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_103                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_51                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_104                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_105                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_52                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_106                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_107                     : signed(32 DOWNTO 0); --
```

```
    sfix33_En30
SIGNAL add_temp_53                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_108                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_109                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_54                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_110                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_111                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_55                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_112                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_113                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_56                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_114                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_115                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_57                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_116                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_117                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_58                    : signed(33 DOWNTO 0); --
    sfix34_En30
```

```
SIGNAL add_cast_118                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_119                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_59                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_120                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_121                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_60                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_122                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_123                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_61                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_124                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_125                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_62                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_126                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_127                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_63                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_128                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_129                     : signed(32 DOWNTO 0); --
```

```
    sfix33_En30
SIGNAL add_temp_64                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_130                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_131                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_65                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_132                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_133                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_66                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_134                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_135                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_67                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_136                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_137                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_68                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_138                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_139                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_69                    : signed(33 DOWNTO 0); --
    sfix34_En30
```

```
SIGNAL add_cast_140                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_141                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_70                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_142                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_143                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_71                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_144                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_145                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_72                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_146                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_147                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_73                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_148                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_149                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_74                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_150                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_151                     : signed(32 DOWNTO 0); --
```

```
    sfix33_En30
SIGNAL add_temp_75                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_152                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_153                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_76                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_154                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_155                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_77                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_156                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_157                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_78                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_158                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_159                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_79                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL sumdelay_pipeline1_re            : sumdelay_pipeline_type(0 TO
    40); -- sfix33_En30
SIGNAL sumdelay_pipeline1_im            : sumdelay_pipeline_type(0 TO
    40); -- sfix33_En30
SIGNAL sumvector2_re                    : sumdelay_pipeline_type(0 TO
    20); -- sfix33_En30
```

```
SIGNAL sumvector2_im                    : sumdelay_pipeline_type(0 TO
    20); -- sfix33_En30
SIGNAL add_cast_160                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_161                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_80                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_162                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_163                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_81                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_164                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_165                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_82                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_166                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_167                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_83                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_168                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_169                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_84                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_170                     : signed(32 DOWNTO 0); --
```

```
    sfix33_En30
SIGNAL add_cast_171                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_85                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_172                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_173                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_86                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_174                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_175                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_87                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_176                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_177                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_88                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_178                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_179                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_89                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_180                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_181                     : signed(32 DOWNTO 0); --
    sfix33_En30
```

```
SIGNAL add_temp_90                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_182                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_183                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_91                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_184                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_185                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_92                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_186                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_187                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_93                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_188                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_189                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_94                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_190                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_191                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_95                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_192                     : signed(32 DOWNTO 0); --
```

```
    sfix33_En30
SIGNAL add_cast_193                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_96                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_194                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_195                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_97                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_196                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_197                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_98                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_198                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_199                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_99                      : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_200                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_201                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_100                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_202                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_203                     : signed(32 DOWNTO 0); --
    sfix33_En30
```

```
SIGNAL add_temp_101                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_204                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_205                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_102                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_206                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_207                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_103                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_208                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_209                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_104                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_210                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_211                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_105                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_212                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_213                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_106                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_214                     : signed(32 DOWNTO 0); --
```

```
    sfix33_En30
SIGNAL add_cast_215                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_107                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_216                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_217                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_108                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_218                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_219                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_109                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_220                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_221                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_110                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_222                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_223                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_111                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_224                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_225                    : signed(32 DOWNTO 0); --
    sfix33_En30
```

```
SIGNAL add_temp_112                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_226                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_227                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_113                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_228                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_229                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_114                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_230                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_231                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_115                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_232                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_233                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_116                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_234                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_235                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_117                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_236                     : signed(32 DOWNTO 0); --
```

```
    sfix33_En30
SIGNAL add_cast_237                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_118                 : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_238                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_239                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_119                 : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL sumdelay_pipeline2_re        : sumdelay_pipeline_type(0 TO
    20); -- sfix33_En30
SIGNAL sumdelay_pipeline2_im        : sumdelay_pipeline_type(0 TO
    20); -- sfix33_En30
SIGNAL sumvector3_re                : sumdelay_pipeline_type(0 TO
    10); -- sfix33_En30
SIGNAL sumvector3_im                : sumdelay_pipeline_type(0 TO
    10); -- sfix33_En30
SIGNAL add_cast_240                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_241                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_120                 : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_242                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_243                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_121                 : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_244                 : signed(32 DOWNTO 0); --
    sfix33_En30
```

```
SIGNAL add_cast_245                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_122                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_246                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_247                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_123                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_248                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_249                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_124                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_250                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_251                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_125                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_252                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_253                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_126                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_254                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_255                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_127                     : signed(33 DOWNTO 0); --
```

```
    sfix34_En30
SIGNAL add_cast_256                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_257                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_128                 : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_258                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_259                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_129                 : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_260                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_261                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_130                 : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_262                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_263                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_131                 : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_264                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_265                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_132                 : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_266                 : signed(32 DOWNTO 0); --
    sfix33_En30
```

```
SIGNAL add_cast_267                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_133                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_268                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_269                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_134                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_270                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_271                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_135                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_272                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_273                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_136                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_274                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_275                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_137                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_276                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_277                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_138                     : signed(33 DOWNTO 0); --
```

```
    sfix34_En30
SIGNAL add_cast_278                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_279                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_139                 : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL sumdelay_pipeline3_re        : sumdelay_pipeline_type(0 TO
    10); -- sfix33_En30
SIGNAL sumdelay_pipeline3_im        : sumdelay_pipeline_type(0 TO
    10); -- sfix33_En30
SIGNAL sumvector4_re                : sumdelay_pipeline_type(0 TO
    5); -- sfix33_En30
SIGNAL sumvector4_im                : sumdelay_pipeline_type(0 TO
    5); -- sfix33_En30
SIGNAL add_cast_280                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_281                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_140                 : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_282                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_283                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_141                 : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_284                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_285                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_142                 : signed(33 DOWNTO 0); --
    sfix34_En30
```

```
SIGNAL add_cast_286                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_287                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_143                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_288                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_289                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_144                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_290                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_291                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_145                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_292                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_293                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_146                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_294                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_295                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_147                     : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_296                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_297                     : signed(32 DOWNTO 0); --
```

```
    sfix33_En30
SIGNAL add_temp_148                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_298                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_299                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_149                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL sumdelay_pipeline4_re       : sumdelay_pipeline_type(0 TO
    5); -- sfix33_En30
SIGNAL sumdelay_pipeline4_im       : sumdelay_pipeline_type(0 TO
    5); -- sfix33_En30
SIGNAL sumvector5_re               : sumdelay_pipeline_type(0 TO
    2); -- sfix33_En30
SIGNAL sumvector5_im               : sumdelay_pipeline_type(0 TO
    2); -- sfix33_En30
SIGNAL add_cast_300                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_301                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_150                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_302                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_303                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_151                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_304                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_305                : signed(32 DOWNTO 0); --
    sfix33_En30
```

```
SIGNAL add_temp_152                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_306                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_307                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_153                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_308                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_309                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_154                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_310                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_311                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_155                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL sumdelay_pipeline5_re           : sumdelay_pipeline_type(0 TO
    2); -- sfix33_En30
SIGNAL sumdelay_pipeline5_im           : sumdelay_pipeline_type(0 TO
    2); -- sfix33_En30
SIGNAL sumvector6_re                   : sumdelay_pipeline_type(0 TO
    1); -- sfix33_En30
SIGNAL sumvector6_im                   : sumdelay_pipeline_type(0 TO
    1); -- sfix33_En30
SIGNAL add_cast_312                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_313                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_156                    : signed(33 DOWNTO 0); --
```

```
    sfix34_En30
SIGNAL add_cast_314                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_315                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_157                 : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL sumdelay_pipeline6_re        : sumdelay_pipeline_type(0 TO
    1); -- sfix33_En30
SIGNAL sumdelay_pipeline6_im        : sumdelay_pipeline_type(0 TO
    1); -- sfix33_En30
SIGNAL sum7_re                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL sum7_im                      : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_316                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_317                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_158                 : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_cast_318                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_319                 : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_temp_159                 : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL output_typeconvert_re        : signed(15 DOWNTO 0); --
    sfix16_En14
SIGNAL output_typeconvert_im        : signed(15 DOWNTO 0); --
    sfix16_En14
SIGNAL regout_re                    : signed(15 DOWNTO 0); --
    sfix16_En14
```

```
  SIGNAL regout_im                        : signed(15 DOWNTO 0); --
      sfix16_En14
  SIGNAL muxout_re                        : signed(15 DOWNTO 0); --
      sfix16_En14
  SIGNAL muxout_im                        : signed(15 DOWNTO 0); --
      sfix16_En14


BEGIN

  -- Block Statements
  ce_output : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      ring_count <= to_unsigned(1, 4);
    ELSIF clk'event AND clk = '1' THEN
      IF enb_1_1_1 = '1' THEN
        ring_count <= ring_count(0) & ring_count(3 DOWNTO 1);
      END IF;
    END IF;
  END PROCESS ce_output;

  phase_0 <= ring_count(0)  AND enb_1_1_1;

  phase_1 <= ring_count(1)  AND enb_1_1_1;

  phase_2 <= ring_count(2)  AND enb_1_1_1;

  phase_3 <= ring_count(3)  AND enb_1_1_1;

  input_typeconvert_re <= signed(FIR_Decimation_in_re);
  input_typeconvert_im <= signed(FIR_Decimation_in_im);
```

```
Delay_Pipeline_Phase0_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    input_pipeline_phase0_re(0 TO 19) <= (OTHERS => (OTHERS => '0'));
    input_pipeline_phase0_im(0 TO 19) <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF phase_0 = '1' THEN
      input_pipeline_phase0_re(0) <= input_typeconvert_re;
      input_pipeline_phase0_re(1 TO 19) <= input_pipeline_phase0_re(0
          TO 18);
      input_pipeline_phase0_im(0) <= input_typeconvert_im;
      input_pipeline_phase0_im(1 TO 19) <= input_pipeline_phase0_im(0
          TO 18);
    END IF;
  END IF;
END PROCESS Delay_Pipeline_Phase0_process;

Delay_Pipeline_Phase1_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    input_pipeline_phase1_re(0 TO 19) <= (OTHERS => (OTHERS => '0'));
    input_pipeline_phase1_im(0 TO 19) <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF phase_1 = '1' THEN
      input_pipeline_phase1_re(0) <= input_typeconvert_re;
      input_pipeline_phase1_re(1 TO 19) <= input_pipeline_phase1_re(0
          TO 18);
      input_pipeline_phase1_im(0) <= input_typeconvert_im;
      input_pipeline_phase1_im(1 TO 19) <= input_pipeline_phase1_im(0
          TO 18);
    END IF;
  END IF;
END PROCESS Delay_Pipeline_Phase1_process;
```

```
Delay_Pipeline_Phase2_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    input_pipeline_phase2_re(0 TO 19) <= (OTHERS => (OTHERS => '0'));
    input_pipeline_phase2_im(0 TO 19) <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF phase_2 = '1' THEN
      input_pipeline_phase2_re(0) <= input_typeconvert_re;
      input_pipeline_phase2_re(1 TO 19) <= input_pipeline_phase2_re(0
          TO 18);
      input_pipeline_phase2_im(0) <= input_typeconvert_im;
      input_pipeline_phase2_im(1 TO 19) <= input_pipeline_phase2_im(0
          TO 18);
    END IF;
  END IF;
END PROCESS Delay_Pipeline_Phase2_process;

Delay_Pipeline_Phase3_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    input_pipeline_phase3_re(0 TO 19) <= (OTHERS => (OTHERS => '0'));
    input_pipeline_phase3_im(0 TO 19) <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF phase_3 = '1' THEN
      input_pipeline_phase3_re(0) <= input_typeconvert_re;
      input_pipeline_phase3_re(1 TO 19) <= input_pipeline_phase3_re(0
          TO 18);
      input_pipeline_phase3_im(0) <= input_typeconvert_im;
      input_pipeline_phase3_im(1 TO 19) <= input_pipeline_phase3_im(0
          TO 18);
    END IF;
  END IF;
```

```
END PROCESS Delay_Pipeline_Phase3_process;

mul_temp <= input_typeconvert_re * coeffphase1_1;
product_phase0_1_re <= (30 => '0', OTHERS => '1') WHEN mul_temp(31) =
    '0' AND mul_temp(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp(31) = '1' AND
        mul_temp(30) /= '1'
    ELSE (mul_temp(30 DOWNTO 0));

mul_temp_1 <= input_typeconvert_im * coeffphase1_1;
product_phase0_1_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_1(31)
   = '0' AND mul_temp_1(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_1(31) = '1' AND
        mul_temp_1(30) /= '1'
    ELSE (mul_temp_1(30 DOWNTO 0));

mul_temp_2 <= input_pipeline_phase0_re(0) * coeffphase1_2;
product_phase0_2_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_2(31)
   = '0' AND mul_temp_2(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_2(31) = '1' AND
        mul_temp_2(30) /= '1'
    ELSE (mul_temp_2(30 DOWNTO 0));

mul_temp_3 <= input_pipeline_phase0_im(0) * coeffphase1_2;
product_phase0_2_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_3(31)
   = '0' AND mul_temp_3(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_3(31) = '1' AND
        mul_temp_3(30) /= '1'
    ELSE (mul_temp_3(30 DOWNTO 0));

mul_temp_4 <= input_pipeline_phase0_re(1) * coeffphase1_3;
product_phase0_3_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_4(31)
   = '0' AND mul_temp_4(30) /= '0'
```

```
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_4(31) = '1' AND
        mul_temp_4(30) /= '1'
    ELSE (mul_temp_4(30 DOWNTO 0));

mul_temp_5 <= input_pipeline_phase0_im(1) * coeffphase1_3;
product_phase0_3_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_5(31)
   = '0' AND mul_temp_5(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_5(31) = '1' AND
        mul_temp_5(30) /= '1'
    ELSE (mul_temp_5(30 DOWNTO 0));

mul_temp_6 <= input_pipeline_phase0_re(2) * coeffphase1_4;
product_phase0_4_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_6(31)
   = '0' AND mul_temp_6(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_6(31) = '1' AND
        mul_temp_6(30) /= '1'
    ELSE (mul_temp_6(30 DOWNTO 0));

mul_temp_7 <= input_pipeline_phase0_im(2) * coeffphase1_4;
product_phase0_4_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_7(31)
   = '0' AND mul_temp_7(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_7(31) = '1' AND
        mul_temp_7(30) /= '1'
    ELSE (mul_temp_7(30 DOWNTO 0));

mul_temp_8 <= input_pipeline_phase0_re(3) * coeffphase1_5;
product_phase0_5_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_8(31)
   = '0' AND mul_temp_8(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_8(31) = '1' AND
        mul_temp_8(30) /= '1'
    ELSE (mul_temp_8(30 DOWNTO 0));

mul_temp_9 <= input_pipeline_phase0_im(3) * coeffphase1_5;
```

```
product_phase0_5_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_9(31)
   = '0' AND mul_temp_9(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_9(31) = '1' AND
        mul_temp_9(30) /= '1'
    ELSE (mul_temp_9(30 DOWNTO 0));

mul_temp_10 <= input_pipeline_phase0_re(4) * coeffphase1_6;
product_phase0_6_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_10(31)
    = '0' AND mul_temp_10(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_10(31) = '1' AND
        mul_temp_10(30) /= '1'
    ELSE (mul_temp_10(30 DOWNTO 0));

mul_temp_11 <= input_pipeline_phase0_im(4) * coeffphase1_6;
product_phase0_6_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_11(31)
    = '0' AND mul_temp_11(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_11(31) = '1' AND
        mul_temp_11(30) /= '1'
    ELSE (mul_temp_11(30 DOWNTO 0));

mul_temp_12 <= input_pipeline_phase0_re(5) * coeffphase1_7;
product_phase0_7_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_12(31)
    = '0' AND mul_temp_12(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_12(31) = '1' AND
        mul_temp_12(30) /= '1'
    ELSE (mul_temp_12(30 DOWNTO 0));

mul_temp_13 <= input_pipeline_phase0_im(5) * coeffphase1_7;
product_phase0_7_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_13(31)
    = '0' AND mul_temp_13(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_13(31) = '1' AND
        mul_temp_13(30) /= '1'
    ELSE (mul_temp_13(30 DOWNTO 0));
```

```
mul_temp_14 <= input_pipeline_phase0_re(6) * coeffphase1_8;
product_phase0_8_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_14(31)
    = '0' AND mul_temp_14(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_14(31) = '1' AND
        mul_temp_14(30) /= '1'
    ELSE (mul_temp_14(30 DOWNTO 0));

mul_temp_15 <= input_pipeline_phase0_im(6) * coeffphase1_8;
product_phase0_8_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_15(31)
    = '0' AND mul_temp_15(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_15(31) = '1' AND
        mul_temp_15(30) /= '1'
    ELSE (mul_temp_15(30 DOWNTO 0));

mul_temp_16 <= input_pipeline_phase0_re(7) * coeffphase1_9;
product_phase0_9_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_16(31)
    = '0' AND mul_temp_16(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_16(31) = '1' AND
        mul_temp_16(30) /= '1'
    ELSE (mul_temp_16(30 DOWNTO 0));

mul_temp_17 <= input_pipeline_phase0_im(7) * coeffphase1_9;
product_phase0_9_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_17(31)
    = '0' AND mul_temp_17(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_17(31) = '1' AND
        mul_temp_17(30) /= '1'
    ELSE (mul_temp_17(30 DOWNTO 0));

mul_temp_18 <= input_pipeline_phase0_re(8) * coeffphase1_10;
product_phase0_10_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_18
   (31) = '0' AND mul_temp_18(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_18(31) = '1' AND
```

```
        mul_temp_18(30) /= '1'
    ELSE (mul_temp_18(30 DOWNTO 0));

mul_temp_19 <= input_pipeline_phase0_im(8) * coeffphase1_10;
product_phase0_10_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_19
    (31) = '0' AND mul_temp_19(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_19(31) = '1' AND
        mul_temp_19(30) /= '1'
    ELSE (mul_temp_19(30 DOWNTO 0));

mul_temp_20 <= input_pipeline_phase0_re(9) * coeffphase1_11;
product_phase0_11_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_20
    (31) = '0' AND mul_temp_20(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_20(31) = '1' AND
        mul_temp_20(30) /= '1'
    ELSE (mul_temp_20(30 DOWNTO 0));

mul_temp_21 <= input_pipeline_phase0_im(9) * coeffphase1_11;
product_phase0_11_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_21
    (31) = '0' AND mul_temp_21(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_21(31) = '1' AND
        mul_temp_21(30) /= '1'
    ELSE (mul_temp_21(30 DOWNTO 0));

mul_temp_22 <= input_pipeline_phase0_re(10) * coeffphase1_12;
product_phase0_12_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_22
    (31) = '0' AND mul_temp_22(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_22(31) = '1' AND
        mul_temp_22(30) /= '1'
    ELSE (mul_temp_22(30 DOWNTO 0));

mul_temp_23 <= input_pipeline_phase0_im(10) * coeffphase1_12;
product_phase0_12_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_23
```

```
    (31) = '0' AND mul_temp_23(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_23(31) = '1' AND
        mul_temp_23(30) /= '1'
    ELSE (mul_temp_23(30 DOWNTO 0));

mul_temp_24 <= input_pipeline_phase0_re(11) * coeffphase1_13;
product_phase0_13_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_24
    (31) = '0' AND mul_temp_24(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_24(31) = '1' AND
        mul_temp_24(30) /= '1'
    ELSE (mul_temp_24(30 DOWNTO 0));

mul_temp_25 <= input_pipeline_phase0_im(11) * coeffphase1_13;
product_phase0_13_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_25
    (31) = '0' AND mul_temp_25(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_25(31) = '1' AND
        mul_temp_25(30) /= '1'
    ELSE (mul_temp_25(30 DOWNTO 0));

mul_temp_26 <= input_pipeline_phase0_re(12) * coeffphase1_14;
product_phase0_14_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_26
    (31) = '0' AND mul_temp_26(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_26(31) = '1' AND
        mul_temp_26(30) /= '1'
    ELSE (mul_temp_26(30 DOWNTO 0));

mul_temp_27 <= input_pipeline_phase0_im(12) * coeffphase1_14;
product_phase0_14_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_27
    (31) = '0' AND mul_temp_27(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_27(31) = '1' AND
        mul_temp_27(30) /= '1'
    ELSE (mul_temp_27(30 DOWNTO 0));
```

```
mul_temp_28 <= input_pipeline_phase0_re(13) * coeffphase1_15;
product_phase0_15_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_28
    (31) = '0' AND mul_temp_28(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_28(31) = '1' AND
        mul_temp_28(30) /= '1'
    ELSE (mul_temp_28(30 DOWNTO 0));

mul_temp_29 <= input_pipeline_phase0_im(13) * coeffphase1_15;
product_phase0_15_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_29
    (31) = '0' AND mul_temp_29(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_29(31) = '1' AND
        mul_temp_29(30) /= '1'
    ELSE (mul_temp_29(30 DOWNTO 0));

mul_temp_30 <= input_pipeline_phase0_re(14) * coeffphase1_16;
product_phase0_16_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_30
    (31) = '0' AND mul_temp_30(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_30(31) = '1' AND
        mul_temp_30(30) /= '1'
    ELSE (mul_temp_30(30 DOWNTO 0));

mul_temp_31 <= input_pipeline_phase0_im(14) * coeffphase1_16;
product_phase0_16_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_31
    (31) = '0' AND mul_temp_31(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_31(31) = '1' AND
        mul_temp_31(30) /= '1'
    ELSE (mul_temp_31(30 DOWNTO 0));

mul_temp_32 <= input_pipeline_phase0_re(15) * coeffphase1_17;
product_phase0_17_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_32
    (31) = '0' AND mul_temp_32(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_32(31) = '1' AND
        mul_temp_32(30) /= '1'
```

```
    ELSE (mul_temp_32(30 DOWNTO 0));

mul_temp_33 <= input_pipeline_phase0_im(15) * coeffphase1_17;
product_phase0_17_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_33
    (31) = '0' AND mul_temp_33(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_33(31) = '1' AND
        mul_temp_33(30) /= '1'
    ELSE (mul_temp_33(30 DOWNTO 0));

mul_temp_34 <= input_pipeline_phase0_re(16) * coeffphase1_18;
product_phase0_18_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_34
    (31) = '0' AND mul_temp_34(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_34(31) = '1' AND
        mul_temp_34(30) /= '1'
    ELSE (mul_temp_34(30 DOWNTO 0));

mul_temp_35 <= input_pipeline_phase0_im(16) * coeffphase1_18;
product_phase0_18_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_35
    (31) = '0' AND mul_temp_35(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_35(31) = '1' AND
        mul_temp_35(30) /= '1'
    ELSE (mul_temp_35(30 DOWNTO 0));

mul_temp_36 <= input_pipeline_phase0_re(17) * coeffphase1_19;
product_phase0_19_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_36
    (31) = '0' AND mul_temp_36(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_36(31) = '1' AND
        mul_temp_36(30) /= '1'
    ELSE (mul_temp_36(30 DOWNTO 0));

mul_temp_37 <= input_pipeline_phase0_im(17) * coeffphase1_19;
product_phase0_19_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_37
    (31) = '0' AND mul_temp_37(30) /= '0'
```

```
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_37(31) = '1' AND
        mul_temp_37(30) /= '1'
    ELSE (mul_temp_37(30 DOWNTO 0));

mul_temp_38 <= input_pipeline_phase0_re(18) * coeffphase1_20;
product_phase0_20_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_38
    (31) = '0' AND mul_temp_38(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_38(31) = '1' AND
        mul_temp_38(30) /= '1'
    ELSE (mul_temp_38(30 DOWNTO 0));

mul_temp_39 <= input_pipeline_phase0_im(18) * coeffphase1_20;
product_phase0_20_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_39
    (31) = '0' AND mul_temp_39(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_39(31) = '1' AND
        mul_temp_39(30) /= '1'
    ELSE (mul_temp_39(30 DOWNTO 0));

mul_temp_40 <= input_pipeline_phase0_re(19) * coeffphase1_21;
product_phase0_21_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_40
    (31) = '0' AND mul_temp_40(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_40(31) = '1' AND
        mul_temp_40(30) /= '1'
    ELSE (mul_temp_40(30 DOWNTO 0));

mul_temp_41 <= input_pipeline_phase0_im(19) * coeffphase1_21;
product_phase0_21_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_41
    (31) = '0' AND mul_temp_41(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_41(31) = '1' AND
        mul_temp_41(30) /= '1'
    ELSE (mul_temp_41(30 DOWNTO 0));

mul_temp_42 <= input_pipeline_phase1_re(0) * coeffphase2_1;
```

```
product_phase1_1_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_42(31)
    = '0' AND mul_temp_42(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_42(31) = '1' AND
        mul_temp_42(30) /= '1'
    ELSE (mul_temp_42(30 DOWNTO 0));

mul_temp_43 <= input_pipeline_phase1_im(0) * coeffphase2_1;
product_phase1_1_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_43(31)
    = '0' AND mul_temp_43(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_43(31) = '1' AND
        mul_temp_43(30) /= '1'
    ELSE (mul_temp_43(30 DOWNTO 0));

mul_temp_44 <= input_pipeline_phase1_re(1) * coeffphase2_2;
product_phase1_2_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_44(31)
    = '0' AND mul_temp_44(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_44(31) = '1' AND
        mul_temp_44(30) /= '1'
    ELSE (mul_temp_44(30 DOWNTO 0));

mul_temp_45 <= input_pipeline_phase1_im(1) * coeffphase2_2;
product_phase1_2_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_45(31)
    = '0' AND mul_temp_45(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_45(31) = '1' AND
        mul_temp_45(30) /= '1'
    ELSE (mul_temp_45(30 DOWNTO 0));

mul_temp_46 <= input_pipeline_phase1_re(2) * coeffphase2_3;
product_phase1_3_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_46(31)
    = '0' AND mul_temp_46(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_46(31) = '1' AND
        mul_temp_46(30) /= '1'
    ELSE (mul_temp_46(30 DOWNTO 0));
```

```
mul_temp_47 <= input_pipeline_phase1_im(2) * coeffphase2_3;
product_phase1_3_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_47(31)
    = '0' AND mul_temp_47(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_47(31) = '1' AND
        mul_temp_47(30) /= '1'
    ELSE (mul_temp_47(30 DOWNTO 0));

mul_temp_48 <= input_pipeline_phase1_re(3) * coeffphase2_4;
product_phase1_4_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_48(31)
    = '0' AND mul_temp_48(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_48(31) = '1' AND
        mul_temp_48(30) /= '1'
    ELSE (mul_temp_48(30 DOWNTO 0));

mul_temp_49 <= input_pipeline_phase1_im(3) * coeffphase2_4;
product_phase1_4_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_49(31)
    = '0' AND mul_temp_49(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_49(31) = '1' AND
        mul_temp_49(30) /= '1'
    ELSE (mul_temp_49(30 DOWNTO 0));

mul_temp_50 <= input_pipeline_phase1_re(4) * coeffphase2_5;
product_phase1_5_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_50(31)
    = '0' AND mul_temp_50(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_50(31) = '1' AND
        mul_temp_50(30) /= '1'
    ELSE (mul_temp_50(30 DOWNTO 0));

mul_temp_51 <= input_pipeline_phase1_im(4) * coeffphase2_5;
product_phase1_5_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_51(31)
    = '0' AND mul_temp_51(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_51(31) = '1' AND
```

```
        mul_temp_51(30) /= '1'
    ELSE (mul_temp_51(30 DOWNTO 0));

mul_temp_52 <= input_pipeline_phase1_re(5) * coeffphase2_6;
product_phase1_6_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_52(31)
    = '0' AND mul_temp_52(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_52(31) = '1' AND
        mul_temp_52(30) /= '1'
    ELSE (mul_temp_52(30 DOWNTO 0));

mul_temp_53 <= input_pipeline_phase1_im(5) * coeffphase2_6;
product_phase1_6_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_53(31)
    = '0' AND mul_temp_53(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_53(31) = '1' AND
        mul_temp_53(30) /= '1'
    ELSE (mul_temp_53(30 DOWNTO 0));

mul_temp_54 <= input_pipeline_phase1_re(6) * coeffphase2_7;
product_phase1_7_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_54(31)
    = '0' AND mul_temp_54(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_54(31) = '1' AND
        mul_temp_54(30) /= '1'
    ELSE (mul_temp_54(30 DOWNTO 0));

mul_temp_55 <= input_pipeline_phase1_im(6) * coeffphase2_7;
product_phase1_7_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_55(31)
    = '0' AND mul_temp_55(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_55(31) = '1' AND
        mul_temp_55(30) /= '1'
    ELSE (mul_temp_55(30 DOWNTO 0));

mul_temp_56 <= input_pipeline_phase1_re(7) * coeffphase2_8;
product_phase1_8_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_56(31)
```

```
    = '0' AND mul_temp_56(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_56(31) = '1' AND
        mul_temp_56(30) /= '1'
    ELSE (mul_temp_56(30 DOWNTO 0));

mul_temp_57 <= input_pipeline_phase1_im(7) * coeffphase2_8;
product_phase1_8_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_57(31)
    = '0' AND mul_temp_57(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_57(31) = '1' AND
        mul_temp_57(30) /= '1'
    ELSE (mul_temp_57(30 DOWNTO 0));

mul_temp_58 <= input_pipeline_phase1_re(8) * coeffphase2_9;
product_phase1_9_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_58(31)
    = '0' AND mul_temp_58(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_58(31) = '1' AND
        mul_temp_58(30) /= '1'
    ELSE (mul_temp_58(30 DOWNTO 0));

mul_temp_59 <= input_pipeline_phase1_im(8) * coeffphase2_9;
product_phase1_9_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_59(31)
    = '0' AND mul_temp_59(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_59(31) = '1' AND
        mul_temp_59(30) /= '1'
    ELSE (mul_temp_59(30 DOWNTO 0));

mul_temp_60 <= input_pipeline_phase1_re(9) * coeffphase2_10;
product_phase1_10_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_60
   (31) = '0' AND mul_temp_60(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_60(31) = '1' AND
        mul_temp_60(30) /= '1'
    ELSE (mul_temp_60(30 DOWNTO 0));
```

```
mul_temp_61 <= input_pipeline_phase1_im(9) * coeffphase2_10;
product_phase1_10_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_61
    (31) = '0' AND mul_temp_61(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_61(31) = '1' AND
        mul_temp_61(30) /= '1'
    ELSE (mul_temp_61(30 DOWNTO 0));

mul_temp_62 <= input_pipeline_phase1_re(10) * coeffphase2_11;
product_phase1_11_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_62
    (31) = '0' AND mul_temp_62(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_62(31) = '1' AND
        mul_temp_62(30) /= '1'
    ELSE (mul_temp_62(30 DOWNTO 0));

mul_temp_63 <= input_pipeline_phase1_im(10) * coeffphase2_11;
product_phase1_11_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_63
    (31) = '0' AND mul_temp_63(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_63(31) = '1' AND
        mul_temp_63(30) /= '1'
    ELSE (mul_temp_63(30 DOWNTO 0));

mul_temp_64 <= input_pipeline_phase1_re(11) * coeffphase2_12;
product_phase1_12_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_64
    (31) = '0' AND mul_temp_64(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_64(31) = '1' AND
        mul_temp_64(30) /= '1'
    ELSE (mul_temp_64(30 DOWNTO 0));

mul_temp_65 <= input_pipeline_phase1_im(11) * coeffphase2_12;
product_phase1_12_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_65
    (31) = '0' AND mul_temp_65(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_65(31) = '1' AND
        mul_temp_65(30) /= '1'
```

```
    ELSE (mul_temp_65(30 DOWNTO 0));

mul_temp_66 <= input_pipeline_phase1_re(12) * coeffphase2_13;
product_phase1_13_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_66
    (31) = '0' AND mul_temp_66(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_66(31) = '1' AND
        mul_temp_66(30) /= '1'
    ELSE (mul_temp_66(30 DOWNTO 0));

mul_temp_67 <= input_pipeline_phase1_im(12) * coeffphase2_13;
product_phase1_13_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_67
    (31) = '0' AND mul_temp_67(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_67(31) = '1' AND
        mul_temp_67(30) /= '1'
    ELSE (mul_temp_67(30 DOWNTO 0));

mul_temp_68 <= input_pipeline_phase1_re(13) * coeffphase2_14;
product_phase1_14_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_68
    (31) = '0' AND mul_temp_68(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_68(31) = '1' AND
        mul_temp_68(30) /= '1'
    ELSE (mul_temp_68(30 DOWNTO 0));

mul_temp_69 <= input_pipeline_phase1_im(13) * coeffphase2_14;
product_phase1_14_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_69
    (31) = '0' AND mul_temp_69(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_69(31) = '1' AND
        mul_temp_69(30) /= '1'
    ELSE (mul_temp_69(30 DOWNTO 0));

mul_temp_70 <= input_pipeline_phase1_re(14) * coeffphase2_15;
product_phase1_15_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_70
    (31) = '0' AND mul_temp_70(30) /= '0'
```

```
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_70(31) = '1' AND
        mul_temp_70(30) /= '1'
    ELSE (mul_temp_70(30 DOWNTO 0));

mul_temp_71 <= input_pipeline_phase1_im(14) * coeffphase2_15;
product_phase1_15_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_71
    (31) = '0' AND mul_temp_71(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_71(31) = '1' AND
        mul_temp_71(30) /= '1'
    ELSE (mul_temp_71(30 DOWNTO 0));

mul_temp_72 <= input_pipeline_phase1_re(15) * coeffphase2_16;
product_phase1_16_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_72
    (31) = '0' AND mul_temp_72(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_72(31) = '1' AND
        mul_temp_72(30) /= '1'
    ELSE (mul_temp_72(30 DOWNTO 0));

mul_temp_73 <= input_pipeline_phase1_im(15) * coeffphase2_16;
product_phase1_16_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_73
    (31) = '0' AND mul_temp_73(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_73(31) = '1' AND
        mul_temp_73(30) /= '1'
    ELSE (mul_temp_73(30 DOWNTO 0));

mul_temp_74 <= input_pipeline_phase1_re(16) * coeffphase2_17;
product_phase1_17_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_74
    (31) = '0' AND mul_temp_74(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_74(31) = '1' AND
        mul_temp_74(30) /= '1'
    ELSE (mul_temp_74(30 DOWNTO 0));

mul_temp_75 <= input_pipeline_phase1_im(16) * coeffphase2_17;
```

```
product_phase1_17_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_75
    (31) = '0' AND mul_temp_75(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_75(31) = '1' AND
        mul_temp_75(30) /= '1'
    ELSE (mul_temp_75(30 DOWNTO 0));

mul_temp_76 <= input_pipeline_phase1_re(17) * coeffphase2_18;
product_phase1_18_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_76
    (31) = '0' AND mul_temp_76(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_76(31) = '1' AND
        mul_temp_76(30) /= '1'
    ELSE (mul_temp_76(30 DOWNTO 0));

mul_temp_77 <= input_pipeline_phase1_im(17) * coeffphase2_18;
product_phase1_18_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_77
    (31) = '0' AND mul_temp_77(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_77(31) = '1' AND
        mul_temp_77(30) /= '1'
    ELSE (mul_temp_77(30 DOWNTO 0));

mul_temp_78 <= input_pipeline_phase1_re(18) * coeffphase2_19;
product_phase1_19_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_78
    (31) = '0' AND mul_temp_78(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_78(31) = '1' AND
        mul_temp_78(30) /= '1'
    ELSE (mul_temp_78(30 DOWNTO 0));

mul_temp_79 <= input_pipeline_phase1_im(18) * coeffphase2_19;
product_phase1_19_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_79
    (31) = '0' AND mul_temp_79(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_79(31) = '1' AND
        mul_temp_79(30) /= '1'
    ELSE (mul_temp_79(30 DOWNTO 0));
```

```
mul_temp_80 <= input_pipeline_phase1_re(19) * coeffphase2_20;
product_phase1_20_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_80
    (31) = '0' AND mul_temp_80(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_80(31) = '1' AND
        mul_temp_80(30) /= '1'
    ELSE (mul_temp_80(30 DOWNTO 0));

mul_temp_81 <= input_pipeline_phase1_im(19) * coeffphase2_20;
product_phase1_20_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_81
    (31) = '0' AND mul_temp_81(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_81(31) = '1' AND
        mul_temp_81(30) /= '1'
    ELSE (mul_temp_81(30 DOWNTO 0));

mul_temp_82 <= input_pipeline_phase2_re(0) * coeffphase3_1;
product_phase2_1_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_82(31)
    = '0' AND mul_temp_82(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_82(31) = '1' AND
        mul_temp_82(30) /= '1'
    ELSE (mul_temp_82(30 DOWNTO 0));

mul_temp_83 <= input_pipeline_phase2_im(0) * coeffphase3_1;
product_phase2_1_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_83(31)
    = '0' AND mul_temp_83(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_83(31) = '1' AND
        mul_temp_83(30) /= '1'
    ELSE (mul_temp_83(30 DOWNTO 0));

mul_temp_84 <= input_pipeline_phase2_re(1) * coeffphase3_2;
product_phase2_2_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_84(31)
    = '0' AND mul_temp_84(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_84(31) = '1' AND
```

```
        mul_temp_84(30) /= '1'
    ELSE (mul_temp_84(30 DOWNTO 0));

mul_temp_85 <= input_pipeline_phase2_im(1) * coeffphase3_2;
product_phase2_2_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_85(31)
    = '0' AND mul_temp_85(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_85(31) = '1' AND
        mul_temp_85(30) /= '1'
    ELSE (mul_temp_85(30 DOWNTO 0));

mul_temp_86 <= input_pipeline_phase2_re(2) * coeffphase3_3;
product_phase2_3_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_86(31)
    = '0' AND mul_temp_86(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_86(31) = '1' AND
        mul_temp_86(30) /= '1'
    ELSE (mul_temp_86(30 DOWNTO 0));

mul_temp_87 <= input_pipeline_phase2_im(2) * coeffphase3_3;
product_phase2_3_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_87(31)
    = '0' AND mul_temp_87(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_87(31) = '1' AND
        mul_temp_87(30) /= '1'
    ELSE (mul_temp_87(30 DOWNTO 0));

mul_temp_88 <= input_pipeline_phase2_re(3) * coeffphase3_4;
product_phase2_4_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_88(31)
    = '0' AND mul_temp_88(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_88(31) = '1' AND
        mul_temp_88(30) /= '1'
    ELSE (mul_temp_88(30 DOWNTO 0));

mul_temp_89 <= input_pipeline_phase2_im(3) * coeffphase3_4;
product_phase2_4_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_89(31)
```

```
    = '0' AND mul_temp_89(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_89(31) = '1' AND
        mul_temp_89(30) /= '1'
    ELSE (mul_temp_89(30 DOWNTO 0));

mul_temp_90 <= input_pipeline_phase2_re(4) * coeffphase3_5;
product_phase2_5_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_90(31)
    = '0' AND mul_temp_90(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_90(31) = '1' AND
        mul_temp_90(30) /= '1'
    ELSE (mul_temp_90(30 DOWNTO 0));

mul_temp_91 <= input_pipeline_phase2_im(4) * coeffphase3_5;
product_phase2_5_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_91(31)
    = '0' AND mul_temp_91(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_91(31) = '1' AND
        mul_temp_91(30) /= '1'
    ELSE (mul_temp_91(30 DOWNTO 0));

mul_temp_92 <= input_pipeline_phase2_re(5) * coeffphase3_6;
product_phase2_6_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_92(31)
    = '0' AND mul_temp_92(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_92(31) = '1' AND
        mul_temp_92(30) /= '1'
    ELSE (mul_temp_92(30 DOWNTO 0));

mul_temp_93 <= input_pipeline_phase2_im(5) * coeffphase3_6;
product_phase2_6_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_93(31)
    = '0' AND mul_temp_93(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_93(31) = '1' AND
        mul_temp_93(30) /= '1'
    ELSE (mul_temp_93(30 DOWNTO 0));
```

```
mul_temp_94 <= input_pipeline_phase2_re(6) * coeffphase3_7;
product_phase2_7_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_94(31)
    = '0' AND mul_temp_94(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_94(31) = '1' AND
        mul_temp_94(30) /= '1'
    ELSE (mul_temp_94(30 DOWNTO 0));

mul_temp_95 <= input_pipeline_phase2_im(6) * coeffphase3_7;
product_phase2_7_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_95(31)
    = '0' AND mul_temp_95(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_95(31) = '1' AND
        mul_temp_95(30) /= '1'
    ELSE (mul_temp_95(30 DOWNTO 0));

mul_temp_96 <= input_pipeline_phase2_re(7) * coeffphase3_8;
product_phase2_8_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_96(31)
    = '0' AND mul_temp_96(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_96(31) = '1' AND
        mul_temp_96(30) /= '1'
    ELSE (mul_temp_96(30 DOWNTO 0));

mul_temp_97 <= input_pipeline_phase2_im(7) * coeffphase3_8;
product_phase2_8_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_97(31)
    = '0' AND mul_temp_97(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_97(31) = '1' AND
        mul_temp_97(30) /= '1'
    ELSE (mul_temp_97(30 DOWNTO 0));

mul_temp_98 <= input_pipeline_phase2_re(8) * coeffphase3_9;
product_phase2_9_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_98(31)
    = '0' AND mul_temp_98(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_98(31) = '1' AND
        mul_temp_98(30) /= '1'
```

```
    ELSE (mul_temp_98(30 DOWNTO 0));

mul_temp_99 <= input_pipeline_phase2_im(8) * coeffphase3_9;
product_phase2_9_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_99(31)
    = '0' AND mul_temp_99(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_99(31) = '1' AND
        mul_temp_99(30) /= '1'
    ELSE (mul_temp_99(30 DOWNTO 0));

mul_temp_100 <= input_pipeline_phase2_re(9) * coeffphase3_10;
product_phase2_10_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_100
    (31) = '0' AND mul_temp_100(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_100(31) = '1' AND
        mul_temp_100(30) /= '1'
    ELSE (mul_temp_100(30 DOWNTO 0));

mul_temp_101 <= input_pipeline_phase2_im(9) * coeffphase3_10;
product_phase2_10_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_101
    (31) = '0' AND mul_temp_101(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_101(31) = '1' AND
        mul_temp_101(30) /= '1'
    ELSE (mul_temp_101(30 DOWNTO 0));

mul_temp_102 <= input_pipeline_phase2_re(10) * coeffphase3_11;
product_phase2_11_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_102
    (31) = '0' AND mul_temp_102(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_102(31) = '1' AND
        mul_temp_102(30) /= '1'
    ELSE (mul_temp_102(30 DOWNTO 0));

mul_temp_103 <= input_pipeline_phase2_im(10) * coeffphase3_11;
product_phase2_11_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_103
    (31) = '0' AND mul_temp_103(30) /= '0'
```

```
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_103(31) = '1' AND
        mul_temp_103(30) /= '1'
    ELSE (mul_temp_103(30 DOWNTO 0));

mul_temp_104 <= input_pipeline_phase2_re(11) * coeffphase3_12;
product_phase2_12_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_104
    (31) = '0' AND mul_temp_104(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_104(31) = '1' AND
        mul_temp_104(30) /= '1'
    ELSE (mul_temp_104(30 DOWNTO 0));

mul_temp_105 <= input_pipeline_phase2_im(11) * coeffphase3_12;
product_phase2_12_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_105
    (31) = '0' AND mul_temp_105(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_105(31) = '1' AND
        mul_temp_105(30) /= '1'
    ELSE (mul_temp_105(30 DOWNTO 0));

mul_temp_106 <= input_pipeline_phase2_re(12) * coeffphase3_13;
product_phase2_13_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_106
    (31) = '0' AND mul_temp_106(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_106(31) = '1' AND
        mul_temp_106(30) /= '1'
    ELSE (mul_temp_106(30 DOWNTO 0));

mul_temp_107 <= input_pipeline_phase2_im(12) * coeffphase3_13;
product_phase2_13_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_107
    (31) = '0' AND mul_temp_107(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_107(31) = '1' AND
        mul_temp_107(30) /= '1'
    ELSE (mul_temp_107(30 DOWNTO 0));

mul_temp_108 <= input_pipeline_phase2_re(13) * coeffphase3_14;
```

```
product_phase2_14_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_108
    (31) = '0' AND mul_temp_108(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_108(31) = '1' AND
        mul_temp_108(30) /= '1'
    ELSE (mul_temp_108(30 DOWNTO 0));

mul_temp_109 <= input_pipeline_phase2_im(13) * coeffphase3_14;
product_phase2_14_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_109
    (31) = '0' AND mul_temp_109(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_109(31) = '1' AND
        mul_temp_109(30) /= '1'
    ELSE (mul_temp_109(30 DOWNTO 0));

mul_temp_110 <= input_pipeline_phase2_re(14) * coeffphase3_15;
product_phase2_15_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_110
    (31) = '0' AND mul_temp_110(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_110(31) = '1' AND
        mul_temp_110(30) /= '1'
    ELSE (mul_temp_110(30 DOWNTO 0));

mul_temp_111 <= input_pipeline_phase2_im(14) * coeffphase3_15;
product_phase2_15_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_111
    (31) = '0' AND mul_temp_111(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_111(31) = '1' AND
        mul_temp_111(30) /= '1'
    ELSE (mul_temp_111(30 DOWNTO 0));

mul_temp_112 <= input_pipeline_phase2_re(15) * coeffphase3_16;
product_phase2_16_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_112
    (31) = '0' AND mul_temp_112(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_112(31) = '1' AND
        mul_temp_112(30) /= '1'
    ELSE (mul_temp_112(30 DOWNTO 0));
```

```
mul_temp_113 <= input_pipeline_phase2_im(15) * coeffphase3_16;
product_phase2_16_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_113
    (31) = '0' AND mul_temp_113(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_113(31) = '1' AND
        mul_temp_113(30) /= '1'
    ELSE (mul_temp_113(30 DOWNTO 0));

mul_temp_114 <= input_pipeline_phase2_re(16) * coeffphase3_17;
product_phase2_17_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_114
    (31) = '0' AND mul_temp_114(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_114(31) = '1' AND
        mul_temp_114(30) /= '1'
    ELSE (mul_temp_114(30 DOWNTO 0));

mul_temp_115 <= input_pipeline_phase2_im(16) * coeffphase3_17;
product_phase2_17_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_115
    (31) = '0' AND mul_temp_115(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_115(31) = '1' AND
        mul_temp_115(30) /= '1'
    ELSE (mul_temp_115(30 DOWNTO 0));

mul_temp_116 <= input_pipeline_phase2_re(17) * coeffphase3_18;
product_phase2_18_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_116
    (31) = '0' AND mul_temp_116(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_116(31) = '1' AND
        mul_temp_116(30) /= '1'
    ELSE (mul_temp_116(30 DOWNTO 0));

mul_temp_117 <= input_pipeline_phase2_im(17) * coeffphase3_18;
product_phase2_18_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_117
    (31) = '0' AND mul_temp_117(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_117(31) = '1' AND
```

```
        mul_temp_117(30) /= '1'
    ELSE (mul_temp_117(30 DOWNTO 0));

mul_temp_118 <= input_pipeline_phase2_re(18) * coeffphase3_19;
product_phase2_19_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_118
    (31) = '0' AND mul_temp_118(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_118(31) = '1' AND
        mul_temp_118(30) /= '1'
    ELSE (mul_temp_118(30 DOWNTO 0));

mul_temp_119 <= input_pipeline_phase2_im(18) * coeffphase3_19;
product_phase2_19_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_119
    (31) = '0' AND mul_temp_119(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_119(31) = '1' AND
        mul_temp_119(30) /= '1'
    ELSE (mul_temp_119(30 DOWNTO 0));

mul_temp_120 <= input_pipeline_phase2_re(19) * coeffphase3_20;
product_phase2_20_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_120
    (31) = '0' AND mul_temp_120(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_120(31) = '1' AND
        mul_temp_120(30) /= '1'
    ELSE (mul_temp_120(30 DOWNTO 0));

mul_temp_121 <= input_pipeline_phase2_im(19) * coeffphase3_20;
product_phase2_20_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_121
    (31) = '0' AND mul_temp_121(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_121(31) = '1' AND
        mul_temp_121(30) /= '1'
    ELSE (mul_temp_121(30 DOWNTO 0));

mul_temp_122 <= input_pipeline_phase3_re(0) * coeffphase4_1;
product_phase3_1_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_122
```

```
    (31) = '0' AND mul_temp_122(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_122(31) = '1' AND
        mul_temp_122(30) /= '1'
    ELSE (mul_temp_122(30 DOWNTO 0));

mul_temp_123 <= input_pipeline_phase3_im(0) * coeffphase4_1;
product_phase3_1_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_123
    (31) = '0' AND mul_temp_123(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_123(31) = '1' AND
        mul_temp_123(30) /= '1'
    ELSE (mul_temp_123(30 DOWNTO 0));

mul_temp_124 <= input_pipeline_phase3_re(1) * coeffphase4_2;
product_phase3_2_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_124
    (31) = '0' AND mul_temp_124(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_124(31) = '1' AND
        mul_temp_124(30) /= '1'
    ELSE (mul_temp_124(30 DOWNTO 0));

mul_temp_125 <= input_pipeline_phase3_im(1) * coeffphase4_2;
product_phase3_2_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_125
    (31) = '0' AND mul_temp_125(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_125(31) = '1' AND
        mul_temp_125(30) /= '1'
    ELSE (mul_temp_125(30 DOWNTO 0));

mul_temp_126 <= input_pipeline_phase3_re(2) * coeffphase4_3;
product_phase3_3_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_126
    (31) = '0' AND mul_temp_126(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_126(31) = '1' AND
        mul_temp_126(30) /= '1'
    ELSE (mul_temp_126(30 DOWNTO 0));
```

```
mul_temp_127 <= input_pipeline_phase3_im(2) * coeffphase4_3;
product_phase3_3_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_127
    (31) = '0' AND mul_temp_127(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_127(31) = '1' AND
        mul_temp_127(30) /= '1'
    ELSE (mul_temp_127(30 DOWNTO 0));

mul_temp_128 <= input_pipeline_phase3_re(3) * coeffphase4_4;
product_phase3_4_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_128
    (31) = '0' AND mul_temp_128(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_128(31) = '1' AND
        mul_temp_128(30) /= '1'
    ELSE (mul_temp_128(30 DOWNTO 0));

mul_temp_129 <= input_pipeline_phase3_im(3) * coeffphase4_4;
product_phase3_4_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_129
    (31) = '0' AND mul_temp_129(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_129(31) = '1' AND
        mul_temp_129(30) /= '1'
    ELSE (mul_temp_129(30 DOWNTO 0));

mul_temp_130 <= input_pipeline_phase3_re(4) * coeffphase4_5;
product_phase3_5_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_130
    (31) = '0' AND mul_temp_130(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_130(31) = '1' AND
        mul_temp_130(30) /= '1'
    ELSE (mul_temp_130(30 DOWNTO 0));

mul_temp_131 <= input_pipeline_phase3_im(4) * coeffphase4_5;
product_phase3_5_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_131
    (31) = '0' AND mul_temp_131(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_131(31) = '1' AND
        mul_temp_131(30) /= '1'
```

```
    ELSE (mul_temp_131(30 DOWNTO 0));

mul_temp_132 <= input_pipeline_phase3_re(5) * coeffphase4_6;
product_phase3_6_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_132
    (31) = '0' AND mul_temp_132(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_132(31) = '1' AND
        mul_temp_132(30) /= '1'
    ELSE (mul_temp_132(30 DOWNTO 0));

mul_temp_133 <= input_pipeline_phase3_im(5) * coeffphase4_6;
product_phase3_6_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_133
    (31) = '0' AND mul_temp_133(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_133(31) = '1' AND
        mul_temp_133(30) /= '1'
    ELSE (mul_temp_133(30 DOWNTO 0));

mul_temp_134 <= input_pipeline_phase3_re(6) * coeffphase4_7;
product_phase3_7_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_134
    (31) = '0' AND mul_temp_134(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_134(31) = '1' AND
        mul_temp_134(30) /= '1'
    ELSE (mul_temp_134(30 DOWNTO 0));

mul_temp_135 <= input_pipeline_phase3_im(6) * coeffphase4_7;
product_phase3_7_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_135
    (31) = '0' AND mul_temp_135(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_135(31) = '1' AND
        mul_temp_135(30) /= '1'
    ELSE (mul_temp_135(30 DOWNTO 0));

mul_temp_136 <= input_pipeline_phase3_re(7) * coeffphase4_8;
product_phase3_8_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_136
    (31) = '0' AND mul_temp_136(30) /= '0'
```

```
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_136(31) = '1' AND
        mul_temp_136(30) /= '1'
    ELSE (mul_temp_136(30 DOWNTO 0));

mul_temp_137 <= input_pipeline_phase3_im(7) * coeffphase4_8;
product_phase3_8_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_137
    (31) = '0' AND mul_temp_137(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_137(31) = '1' AND
        mul_temp_137(30) /= '1'
    ELSE (mul_temp_137(30 DOWNTO 0));

mul_temp_138 <= input_pipeline_phase3_re(8) * coeffphase4_9;
product_phase3_9_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_138
    (31) = '0' AND mul_temp_138(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_138(31) = '1' AND
        mul_temp_138(30) /= '1'
    ELSE (mul_temp_138(30 DOWNTO 0));

mul_temp_139 <= input_pipeline_phase3_im(8) * coeffphase4_9;
product_phase3_9_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_139
    (31) = '0' AND mul_temp_139(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_139(31) = '1' AND
        mul_temp_139(30) /= '1'
    ELSE (mul_temp_139(30 DOWNTO 0));

mul_temp_140 <= input_pipeline_phase3_re(9) * coeffphase4_10;
product_phase3_10_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_140
    (31) = '0' AND mul_temp_140(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_140(31) = '1' AND
        mul_temp_140(30) /= '1'
    ELSE (mul_temp_140(30 DOWNTO 0));

mul_temp_141 <= input_pipeline_phase3_im(9) * coeffphase4_10;
```

```
product_phase3_10_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_141
    (31) = '0' AND mul_temp_141(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_141(31) = '1' AND
        mul_temp_141(30) /= '1'
    ELSE (mul_temp_141(30 DOWNTO 0));

mul_temp_142 <= input_pipeline_phase3_re(10) * coeffphase4_11;
product_phase3_11_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_142
    (31) = '0' AND mul_temp_142(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_142(31) = '1' AND
        mul_temp_142(30) /= '1'
    ELSE (mul_temp_142(30 DOWNTO 0));

mul_temp_143 <= input_pipeline_phase3_im(10) * coeffphase4_11;
product_phase3_11_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_143
    (31) = '0' AND mul_temp_143(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_143(31) = '1' AND
        mul_temp_143(30) /= '1'
    ELSE (mul_temp_143(30 DOWNTO 0));

mul_temp_144 <= input_pipeline_phase3_re(11) * coeffphase4_12;
product_phase3_12_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_144
    (31) = '0' AND mul_temp_144(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_144(31) = '1' AND
        mul_temp_144(30) /= '1'
    ELSE (mul_temp_144(30 DOWNTO 0));

mul_temp_145 <= input_pipeline_phase3_im(11) * coeffphase4_12;
product_phase3_12_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_145
    (31) = '0' AND mul_temp_145(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_145(31) = '1' AND
        mul_temp_145(30) /= '1'
    ELSE (mul_temp_145(30 DOWNTO 0));
```

```
mul_temp_146 <= input_pipeline_phase3_re(12) * coeffphase4_13;
product_phase3_13_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_146
    (31) = '0' AND mul_temp_146(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_146(31) = '1' AND
        mul_temp_146(30) /= '1'
    ELSE (mul_temp_146(30 DOWNTO 0));

mul_temp_147 <= input_pipeline_phase3_im(12) * coeffphase4_13;
product_phase3_13_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_147
    (31) = '0' AND mul_temp_147(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_147(31) = '1' AND
        mul_temp_147(30) /= '1'
    ELSE (mul_temp_147(30 DOWNTO 0));

mul_temp_148 <= input_pipeline_phase3_re(13) * coeffphase4_14;
product_phase3_14_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_148
    (31) = '0' AND mul_temp_148(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_148(31) = '1' AND
        mul_temp_148(30) /= '1'
    ELSE (mul_temp_148(30 DOWNTO 0));

mul_temp_149 <= input_pipeline_phase3_im(13) * coeffphase4_14;
product_phase3_14_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_149
    (31) = '0' AND mul_temp_149(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_149(31) = '1' AND
        mul_temp_149(30) /= '1'
    ELSE (mul_temp_149(30 DOWNTO 0));

mul_temp_150 <= input_pipeline_phase3_re(14) * coeffphase4_15;
product_phase3_15_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_150
    (31) = '0' AND mul_temp_150(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_150(31) = '1' AND
```

```
        mul_temp_150(30) /= '1'
    ELSE (mul_temp_150(30 DOWNTO 0));

mul_temp_151 <= input_pipeline_phase3_im(14) * coeffphase4_15;
product_phase3_15_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_151
    (31) = '0' AND mul_temp_151(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_151(31) = '1' AND
        mul_temp_151(30) /= '1'
    ELSE (mul_temp_151(30 DOWNTO 0));

mul_temp_152 <= input_pipeline_phase3_re(15) * coeffphase4_16;
product_phase3_16_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_152
    (31) = '0' AND mul_temp_152(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_152(31) = '1' AND
        mul_temp_152(30) /= '1'
    ELSE (mul_temp_152(30 DOWNTO 0));

mul_temp_153 <= input_pipeline_phase3_im(15) * coeffphase4_16;
product_phase3_16_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_153
    (31) = '0' AND mul_temp_153(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_153(31) = '1' AND
        mul_temp_153(30) /= '1'
    ELSE (mul_temp_153(30 DOWNTO 0));

mul_temp_154 <= input_pipeline_phase3_re(16) * coeffphase4_17;
product_phase3_17_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_154
    (31) = '0' AND mul_temp_154(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_154(31) = '1' AND
        mul_temp_154(30) /= '1'
    ELSE (mul_temp_154(30 DOWNTO 0));

mul_temp_155 <= input_pipeline_phase3_im(16) * coeffphase4_17;
product_phase3_17_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_155
```

```
    (31) = '0' AND mul_temp_155(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_155(31) = '1' AND
        mul_temp_155(30) /= '1'
    ELSE (mul_temp_155(30 DOWNTO 0));

mul_temp_156 <= input_pipeline_phase3_re(17) * coeffphase4_18;
product_phase3_18_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_156
    (31) = '0' AND mul_temp_156(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_156(31) = '1' AND
        mul_temp_156(30) /= '1'
    ELSE (mul_temp_156(30 DOWNTO 0));

mul_temp_157 <= input_pipeline_phase3_im(17) * coeffphase4_18;
product_phase3_18_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_157
    (31) = '0' AND mul_temp_157(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_157(31) = '1' AND
        mul_temp_157(30) /= '1'
    ELSE (mul_temp_157(30 DOWNTO 0));

mul_temp_158 <= input_pipeline_phase3_re(18) * coeffphase4_19;
product_phase3_19_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_158
    (31) = '0' AND mul_temp_158(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_158(31) = '1' AND
        mul_temp_158(30) /= '1'
    ELSE (mul_temp_158(30 DOWNTO 0));

mul_temp_159 <= input_pipeline_phase3_im(18) * coeffphase4_19;
product_phase3_19_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_159
    (31) = '0' AND mul_temp_159(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_159(31) = '1' AND
        mul_temp_159(30) /= '1'
    ELSE (mul_temp_159(30 DOWNTO 0));
```

```
mul_temp_160 <= input_pipeline_phase3_re(19) * coeffphase4_20;
product_phase3_20_re <= (30 => '0', OTHERS => '1') WHEN mul_temp_160
    (31) = '0' AND mul_temp_160(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_160(31) = '1' AND
        mul_temp_160(30) /= '1'
    ELSE (mul_temp_160(30 DOWNTO 0));

mul_temp_161 <= input_pipeline_phase3_im(19) * coeffphase4_20;
product_phase3_20_im <= (30 => '0', OTHERS => '1') WHEN mul_temp_161
    (31) = '0' AND mul_temp_161(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_temp_161(31) = '1' AND
        mul_temp_161(30) /= '1'
    ELSE (mul_temp_161(30 DOWNTO 0));

product_pipeline_process3 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    product_pipeline_phase0_1_re <= (OTHERS => '0');
    product_pipeline_phase0_1_im <= (OTHERS => '0');
    product_pipeline_phase1_1_re <= (OTHERS => '0');
    product_pipeline_phase1_1_im <= (OTHERS => '0');
    product_pipeline_phase2_1_re <= (OTHERS => '0');
    product_pipeline_phase2_1_im <= (OTHERS => '0');
    product_pipeline_phase3_1_re <= (OTHERS => '0');
    product_pipeline_phase3_1_im <= (OTHERS => '0');
    product_pipeline_phase0_2_re <= (OTHERS => '0');
    product_pipeline_phase0_2_im <= (OTHERS => '0');
    product_pipeline_phase1_2_re <= (OTHERS => '0');
    product_pipeline_phase1_2_im <= (OTHERS => '0');
    product_pipeline_phase2_2_re <= (OTHERS => '0');
    product_pipeline_phase2_2_im <= (OTHERS => '0');
    product_pipeline_phase3_2_re <= (OTHERS => '0');
    product_pipeline_phase3_2_im <= (OTHERS => '0');
```

```
product_pipeline_phase0_3_re <= (OTHERS => '0');
product_pipeline_phase0_3_im <= (OTHERS => '0');
product_pipeline_phase1_3_re <= (OTHERS => '0');
product_pipeline_phase1_3_im <= (OTHERS => '0');
product_pipeline_phase2_3_re <= (OTHERS => '0');
product_pipeline_phase2_3_im <= (OTHERS => '0');
product_pipeline_phase3_3_re <= (OTHERS => '0');
product_pipeline_phase3_3_im <= (OTHERS => '0');
product_pipeline_phase0_4_re <= (OTHERS => '0');
product_pipeline_phase0_4_im <= (OTHERS => '0');
product_pipeline_phase1_4_re <= (OTHERS => '0');
product_pipeline_phase1_4_im <= (OTHERS => '0');
product_pipeline_phase2_4_re <= (OTHERS => '0');
product_pipeline_phase2_4_im <= (OTHERS => '0');
product_pipeline_phase3_4_re <= (OTHERS => '0');
product_pipeline_phase3_4_im <= (OTHERS => '0');
product_pipeline_phase0_5_re <= (OTHERS => '0');
product_pipeline_phase0_5_im <= (OTHERS => '0');
product_pipeline_phase1_5_re <= (OTHERS => '0');
product_pipeline_phase1_5_im <= (OTHERS => '0');
product_pipeline_phase2_5_re <= (OTHERS => '0');
product_pipeline_phase2_5_im <= (OTHERS => '0');
product_pipeline_phase3_5_re <= (OTHERS => '0');
product_pipeline_phase3_5_im <= (OTHERS => '0');
product_pipeline_phase0_6_re <= (OTHERS => '0');
product_pipeline_phase0_6_im <= (OTHERS => '0');
product_pipeline_phase1_6_re <= (OTHERS => '0');
product_pipeline_phase1_6_im <= (OTHERS => '0');
product_pipeline_phase2_6_re <= (OTHERS => '0');
product_pipeline_phase2_6_im <= (OTHERS => '0');
product_pipeline_phase3_6_re <= (OTHERS => '0');
product_pipeline_phase3_6_im <= (OTHERS => '0');
product_pipeline_phase0_7_re <= (OTHERS => '0');
```

```
product_pipeline_phase0_7_im <= (OTHERS => '0');
product_pipeline_phase1_7_re <= (OTHERS => '0');
product_pipeline_phase1_7_im <= (OTHERS => '0');
product_pipeline_phase2_7_re <= (OTHERS => '0');
product_pipeline_phase2_7_im <= (OTHERS => '0');
product_pipeline_phase3_7_re <= (OTHERS => '0');
product_pipeline_phase3_7_im <= (OTHERS => '0');
product_pipeline_phase0_8_re <= (OTHERS => '0');
product_pipeline_phase0_8_im <= (OTHERS => '0');
product_pipeline_phase1_8_re <= (OTHERS => '0');
product_pipeline_phase1_8_im <= (OTHERS => '0');
product_pipeline_phase2_8_re <= (OTHERS => '0');
product_pipeline_phase2_8_im <= (OTHERS => '0');
product_pipeline_phase3_8_re <= (OTHERS => '0');
product_pipeline_phase3_8_im <= (OTHERS => '0');
product_pipeline_phase0_9_re <= (OTHERS => '0');
product_pipeline_phase0_9_im <= (OTHERS => '0');
product_pipeline_phase1_9_re <= (OTHERS => '0');
product_pipeline_phase1_9_im <= (OTHERS => '0');
product_pipeline_phase2_9_re <= (OTHERS => '0');
product_pipeline_phase2_9_im <= (OTHERS => '0');
product_pipeline_phase3_9_re <= (OTHERS => '0');
product_pipeline_phase3_9_im <= (OTHERS => '0');
product_pipeline_phase0_10_re <= (OTHERS => '0');
product_pipeline_phase0_10_im <= (OTHERS => '0');
product_pipeline_phase1_10_re <= (OTHERS => '0');
product_pipeline_phase1_10_im <= (OTHERS => '0');
product_pipeline_phase2_10_re <= (OTHERS => '0');
product_pipeline_phase2_10_im <= (OTHERS => '0');
product_pipeline_phase3_10_re <= (OTHERS => '0');
product_pipeline_phase3_10_im <= (OTHERS => '0');
product_pipeline_phase0_11_re <= (OTHERS => '0');
product_pipeline_phase0_11_im <= (OTHERS => '0');
```

```
product_pipeline_phase1_11_re <= (OTHERS => '0');
product_pipeline_phase1_11_im <= (OTHERS => '0');
product_pipeline_phase2_11_re <= (OTHERS => '0');
product_pipeline_phase2_11_im <= (OTHERS => '0');
product_pipeline_phase3_11_re <= (OTHERS => '0');
product_pipeline_phase3_11_im <= (OTHERS => '0');
product_pipeline_phase0_12_re <= (OTHERS => '0');
product_pipeline_phase0_12_im <= (OTHERS => '0');
product_pipeline_phase1_12_re <= (OTHERS => '0');
product_pipeline_phase1_12_im <= (OTHERS => '0');
product_pipeline_phase2_12_re <= (OTHERS => '0');
product_pipeline_phase2_12_im <= (OTHERS => '0');
product_pipeline_phase3_12_re <= (OTHERS => '0');
product_pipeline_phase3_12_im <= (OTHERS => '0');
product_pipeline_phase0_13_re <= (OTHERS => '0');
product_pipeline_phase0_13_im <= (OTHERS => '0');
product_pipeline_phase1_13_re <= (OTHERS => '0');
product_pipeline_phase1_13_im <= (OTHERS => '0');
product_pipeline_phase2_13_re <= (OTHERS => '0');
product_pipeline_phase2_13_im <= (OTHERS => '0');
product_pipeline_phase3_13_re <= (OTHERS => '0');
product_pipeline_phase3_13_im <= (OTHERS => '0');
product_pipeline_phase0_14_re <= (OTHERS => '0');
product_pipeline_phase0_14_im <= (OTHERS => '0');
product_pipeline_phase1_14_re <= (OTHERS => '0');
product_pipeline_phase1_14_im <= (OTHERS => '0');
product_pipeline_phase2_14_re <= (OTHERS => '0');
product_pipeline_phase2_14_im <= (OTHERS => '0');
product_pipeline_phase3_14_re <= (OTHERS => '0');
product_pipeline_phase3_14_im <= (OTHERS => '0');
product_pipeline_phase0_15_re <= (OTHERS => '0');
product_pipeline_phase0_15_im <= (OTHERS => '0');
product_pipeline_phase1_15_re <= (OTHERS => '0');
```

```
product_pipeline_phase1_15_im <= (OTHERS => '0');
product_pipeline_phase2_15_re <= (OTHERS => '0');
product_pipeline_phase2_15_im <= (OTHERS => '0');
product_pipeline_phase3_15_re <= (OTHERS => '0');
product_pipeline_phase3_15_im <= (OTHERS => '0');
product_pipeline_phase0_16_re <= (OTHERS => '0');
product_pipeline_phase0_16_im <= (OTHERS => '0');
product_pipeline_phase1_16_re <= (OTHERS => '0');
product_pipeline_phase1_16_im <= (OTHERS => '0');
product_pipeline_phase2_16_re <= (OTHERS => '0');
product_pipeline_phase2_16_im <= (OTHERS => '0');
product_pipeline_phase3_16_re <= (OTHERS => '0');
product_pipeline_phase3_16_im <= (OTHERS => '0');
product_pipeline_phase0_17_re <= (OTHERS => '0');
product_pipeline_phase0_17_im <= (OTHERS => '0');
product_pipeline_phase1_17_re <= (OTHERS => '0');
product_pipeline_phase1_17_im <= (OTHERS => '0');
product_pipeline_phase2_17_re <= (OTHERS => '0');
product_pipeline_phase2_17_im <= (OTHERS => '0');
product_pipeline_phase3_17_re <= (OTHERS => '0');
product_pipeline_phase3_17_im <= (OTHERS => '0');
product_pipeline_phase0_18_re <= (OTHERS => '0');
product_pipeline_phase0_18_im <= (OTHERS => '0');
product_pipeline_phase1_18_re <= (OTHERS => '0');
product_pipeline_phase1_18_im <= (OTHERS => '0');
product_pipeline_phase2_18_re <= (OTHERS => '0');
product_pipeline_phase2_18_im <= (OTHERS => '0');
product_pipeline_phase3_18_re <= (OTHERS => '0');
product_pipeline_phase3_18_im <= (OTHERS => '0');
product_pipeline_phase0_19_re <= (OTHERS => '0');
product_pipeline_phase0_19_im <= (OTHERS => '0');
product_pipeline_phase1_19_re <= (OTHERS => '0');
product_pipeline_phase1_19_im <= (OTHERS => '0');
```

```
    product_pipeline_phase2_19_re <= (OTHERS => '0');
    product_pipeline_phase2_19_im <= (OTHERS => '0');
    product_pipeline_phase3_19_re <= (OTHERS => '0');
    product_pipeline_phase3_19_im <= (OTHERS => '0');
    product_pipeline_phase0_20_re <= (OTHERS => '0');
    product_pipeline_phase0_20_im <= (OTHERS => '0');
    product_pipeline_phase1_20_re <= (OTHERS => '0');
    product_pipeline_phase1_20_im <= (OTHERS => '0');
    product_pipeline_phase2_20_re <= (OTHERS => '0');
    product_pipeline_phase2_20_im <= (OTHERS => '0');
    product_pipeline_phase3_20_re <= (OTHERS => '0');
    product_pipeline_phase3_20_im <= (OTHERS => '0');
    product_pipeline_phase0_21_re <= (OTHERS => '0');
    product_pipeline_phase0_21_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF phase_0 = '1' THEN
      product_pipeline_phase0_1_re <= product_phase0_1_re;
      product_pipeline_phase0_1_im <= product_phase0_1_im;
      product_pipeline_phase1_1_re <= product_phase1_1_re;
      product_pipeline_phase1_1_im <= product_phase1_1_im;
      product_pipeline_phase2_1_re <= product_phase2_1_re;
      product_pipeline_phase2_1_im <= product_phase2_1_im;
      product_pipeline_phase3_1_re <= product_phase3_1_re;
      product_pipeline_phase3_1_im <= product_phase3_1_im;
      product_pipeline_phase0_2_re <= product_phase0_2_re;
      product_pipeline_phase0_2_im <= product_phase0_2_im;
      product_pipeline_phase1_2_re <= product_phase1_2_re;
      product_pipeline_phase1_2_im <= product_phase1_2_im;
      product_pipeline_phase2_2_re <= product_phase2_2_re;
      product_pipeline_phase2_2_im <= product_phase2_2_im;
      product_pipeline_phase3_2_re <= product_phase3_2_re;
      product_pipeline_phase3_2_im <= product_phase3_2_im;
      product_pipeline_phase0_3_re <= product_phase0_3_re;
```

```
product_pipeline_phase0_3_im <= product_phase0_3_im;
product_pipeline_phase1_3_re <= product_phase1_3_re;
product_pipeline_phase1_3_im <= product_phase1_3_im;
product_pipeline_phase2_3_re <= product_phase2_3_re;
product_pipeline_phase2_3_im <= product_phase2_3_im;
product_pipeline_phase3_3_re <= product_phase3_3_re;
product_pipeline_phase3_3_im <= product_phase3_3_im;
product_pipeline_phase0_4_re <= product_phase0_4_re;
product_pipeline_phase0_4_im <= product_phase0_4_im;
product_pipeline_phase1_4_re <= product_phase1_4_re;
product_pipeline_phase1_4_im <= product_phase1_4_im;
product_pipeline_phase2_4_re <= product_phase2_4_re;
product_pipeline_phase2_4_im <= product_phase2_4_im;
product_pipeline_phase3_4_re <= product_phase3_4_re;
product_pipeline_phase3_4_im <= product_phase3_4_im;
product_pipeline_phase0_5_re <= product_phase0_5_re;
product_pipeline_phase0_5_im <= product_phase0_5_im;
product_pipeline_phase1_5_re <= product_phase1_5_re;
product_pipeline_phase1_5_im <= product_phase1_5_im;
product_pipeline_phase2_5_re <= product_phase2_5_re;
product_pipeline_phase2_5_im <= product_phase2_5_im;
product_pipeline_phase3_5_re <= product_phase3_5_re;
product_pipeline_phase3_5_im <= product_phase3_5_im;
product_pipeline_phase0_6_re <= product_phase0_6_re;
product_pipeline_phase0_6_im <= product_phase0_6_im;
product_pipeline_phase1_6_re <= product_phase1_6_re;
product_pipeline_phase1_6_im <= product_phase1_6_im;
product_pipeline_phase2_6_re <= product_phase2_6_re;
product_pipeline_phase2_6_im <= product_phase2_6_im;
product_pipeline_phase3_6_re <= product_phase3_6_re;
product_pipeline_phase3_6_im <= product_phase3_6_im;
product_pipeline_phase0_7_re <= product_phase0_7_re;
product_pipeline_phase0_7_im <= product_phase0_7_im;
```

```
product_pipeline_phase1_7_re <= product_phase1_7_re;
product_pipeline_phase1_7_im <= product_phase1_7_im;
product_pipeline_phase2_7_re <= product_phase2_7_re;
product_pipeline_phase2_7_im <= product_phase2_7_im;
product_pipeline_phase3_7_re <= product_phase3_7_re;
product_pipeline_phase3_7_im <= product_phase3_7_im;
product_pipeline_phase0_8_re <= product_phase0_8_re;
product_pipeline_phase0_8_im <= product_phase0_8_im;
product_pipeline_phase1_8_re <= product_phase1_8_re;
product_pipeline_phase1_8_im <= product_phase1_8_im;
product_pipeline_phase2_8_re <= product_phase2_8_re;
product_pipeline_phase2_8_im <= product_phase2_8_im;
product_pipeline_phase3_8_re <= product_phase3_8_re;
product_pipeline_phase3_8_im <= product_phase3_8_im;
product_pipeline_phase0_9_re <= product_phase0_9_re;
product_pipeline_phase0_9_im <= product_phase0_9_im;
product_pipeline_phase1_9_re <= product_phase1_9_re;
product_pipeline_phase1_9_im <= product_phase1_9_im;
product_pipeline_phase2_9_re <= product_phase2_9_re;
product_pipeline_phase2_9_im <= product_phase2_9_im;
product_pipeline_phase3_9_re <= product_phase3_9_re;
product_pipeline_phase3_9_im <= product_phase3_9_im;
product_pipeline_phase0_10_re <= product_phase0_10_re;
product_pipeline_phase0_10_im <= product_phase0_10_im;
product_pipeline_phase1_10_re <= product_phase1_10_re;
product_pipeline_phase1_10_im <= product_phase1_10_im;
product_pipeline_phase2_10_re <= product_phase2_10_re;
product_pipeline_phase2_10_im <= product_phase2_10_im;
product_pipeline_phase3_10_re <= product_phase3_10_re;
product_pipeline_phase3_10_im <= product_phase3_10_im;
product_pipeline_phase0_11_re <= product_phase0_11_re;
product_pipeline_phase0_11_im <= product_phase0_11_im;
product_pipeline_phase1_11_re <= product_phase1_11_re;
```

```
product_pipeline_phase1_11_im <= product_phase1_11_im;
product_pipeline_phase2_11_re <= product_phase2_11_re;
product_pipeline_phase2_11_im <= product_phase2_11_im;
product_pipeline_phase3_11_re <= product_phase3_11_re;
product_pipeline_phase3_11_im <= product_phase3_11_im;
product_pipeline_phase0_12_re <= product_phase0_12_re;
product_pipeline_phase0_12_im <= product_phase0_12_im;
product_pipeline_phase1_12_re <= product_phase1_12_re;
product_pipeline_phase1_12_im <= product_phase1_12_im;
product_pipeline_phase2_12_re <= product_phase2_12_re;
product_pipeline_phase2_12_im <= product_phase2_12_im;
product_pipeline_phase3_12_re <= product_phase3_12_re;
product_pipeline_phase3_12_im <= product_phase3_12_im;
product_pipeline_phase0_13_re <= product_phase0_13_re;
product_pipeline_phase0_13_im <= product_phase0_13_im;
product_pipeline_phase1_13_re <= product_phase1_13_re;
product_pipeline_phase1_13_im <= product_phase1_13_im;
product_pipeline_phase2_13_re <= product_phase2_13_re;
product_pipeline_phase2_13_im <= product_phase2_13_im;
product_pipeline_phase3_13_re <= product_phase3_13_re;
product_pipeline_phase3_13_im <= product_phase3_13_im;
product_pipeline_phase0_14_re <= product_phase0_14_re;
product_pipeline_phase0_14_im <= product_phase0_14_im;
product_pipeline_phase1_14_re <= product_phase1_14_re;
product_pipeline_phase1_14_im <= product_phase1_14_im;
product_pipeline_phase2_14_re <= product_phase2_14_re;
product_pipeline_phase2_14_im <= product_phase2_14_im;
product_pipeline_phase3_14_re <= product_phase3_14_re;
product_pipeline_phase3_14_im <= product_phase3_14_im;
product_pipeline_phase0_15_re <= product_phase0_15_re;
product_pipeline_phase0_15_im <= product_phase0_15_im;
product_pipeline_phase1_15_re <= product_phase1_15_re;
product_pipeline_phase1_15_im <= product_phase1_15_im;
```

```
product_pipeline_phase2_15_re <= product_phase2_15_re;
product_pipeline_phase2_15_im <= product_phase2_15_im;
product_pipeline_phase3_15_re <= product_phase3_15_re;
product_pipeline_phase3_15_im <= product_phase3_15_im;
product_pipeline_phase0_16_re <= product_phase0_16_re;
product_pipeline_phase0_16_im <= product_phase0_16_im;
product_pipeline_phase1_16_re <= product_phase1_16_re;
product_pipeline_phase1_16_im <= product_phase1_16_im;
product_pipeline_phase2_16_re <= product_phase2_16_re;
product_pipeline_phase2_16_im <= product_phase2_16_im;
product_pipeline_phase3_16_re <= product_phase3_16_re;
product_pipeline_phase3_16_im <= product_phase3_16_im;
product_pipeline_phase0_17_re <= product_phase0_17_re;
product_pipeline_phase0_17_im <= product_phase0_17_im;
product_pipeline_phase1_17_re <= product_phase1_17_re;
product_pipeline_phase1_17_im <= product_phase1_17_im;
product_pipeline_phase2_17_re <= product_phase2_17_re;
product_pipeline_phase2_17_im <= product_phase2_17_im;
product_pipeline_phase3_17_re <= product_phase3_17_re;
product_pipeline_phase3_17_im <= product_phase3_17_im;
product_pipeline_phase0_18_re <= product_phase0_18_re;
product_pipeline_phase0_18_im <= product_phase0_18_im;
product_pipeline_phase1_18_re <= product_phase1_18_re;
product_pipeline_phase1_18_im <= product_phase1_18_im;
product_pipeline_phase2_18_re <= product_phase2_18_re;
product_pipeline_phase2_18_im <= product_phase2_18_im;
product_pipeline_phase3_18_re <= product_phase3_18_re;
product_pipeline_phase3_18_im <= product_phase3_18_im;
product_pipeline_phase0_19_re <= product_phase0_19_re;
product_pipeline_phase0_19_im <= product_phase0_19_im;
product_pipeline_phase1_19_re <= product_phase1_19_re;
product_pipeline_phase1_19_im <= product_phase1_19_im;
product_pipeline_phase2_19_re <= product_phase2_19_re;
```

```
        product_pipeline_phase2_19_im <= product_phase2_19_im;
        product_pipeline_phase3_19_re <= product_phase3_19_re;
        product_pipeline_phase3_19_im <= product_phase3_19_im;
        product_pipeline_phase0_20_re <= product_phase0_20_re;
        product_pipeline_phase0_20_im <= product_phase0_20_im;
        product_pipeline_phase1_20_re <= product_phase1_20_re;
        product_pipeline_phase1_20_im <= product_phase1_20_im;
        product_pipeline_phase2_20_re <= product_phase2_20_re;
        product_pipeline_phase2_20_im <= product_phase2_20_im;
        product_pipeline_phase3_20_re <= product_phase3_20_re;
        product_pipeline_phase3_20_im <= product_phase3_20_im;
        product_pipeline_phase0_21_re <= product_phase0_21_re;
        product_pipeline_phase0_21_im <= product_phase0_21_im;
    END IF;
  END IF;
END PROCESS product_pipeline_process3;

quantized_sum_re <= resize(product_pipeline_phase3_1_re, 33);
quantized_sum_im <= resize(product_pipeline_phase3_1_im, 33);

add_cast <= quantized_sum_re;
add_cast_1 <= resize(product_pipeline_phase3_2_re, 33);
add_temp <= resize(add_cast, 34) + resize(add_cast_1, 34);
sumvector1_re(0) <= (32 => '0', OTHERS => '1') WHEN add_temp(33) = '0'
    AND add_temp(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp(33) = '1' AND
        add_temp(32) /= '1'
    ELSE (add_temp(32 DOWNTO 0));

add_cast_2 <= quantized_sum_im;
add_cast_3 <= resize(product_pipeline_phase3_2_im, 33);
add_temp_1 <= resize(add_cast_2, 34) + resize(add_cast_3, 34);
sumvector1_im(0) <= (32 => '0', OTHERS => '1') WHEN add_temp_1(33) =
```

```
    '0' AND add_temp_1(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1(33) = '1' AND
        add_temp_1(32) /= '1'
    ELSE (add_temp_1(32 DOWNTO 0));

add_cast_4 <= resize(product_pipeline_phase3_3_re, 33);
add_cast_5 <= resize(product_pipeline_phase3_4_re, 33);
add_temp_2 <= resize(add_cast_4, 34) + resize(add_cast_5, 34);
sumvector1_re(1) <= (32 => '0', OTHERS => '1') WHEN add_temp_2(33) =
    '0' AND add_temp_2(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_2(33) = '1' AND
        add_temp_2(32) /= '1'
    ELSE (add_temp_2(32 DOWNTO 0));

add_cast_6 <= resize(product_pipeline_phase3_3_im, 33);
add_cast_7 <= resize(product_pipeline_phase3_4_im, 33);
add_temp_3 <= resize(add_cast_6, 34) + resize(add_cast_7, 34);
sumvector1_im(1) <= (32 => '0', OTHERS => '1') WHEN add_temp_3(33) =
    '0' AND add_temp_3(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_3(33) = '1' AND
        add_temp_3(32) /= '1'
    ELSE (add_temp_3(32 DOWNTO 0));

add_cast_8 <= resize(product_pipeline_phase3_5_re, 33);
add_cast_9 <= resize(product_pipeline_phase3_6_re, 33);
add_temp_4 <= resize(add_cast_8, 34) + resize(add_cast_9, 34);
sumvector1_re(2) <= (32 => '0', OTHERS => '1') WHEN add_temp_4(33) =
    '0' AND add_temp_4(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_4(33) = '1' AND
        add_temp_4(32) /= '1'
    ELSE (add_temp_4(32 DOWNTO 0));

add_cast_10 <= resize(product_pipeline_phase3_5_im, 33);
```

```
add_cast_11 <= resize(product_pipeline_phase3_6_im, 33);
add_temp_5 <= resize(add_cast_10, 34) + resize(add_cast_11, 34);
sumvector1_im(2) <= (32 => '0', OTHERS => '1') WHEN add_temp_5(33) =
    '0' AND add_temp_5(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_5(33) = '1' AND
        add_temp_5(32) /= '1'
    ELSE (add_temp_5(32 DOWNTO 0));

add_cast_12 <= resize(product_pipeline_phase3_7_re, 33);
add_cast_13 <= resize(product_pipeline_phase3_8_re, 33);
add_temp_6 <= resize(add_cast_12, 34) + resize(add_cast_13, 34);
sumvector1_re(3) <= (32 => '0', OTHERS => '1') WHEN add_temp_6(33) =
    '0' AND add_temp_6(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_6(33) = '1' AND
        add_temp_6(32) /= '1'
    ELSE (add_temp_6(32 DOWNTO 0));

add_cast_14 <= resize(product_pipeline_phase3_7_im, 33);
add_cast_15 <= resize(product_pipeline_phase3_8_im, 33);
add_temp_7 <= resize(add_cast_14, 34) + resize(add_cast_15, 34);
sumvector1_im(3) <= (32 => '0', OTHERS => '1') WHEN add_temp_7(33) =
    '0' AND add_temp_7(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_7(33) = '1' AND
        add_temp_7(32) /= '1'
    ELSE (add_temp_7(32 DOWNTO 0));

add_cast_16 <= resize(product_pipeline_phase3_9_re, 33);
add_cast_17 <= resize(product_pipeline_phase3_10_re, 33);
add_temp_8 <= resize(add_cast_16, 34) + resize(add_cast_17, 34);
sumvector1_re(4) <= (32 => '0', OTHERS => '1') WHEN add_temp_8(33) =
    '0' AND add_temp_8(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_8(33) = '1' AND
        add_temp_8(32) /= '1'
```

```
    ELSE (add_temp_8(32 DOWNTO 0));

add_cast_18 <= resize(product_pipeline_phase3_9_im, 33);
add_cast_19 <= resize(product_pipeline_phase3_10_im, 33);
add_temp_9 <= resize(add_cast_18, 34) + resize(add_cast_19, 34);
sumvector1_im(4) <= (32 => '0', OTHERS => '1') WHEN add_temp_9(33) =
    '0' AND add_temp_9(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_9(33) = '1' AND
        add_temp_9(32) /= '1'
    ELSE (add_temp_9(32 DOWNTO 0));

add_cast_20 <= resize(product_pipeline_phase3_11_re, 33);
add_cast_21 <= resize(product_pipeline_phase3_12_re, 33);
add_temp_10 <= resize(add_cast_20, 34) + resize(add_cast_21, 34);
sumvector1_re(5) <= (32 => '0', OTHERS => '1') WHEN add_temp_10(33) =
    '0' AND add_temp_10(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_10(33) = '1' AND
        add_temp_10(32) /= '1'
    ELSE (add_temp_10(32 DOWNTO 0));

add_cast_22 <= resize(product_pipeline_phase3_11_im, 33);
add_cast_23 <= resize(product_pipeline_phase3_12_im, 33);
add_temp_11 <= resize(add_cast_22, 34) + resize(add_cast_23, 34);
sumvector1_im(5) <= (32 => '0', OTHERS => '1') WHEN add_temp_11(33) =
    '0' AND add_temp_11(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_11(33) = '1' AND
        add_temp_11(32) /= '1'
    ELSE (add_temp_11(32 DOWNTO 0));

add_cast_24 <= resize(product_pipeline_phase3_13_re, 33);
add_cast_25 <= resize(product_pipeline_phase3_14_re, 33);
add_temp_12 <= resize(add_cast_24, 34) + resize(add_cast_25, 34);
sumvector1_re(6) <= (32 => '0', OTHERS => '1') WHEN add_temp_12(33) =
```

```
    '0' AND add_temp_12(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_12(33) = '1' AND
        add_temp_12(32) /= '1'
    ELSE (add_temp_12(32 DOWNTO 0));

add_cast_26 <= resize(product_pipeline_phase3_13_im, 33);
add_cast_27 <= resize(product_pipeline_phase3_14_im, 33);
add_temp_13 <= resize(add_cast_26, 34) + resize(add_cast_27, 34);
sumvector1_im(6) <= (32 => '0', OTHERS => '1') WHEN add_temp_13(33) =
    '0' AND add_temp_13(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_13(33) = '1' AND
        add_temp_13(32) /= '1'
    ELSE (add_temp_13(32 DOWNTO 0));

add_cast_28 <= resize(product_pipeline_phase3_15_re, 33);
add_cast_29 <= resize(product_pipeline_phase3_16_re, 33);
add_temp_14 <= resize(add_cast_28, 34) + resize(add_cast_29, 34);
sumvector1_re(7) <= (32 => '0', OTHERS => '1') WHEN add_temp_14(33) =
    '0' AND add_temp_14(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_14(33) = '1' AND
        add_temp_14(32) /= '1'
    ELSE (add_temp_14(32 DOWNTO 0));

add_cast_30 <= resize(product_pipeline_phase3_15_im, 33);
add_cast_31 <= resize(product_pipeline_phase3_16_im, 33);
add_temp_15 <= resize(add_cast_30, 34) + resize(add_cast_31, 34);
sumvector1_im(7) <= (32 => '0', OTHERS => '1') WHEN add_temp_15(33) =
    '0' AND add_temp_15(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_15(33) = '1' AND
        add_temp_15(32) /= '1'
    ELSE (add_temp_15(32 DOWNTO 0));

add_cast_32 <= resize(product_pipeline_phase3_17_re, 33);
```

```
add_cast_33 <= resize(product_pipeline_phase3_18_re, 33);
add_temp_16 <= resize(add_cast_32, 34) + resize(add_cast_33, 34);
sumvector1_re(8) <= (32 => '0', OTHERS => '1') WHEN add_temp_16(33) =
    '0' AND add_temp_16(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_16(33) = '1' AND
        add_temp_16(32) /= '1'
    ELSE (add_temp_16(32 DOWNTO 0));

add_cast_34 <= resize(product_pipeline_phase3_17_im, 33);
add_cast_35 <= resize(product_pipeline_phase3_18_im, 33);
add_temp_17 <= resize(add_cast_34, 34) + resize(add_cast_35, 34);
sumvector1_im(8) <= (32 => '0', OTHERS => '1') WHEN add_temp_17(33) =
    '0' AND add_temp_17(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_17(33) = '1' AND
        add_temp_17(32) /= '1'
    ELSE (add_temp_17(32 DOWNTO 0));

add_cast_36 <= resize(product_pipeline_phase3_19_re, 33);
add_cast_37 <= resize(product_pipeline_phase3_20_re, 33);
add_temp_18 <= resize(add_cast_36, 34) + resize(add_cast_37, 34);
sumvector1_re(9) <= (32 => '0', OTHERS => '1') WHEN add_temp_18(33) =
    '0' AND add_temp_18(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_18(33) = '1' AND
        add_temp_18(32) /= '1'
    ELSE (add_temp_18(32 DOWNTO 0));

add_cast_38 <= resize(product_pipeline_phase3_19_im, 33);
add_cast_39 <= resize(product_pipeline_phase3_20_im, 33);
add_temp_19 <= resize(add_cast_38, 34) + resize(add_cast_39, 34);
sumvector1_im(9) <= (32 => '0', OTHERS => '1') WHEN add_temp_19(33) =
    '0' AND add_temp_19(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_19(33) = '1' AND
        add_temp_19(32) /= '1'
```

```
    ELSE (add_temp_19(32 DOWNTO 0));

add_cast_40 <= resize(product_pipeline_phase2_1_re, 33);
add_cast_41 <= resize(product_pipeline_phase2_2_re, 33);
add_temp_20 <= resize(add_cast_40, 34) + resize(add_cast_41, 34);
sumvector1_re(10) <= (32 => '0', OTHERS => '1') WHEN add_temp_20(33) =
    '0' AND add_temp_20(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_20(33) = '1' AND
        add_temp_20(32) /= '1'
    ELSE (add_temp_20(32 DOWNTO 0));

add_cast_42 <= resize(product_pipeline_phase2_1_im, 33);
add_cast_43 <= resize(product_pipeline_phase2_2_im, 33);
add_temp_21 <= resize(add_cast_42, 34) + resize(add_cast_43, 34);
sumvector1_im(10) <= (32 => '0', OTHERS => '1') WHEN add_temp_21(33) =
    '0' AND add_temp_21(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_21(33) = '1' AND
        add_temp_21(32) /= '1'
    ELSE (add_temp_21(32 DOWNTO 0));

add_cast_44 <= resize(product_pipeline_phase2_3_re, 33);
add_cast_45 <= resize(product_pipeline_phase2_4_re, 33);
add_temp_22 <= resize(add_cast_44, 34) + resize(add_cast_45, 34);
sumvector1_re(11) <= (32 => '0', OTHERS => '1') WHEN add_temp_22(33) =
    '0' AND add_temp_22(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_22(33) = '1' AND
        add_temp_22(32) /= '1'
    ELSE (add_temp_22(32 DOWNTO 0));

add_cast_46 <= resize(product_pipeline_phase2_3_im, 33);
add_cast_47 <= resize(product_pipeline_phase2_4_im, 33);
add_temp_23 <= resize(add_cast_46, 34) + resize(add_cast_47, 34);
sumvector1_im(11) <= (32 => '0', OTHERS => '1') WHEN add_temp_23(33) =
```

```
     '0' AND add_temp_23(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_23(33) = '1' AND
        add_temp_23(32) /= '1'
    ELSE (add_temp_23(32 DOWNTO 0));

add_cast_48 <= resize(product_pipeline_phase2_5_re, 33);
add_cast_49 <= resize(product_pipeline_phase2_6_re, 33);
add_temp_24 <= resize(add_cast_48, 34) + resize(add_cast_49, 34);
sumvector1_re(12) <= (32 => '0', OTHERS => '1') WHEN add_temp_24(33) =
     '0' AND add_temp_24(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_24(33) = '1' AND
        add_temp_24(32) /= '1'
    ELSE (add_temp_24(32 DOWNTO 0));

add_cast_50 <= resize(product_pipeline_phase2_5_im, 33);
add_cast_51 <= resize(product_pipeline_phase2_6_im, 33);
add_temp_25 <= resize(add_cast_50, 34) + resize(add_cast_51, 34);
sumvector1_im(12) <= (32 => '0', OTHERS => '1') WHEN add_temp_25(33) =
     '0' AND add_temp_25(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_25(33) = '1' AND
        add_temp_25(32) /= '1'
    ELSE (add_temp_25(32 DOWNTO 0));

add_cast_52 <= resize(product_pipeline_phase2_7_re, 33);
add_cast_53 <= resize(product_pipeline_phase2_8_re, 33);
add_temp_26 <= resize(add_cast_52, 34) + resize(add_cast_53, 34);
sumvector1_re(13) <= (32 => '0', OTHERS => '1') WHEN add_temp_26(33) =
     '0' AND add_temp_26(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_26(33) = '1' AND
        add_temp_26(32) /= '1'
    ELSE (add_temp_26(32 DOWNTO 0));

add_cast_54 <= resize(product_pipeline_phase2_7_im, 33);
```

```
add_cast_55 <= resize(product_pipeline_phase2_8_im, 33);
add_temp_27 <= resize(add_cast_54, 34) + resize(add_cast_55, 34);
sumvector1_im(13) <= (32 => '0', OTHERS => '1') WHEN add_temp_27(33) =
    '0' AND add_temp_27(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_27(33) = '1' AND
        add_temp_27(32) /= '1'
    ELSE (add_temp_27(32 DOWNTO 0));

add_cast_56 <= resize(product_pipeline_phase2_9_re, 33);
add_cast_57 <= resize(product_pipeline_phase2_10_re, 33);
add_temp_28 <= resize(add_cast_56, 34) + resize(add_cast_57, 34);
sumvector1_re(14) <= (32 => '0', OTHERS => '1') WHEN add_temp_28(33) =
    '0' AND add_temp_28(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_28(33) = '1' AND
        add_temp_28(32) /= '1'
    ELSE (add_temp_28(32 DOWNTO 0));

add_cast_58 <= resize(product_pipeline_phase2_9_im, 33);
add_cast_59 <= resize(product_pipeline_phase2_10_im, 33);
add_temp_29 <= resize(add_cast_58, 34) + resize(add_cast_59, 34);
sumvector1_im(14) <= (32 => '0', OTHERS => '1') WHEN add_temp_29(33) =
    '0' AND add_temp_29(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_29(33) = '1' AND
        add_temp_29(32) /= '1'
    ELSE (add_temp_29(32 DOWNTO 0));

add_cast_60 <= resize(product_pipeline_phase2_11_re, 33);
add_cast_61 <= resize(product_pipeline_phase2_12_re, 33);
add_temp_30 <= resize(add_cast_60, 34) + resize(add_cast_61, 34);
sumvector1_re(15) <= (32 => '0', OTHERS => '1') WHEN add_temp_30(33) =
    '0' AND add_temp_30(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_30(33) = '1' AND
        add_temp_30(32) /= '1'
```

```
    ELSE (add_temp_30(32 DOWNTO 0));

add_cast_62 <= resize(product_pipeline_phase2_11_im, 33);
add_cast_63 <= resize(product_pipeline_phase2_12_im, 33);
add_temp_31 <= resize(add_cast_62, 34) + resize(add_cast_63, 34);
sumvector1_im(15) <= (32 => '0', OTHERS => '1') WHEN add_temp_31(33) =
    '0' AND add_temp_31(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_31(33) = '1' AND
        add_temp_31(32) /= '1'
    ELSE (add_temp_31(32 DOWNTO 0));

add_cast_64 <= resize(product_pipeline_phase2_13_re, 33);
add_cast_65 <= resize(product_pipeline_phase2_14_re, 33);
add_temp_32 <= resize(add_cast_64, 34) + resize(add_cast_65, 34);
sumvector1_re(16) <= (32 => '0', OTHERS => '1') WHEN add_temp_32(33) =
    '0' AND add_temp_32(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_32(33) = '1' AND
        add_temp_32(32) /= '1'
    ELSE (add_temp_32(32 DOWNTO 0));

add_cast_66 <= resize(product_pipeline_phase2_13_im, 33);
add_cast_67 <= resize(product_pipeline_phase2_14_im, 33);
add_temp_33 <= resize(add_cast_66, 34) + resize(add_cast_67, 34);
sumvector1_im(16) <= (32 => '0', OTHERS => '1') WHEN add_temp_33(33) =
    '0' AND add_temp_33(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_33(33) = '1' AND
        add_temp_33(32) /= '1'
    ELSE (add_temp_33(32 DOWNTO 0));

add_cast_68 <= resize(product_pipeline_phase2_15_re, 33);
add_cast_69 <= resize(product_pipeline_phase2_16_re, 33);
add_temp_34 <= resize(add_cast_68, 34) + resize(add_cast_69, 34);
sumvector1_re(17) <= (32 => '0', OTHERS => '1') WHEN add_temp_34(33) =
```

```
     '0' AND add_temp_34(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_34(33) = '1' AND
        add_temp_34(32) /= '1'
    ELSE (add_temp_34(32 DOWNTO 0));

add_cast_70 <= resize(product_pipeline_phase2_15_im, 33);
add_cast_71 <= resize(product_pipeline_phase2_16_im, 33);
add_temp_35 <= resize(add_cast_70, 34) + resize(add_cast_71, 34);
sumvector1_im(17) <= (32 => '0', OTHERS => '1') WHEN add_temp_35(33) =
     '0' AND add_temp_35(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_35(33) = '1' AND
        add_temp_35(32) /= '1'
    ELSE (add_temp_35(32 DOWNTO 0));

add_cast_72 <= resize(product_pipeline_phase2_17_re, 33);
add_cast_73 <= resize(product_pipeline_phase2_18_re, 33);
add_temp_36 <= resize(add_cast_72, 34) + resize(add_cast_73, 34);
sumvector1_re(18) <= (32 => '0', OTHERS => '1') WHEN add_temp_36(33) =
     '0' AND add_temp_36(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_36(33) = '1' AND
        add_temp_36(32) /= '1'
    ELSE (add_temp_36(32 DOWNTO 0));

add_cast_74 <= resize(product_pipeline_phase2_17_im, 33);
add_cast_75 <= resize(product_pipeline_phase2_18_im, 33);
add_temp_37 <= resize(add_cast_74, 34) + resize(add_cast_75, 34);
sumvector1_im(18) <= (32 => '0', OTHERS => '1') WHEN add_temp_37(33) =
     '0' AND add_temp_37(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_37(33) = '1' AND
        add_temp_37(32) /= '1'
    ELSE (add_temp_37(32 DOWNTO 0));

add_cast_76 <= resize(product_pipeline_phase2_19_re, 33);
```

```
add_cast_77 <= resize(product_pipeline_phase2_20_re, 33);
add_temp_38 <= resize(add_cast_76, 34) + resize(add_cast_77, 34);
sumvector1_re(19) <= (32 => '0', OTHERS => '1') WHEN add_temp_38(33) =
    '0' AND add_temp_38(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_38(33) = '1' AND
        add_temp_38(32) /= '1'
    ELSE (add_temp_38(32 DOWNTO 0));

add_cast_78 <= resize(product_pipeline_phase2_19_im, 33);
add_cast_79 <= resize(product_pipeline_phase2_20_im, 33);
add_temp_39 <= resize(add_cast_78, 34) + resize(add_cast_79, 34);
sumvector1_im(19) <= (32 => '0', OTHERS => '1') WHEN add_temp_39(33) =
    '0' AND add_temp_39(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_39(33) = '1' AND
        add_temp_39(32) /= '1'
    ELSE (add_temp_39(32 DOWNTO 0));

add_cast_80 <= resize(product_pipeline_phase1_1_re, 33);
add_cast_81 <= resize(product_pipeline_phase1_2_re, 33);
add_temp_40 <= resize(add_cast_80, 34) + resize(add_cast_81, 34);
sumvector1_re(20) <= (32 => '0', OTHERS => '1') WHEN add_temp_40(33) =
    '0' AND add_temp_40(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_40(33) = '1' AND
        add_temp_40(32) /= '1'
    ELSE (add_temp_40(32 DOWNTO 0));

add_cast_82 <= resize(product_pipeline_phase1_1_im, 33);
add_cast_83 <= resize(product_pipeline_phase1_2_im, 33);
add_temp_41 <= resize(add_cast_82, 34) + resize(add_cast_83, 34);
sumvector1_im(20) <= (32 => '0', OTHERS => '1') WHEN add_temp_41(33) =
    '0' AND add_temp_41(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_41(33) = '1' AND
        add_temp_41(32) /= '1'
```

```
    ELSE (add_temp_41(32 DOWNTO 0));

add_cast_84 <= resize(product_pipeline_phase1_3_re, 33);
add_cast_85 <= resize(product_pipeline_phase1_4_re, 33);
add_temp_42 <= resize(add_cast_84, 34) + resize(add_cast_85, 34);
sumvector1_re(21) <= (32 => '0', OTHERS => '1') WHEN add_temp_42(33) =
     '0' AND add_temp_42(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_42(33) = '1' AND
        add_temp_42(32) /= '1'
    ELSE (add_temp_42(32 DOWNTO 0));

add_cast_86 <= resize(product_pipeline_phase1_3_im, 33);
add_cast_87 <= resize(product_pipeline_phase1_4_im, 33);
add_temp_43 <= resize(add_cast_86, 34) + resize(add_cast_87, 34);
sumvector1_im(21) <= (32 => '0', OTHERS => '1') WHEN add_temp_43(33) =
     '0' AND add_temp_43(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_43(33) = '1' AND
        add_temp_43(32) /= '1'
    ELSE (add_temp_43(32 DOWNTO 0));

add_cast_88 <= resize(product_pipeline_phase1_5_re, 33);
add_cast_89 <= resize(product_pipeline_phase1_6_re, 33);
add_temp_44 <= resize(add_cast_88, 34) + resize(add_cast_89, 34);
sumvector1_re(22) <= (32 => '0', OTHERS => '1') WHEN add_temp_44(33) =
     '0' AND add_temp_44(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_44(33) = '1' AND
        add_temp_44(32) /= '1'
    ELSE (add_temp_44(32 DOWNTO 0));

add_cast_90 <= resize(product_pipeline_phase1_5_im, 33);
add_cast_91 <= resize(product_pipeline_phase1_6_im, 33);
add_temp_45 <= resize(add_cast_90, 34) + resize(add_cast_91, 34);
sumvector1_im(22) <= (32 => '0', OTHERS => '1') WHEN add_temp_45(33) =
```

```
     '0' AND add_temp_45(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_45(33) = '1' AND
        add_temp_45(32) /= '1'
    ELSE (add_temp_45(32 DOWNTO 0));

add_cast_92 <= resize(product_pipeline_phase1_7_re, 33);
add_cast_93 <= resize(product_pipeline_phase1_8_re, 33);
add_temp_46 <= resize(add_cast_92, 34) + resize(add_cast_93, 34);
sumvector1_re(23) <= (32 => '0', OTHERS => '1') WHEN add_temp_46(33) =
     '0' AND add_temp_46(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_46(33) = '1' AND
        add_temp_46(32) /= '1'
    ELSE (add_temp_46(32 DOWNTO 0));

add_cast_94 <= resize(product_pipeline_phase1_7_im, 33);
add_cast_95 <= resize(product_pipeline_phase1_8_im, 33);
add_temp_47 <= resize(add_cast_94, 34) + resize(add_cast_95, 34);
sumvector1_im(23) <= (32 => '0', OTHERS => '1') WHEN add_temp_47(33) =
     '0' AND add_temp_47(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_47(33) = '1' AND
        add_temp_47(32) /= '1'
    ELSE (add_temp_47(32 DOWNTO 0));

add_cast_96 <= resize(product_pipeline_phase1_9_re, 33);
add_cast_97 <= resize(product_pipeline_phase1_10_re, 33);
add_temp_48 <= resize(add_cast_96, 34) + resize(add_cast_97, 34);
sumvector1_re(24) <= (32 => '0', OTHERS => '1') WHEN add_temp_48(33) =
     '0' AND add_temp_48(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_48(33) = '1' AND
        add_temp_48(32) /= '1'
    ELSE (add_temp_48(32 DOWNTO 0));

add_cast_98 <= resize(product_pipeline_phase1_9_im, 33);
```

```
add_cast_99 <= resize(product_pipeline_phase1_10_im, 33);
add_temp_49 <= resize(add_cast_98, 34) + resize(add_cast_99, 34);
sumvector1_im(24) <= (32 => '0', OTHERS => '1') WHEN add_temp_49(33) =
    '0' AND add_temp_49(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_49(33) = '1' AND
        add_temp_49(32) /= '1'
    ELSE (add_temp_49(32 DOWNTO 0));

add_cast_100 <= resize(product_pipeline_phase1_11_re, 33);
add_cast_101 <= resize(product_pipeline_phase1_12_re, 33);
add_temp_50 <= resize(add_cast_100, 34) + resize(add_cast_101, 34);
sumvector1_re(25) <= (32 => '0', OTHERS => '1') WHEN add_temp_50(33) =
    '0' AND add_temp_50(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_50(33) = '1' AND
        add_temp_50(32) /= '1'
    ELSE (add_temp_50(32 DOWNTO 0));

add_cast_102 <= resize(product_pipeline_phase1_11_im, 33);
add_cast_103 <= resize(product_pipeline_phase1_12_im, 33);
add_temp_51 <= resize(add_cast_102, 34) + resize(add_cast_103, 34);
sumvector1_im(25) <= (32 => '0', OTHERS => '1') WHEN add_temp_51(33) =
    '0' AND add_temp_51(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_51(33) = '1' AND
        add_temp_51(32) /= '1'
    ELSE (add_temp_51(32 DOWNTO 0));

add_cast_104 <= resize(product_pipeline_phase1_13_re, 33);
add_cast_105 <= resize(product_pipeline_phase1_14_re, 33);
add_temp_52 <= resize(add_cast_104, 34) + resize(add_cast_105, 34);
sumvector1_re(26) <= (32 => '0', OTHERS => '1') WHEN add_temp_52(33) =
    '0' AND add_temp_52(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_52(33) = '1' AND
        add_temp_52(32) /= '1'
```

```
    ELSE (add_temp_52(32 DOWNTO 0));

add_cast_106 <= resize(product_pipeline_phase1_13_im, 33);
add_cast_107 <= resize(product_pipeline_phase1_14_im, 33);
add_temp_53 <= resize(add_cast_106, 34) + resize(add_cast_107, 34);
sumvector1_im(26) <= (32 => '0', OTHERS => '1') WHEN add_temp_53(33) =
    '0' AND add_temp_53(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_53(33) = '1' AND
        add_temp_53(32) /= '1'
    ELSE (add_temp_53(32 DOWNTO 0));

add_cast_108 <= resize(product_pipeline_phase1_15_re, 33);
add_cast_109 <= resize(product_pipeline_phase1_16_re, 33);
add_temp_54 <= resize(add_cast_108, 34) + resize(add_cast_109, 34);
sumvector1_re(27) <= (32 => '0', OTHERS => '1') WHEN add_temp_54(33) =
    '0' AND add_temp_54(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_54(33) = '1' AND
        add_temp_54(32) /= '1'
    ELSE (add_temp_54(32 DOWNTO 0));

add_cast_110 <= resize(product_pipeline_phase1_15_im, 33);
add_cast_111 <= resize(product_pipeline_phase1_16_im, 33);
add_temp_55 <= resize(add_cast_110, 34) + resize(add_cast_111, 34);
sumvector1_im(27) <= (32 => '0', OTHERS => '1') WHEN add_temp_55(33) =
    '0' AND add_temp_55(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_55(33) = '1' AND
        add_temp_55(32) /= '1'
    ELSE (add_temp_55(32 DOWNTO 0));

add_cast_112 <= resize(product_pipeline_phase1_17_re, 33);
add_cast_113 <= resize(product_pipeline_phase1_18_re, 33);
add_temp_56 <= resize(add_cast_112, 34) + resize(add_cast_113, 34);
sumvector1_re(28) <= (32 => '0', OTHERS => '1') WHEN add_temp_56(33) =
```

```
     '0' AND add_temp_56(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_56(33) = '1' AND
        add_temp_56(32) /= '1'
    ELSE (add_temp_56(32 DOWNTO 0));

add_cast_114 <= resize(product_pipeline_phase1_17_im, 33);
add_cast_115 <= resize(product_pipeline_phase1_18_im, 33);
add_temp_57 <= resize(add_cast_114, 34) + resize(add_cast_115, 34);
sumvector1_im(28) <= (32 => '0', OTHERS => '1') WHEN add_temp_57(33) =
     '0' AND add_temp_57(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_57(33) = '1' AND
        add_temp_57(32) /= '1'
    ELSE (add_temp_57(32 DOWNTO 0));

add_cast_116 <= resize(product_pipeline_phase1_19_re, 33);
add_cast_117 <= resize(product_pipeline_phase1_20_re, 33);
add_temp_58 <= resize(add_cast_116, 34) + resize(add_cast_117, 34);
sumvector1_re(29) <= (32 => '0', OTHERS => '1') WHEN add_temp_58(33) =
     '0' AND add_temp_58(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_58(33) = '1' AND
        add_temp_58(32) /= '1'
    ELSE (add_temp_58(32 DOWNTO 0));

add_cast_118 <= resize(product_pipeline_phase1_19_im, 33);
add_cast_119 <= resize(product_pipeline_phase1_20_im, 33);
add_temp_59 <= resize(add_cast_118, 34) + resize(add_cast_119, 34);
sumvector1_im(29) <= (32 => '0', OTHERS => '1') WHEN add_temp_59(33) =
     '0' AND add_temp_59(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_59(33) = '1' AND
        add_temp_59(32) /= '1'
    ELSE (add_temp_59(32 DOWNTO 0));

add_cast_120 <= resize(product_pipeline_phase0_1_re, 33);
```

```
add_cast_121 <= resize(product_pipeline_phase0_2_re, 33);
add_temp_60 <= resize(add_cast_120, 34) + resize(add_cast_121, 34);
sumvector1_re(30) <= (32 => '0', OTHERS => '1') WHEN add_temp_60(33) =
    '0' AND add_temp_60(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_60(33) = '1' AND
        add_temp_60(32) /= '1'
    ELSE (add_temp_60(32 DOWNTO 0));

add_cast_122 <= resize(product_pipeline_phase0_1_im, 33);
add_cast_123 <= resize(product_pipeline_phase0_2_im, 33);
add_temp_61 <= resize(add_cast_122, 34) + resize(add_cast_123, 34);
sumvector1_im(30) <= (32 => '0', OTHERS => '1') WHEN add_temp_61(33) =
    '0' AND add_temp_61(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_61(33) = '1' AND
        add_temp_61(32) /= '1'
    ELSE (add_temp_61(32 DOWNTO 0));

add_cast_124 <= resize(product_pipeline_phase0_3_re, 33);
add_cast_125 <= resize(product_pipeline_phase0_4_re, 33);
add_temp_62 <= resize(add_cast_124, 34) + resize(add_cast_125, 34);
sumvector1_re(31) <= (32 => '0', OTHERS => '1') WHEN add_temp_62(33) =
    '0' AND add_temp_62(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_62(33) = '1' AND
        add_temp_62(32) /= '1'
    ELSE (add_temp_62(32 DOWNTO 0));

add_cast_126 <= resize(product_pipeline_phase0_3_im, 33);
add_cast_127 <= resize(product_pipeline_phase0_4_im, 33);
add_temp_63 <= resize(add_cast_126, 34) + resize(add_cast_127, 34);
sumvector1_im(31) <= (32 => '0', OTHERS => '1') WHEN add_temp_63(33) =
    '0' AND add_temp_63(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_63(33) = '1' AND
        add_temp_63(32) /= '1'
```

```
    ELSE (add_temp_63(32 DOWNTO 0));

add_cast_128 <= resize(product_pipeline_phase0_5_re, 33);
add_cast_129 <= resize(product_pipeline_phase0_6_re, 33);
add_temp_64 <= resize(add_cast_128, 34) + resize(add_cast_129, 34);
sumvector1_re(32) <= (32 => '0', OTHERS => '1') WHEN add_temp_64(33) =
     '0' AND add_temp_64(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_64(33) = '1' AND
        add_temp_64(32) /= '1'
    ELSE (add_temp_64(32 DOWNTO 0));

add_cast_130 <= resize(product_pipeline_phase0_5_im, 33);
add_cast_131 <= resize(product_pipeline_phase0_6_im, 33);
add_temp_65 <= resize(add_cast_130, 34) + resize(add_cast_131, 34);
sumvector1_im(32) <= (32 => '0', OTHERS => '1') WHEN add_temp_65(33) =
     '0' AND add_temp_65(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_65(33) = '1' AND
        add_temp_65(32) /= '1'
    ELSE (add_temp_65(32 DOWNTO 0));

add_cast_132 <= resize(product_pipeline_phase0_7_re, 33);
add_cast_133 <= resize(product_pipeline_phase0_8_re, 33);
add_temp_66 <= resize(add_cast_132, 34) + resize(add_cast_133, 34);
sumvector1_re(33) <= (32 => '0', OTHERS => '1') WHEN add_temp_66(33) =
     '0' AND add_temp_66(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_66(33) = '1' AND
        add_temp_66(32) /= '1'
    ELSE (add_temp_66(32 DOWNTO 0));

add_cast_134 <= resize(product_pipeline_phase0_7_im, 33);
add_cast_135 <= resize(product_pipeline_phase0_8_im, 33);
add_temp_67 <= resize(add_cast_134, 34) + resize(add_cast_135, 34);
sumvector1_im(33) <= (32 => '0', OTHERS => '1') WHEN add_temp_67(33) =
```

```
     '0' AND add_temp_67(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_67(33) = '1' AND
        add_temp_67(32) /= '1'
    ELSE (add_temp_67(32 DOWNTO 0));

add_cast_136 <= resize(product_pipeline_phase0_9_re, 33);
add_cast_137 <= resize(product_pipeline_phase0_10_re, 33);
add_temp_68 <= resize(add_cast_136, 34) + resize(add_cast_137, 34);
sumvector1_re(34) <= (32 => '0', OTHERS => '1') WHEN add_temp_68(33) =
     '0' AND add_temp_68(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_68(33) = '1' AND
        add_temp_68(32) /= '1'
    ELSE (add_temp_68(32 DOWNTO 0));

add_cast_138 <= resize(product_pipeline_phase0_9_im, 33);
add_cast_139 <= resize(product_pipeline_phase0_10_im, 33);
add_temp_69 <= resize(add_cast_138, 34) + resize(add_cast_139, 34);
sumvector1_im(34) <= (32 => '0', OTHERS => '1') WHEN add_temp_69(33) =
     '0' AND add_temp_69(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_69(33) = '1' AND
        add_temp_69(32) /= '1'
    ELSE (add_temp_69(32 DOWNTO 0));

add_cast_140 <= resize(product_pipeline_phase0_11_re, 33);
add_cast_141 <= resize(product_pipeline_phase0_12_re, 33);
add_temp_70 <= resize(add_cast_140, 34) + resize(add_cast_141, 34);
sumvector1_re(35) <= (32 => '0', OTHERS => '1') WHEN add_temp_70(33) =
     '0' AND add_temp_70(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_70(33) = '1' AND
        add_temp_70(32) /= '1'
    ELSE (add_temp_70(32 DOWNTO 0));

add_cast_142 <= resize(product_pipeline_phase0_11_im, 33);
```

```
add_cast_143 <= resize(product_pipeline_phase0_12_im, 33);
add_temp_71 <= resize(add_cast_142, 34) + resize(add_cast_143, 34);
sumvector1_im(35) <= (32 => '0', OTHERS => '1') WHEN add_temp_71(33) =
    '0' AND add_temp_71(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_71(33) = '1' AND
        add_temp_71(32) /= '1'
    ELSE (add_temp_71(32 DOWNTO 0));

add_cast_144 <= resize(product_pipeline_phase0_13_re, 33);
add_cast_145 <= resize(product_pipeline_phase0_14_re, 33);
add_temp_72 <= resize(add_cast_144, 34) + resize(add_cast_145, 34);
sumvector1_re(36) <= (32 => '0', OTHERS => '1') WHEN add_temp_72(33) =
    '0' AND add_temp_72(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_72(33) = '1' AND
        add_temp_72(32) /= '1'
    ELSE (add_temp_72(32 DOWNTO 0));

add_cast_146 <= resize(product_pipeline_phase0_13_im, 33);
add_cast_147 <= resize(product_pipeline_phase0_14_im, 33);
add_temp_73 <= resize(add_cast_146, 34) + resize(add_cast_147, 34);
sumvector1_im(36) <= (32 => '0', OTHERS => '1') WHEN add_temp_73(33) =
    '0' AND add_temp_73(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_73(33) = '1' AND
        add_temp_73(32) /= '1'
    ELSE (add_temp_73(32 DOWNTO 0));

add_cast_148 <= resize(product_pipeline_phase0_15_re, 33);
add_cast_149 <= resize(product_pipeline_phase0_16_re, 33);
add_temp_74 <= resize(add_cast_148, 34) + resize(add_cast_149, 34);
sumvector1_re(37) <= (32 => '0', OTHERS => '1') WHEN add_temp_74(33) =
    '0' AND add_temp_74(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_74(33) = '1' AND
        add_temp_74(32) /= '1'
```

```
    ELSE (add_temp_74(32 DOWNTO 0));

add_cast_150 <= resize(product_pipeline_phase0_15_im, 33);
add_cast_151 <= resize(product_pipeline_phase0_16_im, 33);
add_temp_75 <= resize(add_cast_150, 34) + resize(add_cast_151, 34);
sumvector1_im(37) <= (32 => '0', OTHERS => '1') WHEN add_temp_75(33) =
     '0' AND add_temp_75(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_75(33) = '1' AND
        add_temp_75(32) /= '1'
    ELSE (add_temp_75(32 DOWNTO 0));

add_cast_152 <= resize(product_pipeline_phase0_17_re, 33);
add_cast_153 <= resize(product_pipeline_phase0_18_re, 33);
add_temp_76 <= resize(add_cast_152, 34) + resize(add_cast_153, 34);
sumvector1_re(38) <= (32 => '0', OTHERS => '1') WHEN add_temp_76(33) =
     '0' AND add_temp_76(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_76(33) = '1' AND
        add_temp_76(32) /= '1'
    ELSE (add_temp_76(32 DOWNTO 0));

add_cast_154 <= resize(product_pipeline_phase0_17_im, 33);
add_cast_155 <= resize(product_pipeline_phase0_18_im, 33);
add_temp_77 <= resize(add_cast_154, 34) + resize(add_cast_155, 34);
sumvector1_im(38) <= (32 => '0', OTHERS => '1') WHEN add_temp_77(33) =
     '0' AND add_temp_77(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_77(33) = '1' AND
        add_temp_77(32) /= '1'
    ELSE (add_temp_77(32 DOWNTO 0));

add_cast_156 <= resize(product_pipeline_phase0_19_re, 33);
add_cast_157 <= resize(product_pipeline_phase0_20_re, 33);
add_temp_78 <= resize(add_cast_156, 34) + resize(add_cast_157, 34);
sumvector1_re(39) <= (32 => '0', OTHERS => '1') WHEN add_temp_78(33) =
```

```
      '0' AND add_temp_78(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_78(33) = '1' AND
        add_temp_78(32) /= '1'
    ELSE (add_temp_78(32 DOWNTO 0));

add_cast_158 <= resize(product_pipeline_phase0_19_im, 33);
add_cast_159 <= resize(product_pipeline_phase0_20_im, 33);
add_temp_79 <= resize(add_cast_158, 34) + resize(add_cast_159, 34);
sumvector1_im(39) <= (32 => '0', OTHERS => '1') WHEN add_temp_79(33) =
     '0' AND add_temp_79(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_79(33) = '1' AND
        add_temp_79(32) /= '1'
    ELSE (add_temp_79(32 DOWNTO 0));

sumvector1_re(40) <= resize(product_pipeline_phase0_21_re, 33);
sumvector1_im(40) <= resize(product_pipeline_phase0_21_im, 33);

sumdelay_pipeline_process1 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    sumdelay_pipeline1_re <= (OTHERS => (OTHERS => '0'));
    sumdelay_pipeline1_im <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF phase_0 = '1' THEN
      sumdelay_pipeline1_re(0 TO 40) <= sumvector1_re(0 TO 40);
      sumdelay_pipeline1_im(0 TO 40) <= sumvector1_im(0 TO 40);
    END IF;
  END IF;
END PROCESS sumdelay_pipeline_process1;

add_cast_160 <= sumdelay_pipeline1_re(0);
add_cast_161 <= sumdelay_pipeline1_re(1);
add_temp_80 <= resize(add_cast_160, 34) + resize(add_cast_161, 34);
```

```
sumvector2_re(0) <= (32 => '0', OTHERS => '1') WHEN add_temp_80(33) =
    '0' AND add_temp_80(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_80(33) = '1' AND
        add_temp_80(32) /= '1'
    ELSE (add_temp_80(32 DOWNTO 0));

add_cast_162 <= sumdelay_pipeline1_im(0);
add_cast_163 <= sumdelay_pipeline1_im(1);
add_temp_81 <= resize(add_cast_162, 34) + resize(add_cast_163, 34);
sumvector2_im(0) <= (32 => '0', OTHERS => '1') WHEN add_temp_81(33) =
    '0' AND add_temp_81(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_81(33) = '1' AND
        add_temp_81(32) /= '1'
    ELSE (add_temp_81(32 DOWNTO 0));

add_cast_164 <= sumdelay_pipeline1_re(2);
add_cast_165 <= sumdelay_pipeline1_re(3);
add_temp_82 <= resize(add_cast_164, 34) + resize(add_cast_165, 34);
sumvector2_re(1) <= (32 => '0', OTHERS => '1') WHEN add_temp_82(33) =
    '0' AND add_temp_82(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_82(33) = '1' AND
        add_temp_82(32) /= '1'
    ELSE (add_temp_82(32 DOWNTO 0));

add_cast_166 <= sumdelay_pipeline1_im(2);
add_cast_167 <= sumdelay_pipeline1_im(3);
add_temp_83 <= resize(add_cast_166, 34) + resize(add_cast_167, 34);
sumvector2_im(1) <= (32 => '0', OTHERS => '1') WHEN add_temp_83(33) =
    '0' AND add_temp_83(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_83(33) = '1' AND
        add_temp_83(32) /= '1'
    ELSE (add_temp_83(32 DOWNTO 0));
```

```
add_cast_168 <= sumdelay_pipeline1_re(4);
add_cast_169 <= sumdelay_pipeline1_re(5);
add_temp_84 <= resize(add_cast_168, 34) + resize(add_cast_169, 34);
sumvector2_re(2) <= (32 => '0', OTHERS => '1') WHEN add_temp_84(33) =
    '0' AND add_temp_84(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_84(33) = '1' AND
        add_temp_84(32) /= '1'
    ELSE (add_temp_84(32 DOWNTO 0));

add_cast_170 <= sumdelay_pipeline1_im(4);
add_cast_171 <= sumdelay_pipeline1_im(5);
add_temp_85 <= resize(add_cast_170, 34) + resize(add_cast_171, 34);
sumvector2_im(2) <= (32 => '0', OTHERS => '1') WHEN add_temp_85(33) =
    '0' AND add_temp_85(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_85(33) = '1' AND
        add_temp_85(32) /= '1'
    ELSE (add_temp_85(32 DOWNTO 0));

add_cast_172 <= sumdelay_pipeline1_re(6);
add_cast_173 <= sumdelay_pipeline1_re(7);
add_temp_86 <= resize(add_cast_172, 34) + resize(add_cast_173, 34);
sumvector2_re(3) <= (32 => '0', OTHERS => '1') WHEN add_temp_86(33) =
    '0' AND add_temp_86(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_86(33) = '1' AND
        add_temp_86(32) /= '1'
    ELSE (add_temp_86(32 DOWNTO 0));

add_cast_174 <= sumdelay_pipeline1_im(6);
add_cast_175 <= sumdelay_pipeline1_im(7);
add_temp_87 <= resize(add_cast_174, 34) + resize(add_cast_175, 34);
sumvector2_im(3) <= (32 => '0', OTHERS => '1') WHEN add_temp_87(33) =
    '0' AND add_temp_87(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_87(33) = '1' AND
```

```
        add_temp_87(32) /= '1'
    ELSE (add_temp_87(32 DOWNTO 0));

add_cast_176 <= sumdelay_pipeline1_re(8);
add_cast_177 <= sumdelay_pipeline1_re(9);
add_temp_88 <= resize(add_cast_176, 34) + resize(add_cast_177, 34);
sumvector2_re(4) <= (32 => '0', OTHERS => '1') WHEN add_temp_88(33) =
    '0' AND add_temp_88(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_88(33) = '1' AND
        add_temp_88(32) /= '1'
    ELSE (add_temp_88(32 DOWNTO 0));

add_cast_178 <= sumdelay_pipeline1_im(8);
add_cast_179 <= sumdelay_pipeline1_im(9);
add_temp_89 <= resize(add_cast_178, 34) + resize(add_cast_179, 34);
sumvector2_im(4) <= (32 => '0', OTHERS => '1') WHEN add_temp_89(33) =
    '0' AND add_temp_89(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_89(33) = '1' AND
        add_temp_89(32) /= '1'
    ELSE (add_temp_89(32 DOWNTO 0));

add_cast_180 <= sumdelay_pipeline1_re(10);
add_cast_181 <= sumdelay_pipeline1_re(11);
add_temp_90 <= resize(add_cast_180, 34) + resize(add_cast_181, 34);
sumvector2_re(5) <= (32 => '0', OTHERS => '1') WHEN add_temp_90(33) =
    '0' AND add_temp_90(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_90(33) = '1' AND
        add_temp_90(32) /= '1'
    ELSE (add_temp_90(32 DOWNTO 0));

add_cast_182 <= sumdelay_pipeline1_im(10);
add_cast_183 <= sumdelay_pipeline1_im(11);
add_temp_91 <= resize(add_cast_182, 34) + resize(add_cast_183, 34);
```

```
sumvector2_im(5) <= (32 => '0', OTHERS => '1') WHEN add_temp_91(33) =
    '0' AND add_temp_91(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_91(33) = '1' AND
        add_temp_91(32) /= '1'
    ELSE (add_temp_91(32 DOWNTO 0));

add_cast_184 <= sumdelay_pipeline1_re(12);
add_cast_185 <= sumdelay_pipeline1_re(13);
add_temp_92 <= resize(add_cast_184, 34) + resize(add_cast_185, 34);
sumvector2_re(6) <= (32 => '0', OTHERS => '1') WHEN add_temp_92(33) =
    '0' AND add_temp_92(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_92(33) = '1' AND
        add_temp_92(32) /= '1'
    ELSE (add_temp_92(32 DOWNTO 0));

add_cast_186 <= sumdelay_pipeline1_im(12);
add_cast_187 <= sumdelay_pipeline1_im(13);
add_temp_93 <= resize(add_cast_186, 34) + resize(add_cast_187, 34);
sumvector2_im(6) <= (32 => '0', OTHERS => '1') WHEN add_temp_93(33) =
    '0' AND add_temp_93(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_93(33) = '1' AND
        add_temp_93(32) /= '1'
    ELSE (add_temp_93(32 DOWNTO 0));

add_cast_188 <= sumdelay_pipeline1_re(14);
add_cast_189 <= sumdelay_pipeline1_re(15);
add_temp_94 <= resize(add_cast_188, 34) + resize(add_cast_189, 34);
sumvector2_re(7) <= (32 => '0', OTHERS => '1') WHEN add_temp_94(33) =
    '0' AND add_temp_94(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_94(33) = '1' AND
        add_temp_94(32) /= '1'
    ELSE (add_temp_94(32 DOWNTO 0));
```

```
add_cast_190 <= sumdelay_pipeline1_im(14);
add_cast_191 <= sumdelay_pipeline1_im(15);
add_temp_95 <= resize(add_cast_190, 34) + resize(add_cast_191, 34);
sumvector2_im(7) <= (32 => '0', OTHERS => '1') WHEN add_temp_95(33) =
    '0' AND add_temp_95(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_95(33) = '1' AND
        add_temp_95(32) /= '1'
    ELSE (add_temp_95(32 DOWNTO 0));

add_cast_192 <= sumdelay_pipeline1_re(16);
add_cast_193 <= sumdelay_pipeline1_re(17);
add_temp_96 <= resize(add_cast_192, 34) + resize(add_cast_193, 34);
sumvector2_re(8) <= (32 => '0', OTHERS => '1') WHEN add_temp_96(33) =
    '0' AND add_temp_96(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_96(33) = '1' AND
        add_temp_96(32) /= '1'
    ELSE (add_temp_96(32 DOWNTO 0));

add_cast_194 <= sumdelay_pipeline1_im(16);
add_cast_195 <= sumdelay_pipeline1_im(17);
add_temp_97 <= resize(add_cast_194, 34) + resize(add_cast_195, 34);
sumvector2_im(8) <= (32 => '0', OTHERS => '1') WHEN add_temp_97(33) =
    '0' AND add_temp_97(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_97(33) = '1' AND
        add_temp_97(32) /= '1'
    ELSE (add_temp_97(32 DOWNTO 0));

add_cast_196 <= sumdelay_pipeline1_re(18);
add_cast_197 <= sumdelay_pipeline1_re(19);
add_temp_98 <= resize(add_cast_196, 34) + resize(add_cast_197, 34);
sumvector2_re(9) <= (32 => '0', OTHERS => '1') WHEN add_temp_98(33) =
    '0' AND add_temp_98(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_98(33) = '1' AND
```

```
        add_temp_98(32) /= '1'
    ELSE (add_temp_98(32 DOWNTO 0));

add_cast_198 <= sumdelay_pipeline1_im(18);
add_cast_199 <= sumdelay_pipeline1_im(19);
add_temp_99 <= resize(add_cast_198, 34) + resize(add_cast_199, 34);
sumvector2_im(9) <= (32 => '0', OTHERS => '1') WHEN add_temp_99(33) =
    '0' AND add_temp_99(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_99(33) = '1' AND
        add_temp_99(32) /= '1'
    ELSE (add_temp_99(32 DOWNTO 0));

add_cast_200 <= sumdelay_pipeline1_re(20);
add_cast_201 <= sumdelay_pipeline1_re(21);
add_temp_100 <= resize(add_cast_200, 34) + resize(add_cast_201, 34);
sumvector2_re(10) <= (32 => '0', OTHERS => '1') WHEN add_temp_100(33)
   = '0' AND add_temp_100(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_100(33) = '1' AND
        add_temp_100(32) /= '1'
    ELSE (add_temp_100(32 DOWNTO 0));

add_cast_202 <= sumdelay_pipeline1_im(20);
add_cast_203 <= sumdelay_pipeline1_im(21);
add_temp_101 <= resize(add_cast_202, 34) + resize(add_cast_203, 34);
sumvector2_im(10) <= (32 => '0', OTHERS => '1') WHEN add_temp_101(33)
   = '0' AND add_temp_101(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_101(33) = '1' AND
        add_temp_101(32) /= '1'
    ELSE (add_temp_101(32 DOWNTO 0));

add_cast_204 <= sumdelay_pipeline1_re(22);
add_cast_205 <= sumdelay_pipeline1_re(23);
add_temp_102 <= resize(add_cast_204, 34) + resize(add_cast_205, 34);
```

```
sumvector2_re(11) <= (32 => '0', OTHERS => '1') WHEN add_temp_102(33)
    = '0' AND add_temp_102(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_102(33) = '1' AND
        add_temp_102(32) /= '1'
    ELSE (add_temp_102(32 DOWNTO 0));

add_cast_206 <= sumdelay_pipeline1_im(22);
add_cast_207 <= sumdelay_pipeline1_im(23);
add_temp_103 <= resize(add_cast_206, 34) + resize(add_cast_207, 34);
sumvector2_im(11) <= (32 => '0', OTHERS => '1') WHEN add_temp_103(33)
    = '0' AND add_temp_103(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_103(33) = '1' AND
        add_temp_103(32) /= '1'
    ELSE (add_temp_103(32 DOWNTO 0));

add_cast_208 <= sumdelay_pipeline1_re(24);
add_cast_209 <= sumdelay_pipeline1_re(25);
add_temp_104 <= resize(add_cast_208, 34) + resize(add_cast_209, 34);
sumvector2_re(12) <= (32 => '0', OTHERS => '1') WHEN add_temp_104(33)
    = '0' AND add_temp_104(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_104(33) = '1' AND
        add_temp_104(32) /= '1'
    ELSE (add_temp_104(32 DOWNTO 0));

add_cast_210 <= sumdelay_pipeline1_im(24);
add_cast_211 <= sumdelay_pipeline1_im(25);
add_temp_105 <= resize(add_cast_210, 34) + resize(add_cast_211, 34);
sumvector2_im(12) <= (32 => '0', OTHERS => '1') WHEN add_temp_105(33)
    = '0' AND add_temp_105(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_105(33) = '1' AND
        add_temp_105(32) /= '1'
    ELSE (add_temp_105(32 DOWNTO 0));
```

```
add_cast_212 <= sumdelay_pipeline1_re(26);
add_cast_213 <= sumdelay_pipeline1_re(27);
add_temp_106 <= resize(add_cast_212, 34) + resize(add_cast_213, 34);
sumvector2_re(13) <= (32 => '0', OTHERS => '1') WHEN add_temp_106(33)
   = '0' AND add_temp_106(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_106(33) = '1' AND
        add_temp_106(32) /= '1'
    ELSE (add_temp_106(32 DOWNTO 0));

add_cast_214 <= sumdelay_pipeline1_im(26);
add_cast_215 <= sumdelay_pipeline1_im(27);
add_temp_107 <= resize(add_cast_214, 34) + resize(add_cast_215, 34);
sumvector2_im(13) <= (32 => '0', OTHERS => '1') WHEN add_temp_107(33)
   = '0' AND add_temp_107(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_107(33) = '1' AND
        add_temp_107(32) /= '1'
    ELSE (add_temp_107(32 DOWNTO 0));

add_cast_216 <= sumdelay_pipeline1_re(28);
add_cast_217 <= sumdelay_pipeline1_re(29);
add_temp_108 <= resize(add_cast_216, 34) + resize(add_cast_217, 34);
sumvector2_re(14) <= (32 => '0', OTHERS => '1') WHEN add_temp_108(33)
   = '0' AND add_temp_108(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_108(33) = '1' AND
        add_temp_108(32) /= '1'
    ELSE (add_temp_108(32 DOWNTO 0));

add_cast_218 <= sumdelay_pipeline1_im(28);
add_cast_219 <= sumdelay_pipeline1_im(29);
add_temp_109 <= resize(add_cast_218, 34) + resize(add_cast_219, 34);
sumvector2_im(14) <= (32 => '0', OTHERS => '1') WHEN add_temp_109(33)
   = '0' AND add_temp_109(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_109(33) = '1' AND
```

```
        add_temp_109(32) /= '1'
    ELSE (add_temp_109(32 DOWNTO 0));

add_cast_220 <= sumdelay_pipeline1_re(30);
add_cast_221 <= sumdelay_pipeline1_re(31);
add_temp_110 <= resize(add_cast_220, 34) + resize(add_cast_221, 34);
sumvector2_re(15) <= (32 => '0', OTHERS => '1') WHEN add_temp_110(33)
   = '0' AND add_temp_110(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_110(33) = '1' AND
        add_temp_110(32) /= '1'
    ELSE (add_temp_110(32 DOWNTO 0));

add_cast_222 <= sumdelay_pipeline1_im(30);
add_cast_223 <= sumdelay_pipeline1_im(31);
add_temp_111 <= resize(add_cast_222, 34) + resize(add_cast_223, 34);
sumvector2_im(15) <= (32 => '0', OTHERS => '1') WHEN add_temp_111(33)
   = '0' AND add_temp_111(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_111(33) = '1' AND
        add_temp_111(32) /= '1'
    ELSE (add_temp_111(32 DOWNTO 0));

add_cast_224 <= sumdelay_pipeline1_re(32);
add_cast_225 <= sumdelay_pipeline1_re(33);
add_temp_112 <= resize(add_cast_224, 34) + resize(add_cast_225, 34);
sumvector2_re(16) <= (32 => '0', OTHERS => '1') WHEN add_temp_112(33)
   = '0' AND add_temp_112(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_112(33) = '1' AND
        add_temp_112(32) /= '1'
    ELSE (add_temp_112(32 DOWNTO 0));

add_cast_226 <= sumdelay_pipeline1_im(32);
add_cast_227 <= sumdelay_pipeline1_im(33);
add_temp_113 <= resize(add_cast_226, 34) + resize(add_cast_227, 34);
```

```
sumvector2_im(16) <= (32 => '0', OTHERS => '1') WHEN add_temp_113(33)
    = '0' AND add_temp_113(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_113(33) = '1' AND
        add_temp_113(32) /= '1'
    ELSE (add_temp_113(32 DOWNTO 0));

add_cast_228 <= sumdelay_pipeline1_re(34);
add_cast_229 <= sumdelay_pipeline1_re(35);
add_temp_114 <= resize(add_cast_228, 34) + resize(add_cast_229, 34);
sumvector2_re(17) <= (32 => '0', OTHERS => '1') WHEN add_temp_114(33)
    = '0' AND add_temp_114(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_114(33) = '1' AND
        add_temp_114(32) /= '1'
    ELSE (add_temp_114(32 DOWNTO 0));

add_cast_230 <= sumdelay_pipeline1_im(34);
add_cast_231 <= sumdelay_pipeline1_im(35);
add_temp_115 <= resize(add_cast_230, 34) + resize(add_cast_231, 34);
sumvector2_im(17) <= (32 => '0', OTHERS => '1') WHEN add_temp_115(33)
    = '0' AND add_temp_115(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_115(33) = '1' AND
        add_temp_115(32) /= '1'
    ELSE (add_temp_115(32 DOWNTO 0));

add_cast_232 <= sumdelay_pipeline1_re(36);
add_cast_233 <= sumdelay_pipeline1_re(37);
add_temp_116 <= resize(add_cast_232, 34) + resize(add_cast_233, 34);
sumvector2_re(18) <= (32 => '0', OTHERS => '1') WHEN add_temp_116(33)
    = '0' AND add_temp_116(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_116(33) = '1' AND
        add_temp_116(32) /= '1'
    ELSE (add_temp_116(32 DOWNTO 0));
```

```
add_cast_234 <= sumdelay_pipeline1_im(36);
add_cast_235 <= sumdelay_pipeline1_im(37);
add_temp_117 <= resize(add_cast_234, 34) + resize(add_cast_235, 34);
sumvector2_im(18) <= (32 => '0', OTHERS => '1') WHEN add_temp_117(33)
    = '0' AND add_temp_117(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_117(33) = '1' AND
        add_temp_117(32) /= '1'
    ELSE (add_temp_117(32 DOWNTO 0));

add_cast_236 <= sumdelay_pipeline1_re(38);
add_cast_237 <= sumdelay_pipeline1_re(39);
add_temp_118 <= resize(add_cast_236, 34) + resize(add_cast_237, 34);
sumvector2_re(19) <= (32 => '0', OTHERS => '1') WHEN add_temp_118(33)
    = '0' AND add_temp_118(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_118(33) = '1' AND
        add_temp_118(32) /= '1'
    ELSE (add_temp_118(32 DOWNTO 0));

add_cast_238 <= sumdelay_pipeline1_im(38);
add_cast_239 <= sumdelay_pipeline1_im(39);
add_temp_119 <= resize(add_cast_238, 34) + resize(add_cast_239, 34);
sumvector2_im(19) <= (32 => '0', OTHERS => '1') WHEN add_temp_119(33)
    = '0' AND add_temp_119(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_119(33) = '1' AND
        add_temp_119(32) /= '1'
    ELSE (add_temp_119(32 DOWNTO 0));

sumvector2_re(20) <= sumdelay_pipeline1_re(40);
sumvector2_im(20) <= sumdelay_pipeline1_im(40);

sumdelay_pipeline_process2 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
```

```
      sumdelay_pipeline2_re <= (OTHERS => (OTHERS => '0'));
      sumdelay_pipeline2_im <= (OTHERS => (OTHERS => '0'));
    ELSIF clk'event AND clk = '1' THEN
      IF phase_0 = '1' THEN
        sumdelay_pipeline2_re(0 TO 20) <= sumvector2_re(0 TO 20);
        sumdelay_pipeline2_im(0 TO 20) <= sumvector2_im(0 TO 20);
      END IF;
    END IF;
  END PROCESS sumdelay_pipeline_process2;

  add_cast_240 <= sumdelay_pipeline2_re(0);
  add_cast_241 <= sumdelay_pipeline2_re(1);
  add_temp_120 <= resize(add_cast_240, 34) + resize(add_cast_241, 34);
  sumvector3_re(0) <= (32 => '0', OTHERS => '1') WHEN add_temp_120(33) =
      '0' AND add_temp_120(32) /= '0'
      ELSE (32 => '1', OTHERS => '0') WHEN add_temp_120(33) = '1' AND
          add_temp_120(32) /= '1'
      ELSE (add_temp_120(32 DOWNTO 0));

  add_cast_242 <= sumdelay_pipeline2_im(0);
  add_cast_243 <= sumdelay_pipeline2_im(1);
  add_temp_121 <= resize(add_cast_242, 34) + resize(add_cast_243, 34);
  sumvector3_im(0) <= (32 => '0', OTHERS => '1') WHEN add_temp_121(33) =
      '0' AND add_temp_121(32) /= '0'
      ELSE (32 => '1', OTHERS => '0') WHEN add_temp_121(33) = '1' AND
          add_temp_121(32) /= '1'
      ELSE (add_temp_121(32 DOWNTO 0));

  add_cast_244 <= sumdelay_pipeline2_re(2);
  add_cast_245 <= sumdelay_pipeline2_re(3);
  add_temp_122 <= resize(add_cast_244, 34) + resize(add_cast_245, 34);
  sumvector3_re(1) <= (32 => '0', OTHERS => '1') WHEN add_temp_122(33) =
      '0' AND add_temp_122(32) /= '0'
```

```
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_122(33) = '1' AND
        add_temp_122(32) /= '1'
    ELSE (add_temp_122(32 DOWNTO 0));

add_cast_246 <= sumdelay_pipeline2_im(2);
add_cast_247 <= sumdelay_pipeline2_im(3);
add_temp_123 <= resize(add_cast_246, 34) + resize(add_cast_247, 34);
sumvector3_im(1) <= (32 => '0', OTHERS => '1') WHEN add_temp_123(33) =
     '0' AND add_temp_123(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_123(33) = '1' AND
        add_temp_123(32) /= '1'
    ELSE (add_temp_123(32 DOWNTO 0));

add_cast_248 <= sumdelay_pipeline2_re(4);
add_cast_249 <= sumdelay_pipeline2_re(5);
add_temp_124 <= resize(add_cast_248, 34) + resize(add_cast_249, 34);
sumvector3_re(2) <= (32 => '0', OTHERS => '1') WHEN add_temp_124(33) =
     '0' AND add_temp_124(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_124(33) = '1' AND
        add_temp_124(32) /= '1'
    ELSE (add_temp_124(32 DOWNTO 0));

add_cast_250 <= sumdelay_pipeline2_im(4);
add_cast_251 <= sumdelay_pipeline2_im(5);
add_temp_125 <= resize(add_cast_250, 34) + resize(add_cast_251, 34);
sumvector3_im(2) <= (32 => '0', OTHERS => '1') WHEN add_temp_125(33) =
     '0' AND add_temp_125(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_125(33) = '1' AND
        add_temp_125(32) /= '1'
    ELSE (add_temp_125(32 DOWNTO 0));

add_cast_252 <= sumdelay_pipeline2_re(6);
add_cast_253 <= sumdelay_pipeline2_re(7);
```

```
add_temp_126 <= resize(add_cast_252, 34) + resize(add_cast_253, 34);
sumvector3_re(3) <= (32 => '0', OTHERS => '1') WHEN add_temp_126(33) =
    '0' AND add_temp_126(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_126(33) = '1' AND
        add_temp_126(32) /= '1'
    ELSE (add_temp_126(32 DOWNTO 0));

add_cast_254 <= sumdelay_pipeline2_im(6);
add_cast_255 <= sumdelay_pipeline2_im(7);
add_temp_127 <= resize(add_cast_254, 34) + resize(add_cast_255, 34);
sumvector3_im(3) <= (32 => '0', OTHERS => '1') WHEN add_temp_127(33) =
    '0' AND add_temp_127(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_127(33) = '1' AND
        add_temp_127(32) /= '1'
    ELSE (add_temp_127(32 DOWNTO 0));

add_cast_256 <= sumdelay_pipeline2_re(8);
add_cast_257 <= sumdelay_pipeline2_re(9);
add_temp_128 <= resize(add_cast_256, 34) + resize(add_cast_257, 34);
sumvector3_re(4) <= (32 => '0', OTHERS => '1') WHEN add_temp_128(33) =
    '0' AND add_temp_128(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_128(33) = '1' AND
        add_temp_128(32) /= '1'
    ELSE (add_temp_128(32 DOWNTO 0));

add_cast_258 <= sumdelay_pipeline2_im(8);
add_cast_259 <= sumdelay_pipeline2_im(9);
add_temp_129 <= resize(add_cast_258, 34) + resize(add_cast_259, 34);
sumvector3_im(4) <= (32 => '0', OTHERS => '1') WHEN add_temp_129(33) =
    '0' AND add_temp_129(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_129(33) = '1' AND
        add_temp_129(32) /= '1'
    ELSE (add_temp_129(32 DOWNTO 0));
```

```
add_cast_260 <= sumdelay_pipeline2_re(10);
add_cast_261 <= sumdelay_pipeline2_re(11);
add_temp_130 <= resize(add_cast_260, 34) + resize(add_cast_261, 34);
sumvector3_re(5) <= (32 => '0', OTHERS => '1') WHEN add_temp_130(33) =
    '0' AND add_temp_130(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_130(33) = '1' AND
        add_temp_130(32) /= '1'
    ELSE (add_temp_130(32 DOWNTO 0));

add_cast_262 <= sumdelay_pipeline2_im(10);
add_cast_263 <= sumdelay_pipeline2_im(11);
add_temp_131 <= resize(add_cast_262, 34) + resize(add_cast_263, 34);
sumvector3_im(5) <= (32 => '0', OTHERS => '1') WHEN add_temp_131(33) =
    '0' AND add_temp_131(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_131(33) = '1' AND
        add_temp_131(32) /= '1'
    ELSE (add_temp_131(32 DOWNTO 0));

add_cast_264 <= sumdelay_pipeline2_re(12);
add_cast_265 <= sumdelay_pipeline2_re(13);
add_temp_132 <= resize(add_cast_264, 34) + resize(add_cast_265, 34);
sumvector3_re(6) <= (32 => '0', OTHERS => '1') WHEN add_temp_132(33) =
    '0' AND add_temp_132(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_132(33) = '1' AND
        add_temp_132(32) /= '1'
    ELSE (add_temp_132(32 DOWNTO 0));

add_cast_266 <= sumdelay_pipeline2_im(12);
add_cast_267 <= sumdelay_pipeline2_im(13);
add_temp_133 <= resize(add_cast_266, 34) + resize(add_cast_267, 34);
sumvector3_im(6) <= (32 => '0', OTHERS => '1') WHEN add_temp_133(33) =
    '0' AND add_temp_133(32) /= '0'
```

```
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_133(33) = '1' AND
        add_temp_133(32) /= '1'
    ELSE (add_temp_133(32 DOWNTO 0));

add_cast_268 <= sumdelay_pipeline2_re(14);
add_cast_269 <= sumdelay_pipeline2_re(15);
add_temp_134 <= resize(add_cast_268, 34) + resize(add_cast_269, 34);
sumvector3_re(7) <= (32 => '0', OTHERS => '1') WHEN add_temp_134(33) =
     '0' AND add_temp_134(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_134(33) = '1' AND
        add_temp_134(32) /= '1'
    ELSE (add_temp_134(32 DOWNTO 0));

add_cast_270 <= sumdelay_pipeline2_im(14);
add_cast_271 <= sumdelay_pipeline2_im(15);
add_temp_135 <= resize(add_cast_270, 34) + resize(add_cast_271, 34);
sumvector3_im(7) <= (32 => '0', OTHERS => '1') WHEN add_temp_135(33) =
     '0' AND add_temp_135(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_135(33) = '1' AND
        add_temp_135(32) /= '1'
    ELSE (add_temp_135(32 DOWNTO 0));

add_cast_272 <= sumdelay_pipeline2_re(16);
add_cast_273 <= sumdelay_pipeline2_re(17);
add_temp_136 <= resize(add_cast_272, 34) + resize(add_cast_273, 34);
sumvector3_re(8) <= (32 => '0', OTHERS => '1') WHEN add_temp_136(33) =
     '0' AND add_temp_136(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_136(33) = '1' AND
        add_temp_136(32) /= '1'
    ELSE (add_temp_136(32 DOWNTO 0));

add_cast_274 <= sumdelay_pipeline2_im(16);
add_cast_275 <= sumdelay_pipeline2_im(17);
```

```
add_temp_137 <= resize(add_cast_274, 34) + resize(add_cast_275, 34);
sumvector3_im(8) <= (32 => '0', OTHERS => '1') WHEN add_temp_137(33) =
    '0' AND add_temp_137(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_137(33) = '1' AND
        add_temp_137(32) /= '1'
    ELSE (add_temp_137(32 DOWNTO 0));

add_cast_276 <= sumdelay_pipeline2_re(18);
add_cast_277 <= sumdelay_pipeline2_re(19);
add_temp_138 <= resize(add_cast_276, 34) + resize(add_cast_277, 34);
sumvector3_re(9) <= (32 => '0', OTHERS => '1') WHEN add_temp_138(33) =
    '0' AND add_temp_138(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_138(33) = '1' AND
        add_temp_138(32) /= '1'
    ELSE (add_temp_138(32 DOWNTO 0));

add_cast_278 <= sumdelay_pipeline2_im(18);
add_cast_279 <= sumdelay_pipeline2_im(19);
add_temp_139 <= resize(add_cast_278, 34) + resize(add_cast_279, 34);
sumvector3_im(9) <= (32 => '0', OTHERS => '1') WHEN add_temp_139(33) =
    '0' AND add_temp_139(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_139(33) = '1' AND
        add_temp_139(32) /= '1'
    ELSE (add_temp_139(32 DOWNTO 0));

sumvector3_re(10) <= sumdelay_pipeline2_re(20);
sumvector3_im(10) <= sumdelay_pipeline2_im(20);

sumdelay_pipeline_process3 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    sumdelay_pipeline3_re <= (OTHERS => (OTHERS => '0'));
    sumdelay_pipeline3_im <= (OTHERS => (OTHERS => '0'));
```

```
  ELSIF clk'event AND clk = '1' THEN
    IF phase_0 = '1' THEN
      sumdelay_pipeline3_re(0 TO 10) <= sumvector3_re(0 TO 10);
      sumdelay_pipeline3_im(0 TO 10) <= sumvector3_im(0 TO 10);
    END IF;
  END IF;
END PROCESS sumdelay_pipeline_process3;

add_cast_280 <= sumdelay_pipeline3_re(0);
add_cast_281 <= sumdelay_pipeline3_re(1);
add_temp_140 <= resize(add_cast_280, 34) + resize(add_cast_281, 34);
sumvector4_re(0) <= (32 => '0', OTHERS => '1') WHEN add_temp_140(33) =
    '0' AND add_temp_140(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_140(33) = '1' AND
        add_temp_140(32) /= '1'
    ELSE (add_temp_140(32 DOWNTO 0));

add_cast_282 <= sumdelay_pipeline3_im(0);
add_cast_283 <= sumdelay_pipeline3_im(1);
add_temp_141 <= resize(add_cast_282, 34) + resize(add_cast_283, 34);
sumvector4_im(0) <= (32 => '0', OTHERS => '1') WHEN add_temp_141(33) =
    '0' AND add_temp_141(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_141(33) = '1' AND
        add_temp_141(32) /= '1'
    ELSE (add_temp_141(32 DOWNTO 0));

add_cast_284 <= sumdelay_pipeline3_re(2);
add_cast_285 <= sumdelay_pipeline3_re(3);
add_temp_142 <= resize(add_cast_284, 34) + resize(add_cast_285, 34);
sumvector4_re(1) <= (32 => '0', OTHERS => '1') WHEN add_temp_142(33) =
    '0' AND add_temp_142(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_142(33) = '1' AND
        add_temp_142(32) /= '1'
```

```
    ELSE (add_temp_142(32 DOWNTO 0));

add_cast_286 <= sumdelay_pipeline3_im(2);
add_cast_287 <= sumdelay_pipeline3_im(3);
add_temp_143 <= resize(add_cast_286, 34) + resize(add_cast_287, 34);
sumvector4_im(1) <= (32 => '0', OTHERS => '1') WHEN add_temp_143(33) =
     '0' AND add_temp_143(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_143(33) = '1' AND
        add_temp_143(32) /= '1'
    ELSE (add_temp_143(32 DOWNTO 0));

add_cast_288 <= sumdelay_pipeline3_re(4);
add_cast_289 <= sumdelay_pipeline3_re(5);
add_temp_144 <= resize(add_cast_288, 34) + resize(add_cast_289, 34);
sumvector4_re(2) <= (32 => '0', OTHERS => '1') WHEN add_temp_144(33) =
     '0' AND add_temp_144(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_144(33) = '1' AND
        add_temp_144(32) /= '1'
    ELSE (add_temp_144(32 DOWNTO 0));

add_cast_290 <= sumdelay_pipeline3_im(4);
add_cast_291 <= sumdelay_pipeline3_im(5);
add_temp_145 <= resize(add_cast_290, 34) + resize(add_cast_291, 34);
sumvector4_im(2) <= (32 => '0', OTHERS => '1') WHEN add_temp_145(33) =
     '0' AND add_temp_145(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_145(33) = '1' AND
        add_temp_145(32) /= '1'
    ELSE (add_temp_145(32 DOWNTO 0));

add_cast_292 <= sumdelay_pipeline3_re(6);
add_cast_293 <= sumdelay_pipeline3_re(7);
add_temp_146 <= resize(add_cast_292, 34) + resize(add_cast_293, 34);
sumvector4_re(3) <= (32 => '0', OTHERS => '1') WHEN add_temp_146(33) =
```

```
     '0' AND add_temp_146(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_146(33) = '1' AND
        add_temp_146(32) /= '1'
    ELSE (add_temp_146(32 DOWNTO 0));

add_cast_294 <= sumdelay_pipeline3_im(6);
add_cast_295 <= sumdelay_pipeline3_im(7);
add_temp_147 <= resize(add_cast_294, 34) + resize(add_cast_295, 34);
sumvector4_im(3) <= (32 => '0', OTHERS => '1') WHEN add_temp_147(33) =
     '0' AND add_temp_147(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_147(33) = '1' AND
        add_temp_147(32) /= '1'
    ELSE (add_temp_147(32 DOWNTO 0));

add_cast_296 <= sumdelay_pipeline3_re(8);
add_cast_297 <= sumdelay_pipeline3_re(9);
add_temp_148 <= resize(add_cast_296, 34) + resize(add_cast_297, 34);
sumvector4_re(4) <= (32 => '0', OTHERS => '1') WHEN add_temp_148(33) =
     '0' AND add_temp_148(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_148(33) = '1' AND
        add_temp_148(32) /= '1'
    ELSE (add_temp_148(32 DOWNTO 0));

add_cast_298 <= sumdelay_pipeline3_im(8);
add_cast_299 <= sumdelay_pipeline3_im(9);
add_temp_149 <= resize(add_cast_298, 34) + resize(add_cast_299, 34);
sumvector4_im(4) <= (32 => '0', OTHERS => '1') WHEN add_temp_149(33) =
     '0' AND add_temp_149(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_149(33) = '1' AND
        add_temp_149(32) /= '1'
    ELSE (add_temp_149(32 DOWNTO 0));

sumvector4_re(5) <= sumdelay_pipeline3_re(10);
```

```
sumvector4_im(5) <= sumdelay_pipeline3_im(10);

sumdelay_pipeline_process4 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    sumdelay_pipeline4_re <= (OTHERS => (OTHERS => '0'));
    sumdelay_pipeline4_im <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF phase_0 = '1' THEN
      sumdelay_pipeline4_re(0 TO 5) <= sumvector4_re(0 TO 5);
      sumdelay_pipeline4_im(0 TO 5) <= sumvector4_im(0 TO 5);
    END IF;
  END IF;
END PROCESS sumdelay_pipeline_process4;

add_cast_300 <= sumdelay_pipeline4_re(0);
add_cast_301 <= sumdelay_pipeline4_re(1);
add_temp_150 <= resize(add_cast_300, 34) + resize(add_cast_301, 34);
sumvector5_re(0) <= (32 => '0', OTHERS => '1') WHEN add_temp_150(33) =
    '0' AND add_temp_150(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_150(33) = '1' AND
        add_temp_150(32) /= '1'
    ELSE (add_temp_150(32 DOWNTO 0));

add_cast_302 <= sumdelay_pipeline4_im(0);
add_cast_303 <= sumdelay_pipeline4_im(1);
add_temp_151 <= resize(add_cast_302, 34) + resize(add_cast_303, 34);
sumvector5_im(0) <= (32 => '0', OTHERS => '1') WHEN add_temp_151(33) =
    '0' AND add_temp_151(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_151(33) = '1' AND
        add_temp_151(32) /= '1'
    ELSE (add_temp_151(32 DOWNTO 0));
```

```
add_cast_304 <= sumdelay_pipeline4_re(2);
add_cast_305 <= sumdelay_pipeline4_re(3);
add_temp_152 <= resize(add_cast_304, 34) + resize(add_cast_305, 34);
sumvector5_re(1) <= (32 => '0', OTHERS => '1') WHEN add_temp_152(33) =
    '0' AND add_temp_152(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_152(33) = '1' AND
        add_temp_152(32) /= '1'
    ELSE (add_temp_152(32 DOWNTO 0));

add_cast_306 <= sumdelay_pipeline4_im(2);
add_cast_307 <= sumdelay_pipeline4_im(3);
add_temp_153 <= resize(add_cast_306, 34) + resize(add_cast_307, 34);
sumvector5_im(1) <= (32 => '0', OTHERS => '1') WHEN add_temp_153(33) =
    '0' AND add_temp_153(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_153(33) = '1' AND
        add_temp_153(32) /= '1'
    ELSE (add_temp_153(32 DOWNTO 0));

add_cast_308 <= sumdelay_pipeline4_re(4);
add_cast_309 <= sumdelay_pipeline4_re(5);
add_temp_154 <= resize(add_cast_308, 34) + resize(add_cast_309, 34);
sumvector5_re(2) <= (32 => '0', OTHERS => '1') WHEN add_temp_154(33) =
    '0' AND add_temp_154(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_154(33) = '1' AND
        add_temp_154(32) /= '1'
    ELSE (add_temp_154(32 DOWNTO 0));

add_cast_310 <= sumdelay_pipeline4_im(4);
add_cast_311 <= sumdelay_pipeline4_im(5);
add_temp_155 <= resize(add_cast_310, 34) + resize(add_cast_311, 34);
sumvector5_im(2) <= (32 => '0', OTHERS => '1') WHEN add_temp_155(33) =
    '0' AND add_temp_155(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_155(33) = '1' AND
```

```
        add_temp_155(32) /= '1'
    ELSE (add_temp_155(32 DOWNTO 0));

sumdelay_pipeline_process5 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    sumdelay_pipeline5_re <= (OTHERS => (OTHERS => '0'));
    sumdelay_pipeline5_im <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF phase_0 = '1' THEN
      sumdelay_pipeline5_re(0 TO 2) <= sumvector5_re(0 TO 2);
      sumdelay_pipeline5_im(0 TO 2) <= sumvector5_im(0 TO 2);
    END IF;
  END IF;
END PROCESS sumdelay_pipeline_process5;

add_cast_312 <= sumdelay_pipeline5_re(0);
add_cast_313 <= sumdelay_pipeline5_re(1);
add_temp_156 <= resize(add_cast_312, 34) + resize(add_cast_313, 34);
sumvector6_re(0) <= (32 => '0', OTHERS => '1') WHEN add_temp_156(33) =
    '0' AND add_temp_156(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_156(33) = '1' AND
        add_temp_156(32) /= '1'
    ELSE (add_temp_156(32 DOWNTO 0));

add_cast_314 <= sumdelay_pipeline5_im(0);
add_cast_315 <= sumdelay_pipeline5_im(1);
add_temp_157 <= resize(add_cast_314, 34) + resize(add_cast_315, 34);
sumvector6_im(0) <= (32 => '0', OTHERS => '1') WHEN add_temp_157(33) =
    '0' AND add_temp_157(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_157(33) = '1' AND
        add_temp_157(32) /= '1'
    ELSE (add_temp_157(32 DOWNTO 0));
```

```
sumvector6_re(1) <= sumdelay_pipeline5_re(2);
sumvector6_im(1) <= sumdelay_pipeline5_im(2);

sumdelay_pipeline_process6 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    sumdelay_pipeline6_re <= (OTHERS => (OTHERS => '0'));
    sumdelay_pipeline6_im <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF phase_0 = '1' THEN
      sumdelay_pipeline6_re(0 TO 1) <= sumvector6_re(0 TO 1);
      sumdelay_pipeline6_im(0 TO 1) <= sumvector6_im(0 TO 1);
    END IF;
  END IF;
END PROCESS sumdelay_pipeline_process6;

add_cast_316 <= sumdelay_pipeline6_re(0);
add_cast_317 <= sumdelay_pipeline6_re(1);
add_temp_158 <= resize(add_cast_316, 34) + resize(add_cast_317, 34);
sum7_re <= (32 => '0', OTHERS => '1') WHEN add_temp_158(33) = '0' AND
    add_temp_158(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_158(33) = '1' AND
        add_temp_158(32) /= '1'
    ELSE (add_temp_158(32 DOWNTO 0));

add_cast_318 <= sumdelay_pipeline6_im(0);
add_cast_319 <= sumdelay_pipeline6_im(1);
add_temp_159 <= resize(add_cast_318, 34) + resize(add_cast_319, 34);
sum7_im <= (32 => '0', OTHERS => '1') WHEN add_temp_159(33) = '0' AND
    add_temp_159(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_159(33) = '1' AND
        add_temp_159(32) /= '1'
```

```
    ELSE (add_temp_159(32 DOWNTO 0));

output_typeconvert_re <= (15 => '0', OTHERS => '1') WHEN sum7_re(32) =
    '0' AND sum7_re(31) /= '0'
    ELSE (15 => '1', OTHERS => '0') WHEN sum7_re(32) = '1' AND sum7_re
        (31) /= '1'
    ELSE (sum7_re(31 DOWNTO 16));
output_typeconvert_im <= (15 => '0', OTHERS => '1') WHEN sum7_im(32) =
    '0' AND sum7_im(31) /= '0'
    ELSE (15 => '1', OTHERS => '0') WHEN sum7_im(32) = '1' AND sum7_im
        (31) /= '1'
    ELSE (sum7_im(31 DOWNTO 16));

DataHoldRegister_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    regout_re <= (OTHERS => '0');
    regout_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF phase_0 = '1' THEN
      regout_re <= output_typeconvert_re;
      regout_im <= output_typeconvert_im;
    END IF;
  END IF;
END PROCESS DataHoldRegister_process;

muxout_re <= output_typeconvert_re WHEN ( phase_0 = '1' ) ELSE
             regout_re;
muxout_im <= output_typeconvert_im WHEN ( phase_0 = '1' ) ELSE
             regout_im;
-- Assignment Statements
FIR_Decimation_out_re <= std_logic_vector(muxout_re);
FIR_Decimation_out_im <= std_logic_vector(muxout_im);
```

```
END rtl;
```

A.1.7 BPSK Demodulator

```
-- -------------------------------------------------------------
--
-- File Name: BPSK_Demodulator_Baseband.vhd
-- Created: 2014-03-16 01:42:09
--
-- Generated by MATLAB 8.2 and HDL Coder 3.3
--
-- -------------------------------------------------------------



-- -------------------------------------------------------------
--
-- Module: BPSK_Demodulator_Baseband
-- Source Path: CommSystem/BPSK Demodulator Baseband
-- Hierarchy Level: 1
--
-- -------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY BPSK_Demodulator_Baseband IS
  PORT( in0_re                            :   IN    std_logic_vector(15
    DOWNTO 0);  -- sfix16_En14
        in0_im                            :   IN    std_logic_vector(15
          DOWNTO 0);  -- sfix16_En14
        out0                              :   OUT   std_logic  -- ufix1
        );
END BPSK_Demodulator_Baseband;


ARCHITECTURE rtl OF BPSK_Demodulator_Baseband IS
```

```
  -- Constants
  CONSTANT nc                              : std_logic_vector(0 TO 7) :=
      ( '0', '0', '1', '0', '1', '1', '1', '0' );  -- ufix1 [8]

  -- Signals
  SIGNAL in0_re_signed                     : signed(15 DOWNTO 0);  --
      sfix16_En14
  SIGNAL in0_im_signed                     : signed(15 DOWNTO 0);  --
      sfix16_En14
  SIGNAL inphase_lt_zero                   : std_logic;
  SIGNAL inphase_eq_zero                   : std_logic;
  SIGNAL quadrature_eq_zero                : std_logic;
  SIGNAL decisionLUTaddr                   : unsigned(2 DOWNTO 0);  --
      ufix3
  SIGNAL hardDecision                      : std_logic;  -- ufix1

BEGIN
  in0_re_signed <= signed(in0_re);


  inphase_lt_zero <= '1' WHEN in0_re_signed < 0 ELSE
      '0';


  inphase_eq_zero <= '1' WHEN in0_re_signed = 0 ELSE
      '0';

  in0_im_signed <= signed(in0_im);


  quadrature_eq_zero <= '1' WHEN in0_im_signed = 0 ELSE
      '0';
```

```
    decisionLUTaddr <= unsigned'(inphase_lt_zero & inphase_eq_zero &
        quadrature_eq_zero);

    hardDecision <= nc(to_integer(decisionLUTaddr));

    out0 <= hardDecision;

END rtl;
```

A.2 BPSK Hand-Optimized Design

A.2.1 Top Module

```
-- -------------------------------------------------------------
--
-- File Name: CommSystem.vhd
-- Created: 2014-03-09 23:35:39
--
-- Generated by MATLAB 8.2 and HDL Coder 3.3
--
--
-- -------------------------------------------------------------
-- Rate and Clocking Details
-- -------------------------------------------------------------
-- Model base rate: 1e-08
-- Target subsystem base rate: 1e-08
--
--
-- Clock Enable  Sample Time
-- -------------------------------------------------------------
-- ce_out        4e-08
-- -------------------------------------------------------------
--
--
-- Output Signal                   Clock Enable  Sample Time
-- -------------------------------------------------------------
-- To_DataSink                     ce_out        4e-08
-- -------------------------------------------------------------
--
-- -------------------------------------------------------------


-- -------------------------------------------------------------
```

```
--
-- Module: CommSystem
-- Source Path: CommSystem
-- Hierarchy Level: 0
--
-- ----------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE work.CommSystem_pkg.ALL;

ENTITY CommSystem IS
  PORT( clk                               :   IN    std_logic;
        reset                             :   IN    std_logic;
        clk_enable                        :   IN    std_logic;
        ce_out                            :   OUT   std_logic;
        To_DataSink                       :   OUT   std_logic  -- ufix1
        );
END CommSystem;


ARCHITECTURE rtl OF CommSystem IS

  -- Component Declarations
  COMPONENT CommSystem_tc
    PORT( clk                             :   IN    std_logic;
          reset                           :   IN    std_logic;
          clk_enable                      :   IN    std_logic;
          enb                             :   OUT   std_logic;
          enb_1_1_1                       :   OUT   std_logic;
          enb_1_4_0                       :   OUT   std_logic;
          enb_1_4_1                       :   OUT   std_logic
          );
```

```
END COMPONENT;

COMPONENT BPSK_Modulator_Baseband
  PORT( bin                               :   IN    std_logic;  -- ufix1
        sym                               :   OUT   std_logic_vector(1
           DOWNTO 0)  -- ufix2
        );
END COMPONENT;

COMPONENT FIR_Interpolation
  PORT( clk                               :   IN    std_logic;
        enb_1_1_1                         :   IN    std_logic;
        reset                             :   IN    std_logic;
        FIR_Interpolation_in              :   IN    std_logic_vector(1
           DOWNTO 0);  -- ufix2
        FIR_Interpolation_out             :   OUT   std_logic_vector(15
           DOWNTO 0)  -- sfix16_En15
        );
END COMPONENT;

COMPONENT FIR_Decimation
  PORT( clk                               :   IN    std_logic;
        enb_1_1_1                         :   IN    std_logic;
        reset                             :   IN    std_logic;
        FIR_Decimation_in                 :   IN    std_logic_vector(15
           DOWNTO 0);  -- ufix16
        FIR_Decimation_out                :   OUT   std_logic --_vector
           (15 DOWNTO 0)  -- sfix16_En14
        );
END COMPONENT;

COMPONENT BPSK_Demodulator_Baseband
  PORT( sym                               :   IN    std_logic; --_vector
```

```
        (15 DOWNTO 0);  -- ufix16
          bin                             :   OUT   std_logic  -- ufix1
          );
  END COMPONENT;

  -- Component Configuration Statements
  FOR ALL : CommSystem_tc
    USE ENTITY work.CommSystem_tc(rtl);

  FOR ALL : BPSK_Modulator_Baseband
    USE ENTITY work.BPSK_Modulator_Baseband(rtl);

  FOR ALL : FIR_Interpolation
    USE ENTITY work.FIR_Interpolation(rtl);

  FOR ALL : FIR_Decimation
    USE ENTITY work.FIR_Decimation(rtl);

  FOR ALL : BPSK_Demodulator_Baseband
    USE ENTITY work.BPSK_Demodulator_Baseband(rtl);

  -- Signals
  SIGNAL enb_1_4_0                        : std_logic;
  SIGNAL enb_1_1_1                        : std_logic;
  SIGNAL enb_1_4_1                        : std_logic;
  SIGNAL enb                              : std_logic;
  SIGNAL PN_Sequence_Generator_out1       : std_logic;  -- ufix1
  SIGNAL BPSK_Modulator_Baseband_out1     : std_logic_vector(1 DOWNTO 0)
      ;  -- ufix2
--  SIGNAL BPSK_Modulator_Baseband_out1_signed : signed(1 DOWNTO 0);  --
     sfix2
--  SIGNAL BPSK_Modulator_Baseband_out1_1   : signed(1 DOWNTO 0);  --
    sfix2
```

```
--  SIGNAL BPSK_Modulator_Baseband_out1_2   : std_logic_vector(1 DOWNTO
    0);  -- ufix2
  SIGNAL FIR_Interpolation_out1           : std_logic_vector(15 DOWNTO
     0);  -- ufix16
  SIGNAL FIR_Interpolation_out1_signed    : signed(15 DOWNTO 0);  --
     sfix16_En15
--  SIGNAL FIR_Interpolation_out1_1         : signed(15 DOWNTO 0);  --
    sfix16_En15
  SIGNAL delayMatch_reg                   : vector_of_signed16(0 TO 1);
      -- sfix16_En15 [2]
  SIGNAL FIR_Interpolation_out1_2         : signed(15 DOWNTO 0);  --
     sfix16_En15
  SIGNAL FIR_Interpolation_out1_3         : std_logic_vector(15 DOWNTO
     0);  -- ufix16
  SIGNAL FIR_Decimation_out1              : std_logic; --_vector(15
     DOWNTO 0);  -- ufix16
--  SIGNAL FIR_Decimation_out1_signed       : signed(15 DOWNTO 0);  --
    sfix16_En14
  SIGNAL FIR_Decimation_out1_1            : std_logic; --signed(15
     DOWNTO 0);  -- sfix16_En14
  SIGNAL BPSK_Demodulator_Baseband_out1   : std_logic;  -- ufix1
  SIGNAL BPSK_Demodulator_Baseband_out1_1 : std_logic;  -- ufix1
  SIGNAL pn_reg                           : unsigned(5 DOWNTO 0);  --
     ufix6
  SIGNAL pn_out                           : std_logic;
  SIGNAL pn_xorout                        : std_logic;
  SIGNAL pn_newvalue                      : vector_of_unsigned6(0 TO 1);
       -- ufix6 [2]
  SIGNAL pn_value_shifted                 : unsigned(4 DOWNTO 0);  --
     ufix5_E1

BEGIN
  u_CommSystem_tc : CommSystem_tc
```

```
  PORT MAP( clk => clk,
            reset => reset,
            clk_enable => clk_enable,
            enb => enb,
            enb_1_1_1 => enb_1_1_1,
            enb_1_4_0 => enb_1_4_0,
            enb_1_4_1 => enb_1_4_1
            );

-------- PN Sequence Generator --------
pn_newvalue(0) <= pn_reg;

pn_xorout <= pn_newvalue(0)(0) XOR pn_newvalue(0)(1);

pn_value_shifted <= pn_newvalue(0)(5 DOWNTO 1);

pn_newvalue(1) <= pn_xorout & pn_value_shifted;

pn_out <= pn_newvalue(0)(0);

PN_generation_temp_process3 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    pn_reg <= to_unsigned(1, 6);
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_4_0 = '1' THEN
      pn_reg <= pn_newvalue(1);
    END IF;
  END IF;
END PROCESS PN_generation_temp_process3;

PN_Sequence_Generator_out1 <= pn_out;
```

```
  ---------- BPSK Modulator Baseband ----------
  u_BPSK_Modulator_Baseband : BPSK_Modulator_Baseband
    PORT MAP( bin => PN_Sequence_Generator_out1,  -- ufix1
              sym => BPSK_Modulator_Baseband_out1  -- ufix2
              );

--  BPSK_Modulator_Baseband_out1_signed <= signed(
    BPSK_Modulator_Baseband_out1);
--
--  FIR_Interpolation_in_pipe_process : PROCESS (clk, reset)
--  BEGIN
--    IF reset = '1' THEN
--      BPSK_Modulator_Baseband_out1_1 <= to_signed(2#00#, 2);
--    ELSIF clk'EVENT AND clk = '1' THEN
--      IF enb_1_4_0 = '1' THEN
--        BPSK_Modulator_Baseband_out1_1 <=
    BPSK_Modulator_Baseband_out1_signed;
--      END IF;
--    END IF;
--  END PROCESS FIR_Interpolation_in_pipe_process;
--
--
--  BPSK_Modulator_Baseband_out1_2 <= std_logic_vector(
    BPSK_Modulator_Baseband_out1_1);

  ---------- FIR Interpolation ----------
  u_FIR_Interpolation : FIR_Interpolation
    PORT MAP( clk => clk,
              enb_1_1_1 => enb_1_1_1,
              reset => reset,
              FIR_Interpolation_in => BPSK_Modulator_Baseband_out1,  --
                  ufix2
              FIR_Interpolation_out => FIR_Interpolation_out1  --
```

```
                sfix16_En15
            );

  FIR_Interpolation_out1_signed <= signed(FIR_Interpolation_out1);

--  FIR_Decimation_in_pipe_process : PROCESS (clk, reset)
--  BEGIN
--    IF reset = '1' THEN
--      FIR_Interpolation_out1_1 <= to_signed(16#0000#, 16);
--    ELSIF clk'EVENT AND clk = '1' THEN
--      IF enb = '1' THEN
--        FIR_Interpolation_out1_1 <= FIR_Interpolation_out1_signed;
--      END IF;
--    END IF;
--  END PROCESS FIR_Decimation_in_pipe_process;

  delayMatch_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      delayMatch_reg <= (OTHERS => to_signed(16#0000#, 16));
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
        delayMatch_reg(0) <= FIR_Interpolation_out1_signed;
        delayMatch_reg(1) <= delayMatch_reg(0);
      END IF;
    END IF;
  END PROCESS delayMatch_process;

  FIR_Interpolation_out1_2 <= delayMatch_reg(1);

  FIR_Interpolation_out1_3 <= std_logic_vector(FIR_Interpolation_out1_2)
      ;
```

```
  ---------- FIR Decimation
  u_FIR_Decimation : FIR_Decimation
    PORT MAP( clk => clk,
              enb_1_1_1 => enb_1_1_1,
              reset => reset,
              FIR_Decimation_in => FIR_Interpolation_out1_3,  -- ufix16
              FIR_Decimation_out => FIR_Decimation_out1  -- sfix16_En14
              );

--  FIR_Decimation_out1_signed <= signed(FIR_Decimation_out1);

  BPSK_Demodulator_Baseband_in_pipe_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      FIR_Decimation_out1_1 <= '0'; --to_signed(16#0000#, 16);
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb_1_4_0 = '1' THEN
        FIR_Decimation_out1_1 <= FIR_Decimation_out1; --
          FIR_Decimation_out1_signed;
      END IF;
    END IF;
  END PROCESS BPSK_Demodulator_Baseband_in_pipe_process;

  ---------- BPSK Demodulator Baseband ----------
  u_BPSK_Demodulator_Baseband : BPSK_Demodulator_Baseband
    PORT MAP( sym => FIR_Decimation_out1_1, --std_logic_vector(
        FIR_Decimation_out1_1),  -- ufix16
              bin => BPSK_Demodulator_Baseband_out1  -- ufix1
              );

  BPSK_Demodulator_Baseband_out_pipe_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
```

```
        BPSK_Demodulator_Baseband_out1_1 <= '0';
      ELSIF clk'EVENT AND clk = '1' THEN
        IF enb_1_4_0 = '1' THEN
          BPSK_Demodulator_Baseband_out1_1 <=
              BPSK_Demodulator_Baseband_out1;
        END IF;
      END IF;
    END PROCESS BPSK_Demodulator_Baseband_out_pipe_process;

    ce_out <= enb_1_4_1;

    To_DataSink <= BPSK_Demodulator_Baseband_out1_1;

END rtl;
```

A.2.2 Package

```
-- -------------------------------------------------------------
--
-- File Name: CommSystem_pkg.vhd
-- Created: 2014-03-09 23:35:39
--
-- Generated by MATLAB 8.2 and HDL Coder 3.3
--
-- -------------------------------------------------------------


LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

PACKAGE CommSystem_pkg IS
  TYPE vector_of_signed16 IS ARRAY (NATURAL RANGE <>) OF signed(15
      DOWNTO 0);
  TYPE vector_of_unsigned6 IS ARRAY (NATURAL RANGE <>) OF unsigned(5
      DOWNTO 0);
END CommSystem_pkg;
```

A.2.3 Timing Controller

```
-- -------------------------------------------------------------
--
-- File Name: CommSystem_tc.vhd
-- Created: 2014-03-09 23:35:08
--
-- Generated by MATLAB 8.2 and HDL Coder 3.3
--
-- -------------------------------------------------------------



-- -------------------------------------------------------------
--
-- Module: CommSystem_tc
-- Source Path: CommSystem_tc
-- Hierarchy Level: 1
--
-- Master clock enable input: clk_enable
--
-- enb         : identical to clk_enable
-- enb_1_1_1   : identical to clk_enable
-- enb_1_4_0   : 4x slower than clk_enable with last phase
-- enb_1_4_1   : 4x slower than clk_enable with phase 1
--
-- -------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY CommSystem_tc IS
  PORT( clk                               :   IN    std_logic;
        reset                             :   IN    std_logic;
        clk_enable                        :   IN    std_logic;
```

```
        enb                                 :   OUT   std_logic;
        enb_1_1_1                           :   OUT   std_logic;
        enb_1_4_0                           :   OUT   std_logic;
        enb_1_4_1                           :   OUT   std_logic
        );
END CommSystem_tc;


ARCHITECTURE rtl OF CommSystem_tc IS

  -- Signals
  SIGNAL count4                             : unsigned(1 DOWNTO 0);  --
      ufix2
  SIGNAL phase_all                          : std_logic;
  SIGNAL phase_0                            : std_logic;
  SIGNAL phase_0_tmp                        : std_logic;
  SIGNAL phase_1                            : std_logic;
  SIGNAL phase_1_tmp                        : std_logic;

BEGIN
  Counter4 : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      count4 <= to_unsigned(1, 2);
    ELSIF clk'event AND clk = '1' THEN
      IF clk_enable = '1' THEN
        IF count4 = to_unsigned(3, 2) THEN
          count4 <= to_unsigned(0, 2);
        ELSE
          count4 <= count4 + 1;
        END IF;
      END IF;
    END IF;
```

```
END PROCESS Counter4;

phase_all <= '1' WHEN clk_enable = '1' ELSE '0';

temp_process1 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    phase_0 <= '0';
  ELSIF clk'event AND clk = '1' THEN
    IF clk_enable = '1' THEN
      phase_0 <= phase_0_tmp;
    END IF;
  END IF;
END PROCESS temp_process1;

phase_0_tmp <= '1' WHEN count4 = to_unsigned(3, 2) AND clk_enable =
    '1' ELSE '0';

temp_process2 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    phase_1 <= '1';
  ELSIF clk'event AND clk = '1' THEN
    IF clk_enable = '1' THEN
      phase_1 <= phase_1_tmp;
    END IF;
  END IF;
END PROCESS temp_process2;

phase_1_tmp <= '1' WHEN count4 = to_unsigned(0, 2) AND clk_enable =
    '1' ELSE '0';

enb <=  phase_all AND clk_enable;
```

```
  enb_1_1_1 <=  phase_all AND clk_enable;

  enb_1_4_0 <=  phase_0 AND clk_enable;

  enb_1_4_1 <=  phase_1 AND clk_enable;


END rtl;
```

A.2.4 BPSK Modulator

```
-- -------------------------------------------------------------
--
-- File Name: BPSK_Modulator_Baseband.vhd
-- Created: 2014-03-09 23:35:08
--
-- Generated by MATLAB 8.2 and HDL Coder 3.3
--
-- -------------------------------------------------------------



-- -------------------------------------------------------------
--
-- Module: BPSK_Modulator_Baseband
-- Source Path: CommSystem/BPSK Modulator Baseband
-- Hierarchy Level: 1
--
-- -------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY BPSK_Modulator_Baseband IS
  PORT( bin                               :   IN    std_logic; -- ufix1
        sym                               :   OUT   std_logic_vector(1
           DOWNTO 0)  -- sfix2
        );
END BPSK_Modulator_Baseband;


ARCHITECTURE rtl OF BPSK_Modulator_Baseband IS
```

```
BEGIN

   sym <= "01" WHEN bin = '0' ELSE "11";

END rtl;
```

A.2.5 FIR Interpolation Filter

```
-- -------------------------------------------------------------
--
-- File Name: FIR_Interpolation
-- Created: 2014-03-09 23:35:09
-- Generated by MATLAB 8.2 and HDL Coder 3.3
--
-- -------------------------------------------------------------
--
--
-- -------------------------------------------------------------
--
-- Module: FIR_Interpolation
-- Source Path: /FIR_Interpolation
--
-- -------------------------------------------------------------
--
-- HDL Implementation    : Fully parallel
-- Multipliers           : 20
-- Folding Factor        : 1


LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.ALL;

ENTITY FIR_Interpolation IS
  PORT( clk                             :   IN    std_logic;
        enb_1_1_1                       :   IN    std_logic;
        reset                           :   IN    std_logic;
        FIR_Interpolation_in            :   IN    std_logic_vector(1
          DOWNTO 0); -- sfix2
        FIR_Interpolation_out           :   OUT   std_logic_vector(15
```

```
            DOWNTO 0) -- sfix16_En15
        );

END FIR_Interpolation;


-------------------------------------------------------------------------------
--Module Architecture: FIR_Interpolation
-------------------------------------------------------------------------------

ARCHITECTURE rtl OF FIR_Interpolation IS
  -- Local Functions
  -- Type Definitions
  TYPE delay_pipeline_type IS ARRAY (NATURAL range <>) OF signed(1
      DOWNTO 0); -- sfix2
  TYPE sumdelay_pipeline_type IS ARRAY (NATURAL range <>) OF signed(15
      DOWNTO 0); -- sfix17_En15
  -- Constants
  CONSTANT coeffphase1_1                  : signed(15 DOWNTO 0) :=
      to_signed(5, 16); -- sfix16_En15
  CONSTANT coeffphase1_2                  : signed(15 DOWNTO 0) :=
      to_signed(41, 16); -- sfix16_En15
  CONSTANT coeffphase1_3                  : signed(15 DOWNTO 0) :=
      to_signed(-83, 16); -- sfix16_En15
  CONSTANT coeffphase1_4                  : signed(15 DOWNTO 0) :=
      to_signed(88, 16); -- sfix16_En15
  CONSTANT coeffphase1_5                  : signed(15 DOWNTO 0) :=
      to_signed(-25, 16); -- sfix16_En15
  CONSTANT coeffphase1_6                  : signed(15 DOWNTO 0) :=
      to_signed(-123, 16); -- sfix16_En15
  CONSTANT coeffphase1_7                  : signed(15 DOWNTO 0) :=
      to_signed(348, 16); -- sfix16_En15
  CONSTANT coeffphase1_8                  : signed(15 DOWNTO 0) :=
      to_signed(-615, 16); -- sfix16_En15
```

```
CONSTANT coeffphase1_9                  : signed(15 DOWNTO 0) :=
    to_signed(869, 16); -- sfix16_En15
CONSTANT coeffphase1_10                 : signed(15 DOWNTO 0) :=
    to_signed(-1052, 16); -- sfix16_En15
CONSTANT coeffphase1_11                 : signed(15 DOWNTO 0) :=
    to_signed(17504, 16); -- sfix16_En15
CONSTANT coeffphase1_12                 : signed(15 DOWNTO 0) :=
    to_signed(-1052, 16); -- sfix16_En15
CONSTANT coeffphase1_13                 : signed(15 DOWNTO 0) :=
    to_signed(869, 16); -- sfix16_En15
CONSTANT coeffphase1_14                 : signed(15 DOWNTO 0) :=
    to_signed(-615, 16); -- sfix16_En15
CONSTANT coeffphase1_15                 : signed(15 DOWNTO 0) :=
    to_signed(348, 16); -- sfix16_En15
CONSTANT coeffphase1_16                 : signed(15 DOWNTO 0) :=
    to_signed(-123, 16); -- sfix16_En15
CONSTANT coeffphase1_17                 : signed(15 DOWNTO 0) :=
    to_signed(-25, 16); -- sfix16_En15
CONSTANT coeffphase1_18                 : signed(15 DOWNTO 0) :=
    to_signed(88, 16); -- sfix16_En15
CONSTANT coeffphase1_19                 : signed(15 DOWNTO 0) :=
    to_signed(-83, 16); -- sfix16_En15
CONSTANT coeffphase1_20                 : signed(15 DOWNTO 0) :=
    to_signed(41, 16); -- sfix16_En15
CONSTANT coeffphase1_21                 : signed(15 DOWNTO 0) :=
    to_signed(5, 16); -- sfix16_En15

CONSTANT coeffphase2_1                  : signed(15 DOWNTO 0) :=
    to_signed(-41, 16); -- sfix16_En15
CONSTANT coeffphase2_2                  : signed(15 DOWNTO 0) :=
    to_signed(60, 16); -- sfix16_En15
CONSTANT coeffphase2_3                  : signed(15 DOWNTO 0) :=
    to_signed(-43, 16); -- sfix16_En15
```

```
CONSTANT coeffphase2_4                  : signed(15 DOWNTO 0) :=
    to_signed(-19, 16); -- sfix16_En15
CONSTANT coeffphase2_5                  : signed(15 DOWNTO 0) :=
    to_signed(112, 16); -- sfix16_En15
CONSTANT coeffphase2_6                  : signed(15 DOWNTO 0) :=
    to_signed(-187, 16); -- sfix16_En15
CONSTANT coeffphase2_7                  : signed(15 DOWNTO 0) :=
    to_signed(163, 16); -- sfix16_En15
CONSTANT coeffphase2_8                  : signed(15 DOWNTO 0) :=
    to_signed(99, 16); -- sfix16_En15
CONSTANT coeffphase2_9                  : signed(15 DOWNTO 0) :=
    to_signed(-901, 16); -- sfix16_En15
CONSTANT coeffphase2_10                 : signed(15 DOWNTO 0) :=
    to_signed(3897, 16); -- sfix16_En15
CONSTANT coeffphase2_11                 : signed(15 DOWNTO 0) :=
    to_signed(15453, 16); -- sfix16_En15
CONSTANT coeffphase2_12                 : signed(15 DOWNTO 0) :=
    to_signed(-3256, 16); -- sfix16_En15
CONSTANT coeffphase2_13                 : signed(15 DOWNTO 0) :=
    to_signed(1542, 16); -- sfix16_En15
CONSTANT coeffphase2_14                 : signed(15 DOWNTO 0) :=
    to_signed(-699, 16); -- sfix16_En15
CONSTANT coeffphase2_15                 : signed(15 DOWNTO 0) :=
    to_signed(210, 16); -- sfix16_En15
CONSTANT coeffphase2_16                 : signed(15 DOWNTO 0) :=
    to_signed(46, 16); -- sfix16_En15
CONSTANT coeffphase2_17                 : signed(15 DOWNTO 0) :=
    to_signed(-132, 16); -- sfix16_En15
CONSTANT coeffphase2_18                 : signed(15 DOWNTO 0) :=
    to_signed(113, 16); -- sfix16_En15
CONSTANT coeffphase2_19                 : signed(15 DOWNTO 0) :=
    to_signed(-48, 16); -- sfix16_En15
CONSTANT coeffphase2_20                 : signed(15 DOWNTO 0) :=
```

```
    to_signed(-13, 16); -- sfix16_En15
CONSTANT coeffphase2_21                 : signed(15 DOWNTO 0) :=
    to_signed(0, 16); -- sfix16_En15

CONSTANT coeffphase3_1                  : signed(15 DOWNTO 0) :=
    to_signed(-52, 16); -- sfix16_En15
CONSTANT coeffphase3_2                  : signed(15 DOWNTO 0) :=
    to_signed(20, 16); -- sfix16_En15
CONSTANT coeffphase3_3                  : signed(15 DOWNTO 0) :=
    to_signed(48, 16); -- sfix16_En15
CONSTANT coeffphase3_4                  : signed(15 DOWNTO 0) :=
    to_signed(-124, 16); -- sfix16_En15
CONSTANT coeffphase3_5                  : signed(15 DOWNTO 0) :=
    to_signed(152, 16); -- sfix16_En15
CONSTANT coeffphase3_6                  : signed(15 DOWNTO 0) :=
    to_signed(-48, 16); -- sfix16_En15
CONSTANT coeffphase3_7                  : signed(15 DOWNTO 0) :=
    to_signed(-300, 16); -- sfix16_En15
CONSTANT coeffphase3_8                  : signed(15 DOWNTO 0) :=
    to_signed(1070, 16); -- sfix16_En15
CONSTANT coeffphase3_9                  : signed(15 DOWNTO 0) :=
    to_signed(-2790, 16); -- sfix16_En15
CONSTANT coeffphase3_10                 : signed(15 DOWNTO 0) :=
    to_signed(10188, 16); -- sfix16_En15
CONSTANT coeffphase3_11                 : signed(15 DOWNTO 0) :=
    to_signed(10188, 16); -- sfix16_En15
CONSTANT coeffphase3_12                 : signed(15 DOWNTO 0) :=
    to_signed(-2790, 16); -- sfix16_En15
CONSTANT coeffphase3_13                 : signed(15 DOWNTO 0) :=
    to_signed(1070, 16); -- sfix16_En15
CONSTANT coeffphase3_14                 : signed(15 DOWNTO 0) :=
    to_signed(-300, 16); -- sfix16_En15
CONSTANT coeffphase3_15                 : signed(15 DOWNTO 0) :=
```

```
    to_signed(-48, 16); -- sfix16_En15
CONSTANT coeffphase3_16                 : signed(15 DOWNTO 0) :=
    to_signed(152, 16); -- sfix16_En15
CONSTANT coeffphase3_17                 : signed(15 DOWNTO 0) :=
    to_signed(-124, 16); -- sfix16_En15
CONSTANT coeffphase3_18                 : signed(15 DOWNTO 0) :=
    to_signed(48, 16); -- sfix16_En15
CONSTANT coeffphase3_19                 : signed(15 DOWNTO 0) :=
    to_signed(20, 16); -- sfix16_En15
CONSTANT coeffphase3_20                 : signed(15 DOWNTO 0) :=
    to_signed(-52, 16); -- sfix16_En15
CONSTANT coeffphase3_21                 : signed(15 DOWNTO 0) :=
    to_signed(0, 16); -- sfix16_En15

CONSTANT coeffphase4_1                  : signed(15 DOWNTO 0) :=
    to_signed(-13, 16); -- sfix16_En15
CONSTANT coeffphase4_2                  : signed(15 DOWNTO 0) :=
    to_signed(-48, 16); -- sfix16_En15
CONSTANT coeffphase4_3                  : signed(15 DOWNTO 0) :=
    to_signed(113, 16); -- sfix16_En15
CONSTANT coeffphase4_4                  : signed(15 DOWNTO 0) :=
    to_signed(-132, 16); -- sfix16_En15
CONSTANT coeffphase4_5                  : signed(15 DOWNTO 0) :=
    to_signed(46, 16); -- sfix16_En15
CONSTANT coeffphase4_6                  : signed(15 DOWNTO 0) :=
    to_signed(210, 16); -- sfix16_En15
CONSTANT coeffphase4_7                  : signed(15 DOWNTO 0) :=
    to_signed(-699, 16); -- sfix16_En15
CONSTANT coeffphase4_8                  : signed(15 DOWNTO 0) :=
    to_signed(1542, 16); -- sfix16_En15
CONSTANT coeffphase4_9                  : signed(15 DOWNTO 0) :=
    to_signed(-3256, 16); -- sfix16_En15
CONSTANT coeffphase4_10                 : signed(15 DOWNTO 0) :=
```

```
    to_signed(15453, 16); -- sfix16_En15
CONSTANT coeffphase4_11                  : signed(15 DOWNTO 0) :=
    to_signed(3897, 16); -- sfix16_En15
CONSTANT coeffphase4_12                  : signed(15 DOWNTO 0) :=
    to_signed(-901, 16); -- sfix16_En15
CONSTANT coeffphase4_13                  : signed(15 DOWNTO 0) :=
    to_signed(99, 16); -- sfix16_En15
CONSTANT coeffphase4_14                  : signed(15 DOWNTO 0) :=
    to_signed(163, 16); -- sfix16_En15
CONSTANT coeffphase4_15                  : signed(15 DOWNTO 0) :=
    to_signed(-187, 16); -- sfix16_En15
CONSTANT coeffphase4_16                  : signed(15 DOWNTO 0) :=
    to_signed(112, 16); -- sfix16_En15
CONSTANT coeffphase4_17                  : signed(15 DOWNTO 0) :=
    to_signed(-19, 16); -- sfix16_En15
CONSTANT coeffphase4_18                  : signed(15 DOWNTO 0) :=
    to_signed(-43, 16); -- sfix16_En15
CONSTANT coeffphase4_19                  : signed(15 DOWNTO 0) :=
    to_signed(60, 16); -- sfix16_En15
CONSTANT coeffphase4_20                  : signed(15 DOWNTO 0) :=
    to_signed(-41, 16); -- sfix16_En15
CONSTANT coeffphase4_21                  : signed(15 DOWNTO 0) :=
    to_signed(0, 16); -- sfix16_En15

-- Signals
SIGNAL cur_count                         : unsigned(1 DOWNTO 0); --
    ufix2
SIGNAL phase_3                           : std_logic; -- boolean
SIGNAL delay_pipeline                    : delay_pipeline_type(0 TO 20)
    ; -- sfix2

SIGNAL coef_01                                : signed(15 DOWNTO 0);
    -- sfix16_En15
```

```
SIGNAL coef_02                          : signed(15 DOWNTO 0);
    -- sfix16_En15
SIGNAL coef_03                          : signed(15 DOWNTO 0);
    -- sfix16_En15
SIGNAL coef_04                          : signed(15 DOWNTO 0);
    -- sfix16_En15
SIGNAL coef_05                          : signed(15 DOWNTO 0);
    -- sfix16_En15
SIGNAL coef_06                          : signed(15 DOWNTO 0);
    -- sfix16_En15
SIGNAL coef_07                          : signed(15 DOWNTO 0);
    -- sfix16_En15
SIGNAL coef_08                          : signed(15 DOWNTO 0);
    -- sfix16_En15
SIGNAL coef_09                          : signed(15 DOWNTO 0);
    -- sfix16_En15
SIGNAL coef_10                          : signed(15 DOWNTO 0);
    -- sfix16_En15
SIGNAL coef_11                          : signed(15 DOWNTO 0);
    -- sfix16_En15
SIGNAL coef_12                          : signed(15 DOWNTO 0);
    -- sfix16_En15
SIGNAL coef_13                          : signed(15 DOWNTO 0);
    -- sfix16_En15
SIGNAL coef_14                          : signed(15 DOWNTO 0);
    -- sfix16_En15
SIGNAL coef_15                          : signed(15 DOWNTO 0);
    -- sfix16_En15
SIGNAL coef_16                          : signed(15 DOWNTO 0);
    -- sfix16_En15
SIGNAL coef_17                          : signed(15 DOWNTO 0);
    -- sfix16_En15
SIGNAL coef_18                          : signed(15 DOWNTO 0);
```

```
    -- sfix16_En15
SIGNAL coef_19                          : signed(15 DOWNTO 0);
    -- sfix16_En15
SIGNAL coef_20                          : signed(15 DOWNTO 0);
    -- sfix16_En15
SIGNAL coef_21                          : signed(15 DOWNTO 0);
    -- sfix16_En15

SIGNAL prod_temp_01                     : signed(17 DOWNTO 0);
    -- sfix18_En15
SIGNAL prod_temp_02                     : signed(17 DOWNTO 0);
    -- sfix18_En15
SIGNAL prod_temp_03                     : signed(17 DOWNTO 0);
    -- sfix18_En15
SIGNAL prod_temp_04                     : signed(17 DOWNTO 0);
    -- sfix18_En15
SIGNAL prod_temp_05                     : signed(17 DOWNTO 0);
    -- sfix18_En15
SIGNAL prod_temp_06                     : signed(17 DOWNTO 0);
    -- sfix18_En15
SIGNAL prod_temp_07                     : signed(17 DOWNTO 0);
    -- sfix18_En15
SIGNAL prod_temp_08                     : signed(17 DOWNTO 0);
    -- sfix18_En15
SIGNAL prod_temp_09                     : signed(17 DOWNTO 0);
    -- sfix18_En15
SIGNAL prod_temp_10                     : signed(17 DOWNTO 0);
    -- sfix18_En15
SIGNAL prod_temp_11                     : signed(17 DOWNTO 0);
    -- sfix18_En15
SIGNAL prod_temp_12                     : signed(17 DOWNTO 0);
    -- sfix18_En15
SIGNAL prod_temp_13                     : signed(17 DOWNTO 0);
```

```
    -- sfix18_En15
SIGNAL prod_temp_14                     : signed(17 DOWNTO 0);
    -- sfix18_En15
SIGNAL prod_temp_15                     : signed(17 DOWNTO 0);
    -- sfix18_En15
SIGNAL prod_temp_16                     : signed(17 DOWNTO 0);
    -- sfix18_En15
SIGNAL prod_temp_17                     : signed(17 DOWNTO 0);
    -- sfix18_En15
SIGNAL prod_temp_18                     : signed(17 DOWNTO 0);
    -- sfix18_En15
SIGNAL prod_temp_19                     : signed(17 DOWNTO 0);
    -- sfix18_En15
SIGNAL prod_temp_20                     : signed(17 DOWNTO 0);
    -- sfix18_En15
SIGNAL prod_temp_21                     : signed(17 DOWNTO 0);
    -- sfix18_En15

SIGNAL product_01                       : signed(15 DOWNTO 0);
SIGNAL product_02                       : signed(15 DOWNTO 0);
SIGNAL product_03                       : signed(15 DOWNTO 0);
SIGNAL product_04                       : signed(15 DOWNTO 0);
SIGNAL product_05                       : signed(15 DOWNTO 0);
SIGNAL product_06                       : signed(15 DOWNTO 0);
SIGNAL product_07                       : signed(15 DOWNTO 0);
SIGNAL product_08                       : signed(15 DOWNTO 0);
SIGNAL product_09                       : signed(15 DOWNTO 0);
SIGNAL product_10                       : signed(15 DOWNTO 0);
SIGNAL product_11                       : signed(15 DOWNTO 0);
SIGNAL product_12                       : signed(15 DOWNTO 0);
SIGNAL product_13                       : signed(15 DOWNTO 0);
SIGNAL product_14                       : signed(15 DOWNTO 0);
SIGNAL product_15                       : signed(15 DOWNTO 0);
```

```
SIGNAL product_16                       : signed(15 DOWNTO 0);
SIGNAL product_17                       : signed(15 DOWNTO 0);
SIGNAL product_18                       : signed(15 DOWNTO 0);
SIGNAL product_19                       : signed(15 DOWNTO 0);
SIGNAL product_20                       : signed(15 DOWNTO 0);
SIGNAL product_21                       : signed(15 DOWNTO 0);

SIGNAL sumvector1                       : sumdelay_pipeline_type(0 TO
    10);
SIGNAL sumdelay_pipeline1               : sumdelay_pipeline_type(0 TO
    10);

SIGNAL sumvector2                       : sumdelay_pipeline_type(0 TO
    5);
SIGNAL sumdelay_pipeline2               : sumdelay_pipeline_type(0 TO
    5);

SIGNAL sumvector3                       : sumdelay_pipeline_type(0 TO
    2);
SIGNAL sumdelay_pipeline3               : sumdelay_pipeline_type(0 TO
    2);

SIGNAL sumvector4                       : sumdelay_pipeline_type(0 TO
    1);
SIGNAL sumdelay_pipeline4               : sumdelay_pipeline_type(0 TO
    1);

SIGNAL sum5                             : signed(15 DOWNTO 0);


BEGIN

  -- Block Statements
```

```
ce_output : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    cur_count <= to_unsigned(0, 2);
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN
      IF cur_count = to_unsigned(3, 2) THEN
        cur_count <= to_unsigned(0, 2);
      ELSE
        cur_count <= cur_count + 1;
      END IF;
    END IF;
  END IF;
END PROCESS ce_output;

phase_3 <= '1' WHEN cur_count = to_unsigned(3, 2) AND enb_1_1_1 = '1'
    ELSE '0';

--  ----------------- Delay Registers -----------------

Delay_Pipeline_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    delay_pipeline(0 TO 20) <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF phase_3 = '1' THEN
      delay_pipeline(0) <= signed(FIR_Interpolation_in);
      delay_pipeline(1 TO 20) <= delay_pipeline(0 TO 19);
    END IF;
  END IF;
END PROCESS Delay_Pipeline_process;
```

```
coef_01 <= coeffphase1_21 WHEN ( cur_count = to_unsigned(0, 2) ) ELSE
                    coeffphase2_21 WHEN ( cur_count = to_unsigned(1,
                        2) ) ELSE
                    coeffphase3_21 WHEN ( cur_count = to_unsigned(2,
                        2) ) ELSE
                    coeffphase4_21;
coef_02 <= coeffphase1_20 WHEN ( cur_count = to_unsigned(0, 2) ) ELSE
                      coeffphase2_20 WHEN ( cur_count = to_unsigned(1,
                          2) ) ELSE
                      coeffphase3_20 WHEN ( cur_count = to_unsigned(2,
                          2) ) ELSE
                      coeffphase4_20;
coef_03 <= coeffphase1_19 WHEN ( cur_count = to_unsigned(0, 2) ) ELSE
                      coeffphase2_19 WHEN ( cur_count = to_unsigned(1,
                          2) ) ELSE
                      coeffphase3_19 WHEN ( cur_count = to_unsigned(2,
                          2) ) ELSE
                      coeffphase4_19;
coef_04 <= coeffphase1_18 WHEN ( cur_count = to_unsigned(0, 2) ) ELSE
                      coeffphase2_18 WHEN ( cur_count = to_unsigned(1,
                          2) ) ELSE
                      coeffphase3_18 WHEN ( cur_count = to_unsigned(2,
                          2) ) ELSE
                      coeffphase4_18;
coef_05 <= coeffphase1_17 WHEN ( cur_count = to_unsigned(0, 2) ) ELSE
                      coeffphase2_17 WHEN ( cur_count = to_unsigned(1,
                          2) ) ELSE
                      coeffphase3_17 WHEN ( cur_count = to_unsigned(2,
                          2) ) ELSE
                      coeffphase4_17;
coef_06 <= coeffphase1_16 WHEN ( cur_count = to_unsigned(0, 2) ) ELSE
                      coeffphase2_16 WHEN ( cur_count = to_unsigned(1,
                          2) ) ELSE
```

```
                    coeffphase3_16 WHEN ( cur_count = to_unsigned(2,
                        2) ) ELSE
                    coeffphase4_16;
coef_07 <= coeffphase1_15 WHEN ( cur_count = to_unsigned(0, 2) ) ELSE
                    coeffphase2_15 WHEN ( cur_count = to_unsigned(1,
                        2) ) ELSE
                    coeffphase3_15 WHEN ( cur_count = to_unsigned(2,
                        2) ) ELSE
                    coeffphase4_15;
coef_08 <= coeffphase1_14 WHEN ( cur_count = to_unsigned(0, 2) ) ELSE
                    coeffphase2_14 WHEN ( cur_count = to_unsigned(1,
                        2) ) ELSE
                    coeffphase3_14 WHEN ( cur_count = to_unsigned(2,
                        2) ) ELSE
                    coeffphase4_14;
coef_09 <= coeffphase1_13 WHEN ( cur_count = to_unsigned(0, 2) ) ELSE
                    coeffphase2_13 WHEN ( cur_count = to_unsigned(1,
                        2) ) ELSE
                    coeffphase3_13 WHEN ( cur_count = to_unsigned(2,
                        2) ) ELSE
                    coeffphase4_13;
coef_10 <= coeffphase1_12 WHEN ( cur_count = to_unsigned(0, 2) ) ELSE
                    coeffphase2_12 WHEN ( cur_count = to_unsigned(1,
                        2) ) ELSE
                    coeffphase3_12 WHEN ( cur_count = to_unsigned(2,
                        2) ) ELSE
                    coeffphase4_12;
coef_11 <= coeffphase1_11 WHEN ( cur_count = to_unsigned(0, 2) ) ELSE
                     coeffphase2_11 WHEN ( cur_count = to_unsigned
                        (1, 2) ) ELSE
                     coeffphase3_11 WHEN ( cur_count = to_unsigned
                        (2, 2) ) ELSE
                     coeffphase4_11;
```

```
coef_12 <= coeffphase1_10 WHEN ( cur_count = to_unsigned(0, 2) ) ELSE
                          coeffphase2_10 WHEN ( cur_count = to_unsigned
                              (1, 2) ) ELSE
                          coeffphase3_10 WHEN ( cur_count = to_unsigned
                              (2, 2) ) ELSE
                          coeffphase4_10;
coef_13 <= coeffphase1_9 WHEN ( cur_count = to_unsigned(0, 2) ) ELSE
                          coeffphase2_9 WHEN ( cur_count = to_unsigned(1,
                              2) ) ELSE
                          coeffphase3_9 WHEN ( cur_count = to_unsigned(2,
                              2) ) ELSE
                          coeffphase4_9;
coef_14 <= coeffphase1_8 WHEN ( cur_count = to_unsigned(0, 2) ) ELSE
                          coeffphase2_8 WHEN ( cur_count = to_unsigned(1,
                              2) ) ELSE
                          coeffphase3_8 WHEN ( cur_count = to_unsigned(2,
                              2) ) ELSE
                          coeffphase4_8;
coef_15 <= coeffphase1_7 WHEN ( cur_count = to_unsigned(0, 2) ) ELSE
                          coeffphase2_7 WHEN ( cur_count = to_unsigned(1,
                              2) ) ELSE
                          coeffphase3_7 WHEN ( cur_count = to_unsigned(2,
                              2) ) ELSE
                          coeffphase4_7;
coef_16 <= coeffphase1_6 WHEN ( cur_count = to_unsigned(0, 2) ) ELSE
                          coeffphase2_6 WHEN ( cur_count = to_unsigned(1,
                              2) ) ELSE
                          coeffphase3_6 WHEN ( cur_count = to_unsigned(2,
                              2) ) ELSE
                          coeffphase4_6;
coef_17 <= coeffphase1_5 WHEN ( cur_count = to_unsigned(0, 2) ) ELSE
                          coeffphase2_5 WHEN ( cur_count = to_unsigned(1,
                              2) ) ELSE
```

```
                        coeffphase3_5 WHEN ( cur_count = to_unsigned(2,
                            2) ) ELSE
                        coeffphase4_5;
coef_18 <= coeffphase1_4 WHEN ( cur_count = to_unsigned(0, 2) ) ELSE
                        coeffphase2_4 WHEN ( cur_count = to_unsigned(1,
                            2) ) ELSE
                        coeffphase3_4 WHEN ( cur_count = to_unsigned(2,
                            2) ) ELSE
                        coeffphase4_4;
coef_19 <= coeffphase1_3 WHEN ( cur_count = to_unsigned(0, 2) ) ELSE
                        coeffphase2_3 WHEN ( cur_count = to_unsigned(1,
                            2) ) ELSE
                        coeffphase3_3 WHEN ( cur_count = to_unsigned(2,
                            2) ) ELSE
                        coeffphase4_3;
coef_20 <= coeffphase1_2 WHEN ( cur_count = to_unsigned(0, 2) ) ELSE
                        coeffphase2_2 WHEN ( cur_count = to_unsigned(1,
                            2) ) ELSE
                        coeffphase3_2 WHEN ( cur_count = to_unsigned(2,
                            2) ) ELSE
                        coeffphase4_2;
coef_21 <= coeffphase1_1 WHEN ( cur_count = to_unsigned(0, 2) ) ELSE
                        coeffphase2_1 WHEN ( cur_count = to_unsigned(1,
                            2) ) ELSE
                        coeffphase3_1 WHEN ( cur_count = to_unsigned(2,
                            2) ) ELSE
                        coeffphase4_1;


temp_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    prod_temp_01 <= (OTHERS => '0');
    prod_temp_02 <= (OTHERS => '0');
```

```
    prod_temp_03 <= (OTHERS => '0');
    prod_temp_04 <= (OTHERS => '0');
    prod_temp_05 <= (OTHERS => '0');
    prod_temp_06 <= (OTHERS => '0');
    prod_temp_07 <= (OTHERS => '0');
    prod_temp_08 <= (OTHERS => '0');
    prod_temp_09 <= (OTHERS => '0');
    prod_temp_10 <= (OTHERS => '0');
    prod_temp_11 <= (OTHERS => '0');
    prod_temp_12 <= (OTHERS => '0');
    prod_temp_13 <= (OTHERS => '0');
    prod_temp_14 <= (OTHERS => '0');
    prod_temp_15 <= (OTHERS => '0');
    prod_temp_16 <= (OTHERS => '0');
    prod_temp_17 <= (OTHERS => '0');
    prod_temp_18 <= (OTHERS => '0');
    prod_temp_19 <= (OTHERS => '0');
    prod_temp_20 <= (OTHERS => '0');
    prod_temp_21 <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN
      prod_temp_01 <= delay_pipeline(20) * coef_01;
      prod_temp_02 <= delay_pipeline(19) * coef_02;
      prod_temp_03 <= delay_pipeline(18) * coef_03;
      prod_temp_04 <= delay_pipeline(17) * coef_04;
      prod_temp_05 <= delay_pipeline(16) * coef_05;
      prod_temp_06 <= delay_pipeline(15) * coef_06;
      prod_temp_07 <= delay_pipeline(14) * coef_07;
      prod_temp_08 <= delay_pipeline(13) * coef_08;
      prod_temp_09 <= delay_pipeline(12) * coef_09;
      prod_temp_10 <= delay_pipeline(11) * coef_10;
      prod_temp_11 <= delay_pipeline(10) * coef_11;
      prod_temp_12 <= delay_pipeline(9) * coef_12;
```

```
        prod_temp_13 <= delay_pipeline(8) * coef_13;
        prod_temp_14 <= delay_pipeline(7) * coef_14;
        prod_temp_15 <= delay_pipeline(6) * coef_15;
        prod_temp_16 <= delay_pipeline(5) * coef_16;
        prod_temp_17 <= delay_pipeline(4) * coef_17;
        prod_temp_18 <= delay_pipeline(3) * coef_18;
        prod_temp_19 <= delay_pipeline(2) * coef_19;
        prod_temp_20 <= delay_pipeline(1) * coef_20;
        prod_temp_21 <= delay_pipeline(0) * coef_21;
    END IF;
  END IF;
END PROCESS temp_process;


product_01 <= prod_temp_01(15 downto 0);
product_02 <= prod_temp_02(15 downto 0);
product_03 <= prod_temp_03(15 downto 0);
product_04 <= prod_temp_04(15 downto 0);
product_05 <= prod_temp_05(15 downto 0);
product_06 <= prod_temp_06(15 downto 0);
product_07 <= prod_temp_07(15 downto 0);
product_08 <= prod_temp_08(15 downto 0);
product_09 <= prod_temp_09(15 downto 0);
product_10 <= prod_temp_10(15 downto 0);
product_11 <= prod_temp_11(15 downto 0);
product_12 <= prod_temp_12(15 downto 0);
product_13 <= prod_temp_13(15 downto 0);
product_14 <= prod_temp_14(15 downto 0);
product_15 <= prod_temp_15(15 downto 0);
product_16 <= prod_temp_16(15 downto 0);
product_17 <= prod_temp_17(15 downto 0);
product_18 <= prod_temp_18(15 downto 0);
product_19 <= prod_temp_19(15 downto 0);
```

```
product_20 <= prod_temp_20(15 downto 0);
product_21 <= prod_temp_21(15 downto 0);

sumvector1(0) <= product_01 + product_02;
sumvector1(1) <= product_03 + product_04;
sumvector1(2) <= product_05 + product_06;
sumvector1(3) <= product_07 + product_08;
sumvector1(4) <= product_09 + product_10;
sumvector1(5) <= product_11 + product_12;
sumvector1(6) <= product_13 + product_14;
sumvector1(7) <= product_15 + product_16;
sumvector1(8) <= product_17 + product_18;
sumvector1(9) <= product_19 + product_20;
sumvector1(10) <= product_21;

sumdelay_pipeline_process1 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    sumdelay_pipeline1 <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN
      sumdelay_pipeline1(0 TO 10) <= sumvector1(0 TO 10);
    END IF;
  END IF;
END PROCESS sumdelay_pipeline_process1;

sumvector2(0) <= sumdelay_pipeline1(0) + sumdelay_pipeline1(1);
sumvector2(1) <= sumdelay_pipeline1(2) + sumdelay_pipeline1(3);
sumvector2(2) <= sumdelay_pipeline1(4) + sumdelay_pipeline1(5);
sumvector2(3) <= sumdelay_pipeline1(6) + sumdelay_pipeline1(7);
sumvector2(4) <= sumdelay_pipeline1(8) + sumdelay_pipeline1(9);
sumvector2(5) <= sumvector1(10);
```

```
sumdelay_pipeline_process2 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    sumdelay_pipeline2 <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN
      sumdelay_pipeline2(0 TO 5) <= sumvector2(0 TO 5);
    END IF;
  END IF;
END PROCESS sumdelay_pipeline_process2;

sumvector3(0) <= sumdelay_pipeline2(0) + sumdelay_pipeline2(1);
sumvector3(1) <= sumdelay_pipeline2(2) + sumdelay_pipeline2(3);
sumvector3(2) <= sumdelay_pipeline2(4) + sumdelay_pipeline2(5);

sumdelay_pipeline_process3 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    sumdelay_pipeline3 <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF enb_1_1_1 = '1' THEN
      sumdelay_pipeline3(0 TO 2) <= sumvector3(0 TO 2);
    END IF;
  END IF;
END PROCESS sumdelay_pipeline_process3;

sumvector4(0) <= sumdelay_pipeline3(0) + sumdelay_pipeline3(1);
sumvector4(1) <= sumdelay_pipeline3(2);

sumdelay_pipeline_process4 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    sumdelay_pipeline4 <= (OTHERS => (OTHERS => '0'));
```

```
    ELSIF clk'event AND clk = '1' THEN
      IF enb_1_1_1 = '1' THEN
        sumdelay_pipeline4(0 TO 1) <= sumvector4(0 TO 1);
      END IF;
    END IF;
  END PROCESS sumdelay_pipeline_process4;

  sum5 <= sumdelay_pipeline4(0) + sumdelay_pipeline4(1);


  -- Assignment Statements
  FIR_Interpolation_out <= std_logic_vector(sum5);

END rtl;
```

A.2.6 FIR Decimation Filter

```
-- -------------------------------------------------------------
--
-- File Name: FIR_Decimation
-- Created: 2014-03-09 23:35:15
-- Generated by MATLAB 8.2 and HDL Coder 3.3
--
-- -------------------------------------------------------------


-- -------------------------------------------------------------
--
-- Module: FIR_Decimation
-- Source Path: /FIR_Decimation
--
-- -------------------------------------------------------------
--
-- HDL Implementation    : Fully parallel
-- Multipliers           : 81
-- Folding Factor        : 1


LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.ALL;

ENTITY FIR_Decimation IS
  PORT( clk                               :   IN    std_logic;
        enb_1_1_1                         :   IN    std_logic;
        reset                             :   IN    std_logic;
        FIR_Decimation_in                 :   IN    std_logic_vector(15
          DOWNTO 0); -- sfix16_En15
        FIR_Decimation_out                :   OUT   std_logic --_vector
```

```
            (15 DOWNTO 0) -- sfix16_En14
        );

END FIR_Decimation;


-------------------------------------------------------------------------
--Module Architecture: FIR_Decimation
-------------------------------------------------------------------------

ARCHITECTURE rtl OF FIR_Decimation IS
  -- Local Functions
  -- Type Definitions
  TYPE input_pipeline_type IS ARRAY (NATURAL range <>) OF signed(15
      DOWNTO 0); -- sfix16_En15
  TYPE sumdelay_pipeline_type IS ARRAY (NATURAL range <>) OF signed(32
      DOWNTO 0); -- sfix33_En30
  -- Constants
  CONSTANT coeffphase1_1                  : signed(15 DOWNTO 0) :=
      to_signed(5, 16); -- sfix16_En15
  CONSTANT coeffphase1_2                  : signed(15 DOWNTO 0) :=
      to_signed(41, 16); -- sfix16_En15
  CONSTANT coeffphase1_3                  : signed(15 DOWNTO 0) :=
      to_signed(-83, 16); -- sfix16_En15
  CONSTANT coeffphase1_4                  : signed(15 DOWNTO 0) :=
      to_signed(88, 16); -- sfix16_En15
  CONSTANT coeffphase1_5                  : signed(15 DOWNTO 0) :=
      to_signed(-25, 16); -- sfix16_En15
  CONSTANT coeffphase1_6                  : signed(15 DOWNTO 0) :=
      to_signed(-123, 16); -- sfix16_En15
  CONSTANT coeffphase1_7                  : signed(15 DOWNTO 0) :=
      to_signed(348, 16); -- sfix16_En15
  CONSTANT coeffphase1_8                  : signed(15 DOWNTO 0) :=
      to_signed(-615, 16); -- sfix16_En15
```

```
CONSTANT coeffphase1_9                  : signed(15 DOWNTO 0) :=
    to_signed(869, 16); -- sfix16_En15
CONSTANT coeffphase1_10                 : signed(15 DOWNTO 0) :=
    to_signed(-1052, 16); -- sfix16_En15
CONSTANT coeffphase1_11                 : signed(15 DOWNTO 0) :=
    to_signed(17504, 16); -- sfix16_En15
CONSTANT coeffphase1_12                 : signed(15 DOWNTO 0) :=
    to_signed(-1052, 16); -- sfix16_En15
CONSTANT coeffphase1_13                 : signed(15 DOWNTO 0) :=
    to_signed(869, 16); -- sfix16_En15
CONSTANT coeffphase1_14                 : signed(15 DOWNTO 0) :=
    to_signed(-615, 16); -- sfix16_En15
CONSTANT coeffphase1_15                 : signed(15 DOWNTO 0) :=
    to_signed(348, 16); -- sfix16_En15
CONSTANT coeffphase1_16                 : signed(15 DOWNTO 0) :=
    to_signed(-123, 16); -- sfix16_En15
CONSTANT coeffphase1_17                 : signed(15 DOWNTO 0) :=
    to_signed(-25, 16); -- sfix16_En15
CONSTANT coeffphase1_18                 : signed(15 DOWNTO 0) :=
    to_signed(88, 16); -- sfix16_En15
CONSTANT coeffphase1_19                 : signed(15 DOWNTO 0) :=
    to_signed(-83, 16); -- sfix16_En15
CONSTANT coeffphase1_20                 : signed(15 DOWNTO 0) :=
    to_signed(41, 16); -- sfix16_En15
CONSTANT coeffphase1_21                 : signed(15 DOWNTO 0) :=
    to_signed(5, 16); -- sfix16_En15

CONSTANT coeffphase2_1                  : signed(15 DOWNTO 0) :=
    to_signed(-41, 16); -- sfix16_En15
CONSTANT coeffphase2_2                  : signed(15 DOWNTO 0) :=
    to_signed(60, 16); -- sfix16_En15
CONSTANT coeffphase2_3                  : signed(15 DOWNTO 0) :=
    to_signed(-43, 16); -- sfix16_En15
```

```
CONSTANT coeffphase2_4                  : signed(15 DOWNTO 0) :=
    to_signed(-19, 16); -- sfix16_En15
CONSTANT coeffphase2_5                  : signed(15 DOWNTO 0) :=
    to_signed(112, 16); -- sfix16_En15
CONSTANT coeffphase2_6                  : signed(15 DOWNTO 0) :=
    to_signed(-187, 16); -- sfix16_En15
CONSTANT coeffphase2_7                  : signed(15 DOWNTO 0) :=
    to_signed(163, 16); -- sfix16_En15
CONSTANT coeffphase2_8                  : signed(15 DOWNTO 0) :=
    to_signed(99, 16); -- sfix16_En15
CONSTANT coeffphase2_9                  : signed(15 DOWNTO 0) :=
    to_signed(-901, 16); -- sfix16_En15
CONSTANT coeffphase2_10                 : signed(15 DOWNTO 0) :=
    to_signed(3897, 16); -- sfix16_En15
CONSTANT coeffphase2_11                 : signed(15 DOWNTO 0) :=
    to_signed(15453, 16); -- sfix16_En15
CONSTANT coeffphase2_12                 : signed(15 DOWNTO 0) :=
    to_signed(-3256, 16); -- sfix16_En15
CONSTANT coeffphase2_13                 : signed(15 DOWNTO 0) :=
    to_signed(1542, 16); -- sfix16_En15
CONSTANT coeffphase2_14                 : signed(15 DOWNTO 0) :=
    to_signed(-699, 16); -- sfix16_En15
CONSTANT coeffphase2_15                 : signed(15 DOWNTO 0) :=
    to_signed(210, 16); -- sfix16_En15
CONSTANT coeffphase2_16                 : signed(15 DOWNTO 0) :=
    to_signed(46, 16); -- sfix16_En15
CONSTANT coeffphase2_17                 : signed(15 DOWNTO 0) :=
    to_signed(-132, 16); -- sfix16_En15
CONSTANT coeffphase2_18                 : signed(15 DOWNTO 0) :=
    to_signed(113, 16); -- sfix16_En15
CONSTANT coeffphase2_19                 : signed(15 DOWNTO 0) :=
    to_signed(-48, 16); -- sfix16_En15
CONSTANT coeffphase2_20                 : signed(15 DOWNTO 0) :=
```

```
    to_signed(-13, 16); -- sfix16_En15
CONSTANT coeffphase2_21                 : signed(15 DOWNTO 0) :=
    to_signed(0, 16); -- sfix16_En15

CONSTANT coeffphase3_1                  : signed(15 DOWNTO 0) :=
    to_signed(-52, 16); -- sfix16_En15
CONSTANT coeffphase3_2                  : signed(15 DOWNTO 0) :=
    to_signed(20, 16); -- sfix16_En15
CONSTANT coeffphase3_3                  : signed(15 DOWNTO 0) :=
    to_signed(48, 16); -- sfix16_En15
CONSTANT coeffphase3_4                  : signed(15 DOWNTO 0) :=
    to_signed(-124, 16); -- sfix16_En15
CONSTANT coeffphase3_5                  : signed(15 DOWNTO 0) :=
    to_signed(152, 16); -- sfix16_En15
CONSTANT coeffphase3_6                  : signed(15 DOWNTO 0) :=
    to_signed(-48, 16); -- sfix16_En15
CONSTANT coeffphase3_7                  : signed(15 DOWNTO 0) :=
    to_signed(-300, 16); -- sfix16_En15
CONSTANT coeffphase3_8                  : signed(15 DOWNTO 0) :=
    to_signed(1070, 16); -- sfix16_En15
CONSTANT coeffphase3_9                  : signed(15 DOWNTO 0) :=
    to_signed(-2790, 16); -- sfix16_En15
CONSTANT coeffphase3_10                 : signed(15 DOWNTO 0) :=
    to_signed(10188, 16); -- sfix16_En15
CONSTANT coeffphase3_11                 : signed(15 DOWNTO 0) :=
    to_signed(10188, 16); -- sfix16_En15
CONSTANT coeffphase3_12                 : signed(15 DOWNTO 0) :=
    to_signed(-2790, 16); -- sfix16_En15
CONSTANT coeffphase3_13                 : signed(15 DOWNTO 0) :=
    to_signed(1070, 16); -- sfix16_En15
CONSTANT coeffphase3_14                 : signed(15 DOWNTO 0) :=
    to_signed(-300, 16); -- sfix16_En15
CONSTANT coeffphase3_15                 : signed(15 DOWNTO 0) :=
```

```
    to_signed(-48, 16); -- sfix16_En15
CONSTANT coeffphase3_16                 : signed(15 DOWNTO 0) :=
    to_signed(152, 16); -- sfix16_En15
CONSTANT coeffphase3_17                 : signed(15 DOWNTO 0) :=
    to_signed(-124, 16); -- sfix16_En15
CONSTANT coeffphase3_18                 : signed(15 DOWNTO 0) :=
    to_signed(48, 16); -- sfix16_En15
CONSTANT coeffphase3_19                 : signed(15 DOWNTO 0) :=
    to_signed(20, 16); -- sfix16_En15
CONSTANT coeffphase3_20                 : signed(15 DOWNTO 0) :=
    to_signed(-52, 16); -- sfix16_En15
CONSTANT coeffphase3_21                 : signed(15 DOWNTO 0) :=
    to_signed(0, 16); -- sfix16_En15

CONSTANT coeffphase4_1                  : signed(15 DOWNTO 0) :=
    to_signed(-13, 16); -- sfix16_En15
CONSTANT coeffphase4_2                  : signed(15 DOWNTO 0) :=
    to_signed(-48, 16); -- sfix16_En15
CONSTANT coeffphase4_3                  : signed(15 DOWNTO 0) :=
    to_signed(113, 16); -- sfix16_En15
CONSTANT coeffphase4_4                  : signed(15 DOWNTO 0) :=
    to_signed(-132, 16); -- sfix16_En15
CONSTANT coeffphase4_5                  : signed(15 DOWNTO 0) :=
    to_signed(46, 16); -- sfix16_En15
CONSTANT coeffphase4_6                  : signed(15 DOWNTO 0) :=
    to_signed(210, 16); -- sfix16_En15
CONSTANT coeffphase4_7                  : signed(15 DOWNTO 0) :=
    to_signed(-699, 16); -- sfix16_En15
CONSTANT coeffphase4_8                  : signed(15 DOWNTO 0) :=
    to_signed(1542, 16); -- sfix16_En15
CONSTANT coeffphase4_9                  : signed(15 DOWNTO 0) :=
    to_signed(-3256, 16); -- sfix16_En15
CONSTANT coeffphase4_10                 : signed(15 DOWNTO 0) :=
```

```
    to_signed(15453, 16); -- sfix16_En15
CONSTANT coeffphase4_11                 : signed(15 DOWNTO 0) :=
    to_signed(3897, 16); -- sfix16_En15
CONSTANT coeffphase4_12                 : signed(15 DOWNTO 0) :=
    to_signed(-901, 16); -- sfix16_En15
CONSTANT coeffphase4_13                 : signed(15 DOWNTO 0) :=
    to_signed(99, 16); -- sfix16_En15
CONSTANT coeffphase4_14                 : signed(15 DOWNTO 0) :=
    to_signed(163, 16); -- sfix16_En15
CONSTANT coeffphase4_15                 : signed(15 DOWNTO 0) :=
    to_signed(-187, 16); -- sfix16_En15
CONSTANT coeffphase4_16                 : signed(15 DOWNTO 0) :=
    to_signed(112, 16); -- sfix16_En15
CONSTANT coeffphase4_17                 : signed(15 DOWNTO 0) :=
    to_signed(-19, 16); -- sfix16_En15
CONSTANT coeffphase4_18                 : signed(15 DOWNTO 0) :=
    to_signed(-43, 16); -- sfix16_En15
CONSTANT coeffphase4_19                 : signed(15 DOWNTO 0) :=
    to_signed(60, 16); -- sfix16_En15
CONSTANT coeffphase4_20                 : signed(15 DOWNTO 0) :=
    to_signed(-41, 16); -- sfix16_En15
CONSTANT coeffphase4_21                 : signed(15 DOWNTO 0) :=
    to_signed(0, 16); -- sfix16_En15

-- Signals
SIGNAL ring_count                       : unsigned(3 DOWNTO 0); --
    ufix4
SIGNAL phase_0                          : std_logic; -- boolean
SIGNAL phase_1                          : std_logic; -- boolean
SIGNAL phase_2                          : std_logic; -- boolean
SIGNAL phase_3                          : std_logic; -- boolean
SIGNAL input_typeconvert                : signed(15 DOWNTO 0); --
    sfix16_En15
```

```
SIGNAL input_others                 : signed(15 DOWNTO 0); --
    sfix16_En15
SIGNAL input_pipeline_phase0        : input_pipeline_type(0 TO 20)
    ; -- sfix16_En15
SIGNAL input_pipeline_phase1        : input_pipeline_type(0 TO 19)
    ; -- sfix16_En15
SIGNAL input_pipeline_phase2        : input_pipeline_type(0 TO 19)
    ; -- sfix16_En15
SIGNAL input_pipeline_phase3        : input_pipeline_type(0 TO 19)
    ; -- sfix16_En15

SIGNAL mul_1_01                     : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_1_02                     : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_1_03                     : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_1_04                     : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_1_05                     : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_1_06                     : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_1_07                     : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_1_08                     : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_1_09                     : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_1_10                     : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_1_11                     : signed(31 DOWNTO 0); --
    sfix32_En30;
```

```
SIGNAL mul_1_12                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_1_13                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_1_14                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_1_15                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_1_16                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_1_17                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_1_18                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_1_19                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_1_20                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_1_21                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_2_01                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_2_02                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_2_03                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_2_04                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_2_05                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_2_06                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_2_07                         : signed(31 DOWNTO 0); --
```

```
    sfix32_En30;
SIGNAL mul_2_08                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_2_09                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_2_10                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_2_11                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_2_12                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_2_13                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_2_14                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_2_15                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_2_16                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_2_17                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_2_18                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_2_19                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_2_20                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_3_01                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_3_02                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_3_03                          : signed(31 DOWNTO 0); --
    sfix32_En30;
```

```
SIGNAL mul_3_04                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_3_05                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_3_06                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_3_07                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_3_08                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_3_09                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_3_10                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_3_11                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_3_12                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_3_13                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_3_14                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_3_15                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_3_16                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_3_17                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_3_18                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_3_19                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_3_20                         : signed(31 DOWNTO 0); --
```

```
    sfix32_En30;
SIGNAL mul_4_01                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_4_02                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_4_03                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_4_04                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_4_05                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_4_06                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_4_07                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_4_08                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_4_09                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_4_10                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_4_11                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_4_12                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_4_13                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_4_14                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_4_15                          : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_4_16                          : signed(31 DOWNTO 0); --
    sfix32_En30;
```

```
SIGNAL mul_4_17                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_4_18                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_4_19                         : signed(31 DOWNTO 0); --
    sfix32_En30;
SIGNAL mul_4_20                         : signed(31 DOWNTO 0); --
    sfix32_En30;

SIGNAL prod_1_01                        : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_1_02                        : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_1_03                        : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_1_04                        : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_1_05                        : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_1_06                        : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_1_07                        : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_1_08                        : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_1_09                        : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_1_10                        : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_1_11                        : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_1_12                        : signed(30 DOWNTO 0); --
    sfix31_En30
```

```
SIGNAL prod_1_13                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_1_14                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_1_15                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_1_16                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_1_17                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_1_18                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_1_19                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_1_20                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_1_21                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_2_01                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_2_02                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_2_03                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_2_04                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_2_05                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_2_06                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_2_07                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_2_08                    : signed(30 DOWNTO 0); --
```

```
    sfix31_En30
SIGNAL prod_2_09                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_2_10                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_2_11                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_2_12                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_2_13                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_2_14                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_2_15                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_2_16                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_2_17                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_2_18                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_2_19                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_2_20                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_3_01                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_3_02                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_3_03                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_3_04                    : signed(30 DOWNTO 0); --
    sfix31_En30
```

```
SIGNAL prod_3_05                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_3_06                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_3_07                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_3_08                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_3_09                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_3_10                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_3_11                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_3_12                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_3_13                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_3_14                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_3_15                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_3_16                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_3_17                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_3_18                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_3_19                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_3_20                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_4_01                    : signed(30 DOWNTO 0); --
```

```
    sfix31_En30
SIGNAL prod_4_02                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_4_03                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_4_04                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_4_05                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_4_06                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_4_07                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_4_08                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_4_09                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_4_10                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_4_11                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_4_12                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_4_13                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_4_14                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_4_15                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_4_16                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_4_17                    : signed(30 DOWNTO 0); --
    sfix31_En30
```

```
SIGNAL prod_4_18                        : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_4_19                        : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_4_20                        : signed(30 DOWNTO 0); --
    sfix31_En30

SIGNAL prod_pipe_1_01                   : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_1_02                   : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_1_03                   : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_1_04                   : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_1_05                   : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_1_06                   : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_1_07                   : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_1_08                   : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_1_09                   : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_1_10                   : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_1_11                   : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_1_12                   : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_1_13                   : signed(30 DOWNTO 0); --
    sfix31_En30
```

```
SIGNAL prod_pipe_1_14                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_1_15                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_1_16                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_1_17                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_1_18                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_1_19                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_1_20                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_1_21                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_2_01                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_2_02                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_2_03                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_2_04                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_2_05                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_2_06                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_2_07                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_2_08                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_2_09                    : signed(30 DOWNTO 0); --
```

```
    sfix31_En30
SIGNAL prod_pipe_2_10                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_2_11                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_2_12                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_2_13                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_2_14                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_2_15                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_2_16                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_2_17                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_2_18                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_2_19                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_2_20                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_3_01                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_3_02                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_3_03                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_3_04                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_3_05                : signed(30 DOWNTO 0); --
    sfix31_En30
```

```
SIGNAL prod_pipe_3_06                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_3_07                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_3_08                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_3_09                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_3_10                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_3_11                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_3_12                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_3_13                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_3_14                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_3_15                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_3_16                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_3_17                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_3_18                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_3_19                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_3_20                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_4_01                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_4_02                    : signed(30 DOWNTO 0); --
```

```
    sfix31_En30
SIGNAL prod_pipe_4_03                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_4_04                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_4_05                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_4_06                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_4_07                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_4_08                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_4_09                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_4_10                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_4_11                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_4_12                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_4_13                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_4_14                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_4_15                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_4_16                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_4_17                : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_4_18                : signed(30 DOWNTO 0); --
    sfix31_En30
```

```
SIGNAL prod_pipe_4_19                    : signed(30 DOWNTO 0); --
    sfix31_En30
SIGNAL prod_pipe_4_20                    : signed(30 DOWNTO 0); --
    sfix31_En30

SIGNAL sumvector1                        : sumdelay_pipeline_type(0 TO
    40); -- sfix33_En30

SIGNAL add_cast_1_01                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_02                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_03                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_04                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_05                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_06                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_07                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_08                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_09                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_10                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_11                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_12                     : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_13                     : signed(32 DOWNTO 0); --
```

```
    sfix33_En30
SIGNAL add_cast_1_14                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_15                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_16                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_17                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_18                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_19                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_20                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_21                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_22                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_23                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_24                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_25                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_26                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_27                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_28                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_29                    : signed(32 DOWNTO 0); --
    sfix33_En30
```

```
SIGNAL add_cast_1_30                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_31                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_32                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_33                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_34                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_35                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_36                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_37                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_38                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_39                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_40                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_41                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_42                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_43                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_44                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_45                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_46                    : signed(32 DOWNTO 0); --
```

```
    sfix33_En30
SIGNAL add_cast_1_47                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_48                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_49                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_50                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_51                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_52                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_53                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_54                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_55                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_56                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_57                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_58                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_59                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_60                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_61                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_62                    : signed(32 DOWNTO 0); --
    sfix33_En30
```

```
SIGNAL add_cast_1_63                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_64                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_65                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_66                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_67                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_68                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_69                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_70                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_71                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_72                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_73                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_74                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_75                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_76                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_77                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_78                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_1_79                    : signed(32 DOWNTO 0); --
```

```
    sfix33_En30
SIGNAL add_cast_1_80                : signed(32 DOWNTO 0); --
    sfix33_En30

SIGNAL add_temp_1_01                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_02                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_03                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_04                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_05                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_06                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_07                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_08                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_09                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_10                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_11                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_12                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_13                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_14                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_15                : signed(33 DOWNTO 0); --
```

```
    sfix34_En30
SIGNAL add_temp_1_16                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_17                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_18                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_19                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_20                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_21                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_22                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_23                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_24                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_25                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_26                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_27                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_28                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_29                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_30                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_31                : signed(33 DOWNTO 0); --
    sfix34_En30
```

```
SIGNAL add_temp_1_32                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_33                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_34                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_35                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_36                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_37                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_38                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_39                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_40                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_41                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_42                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_43                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_44                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_45                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_46                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_47                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_48                    : signed(33 DOWNTO 0); --
```

```
    sfix34_En30
SIGNAL add_temp_1_49                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_50                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_51                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_52                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_53                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_54                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_55                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_56                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_57                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_58                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_59                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_60                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_61                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_62                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_63                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_64                : signed(33 DOWNTO 0); --
    sfix34_En30
```

```
SIGNAL add_temp_1_65                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_66                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_67                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_68                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_69                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_70                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_71                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_72                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_73                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_74                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_75                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_76                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_77                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_78                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_79                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_1_80                    : signed(33 DOWNTO 0); --
    sfix34_En30
```

```
SIGNAL sumdelay_pipeline1              : sumdelay_pipeline_type(0 TO
    40); -- sfix33_En30
SIGNAL sumvector2                      : sumdelay_pipeline_type(0 TO
    20); -- sfix33_En30

SIGNAL add_cast_2_01                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_02                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_03                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_04                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_05                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_06                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_07                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_08                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_09                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_10                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_11                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_12                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_13                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_14                   : signed(32 DOWNTO 0); --
    sfix33_En30
```

```
SIGNAL add_cast_2_15                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_16                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_17                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_18                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_19                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_20                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_21                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_22                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_23                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_24                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_25                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_26                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_27                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_28                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_29                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_30                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_31                    : signed(32 DOWNTO 0); --
```

```
    sfix33_En30
SIGNAL add_cast_2_32                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_33                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_34                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_35                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_36                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_37                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_38                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_39                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_2_40                : signed(32 DOWNTO 0); --
    sfix33_En30

SIGNAL add_temp_2_01                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_2_02                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_2_03                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_2_04                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_2_05                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_2_06                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_2_07                : signed(33 DOWNTO 0); --
```

```
    sfix34_En30
SIGNAL add_temp_2_08                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_2_09                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_2_10                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_2_11                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_2_12                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_2_13                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_2_14                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_2_15                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_2_16                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_2_17                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_2_18                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_2_19                : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_2_20                : signed(33 DOWNTO 0); --
    sfix34_En30


SIGNAL sumdelay_pipeline2           : sumdelay_pipeline_type(0 TO
    20); -- sfix33_En30
SIGNAL sumvector3                   : sumdelay_pipeline_type(0 TO
    10); -- sfix33_En30
```

```
SIGNAL add_cast_3_01                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_3_02                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_3_03                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_3_04                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_3_05                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_3_06                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_3_07                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_3_08                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_3_09                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_3_10                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_3_11                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_3_12                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_3_13                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_3_14                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_3_15                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_3_16                : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_3_17                : signed(32 DOWNTO 0); --
```

```
    sfix33_En30
SIGNAL add_cast_3_18                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_3_19                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_3_20                    : signed(32 DOWNTO 0); --
    sfix33_En30

SIGNAL add_temp_3_01                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_3_02                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_3_03                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_3_04                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_3_05                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_3_06                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_3_07                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_3_08                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_3_09                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_3_10                    : signed(33 DOWNTO 0); --
    sfix34_En30

SIGNAL sumdelay_pipeline3               : sumdelay_pipeline_type(0 TO
    10); -- sfix33_En30
SIGNAL sumvector4                       : sumdelay_pipeline_type(0 TO
    5); -- sfix33_En30
```

```
SIGNAL add_cast_4_01                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_4_02                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_4_03                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_4_04                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_4_05                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_4_06                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_4_07                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_4_08                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_4_09                    : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_4_10                    : signed(32 DOWNTO 0); --
    sfix33_En30

SIGNAL add_temp_4_01                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_4_02                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_4_03                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_4_04                    : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_4_05                    : signed(33 DOWNTO 0); --
    sfix34_En30
```

```
SIGNAL sumdelay_pipeline4              : sumdelay_pipeline_type(0 TO
    5); -- sfix33_En30
SIGNAL sumvector5                      : sumdelay_pipeline_type(0 TO
    2); -- sfix33_En30

SIGNAL add_cast_5_01                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_5_02                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_5_03                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_5_04                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_5_05                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_5_06                   : signed(32 DOWNTO 0); --
    sfix33_En30

SIGNAL add_temp_5_01                   : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_5_02                   : signed(33 DOWNTO 0); --
    sfix34_En30
SIGNAL add_temp_5_03                   : signed(33 DOWNTO 0); --
    sfix34_En30

SIGNAL sumdelay_pipeline5              : sumdelay_pipeline_type(0 TO
    2); -- sfix33_En30
SIGNAL sumvector6                      : sumdelay_pipeline_type(0 TO
    1); -- sfix33_En30

SIGNAL add_cast_6_01                   : signed(32 DOWNTO 0); --
    sfix33_En30
SIGNAL add_cast_6_02                   : signed(32 DOWNTO 0); --
```

```
    sfix33_En30

  SIGNAL add_temp_6_01                    : signed(33 DOWNTO 0); --
    sfix34_En30

  SIGNAL sumdelay_pipeline6               : sumdelay_pipeline_type(0 TO
    1); -- sfix33_En30

  SIGNAL add_cast_7_01                    : signed(32 DOWNTO 0); --
    sfix33_En30
  SIGNAL add_cast_7_02                    : signed(32 DOWNTO 0); --
    sfix33_En30

  SIGNAL add_temp_7_01                    : signed(33 DOWNTO 0); --
    sfix34_En30

  SIGNAL sum7                             : std_logic_vector(33 DOWNTO
    0); -- sfix34_En30


BEGIN

  -- Block Statements
  ce_output : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      ring_count <= to_unsigned(1, 4);
    ELSIF clk'event AND clk = '1' THEN
      IF enb_1_1_1 = '1' THEN
        ring_count <= ring_count(0) & ring_count(3 DOWNTO 1);
      END IF;
    END IF;
  END PROCESS ce_output;
```

```
phase_0 <= ring_count(0)  AND enb_1_1_1;

phase_1 <= ring_count(1)  AND enb_1_1_1;

phase_2 <= ring_count(2)  AND enb_1_1_1;

phase_3 <= ring_count(3)  AND enb_1_1_1;

input_typeconvert <= signed(FIR_Decimation_in);

Delay_Pipeline_Phase0_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    input_pipeline_phase0(0 TO 20) <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF phase_0 = '1' THEN
      input_pipeline_phase0(0) <= input_typeconvert;
      input_pipeline_phase0(1 TO 20) <= input_pipeline_phase0(0 TO 19)
          ;
    END IF;
  END IF;
END PROCESS Delay_Pipeline_Phase0_process;

input_others <= input_pipeline_phase0(0);

Delay_Pipeline_Phase1_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    input_pipeline_phase1(0 TO 19) <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF phase_1 = '1' THEN
      input_pipeline_phase1(0) <= input_others;
```

```
      input_pipeline_phase1(1 TO 19) <= input_pipeline_phase1(0 TO 18)
          ;
    END IF;
  END IF;
END PROCESS Delay_Pipeline_Phase1_process;

Delay_Pipeline_Phase2_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    input_pipeline_phase2(0 TO 19) <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF phase_2 = '1' THEN
      input_pipeline_phase2(0) <= input_others;
      input_pipeline_phase2(1 TO 19) <= input_pipeline_phase2(0 TO 18)
          ;
    END IF;
  END IF;
END PROCESS Delay_Pipeline_Phase2_process;

Delay_Pipeline_Phase3_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    input_pipeline_phase3(0 TO 19) <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF phase_3 = '1' THEN
      input_pipeline_phase3(0) <= input_others;
      input_pipeline_phase3(1 TO 19) <= input_pipeline_phase3(0 TO 18)
          ;
    END IF;
  END IF;
END PROCESS Delay_Pipeline_Phase3_process;
```

```
mul_1_01 <= input_pipeline_phase0(0) * coeffphase1_1;
mul_1_02 <= input_pipeline_phase0(1) * coeffphase1_2;
mul_1_03 <= input_pipeline_phase0(2) * coeffphase1_3;
mul_1_04 <= input_pipeline_phase0(3) * coeffphase1_4;
mul_1_05 <= input_pipeline_phase0(4) * coeffphase1_5;
mul_1_06 <= input_pipeline_phase0(5) * coeffphase1_6;
mul_1_07 <= input_pipeline_phase0(6) * coeffphase1_7;
mul_1_08 <= input_pipeline_phase0(7) * coeffphase1_8;
mul_1_09 <= input_pipeline_phase0(8) * coeffphase1_9;
mul_1_10 <= input_pipeline_phase0(9) * coeffphase1_10;
mul_1_11 <= input_pipeline_phase0(10) * coeffphase1_11;
mul_1_12 <= input_pipeline_phase0(11) * coeffphase1_12;
mul_1_13 <= input_pipeline_phase0(12) * coeffphase1_13;
mul_1_14 <= input_pipeline_phase0(13) * coeffphase1_14;
mul_1_15 <= input_pipeline_phase0(14) * coeffphase1_15;
mul_1_16 <= input_pipeline_phase0(15) * coeffphase1_16;
mul_1_17 <= input_pipeline_phase0(16) * coeffphase1_17;
mul_1_18 <= input_pipeline_phase0(17) * coeffphase1_18;
mul_1_19 <= input_pipeline_phase0(18) * coeffphase1_19;
mul_1_20 <= input_pipeline_phase0(19) * coeffphase1_20;
mul_1_21 <= input_pipeline_phase0(20) * coeffphase1_21;

mul_2_01 <= input_pipeline_phase1(0) * coeffphase2_1;
mul_2_02 <= input_pipeline_phase1(1) * coeffphase2_2;
mul_2_03 <= input_pipeline_phase1(2) * coeffphase2_3;
mul_2_04 <= input_pipeline_phase1(3) * coeffphase2_4;
mul_2_05 <= input_pipeline_phase1(4) * coeffphase2_5;
mul_2_06 <= input_pipeline_phase1(5) * coeffphase2_6;
mul_2_07 <= input_pipeline_phase1(6) * coeffphase2_7;
mul_2_08 <= input_pipeline_phase1(7) * coeffphase2_8;
mul_2_09 <= input_pipeline_phase1(8) * coeffphase2_9;
mul_2_10 <= input_pipeline_phase1(9) * coeffphase2_10;
mul_2_11 <= input_pipeline_phase1(10) * coeffphase2_11;
```

```
mul_2_12 <= input_pipeline_phase1(11) * coeffphase2_12;
mul_2_13 <= input_pipeline_phase1(12) * coeffphase2_13;
mul_2_14 <= input_pipeline_phase1(13) * coeffphase2_14;
mul_2_15 <= input_pipeline_phase1(14) * coeffphase2_15;
mul_2_16 <= input_pipeline_phase1(15) * coeffphase2_16;
mul_2_17 <= input_pipeline_phase1(16) * coeffphase2_17;
mul_2_18 <= input_pipeline_phase1(17) * coeffphase2_18;
mul_2_19 <= input_pipeline_phase1(18) * coeffphase2_19;
mul_2_20 <= input_pipeline_phase1(19) * coeffphase2_20;

mul_3_01 <= input_pipeline_phase2(0) * coeffphase3_1;
mul_3_02 <= input_pipeline_phase2(1) * coeffphase3_2;
mul_3_03 <= input_pipeline_phase2(2) * coeffphase3_3;
mul_3_04 <= input_pipeline_phase2(3) * coeffphase3_4;
mul_3_05 <= input_pipeline_phase2(4) * coeffphase3_5;
mul_3_06 <= input_pipeline_phase2(5) * coeffphase3_6;
mul_3_07 <= input_pipeline_phase2(6) * coeffphase3_7;
mul_3_08 <= input_pipeline_phase2(7) * coeffphase3_8;
mul_3_09 <= input_pipeline_phase2(8) * coeffphase3_9;
mul_3_10 <= input_pipeline_phase2(9) * coeffphase3_10;
mul_3_11 <= input_pipeline_phase2(10) * coeffphase3_11;
mul_3_12 <= input_pipeline_phase2(11) * coeffphase3_12;
mul_3_13 <= input_pipeline_phase2(12) * coeffphase3_13;
mul_3_14 <= input_pipeline_phase2(13) * coeffphase3_14;
mul_3_15 <= input_pipeline_phase2(14) * coeffphase3_15;
mul_3_16 <= input_pipeline_phase2(15) * coeffphase3_16;
mul_3_17 <= input_pipeline_phase2(16) * coeffphase3_17;
mul_3_18 <= input_pipeline_phase2(17) * coeffphase3_18;
mul_3_19 <= input_pipeline_phase2(18) * coeffphase3_19;
mul_3_20 <= input_pipeline_phase2(19) * coeffphase3_20;

mul_4_01 <= input_pipeline_phase3(0) * coeffphase4_1;
mul_4_02 <= input_pipeline_phase3(1) * coeffphase4_2;
```

```
mul_4_03 <= input_pipeline_phase3(2) * coeffphase4_3;
mul_4_04 <= input_pipeline_phase3(3) * coeffphase4_4;
mul_4_05 <= input_pipeline_phase3(4) * coeffphase4_5;
mul_4_06 <= input_pipeline_phase3(5) * coeffphase4_6;
mul_4_07 <= input_pipeline_phase3(6) * coeffphase4_7;
mul_4_08 <= input_pipeline_phase3(7) * coeffphase4_8;
mul_4_09 <= input_pipeline_phase3(8) * coeffphase4_9;
mul_4_10 <= input_pipeline_phase3(9) * coeffphase4_10;
mul_4_11 <= input_pipeline_phase3(10) * coeffphase4_11;
mul_4_12 <= input_pipeline_phase3(11) * coeffphase4_12;
mul_4_13 <= input_pipeline_phase3(12) * coeffphase4_13;
mul_4_14 <= input_pipeline_phase3(13) * coeffphase4_14;
mul_4_15 <= input_pipeline_phase3(14) * coeffphase4_15;
mul_4_16 <= input_pipeline_phase3(15) * coeffphase4_16;
mul_4_17 <= input_pipeline_phase3(16) * coeffphase4_17;
mul_4_18 <= input_pipeline_phase3(17) * coeffphase4_18;
mul_4_19 <= input_pipeline_phase3(18) * coeffphase4_19;
mul_4_20 <= input_pipeline_phase3(19) * coeffphase4_20;


prod_1_01 <= (30 => '0', OTHERS => '1') WHEN mul_1_01(31) = '0' AND
    mul_1_01(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_1_01(31) = '1' AND
        mul_1_01(30) /= '1'
    ELSE (mul_1_01(30 DOWNTO 0));
prod_1_02 <= (30 => '0', OTHERS => '1') WHEN mul_1_02(31) = '0' AND
    mul_1_02(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_1_02(31) = '1' AND
        mul_1_02(30) /= '1'
    ELSE (mul_1_02(30 DOWNTO 0));
prod_1_03 <= (30 => '0', OTHERS => '1') WHEN mul_1_03(31) = '0' AND
    mul_1_03(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_1_03(31) = '1' AND
```

```
        mul_1_03(30) /= '1'
    ELSE (mul_1_03(30 DOWNTO 0));
prod_1_04 <= (30 => '0', OTHERS => '1') WHEN mul_1_04(31) = '0' AND
    mul_1_04(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_1_04(31) = '1' AND
        mul_1_04(30) /= '1'
    ELSE (mul_1_04(30 DOWNTO 0));
prod_1_05 <= (30 => '0', OTHERS => '1') WHEN mul_1_05(31) = '0' AND
    mul_1_05(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_1_05(31) = '1' AND
        mul_1_05(30) /= '1'
    ELSE (mul_1_05(30 DOWNTO 0));
prod_1_06 <= (30 => '0', OTHERS => '1') WHEN mul_1_06(31) = '0' AND
    mul_1_06(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_1_06(31) = '1' AND
        mul_1_06(30) /= '1'
    ELSE (mul_1_06(30 DOWNTO 0));
prod_1_07 <= (30 => '0', OTHERS => '1') WHEN mul_1_07(31) = '0' AND
    mul_1_07(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_1_07(31) = '1' AND
        mul_1_07(30) /= '1'
    ELSE (mul_1_07(30 DOWNTO 0));
prod_1_08 <= (30 => '0', OTHERS => '1') WHEN mul_1_08(31) = '0' AND
    mul_1_08(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_1_08(31) = '1' AND
        mul_1_08(30) /= '1'
    ELSE (mul_1_08(30 DOWNTO 0));
prod_1_09 <= (30 => '0', OTHERS => '1') WHEN mul_1_09(31) = '0' AND
    mul_1_09(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_1_09(31) = '1' AND
        mul_1_09(30) /= '1'
    ELSE (mul_1_09(30 DOWNTO 0));
prod_1_10 <= (30 => '0', OTHERS => '1') WHEN mul_1_10(31) = '0' AND
```

```
    mul_1_10(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_1_10(31) = '1' AND
        mul_1_10(30) /= '1'
    ELSE (mul_1_10(30 DOWNTO 0));
prod_1_11 <= (30 => '0', OTHERS => '1') WHEN mul_1_11(31) = '0' AND
    mul_1_11(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_1_11(31) = '1' AND
        mul_1_11(30) /= '1'
    ELSE (mul_1_11(30 DOWNTO 0));
prod_1_12 <= (30 => '0', OTHERS => '1') WHEN mul_1_12(31) = '0' AND
    mul_1_12(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_1_12(31) = '1' AND
        mul_1_12(30) /= '1'
    ELSE (mul_1_12(30 DOWNTO 0));
prod_1_13 <= (30 => '0', OTHERS => '1') WHEN mul_1_13(31) = '0' AND
    mul_1_13(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_1_13(31) = '1' AND
        mul_1_13(30) /= '1'
    ELSE (mul_1_13(30 DOWNTO 0));
prod_1_14 <= (30 => '0', OTHERS => '1') WHEN mul_1_14(31) = '0' AND
    mul_1_14(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_1_14(31) = '1' AND
        mul_1_14(30) /= '1'
    ELSE (mul_1_14(30 DOWNTO 0));
prod_1_15 <= (30 => '0', OTHERS => '1') WHEN mul_1_15(31) = '0' AND
    mul_1_15(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_1_15(31) = '1' AND
        mul_1_15(30) /= '1'
    ELSE (mul_1_15(30 DOWNTO 0));
prod_1_16 <= (30 => '0', OTHERS => '1') WHEN mul_1_16(31) = '0' AND
    mul_1_16(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_1_16(31) = '1' AND
        mul_1_16(30) /= '1'
```

```
    ELSE (mul_1_16(30 DOWNTO 0));
prod_1_17 <= (30 => '0', OTHERS => '1') WHEN mul_1_17(31) = '0' AND
    mul_1_17(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_1_17(31) = '1' AND
        mul_1_17(30) /= '1'
    ELSE (mul_1_17(30 DOWNTO 0));
prod_1_18 <= (30 => '0', OTHERS => '1') WHEN mul_1_18(31) = '0' AND
    mul_1_18(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_1_18(31) = '1' AND
        mul_1_18(30) /= '1'
    ELSE (mul_1_18(30 DOWNTO 0));
prod_1_19 <= (30 => '0', OTHERS => '1') WHEN mul_1_19(31) = '0' AND
    mul_1_19(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_1_19(31) = '1' AND
        mul_1_19(30) /= '1'
    ELSE (mul_1_19(30 DOWNTO 0));
prod_1_20 <= (30 => '0', OTHERS => '1') WHEN mul_1_20(31) = '0' AND
    mul_1_20(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_1_20(31) = '1' AND
        mul_1_20(30) /= '1'
    ELSE (mul_1_20(30 DOWNTO 0));
prod_1_21 <= (30 => '0', OTHERS => '1') WHEN mul_1_21(31) = '0' AND
    mul_1_21(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_1_21(31) = '1' AND
        mul_1_21(30) /= '1'
    ELSE (mul_1_21(30 DOWNTO 0));
prod_2_01 <= (30 => '0', OTHERS => '1') WHEN mul_2_01(31) = '0' AND
    mul_2_01(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_2_01(31) = '1' AND
        mul_2_01(30) /= '1'
    ELSE (mul_2_01(30 DOWNTO 0));
prod_2_02 <= (30 => '0', OTHERS => '1') WHEN mul_2_02(31) = '0' AND
    mul_2_02(30) /= '0'
```

```
    ELSE (30 => '1', OTHERS => '0') WHEN mul_2_02(31) = '1' AND
        mul_2_02(30) /= '1'
    ELSE (mul_2_02(30 DOWNTO 0));
prod_2_03 <= (30 => '0', OTHERS => '1') WHEN mul_2_03(31) = '0' AND
    mul_2_03(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_2_03(31) = '1' AND
        mul_2_03(30) /= '1'
    ELSE (mul_2_03(30 DOWNTO 0));
prod_2_04 <= (30 => '0', OTHERS => '1') WHEN mul_2_04(31) = '0' AND
    mul_2_04(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_2_04(31) = '1' AND
        mul_2_04(30) /= '1'
    ELSE (mul_2_04(30 DOWNTO 0));
prod_2_05 <= (30 => '0', OTHERS => '1') WHEN mul_2_05(31) = '0' AND
    mul_2_05(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_2_05(31) = '1' AND
        mul_2_05(30) /= '1'
    ELSE (mul_2_05(30 DOWNTO 0));
prod_2_06 <= (30 => '0', OTHERS => '1') WHEN mul_2_06(31) = '0' AND
    mul_2_06(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_2_06(31) = '1' AND
        mul_2_06(30) /= '1'
    ELSE (mul_2_06(30 DOWNTO 0));
prod_2_07 <= (30 => '0', OTHERS => '1') WHEN mul_2_07(31) = '0' AND
    mul_2_07(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_2_07(31) = '1' AND
        mul_2_07(30) /= '1'
    ELSE (mul_2_07(30 DOWNTO 0));
prod_2_08 <= (30 => '0', OTHERS => '1') WHEN mul_2_08(31) = '0' AND
    mul_2_08(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_2_08(31) = '1' AND
        mul_2_08(30) /= '1'
    ELSE (mul_2_08(30 DOWNTO 0));
```

```
prod_2_09 <= (30 => '0', OTHERS => '1') WHEN mul_2_09(31) = '0' AND
    mul_2_09(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_2_09(31) = '1' AND
        mul_2_09(30) /= '1'
    ELSE (mul_2_09(30 DOWNTO 0));
prod_2_10 <= (30 => '0', OTHERS => '1') WHEN mul_2_10(31) = '0' AND
    mul_2_10(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_2_10(31) = '1' AND
        mul_2_10(30) /= '1'
    ELSE (mul_2_10(30 DOWNTO 0));
prod_2_11 <= (30 => '0', OTHERS => '1') WHEN mul_2_11(31) = '0' AND
    mul_2_11(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_2_11(31) = '1' AND
        mul_2_11(30) /= '1'
    ELSE (mul_2_11(30 DOWNTO 0));
prod_2_12 <= (30 => '0', OTHERS => '1') WHEN mul_2_12(31) = '0' AND
    mul_2_12(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_2_12(31) = '1' AND
        mul_2_12(30) /= '1'
    ELSE (mul_2_12(30 DOWNTO 0));
prod_2_13 <= (30 => '0', OTHERS => '1') WHEN mul_2_13(31) = '0' AND
    mul_2_13(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_2_13(31) = '1' AND
        mul_2_13(30) /= '1'
    ELSE (mul_2_13(30 DOWNTO 0));
prod_2_14 <= (30 => '0', OTHERS => '1') WHEN mul_2_14(31) = '0' AND
    mul_2_14(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_2_14(31) = '1' AND
        mul_2_14(30) /= '1'
    ELSE (mul_2_14(30 DOWNTO 0));
prod_2_15 <= (30 => '0', OTHERS => '1') WHEN mul_2_15(31) = '0' AND
    mul_2_15(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_2_15(31) = '1' AND
```

```
        mul_2_15(30) /= '1'
    ELSE (mul_2_15(30 DOWNTO 0));
prod_2_16 <= (30 => '0', OTHERS => '1') WHEN mul_2_16(31) = '0' AND
    mul_2_16(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_2_16(31) = '1' AND
        mul_2_16(30) /= '1'
    ELSE (mul_2_16(30 DOWNTO 0));
prod_2_17 <= (30 => '0', OTHERS => '1') WHEN mul_2_17(31) = '0' AND
    mul_2_17(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_2_17(31) = '1' AND
        mul_2_17(30) /= '1'
    ELSE (mul_2_17(30 DOWNTO 0));
prod_2_18 <= (30 => '0', OTHERS => '1') WHEN mul_2_18(31) = '0' AND
    mul_2_18(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_2_18(31) = '1' AND
        mul_2_18(30) /= '1'
    ELSE (mul_2_18(30 DOWNTO 0));
prod_2_19 <= (30 => '0', OTHERS => '1') WHEN mul_2_19(31) = '0' AND
    mul_2_19(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_2_19(31) = '1' AND
        mul_2_19(30) /= '1'
    ELSE (mul_2_19(30 DOWNTO 0));
prod_2_20 <= (30 => '0', OTHERS => '1') WHEN mul_2_20(31) = '0' AND
    mul_2_20(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_2_20(31) = '1' AND
        mul_2_20(30) /= '1'
    ELSE (mul_2_20(30 DOWNTO 0));
prod_3_01 <= (30 => '0', OTHERS => '1') WHEN mul_3_01(31) = '0' AND
    mul_3_01(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_3_01(31) = '1' AND
        mul_3_01(30) /= '1'
    ELSE (mul_3_01(30 DOWNTO 0));
prod_3_02 <= (30 => '0', OTHERS => '1') WHEN mul_3_02(31) = '0' AND
```

```
    mul_3_02(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_3_02(31) = '1' AND
        mul_3_02(30) /= '1'
    ELSE (mul_3_02(30 DOWNTO 0));
prod_3_03 <= (30 => '0', OTHERS => '1') WHEN mul_3_03(31) = '0' AND
    mul_3_03(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_3_03(31) = '1' AND
        mul_3_03(30) /= '1'
    ELSE (mul_3_03(30 DOWNTO 0));
prod_3_04 <= (30 => '0', OTHERS => '1') WHEN mul_3_04(31) = '0' AND
    mul_3_04(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_3_04(31) = '1' AND
        mul_3_04(30) /= '1'
    ELSE (mul_3_04(30 DOWNTO 0));
prod_3_05 <= (30 => '0', OTHERS => '1') WHEN mul_3_05(31) = '0' AND
    mul_3_05(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_3_05(31) = '1' AND
        mul_3_05(30) /= '1'
    ELSE (mul_3_05(30 DOWNTO 0));
prod_3_06 <= (30 => '0', OTHERS => '1') WHEN mul_3_06(31) = '0' AND
    mul_3_06(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_3_06(31) = '1' AND
        mul_3_06(30) /= '1'
    ELSE (mul_3_06(30 DOWNTO 0));
prod_3_07 <= (30 => '0', OTHERS => '1') WHEN mul_3_07(31) = '0' AND
    mul_3_07(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_3_07(31) = '1' AND
        mul_3_07(30) /= '1'
    ELSE (mul_3_07(30 DOWNTO 0));
prod_3_08 <= (30 => '0', OTHERS => '1') WHEN mul_3_08(31) = '0' AND
    mul_3_08(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_3_08(31) = '1' AND
        mul_3_08(30) /= '1'
```

```
    ELSE (mul_3_08(30 DOWNTO 0));
prod_3_09 <= (30 => '0', OTHERS => '1') WHEN mul_3_09(31) = '0' AND
    mul_3_09(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_3_09(31) = '1' AND
        mul_3_09(30) /= '1'
    ELSE (mul_3_09(30 DOWNTO 0));
prod_3_10 <= (30 => '0', OTHERS => '1') WHEN mul_3_10(31) = '0' AND
    mul_3_10(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_3_10(31) = '1' AND
        mul_3_10(30) /= '1'
    ELSE (mul_3_10(30 DOWNTO 0));
prod_3_11 <= (30 => '0', OTHERS => '1') WHEN mul_3_11(31) = '0' AND
    mul_3_11(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_3_11(31) = '1' AND
        mul_3_11(30) /= '1'
    ELSE (mul_3_11(30 DOWNTO 0));
prod_3_12 <= (30 => '0', OTHERS => '1') WHEN mul_3_12(31) = '0' AND
    mul_3_12(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_3_12(31) = '1' AND
        mul_3_12(30) /= '1'
    ELSE (mul_3_12(30 DOWNTO 0));
prod_3_13 <= (30 => '0', OTHERS => '1') WHEN mul_3_13(31) = '0' AND
    mul_3_13(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_3_13(31) = '1' AND
        mul_3_13(30) /= '1'
    ELSE (mul_3_13(30 DOWNTO 0));
prod_3_14 <= (30 => '0', OTHERS => '1') WHEN mul_3_14(31) = '0' AND
    mul_3_14(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_3_14(31) = '1' AND
        mul_3_14(30) /= '1'
    ELSE (mul_3_14(30 DOWNTO 0));
prod_3_15 <= (30 => '0', OTHERS => '1') WHEN mul_3_15(31) = '0' AND
    mul_3_15(30) /= '0'
```

```
    ELSE (30 => '1', OTHERS => '0') WHEN mul_3_15(31) = '1' AND
        mul_3_15(30) /= '1'
    ELSE (mul_3_15(30 DOWNTO 0));
prod_3_16 <= (30 => '0', OTHERS => '1') WHEN mul_3_16(31) = '0' AND
    mul_3_16(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_3_16(31) = '1' AND
        mul_3_16(30) /= '1'
    ELSE (mul_3_16(30 DOWNTO 0));
prod_3_17 <= (30 => '0', OTHERS => '1') WHEN mul_3_17(31) = '0' AND
    mul_3_17(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_3_17(31) = '1' AND
        mul_3_17(30) /= '1'
    ELSE (mul_3_17(30 DOWNTO 0));
prod_3_18 <= (30 => '0', OTHERS => '1') WHEN mul_3_18(31) = '0' AND
    mul_3_18(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_3_18(31) = '1' AND
        mul_3_18(30) /= '1'
    ELSE (mul_3_18(30 DOWNTO 0));
prod_3_19 <= (30 => '0', OTHERS => '1') WHEN mul_3_19(31) = '0' AND
    mul_3_19(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_3_19(31) = '1' AND
        mul_3_19(30) /= '1'
    ELSE (mul_3_19(30 DOWNTO 0));
prod_3_20 <= (30 => '0', OTHERS => '1') WHEN mul_3_20(31) = '0' AND
    mul_3_20(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_3_20(31) = '1' AND
        mul_3_20(30) /= '1'
    ELSE (mul_3_20(30 DOWNTO 0));
prod_4_01 <= (30 => '0', OTHERS => '1') WHEN mul_4_01(31) = '0' AND
    mul_4_01(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_4_01(31) = '1' AND
        mul_4_01(30) /= '1'
    ELSE (mul_4_01(30 DOWNTO 0));
```

```
prod_4_02 <= (30 => '0', OTHERS => '1') WHEN mul_4_02(31) = '0' AND
    mul_4_02(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_4_02(31) = '1' AND
        mul_4_02(30) /= '1'
    ELSE (mul_4_02(30 DOWNTO 0));
prod_4_03 <= (30 => '0', OTHERS => '1') WHEN mul_4_03(31) = '0' AND
    mul_4_03(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_4_03(31) = '1' AND
        mul_4_03(30) /= '1'
    ELSE (mul_4_03(30 DOWNTO 0));
prod_4_04 <= (30 => '0', OTHERS => '1') WHEN mul_4_04(31) = '0' AND
    mul_4_04(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_4_04(31) = '1' AND
        mul_4_04(30) /= '1'
    ELSE (mul_4_04(30 DOWNTO 0));
prod_4_05 <= (30 => '0', OTHERS => '1') WHEN mul_4_05(31) = '0' AND
    mul_4_05(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_4_05(31) = '1' AND
        mul_4_05(30) /= '1'
    ELSE (mul_4_05(30 DOWNTO 0));
prod_4_06 <= (30 => '0', OTHERS => '1') WHEN mul_4_06(31) = '0' AND
    mul_4_06(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_4_06(31) = '1' AND
        mul_4_06(30) /= '1'
    ELSE (mul_4_06(30 DOWNTO 0));
prod_4_07 <= (30 => '0', OTHERS => '1') WHEN mul_4_07(31) = '0' AND
    mul_4_07(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_4_07(31) = '1' AND
        mul_4_07(30) /= '1'
    ELSE (mul_4_07(30 DOWNTO 0));
prod_4_08 <= (30 => '0', OTHERS => '1') WHEN mul_4_08(31) = '0' AND
    mul_4_08(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_4_08(31) = '1' AND
```

```
        mul_4_08(30) /= '1'
    ELSE (mul_4_08(30 DOWNTO 0));
prod_4_09 <= (30 => '0', OTHERS => '1') WHEN mul_4_09(31) = '0' AND
    mul_4_09(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_4_09(31) = '1' AND
        mul_4_09(30) /= '1'
    ELSE (mul_4_09(30 DOWNTO 0));
prod_4_10 <= (30 => '0', OTHERS => '1') WHEN mul_4_10(31) = '0' AND
    mul_4_10(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_4_10(31) = '1' AND
        mul_4_10(30) /= '1'
    ELSE (mul_4_10(30 DOWNTO 0));
prod_4_11 <= (30 => '0', OTHERS => '1') WHEN mul_4_11(31) = '0' AND
    mul_4_11(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_4_11(31) = '1' AND
        mul_4_11(30) /= '1'
    ELSE (mul_4_11(30 DOWNTO 0));
prod_4_12 <= (30 => '0', OTHERS => '1') WHEN mul_4_12(31) = '0' AND
    mul_4_12(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_4_12(31) = '1' AND
        mul_4_12(30) /= '1'
    ELSE (mul_4_12(30 DOWNTO 0));
prod_4_13 <= (30 => '0', OTHERS => '1') WHEN mul_4_13(31) = '0' AND
    mul_4_13(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_4_13(31) = '1' AND
        mul_4_13(30) /= '1'
    ELSE (mul_4_13(30 DOWNTO 0));
prod_4_14 <= (30 => '0', OTHERS => '1') WHEN mul_4_14(31) = '0' AND
    mul_4_14(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_4_14(31) = '1' AND
        mul_4_14(30) /= '1'
    ELSE (mul_4_14(30 DOWNTO 0));
prod_4_15 <= (30 => '0', OTHERS => '1') WHEN mul_4_15(31) = '0' AND
```

```
    mul_4_15(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_4_15(31) = '1' AND
        mul_4_15(30) /= '1'
    ELSE (mul_4_15(30 DOWNTO 0));
prod_4_16 <= (30 => '0', OTHERS => '1') WHEN mul_4_16(31) = '0' AND
    mul_4_16(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_4_16(31) = '1' AND
        mul_4_16(30) /= '1'
    ELSE (mul_4_16(30 DOWNTO 0));
prod_4_17 <= (30 => '0', OTHERS => '1') WHEN mul_4_17(31) = '0' AND
    mul_4_17(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_4_17(31) = '1' AND
        mul_4_17(30) /= '1'
    ELSE (mul_4_17(30 DOWNTO 0));
prod_4_18 <= (30 => '0', OTHERS => '1') WHEN mul_4_18(31) = '0' AND
    mul_4_18(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_4_18(31) = '1' AND
        mul_4_18(30) /= '1'
    ELSE (mul_4_18(30 DOWNTO 0));
prod_4_19 <= (30 => '0', OTHERS => '1') WHEN mul_4_19(31) = '0' AND
    mul_4_19(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_4_19(31) = '1' AND
        mul_4_19(30) /= '1'
    ELSE (mul_4_19(30 DOWNTO 0));
prod_4_20 <= (30 => '0', OTHERS => '1') WHEN mul_4_20(31) = '0' AND
    mul_4_20(30) /= '0'
    ELSE (30 => '1', OTHERS => '0') WHEN mul_4_20(31) = '1' AND
        mul_4_20(30) /= '1'
    ELSE (mul_4_20(30 DOWNTO 0));


prod_pipe_process : PROCESS (clk, reset)
BEGIN
```

```
IF reset = '1' THEN
  prod_pipe_1_01 <= (OTHERS => '0');
  prod_pipe_1_02 <= (OTHERS => '0');
  prod_pipe_1_03 <= (OTHERS => '0');
  prod_pipe_1_04 <= (OTHERS => '0');
  prod_pipe_1_05 <= (OTHERS => '0');
  prod_pipe_1_06 <= (OTHERS => '0');
  prod_pipe_1_07 <= (OTHERS => '0');
  prod_pipe_1_08 <= (OTHERS => '0');
  prod_pipe_1_09 <= (OTHERS => '0');
  prod_pipe_1_10 <= (OTHERS => '0');
  prod_pipe_1_11 <= (OTHERS => '0');
  prod_pipe_1_12 <= (OTHERS => '0');
  prod_pipe_1_13 <= (OTHERS => '0');
  prod_pipe_1_14 <= (OTHERS => '0');
  prod_pipe_1_15 <= (OTHERS => '0');
  prod_pipe_1_16 <= (OTHERS => '0');
  prod_pipe_1_17 <= (OTHERS => '0');
  prod_pipe_1_18 <= (OTHERS => '0');
  prod_pipe_1_19 <= (OTHERS => '0');
  prod_pipe_1_20 <= (OTHERS => '0');
  prod_pipe_1_21 <= (OTHERS => '0');
  prod_pipe_2_01 <= (OTHERS => '0');
  prod_pipe_2_02 <= (OTHERS => '0');
  prod_pipe_2_03 <= (OTHERS => '0');
  prod_pipe_2_04 <= (OTHERS => '0');
  prod_pipe_2_05 <= (OTHERS => '0');
  prod_pipe_2_06 <= (OTHERS => '0');
  prod_pipe_2_07 <= (OTHERS => '0');
  prod_pipe_2_08 <= (OTHERS => '0');
  prod_pipe_2_09 <= (OTHERS => '0');
  prod_pipe_2_10 <= (OTHERS => '0');
  prod_pipe_2_11 <= (OTHERS => '0');
```

```
prod_pipe_2_12 <= (OTHERS => '0');
prod_pipe_2_13 <= (OTHERS => '0');
prod_pipe_2_14 <= (OTHERS => '0');
prod_pipe_2_15 <= (OTHERS => '0');
prod_pipe_2_16 <= (OTHERS => '0');
prod_pipe_2_17 <= (OTHERS => '0');
prod_pipe_2_18 <= (OTHERS => '0');
prod_pipe_2_19 <= (OTHERS => '0');
prod_pipe_2_20 <= (OTHERS => '0');
prod_pipe_3_01 <= (OTHERS => '0');
prod_pipe_3_02 <= (OTHERS => '0');
prod_pipe_3_03 <= (OTHERS => '0');
prod_pipe_3_04 <= (OTHERS => '0');
prod_pipe_3_05 <= (OTHERS => '0');
prod_pipe_3_06 <= (OTHERS => '0');
prod_pipe_3_07 <= (OTHERS => '0');
prod_pipe_3_08 <= (OTHERS => '0');
prod_pipe_3_09 <= (OTHERS => '0');
prod_pipe_3_10 <= (OTHERS => '0');
prod_pipe_3_11 <= (OTHERS => '0');
prod_pipe_3_12 <= (OTHERS => '0');
prod_pipe_3_13 <= (OTHERS => '0');
prod_pipe_3_14 <= (OTHERS => '0');
prod_pipe_3_15 <= (OTHERS => '0');
prod_pipe_3_16 <= (OTHERS => '0');
prod_pipe_3_17 <= (OTHERS => '0');
prod_pipe_3_18 <= (OTHERS => '0');
prod_pipe_3_19 <= (OTHERS => '0');
prod_pipe_3_20 <= (OTHERS => '0');
prod_pipe_4_01 <= (OTHERS => '0');
prod_pipe_4_02 <= (OTHERS => '0');
prod_pipe_4_03 <= (OTHERS => '0');
prod_pipe_4_04 <= (OTHERS => '0');
```

```
    prod_pipe_4_05 <= (OTHERS => '0');
    prod_pipe_4_06 <= (OTHERS => '0');
    prod_pipe_4_07 <= (OTHERS => '0');
    prod_pipe_4_08 <= (OTHERS => '0');
    prod_pipe_4_09 <= (OTHERS => '0');
    prod_pipe_4_10 <= (OTHERS => '0');
    prod_pipe_4_11 <= (OTHERS => '0');
    prod_pipe_4_12 <= (OTHERS => '0');
    prod_pipe_4_13 <= (OTHERS => '0');
    prod_pipe_4_14 <= (OTHERS => '0');
    prod_pipe_4_15 <= (OTHERS => '0');
    prod_pipe_4_16 <= (OTHERS => '0');
    prod_pipe_4_17 <= (OTHERS => '0');
    prod_pipe_4_18 <= (OTHERS => '0');
    prod_pipe_4_19 <= (OTHERS => '0');
    prod_pipe_4_20 <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF phase_0 = '1' THEN
      prod_pipe_1_01 <= prod_1_01;
      prod_pipe_1_02 <= prod_1_02;
      prod_pipe_1_03 <= prod_1_03;
      prod_pipe_1_04 <= prod_1_04;
      prod_pipe_1_05 <= prod_1_05;
      prod_pipe_1_06 <= prod_1_06;
      prod_pipe_1_07 <= prod_1_07;
      prod_pipe_1_08 <= prod_1_08;
      prod_pipe_1_09 <= prod_1_09;
      prod_pipe_1_10 <= prod_1_10;
      prod_pipe_1_11 <= prod_1_11;
      prod_pipe_1_12 <= prod_1_12;
      prod_pipe_1_13 <= prod_1_13;
      prod_pipe_1_14 <= prod_1_14;
      prod_pipe_1_15 <= prod_1_15;
```

```
prod_pipe_1_16 <= prod_1_16;
prod_pipe_1_17 <= prod_1_17;
prod_pipe_1_18 <= prod_1_18;
prod_pipe_1_19 <= prod_1_19;
prod_pipe_1_20 <= prod_1_20;
prod_pipe_1_21 <= prod_1_21;
prod_pipe_2_01 <= prod_2_01;
prod_pipe_2_02 <= prod_2_02;
prod_pipe_2_03 <= prod_2_03;
prod_pipe_2_04 <= prod_2_04;
prod_pipe_2_05 <= prod_2_05;
prod_pipe_2_06 <= prod_2_06;
prod_pipe_2_07 <= prod_2_07;
prod_pipe_2_08 <= prod_2_08;
prod_pipe_2_09 <= prod_2_09;
prod_pipe_2_10 <= prod_2_10;
prod_pipe_2_11 <= prod_2_11;
prod_pipe_2_12 <= prod_2_12;
prod_pipe_2_13 <= prod_2_13;
prod_pipe_2_14 <= prod_2_14;
prod_pipe_2_15 <= prod_2_15;
prod_pipe_2_16 <= prod_2_16;
prod_pipe_2_17 <= prod_2_17;
prod_pipe_2_18 <= prod_2_18;
prod_pipe_2_19 <= prod_2_19;
prod_pipe_2_20 <= prod_2_20;
prod_pipe_3_01 <= prod_3_01;
prod_pipe_3_02 <= prod_3_02;
prod_pipe_3_03 <= prod_3_03;
prod_pipe_3_04 <= prod_3_04;
prod_pipe_3_05 <= prod_3_05;
prod_pipe_3_06 <= prod_3_06;
prod_pipe_3_07 <= prod_3_07;
```

```
prod_pipe_3_08 <= prod_3_08;
prod_pipe_3_09 <= prod_3_09;
prod_pipe_3_10 <= prod_3_10;
prod_pipe_3_11 <= prod_3_11;
prod_pipe_3_12 <= prod_3_12;
prod_pipe_3_13 <= prod_3_13;
prod_pipe_3_14 <= prod_3_14;
prod_pipe_3_15 <= prod_3_15;
prod_pipe_3_16 <= prod_3_16;
prod_pipe_3_17 <= prod_3_17;
prod_pipe_3_18 <= prod_3_18;
prod_pipe_3_19 <= prod_3_19;
prod_pipe_3_20 <= prod_3_20;
prod_pipe_4_01 <= prod_4_01;
prod_pipe_4_02 <= prod_4_02;
prod_pipe_4_03 <= prod_4_03;
prod_pipe_4_04 <= prod_4_04;
prod_pipe_4_05 <= prod_4_05;
prod_pipe_4_06 <= prod_4_06;
prod_pipe_4_07 <= prod_4_07;
prod_pipe_4_08 <= prod_4_08;
prod_pipe_4_09 <= prod_4_09;
prod_pipe_4_10 <= prod_4_10;
prod_pipe_4_11 <= prod_4_11;
prod_pipe_4_12 <= prod_4_12;
prod_pipe_4_13 <= prod_4_13;
prod_pipe_4_14 <= prod_4_14;
prod_pipe_4_15 <= prod_4_15;
prod_pipe_4_16 <= prod_4_16;
prod_pipe_4_17 <= prod_4_17;
prod_pipe_4_18 <= prod_4_18;
prod_pipe_4_19 <= prod_4_19;
prod_pipe_4_20 <= prod_4_20;
```

```
    END IF;
  END IF;
END PROCESS prod_pipe_process;

add_cast_1_01 <= resize(prod_pipe_1_01, 33);
add_cast_1_02 <= resize(prod_pipe_1_02, 33);
add_cast_1_03 <= resize(prod_pipe_1_03, 33);
add_cast_1_04 <= resize(prod_pipe_1_04, 33);
add_cast_1_05 <= resize(prod_pipe_1_05, 33);
add_cast_1_06 <= resize(prod_pipe_1_06, 33);
add_cast_1_07 <= resize(prod_pipe_1_07, 33);
add_cast_1_08 <= resize(prod_pipe_1_08, 33);
add_cast_1_09 <= resize(prod_pipe_1_09, 33);
add_cast_1_10 <= resize(prod_pipe_1_10, 33);
add_cast_1_11 <= resize(prod_pipe_1_11, 33);
add_cast_1_12 <= resize(prod_pipe_1_12, 33);
add_cast_1_13 <= resize(prod_pipe_1_13, 33);
add_cast_1_14 <= resize(prod_pipe_1_14, 33);
add_cast_1_15 <= resize(prod_pipe_1_15, 33);
add_cast_1_16 <= resize(prod_pipe_1_16, 33);
add_cast_1_17 <= resize(prod_pipe_1_17, 33);
add_cast_1_18 <= resize(prod_pipe_1_18, 33);
add_cast_1_19 <= resize(prod_pipe_1_19, 33);
add_cast_1_20 <= resize(prod_pipe_1_20, 33);
add_cast_1_21 <= resize(prod_pipe_2_01, 33);
add_cast_1_22 <= resize(prod_pipe_2_02, 33);
add_cast_1_23 <= resize(prod_pipe_2_03, 33);
add_cast_1_24 <= resize(prod_pipe_2_04, 33);
add_cast_1_25 <= resize(prod_pipe_2_05, 33);
add_cast_1_26 <= resize(prod_pipe_2_06, 33);
add_cast_1_27 <= resize(prod_pipe_2_07, 33);
add_cast_1_28 <= resize(prod_pipe_2_08, 33);
add_cast_1_29 <= resize(prod_pipe_2_09, 33);
```

```
add_cast_1_30 <= resize(prod_pipe_2_10, 33);
add_cast_1_31 <= resize(prod_pipe_2_11, 33);
add_cast_1_32 <= resize(prod_pipe_2_12, 33);
add_cast_1_33 <= resize(prod_pipe_2_13, 33);
add_cast_1_34 <= resize(prod_pipe_2_14, 33);
add_cast_1_35 <= resize(prod_pipe_2_15, 33);
add_cast_1_36 <= resize(prod_pipe_2_16, 33);
add_cast_1_37 <= resize(prod_pipe_2_17, 33);
add_cast_1_38 <= resize(prod_pipe_2_18, 33);
add_cast_1_39 <= resize(prod_pipe_2_19, 33);
add_cast_1_40 <= resize(prod_pipe_2_20, 33);
add_cast_1_41 <= resize(prod_pipe_3_01, 33);
add_cast_1_42 <= resize(prod_pipe_3_02, 33);
add_cast_1_43 <= resize(prod_pipe_3_03, 33);
add_cast_1_44 <= resize(prod_pipe_3_04, 33);
add_cast_1_45 <= resize(prod_pipe_3_05, 33);
add_cast_1_46 <= resize(prod_pipe_3_06, 33);
add_cast_1_47 <= resize(prod_pipe_3_07, 33);
add_cast_1_48 <= resize(prod_pipe_3_08, 33);
add_cast_1_49 <= resize(prod_pipe_3_09, 33);
add_cast_1_50 <= resize(prod_pipe_3_10, 33);
add_cast_1_51 <= resize(prod_pipe_3_11, 33);
add_cast_1_52 <= resize(prod_pipe_3_12, 33);
add_cast_1_53 <= resize(prod_pipe_3_13, 33);
add_cast_1_54 <= resize(prod_pipe_3_14, 33);
add_cast_1_55 <= resize(prod_pipe_3_15, 33);
add_cast_1_56 <= resize(prod_pipe_3_16, 33);
add_cast_1_57 <= resize(prod_pipe_3_17, 33);
add_cast_1_58 <= resize(prod_pipe_3_18, 33);
add_cast_1_59 <= resize(prod_pipe_3_19, 33);
add_cast_1_60 <= resize(prod_pipe_3_20, 33);
add_cast_1_61 <= resize(prod_pipe_4_01, 33);
add_cast_1_62 <= resize(prod_pipe_4_02, 33);
```

```
add_cast_1_63 <= resize(prod_pipe_4_03, 33);
add_cast_1_64 <= resize(prod_pipe_4_04, 33);
add_cast_1_65 <= resize(prod_pipe_4_05, 33);
add_cast_1_66 <= resize(prod_pipe_4_06, 33);
add_cast_1_67 <= resize(prod_pipe_4_07, 33);
add_cast_1_68 <= resize(prod_pipe_4_08, 33);
add_cast_1_69 <= resize(prod_pipe_4_09, 33);
add_cast_1_70 <= resize(prod_pipe_4_10, 33);
add_cast_1_71 <= resize(prod_pipe_4_11, 33);
add_cast_1_72 <= resize(prod_pipe_4_12, 33);
add_cast_1_73 <= resize(prod_pipe_4_13, 33);
add_cast_1_74 <= resize(prod_pipe_4_14, 33);
add_cast_1_75 <= resize(prod_pipe_4_15, 33);
add_cast_1_76 <= resize(prod_pipe_4_16, 33);
add_cast_1_77 <= resize(prod_pipe_4_17, 33);
add_cast_1_78 <= resize(prod_pipe_4_18, 33);
add_cast_1_79 <= resize(prod_pipe_4_19, 33);
add_cast_1_80 <= resize(prod_pipe_4_20, 33);

add_temp_1_01 <= resize(add_cast_1_01, 34) + resize(add_cast_1_02, 34)
    ;
add_temp_1_02 <= resize(add_cast_1_03, 34) + resize(add_cast_1_04, 34)
    ;
add_temp_1_03 <= resize(add_cast_1_05, 34) + resize(add_cast_1_06, 34)
    ;
add_temp_1_04 <= resize(add_cast_1_07, 34) + resize(add_cast_1_08, 34)
    ;
add_temp_1_05 <= resize(add_cast_1_09, 34) + resize(add_cast_1_10, 34)
    ;
add_temp_1_06 <= resize(add_cast_1_11, 34) + resize(add_cast_1_12, 34)
    ;
add_temp_1_07 <= resize(add_cast_1_13, 34) + resize(add_cast_1_14, 34)
    ;
```

```
add_temp_1_08 <= resize(add_cast_1_15, 34) + resize(add_cast_1_16, 34)
    ;
add_temp_1_09 <= resize(add_cast_1_17, 34) + resize(add_cast_1_18, 34)
    ;
add_temp_1_10 <= resize(add_cast_1_19, 34) + resize(add_cast_1_20, 34)
    ;
add_temp_1_11 <= resize(add_cast_1_21, 34) + resize(add_cast_1_22, 34)
    ;
add_temp_1_12 <= resize(add_cast_1_23, 34) + resize(add_cast_1_24, 34)
    ;
add_temp_1_13 <= resize(add_cast_1_25, 34) + resize(add_cast_1_26, 34)
    ;
add_temp_1_14 <= resize(add_cast_1_27, 34) + resize(add_cast_1_28, 34)
    ;
add_temp_1_15 <= resize(add_cast_1_29, 34) + resize(add_cast_1_30, 34)
    ;
add_temp_1_16 <= resize(add_cast_1_31, 34) + resize(add_cast_1_32, 34)
    ;
add_temp_1_17 <= resize(add_cast_1_33, 34) + resize(add_cast_1_34, 34)
    ;
add_temp_1_18 <= resize(add_cast_1_35, 34) + resize(add_cast_1_36, 34)
    ;
add_temp_1_19 <= resize(add_cast_1_37, 34) + resize(add_cast_1_38, 34)
    ;
add_temp_1_20 <= resize(add_cast_1_39, 34) + resize(add_cast_1_40, 34)
    ;
add_temp_1_21 <= resize(add_cast_1_41, 34) + resize(add_cast_1_42, 34)
    ;
add_temp_1_22 <= resize(add_cast_1_43, 34) + resize(add_cast_1_44, 34)
    ;
add_temp_1_23 <= resize(add_cast_1_45, 34) + resize(add_cast_1_46, 34)
    ;
add_temp_1_24 <= resize(add_cast_1_47, 34) + resize(add_cast_1_48, 34)
```

```
    ;
add_temp_1_25 <= resize(add_cast_1_49, 34) + resize(add_cast_1_50, 34)
    ;
add_temp_1_26 <= resize(add_cast_1_51, 34) + resize(add_cast_1_52, 34)
    ;
add_temp_1_27 <= resize(add_cast_1_53, 34) + resize(add_cast_1_54, 34)
    ;
add_temp_1_28 <= resize(add_cast_1_55, 34) + resize(add_cast_1_56, 34)
    ;
add_temp_1_29 <= resize(add_cast_1_57, 34) + resize(add_cast_1_58, 34)
    ;
add_temp_1_30 <= resize(add_cast_1_59, 34) + resize(add_cast_1_60, 34)
    ;
add_temp_1_31 <= resize(add_cast_1_61, 34) + resize(add_cast_1_62, 34)
    ;
add_temp_1_32 <= resize(add_cast_1_63, 34) + resize(add_cast_1_64, 34)
    ;
add_temp_1_33 <= resize(add_cast_1_65, 34) + resize(add_cast_1_66, 34)
    ;
add_temp_1_34 <= resize(add_cast_1_67, 34) + resize(add_cast_1_68, 34)
    ;
add_temp_1_35 <= resize(add_cast_1_69, 34) + resize(add_cast_1_70, 34)
    ;
add_temp_1_36 <= resize(add_cast_1_71, 34) + resize(add_cast_1_72, 34)
    ;
add_temp_1_37 <= resize(add_cast_1_73, 34) + resize(add_cast_1_74, 34)
    ;
add_temp_1_38 <= resize(add_cast_1_75, 34) + resize(add_cast_1_76, 34)
    ;
add_temp_1_39 <= resize(add_cast_1_77, 34) + resize(add_cast_1_78, 34)
    ;
add_temp_1_40 <= resize(add_cast_1_79, 34) + resize(add_cast_1_80, 34)
    ;
```

```
sumvector1(0) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_01(33) =
    '0' AND add_temp_1_01(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_01(33) = '1' AND
        add_temp_1_01(32) /= '1'
    ELSE (add_temp_1_01(32 DOWNTO 0));
sumvector1(1) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_02(33) =
    '0' AND add_temp_1_02(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_02(33) = '1' AND
        add_temp_1_02(32) /= '1'
    ELSE (add_temp_1_02(32 DOWNTO 0));
sumvector1(2) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_03(33) =
    '0' AND add_temp_1_03(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_03(33) = '1' AND
        add_temp_1_03(32) /= '1'
    ELSE (add_temp_1_03(32 DOWNTO 0));
sumvector1(3) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_04(33) =
    '0' AND add_temp_1_04(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_04(33) = '1' AND
        add_temp_1_04(32) /= '1'
    ELSE (add_temp_1_04(32 DOWNTO 0));
sumvector1(4) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_05(33) =
    '0' AND add_temp_1_05(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_05(33) = '1' AND
        add_temp_1_05(32) /= '1'
    ELSE (add_temp_1_05(32 DOWNTO 0));
sumvector1(5) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_06(33) =
    '0' AND add_temp_1_06(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_06(33) = '1' AND
        add_temp_1_06(32) /= '1'
    ELSE (add_temp_1_06(32 DOWNTO 0));
sumvector1(6) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_07(33) =
    '0' AND add_temp_1_07(32) /= '0'
```

```
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_07(33) = '1' AND
        add_temp_1_07(32) /= '1'
    ELSE (add_temp_1_07(32 DOWNTO 0));
sumvector1(7) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_08(33) =
    '0' AND add_temp_1_08(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_08(33) = '1' AND
        add_temp_1_08(32) /= '1'
    ELSE (add_temp_1_08(32 DOWNTO 0));
sumvector1(8) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_09(33) =
    '0' AND add_temp_1_09(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_09(33) = '1' AND
        add_temp_1_09(32) /= '1'
    ELSE (add_temp_1_09(32 DOWNTO 0));
sumvector1(9) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_10(33) =
    '0' AND add_temp_1_10(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_10(33) = '1' AND
        add_temp_1_10(32) /= '1'
    ELSE (add_temp_1_10(32 DOWNTO 0));
sumvector1(10) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_11(33) =
    '0' AND add_temp_1_11(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_11(33) = '1' AND
        add_temp_1_11(32) /= '1'
    ELSE (add_temp_1_11(32 DOWNTO 0));
sumvector1(11) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_12(33) =
    '0' AND add_temp_1_12(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_12(33) = '1' AND
        add_temp_1_12(32) /= '1'
    ELSE (add_temp_1_12(32 DOWNTO 0));
sumvector1(12) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_13(33) =
    '0' AND add_temp_1_13(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_13(33) = '1' AND
        add_temp_1_13(32) /= '1'
    ELSE (add_temp_1_13(32 DOWNTO 0));
```

```
sumvector1(13) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_14(33) =
    '0' AND add_temp_1_14(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_14(33) = '1' AND
        add_temp_1_14(32) /= '1'
    ELSE (add_temp_1_14(32 DOWNTO 0));
sumvector1(14) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_15(33) =
    '0' AND add_temp_1_15(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_15(33) = '1' AND
        add_temp_1_15(32) /= '1'
    ELSE (add_temp_1_15(32 DOWNTO 0));
sumvector1(15) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_16(33) =
    '0' AND add_temp_1_16(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_16(33) = '1' AND
        add_temp_1_16(32) /= '1'
    ELSE (add_temp_1_16(32 DOWNTO 0));
sumvector1(16) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_17(33) =
    '0' AND add_temp_1_17(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_17(33) = '1' AND
        add_temp_1_17(32) /= '1'
    ELSE (add_temp_1_17(32 DOWNTO 0));
sumvector1(17) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_18(33) =
    '0' AND add_temp_1_18(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_18(33) = '1' AND
        add_temp_1_18(32) /= '1'
    ELSE (add_temp_1_18(32 DOWNTO 0));
sumvector1(18) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_19(33) =
    '0' AND add_temp_1_19(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_19(33) = '1' AND
        add_temp_1_19(32) /= '1'
    ELSE (add_temp_1_19(32 DOWNTO 0));
sumvector1(19) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_20(33) =
    '0' AND add_temp_1_20(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_20(33) = '1' AND
```

```
        add_temp_1_20(32) /= '1'
    ELSE (add_temp_1_20(32 DOWNTO 0));
sumvector1(20) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_21(33) =
    '0' AND add_temp_1_21(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_21(33) = '1' AND
        add_temp_1_21(32) /= '1'
    ELSE (add_temp_1_21(32 DOWNTO 0));
sumvector1(21) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_22(33) =
    '0' AND add_temp_1_22(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_22(33) = '1' AND
        add_temp_1_22(32) /= '1'
    ELSE (add_temp_1_22(32 DOWNTO 0));
sumvector1(22) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_23(33) =
    '0' AND add_temp_1_23(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_23(33) = '1' AND
        add_temp_1_23(32) /= '1'
    ELSE (add_temp_1_23(32 DOWNTO 0));
sumvector1(23) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_24(33) =
    '0' AND add_temp_1_24(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_24(33) = '1' AND
        add_temp_1_24(32) /= '1'
    ELSE (add_temp_1_24(32 DOWNTO 0));
sumvector1(24) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_25(33) =
    '0' AND add_temp_1_25(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_25(33) = '1' AND
        add_temp_1_25(32) /= '1'
    ELSE (add_temp_1_25(32 DOWNTO 0));
sumvector1(25) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_26(33) =
    '0' AND add_temp_1_26(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_26(33) = '1' AND
        add_temp_1_26(32) /= '1'
    ELSE (add_temp_1_26(32 DOWNTO 0));
sumvector1(26) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_27(33) =
```

```
    '0' AND add_temp_1_27(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_27(33) = '1' AND
        add_temp_1_27(32) /= '1'
    ELSE (add_temp_1_27(32 DOWNTO 0));
sumvector1(27) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_28(33) =
    '0' AND add_temp_1_28(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_28(33) = '1' AND
        add_temp_1_28(32) /= '1'
    ELSE (add_temp_1_28(32 DOWNTO 0));
sumvector1(28) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_29(33) =
    '0' AND add_temp_1_29(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_29(33) = '1' AND
        add_temp_1_29(32) /= '1'
    ELSE (add_temp_1_29(32 DOWNTO 0));
sumvector1(29) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_30(33) =
    '0' AND add_temp_1_30(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_30(33) = '1' AND
        add_temp_1_30(32) /= '1'
    ELSE (add_temp_1_30(32 DOWNTO 0));
sumvector1(30) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_31(33) =
    '0' AND add_temp_1_31(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_31(33) = '1' AND
        add_temp_1_31(32) /= '1'
    ELSE (add_temp_1_31(32 DOWNTO 0));
sumvector1(31) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_32(33) =
    '0' AND add_temp_1_32(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_32(33) = '1' AND
        add_temp_1_32(32) /= '1'
    ELSE (add_temp_1_32(32 DOWNTO 0));
sumvector1(32) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_33(33) =
    '0' AND add_temp_1_33(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_33(33) = '1' AND
        add_temp_1_33(32) /= '1'
```

```
    ELSE (add_temp_1_33(32 DOWNTO 0));
sumvector1(33) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_34(33) =
    '0' AND add_temp_1_34(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_34(33) = '1' AND
        add_temp_1_34(32) /= '1'
    ELSE (add_temp_1_34(32 DOWNTO 0));
sumvector1(34) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_35(33) =
    '0' AND add_temp_1_35(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_35(33) = '1' AND
        add_temp_1_35(32) /= '1'
    ELSE (add_temp_1_35(32 DOWNTO 0));
sumvector1(35) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_36(33) =
    '0' AND add_temp_1_36(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_36(33) = '1' AND
        add_temp_1_36(32) /= '1'
    ELSE (add_temp_1_36(32 DOWNTO 0));
sumvector1(36) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_37(33) =
    '0' AND add_temp_1_37(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_37(33) = '1' AND
        add_temp_1_37(32) /= '1'
    ELSE (add_temp_1_37(32 DOWNTO 0));
sumvector1(37) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_38(33) =
    '0' AND add_temp_1_38(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_38(33) = '1' AND
        add_temp_1_38(32) /= '1'
    ELSE (add_temp_1_38(32 DOWNTO 0));
sumvector1(38) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_39(33) =
    '0' AND add_temp_1_39(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_39(33) = '1' AND
        add_temp_1_39(32) /= '1'
    ELSE (add_temp_1_39(32 DOWNTO 0));
sumvector1(39) <= (32 => '0', OTHERS => '1') WHEN add_temp_1_40(33) =
    '0' AND add_temp_1_40(32) /= '0'
```

```
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_1_40(33) = '1' AND
        add_temp_1_40(32) /= '1'
    ELSE (add_temp_1_40(32 DOWNTO 0));
sumvector1(40) <= resize(prod_pipe_1_21, 33);

sumdelay_pipeline_process1 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    sumdelay_pipeline1 <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF phase_0 = '1' THEN
      sumdelay_pipeline1(0 TO 40) <= sumvector1(0 TO 40);
    END IF;
  END IF;
END PROCESS sumdelay_pipeline_process1;

add_cast_2_01 <= sumdelay_pipeline1(0);
add_cast_2_02 <= sumdelay_pipeline1(1);
add_cast_2_03 <= sumdelay_pipeline1(2);
add_cast_2_04 <= sumdelay_pipeline1(3);
add_cast_2_05 <= sumdelay_pipeline1(4);
add_cast_2_06 <= sumdelay_pipeline1(5);
add_cast_2_07 <= sumdelay_pipeline1(6);
add_cast_2_08 <= sumdelay_pipeline1(7);
add_cast_2_09 <= sumdelay_pipeline1(8);
add_cast_2_10 <= sumdelay_pipeline1(9);
add_cast_2_11 <= sumdelay_pipeline1(10);
add_cast_2_12 <= sumdelay_pipeline1(11);
add_cast_2_13 <= sumdelay_pipeline1(12);
add_cast_2_14 <= sumdelay_pipeline1(13);
add_cast_2_15 <= sumdelay_pipeline1(14);
add_cast_2_16 <= sumdelay_pipeline1(15);
add_cast_2_17 <= sumdelay_pipeline1(16);
```

```
add_cast_2_18 <= sumdelay_pipeline1(17);
add_cast_2_19 <= sumdelay_pipeline1(18);
add_cast_2_20 <= sumdelay_pipeline1(19);
add_cast_2_21 <= sumdelay_pipeline1(20);
add_cast_2_22 <= sumdelay_pipeline1(21);
add_cast_2_23 <= sumdelay_pipeline1(22);
add_cast_2_24 <= sumdelay_pipeline1(23);
add_cast_2_25 <= sumdelay_pipeline1(24);
add_cast_2_26 <= sumdelay_pipeline1(25);
add_cast_2_27 <= sumdelay_pipeline1(26);
add_cast_2_28 <= sumdelay_pipeline1(27);
add_cast_2_29 <= sumdelay_pipeline1(28);
add_cast_2_30 <= sumdelay_pipeline1(29);
add_cast_2_31 <= sumdelay_pipeline1(30);
add_cast_2_32 <= sumdelay_pipeline1(31);
add_cast_2_33 <= sumdelay_pipeline1(32);
add_cast_2_34 <= sumdelay_pipeline1(33);
add_cast_2_35 <= sumdelay_pipeline1(34);
add_cast_2_36 <= sumdelay_pipeline1(35);
add_cast_2_37 <= sumdelay_pipeline1(36);
add_cast_2_38 <= sumdelay_pipeline1(37);
add_cast_2_39 <= sumdelay_pipeline1(38);
add_cast_2_40 <= sumdelay_pipeline1(39);

add_temp_2_01 <= resize(add_cast_2_01, 34) + resize(add_cast_2_02, 34)
    ;
add_temp_2_02 <= resize(add_cast_2_03, 34) + resize(add_cast_2_04, 34)
    ;
add_temp_2_03 <= resize(add_cast_2_05, 34) + resize(add_cast_2_06, 34)
    ;
add_temp_2_04 <= resize(add_cast_2_07, 34) + resize(add_cast_2_08, 34)
    ;
add_temp_2_05 <= resize(add_cast_2_09, 34) + resize(add_cast_2_10, 34)
```

```
    ;
add_temp_2_06 <= resize(add_cast_2_11, 34) + resize(add_cast_2_12, 34)
    ;
add_temp_2_07 <= resize(add_cast_2_13, 34) + resize(add_cast_2_14, 34)
    ;
add_temp_2_08 <= resize(add_cast_2_15, 34) + resize(add_cast_2_16, 34)
    ;
add_temp_2_09 <= resize(add_cast_2_17, 34) + resize(add_cast_2_18, 34)
    ;
add_temp_2_10 <= resize(add_cast_2_19, 34) + resize(add_cast_2_20, 34)
    ;
add_temp_2_11 <= resize(add_cast_2_21, 34) + resize(add_cast_2_22, 34)
    ;
add_temp_2_12 <= resize(add_cast_2_23, 34) + resize(add_cast_2_24, 34)
    ;
add_temp_2_13 <= resize(add_cast_2_25, 34) + resize(add_cast_2_26, 34)
    ;
add_temp_2_14 <= resize(add_cast_2_27, 34) + resize(add_cast_2_28, 34)
    ;
add_temp_2_15 <= resize(add_cast_2_29, 34) + resize(add_cast_2_30, 34)
    ;
add_temp_2_16 <= resize(add_cast_2_31, 34) + resize(add_cast_2_32, 34)
    ;
add_temp_2_17 <= resize(add_cast_2_33, 34) + resize(add_cast_2_34, 34)
    ;
add_temp_2_18 <= resize(add_cast_2_35, 34) + resize(add_cast_2_36, 34)
    ;
add_temp_2_19 <= resize(add_cast_2_37, 34) + resize(add_cast_2_38, 34)
    ;
add_temp_2_20 <= resize(add_cast_2_39, 34) + resize(add_cast_2_40, 34)
    ;

sumvector2(0) <= (32 => '0', OTHERS => '1') WHEN add_temp_2_01(33) =
```

```
    '0' AND add_temp_2_01(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_2_01(33) = '1' AND
        add_temp_2_01(32) /= '1'
    ELSE (add_temp_2_01(32 DOWNTO 0));
sumvector2(1) <= (32 => '0', OTHERS => '1') WHEN add_temp_2_02(33) =
    '0' AND add_temp_2_02(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_2_02(33) = '1' AND
        add_temp_2_02(32) /= '1'
    ELSE (add_temp_2_02(32 DOWNTO 0));
sumvector2(2) <= (32 => '0', OTHERS => '1') WHEN add_temp_2_03(33) =
    '0' AND add_temp_2_03(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_2_03(33) = '1' AND
        add_temp_2_03(32) /= '1'
    ELSE (add_temp_2_03(32 DOWNTO 0));
sumvector2(3) <= (32 => '0', OTHERS => '1') WHEN add_temp_2_04(33) =
    '0' AND add_temp_2_04(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_2_04(33) = '1' AND
        add_temp_2_04(32) /= '1'
    ELSE (add_temp_2_04(32 DOWNTO 0));
sumvector2(4) <= (32 => '0', OTHERS => '1') WHEN add_temp_2_05(33) =
    '0' AND add_temp_2_05(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_2_05(33) = '1' AND
        add_temp_2_05(32) /= '1'
    ELSE (add_temp_2_05(32 DOWNTO 0));
sumvector2(5) <= (32 => '0', OTHERS => '1') WHEN add_temp_2_06(33) =
    '0' AND add_temp_2_06(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_2_06(33) = '1' AND
        add_temp_2_06(32) /= '1'
    ELSE (add_temp_2_06(32 DOWNTO 0));
sumvector2(6) <= (32 => '0', OTHERS => '1') WHEN add_temp_2_07(33) =
    '0' AND add_temp_2_07(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_2_07(33) = '1' AND
        add_temp_2_07(32) /= '1'
```

```
    ELSE (add_temp_2_07(32 DOWNTO 0));
sumvector2(7) <= (32 => '0', OTHERS => '1') WHEN add_temp_2_08(33) =
    '0' AND add_temp_2_08(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_2_08(33) = '1' AND
        add_temp_2_08(32) /= '1'
    ELSE (add_temp_2_08(32 DOWNTO 0));
sumvector2(8) <= (32 => '0', OTHERS => '1') WHEN add_temp_2_09(33) =
    '0' AND add_temp_2_09(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_2_09(33) = '1' AND
        add_temp_2_09(32) /= '1'
    ELSE (add_temp_2_09(32 DOWNTO 0));
sumvector2(9) <= (32 => '0', OTHERS => '1') WHEN add_temp_2_10(33) =
    '0' AND add_temp_2_10(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_2_10(33) = '1' AND
        add_temp_2_10(32) /= '1'
    ELSE (add_temp_2_10(32 DOWNTO 0));
sumvector2(10) <= (32 => '0', OTHERS => '1') WHEN add_temp_2_11(33) =
    '0' AND add_temp_2_11(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_2_11(33) = '1' AND
        add_temp_2_11(32) /= '1'
    ELSE (add_temp_2_11(32 DOWNTO 0));
sumvector2(11) <= (32 => '0', OTHERS => '1') WHEN add_temp_2_12(33) =
    '0' AND add_temp_2_12(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_2_12(33) = '1' AND
        add_temp_2_12(32) /= '1'
    ELSE (add_temp_2_12(32 DOWNTO 0));
sumvector2(12) <= (32 => '0', OTHERS => '1') WHEN add_temp_2_13(33) =
    '0' AND add_temp_2_13(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_2_13(33) = '1' AND
        add_temp_2_13(32) /= '1'
    ELSE (add_temp_2_13(32 DOWNTO 0));
sumvector2(13) <= (32 => '0', OTHERS => '1') WHEN add_temp_2_14(33) =
    '0' AND add_temp_2_14(32) /= '0'
```

```
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_2_14(33) = '1' AND
        add_temp_2_14(32) /= '1'
    ELSE (add_temp_2_14(32 DOWNTO 0));
sumvector2(14) <= (32 => '0', OTHERS => '1') WHEN add_temp_2_15(33) =
    '0' AND add_temp_2_15(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_2_15(33) = '1' AND
        add_temp_2_15(32) /= '1'
    ELSE (add_temp_2_15(32 DOWNTO 0));
sumvector2(15) <= (32 => '0', OTHERS => '1') WHEN add_temp_2_16(33) =
    '0' AND add_temp_2_16(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_2_16(33) = '1' AND
        add_temp_2_16(32) /= '1'
    ELSE (add_temp_2_16(32 DOWNTO 0));
sumvector2(16) <= (32 => '0', OTHERS => '1') WHEN add_temp_2_17(33) =
    '0' AND add_temp_2_17(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_2_17(33) = '1' AND
        add_temp_2_17(32) /= '1'
    ELSE (add_temp_2_17(32 DOWNTO 0));
sumvector2(17) <= (32 => '0', OTHERS => '1') WHEN add_temp_2_18(33) =
    '0' AND add_temp_2_18(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_2_18(33) = '1' AND
        add_temp_2_18(32) /= '1'
    ELSE (add_temp_2_18(32 DOWNTO 0));
sumvector2(18) <= (32 => '0', OTHERS => '1') WHEN add_temp_2_19(33) =
    '0' AND add_temp_2_19(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_2_19(33) = '1' AND
        add_temp_2_19(32) /= '1'
    ELSE (add_temp_2_19(32 DOWNTO 0));
sumvector2(19) <= (32 => '0', OTHERS => '1') WHEN add_temp_2_20(33) =
    '0' AND add_temp_2_20(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_2_20(33) = '1' AND
        add_temp_2_20(32) /= '1'
    ELSE (add_temp_2_20(32 DOWNTO 0));
```

```
sumvector2(20) <= sumdelay_pipeline1(40);

sumdelay_pipeline_process2 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    sumdelay_pipeline2 <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF phase_0 = '1' THEN
      sumdelay_pipeline2(0 TO 20) <= sumvector2(0 TO 20);
    END IF;
  END IF;
END PROCESS sumdelay_pipeline_process2;

add_cast_3_01 <= sumdelay_pipeline2(0);
add_cast_3_02 <= sumdelay_pipeline2(1);
add_cast_3_03 <= sumdelay_pipeline2(2);
add_cast_3_04 <= sumdelay_pipeline2(3);
add_cast_3_05 <= sumdelay_pipeline2(4);
add_cast_3_06 <= sumdelay_pipeline2(5);
add_cast_3_07 <= sumdelay_pipeline2(6);
add_cast_3_08 <= sumdelay_pipeline2(7);
add_cast_3_09 <= sumdelay_pipeline2(8);
add_cast_3_10 <= sumdelay_pipeline2(9);
add_cast_3_11 <= sumdelay_pipeline2(10);
add_cast_3_12 <= sumdelay_pipeline2(11);
add_cast_3_13 <= sumdelay_pipeline2(12);
add_cast_3_14 <= sumdelay_pipeline2(13);
add_cast_3_15 <= sumdelay_pipeline2(14);
add_cast_3_16 <= sumdelay_pipeline2(15);
add_cast_3_17 <= sumdelay_pipeline2(16);
add_cast_3_18 <= sumdelay_pipeline2(17);
add_cast_3_19 <= sumdelay_pipeline2(18);
add_cast_3_20 <= sumdelay_pipeline2(19);
```

```
add_temp_3_01 <= resize(add_cast_3_01, 34) + resize(add_cast_3_02, 34)
    ;
add_temp_3_02 <= resize(add_cast_3_03, 34) + resize(add_cast_3_04, 34)
    ;
add_temp_3_03 <= resize(add_cast_3_05, 34) + resize(add_cast_3_06, 34)
    ;
add_temp_3_04 <= resize(add_cast_3_07, 34) + resize(add_cast_3_08, 34)
    ;
add_temp_3_05 <= resize(add_cast_3_09, 34) + resize(add_cast_3_10, 34)
    ;
add_temp_3_06 <= resize(add_cast_3_11, 34) + resize(add_cast_3_12, 34)
    ;
add_temp_3_07 <= resize(add_cast_3_13, 34) + resize(add_cast_3_14, 34)
    ;
add_temp_3_08 <= resize(add_cast_3_15, 34) + resize(add_cast_3_16, 34)
    ;
add_temp_3_09 <= resize(add_cast_3_17, 34) + resize(add_cast_3_18, 34)
    ;
add_temp_3_10 <= resize(add_cast_3_19, 34) + resize(add_cast_3_20, 34)
    ;

sumvector3(0) <= (32 => '0', OTHERS => '1') WHEN add_temp_3_01(33) =
    '0' AND add_temp_3_01(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_3_01(33) = '1' AND
        add_temp_3_01(32) /= '1'
    ELSE (add_temp_3_01(32 DOWNTO 0));
sumvector3(1) <= (32 => '0', OTHERS => '1') WHEN add_temp_3_02(33) =
    '0' AND add_temp_3_02(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_3_02(33) = '1' AND
        add_temp_3_02(32) /= '1'
    ELSE (add_temp_3_02(32 DOWNTO 0));
sumvector3(2) <= (32 => '0', OTHERS => '1') WHEN add_temp_3_03(33) =
```

```
    '0' AND add_temp_3_03(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_3_03(33) = '1' AND
        add_temp_3_03(32) /= '1'
    ELSE (add_temp_3_03(32 DOWNTO 0));
sumvector3(3) <= (32 => '0', OTHERS => '1') WHEN add_temp_3_04(33) =
    '0' AND add_temp_3_04(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_3_04(33) = '1' AND
        add_temp_3_04(32) /= '1'
    ELSE (add_temp_3_04(32 DOWNTO 0));
sumvector3(4) <= (32 => '0', OTHERS => '1') WHEN add_temp_3_05(33) =
    '0' AND add_temp_3_05(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_3_05(33) = '1' AND
        add_temp_3_05(32) /= '1'
    ELSE (add_temp_3_05(32 DOWNTO 0));
sumvector3(5) <= (32 => '0', OTHERS => '1') WHEN add_temp_3_06(33) =
    '0' AND add_temp_3_06(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_3_06(33) = '1' AND
        add_temp_3_06(32) /= '1'
    ELSE (add_temp_3_06(32 DOWNTO 0));
sumvector3(6) <= (32 => '0', OTHERS => '1') WHEN add_temp_3_07(33) =
    '0' AND add_temp_3_07(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_3_07(33) = '1' AND
        add_temp_3_07(32) /= '1'
    ELSE (add_temp_3_07(32 DOWNTO 0));
sumvector3(7) <= (32 => '0', OTHERS => '1') WHEN add_temp_3_08(33) =
    '0' AND add_temp_3_08(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_3_08(33) = '1' AND
        add_temp_3_08(32) /= '1'
    ELSE (add_temp_3_08(32 DOWNTO 0));
sumvector3(8) <= (32 => '0', OTHERS => '1') WHEN add_temp_3_09(33) =
    '0' AND add_temp_3_09(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_3_09(33) = '1' AND
        add_temp_3_09(32) /= '1'
```

```
    ELSE (add_temp_3_09(32 DOWNTO 0));
sumvector3(9) <= (32 => '0', OTHERS => '1') WHEN add_temp_3_10(33) =
    '0' AND add_temp_3_10(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_3_10(33) = '1' AND
        add_temp_3_10(32) /= '1'
    ELSE (add_temp_3_10(32 DOWNTO 0));
sumvector3(10) <= sumdelay_pipeline2(20);

sumdelay_pipeline_process3 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    sumdelay_pipeline3 <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF phase_0 = '1' THEN
      sumdelay_pipeline3(0 TO 10) <= sumvector3(0 TO 10);
    END IF;
  END IF;
END PROCESS sumdelay_pipeline_process3;

add_cast_4_01 <= sumdelay_pipeline3(0);
add_cast_4_02 <= sumdelay_pipeline3(1);
add_cast_4_03 <= sumdelay_pipeline3(2);
add_cast_4_04 <= sumdelay_pipeline3(3);
add_cast_4_05 <= sumdelay_pipeline3(4);
add_cast_4_06 <= sumdelay_pipeline3(5);
add_cast_4_07 <= sumdelay_pipeline3(6);
add_cast_4_08 <= sumdelay_pipeline3(7);
add_cast_4_09 <= sumdelay_pipeline3(8);
add_cast_4_10 <= sumdelay_pipeline3(9);

add_temp_4_01 <= resize(add_cast_4_01, 34) + resize(add_cast_4_02, 34)
    ;
add_temp_4_02 <= resize(add_cast_4_03, 34) + resize(add_cast_4_04, 34)
```

```
    ;
add_temp_4_03 <= resize(add_cast_4_05, 34) + resize(add_cast_4_06, 34)
    ;
add_temp_4_04 <= resize(add_cast_4_07, 34) + resize(add_cast_4_08, 34)
    ;
add_temp_4_05 <= resize(add_cast_4_09, 34) + resize(add_cast_4_10, 34)
    ;

sumvector4(0) <= (32 => '0', OTHERS => '1') WHEN add_temp_4_01(33) =
    '0' AND add_temp_4_01(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_4_01(33) = '1' AND
        add_temp_4_01(32) /= '1'
    ELSE (add_temp_4_01(32 DOWNTO 0));
sumvector4(1) <= (32 => '0', OTHERS => '1') WHEN add_temp_4_02(33) =
    '0' AND add_temp_4_02(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_4_02(33) = '1' AND
        add_temp_4_02(32) /= '1'
    ELSE (add_temp_4_02(32 DOWNTO 0));
sumvector4(2) <= (32 => '0', OTHERS => '1') WHEN add_temp_4_03(33) =
    '0' AND add_temp_4_03(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_4_03(33) = '1' AND
        add_temp_4_03(32) /= '1'
    ELSE (add_temp_4_03(32 DOWNTO 0));
sumvector4(3) <= (32 => '0', OTHERS => '1') WHEN add_temp_4_04(33) =
    '0' AND add_temp_4_04(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_4_04(33) = '1' AND
        add_temp_4_04(32) /= '1'
    ELSE (add_temp_4_04(32 DOWNTO 0));
sumvector4(4) <= (32 => '0', OTHERS => '1') WHEN add_temp_4_05(33) =
    '0' AND add_temp_4_05(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_4_05(33) = '1' AND
        add_temp_4_05(32) /= '1'
    ELSE (add_temp_4_05(32 DOWNTO 0));
```

```
sumvector4(5) <= sumdelay_pipeline3(10);

sumdelay_pipeline_process4 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    sumdelay_pipeline4 <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF phase_0 = '1' THEN
      sumdelay_pipeline4(0 TO 5) <= sumvector4(0 TO 5);
    END IF;
  END IF;
END PROCESS sumdelay_pipeline_process4;

add_cast_5_01 <= sumdelay_pipeline4(0);
add_cast_5_02 <= sumdelay_pipeline4(1);
add_cast_5_03 <= sumdelay_pipeline4(2);
add_cast_5_04 <= sumdelay_pipeline4(3);
add_cast_5_05 <= sumdelay_pipeline4(4);
add_cast_5_06 <= sumdelay_pipeline4(5);

add_temp_5_01 <= resize(add_cast_5_01, 34) + resize(add_cast_5_02, 34)
    ;
add_temp_5_02 <= resize(add_cast_5_03, 34) + resize(add_cast_5_04, 34)
    ;
add_temp_5_03 <= resize(add_cast_5_05, 34) + resize(add_cast_5_06, 34)
    ;

sumvector5(0) <= (32 => '0', OTHERS => '1') WHEN add_temp_5_01(33) =
    '0' AND add_temp_5_01(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_5_01(33) = '1' AND
        add_temp_5_01(32) /= '1'
    ELSE (add_temp_5_01(32 DOWNTO 0));
sumvector5(1) <= (32 => '0', OTHERS => '1') WHEN add_temp_5_02(33) =
```

```
    '0' AND add_temp_5_02(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_5_02(33) = '1' AND
        add_temp_5_02(32) /= '1'
    ELSE (add_temp_5_02(32 DOWNTO 0));
sumvector5(2) <= (32 => '0', OTHERS => '1') WHEN add_temp_5_03(33) =
    '0' AND add_temp_5_03(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_5_03(33) = '1' AND
        add_temp_5_03(32) /= '1'
    ELSE (add_temp_5_03(32 DOWNTO 0));

sumdelay_pipeline_process5 : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    sumdelay_pipeline5 <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF phase_0 = '1' THEN
      sumdelay_pipeline5(0 TO 2) <= sumvector5(0 TO 2);
    END IF;
  END IF;
END PROCESS sumdelay_pipeline_process5;

add_cast_6_01 <= sumdelay_pipeline5(0);
add_cast_6_02 <= sumdelay_pipeline5(1);

add_temp_6_01 <= resize(add_cast_6_01, 34) + resize(add_cast_6_02, 34)
    ;

sumvector6(0) <= (32 => '0', OTHERS => '1') WHEN add_temp_6_01(33) =
    '0' AND add_temp_6_01(32) /= '0'
    ELSE (32 => '1', OTHERS => '0') WHEN add_temp_6_01(33) = '1' AND
        add_temp_6_01(32) /= '1'
    ELSE (add_temp_6_01(32 DOWNTO 0));
sumvector6(1) <= sumdelay_pipeline5(2);
```

```
    sumdelay_pipeline_process6 : PROCESS (clk, reset)
    BEGIN
      IF reset = '1' THEN
        sumdelay_pipeline6 <= (OTHERS => (OTHERS => '0'));
      ELSIF clk'event AND clk = '1' THEN
        IF phase_0 = '1' THEN
          sumdelay_pipeline6(0 TO 1) <= sumvector6(0 TO 1);
        END IF;
      END IF;
    END PROCESS sumdelay_pipeline_process6;

    add_cast_7_01 <= sumdelay_pipeline6(0);
    add_cast_7_02 <= sumdelay_pipeline6(1);

    add_temp_7_01 <= resize(add_cast_7_01, 34) + resize(add_cast_7_02, 34)
        ;

    sum7 <= std_logic_vector(add_temp_7_01);

    -- Assignment Statements
    FIR_Decimation_out <= sum7(33);
END rtl;
```

A.2.7 BPSK Demodulator

```
-- -------------------------------------------------------------
--
-- File Name: BPSK_Demodulator_Baseband.vhd
-- Created: 2014-03-09 23:35:08
--
-- Generated by MATLAB 8.2 and HDL Coder 3.3
--
-- -------------------------------------------------------------



-- -------------------------------------------------------------
--
-- Module: BPSK_Demodulator_Baseband
-- Source Path: CommSystem/BPSK Demodulator Baseband
-- Hierarchy Level: 1
--
-- -------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY BPSK_Demodulator_Baseband IS
  PORT( sym                               :   IN    std_logic;  -- sign
      bit
        bin                               :   OUT   std_logic  -- ufix1
        );
END BPSK_Demodulator_Baseband;


ARCHITECTURE rtl OF BPSK_Demodulator_Baseband IS
```

```
BEGIN

        --          0   -->   1 --> 0
        --                    1   --> -1 --> 1

        bin <= '0' WHEN sym = '0' ELSE '1';

END rtl;
```

A.3 Hardware Verification

A.3.1 Error Counter

```
--
    ----------------------------------------------------------------------------------


-- Company:
-- Engineer: Matthew Allen
--
-- Create Date:     18:46:20 03/26/2014
-- Design Name:
-- Module Name:     ErrorCounter - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
--
    ----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ErrorCounter is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
```

```
        txSig : in  STD_LOGIC;
        rxSig : in  STD_LOGIC;
        ledOut : out  STD_LOGIC_VECTOR (7 downto 0));
end ErrorCounter;

architecture Behavioral of ErrorCounter is

    -- User Input Constant
    constant delay : integer := 128; -- delay between txSig and
        rxSig

    signal txSig_reg : STD_LOGIC_VECTOR(127 downto 0);
        -- INSERT (DELAY - 1)
    signal txSig_delay : STD_LOGIC;
    signal count : unsigned(7 downto 0);
    signal en_count : STD_LOGIC;

begin

    -- Delay register for txSig
    txSig_reg_process : PROCESS (clk, reset)
    BEGIN
        IF reset = '1' THEN
            txSig_reg <= (others => '0');
        ELSIF clk'EVENT AND clk = '1' THEN
            txSig_reg <= (txSig_reg(126 downto 0) & txSig);
                -- INSERT (DELAY - 2)
        END IF;
    END PROCESS txSig_reg_process;

    -- Delayed txSig
    txSig_delay <= txSig_reg(127);
                                    -- INSERT (DELAY - 1)
```

```
        -- Error counter enable
        en_count <= '1' WHEN txSig_delay /= rxSig ELSE '0';

        -- Error counter
        counter_process : PROCESS (clk, reset)
        BEGIN
                IF reset = '1' THEN
                        count <= (others => '0');
                ELSIF clk'EVENT AND clk = '1' THEN
                        IF en_count = '1' THEN
                                IF count = to_unsigned(255,8) THEN
                                        count <= to_unsigned(0,8);
                                ELSE
                                        count <= count + to_unsigned
                                            (1,8);
                                END IF;
                        END IF;
                END IF;
        END PROCESS counter_process;

        -- Output count to LEDs
        ledOut <= std_logic_vector(count);

end Behavioral;
```

A.3.2 BPSK Model-Driven Design Integration Module

```
--
    ----------------------------------------------------------------------------------


-- Company:
-- Engineer:
--
-- Create Date:     17:24:40 03/22/2014
-- Design Name:
-- Module Name:     CommSystemTop - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
--
    ----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity CommSystemTop is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           resetEC : in  STD_LOGIC;
           LED : out  STD_LOGIC_VECTOR (7 downto 0));
end CommSystemTop;
```

```
architecture Behavioral of CommSystemTop is

    COMPONENT CommSystem IS
        PORT( clk                                   :   IN
            std_logic;
                  reset                             :   IN
                      std_logic;
                  clk_enable                        :   IN
                      std_logic;
                  ce_out                            :
                    OUT   std_logic;
                  PNSeq

                        :   OUT   std_logic;  --
                    ufix1
                  To_DataSink                       :
                    OUT   std_logic  -- ufix1
    );
    END COMPONENT;

    COMPONENT ErrorCounter is
  Port ( clk : in  STD_LOGIC;
         reset : in  STD_LOGIC;
         txSig : in  STD_LOGIC;
         rxSig : in  STD_LOGIC;
         ledOut : out  STD_LOGIC_VECTOR (7 downto 0));
    end COMPONENT;

    signal ce_out : std_logic;
    signal txSig : std_logic;
    signal rxSig : std_logic;
    signal clk_enable : std_logic;
```

```
begin

        clk_enable <= '1';

        CommSystem1 : CommSystem
    PORT MAP( clk => clk,
              reset => reset,
              clk_enable => clk_enable,
              ce_out => ce_out,
              PNSeq => txSig,
              To_DataSink => rxSig
              );

        ErrorCounter1 : ErrorCounter
    PORT MAP( clk => clk,
              reset => resetEC,
              txSig => txSig,
              rxSig => rxSig,
              ledOut => LED
              );


end Behavioral;
```

A.3.3 BPSK Hand-Optimized Design Integration Module

```
--
    ----------------------------------------------------------------------------------


-- Company:
-- Engineer:
--
-- Create Date:     17:24:40 03/22/2014
-- Design Name:
-- Module Name:     CommSystemTop - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
--
    ----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity CommSystemTop is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           resetEC : in  STD_LOGIC;
           LED : out  STD_LOGIC_VECTOR (7 downto 0));
end CommSystemTop;
```

```
architecture Behavioral of CommSystemTop is

    COMPONENT CommSystem IS
        PORT( clk                                   :   IN
            std_logic;
                  reset                             :   IN
                      std_logic;
                  clk_enable                        :   IN
                      std_logic;
                  ce_out                            :
                    OUT   std_logic;
                  PNSeq

                        :   OUT   std_logic;  --
                    ufix1
                  To_DataSink                       :
                    OUT   std_logic  -- ufix1
    );
    END COMPONENT;

    COMPONENT ErrorCounter is
  Port ( clk : in  STD_LOGIC;
         reset : in  STD_LOGIC;
         txSig : in  STD_LOGIC;
         rxSig : in  STD_LOGIC;
         ledOut : out  STD_LOGIC_VECTOR (7 downto 0));
    end COMPONENT;

    signal ce_out : std_logic;
    signal txSig : std_logic;
    signal rxSig : std_logic;
    signal clk_enable : std_logic;
```

```
begin

        clk_enable <= '1';

        CommSystem1 : CommSystem
    PORT MAP( clk => clk,
              reset => reset,
              clk_enable => clk_enable,
              ce_out => ce_out,
              PNSeq => txSig,
              To_DataSink => rxSig
              );

        ErrorCounter1 : ErrorCounter
    PORT MAP( clk => clk,
              reset => resetEC,
              txSig => txSig,
              rxSig => rxSig,
              ledOut => LED
              );


end Behavioral;
```

Appendix B

Testbenches

B.1 Hardware Verification

B.1.1 Error Counter Testbench

```
--
--------------------------------------------------------------------------------

-- Company:
-- Engineer:
--
-- Create Date:   17:32:55 03/22/2014
-- Design Name:
-- Module Name:   EC_TB.vhd
-- Project Name:  ModelDriven
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: ErrorCounter
--
-- Dependencies:
--
```

```
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic
    and
-- std_logic_vector for the ports of the unit under test.  Xilinx
    recommends
-- that these types always be used for the top-level I/O of a design in
    order
-- to guarantee that the testbench will bind correctly to the post-
    implementation
-- simulation model.
--
    ------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.conv_std_logic_vector;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY EC_TB IS
END EC_TB;

ARCHITECTURE behavior OF EC_TB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT ErrorCounter
```

```
 PORT(
        clk : IN  std_logic;
        reset : IN  std_logic;
        txSig : IN  std_logic;
        rxSig : IN  std_logic;
        ledOut : OUT  std_logic_vector(7 downto 0)
       );
 END COMPONENT;


    --Inputs
    signal clk : std_logic := '0';
    signal reset : std_logic := '0';
    signal txSig : std_logic := '0';
    signal rxSig : std_logic := '0';

        --Outputs
    signal ledOut : std_logic_vector(7 downto 0);

    -- Clock period definitions
    constant clk_period : time := 10 ns;

        TYPE tx_type IS ARRAY (0 to 5) of std_logic;
        TYPE rx_type IS ARRAY (0 to 7) of std_logic;

        CONSTANT txInput : tx_type := ('0','1','1','0','0','1');
        CONSTANT rxInput : rx_type := ('0','0','0','0','1','1','0','0');


BEGIN

        -- Instantiate the Unit Under Test (UUT)
    uut: ErrorCounter PORT MAP (
```

```
        clk => clk,
        reset => reset,
        txSig => txSig,
        rxSig => rxSig,
        ledOut => ledOut
      );

-- Clock process definitions
clk_process :process
begin
              clk <= '0';
              wait for clk_period/2;
              clk <= '1';
              wait for clk_period/2;
end process;


-- Stimulus process
stim_proc: process
begin
   -- hold reset state for 100 ns.
   wait for 100 ns;

   wait for clk_period*10;

   reset <= '1';

              wait for clk_period;

              reset <= '0';

              wait for 140 ns;
```

```
            reset <= '1';

            wait for clk_period;

            reset <= '0';

    wait;
end process;

    tx_process: process
    begin

            wait until reset = '1';
            wait until reset = '0';
            wait for clk_period*0.25;

            FOR j IN txInput'RANGE LOOP

                    txSig <= txInput(j);

                    wait for clk_period;

            END LOOP;

            wait;
    end process;

    rx_process: process
    begin

            wait until reset = '1';
            wait until reset = '0';
            wait for clk_period*0.25;
```

```
                FOR k IN rxInput'RANGE LOOP

                        rxSig <= rxInput(k);

                        wait for clk_period;

                END LOOP;

                wait;
        end process;

END;
```

B.2 BPSK Model-Driven Design

B.2.1 Top Module Testbench

```
--
--------------------------------------------------------------------------------

-- Company:
-- Engineer:
--
-- Create Date:   12:33:56 03/20/2014
-- Design Name:
-- Module Name:   overviewTB.vhd
-- Project Name:  Overview
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: CommSystem
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic
   and
-- std_logic_vector for the ports of the unit under test.  Xilinx
   recommends
-- that these types always be used for the top-level I/O of a design in
   order
-- to guarantee that the testbench will bind correctly to the post-
```

```
    implementation
-- simulation model.
--
    ---------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY overviewTB IS
END overviewTB;

ARCHITECTURE behavior OF overviewTB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT CommSystem
    PORT(
        clk : IN  std_logic;
        reset : IN  std_logic;
        clk_enable : IN  std_logic;
        ce_out : OUT  std_logic;
        PNSeq : OUT  std_logic;
        pipe_reg1 : OUT  std_logic_vector(1 downto 0);
                    coef_count : OUT  std_logic_vector(1 downto 0);
        int_reg1 : OUT  std_logic_vector(16 downto 0);
        int_reg2 : OUT  std_logic_vector(16 downto 0);
        int_reg3 : OUT  std_logic_vector(16 downto 0);
        int_reg4 : OUT  std_logic_vector(16 downto 0);
                    int_out : OUT  std_logic_vector(15 downto 0);
        pipe_reg2 : OUT  std_logic_vector(15 downto 0);
                    delaymatch1 : OUT  std_logic_vector(15 downto 0)
                        ;
        delaymatch2 : OUT  std_logic_vector(15 downto 0);
```

```
        en_1 : OUT  std_logic;
        en_2 : OUT  std_logic;
        en_3 : OUT  std_logic;
        en_4 : OUT  std_logic;
        dec_SR_1 : OUT  std_logic_vector(15 downto 0);
        dec_SR_2 : OUT  std_logic_vector(15 downto 0);
        dec_SR_3 : OUT  std_logic_vector(15 downto 0);
        dec_SR_4 : OUT  std_logic_vector(15 downto 0);
        dec_reg1 : OUT  std_logic_vector(30 downto 0);
        dec_reg2 : OUT  std_logic_vector(32 downto 0);
        dec_reg3 : OUT  std_logic_vector(32 downto 0);
        dec_reg4 : OUT  std_logic_vector(32 downto 0);
        dec_reg5 : OUT  std_logic_vector(32 downto 0);
        dec_reg6 : OUT  std_logic_vector(32 downto 0);
                        dec_reg7 : OUT  std_logic_vector(32 downto 0);
                        dec_out : OUT  std_logic_vector(15 downto 0);
        pipe_reg3 : OUT  std_logic_vector(15 downto 0);
        To_DataSink : OUT  std_logic
      );
END COMPONENT;


--Inputs
signal clk : std_logic := '0';
signal reset : std_logic := '0';
signal clk_enable : std_logic := '0';

      --Outputs
signal PNSeq : std_logic;
signal pipe_reg1 : std_logic_vector(1 downto 0);
      signal coef_count : std_logic_vector(1 downto 0);
signal int_reg1 : std_logic_vector(16 downto 0);
signal int_reg2 : std_logic_vector(16 downto 0);
```

```
signal int_reg3 : std_logic_vector(16 downto 0);
signal int_reg4 : std_logic_vector(16 downto 0);
    signal int_out : std_logic_vector(15 downto 0);
signal pipe_reg2 : std_logic_vector(15 downto 0);
    signal delaymatch1 : std_logic_vector(15 downto 0);
signal delaymatch2 : std_logic_vector(15 downto 0);
signal en_1 : std_logic;
signal en_2 : std_logic;
signal en_3 : std_logic;
signal en_4 : std_logic;
    signal dec_sr_en : std_logic_vector(2 downto 0);
signal dec_SR_1 : std_logic_vector(15 downto 0);
signal dec_SR_2 : std_logic_vector(15 downto 0);
signal dec_SR_3 : std_logic_vector(15 downto 0);
signal dec_SR_4 : std_logic_vector(15 downto 0);
signal dec_reg1 : std_logic_vector(30 downto 0);
signal dec_reg2 : std_logic_vector(32 downto 0);
signal dec_reg3 : std_logic_vector(32 downto 0);
signal dec_reg4 : std_logic_vector(32 downto 0);
signal dec_reg5 : std_logic_vector(32 downto 0);
signal dec_reg6 : std_logic_vector(32 downto 0);
    signal dec_reg7 : std_logic_vector(32 downto 0);
    signal dec_out : std_logic_vector(15 downto 0);
signal pipe_reg3 : std_logic_vector(15 downto 0);
signal To_DataSink : std_logic;
signal ce_out : std_logic;

-- Clock period definitions
constant clk_period : time := 10 ns;

    TYPE output_type IS ARRAY (0 to 62) of std_logic;
    CONSTANT FIR_out : output_type := ('1', '0', '0', '0', '0', '0',
        '1',
```

```
                    '0', '0', '0', '0', '1', '1', '0', '0', '0',
                        '1', '0', '1', '0',
                    '0', '1', '1', '1', '1', '0', '1', '0', '0',
                        '0', '1', '1', '1',
                    '0', '0', '1', '0', '0', '1', '0', '1', '1',
                        '0', '1', '1', '1',
                    '0', '1', '1', '0', '0', '1', '1', '0', '1',
                        '0', '1', '0', '1',
                    '1', '1', '1', '1');
    CONSTANT delay : integer := 128;


BEGIN

    -- Instantiate the Unit Under Test (UUT)
uut: CommSystem PORT MAP (
        clk => clk,
        reset => reset,
        clk_enable => clk_enable,
        ce_out => ce_out,
        PNSeq => PNSeq,
        pipe_reg1 => pipe_reg1,
                     coef_count => coef_count,
        int_reg1 => int_reg1,
        int_reg2 => int_reg2,
        int_reg3 => int_reg3,
        int_reg4 => int_reg4,
                     int_out => int_out,
        pipe_reg2 => pipe_reg2,
                     delaymatch1 => delaymatch1,
        delaymatch2 => delaymatch2,
        en_1 => en_1,
        en_2 => en_2,
```

```
        en_3 => en_3,
        en_4 => en_4,
        dec_SR_1 => dec_SR_1,
        dec_SR_2 => dec_SR_2,
        dec_SR_3 => dec_SR_3,
        dec_SR_4 => dec_SR_4,
        dec_reg1 => dec_reg1,
        dec_reg2 => dec_reg2,
        dec_reg3 => dec_reg3,
        dec_reg4 => dec_reg4,
        dec_reg5 => dec_reg5,
        dec_reg6 => dec_reg6,
                    dec_reg7 => dec_reg7,
                    dec_out => dec_out,
        pipe_reg3 => pipe_reg3,
        To_DataSink => To_DataSink
      );

-- Clock process definitions
clk_process :process
begin
             clk <= '0';
             wait for clk_period/2;
             clk <= '1';
             wait for clk_period/2;
end process;

      dec_sr_en <= "001" when en_1 = '1' else
                                          "010" when en_2 = '1'
                                             else
                                          "011" when en_3 = '1'
                                             else
                                          "100";
```

```
-- Stimulus process
stim_proc: process
begin
   -- hold reset state for 100 ns.
   wait for 100 ns;

   wait for clk_period*10;

            clk_enable <= '1';
            reset <= '1';

            wait for clk_period;

            reset <= '0';

            wait for clk_period*(delay+65*4);

            clk_enable <= '0';

            wait for clk_period*5;

            clk_enable <= '1';

            wait for clk_period*8;

            reset <= '1';

            wait for clk_period;

            reset <= '0';

   wait;
```

```
    end process;

        ---------- Output Verification ----------
        ouput_process: process
        begin

                wait until reset = '1';
                wait until rising_edge(clk);
                wait until rising_edge(clk);

                wait for clk_period*delay;

                FOR j IN FIR_out'RANGE LOOP

                        ASSERT To_DataSink = FIR_out(j)
                                REPORT "Incorrect Output"
                                SEVERITY ERROR;

                        wait for clk_period*4;

                END LOOP;

                wait;
        end process;

END;
```

B.2.2 BPSK Modulator Testbench

```
--
--------------------------------------------------------------------------------

-- Company:
-- Engineer:
--
-- Create Date:   01:30:53 03/21/2014
-- Design Name:
-- Module Name:   BPSKModTB.vhd
-- Project Name:  BPSKMod
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: BPSK_Modulator_Baseband
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic
    and
-- std_logic_vector for the ports of the unit under test.  Xilinx
    recommends
-- that these types always be used for the top-level I/O of a design in
    order
-- to guarantee that the testbench will bind correctly to the post-
    implementation
-- simulation model.
```

```
--
    ------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY BPSKModTB IS
END BPSKModTB;

ARCHITECTURE behavior OF BPSKModTB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT BPSK_Modulator_Baseband
    PORT(
         in0 : IN  std_logic;
         out0_re : OUT  std_logic_vector(1 downto 0);
         out0_im : OUT  std_logic_vector(1 downto 0)
        );
    END COMPONENT;

   --Inputs
   signal clk : std_logic := '0';
        signal in0 : std_logic := '1';

        --Outputs
   signal out0_re : std_logic_vector(1 downto 0);
   signal out0_im : std_logic_vector(1 downto 0);

   constant clk_period : time := 10 ns;

BEGIN
```

```
      -- Instantiate the Unit Under Test (UUT)
uut: BPSK_Modulator_Baseband PORT MAP (
        in0 => in0,
        out0_re => out0_re,
        out0_im => out0_im
      );

-- Clock process definitions
clk_process :process
begin
              clk <= '0';
              wait for clk_period/2;
              clk <= '1';
              wait for clk_period/2;
end process;


-- Stimulus process
stim_proc: process
begin
   -- hold reset state for 100 ns.
   wait for 100 ns;



   wait for clk_period*11;


              in0 <= '1';

              wait for clk_period*2;
```

```
    in0 <= '0';

                wait for clk_period*2;

                ASSERT out0_re = "01"
                        REPORT "Output Incorrect"
                        SEVERITY NOTE;

                ASSERT out0_im = "00"
                        REPORT "Output Incorrect"
                        SEVERITY NOTE;

                wait for clk_period*2;

                in0 <= '1';

                wait for clk_period*2;

                ASSERT out0_re = "11"
                        REPORT "Output Incorrect"
                        SEVERITY NOTE;

                ASSERT out0_im = "00"
                        REPORT "Output Incorrect"
                        SEVERITY NOTE;

                wait for clk_period*2;

                in0 <= '0';

    wait;
end process;
```

```
END;
```

B.2.3 FIR Interpolation Filter Impulse Test Testbench

```
--
--------------------------------------------------------------------------------

-- Company:
-- Engineer:
--
-- Create Date:   17:11:46 03/21/2014
-- Design Name:
-- Module Name:   InterpTB.vhd
-- Project Name:  Interp
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: FIR_Interpolation
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic
    and
-- std_logic_vector for the ports of the unit under test.  Xilinx
    recommends
-- that these types always be used for the top-level I/O of a design in
    order
-- to guarantee that the testbench will bind correctly to the post-
    implementation
-- simulation model.
```

```
--
--------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.conv_std_logic_vector;
USE ieee.std_logic_arith.conv_integer;

ENTITY InterpTB IS
END InterpTB;

ARCHITECTURE behavior OF InterpTB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT FIR_Interpolation
    PORT(
         clk : IN  std_logic;
         enb_1_1_1 : IN  std_logic;
         reset : IN  std_logic;
         FIR_Interpolation_in_re : IN  std_logic_vector(1 downto 0);
         FIR_Interpolation_in_im : IN  std_logic_vector(1 downto 0);
         coef_count : OUT  std_logic_vector(1 downto 0);
         int_reg1 : OUT  std_logic_vector(16 downto 0);
         int_reg2 : OUT  std_logic_vector(16 downto 0);
         int_reg3 : OUT  std_logic_vector(16 downto 0);
         int_reg4 : OUT  std_logic_vector(16 downto 0);
         FIR_Interpolation_out_re : OUT  std_logic_vector(15 downto 0);
         FIR_Interpolation_out_im : OUT  std_logic_vector(15 downto 0)
        );
    END COMPONENT;
```

```
--Inputs
signal clk : std_logic := '0';
signal enb_1_1_1 : std_logic := '1';
signal reset : std_logic := '0';
signal FIR_Interpolation_in_re : std_logic_vector(1 downto 0) := (
   others => '0');
signal FIR_Interpolation_in_im : std_logic_vector(1 downto 0) := (
   others => '0');

    --Outputs
signal coef_count : std_logic_vector(1 downto 0);
signal int_reg1 : std_logic_vector(16 downto 0);
signal int_reg2 : std_logic_vector(16 downto 0);
signal int_reg3 : std_logic_vector(16 downto 0);
signal int_reg4 : std_logic_vector(16 downto 0);
signal FIR_Interpolation_out_re : std_logic_vector(15 downto 0);
signal FIR_Interpolation_out_im : std_logic_vector(15 downto 0);

-- Clock period definitions
constant clk_period : time := 10 ns;

    TYPE coef_type IS ARRAY (0 to 83) of integer;
    CONSTANT FIR_out : coef_type := (5, -41, -52, -13, 41, 60, 20,
        -48, -83, -43,
     48, 113, 88, -19, -124, -132, -25, 112, 152, 46, -123, -187,
        -48, 210, 348,
     163, -300, -699, -615, 99, 1070, 1542, 869, -901, -2790, -3256,
        -1052, 3897,
     10188, 15453, 17504, 15453, 10188, 3897, -1052, -3256, -2790,
        -901, 869,
     1542, 1070, 99, -615, -699, -300, 163, 348, 210, -48, -187,
        -123, 46, 152,
     112, -25, -132, -124, -19, 88, 113, 48, -43, -83, -48, 20, 60,
```

```
        41, -13,
      -52, -41, 5, 0, 0, 0);

BEGIN

      -- Instantiate the Unit Under Test (UUT)
   uut: FIR_Interpolation PORT MAP (
        clk => clk,
        enb_1_1_1 => enb_1_1_1,
        reset => reset,
        FIR_Interpolation_in_re => FIR_Interpolation_in_re,
        FIR_Interpolation_in_im => FIR_Interpolation_in_im,
        coef_count => coef_count,
        int_reg1 => int_reg1,
        int_reg2 => int_reg2,
        int_reg3 => int_reg3,
        int_reg4 => int_reg4,
        FIR_Interpolation_out_re => FIR_Interpolation_out_re,
        FIR_Interpolation_out_im => FIR_Interpolation_out_im
      );

   -- Clock process definitions
   clk_process :process
   begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
```

```
begin
   -- hold reset state for 100 ns.
   wait for 100 ns;

   wait for clk_period*10;

   reset <= '1';
            enb_1_1_1 <= '1';

            wait for clk_period;

            reset <= '0';
            FIR_Interpolation_in_re <= "01";

            wait for clk_period*4;

            FIR_Interpolation_in_re <= "00";

            ------------ Start Testing
                  ------------------------------------

            wait for clk_period;

            FOR j IN FIR_out'RANGE LOOP
                     ASSERT FIR_Interpolation_out_re =
                        conv_std_logic_vector(FIR_out(j),16)
                           REPORT "Incorrect Output"
                           SEVERITY ERROR;

                     wait for clk_period;
            END LOOP;


   wait;
```

```
    end process;

END;
```

B.2.4 FIR Interpolation Filter Maximum Test Testbench

```
--
--------------------------------------------------------------------------------

-- Company:
-- Engineer:
--
-- Create Date:   13:39:15 03/22/2014
-- Design Name:
-- Module Name:   Interp_MaxTB.vhd
-- Project Name:  Interp
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: FIR_Interpolation
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic
    and
-- std_logic_vector for the ports of the unit under test.  Xilinx
    recommends
-- that these types always be used for the top-level I/O of a design in
    order
-- to guarantee that the testbench will bind correctly to the post-
    implementation
-- simulation model.
```

```
--
    ------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.conv_signed;

ENTITY Interp_MaxTB IS
END Interp_MaxTB;

ARCHITECTURE behavior OF Interp_MaxTB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT FIR_Interpolation
    PORT(
         clk : IN  std_logic;
         enb_1_1_1 : IN  std_logic;
         reset : IN  std_logic;
         FIR_Interpolation_in_re : IN  std_logic_vector(1 downto 0);
         FIR_Interpolation_in_im : IN  std_logic_vector(1 downto 0);
         coef_count : OUT  std_logic_vector(1 downto 0);
         int_reg1 : OUT  std_logic_vector(16 downto 0);
         int_reg2 : OUT  std_logic_vector(16 downto 0);
         int_reg3 : OUT  std_logic_vector(16 downto 0);
         int_reg4 : OUT  std_logic_vector(16 downto 0);
         FIR_Interpolation_out_re : OUT  std_logic_vector(15 downto 0);
         FIR_Interpolation_out_im : OUT  std_logic_vector(15 downto 0)
        );
    END COMPONENT;


   --Inputs
```

```
signal clk : std_logic := '0';
signal enb_1_1_1 : std_logic := '0';
signal reset : std_logic := '0';
signal FIR_Interpolation_in_re : std_logic_vector(1 downto 0) := (
   others => '0');
signal FIR_Interpolation_in_im : std_logic_vector(1 downto 0) := (
   others => '0');

    --Outputs
signal coef_count : std_logic_vector(1 downto 0);
signal int_reg1 : std_logic_vector(16 downto 0);
signal int_reg2 : std_logic_vector(16 downto 0);
signal int_reg3 : std_logic_vector(16 downto 0);
signal int_reg4 : std_logic_vector(16 downto 0);
signal FIR_Interpolation_out_re : std_logic_vector(15 downto 0);
signal FIR_Interpolation_out_im : std_logic_vector(15 downto 0);

-- Clock period definitions
constant clk_period : time := 10 ns;

    TYPE input_type IS ARRAY (0 to 19) of integer;
    CONSTANT FIR_in : input_type := (-1, 1, 1, -1, 1, -1, -1, 1, -1,
        1,
                    1, -1, 1, -1, -1, 1, -1, 1, 1, -1);
    TYPE output_type IS ARRAY (0 to 79) of integer;
    CONSTANT FIR_out : output_type := (-5, 41, 52, 13, -36, -101,
        -72, 35,
                    129, 62, -80, -174, -135, 77, 244, 210, -6,
                        -275, -300, -30, 305,
                    424, 100, -444, -713, -281, 608, 1261, 1000,
                        -312, -1814, -2287,
                    -1007, 1625, 3908, 3919, 653, -5286, -11708,
                        -16024, -16359, -12275,
```

```
                        -4468, 5497, 15635, 23819, 27770, 25794, 17642,
                            5133, -7768, -16475,
                        -17725, -11182, 100, 11162, 17317, 16164, 8076,
                            -4147, -16648, -25896,
                        -29340, -26029, -16777, -4046, 8296, 16256,
                            17141, 10773, -72, -10839,
                        -17048, -16181, -8324, 3913, 16637, 26106,
                            29584, 26106);
    CONSTANT delay : integer := 5;


BEGIN

    -- Instantiate the Unit Under Test (UUT)
uut: FIR_Interpolation PORT MAP (
        clk => clk,
        enb_1_1_1 => enb_1_1_1,
        reset => reset,
        FIR_Interpolation_in_re => FIR_Interpolation_in_re,
        FIR_Interpolation_in_im => FIR_Interpolation_in_im,
        coef_count => coef_count,
        int_reg1 => int_reg1,
        int_reg2 => int_reg2,
        int_reg3 => int_reg3,
        int_reg4 => int_reg4,
        FIR_Interpolation_out_re => FIR_Interpolation_out_re,
        FIR_Interpolation_out_im => FIR_Interpolation_out_im
    );

-- Clock process definitions
clk_process :process
begin
            clk <= '0';
```

```
            wait for clk_period/2;
            clk <= '1';
            wait for clk_period/2;
end process;


-- Stimulus process
stim_proc: process
begin
   -- hold reset state for 100 ns.
   wait for 100 ns;

   wait for clk_period*10;

            ------------ Input -----------------------------------

            FOR j IN FIR_in'RANGE LOOP

                  IF j = 0 THEN
                        reset <= '1';
                        enb_1_1_1 <= '1';
                        FIR_Interpolation_in_re <=
                            std_logic_vector(conv_signed(FIR_in(
                            j),2));

                        wait for clk_period;

                        reset <= '0';

                        wait for clk_period*3.51;

                  ELSE
```

```
                    FIR_Interpolation_in_re <=
                        std_logic_vector(conv_signed(FIR_in(
                        j),2));

                    wait for clk_period*4;

                END IF;

            END LOOP;

            FIR_Interpolation_in_re <= "00";

    wait;
end process;

    ---------- Output Verification ----------
    ouput_process: process
    begin

            wait until reset = '1';
            wait until reset = '0';

            wait for clk_period*delay;

            FOR j IN FIR_out'RANGE LOOP

                ASSERT FIR_Interpolation_out_re =
                    std_logic_vector(conv_signed(FIR_out(j),16))
                REPORT "Incorrect Output"
                SEVERITY ERROR;

                wait for clk_period;
```

```
                END LOOP;

                wait;
        end process;

END;
```

B.2.5 FIR Decimation Filter Impulse Testbench

```
--
--------------------------------------------------------------------------------

-- Company:
-- Engineer:
--
-- Create Date:   00:26:25 03/23/2014
-- Design Name:
-- Module Name:   Decim_ImpulseTB.vhd
-- Project Name:  Decim
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: FIR_Decimation
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic
    and
-- std_logic_vector for the ports of the unit under test.  Xilinx
    recommends
-- that these types always be used for the top-level I/O of a design in
    order
-- to guarantee that the testbench will bind correctly to the post-
    implementation
-- simulation model.
```

```
--
    ------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.conv_signed;
--USE ieee.numeric_std.ALL;

ENTITY Decim_ImpulseTB IS
END Decim_ImpulseTB;

ARCHITECTURE behavior OF Decim_ImpulseTB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT FIR_Decimation
    PORT(
        clk : IN  std_logic;
        enb_1_1_1 : IN  std_logic;
        reset : IN  std_logic;
        FIR_Decimation_in_re : IN  std_logic_vector(15 downto 0);
        FIR_Decimation_in_im : IN  std_logic_vector(15 downto 0);
        en_1 : OUT  std_logic;
        en_2 : OUT  std_logic;
        en_3 : OUT  std_logic;
        en_4 : OUT  std_logic;
        dec_SR_1 : OUT  std_logic_vector(15 downto 0);
        dec_SR_2 : OUT  std_logic_vector(15 downto 0);
        dec_SR_3 : OUT  std_logic_vector(15 downto 0);
        dec_SR_4 : OUT  std_logic_vector(15 downto 0);
        dec_reg1 : OUT  std_logic_vector(30 downto 0);
        dec_reg2 : OUT  std_logic_vector(32 downto 0);
        dec_reg3 : OUT  std_logic_vector(32 downto 0);
```

```
        dec_reg4 : OUT  std_logic_vector(32 downto 0);
        dec_reg5 : OUT  std_logic_vector(32 downto 0);
        dec_reg6 : OUT  std_logic_vector(32 downto 0);
        dec_reg7 : OUT  std_logic_vector(32 downto 0);
        FIR_Decimation_out_re : OUT  std_logic_vector(15 downto 0);
        FIR_Decimation_out_im : OUT  std_logic_vector(15 downto 0)
      );
END COMPONENT;


--Inputs
signal clk : std_logic := '0';
signal enb_1_1_1 : std_logic := '0';
signal reset : std_logic := '0';
signal FIR_Decimation_in_re : std_logic_vector(15 downto 0) := (
   others => '0');
signal FIR_Decimation_in_im : std_logic_vector(15 downto 0) := (
   others => '0');

     --Outputs
signal en_1 : std_logic;
signal en_2 : std_logic;
signal en_3 : std_logic;
signal en_4 : std_logic;
signal dec_SR_1 : std_logic_vector(15 downto 0);
signal dec_SR_2 : std_logic_vector(15 downto 0);
signal dec_SR_3 : std_logic_vector(15 downto 0);
signal dec_SR_4 : std_logic_vector(15 downto 0);
signal dec_reg1 : std_logic_vector(30 downto 0);
signal dec_reg2 : std_logic_vector(32 downto 0);
signal dec_reg3 : std_logic_vector(32 downto 0);
signal dec_reg4 : std_logic_vector(32 downto 0);
signal dec_reg5 : std_logic_vector(32 downto 0);
```

```
signal dec_reg6 : std_logic_vector(32 downto 0);
signal dec_reg7 : std_logic_vector(32 downto 0);
signal FIR_Decimation_out_re : std_logic_vector(15 downto 0);
signal FIR_Decimation_out_im : std_logic_vector(15 downto 0);

-- Clock period definitions
constant clk_period : time := 10 ns;

    TYPE input_type IS ARRAY (0 to 99) of integer;
    CONSTANT FIR_in : input_type := (-5, 41, 52, 13, -36, -101,
                    -72, 35, 129, 62, -80, -174, -125, -5, 140, 184,
                        76, -155, -260,
                    -126, 149, 256, 92, -244, -455, -199, 400, 901,
                        784, -174, -1414,
                    -1969, -1067, 1131, 3460, 4049, 1381, -4616,
                        -11964, -17426,
                    -18001, -12537, -2328, 9227, 18245, 22305,
                        21342, 17620, 14248,
                    13431, 15156, 17633, 18703, 17346, 14656, 12934,
                        14063, 17896,
                    22112, 23211, 18608, 8364, -4300, -14559,
                        -18313, -14106, -3920,
                    8138, 17843, 22631, 22500, 19299, 15482, 13025,
                        12668, 14193,
                    16787, 19302, 20680, 19900, 16286, 9806, 1228,
                        -7982, -16188,
                    -21942, -24280, -22698, -17182, -8362, 2140,
                        11830, 18150, 19782,
                    17596, 14438, 13532, 16328, 21144, 23782);
    TYPE output_type IS ARRAY (0 to 44) of integer;
    CONSTANT FIR_out : output_type := (-1, -1, 0, 0, -2, 1, -1, -5,
        12, -24,
                    54, -46, -12, -37, -4, -59, 64, -49, -15, -18,
```

```
                    -16426, 16420,
                16418, 16345, 16388, 16311, -16319, 16362,
                    16356, 16446, 16374,
                -16539, -15952, 15496, 17770, 8776, -1866, 731,
                    -254, -10, 68,
                -63, 52, -16, -31);
    CONSTANT delay : integer := 7;


BEGIN

    -- Instantiate the Unit Under Test (UUT)
  uut: FIR_Decimation PORT MAP (
        clk => clk,
        enb_1_1_1 => enb_1_1_1,
        reset => reset,
        FIR_Decimation_in_re => FIR_Decimation_in_re,
        FIR_Decimation_in_im => FIR_Decimation_in_im,
        en_1 => en_1,
        en_2 => en_2,
        en_3 => en_3,
        en_4 => en_4,
        dec_SR_1 => dec_SR_1,
        dec_SR_2 => dec_SR_2,
        dec_SR_3 => dec_SR_3,
        dec_SR_4 => dec_SR_4,
        dec_reg1 => dec_reg1,
        dec_reg2 => dec_reg2,
        dec_reg3 => dec_reg3,
        dec_reg4 => dec_reg4,
        dec_reg5 => dec_reg5,
        dec_reg6 => dec_reg6,
        dec_reg7 => dec_reg7,
```

```
        FIR_Decimation_out_re => FIR_Decimation_out_re,
        FIR_Decimation_out_im => FIR_Decimation_out_im
      );

-- Clock process definitions
clk_process :process
begin
              clk <= '0';
              wait for clk_period/2;
              clk <= '1';
              wait for clk_period/2;
end process;


-- Stimulus process
stim_proc: process
begin
   -- hold reset state for 100 ns.
   wait for 100 ns;

   wait for clk_period*10;

   ------------ Input -----------------------------------

            FOR j IN FIR_in'RANGE LOOP

                  IF j = 0 THEN
                        reset <= '1';
                        enb_1_1_1 <= '1';
                        FIR_Decimation_in_re <= std_logic_vector
                            (conv_signed(FIR_in(j),16));

                        wait for clk_period;
```

```
                    reset <= '0';

                    wait for clk_period*0.51;

                ELSE

                    FIR_Decimation_in_re <= std_logic_vector
                        (conv_signed(FIR_in(j),16));

                    wait for clk_period;

                END IF;

            END LOOP;

            FIR_Decimation_in_re <= (others => '0');

    wait;
end process;

    ---------- Output Verification ----------
    ouput_process: process
    begin

            wait until en_3 = '1';

            wait for clk_period*4*delay;

            FOR j IN FIR_out'RANGE LOOP

                ASSERT FIR_Decimation_out_re = std_logic_vector(
                    conv_signed(FIR_out(j),16))
```

```
                    REPORT "Incorrect Output"
                    SEVERITY ERROR;

                    wait for clk_period*4;

                END LOOP;

                wait;
            end process;

END;
```

B.2.6 BPSK Demodulator Testbench

```
--
    ----------------------------------------------------------------------------------


-- Company:
-- Engineer:
--
-- Create Date:    11:50:38 03/21/2014
-- Design Name:
-- Module Name:    BPSKDemod/BPSKDemodTB.vhd
-- Project Name:   BPSKDemod
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: BPSK_Demodulator_Baseband
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic
     and
-- std_logic_vector for the ports of the unit under test.  Xilinx
    recommends
-- that these types always be used for the top-level I/O of a design in
    order
-- to guarantee that the testbench will bind correctly to the post-
    implementation
-- simulation model.
```

```
--
    ------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY BPSKDemodTB IS
END BPSKDemodTB;

ARCHITECTURE behavior OF BPSKDemodTB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT BPSK_Demodulator_Baseband
    PORT(
         in0_re : IN  std_logic_vector(15 downto 0);
         in0_im : IN  std_logic_vector(15 downto 0);
         out0 : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
        signal clk : std_logic;
   signal in0_re : std_logic_vector(15 downto 0) := (others => '0');
   signal in0_im : std_logic_vector(15 downto 0) := (others => '1');

        --Outputs
   signal out0 : std_logic;


   constant clk_period : time := 10 ns;
```

```
BEGIN

        -- Instantiate the Unit Under Test (UUT)
    uut: BPSK_Demodulator_Baseband PORT MAP (
            in0_re => in0_re,
            in0_im => in0_im,
            out0 => out0
        );

    -- Clock process definitions
    clk_process :process
    begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
    end process;


    -- Stimulus process
    stim_proc: process
    begin
       -- hold reset state for 100 ns.
       wait for 100 ns;

                ------------------------------------------------
                -- re: negative im: zero
       wait for clk_period*10;

       in0_re <= (15 => '1', 0 => '0', others => '0');
                in0_im <= (0 => '0', others => '0');
```

```
        wait for clk_period;

        ASSERT out0 = '1'
                REPORT "Output Incorrect"
                SEVERITY NOTE;

        -- re: negative im: non-zero
        wait for clk_period;

in0_re <= (15 => '1', 0 => '0', others => '0');
        in0_im <= (0 => '1', others => '0');

        wait for clk_period;

        ASSERT out0 = '1'
                REPORT "Output Incorrect"
                SEVERITY NOTE;

        ------------------------------------------

        -- re: negative one im: zero
wait for clk_period;

in0_re <= (15 => '1', 0 => '1', others => '1');
        in0_im <= (0 => '0', others => '0');

        wait for clk_period;

        ASSERT out0 = '1'
                REPORT "Output Incorrect"
                SEVERITY NOTE;

        -- re: negative im: non-zero
        wait for clk_period;
```

```
in0_re <= (15 => '1', 0 => '1', others => '1');
          in0_im <= (0 => '1', others => '0');

          wait for clk_period;

          ASSERT out0 = '1'
                  REPORT "Output Incorrect"
                  SEVERITY NOTE;

          ------------------------------------------

          -- re: zero im: zero
          wait for clk_period;

          in0_re <= (15 => '0', 0 => '0', others => '0');
          in0_im <= (0 => '0', others => '0');

          wait for clk_period;

          ASSERT out0 = '0'
                  REPORT "Output Incorrect"
                  SEVERITY NOTE;

          -- re: zero im: non-zero
          wait for clk_period;

          in0_re <= (15 => '0', 0 => '0', others => '0');
          in0_im <= (0 => '1', others => '0');

          wait for clk_period;

          ASSERT out0 = '1'
                  REPORT "Output Incorrect"
```

```
        SEVERITY NOTE;

-------------------------------------------------

-- re: positive im: zero
wait for clk_period;

in0_re <= (15 => '0', 0 => '1', others => '0');
in0_im <= (0 => '0', others => '0');

wait for clk_period;

ASSERT out0 = '0'
        REPORT "Output Incorrect"
        SEVERITY NOTE;

-- re: positive im: non-zero
wait for clk_period;

in0_re <= (15 => '0', 0 => '1', others => '0');
in0_im <= (0 => '1', others => '0');

wait for clk_period;

ASSERT out0 = '0'
        REPORT "Output Incorrect"
        SEVERITY NOTE;

-------------------------------------------------

-- re: positive im: zero
wait for clk_period;

in0_re <= (15 => '0', 0 => '1', others => '1');
in0_im <= (0 => '0', others => '0');
```

```
                wait for clk_period;

                ASSERT out0 = '0'
                        REPORT "Output Incorrect"
                        SEVERITY NOTE;

                -- re: positive im: non-zero
                wait for clk_period;

                in0_re <= (15 => '0', 0 => '1', others => '1');
                in0_im <= (0 => '1', others => '0');

                wait for clk_period;

                ASSERT out0 = '0'
                        REPORT "Output Incorrect"
                        SEVERITY NOTE;

                ------------------------------------------

                wait for clk_period;

                in0_re <= (15 => '0', 0 => '0', others => '0');
                in0_im <= (0 => '0', others => '0');

        wait;
    end process;

END;
```

B.3 BPSK Hand-Optimized Design

B.3.1 Top Module Testbench

```
--
--------------------------------------------------------------------------------

-- Company:
-- Engineer:
--
-- Create Date:   22:07:20 03/20/2014
-- Design Name:
-- Module Name:   overviewTB.vhd
-- Project Name:  Overview
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: CommSystem
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic
   and
-- std_logic_vector for the ports of the unit under test.  Xilinx
   recommends
-- that these types always be used for the top-level I/O of a design in
   order
-- to guarantee that the testbench will bind correctly to the post-
```

```
    implementation
-- simulation model.
--
    ------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY overviewTB IS
END overviewTB;

ARCHITECTURE behavior OF overviewTB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT CommSystem
    PORT(
        clk : IN  std_logic;
        reset : IN  std_logic;
        clk_enable : IN  std_logic;
        ce_out : OUT  std_logic;
        PNSeq : OUT  std_logic;
                    coef_count : OUT std_logic_vector(1 downto 0);
        int_SR : OUT  std_logic_vector(1 downto 0);
        int_reg1 : OUT  std_logic_vector(15 downto 0);
        int_reg2 : OUT  std_logic_vector(15 downto 0);
        int_reg3 : OUT  std_logic_vector(15 downto 0);
        int_reg4 : OUT  std_logic_vector(15 downto 0);
        int_out : OUT  std_logic_vector(15 downto 0);
        en_1 : OUT  std_logic;
        en_2 : OUT  std_logic;
        en_3 : OUT  std_logic;
        en_4 : OUT  std_logic;
```

```
        dec_SR_1 : OUT  std_logic_vector(15 downto 0);
        dec_SR_2 : OUT  std_logic_vector(15 downto 0);
        dec_SR_3 : OUT  std_logic_vector(15 downto 0);
        dec_SR_4 : OUT  std_logic_vector(15 downto 0);
        dec_reg1 : OUT  std_logic_vector(30 downto 0);
        dec_reg2 : OUT  std_logic_vector(32 downto 0);
        dec_reg3 : OUT  std_logic_vector(32 downto 0);
        dec_reg4 : OUT  std_logic_vector(32 downto 0);
        dec_reg5 : OUT  std_logic_vector(32 downto 0);
        dec_reg6 : OUT  std_logic_vector(32 downto 0);
                        dec_out : OUT  std_logic_vector(15 downto 0);
        pipe_reg1 : OUT  std_logic_vector(15 downto 0);
        To_DataSink : OUT  std_logic
      );
END COMPONENT;


--Inputs
signal clk : std_logic := '0';
signal reset : std_logic := '0';
signal clk_enable : std_logic := '0';

      --Outputs
signal PNSeq : std_logic;
      signal coef_count : std_logic_vector(1 downto 0);
signal int_SR : std_logic_vector(1 downto 0);
signal int_reg1 : std_logic_vector(15 downto 0);
signal int_reg2 : std_logic_vector(15 downto 0);
signal int_reg3 : std_logic_vector(15 downto 0);
signal int_reg4 : std_logic_vector(15 downto 0);
signal int_out : std_logic_vector(15 downto 0);
signal en_1 : std_logic;
signal en_2 : std_logic;
```

```
signal en_3 : std_logic;
signal en_4 : std_logic;
    signal dec_SR_en : std_logic_vector(2 downto 0);
signal dec_SR_1 : std_logic_vector(15 downto 0);
signal dec_SR_2 : std_logic_vector(15 downto 0);
signal dec_SR_3 : std_logic_vector(15 downto 0);
signal dec_SR_4 : std_logic_vector(15 downto 0);
signal dec_reg1 : std_logic_vector(30 downto 0);
signal dec_reg2 : std_logic_vector(32 downto 0);
signal dec_reg3 : std_logic_vector(32 downto 0);
signal dec_reg4 : std_logic_vector(32 downto 0);
signal dec_reg5 : std_logic_vector(32 downto 0);
signal dec_reg6 : std_logic_vector(32 downto 0);
    signal dec_out : std_logic_vector(15 downto 0);
signal pipe_reg1 : std_logic_vector(15 downto 0);
signal To_DataSink : std_logic;
    signal ce_out : std_logic;

-- Clock period definitions
constant clk_period : time := 10 ns;

    TYPE output_type IS ARRAY (0 to 62) of std_logic;
    CONSTANT FIR_out : output_type := ('1', '0', '0', '0', '0', '0',
        '1',
                '0', '0', '0', '0', '1', '1', '0', '0', '0',
                    '1', '0', '1', '0',
                '0', '1', '1', '1', '1', '0', '1', '0', '0',
                    '0', '1', '1', '1',
                '0', '0', '1', '0', '0', '1', '0', '1', '1',
                    '0', '1', '1', '1',
                '0', '1', '1', '0', '0', '1', '1', '0', '1',
                    '0', '1', '0', '1',
                '1', '1', '1', '1');
```

```
        CONSTANT delay : integer := 97;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
    uut: CommSystem PORT MAP (
            clk => clk,
            reset => reset,
            clk_enable => clk_enable,
            ce_out => ce_out,
            PNSeq => PNSeq,
                                coef_count => coef_count,
            int_SR => int_SR,
            int_reg1 => int_reg1,
            int_reg2 => int_reg2,
            int_reg3 => int_reg3,
            int_reg4 => int_reg4,
            int_out => int_out,
            en_1 => en_1,
            en_2 => en_2,
            en_3 => en_3,
            en_4 => en_4,
            dec_SR_1 => dec_SR_1,
            dec_SR_2 => dec_SR_2,
            dec_SR_3 => dec_SR_3,
            dec_SR_4 => dec_SR_4,
            dec_reg1 => dec_reg1,
            dec_reg2 => dec_reg2,
            dec_reg3 => dec_reg3,
            dec_reg4 => dec_reg4,
            dec_reg5 => dec_reg5,
            dec_reg6 => dec_reg6,
                                dec_out => dec_out,
```

```
        pipe_reg1 => pipe_reg1,
        To_DataSink => To_DataSink
      );

-- Clock process definitions
clk_process :process
begin
              clk <= '0';
              wait for clk_period/2;
              clk <= '1';
              wait for clk_period/2;
end process;

      dec_sr_en <= "001" when en_1 = '1' else
                                          "010" when en_2 = '1'
                                             else
                                          "011" when en_3 = '1'
                                             else
                                          "100" when en_4 = '1';

-- Stimulus process
stim_proc: process
begin
   -- hold reset state for 100 ns.
   wait for 100 ns;

   wait for clk_period*10;

   clk_enable <= '1';
              reset <= '1';

              wait for clk_period;
```

```
                reset <= '0';

                wait for clk_period*(delay+65*4);

                clk_enable <= '0';

                wait for clk_period*5;

                clk_enable <= '1';

                wait for clk_period*8;

                reset <= '1';

                wait for clk_period;

                reset <= '0';

    wait;
end process;

    ------------ Output Verification -------------------
    ouput_process: process
    begin

                wait until reset = '1';
                wait until rising_edge(clk);
                wait until rising_edge(clk);

                wait for clk_period*delay;

                FOR j IN FIR_out'RANGE LOOP
```

```
                    ASSERT To_DataSink = FIR_out(j)
                            REPORT "Incorrect Output"
                            SEVERITY ERROR;

                    wait for clk_period*4;

            END LOOP;

            wait;
        end process;

END;
```

B.3.2 BPSK Modulator Testbench

```
--
--------------------------------------------------------------------------------

-- Company:
-- Engineer:
--
-- Create Date:   03:24:53 03/21/2014
-- Design Name:
-- Module Name:   BPSKModTB.vhd
-- Project Name:  BPSKMod
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: BPSK_Modulator_Baseband
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic
    and
-- std_logic_vector for the ports of the unit under test.  Xilinx
    recommends
-- that these types always be used for the top-level I/O of a design in
    order
-- to guarantee that the testbench will bind correctly to the post-
    implementation
-- simulation model.
```

```
--
    ------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY BPSKModTB IS
END BPSKModTB;

ARCHITECTURE behavior OF BPSKModTB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT BPSK_Modulator_Baseband
    PORT(
         bin : IN  std_logic;
         sym : OUT  std_logic_vector(1 downto 0)
        );
    END COMPONENT;


   --Inputs
        signal clk : std_logic := '0';
   signal bin : std_logic := '1';

        --Outputs
   signal sym : std_logic_vector(1 downto 0);


   constant clk_period : time := 10 ns;

BEGIN
```

```
      -- Instantiate the Unit Under Test (UUT)
uut: BPSK_Modulator_Baseband PORT MAP (
        bin => bin,
        sym => sym
      );

-- Clock process definitions
clk_process :process
begin
              clk <= '0';
              wait for clk_period/2;
              clk <= '1';
              wait for clk_period/2;
end process;


-- Stimulus process
stim_proc: process
begin
   -- hold reset state for 100 ns.
   wait for 100 ns;

   wait for clk_period*11;

   bin <= '1';

              wait for clk_period*2;

   bin <= '0';

              wait for clk_period*2;

              ASSERT sym /= "01"
```

```
                    REPORT "Output␣Correct"
                    SEVERITY NOTE;

                wait for clk_period*2;

                bin <= '1';

                wait for clk_period*2;

                ASSERT sym /= "11"
                    REPORT "Output␣Correct"
                    SEVERITY NOTE;

                wait for clk_period*2;

                bin <= '0';

        wait;
    end process;

END;
```

B.3.3 FIR Interpolation Filter Impulse Test Testbench

```
--
--------------------------------------------------------------------------------

-- Company:
-- Engineer:
--
-- Create Date:   21:41:37 03/21/2014
-- Design Name:
-- Module Name:   Interp_Impulse.vhd
-- Project Name:  Interp
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: FIR_Interpolation
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic
   and
-- std_logic_vector for the ports of the unit under test.  Xilinx
   recommends
-- that these types always be used for the top-level I/O of a design in
   order
-- to guarantee that the testbench will bind correctly to the post-
   implementation
-- simulation model.
```

```
--
    ------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.conv_std_logic_vector;

ENTITY Interp_Impulse IS
END Interp_Impulse;

ARCHITECTURE behavior OF Interp_Impulse IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT FIR_Interpolation
    PORT(
         clk : IN  std_logic;
         enb_1_1_1 : IN  std_logic;
         reset : IN  std_logic;
         FIR_Interpolation_in : IN  std_logic_vector(1 downto 0);
                        coef_count : OUT  std_logic_vector(1 downto 0);
         int_SR : OUT  std_logic_vector(1 downto 0);
         int_reg1 : OUT  std_logic_vector(15 downto 0);
         int_reg2 : OUT  std_logic_vector(15 downto 0);
         int_reg3 : OUT  std_logic_vector(15 downto 0);
         int_reg4 : OUT  std_logic_vector(15 downto 0);
         FIR_Interpolation_out : OUT  std_logic_vector(15 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal clk : std_logic := '0';
```

```
signal enb_1_1_1 : std_logic := '0';
signal reset : std_logic := '0';
signal FIR_Interpolation_in : std_logic_vector(1 downto 0) := (others
    => '0');

    --Outputs
    signal coef_count : std_logic_vector(1 downto 0);
signal int_SR : std_logic_vector(1 downto 0);
signal int_reg1 : std_logic_vector(15 downto 0);
signal int_reg2 : std_logic_vector(15 downto 0);
signal int_reg3 : std_logic_vector(15 downto 0);
signal int_reg4 : std_logic_vector(15 downto 0);
signal FIR_Interpolation_out : std_logic_vector(15 downto 0);

-- Clock period definitions
constant clk_period : time := 10 ns;
    TYPE coef_type IS ARRAY (0 to 83) of integer;
    CONSTANT FIR_out : coef_type := (5, -41, -52, -13, 41, 60, 20,
        -48, -83, -43,
    48, 113, 88, -19, -124, -132, -25, 112, 152, 46, -123, -187,
        -48, 210, 348,
    163, -300, -699, -615, 99, 1070, 1542, 869, -901, -2790, -3256,
        -1052, 3897,
    10188, 15453, 17504, 15453, 10188, 3897, -1052, -3256, -2790,
        -901, 869,
    1542, 1070, 99, -615, -699, -300, 163, 348, 210, -48, -187,
        -123, 46, 152,
    112, -25, -132, -124, -19, 88, 113, 48, -43, -83, -48, 20, 60,
        41, -13,
    -52, -41, 5, 0, 0, 0);


BEGIN
```

```
    -- Instantiate the Unit Under Test (UUT)
uut: FIR_Interpolation PORT MAP (
       clk => clk,
       enb_1_1_1 => enb_1_1_1,
       reset => reset,
       FIR_Interpolation_in => FIR_Interpolation_in,
                     coef_count => coef_count,
                     int_SR => int_SR,
       int_reg1 => int_reg1,
       int_reg2 => int_reg2,
       int_reg3 => int_reg3,
       int_reg4 => int_reg4,
       FIR_Interpolation_out => FIR_Interpolation_out
     );

-- Clock process definitions
clk_process :process
begin
             clk <= '0';
             wait for clk_period/2;
             clk <= '1';
             wait for clk_period/2;
end process;


-- Stimulus process
stim_proc: process
begin
   -- hold reset state for 100 ns.
   wait for 100 ns;

   wait for clk_period*10;
```

```
        reset <= '1';
                enb_1_1_1 <= '1';

                wait for clk_period;

                reset <= '0';
                FIR_Interpolation_in <= "01";

                wait for clk_period*4;

                FIR_Interpolation_in <= "00";

                ------------ Start Testing
                    ------------------------------------

                wait for clk_period*3;

                FOR j IN FIR_out'RANGE LOOP
                        ASSERT FIR_Interpolation_out =
                            conv_std_logic_vector(FIR_out(j),16)
                                REPORT "Incorrect Output"
                                SEVERITY ERROR;

                        wait for clk_period;
                END LOOP;

        wait;
    end process;

END;
```

B.3.4 FIR Interpolation Filter Maximum Test Testbench

```
--
--------------------------------------------------------------------------------

-- Company:
-- Engineer:
--
-- Create Date:   22:48:52 03/21/2014
-- Design Name:
-- Module Name:   Interp_max.vhd
-- Project Name:  Interp
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: FIR_Interpolation
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic
    and
-- std_logic_vector for the ports of the unit under test.  Xilinx
    recommends
-- that these types always be used for the top-level I/O of a design in
    order
-- to guarantee that the testbench will bind correctly to the post-
    implementation
-- simulation model.
```

```
--
    ------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.conv_signed;


ENTITY Interp_max IS
END Interp_max;

ARCHITECTURE behavior OF Interp_max IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT FIR_Interpolation
    PORT(
          clk : IN  std_logic;
          enb_1_1_1 : IN  std_logic;
          reset : IN  std_logic;
          FIR_Interpolation_in : IN  std_logic_vector(1 downto 0);
          coef_count : OUT  std_logic_vector(1 downto 0);
          int_SR : OUT  std_logic_vector(1 downto 0);
          int_reg1 : OUT  std_logic_vector(15 downto 0);
          int_reg2 : OUT  std_logic_vector(15 downto 0);
          int_reg3 : OUT  std_logic_vector(15 downto 0);
          int_reg4 : OUT  std_logic_vector(15 downto 0);
          FIR_Interpolation_out : OUT  std_logic_vector(15 downto 0)
        );
    END COMPONENT;


   --Inputs
```

```
signal clk : std_logic := '0';
signal enb_1_1_1 : std_logic := '0';
signal reset : std_logic := '0';
signal FIR_Interpolation_in : std_logic_vector(1 downto 0) := (others
    => '0');

    --Outputs
signal coef_count : std_logic_vector(1 downto 0);
signal int_SR : std_logic_vector(1 downto 0);
signal int_reg1 : std_logic_vector(15 downto 0);
signal int_reg2 : std_logic_vector(15 downto 0);
signal int_reg3 : std_logic_vector(15 downto 0);
signal int_reg4 : std_logic_vector(15 downto 0);
signal FIR_Interpolation_out : std_logic_vector(15 downto 0);

-- Clock period definitions
constant clk_period : time := 10 ns;

    TYPE input_type IS ARRAY (0 to 19) of integer;
    CONSTANT FIR_in : input_type := (-1, 1, 1, -1, 1, -1, -1, 1, -1,
        1,
                1, -1, 1, -1, -1, 1, -1, 1, 1, -1);
    TYPE output_type IS ARRAY (0 to 79) of integer;
    CONSTANT FIR_out : output_type := (-5, 41, 52, 13, -36, -101,
        -72, 35,
                129, 62, -80, -174, -135, 77, 244, 210, -6,
                    -275, -300, -30, 305,
                424, 100, -444, -713, -281, 608, 1261, 1000,
                    -312, -1814, -2287,
                -1007, 1625, 3908, 3919, 653, -5286, -11708,
                    -16024, -16359, -12275,
                -4468, 5497, 15635, 23819, 27770, 25794, 17642,
                    5133, -7768, -16475,
```

```
                        -17725, -11182, 100, 11162, 17317, 16164, 8076,
                            -4147, -16648, -25896,
                        -29340, -26029, -16777, -4046, 8296, 16256,
                            17141, 10773, -72, -10839,
                        -17048, -16181, -8324, 3913, 16637, 26106,
                            29584, 26106);
    CONSTANT delay : integer := 7;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
  uut: FIR_Interpolation PORT MAP (
        clk => clk,
        enb_1_1_1 => enb_1_1_1,
        reset => reset,
        FIR_Interpolation_in => FIR_Interpolation_in,
        coef_count => coef_count,
        int_SR => int_SR,
        int_reg1 => int_reg1,
        int_reg2 => int_reg2,
        int_reg3 => int_reg3,
        int_reg4 => int_reg4,
        FIR_Interpolation_out => FIR_Interpolation_out
      );

  -- Clock process definitions
  clk_process :process
  begin
              clk <= '0';
              wait for clk_period/2;
              clk <= '1';
              wait for clk_period/2;
  end process;
```

```
-- Stimulus process
stim_proc: process
begin
   -- hold reset state for 100 ns.
   wait for 100 ns;

   wait for clk_period*10;

                ------------ Input ------------------------------------

                FOR j IN FIR_in'RANGE LOOP

                    IF j = 0 THEN

                        reset <= '1';
                        enb_1_1_1 <= '1';
                        FIR_Interpolation_in <= std_logic_vector
                            (conv_signed(FIR_in(j),2));

                        wait for clk_period;

                        reset <= '0';

                        wait for clk_period*3.51;

                    ELSE

                        FIR_Interpolation_in <= std_logic_vector
                            (conv_signed(FIR_in(j),2));

                        wait for clk_period*4;
```

```
                END IF;

            END LOOP;

            FIR_Interpolation_in <= "00";

    wait;
end process;

    ---------- Output Verification ----------
    ouput_process: process
    begin

            wait until reset = '1';
            wait until reset = '0';

            wait for clk_period*delay;

            FOR j IN FIR_out'RANGE LOOP

                ASSERT FIR_Interpolation_out = std_logic_vector(
                    conv_signed(FIR_out(j),16))
                REPORT "Incorrect Output"
                SEVERITY ERROR;

                wait for clk_period;

            END LOOP;

            wait;
    end process;
```

```
END;
```

B.3.5 FIR Decimation Filter Impulse Test Testbench

```
--
--------------------------------------------------------------------------------

-- Company:
-- Engineer:
--
-- Create Date:   02:57:56 03/23/2014
-- Design Name:
-- Module Name:   Decim_ImpulseTB.vhd
-- Project Name:  Decim
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: FIR_Decimation
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic
    and
-- std_logic_vector for the ports of the unit under test.  Xilinx
    recommends
-- that these types always be used for the top-level I/O of a design in
    order
-- to guarantee that the testbench will bind correctly to the post-
    implementation
-- simulation model.
```

```
--
-- -------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.conv_signed;

ENTITY Decim_ImpulseTB IS
END Decim_ImpulseTB;

ARCHITECTURE behavior OF Decim_ImpulseTB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT FIR_Decimation
    PORT(
        clk : IN  std_logic;
        enb_1_1_1 : IN  std_logic;
        reset : IN  std_logic;
        FIR_Decimation_in : IN  std_logic_vector(15 downto 0);
        en_1 : OUT  std_logic;
        en_2 : OUT  std_logic;
        en_3 : OUT  std_logic;
        en_4 : OUT  std_logic;
        dec_SR_1 : OUT  std_logic_vector(15 downto 0);
        dec_SR_2 : OUT  std_logic_vector(15 downto 0);
        dec_SR_3 : OUT  std_logic_vector(15 downto 0);
        dec_SR_4 : OUT  std_logic_vector(15 downto 0);
        dec_reg1 : OUT  std_logic_vector(30 downto 0);
        dec_reg2 : OUT  std_logic_vector(32 downto 0);
        dec_reg3 : OUT  std_logic_vector(32 downto 0);
        dec_reg4 : OUT  std_logic_vector(32 downto 0);
        dec_reg5 : OUT  std_logic_vector(32 downto 0);
```

```
        dec_reg6 : OUT  std_logic_vector(32 downto 0);
        dec_reg7 : OUT  std_logic_vector(32 downto 0);
        FIR_Decimation_out : OUT  std_logic_vector(15 downto 0)
      );
END COMPONENT;


--Inputs
signal clk : std_logic := '0';
signal enb_1_1_1 : std_logic := '0';
signal reset : std_logic := '0';
signal FIR_Decimation_in : std_logic_vector(15 downto 0) := (others
   => '0');

     --Outputs
signal en_1 : std_logic;
signal en_2 : std_logic;
signal en_3 : std_logic;
signal en_4 : std_logic;
signal dec_SR_1 : std_logic_vector(15 downto 0);
signal dec_SR_2 : std_logic_vector(15 downto 0);
signal dec_SR_3 : std_logic_vector(15 downto 0);
signal dec_SR_4 : std_logic_vector(15 downto 0);
signal dec_reg1 : std_logic_vector(30 downto 0);
signal dec_reg2 : std_logic_vector(32 downto 0);
signal dec_reg3 : std_logic_vector(32 downto 0);
signal dec_reg4 : std_logic_vector(32 downto 0);
signal dec_reg5 : std_logic_vector(32 downto 0);
signal dec_reg6 : std_logic_vector(32 downto 0);
signal dec_reg7 : std_logic_vector(32 downto 0);
signal FIR_Decimation_out : std_logic_vector(15 downto 0);

-- Clock period definitions
```

```
constant clk_period : time := 10 ns;

    TYPE input_type IS ARRAY (0 to 102) of integer;
    CONSTANT FIR_in : input_type := (0, 0, 0, -5, 41, 52, 13, -36,
        -101,
                    -72, 35, 129, 62, -80, -174, -125, -5, 140, 184,
                        76, -155, -260,
                    -126, 149, 256, 92, -244, -455, -199, 400, 901,
                        784, -174, -1414,
                    -1969, -1067, 1131, 3460, 4049, 1381, -4616,
                        -11964, -17426,
                    -18001, -12537, -2328, 9227, 18245, 22305,
                        21342, 17620, 14248,
                    13431, 15156, 17633, 18703, 17346, 14656, 12934,
                         14063, 17896,
                    22112, 23211, 18608, 8364, -4300, -14559,
                        -18313, -14106, -3920,
                    8138, 17843, 22631, 22500, 19299, 15482, 13025,
                        12668, 14193,
                    16787, 19302, 20680, 19900, 16286, 9806, 1228,
                        -7982, -16188,
                    -21942, -24280, -22698, -17182, -8362, 2140,
                        11830, 18150, 19782,
                    17596, 14438, 13532, 16328, 21144, 23782);
    TYPE output_type IS ARRAY (0 to 44) of integer;
    CONSTANT FIR_out : output_type := (-1, -1, 0, 0, -2, 1, -1, -5,
        12, -24,
                    54, -46, -12, -37, -4, -59, 64, -49, -15, -18,
                        -16426, 16420,
                    16418, 16345, 16388, 16311, -16319, 16362,
                        16356, 16446, 16374,
                    -16539, -15952, 15496, 17770, 8776, -1866, 731,
                        -254, -10, 68,
```

```
                    -63, 52, -16, -31);
    CONSTANT delay : integer := 8;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
  uut: FIR_Decimation PORT MAP (
        clk => clk,
        enb_1_1_1 => enb_1_1_1,
        reset => reset,
        FIR_Decimation_in => FIR_Decimation_in,
        en_1 => en_1,
        en_2 => en_2,
        en_3 => en_3,
        en_4 => en_4,
        dec_SR_1 => dec_SR_1,
        dec_SR_2 => dec_SR_2,
        dec_SR_3 => dec_SR_3,
        dec_SR_4 => dec_SR_4,
        dec_reg1 => dec_reg1,
        dec_reg2 => dec_reg2,
        dec_reg3 => dec_reg3,
        dec_reg4 => dec_reg4,
        dec_reg5 => dec_reg5,
        dec_reg6 => dec_reg6,
        dec_reg7 => dec_reg7,
        FIR_Decimation_out => FIR_Decimation_out
      );

  -- Clock process definitions
  clk_process :process
  begin
            clk <= '0';
```

```
            wait for clk_period/2;
            clk <= '1';
            wait for clk_period/2;
end process;


-- Stimulus process
stim_proc: process
begin
   -- hold reset state for 100 ns.
   wait for 100 ns;

   wait for clk_period*10;

   ------------ Input -----------------------------------

            FOR j IN FIR_in'RANGE LOOP

                  IF j = 0 THEN
                        reset <= '1';
                        enb_1_1_1 <= '1';
                        FIR_Decimation_in <= std_logic_vector(
                            conv_signed(FIR_in(j),16));

                        wait for clk_period;

                        reset <= '0';

                        wait for clk_period*0.51;

                  ELSE

                        FIR_Decimation_in <= std_logic_vector(
```

```
                    conv_signed(FIR_in(j),16));

                wait for clk_period;

            END IF;

        END LOOP;

        FIR_Decimation_in <= (others => '0');

    wait;
end process;

    ---------- Output Verification ----------
    ouput_process: process
    begin

        wait until en_1 = '1';

        wait for clk_period*delay;

        wait for clk_period*2;

        FOR j IN FIR_out'RANGE LOOP

            ASSERT FIR_Decimation_out = std_logic_vector(
                conv_signed(FIR_out(j),16))
            REPORT "Incorrect Output"
            SEVERITY ERROR;

            wait for clk_period*4;

        END LOOP;
```

```
                wait;
        end process;

END;
```

B.3.6 BPSK Demodulator Testbench

```
--
    ----------------------------------------------------------------------------------


-- Company:
-- Engineer:
--
-- Create Date:    16:36:13 03/21/2014
-- Design Name:
-- Module Name:    BPSKDemodTB.vhd
-- Project Name:   BPSKDemod
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: BPSK_Demodulator_Baseband
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic
     and
-- std_logic_vector for the ports of the unit under test.  Xilinx
    recommends
-- that these types always be used for the top-level I/O of a design in
    order
-- to guarantee that the testbench will bind correctly to the post-
    implementation
-- simulation model.
```

```
--
    ------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY BPSKDemodTB IS
END BPSKDemodTB;

ARCHITECTURE behavior OF BPSKDemodTB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT BPSK_Demodulator_Baseband
    PORT(
         sym : IN  std_logic_vector(15 downto 0);
         bin : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
        signal clk : std_logic;
   signal sym : std_logic_vector(15 downto 0) := (others => '0');

        --Outputs
   signal bin : std_logic;

   constant clk_period : time := 10 ns;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
```

```
uut: BPSK_Demodulator_Baseband PORT MAP (
        sym => sym,
        bin => bin
      );

-- Clock process definitions
clk_process :process
begin
              clk <= '0';
              wait for clk_period/2;
              clk <= '1';
              wait for clk_period/2;
end process;


-- Stimulus process
stim_proc: process
begin
   -- hold reset state for 100 ns.
   wait for 100 ns;

   ------------------------------------------------

            -- sym: negative max
   wait for clk_period*10;
            wait until rising_edge(clk);

   sym <= (15 => '1', 0 => '0', others => '0');

            wait for clk_period;

            ASSERT bin /= '1'
                    REPORT "Output Correct"
                    SEVERITY NOTE;
```

```
        ------------------------------------------------
        -- sym: negative one
        wait for clk_period;

sym <= (15 => '1', 0 => '1', others => '1');

        wait for clk_period;

        ASSERT bin /= '1'
                REPORT "Output Correct"
                SEVERITY NOTE;

        ------------------------------------------------
        -- sym: zero
        wait for clk_period;

        sym <= (15 => '0', 0 => '0', others => '0');

        wait for clk_period;

        ASSERT bin /= '0'
                REPORT "Output Correct"
                SEVERITY NOTE;

        ------------------------------------------------
        -- sym: one
        wait for clk_period;

        sym <= (15 => '0', 0 => '1', others => '0');

        wait for clk_period;
```

```
                ASSERT bin /= '0'
                        REPORT "Output Correct"
                        SEVERITY NOTE;

                ------------------------------------------------

                -- sym: positive max
                wait for clk_period;

                sym <= (15 => '0', 0 => '1', others => '1');

                wait for clk_period;

                ASSERT bin /= '0'
                        REPORT "Output Correct"
                        SEVERITY NOTE;

                ------------------------------------------------

                wait for clk_period;

                sym <= (15 => '0', 0 => '0', others => '0');

        wait;
    end process;

END;
```

Appendix C

MATLAB BPSK System Model

```
clear
clc

%% Coefficients (Sub Filters)
ph1 = int16([5, 41, -83, 88, -25, -123, 348, -615, 869, -1052, 17504, ...
    -1052, 869, -615, 348, -123, -25, 88, -83, 41, 5]);
ph2 = int16([-41, 60, -43, -19, 112, -187, 163, 99, -901, 3897, 15453, ...
    -3256, 1542, -699, 210, 46, -132, 113, -48, -13, 0]);
ph3 = int16([-52, 20, 48, -124, 152, -48, -300, 1070, -2790, 10188, ...
    10188, -2790, 1070, -300, -48, 152, -124, 48, 20, -52, 0]);
ph4 = int16([-13, -48, 113, -132, 46, 210, -699, 1542, -3256, 15453, ...
    3897, -901, 99, 163, -187, 112, -19, -43, 60, -41, 0]);

%% Input Sequence and Plot Selection

% Input Selection
input = 1;
% 1 - PN Sequence
% 2 - Interpolation Filter Impulse
% 3 - Interpolation Max Value Test
```

```
% 4 - User Selected Input

% Number of delays in system
delay = 0;

% Number of repetitions of PN Sequence
pnSeqReps = 8;

% Subfilter to check for max value and output
maxValueSubFilter = ph3;

% User selected input entered here
userSelInput = [1 0 0 0 0 1 1 1 1 0 0 0 1 1 0];

plots = [1 1 1 1 1 1 1];
% Plots (on = 1; off = 0)
% 1 - Input Sequence Output
% 2 - Modulation Output
% 3 - Interpolation Filter Output
% 4 - Decimation Filter Output
% 5 - Demodulation Output


%% Data Source

seq = 0;

% PN Sequence Generator (LFSR)

reg = [0 0 0 0 0 1]; % Reset condition of LFSR
pn = [];
for i = 1:63
    pn = [pn reg(6)];
```

```
    reg = [xor(reg(5),reg(6)) reg(1:5)];
end

pn = int16(pn);
pnOut = repmat(pn,[1 pnSeqReps]);

if input == 1
    seq = pnOut;
end

% Check max value (24002, 27034, 29584, 27034)
if input == 3
    seq = maxValueSubFilter;
    seq = seq(seq~=0);
    seq = fliplr(seq);
    seq = sign(seq);

    seq = int16(-1.*(seq-1)./2);
end

%% BPSK Modulation
if input == 4
    seq = int16(userSelInput);
end

Mod = -2*seq+1;

if plots(1) == 1
    figure(1); stem(seq); title('Input Sequence')
end

if input == 2
    Mod = int16([1 zeros(1,41)]); %%% Impulse input
```

```
end

if plots(2) == 1
    figure(2); stem(Mod)
    title('Modulation')
end

Mod = [zeros(1,21) Mod];

%% FIR Interpolation Filter

Interp = [];
for delay = 1:(length(Mod)-21+1)
    reg = fliplr(Mod(delay:delay+21-1));
    temp1 = int16(sum(reg .* ph1));
    temp2 = int16(sum(reg .* ph2));
    temp3 = int16(sum(reg .* ph3));
    temp4 = int16(sum(reg .* ph4));
    Interp = [Interp temp1 temp2 temp3 temp4];
end

if plots(3) == 1;
    figure(3); stem(Interp)
    title('FIR␣Interpolation␣Filter')
end

% Delays
% Interp = [zeros(1,delay) zeros(1,4*20) Interp(1:end-delay)];


%% FIR Decimation Filter
Interp = int64(reshape([zeros(1,4*20) Interp],4,[]));
```

```
Decim = [];
for delay = 1:(size(Interp,2)-21+1)-1
    reg1 = int64(fliplr(Interp(1,delay+1:delay+21+1-1)));
    reg2 = int64(fliplr(Interp(2,delay:delay+21-1)));
    reg3 = int64(fliplr(Interp(3,delay:delay+21-1)));
    reg4 = int64(fliplr(Interp(4,delay:delay+21-1)));
    temp1 = sum(reg1 .* int64(ph1),'native');
    temp2 = sum(reg4 .* int64(ph2),'native');
    temp3 = sum(reg3 .* int64(ph3),'native');
    temp4 = sum(reg2 .* int64(ph4),'native');
    Decim = [Decim sum([temp1 temp2 temp3 temp4],'native')];
end

if plots(4) == 1
    figure(4); decimPlot(1) = subplot(2,1,1);
    stem(Decim); title('Decimation Filter Pre-Bit Reduction')
end

% Final Bit Reduction
Decim = int16(bitshift(Decim,-16,'int64'));

if plots(4) == 1
    figure(4); decimPlot(2) = subplot(2,1,2);
    stem(Decim); title('Decimation Filter Output')
    linkaxes([decimPlot(1) decimPlot(2)],'x')
end

%% Demodulation
Demod = -2*(Decim<0)+1;

if plots(5) == 1
    figure(5); demodPlot(1) = subplot(2,1,1);
    stem(Demod); title('Pre-Demodulation')
```

```
end

dataSink = 1*(Demod<0);
if plots(5) == 1
    figure(5); demodPlot(2) = subplot(2,1,2);
    stem(dataSink); title('Demodulation')
    linkaxes([demodPlot(1) demodPlot(2)],'x')
end
```

Bibliography

[1] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for FPGAs: From prototyping to deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, 2011.

[2] "HDL coder," http://www.mathworks.com/products/hdl-coder/, 2013, [Online; accessed 16-October-2013].

[3] "SystemVue electronic system-level ESL design software," http://www.home.agilent.com/en/pc-1297131/systemvue-electronic-system-level-esl-design-software, 2013, [Online; accessed 9-December-2013].

[4] "Vivado high-level synthesis," http://www.xilinx.com/products/design-tools/vivado/integration/esl-design/, 2013, [Online; accessed 15-Dec-2013].

[5] "Altera sdk for opencl," http://www.altera.com/products/software/opencl/opencl-index.html, 2013, [Online; accessed 7-Dec-2013].

[6] "System generator for DSP," http://www.xilinx.com/products/design-tools/vivado/integration/sysgen/index.htm, 2013, [Online; accessed 11-Dec-2013].

[7] "DSP builder," http://www.altera.com/products/software/products/dsp/dsp-builder.html, 2013, [Online; accessed 5-Dec-2013].

[8] K. Morris, "HLS versus OpenCL: Xilinx and altera square off on the future," *Electronic Engineering Journal*. [Online]. Available: http://www.eejournal.com/archives/articles/20130312-highlevel/

[9] P. Clarke, "Xilinx, ASIC vendors talk licensing," http://www.eetimes.com/document.asp?doc_id=1180867, 2001, [Online; accessed 20-November-2013].

[10] D. Bacon, R. Rabbah, and S. Shukla, "FPGA programming for the masses," *Queue*, vol. 11, no. 2, pp. 40:40–40:52, Feb. 2013. [Online]. Available: http://doi.acm.org/10.1145/2436696.2443836

[11] R. Sgandurra, "Understanding virtex FPGAs," *Embedded Technology*. [Online]. Available: http://www.pentek.com/CATFiles/virtexfpga.pdf

[12] "7 series dsp48e1 slice user guide," http://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf, 2013, [Online; accessed 6-November-2013].

[13] "XC2064/XC2018 logic cell array datasheet," http://www.datasheetarchive.com/XC2064-datasheet.html, 1985, [Online; accessed 23-October-2013].

[14] "Strategic considerations for emerging soc fpgas," White Paper, Altera, February 2011.

[15] "Zynq-7000 all programmable socs," http://www.xilinx.com/publications/prod_mktg/zynq7000/Zynq-7000-combined-product-table.pdf, 2013, [Online; accessed 22-Dec-2013].

[16] "Cyclone v fpga family overview," http://www.altera.com/devices/fpga/cyclone-v-fpgas/overview/cyv-overview.html, 2013, [Online; accessed 22-Dec-2013].

[17] "Arria v fpga family overview," http://www.altera.com/devices/fpga/arria-fpgas/arria-v/overview/arrv-overview.html, 2013, [Online; accessed 22-Dec-2013].

[18] "Arria 10 fpgas overview," http://www.altera.com/devices/fpga/arria-fpgas/arria10/overview/arr10-overview.html, 2013, [Online; accessed 22-Dec-2013].

[19] "Stratix 10 fpgas and socs: Delivering the unimaginable," http://www.altera.com/devices/fpga/stratix-fpgas/stratix10/stx10-index.jsp, 2013, [Online; accessed 22-Dec-2013].

[20] M. Goosman, N. Dorairaj, and Shiflet, "How to take advantage of partial reconfiguration in fpga designs," http://www.eetimes.com/document.asp?doc_id=1274489, 2006, [Online; accessed 29-November-2013].

[21] "Stratix v fpgas: Ultimate flexibility through partial and dynamic reconfiguration," http://www.altera.com/devices/fpga/stratix-fpgas/stratix-v/overview/partial-reconfiguration/stxv-part-reconfig.html, 2013, [Online; accessed 8-Dec-2013].

[22] "Partial reconfiguration in the ise design suite," http://www.xilinx.com/tools/partial-reconfiguration.htm, 2013, [Online; accessed 8-Dec-2013].

[23] "ISE design flow overview," http://www.xilinx.com/itp/xilinx10/isehelp/ise_c_fpga_design_flow_overview.htm, 2008, [Online; accessed 2-November-2013].

[24] "Systemverilog 3.1a language reference manual," http://www.eda.org/sv/SystemVerilog_3.1a.pdf, 2012, [Online; accessed 18-Dec-2013].

[25] "IEEE Standard for Standard SystemC Language Reference Manual," *IEEE Std. 1666-2011*, pp. i –22, 2000.

[26] A. Munshi, "The opencl specification," http://www.khronos.org/registry/cl/specs/opencl-1.0.48.pdf, 2009, [Online; accessed 4-December-2013].

[27] "Matlab - the language of technical computing," http://www.mathworks.com/products/matlab/, 2013, [Online; accessed 19-Dec-2013].

[28] "Python," https://www.python.org/, 2013, [Online; accessed 20-Dec-2013].

[29] "Simulink - simulation and model-based design," http://www.mathworks.com/products/simulink/, 2013, [Online; accessed 20-Dec-2013].

[30] "Stateflow - model and simulate decision logic using state machines and flow charts," http://www.mathworks.com/products/stateflow/, 2013, [Online; accessed 27-Dec-2013].

[31] M. Santarini, "Xilinx unveils vivado design suite for the next decade of "all programmable" devices," *Xcell Journal*, no. 79, 2012.

[32] "System generator for dsp user guide," http://www.xilinx.com/tools/sysgen.htm, 2013, [Online; accessed 23-November-2013].

[33] "System generator for dsp reference guide," http://www.xilinx.com/support/sw_manuals/sysgen_ref.pdf, 2008, [Online; accessed 23-November-2013].

[34] R. Huizen, "Opencl for fpga : bringing fpgas to the (relative) masses : Part 1," http://rtcmagazine.com/articles/view/103367, 2013, [Online; accessed 29-November-2013].

[35] "DSP builder," http://www.altera.com/technology/dsp/advanced-blockset/dsp-advanced-blockset.html, 2013, [Online; accessed 6-December-2013].

[36] "W1465bp systemvue system architect," http://www.keysight.com/en/pd-1454952-pn-W1465BP/systemvue-system-architect?nid=-34264.870753.00\&cc=DE\&lc=ger, 2013, [Online; accessed 4-Jun-2014].

[37] "W1462bp systemvue fpga architect," http://www.keysight.com/ru/pd-1454943-pn-W1462BP/systemvue-fpga-architect?\&cc=US\&lc=eng, 2013, [Online; accessed 4-Jun-2014].

[38] "W1461bp systemvue comms architect," http://www.keysight.com/en/pd-1454940-pn-W1461BP/systemvue-comms-architect?nid=-34264.804607.00\&cc=MX\&lc=spa, 2013, [Online; accessed 4-Jun-2014].

[39] "Liquid metal," http://researcher.watson.ibm.com/researcher/view_project.php?id=122, 2013, [Online; accessed 15-December-2013].

[40] "MyHDL," http://www.myhdl.org/doku.php, 2013, [Online; accessed 17-December-2013].

[41] "Dillon engineering company overview," http://www.dilloneng.com/, 2013, [Online; accessed 23-December-2013].

[42] "An independent evaluation of: The autoesl autopilot high-level synthesis tool," *Berkeley Design Technology*, 2010.

[43] N. Jachlmlec, "High-level synthesis tool delivers optimized packet engine design," *Xcell Journal*, no. 79, 2012.

[44] "Xilinx and its ecosystem expand all programmable abstractions to empower more designers and accelerate productivity up to 15x," http://press.xilinx.com/2013-09-10-Xilinx-and-its-Ecosystem-Expand-All-Programmable-Abstractions-to-Empower-More-De 2013, [Online; accessed 30-November-2013].

[45] "Implementing fpga design with the opencl standard," http://www.altera.com/literature/wp/wp-01173-opencl.pdf, 2013, [Online; accessed 4-December-2013].

[46] D. Chen and D. Singh, "Fractal video compression in opencl: An evaluation of cpus, gpus, and fpgas as acceleration platforms," in *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific*, 2013, pp. 297–304.