2011-04-27

Modular FPGA-Based Software Defined Radio for CubeSats

Steven J. Olivieri
Worcester Polytechnic Institute

Modular FPGA-Based Software Defined Radio for CubeSats

by

Steven J. Olivieri

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

May 2011

APPROVED:

Professor Alexander M. Wyglinski, Research Advisor

Professor Fred J. Looft III

Dr. R. Scott Erwin

Abstract

Digital communications devices designed with application-specific integrated circuit (ASIC) technology suffer from one very significant limitation—the integrated circuits are not programmable. Therefore, deploying a new algorithm or an updated standard requires new hardware. Field-programmable gate arrays (FPGAs) solve this problem by introducing what is essentially reconfigurable hardware. Thus, digital communications devices designed on FPGAs are capable of accommodating multiple communications protocols without the need to deploy new hardware, and can support new protocols in a matter of seconds. In addition, FPGAs provide a means to update systems that are physical difficult to access. For these reasons, FPGAs provide us with an ideal platform for implementing adaptive communications algorithms.

This thesis focuses on using FPGAs to implement an adaptive digital communications system. Using the Universal Software Radio Peripheral (USRP) as a base, this thesis aims to create a highly-adaptive, plug and play software-defined radio (SDR) that fits CubeSat form-factor satellites. Such a radio platform would enable CubeSat engineers to develop new satellites faster and with lower costs. This thesis presents a new system, the CubeSat SDR, that adapts the USRP platform to better suit the space and power limitations of a CubeSat.

Acknowledgements

First, I must thank my advisors, Drs. Alex Wyglinski and Fred Looft. Your advice and support for the past three years has been invaluable. I also thank Dr. Scott Erwin for being part of my defense committee.

COSMIAC and the Air Force Research Lab at Kirtland Air Force Base provided financial support for parts of this project. Thank you.

Thank you, Professors Jim Duckworth and Xinming Huang, for teaching me how to use FPGAs, VHDL, and Verilog. Without your instruction, I almost certainly would not have been able to complete this project.

The professors, students, tutors, and fellow teaching assistants who I have had the pleasure to work with have given me seven fantastic years at WPI. I shall especially thank Isaac, Drew, Zach, Ben, and Zebranky for getting me to play DotA whenever there was work to be done. Mike, Tom, Soe San, and the rest, I thank you for similarly distracting me in labs. Good times!

Finally, I thank my family and friends for putting up with my erratic schedule and for their support over the years. You all are awesome.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

1.1 Research Motivation

Traditionally, communication systems required specialized hardware to implement their functionality. In order to keep production costs low, these systems contained only the hardware necessary to perform the tasks that they were designed for. As a result, these systems were often difficult to modify and upgrade. Today, digital signal processor (DSP) technology has evolved to the point where many of the encoders, modulators, filters, and decoders used by these communication systems can be implemented in software[29]. Some systems rely on software to perform small tasks while others implement all of their baseband functionality in software. Those systems that implement all of their baseband functionality in software are called software-defined radios (SDRs) [42].

SDRs provide engineers with increased flexibility, allowing them to implement any number of different signal processing elements without altering the system hardware. For example, an engineer might design a transceiver with support for 16-symbol quadrature amplitude modulation (16-QAM) and later update the transceiver's software to support 32-QAM. With more traditional systems, such a change would require new modulation and demodulation hardware. In a SDR, this change might require changing only a few lines of C programming language code.

While some SDRs use general-purpose processors or even digital signal processors, many use a technology called field-programmable gate arrays (FPGA). FPGAs consist primary of reconfigurable logic elements and a switch matrix to route signals between them. These devices can be configured to support simple logic operations, such as addition, or more complex systems, such as digital signal filters. In addition, some FPGAs support a technology called dynamic reconfiguration. Dynamic reconfiguration allows a system to swap components as needed, without any reprogramming, as Figure 1.1 shows.

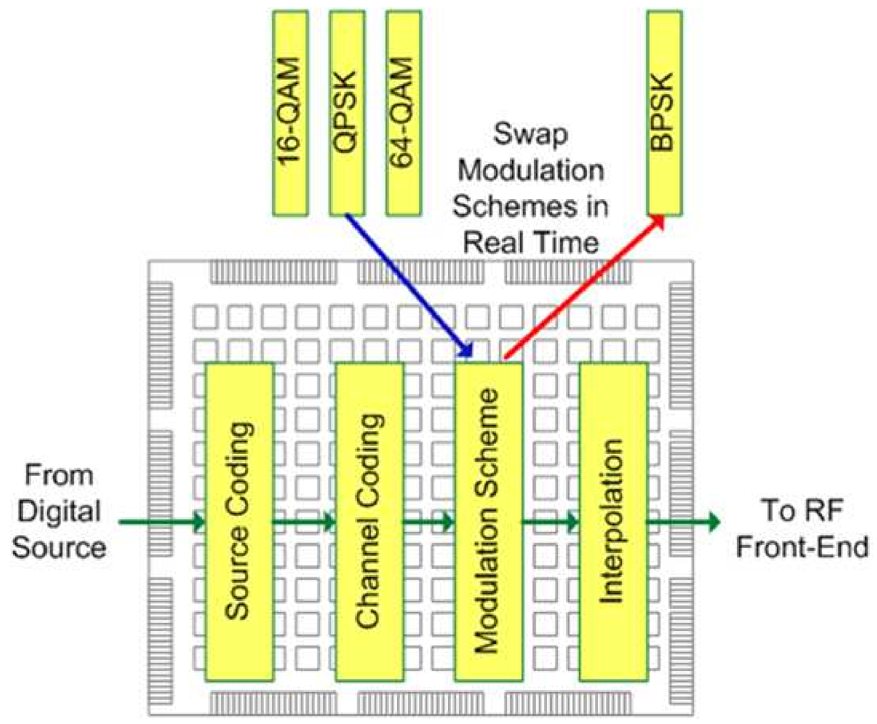


Figure 1.1: A FPGA swapping modulation schemes. The different modulation schemes are stored in memory or in the FPGA logic elements and connected as needed. While not shown here, the other components in a FPGA-based SDR could also be reconfigurable.

The combination of SDR and FPGAs provides engineers and developers with a method for updating communication systems that are physically difficult to access, such as satellites. Using FPGA technology, one can design a satellite to support multiple digital signal processing blocks (such as modulation or source coding) and to swap them as needed [19][29]. Additionally, the firmware on an FPGA can be updated remotely to install new blocks and remove unused blocks, enabling a system to support new protocols while in orbit. These features make FPGA-based SDRs an excellent platform for creating radios for satellites.

1.2 Current State of the Art

A wide variety of SDRs are available today, each optimized for different applications. Lyrtech's small form factor (SFF) SDR development platform incorporates advanced FPGA technology (Xilinx Virtex-4), a low-power general- purpose processor (TI MSP430), and multiple RF frontends to create an advanced development platform at the high cost of $9,900 [14]. Other platforms, such as the Wireless Open-Access Research Platform (WARP) from Rice University, provide similarly powerful, yet more affordable systems for academic use [20]. Finally, Ettus Research LLC produces the Universal Software Radio Peripheral (USRP) and the USRP2. These platforms contain less powerful hardware at a significantly lower cost [16]. Additionally, they use open source hardware and software licenses, making them ideal for academic environments.

Some SDRs are optimized for use in satellites. Vulcan Wireless Inc. has developed two such systems. The first, CubeSat Software Defined Radio (CSR-SDR), provides access to a wide variety of communications protocols and a data rate of up to 10 Mbps at S-Band [19]. The second, Micro Blackbox Transponder, offers fewer protocols and a lower data rate [19]. These two systems support numerous S-Band frequencies (2-4 GHz) and work with a variety of communication protocols and encryption schemes [34]. Finally, these systems use the Space Plug-and-Play Avionics (SPA) protocol for plug and play functionality and the CubeSat small satellite form-factor. Unfortunately, neither system uses open source hardware or software. Therefore, researchers and engineers cannot adapt these systems to their needs, nor can they create derivative solutions from them.

Another FPGA-based SDR optimized for the CubeSat form factor is the Firehose Adaptive Software Defined Radio, designed by Adaptive Radio Technologies LLC [39]. This system boasts features such as in flight reprogramming, and programmable transmit and receive paths. However, this SDR system does not support the SPA protocol for plug and play operation and, like the Vulcan Wireless systems, does not use open source hardware or software.

1.3 Research Objectives

The primary objective of this project is to develop an agile software-defined radio that can operate in a picosatellite known as a CubeSat [26]. Thus far, there is no standard communication protocol for CubeSats [31]. Each of the dozens of CubeSats successfully launched so far has used different baud rates, modulation schemes, and protocols. Many of the satellites use custom transceivers designed specifically for one satellite. As a result, researchers have found it difficult to create base stations that can communicate with a variety of CubeSats. Indeed, only 797 MB of data had been received from CubeSats by the end of 2008, five years after the initial CubeSat launch [31]. Employing SDRs in CubeSats could solve this problem and increase data throughput significantly.

The CubeSat SDR system should have the following characteristics:

- *Reconfigurability.* The system should be easily reconfigurable so that it can be used to support any number of encoding, modulation, or other signal processing schemes. In addition, it should support reprogramming post launch so that it can support new protocols.

- *Small size.* The system should fit the dimensions for a 1U CubeSat, which are $10 \times 10 \times 10$ cm, and should meet all other CubeSat standard requirements [26].

- *Plug and play.* The system should use the Space Plug-and-Play Avionics (SPA) communication protocol [33] to allow engineers to include it in new projects with ease.

- *Open Source.* The system should consist of open source hardware and software so that other researchers and system designers can adapt it to fit their projects and so that they can create derivative systems as needed.

- *Space Ready.* The system should not rely on any external processing elements that would not fit into a CubeSat. Addition, because CubeSats orbit in low Earth orbit, they require radiation hardened components. Finally, the CubeSat SDR should be able to transmit and receive data on frequency bands suitable for an object in orbit, such as S-Band.

1.4 Thesis Contributions

This thesis presents an initial platform for the system described in Section 1.3. Specifically, this thesis presents the following novel contributions:

- *A control system* for the Configurable Space Microsystem Innovations & Applications Center (COSMIAC) CubeSat FPGA board [6]. This hardware borrows ideas from the USRP and adapts them to operate in a CubeSat environment. The control system provides the glue that allows the FPGA, USB controller, memory devices, and RF frontend to communicate.

- *A rewrite of the USRP firmware* targeting Xilinx FPGA devices and using the VHSIC Hardware Description Language (VHDL). This new firmware is better than the original since it utilizes the safer VHDL language instead of Verilog, it contains more extensive source documentation, and it does not use any proprietary logic elements.

- *A Xilinx board definition file* that allows users to create new projects for the COSMIAC FPGA board using the Xilinx Embedded Developer's Kit (EDK). This definition file specifies the inputs, outputs, and configuration parameters for each device on the FPGA board.

1.5 Thesis Organization

This thesis is organized as follows: Chapter 2 describes the USRP platform, its companion software GNU Radio, and the CubeSat small form-factor satellite design. Chapter 3 presents diagrams, flow charts, and descriptions of the COSMIAC FPGA board and the new control system. Chapter 4 details the transition from the original USRP firmware to the new, improved firmware. This chapter also contains simulation and synthesis results to verify the functional equivalence of the new platform. Finally, Chapter 5 offers concluding remarks and directions for future research.

Chapter 2

Software-Defined Radio and CubeSats

This chapter provides an introduction to the technologies utilized in this thesis, including software-defined radio (SDR), a SDR suite called GNU Radio, and the CubeSat satellite platform. GNU Radio is ideal for low-cost, reconfigurable SDR because it is free, open-source software and it works well with relatively inexpensive RF hardware. Moreover, GNU Radio possesses a very large user community that is actively and constantly improving this software. CubeSats are small satellites in which one might use such a SDR system.

2.1 Software-Defined Radio

Traditional radios consist entirely of specialized hardware. While these radios might employ software to handle certain internal operations (e.g. control of the analog-to-digital and digital-to-analog components), almost all of the signal processing remains in the hardware domain. A software-defined radio replaces the majority of the traditional radio hardware with software. Figure 2.1 shows a the typical data flow in a software-defined radio system [41]. In this system, nearly all of the baseband signal processing on both the transmission and receiving ends is performed in the software domain.

Advances in microelectronic technology enable engineers to design smaller, more power

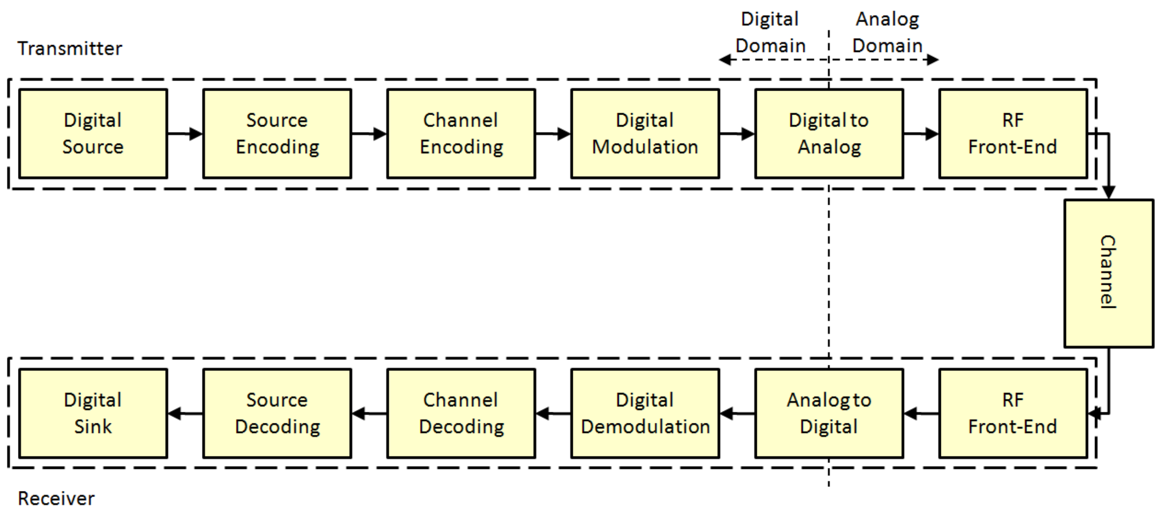


Figure 2.1: Block diagram of a typical software-defined radio. Everything in the digital domain is performed in software while everything in the analog domain remains hardware.

efficient devices. Consistent with Moore's Law [37], the number of transistors packed into a single integrated circuit has doubled approximately every two years [13]. SDR systems, which rely heavily on such integrated circuits, have followed a similar trend. The United States Military developed a software-defined radio system in the early 1990s that required four of the most powerful digital signal processors (DSP) available (the TMS320C40) and some FPGAs to glue them together. This system, known as SpeakEasy, filled the back of a truck [24]. By the time the engineers finished writing the software for SpeakEasy, three years had passed and the hardware in the system was already obsolete [24].

General purpose processors in the early 1990s had approximately 3.1 million transistors [13]. Today, the Intel Core i7 processors found in typical personal computers contain as many as 1,170 million transistors [11]. Similar advances in DSP and FPGA technology have enabled engineers to develop smaller SDR platforms. In addition, increased clock speeds [13] allow for faster than real-time processing of data. As a result, SDRs are capable of more complex coding and modulation schemes than they were in the past.

Performing the baseband signal processing in the software domain allows for much greater reconfigurability than traditional radios provide. In addition, the cost of a software-defined radio platform are often lower than those of a more traditional platform because the software is reusable and often available at relatively low cost. Indeed, some SDR software is available for free under the GNU General Public License (GPL) [10]. These two characteristics make SDR ideal for reusable, plug and play systems, for systems that might be difficult to physically reconfigure, and for systems developed in academic environments.

Software-defined radios have been employed on numerous platforms, including general purpose microprocessors (GPP), digital signal processors (DSP), and graphics processing units (GPU). General purpose microprocessors, such as the Intel and AMD devices commonly found in personal computers, are not specialized for any particular application. Therefore, they are very flexible. However, SDR systems using GPPs are often wasteful since these processors are designed for speed and generality rather than power efficiency or mathematical operations. Digital signal processors solve these two problems. DSPs, such as those manufactured by Texas Instruments, are specialized for performing mathematical operations and typically contain less hardware, increasing power efficiency. On the other hand, their narrow focus makes them potentially slow for other applications. Finally, graphics processing units employ massively parallel architectures that are optimized for vector manipulations and other graphical operations. Such parallel designs are well-suited for signal processing, but GPUs are still relatively difficult to program and they are extremely power hungry.

Table 2.1: Comparison of microelectronic platforms for SDR.

Feature	GPP	DSP	GPU	FPGA
DSP Operations	Moderate	Good	Good	Good
General Operations	Good	Poor	Poor	Poor
Flexibility	High	Low	Moderate	High
Size	Moderate	Small	Large	Large
Power Efficiency	Moderate	Good	Poor	Moderate
Common Brands	Intel, AMD	Texas Instruments	nVidia, AMD	Xilinx, Altera
Programming Language	C, Java	C, Assembly	CUDA, C	Verilog, VHDL

A fourth option is the field-programmable gate array (FPGA). FPGAs are reconfigurable logic devices that enable highly parallel implementations of digital signal processing algorithms. They are becoming increasingly power efficient while simultaneously including specialized hardware for some DSP operations. In additional, they can be reprogrammed remotely. Due to these features, FPGAs make an ideal software-defined radio platform. Table 2.1 compares these four platforms and Table 2.2 highlights some SDR implementations that use them.

Table 2.2: Some examples of SDR implementations.

System	Technology	Released
USRP1 [16]	Cyclone EP1C12 FPGA GPP (off-board)	2005
Lyrtech SFF SDR [12]	Virtex-4 FPGA MSP430 Microprocessor TI DM6446 DSP	2006
Berkeley BEE2 [27]	5 Virtex-II Pro FPGAs	2007
Kansas U. Agile Radio [36]	Virtex-II Pro FPGA Pentium-M Microprocessor	2007
Rice University WARP [20]	Virtex-II Pro FPGA	2008
USRP N210 [16]	Spartan3A-DSP3400 FPGA	2010

2.2 Field-Programmable Gate Arrays

FPGAs are reconfigurable digital logic devices. Like application specific integrated circuits (ASIC), they are often programmed with a hardware description language, such as Verilog or VHDL. However, unlike ASICs, FPGAs are designed to be reprogrammed multiple times to allow for rapid prototyping and system deployment. This technology makes them well-suited for SDR systems. While an FPGA-based SDR might not be as power efficient or as compact as a DSP-based system, it can be more flexible. DSPs have a set instruction set and run executable code in the same manner than general purpose processors do. This means that only a fixed number of operations can run simultaneously (usually one or two) and that computationally intensive functions can be slow to complete. On the other hand, FPGAs allow their users to define what the system is capable of and provide resources for massive parallelization.

An FPGA consists primarily of configurable logic blocks (CLB) and a configurable switch matrix that connects them together. Other features of modern FPGAs include configurable I/O Banks (IOB) that support multiple digital I/O standards, block memories (BRAM) for faster ROMs and RAMs, and specialized hardware for multiplication and other DSP functions. Figure 2.2 shows part of a FPGA fabric. The Spartan3A-1400 used in this project contains 25,344 CLBs, 589,824 bits of BRAM, and 375 user-accessible I/O pins [3]. Not shown in Figure 2.2 is the network of clock signals that synchronizes the various elements in the device. The Spartan3A-1400 does not contain any DSP-specific hardware, though it does contain thirty-two 18×18 multipliers, also not shown in Figure 2.2.

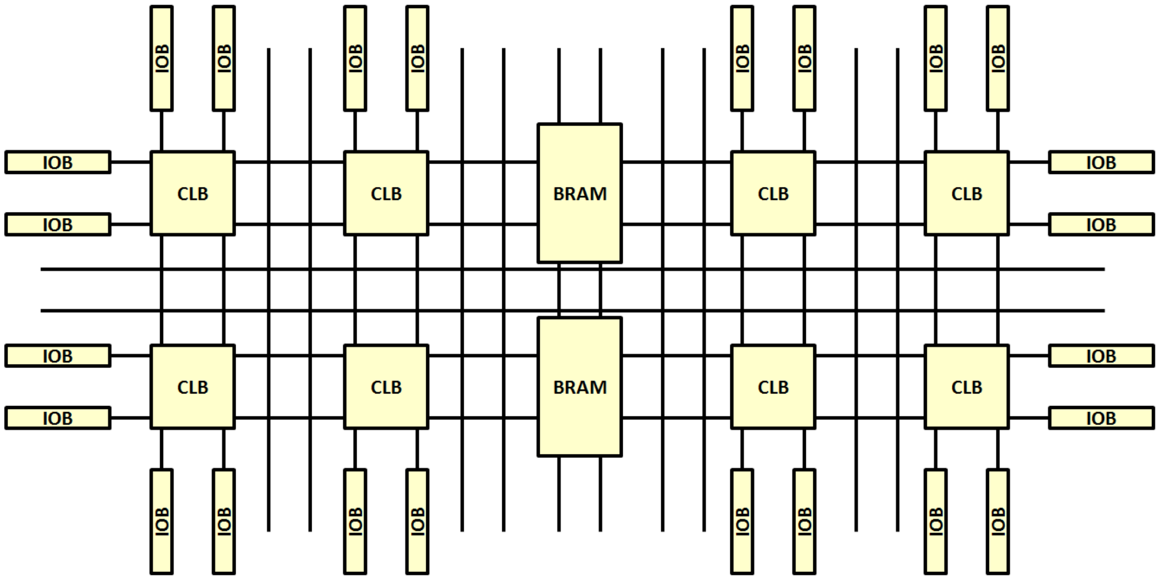


Figure 2.2: Illustration of a FPGA fabric. Each of the intersections can be programmed to connect different IOB, BRAM, and CLB elements together.

The design of each CLB varies from one FPGA to another. Figure 2.3 show a high-level illustration of the CLBs found on the Spartan3A-1400 device [2]. The CLBs in Xilinx FPGAs are called slices. Each slice in the Spartan3A contains two 4-input lookup tables (LUT) that are used to implement digital logic designs. For example, one LUT might be used to implement a 2-bit adder circuit. Though not shown in Figure 2.3, slices contain additional signals to support such operations as carry (for addition) and bit shifting. The output of each LUT is only one bit, making a LUT similar to a 16x1 memory.

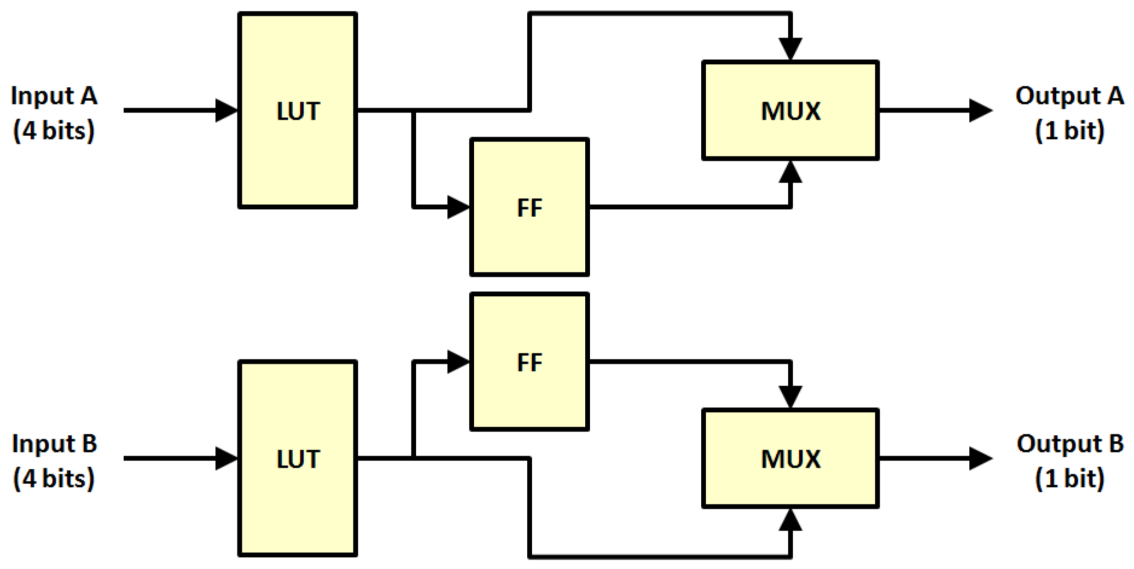


Figure 2.3: Illustration of a Spartan3A slice. Slices contain additional logic gates and signals that enable faster data paths, such as XOR gates and bypass signals.

Some digital logic designs are described as sequential because data moves through them in stages. These designs require synchronous components, such as flip flops. Each slice in the Spartan3A FPGA contains two such flip flops, synchronized with the system-wide clock network. A slice also contains two multiplexers, one for each LUT/FF pair. These multiplexers ensure that each slice has only two output bits, one for each LUT operation.

One particular software-defined radio system known as the Universal Software Radio Peripheral and its companion software, GNU Radio, combines FPGA technology with general purpose microprocessor technology. In this system, the FPGA performs interpolation and decimation while GNU Radio, running on a personal computer, performs all of the baseband signal processing.

2.3 GNU Radio and the USRP

The Universal Software Radio Peripheral Platform consists of the GNU Radio software, the libusrp software libraries, the USRP radio, and a wide variety of RF cards that attach to the USRP. Figure 2.4 shows the flow of data from the user to the point of transmission. Digital data first passes through GNU Radio, where it is compressed, encoded, modulated, and otherwise processed into its final baseband form. Next, it passes through libusrp, which forms the communication link between a personal computer and the USRP. The data then travels via USB2 to the USRP, which upconverts to an intermediate frequency (IF) band and sends the data through the digital to analog converter. Finally, the data arrives at the RF card where it begins its journey through the air. On the receiving end, the data takes the same path in reverse.

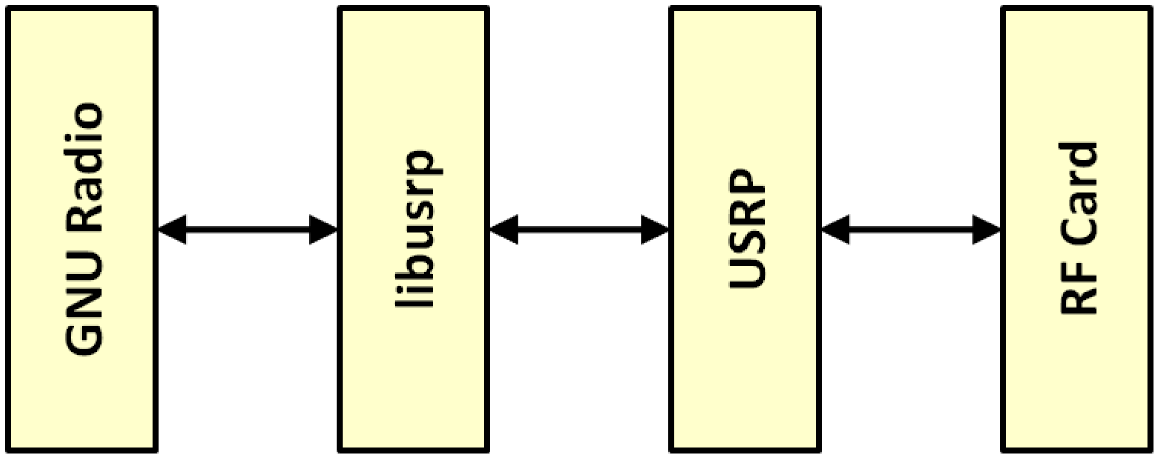


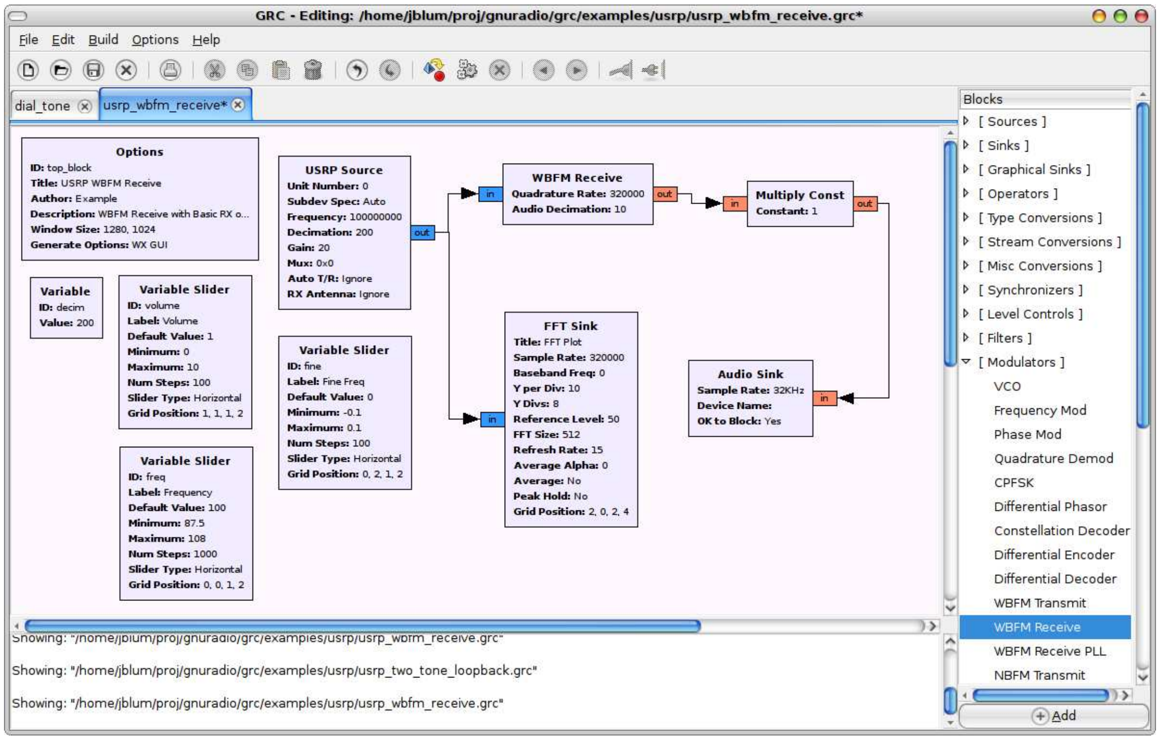
Figure 2.4: Data flow through the USRP platform. GNU Radio and libusrp reside on a general-purpose computer, while the USRP and its RF daughtercards are custom radio hardware.

2.3.1 GNU Radio

GNU Radio is a free, open source software project, licensed under the GNU General Public License 3 (GPL3), that provides users with tools to prepare data for wireless transmission and to receive data from wireless channels. While the GNU Radio software could work with any SDR hardware in theory, it is ideally suited for use with the USRP. GNU Radio runs on the Linux and Windows operating systems and utilizes general-purpose processors to perform digital signal processing tasks.

Applications for the GNU Radio platform use the Python scripting language to describe the flow of data through the system, creating flow graphs. These flow graphs connect data sources (e.g. files, microphones) to digital signal processing blocks (e.g. source encoders, modulators) and finally to data sinks (e.g. USRPs). Users can configure each of the blocks in a flow graph to change the way the system operates. For example, one might include a modulation block and choose between BPSK, QPSK, 16-QAM, or another modulation scheme. In addition, GNU Radio provides users with tools such as Fast Fourier Transform sinks to analyze data.

In an attempt to make GNU Radio easier to use, Josh Blum developed a graphical user interface (GUI) called GNU Radio Companion (GRC). GRC allows users to select premade blocks, adjust their variables with sliders and text fields, and connect them together with onscreen wires. Figure 2.5 shows a flow graph in GRC that receives wideband FM radio signals with a USRP, processes them, and then outputs them with a sound card [23].



Options
ID: top_block
Title: USRP WBFM Receive
Author: Example
Description: WBFM Receive with Basic RX o...
Window Size: 1280, 1024
Generate Options: WX GUI

Variable
ID: decim
Value: 200

Variable Slider
ID: volume
Label: Volume
Default Value: 1
Minimum: 0
Maximum: 10
Num Steps: 100
Slider Type: Horizontal
Grid Position: 1, 1, 1, 2

Variable Slider
ID: freq
Label: Frequency
Default Value: 100
Minimum: 87.5
Maximum: 108
Num Steps: 1000
Slider Type: Horizontal
Grid Position: 0, 0, 1, 2

USRP Source
Unit Number: 0
Subdev Spec: Auto
Frequency: 100000000
Decimation: 200
Gain: 20
Mux: 0x0
Auto T/R: Ignore
RX Antenna: Ignore

Variable Slider
ID: fine
Label: Fine Freq
Default Value: 0
Minimum: -0.1
Maximum: 0.1
Num Steps: 100
Slider Type: Horizontal
Grid Position: 0, 2, 1, 2

WBFM Receive
Quadrature Rate: 320000
Audio Decimation: 10

FFT Sink
Title: FFT Plot
Sample Rate: 320000
Baseband Freq: 0
Y per Div: 10
Y Divs: 8
Reference Level: 50
FFT Size: 512
Refresh Rate: 15
Average Alpha: 0
Average: No
Peak Hold: No
Grid Position: 2, 0, 2, 4

Multiply Const
Constant: 1

Audio Sink
Sample Rate: 32KHz
Device Name:
OK to Block: Yes

```
Showing: "/home/jblum/proj/gnuradio/grc/examples/usrp/usrp_wbfm_receive.grc"
Showing: "/home/jblum/proj/gnuradio/grc/examples/usrp/usrp_two_tone_loopback.grc"
Showing: "/home/jblum/proj/gnuradio/grc/examples/usrp/usrp_wbfm_receive.grc"
```

Figure 2.5: A GNU Radio Companion flow graph for a wideband FM receiver. The arrows indicate the flow of data from the USRP to the sound card [23].

The flow graph shown in Figure 2.5 is stored as an XML file. When the user "runs" the flow graph shown in Figure 2.5, GNU Radio parses the XML file and uses it to create Python code that performs the signal processing operations described by the blocks in the flow graph. In this example, GNU Radio would generate a block to process wideband FM data as well as the code needed to link this block to the USRP and the system's sound card. In addition, it would generate code for the blocks on the left side of Figure 2.5, which allow the user to adjust the frequency band to listen on and the volume of the audio output. Finally, it would generate a real time plot of the fast Fourier transform (FFT) of the incoming data.

Data received on the USRP's antenna would go through the ADC, which translates it to an intermediate frequency, then pass through the USRP itself, which translates the data into baseband. The USRP then provides this data to the USB2 port, where libusrp picks it up in packets and passes it to GNU Radio. The USRP Source block in the flow graph lets GNU Radio know to expect data of this format. When GNU Radio receives the data, it passes it through the code blocks that it generated and then sends this processed data to the system's sound card, where it passes through a DAC and finally to the speaker output port. While this is a simple example, GNU Radio supports more complex systems, including transceivers.

2.3.2 USRP

Like GNU Radio, the USRP is entirely open source. The FPGA software is licensed under the GPL2 and the hardware schematics are all available online from Ettus Research [18]. Together, GNU Radio and the USRP provide a low-cost, open software-defined radio platform that permits experimentation and modification. For this reason, this thesis uses the USRP platform as the basis for the new CubeSat SDR.

The USRP consists of a large motherboard, which contains an Altera Cyclone FPGA, a Cypress EZ-USB FX2 high-speed USB2 controller, two AD9862 mixed signal processors from Analog Devices, and four I/O headers for RF daughtercards. Transmission data enters the USRP via the USB2 controller, proceeds to the FPGA for interpolation, goes through the AD9862's digital-to-analog converter (DAC), and then finally makes its way to the RF daughtercard for wireless transmission. The RF daughtercards contain the antenna and supporting hardware. Figure 2.6 [32] shows a USRP board with three daughtercards attached and Figure 2.7 [41] shows a block diagram of the USRP system.

Ettus Research also makes another SDR platform, the USRP2. This platform offers a

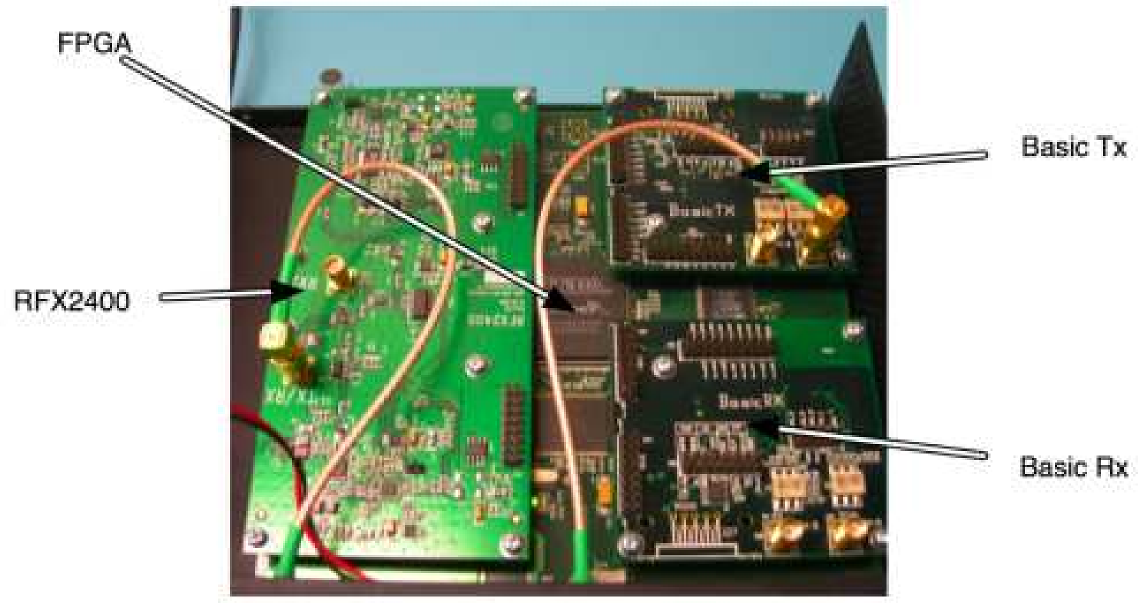


Figure 2.6: A USRP and RF daughtercards. The RFX2400 card contains one Tx RF frontend and one Rx RF frontend. The USRP supports up to four total RF frontends simultaneously with a maximum of two Tx and four Rx [32].

larger FPGA, 1 Gbps Ethernet connectivity, faster and more accurate ADCs and DACs, and greater RF bandwidth [16]. However, the cost of the USRP2 is approximately twice the cost of the USRP1 and the engineers at COSMIAC were more familiar with the USRP1 platform. For these reasons, this thesis uses the USRP1 as the basis for the initial CubeSat SDR prototype.

The USRP exchanges data with the GNU Radio software via USB2. Data prepared for transmission first passes through a USB2 controller and then enters the Tx FIFO, which holds 4,096 lines of 16-bit data. From there, the data passes through the Tx Chain (see Figure 2.9) and then to the AD9862, which converts it to analog signals suitable for RF transmission and passes it along to the RF card. Data received on the RF card first enters the AD9862, which converts it to digital signals, then proceeds to the Rx Chain (see Figure 2.10). Next, this data enters the Rx FIFO ($4,096 \times 16$) while it waits for the USB2 controller to send it through to the GNU Radio software [30]. Figure 2.8 illustrates this process.

As described in Section 2.3, the GNU Radio software handles all of the baseband signal

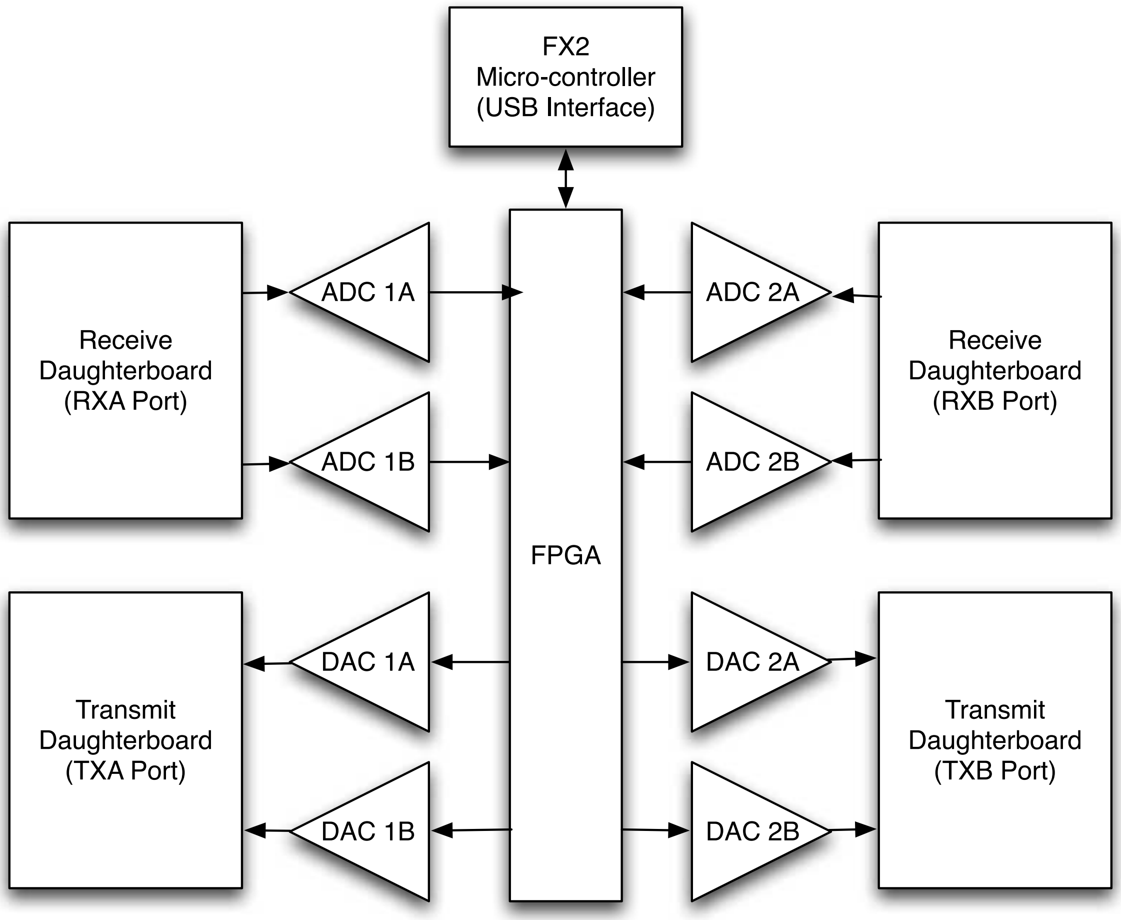


Figure 2.7: Block diagram of the USRP. This particular USRP has two Tx boards and two Rx boards, but the system could support four Rx boards instead [41].

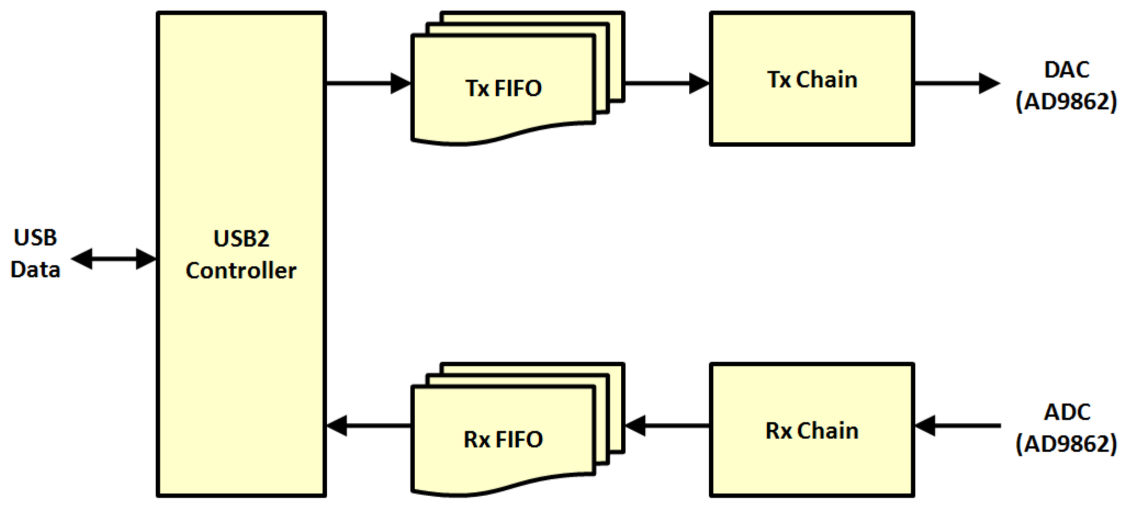


Figure 2.8: Data flow through the USRP. GNU Radio passes data to and from the USRP via the USB2 controller and the USRP handles the remaining signal processing.

processing for outgoing data. The role of the USRP hardware is to transform these baseband signals into an intermediate frequency (IF) band and then present the result to the Analog Devices AD9862 mixed signal processor, which prepares it for the RF daughtercard. The USRP supports two complex transmitters and so does the new firmware. However, the CubeSat SDR uses only one complex transmitter. Therefore, the figures in this chapter will display only one transmission path. Figure 2.9 shows the transmission data path through the USRP (the “Tx Chain” in Figure 2.8).

Each of the two signal components (I and Q) passes through a 4-stage cascaded integrator-comb (CIC) interpolator, which increases the number of data samples. This is necessary because the USB2 controller runs at a slower speed than the AD9862 requires. The rest of the upconversion process happens in the AD9862 mixed signal processor.

Similarly, GNU Radio handles the baseband processing for incoming data. Thus, the role of the USRP is to receive digital data from the AD9862 processor and then downconvert that data to baseband for GNU Radio. The USRP supports two full receivers or four receivers without the halfband filters. Because the CubeSat SDR platform uses only one complex receiver, the figures in this chapter will display only one receive path. Figure 2.10 shows the receive data path through the USRP (the “Rx Chain” in Figure 2.8).

Signals *IN A* and *IN B*, from the AD9862 DACs, first proceed through a complex

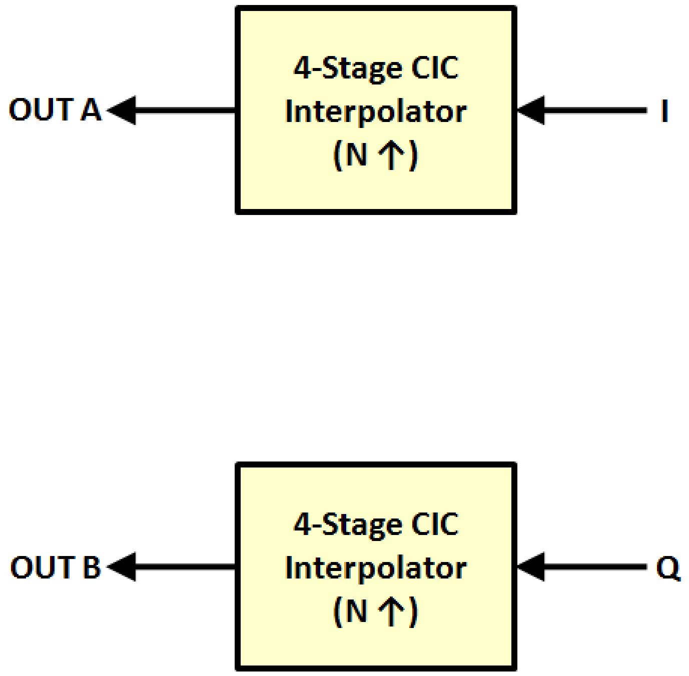

OUT A
4-Stage CIC Interpolator (N ↑)
I
OUT B
4-Stage CIC Interpolator (N ↑)
Q

Figure 2.9: Tx data flow through the USRP. The remainder of the digital upconversion process is handled by the AD9862.

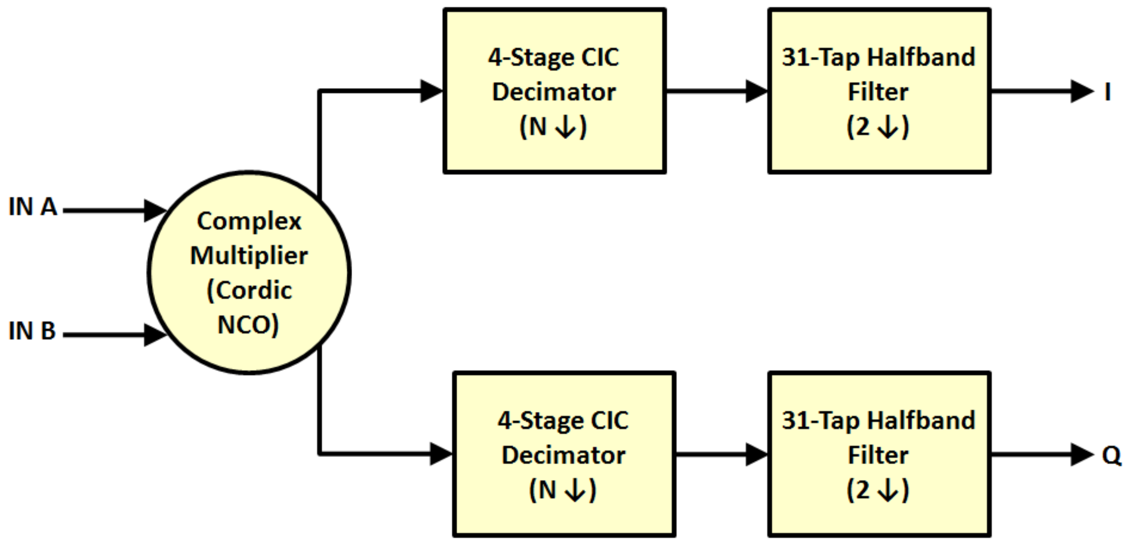

IN A
IN B
Complex Multiplier (Cordic NCO)
4-Stage CIC Decimator (N ↓)
31-Tap Halfband Filter (2 ↓)
I
4-Stage CIC Decimator (N ↓)
31-Tap Halfband Filter (2 ↓)
Q

Figure 2.10: Rx data flow through the USRP. The AD9862 does not provide a Cordic NCO on the receive path, so the FPGA does the conversion from IF band to baseband internally.

multiplier where the other two inputs are generated with a Cordic Numerically Controlled Oscillator (NCO), which transforms the signal from its IF band to baseband. From there, the signal passes through a 4-stage CIC decimator, which samples the incoming data, and then a halfband filter which effectively multiplies the decimation rate by two. These filters are needed to reduce the data rate to something that the USB2 controller can sustain.

2.4 CubeSats

A CubeSat is a small form-factor satellite conforming to the CubeSat specifications published by California Polytechnic State University [26]. CubeSats are, as their name implies, cube shaped (see Figure 2.11 [8]). Each side is 10 cm, giving them a volume of precisely one liter, and the total weight of a CubeSat cannot exceed 1.33 kg. In addition to this 1U configuration, the CubeSat specifications allow for 2U ($10 \times 10 \times 20$ cm) and 3U ($10 \times 10 \times 30$ cm) configurations. The CubeSat SDR was designed to fit into a 1U CubeSat, though it is ideally utilized in a 2U or 3U configuration so that users can the include sensors and other instruments necessary for their satellite's functionality.

The majority of CubeSats are developed at academic institutions in countries all over the world [28]. These CubeSats contain CMOS cameras [17], gamma ray detectors [4], GPS receivers [5], and other scientific instruments and devices. A number of non-academic institutions have also designed and deployed CubeSats, including NASA [9] and The Boeing Company [25]. Most of the hardware in these systems consists of is commercially available off-the-shelf (COTS) components, making CubeSats an affordable way to perform experiments in space.

However, the majority of CubeSats do not contain interchangeable parts. Instead, they are designed like "Swiss watches", using different sensors, different power systems, and different radios [35]. To counter similar problems in other satellites, the Air Force Research Laboratory's Space Vehicle Directorate (AFRL/RV) developed a system that introduces the concept of plug and play components to satellites.

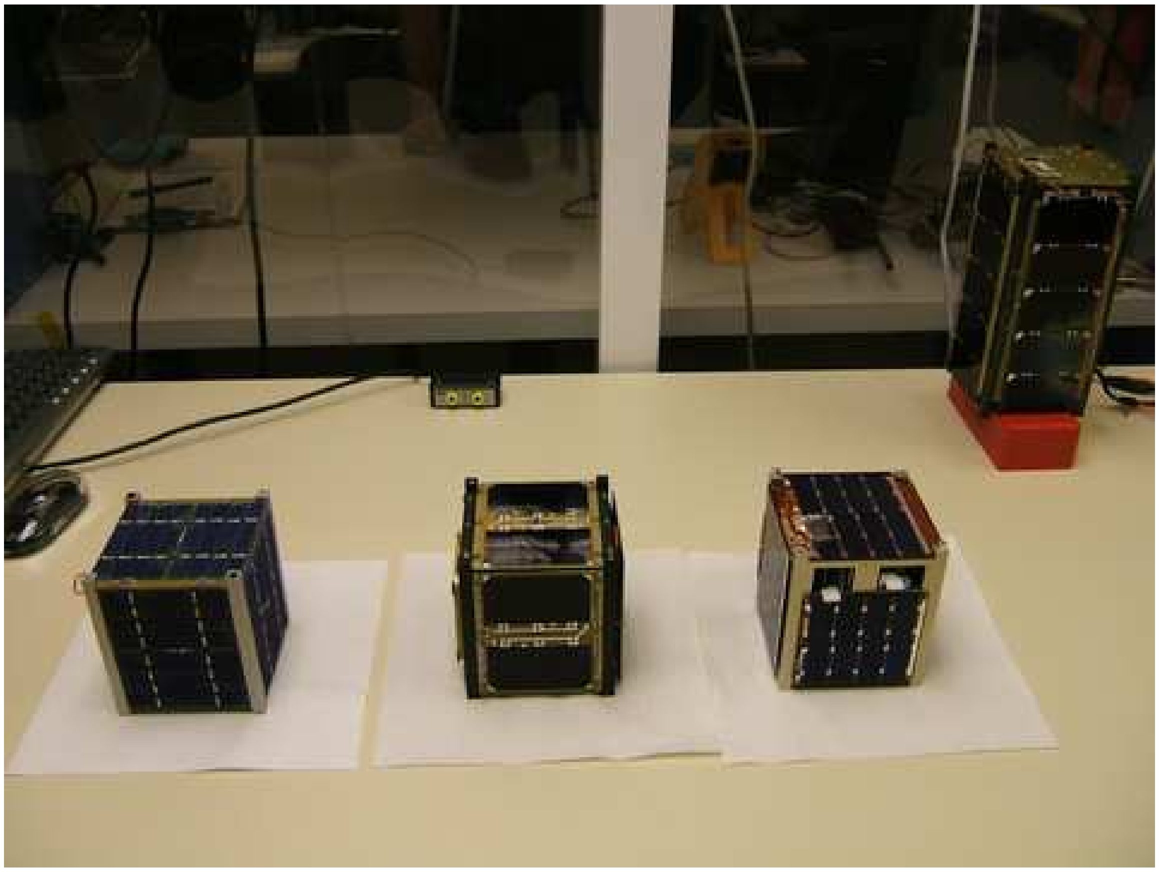


Figure 2.11: A photograph of three CubeSats at California Polytechnic State University. These satellites are ready for integration into the launcher system, called P-POD [8].

2.4.1 Space Plug-and-Play Avionics (SPA)

Space Play-and-Play Avionics (SPA) is a set of standards that combines common commercial standards, such as USB and Ethernet, with hardware and software extensions to create communications busses appropriate for use in modern, real-time embedded systems [33]. SPA treats each device in a satellite as a "black box". What distinguishes SPA from other attempts at such a system is that each "black box" is self-describing [33].

This system comprises three primary components: XTEDs, ASIMs, and the SDM. XTEDs, or eXtended Transducer Electronic Datasheets, are XML documents that describe the inputs, outputs, and control variables for a component. Figure 2.12 illustrates the conceptual organization of XTEDs. These documents reside on ASIMs, or appliqué sensor interface modules. ASIMs function similarly to USB interface chips [35]. They encapsulate the XTEDs that describe the devices they are attached to and provide a SPA interface that allows their client device to communicate with the system. Finally, the SDM (satellite data model) acts as the overall system controller. Its main responsibilities including registering SPA devices and managing subscriptions to those devices [33] [35].

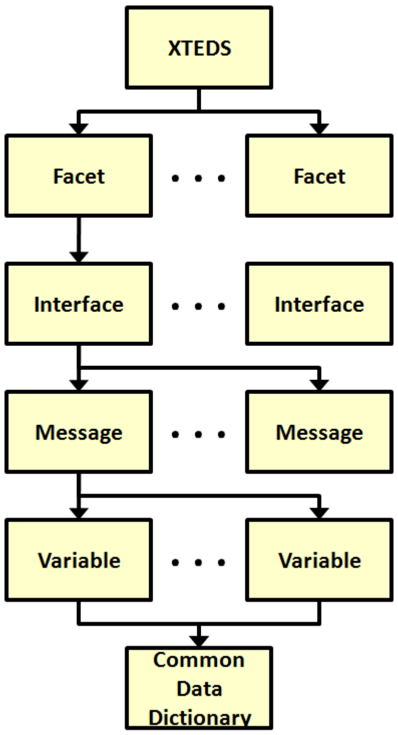


Figure 2.12: Organization of an XTEDs. Like any other XML document, an XTEDs combines user-defined properties and values to describe something. XTEDs are composed of "atoms" drawn from a common data dictionary (CCD). These atoms compose variables, variables compose message, and so on to form an XTEDs document [35].

SPA supports a number of different interfaces, including SPA-U (USB-based), SPA-E (Ethernet-based), SPA-1 (I2C-based), and SPA-S (Spacewire-based). The SPA-U interface is based on the USB 1.1 standard, which provides a 12 Mbps data bus, suitable for most satellite devices (e.g. sensors). As with USB, SPA-U uses the host, endpoint, and hub approach to connect devices together. Unlike USB, SPA-U hubs dynamically determine the direction of a connection. This allows for more complex topologies since any host or endpoint can connect to any port [33].

Devices requiring higher data rates can use the SPA-S standard. SPA-S combines a Spacewire interface with a SPA-U connection. Spacewire is a European Space Agency (ESA) standard [15] that uses a peer-to-peer networking approach. Therefore, SPA-S does not have any hosts. Instead, SPA-S devices connect to a Spacewire router so that they can communicate with each other and to a SPA-U hub so that they can communicate with the rest of the system. Spacewire supports data rates as high as 200 Mbps.

AFRL/RV adapted the SPA design to work with CubeSats in a system called CubeFlow. The CubeFlow system details the power and mechanical requirements for using SPA devices in a CubeSat. With such a system, plug and play CubeSats become easier to develop. One component well-suited to SPA and CubeFlow is the radio. Specifically, a SPA-compatible software-defined radio would provide a common solution for CubeSats that was capable of adapting to the communication needs of those satellites.

2.5 Chapter Summary

Each of the CubeSats described in this chapter used different radio hardware. The goal of this thesis is to ease the development costs of future CubeSat designs by providing a flexible, plug and play radio solution. Software-defined radio systems, SPA, and CubeFlow provide us with a cost-effective way to create such a system. One particular software-defined radio, the Universal Software Radio Peripheral Platform (USRP and GNU Radio), is available at a low cost and consists entirely of open source software and hardware. These characteristics make it an ideal base from which to build the CubeSat SDR platform.

Chapter 3

COSMIAC CubeSat SDR System

Engineers at COSMIAC adapted the ideas of the USRP hardware to create a new FPGA board for the CubeSat SDR system. This board accepts daughtercards and an external power board, all of which fit into a 1U CubeSat. This chapter presents the full CubeSat SDR system, including the FPGA board and additional FPGA firmware designs.

3.1 Hardware

Like the original USRP, the CubeSat SDR system uses two boards. The first board contains the majority of the system functionality, including the USB2 controller and FPGA. The daughtercard contains the AD9862 and the RF frontend. Using this design, the CubeSat SDR system can support any number of RF frontends. However, the CubeSat SDR hardware contains a number of significant changes from the original USRP hardware.

Figure 3.1 shows a block diagram of the primary FPGA board [38]. The largest change to this hardware is the new Spartan3A-1400 FPGA, which replaces the Cyclone FPGA present on the USRP. For more information about this change, please refer to Chapter 4. The new FPGA provides more logic elements and I/O pins, which enabled the engineers at COSMIAC to add additional peripherals to the FPGA board. These peripherals include:

- *NAND Flash Memory.* The flash memory enables the system to store data through power cycles and to retain data without using power to keep volatile memory active in a powered down state. Power can be difficult to come by in space.

- *DDR2 Memory.* This volatile memory can act as a cache for any softcore processors on the FPGA, enabling better performance.

- *NOR Configuration Flash Memory.* The NOR flash memory stores the FPGA configuration so that the system can reboot itself without attaching to a JTAG cable.

- *AT90 Microcontoller and SPA Connectors.* The AT90 microcontroller connects to the FPGA through three PIO ports. Its primary purpose is to enable plug and play functionality via the Space Plug-and-Play Avionics protocol. The AT90 holds configuration information for the FPGA board and provides this information to the primary operating system, which enables communication between different devices in a single CubeSat. The SPA-U connector works similar to USB and the SPA-1 connector like I2C.

- *Ethernet Controller.* The FPGA board supports a variety of daughtercards, some of which might communicate via Ethernet instead of USB. The CubeSat SDR system does not use this functionality.

- *LEDs and Switches.* The FPGA board contains eight LEDs and a pushbutton to aid with debugging.

CubeSats are 10×10 cm on a side, limiting the space available for the FPGA board (see Figure 3.2 [38] and Figure 3.3 [7]). As a result, the Cypress FX2 EZ-USB2 controller present on the original USRP is replaced by a USB3300 USB2 controller on the CubeSat SDR board. The USB3300 offers much less functionality than the FX2. Section 3.2 describes the effects of this change in detail.

The RF daughtercard mirrors the XCVR 2450 [16] daughtercard used with the original USRP. This card operates in the 2.4 GHz industrial, scientific, and medical (ISM) band, making it ideal for testing on Earth. Due to space limitations, the AD9862 mixed-signal processor and its associated circuitry were moved from the primary FPGA board to the RF daughtercard. This change makes RF daughtercards more expensive, but allows for other

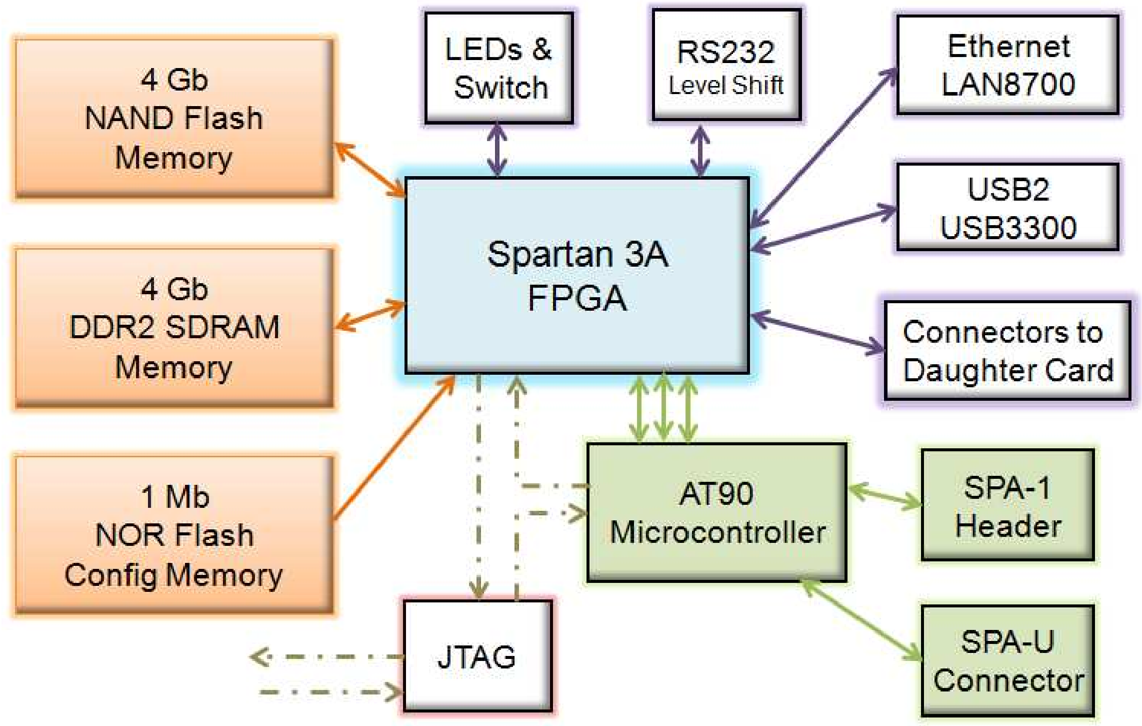


Figure 3.1: Block diagram of the COSMIAC FPGA board. Note the lack of AD9862 processors, which reside instead on the RF daughtercards (see Figure 3.4. This design choice enables CubeSat engineers to design non-SDR systems with the same primary FPGA board [38].

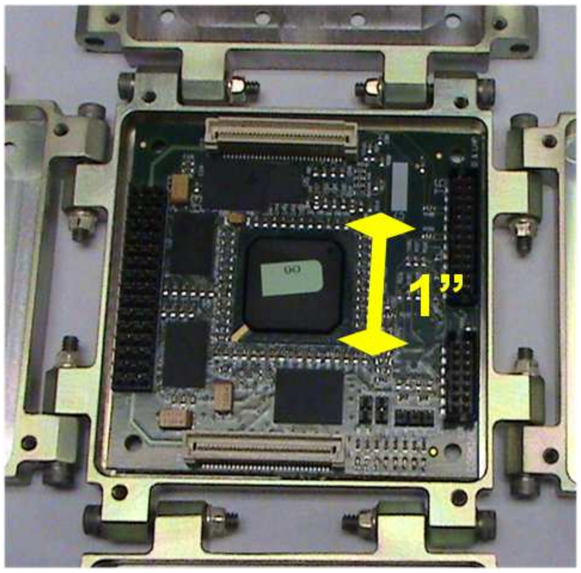


Figure 3.2: A photograph of the COSMIAC FPGA board. The board sits on one side of the CubeSat enclosure and four others flank it. The RF card attaches to the two white connectors on the top and bottom of the board [38].

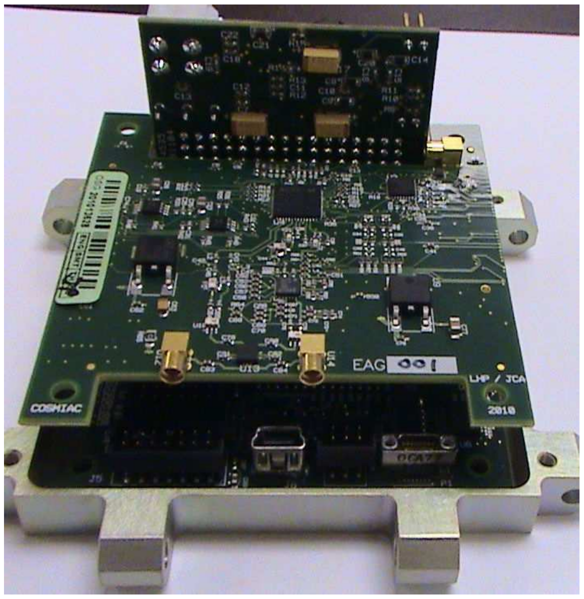


Figure 3.3: A photograph of the COSMIAC CubeSat SDR system. The bottom board is the FPGA board while the one on top is the RF daughtercard. The perpendicular board in the back provides power to the system. All three boards fit into a 1U CubeSat [7].

peripherals on the primary FPGA board and lowers the cost for systems that do not use a RF daughtercard. Figure 3.4 shows a block diagram of the RF daughtercard currently used in the CubeSat SDR system [38].

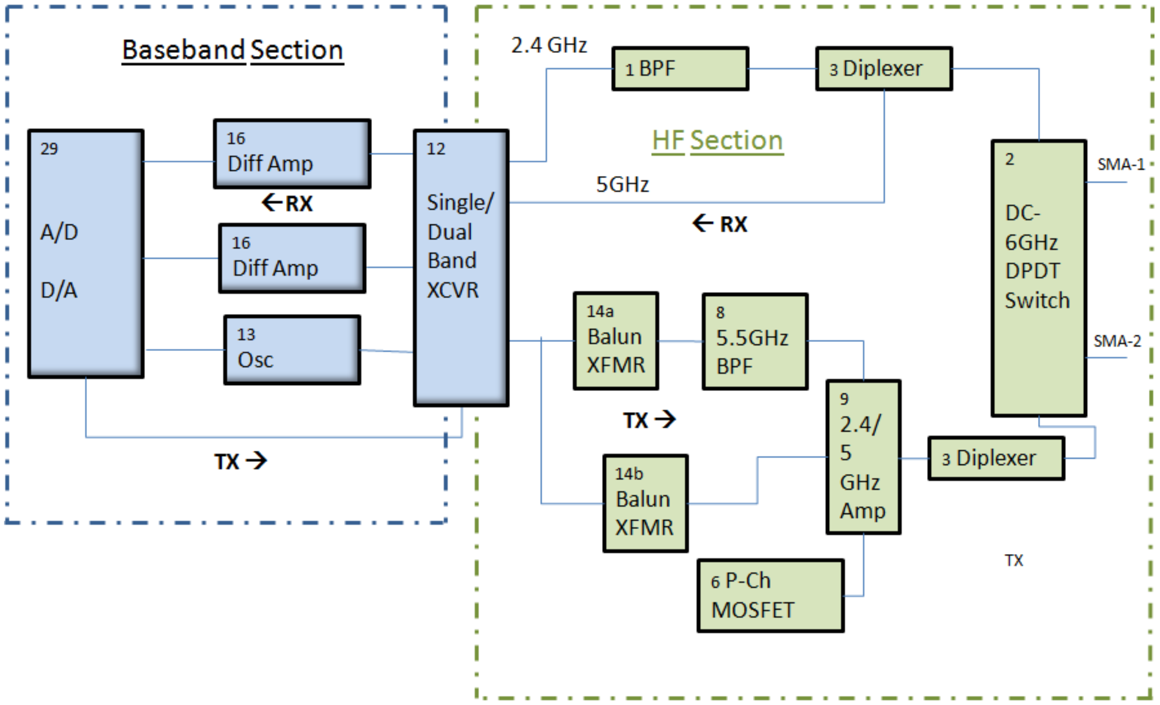


Figure 3.4: Block diagram of the COSMIAC RF daughtercard. The AD9862 mixed-signal processors were moved to the daughtercard to make room for other devices on the primary FPGA board [38].

3.2 Software

With the new hardware components described in Section 3.1, the software in the CubeSat SDR system requires more functionality. Specifically, the loss of the FX2 USB2 controller requires that the FPGA handle all of the data coming into and leaving the system. The FX2 USB2 controller contained a modern 8051 microprocessor, a full USB2 controller with direct memory access (DMA), RS232 controllers, and controllers for the SPI and I2C buses. The USB3300 chip that replaced it on the CubeSat SDR system contains only a partial USB2 controller.

Figure 3.5 shows a block diagram of the proposed CubeSat SDR system design. At the

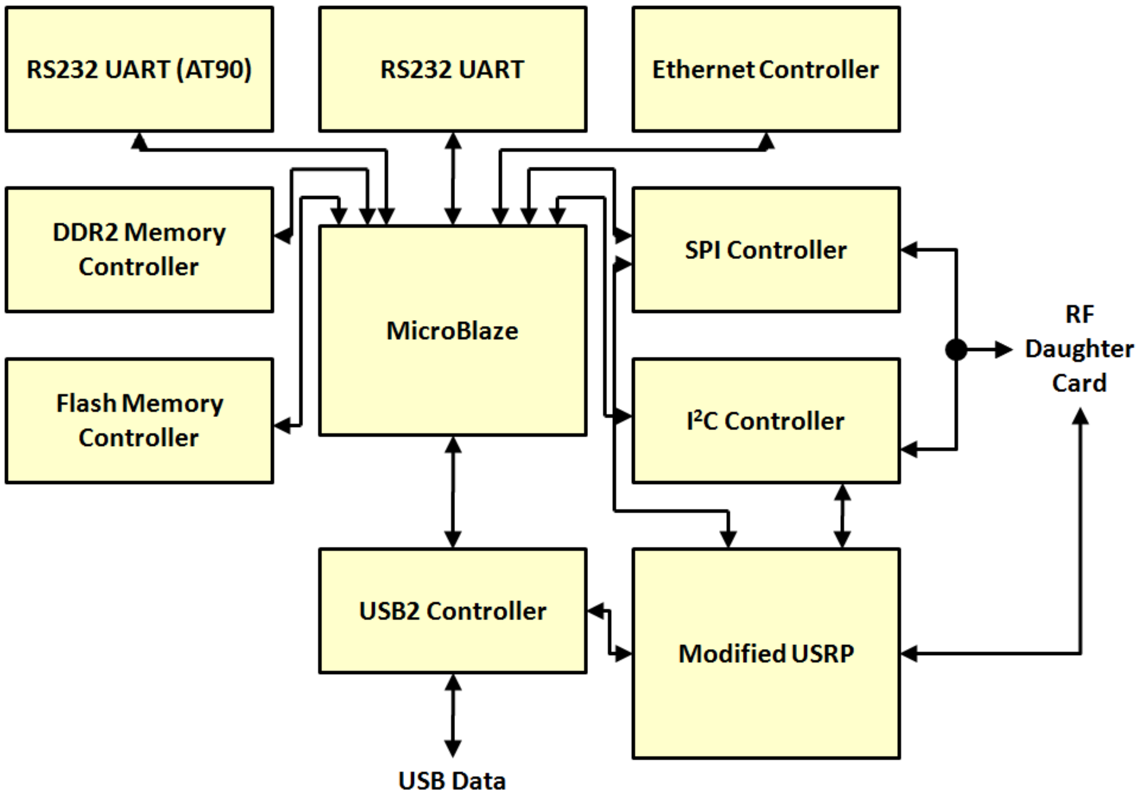


Figure 3.5: Block diagram of the proposed CubeSat SDR firmware. The MicroBlaze processor acts as the central control for all of the various communications blocks.

center of the system is the MicroBlaze softcore microprocessor. This processor acts as a traffic mediator, guiding data between the USB2 port to the various other devices in the system. The SPI and I2C controllers provide users with a means of sending configuration information to the USRP and to the RF daughtercards. One of the RS232 controllers connects to an serial port on the board; the other connects to the AT90, which controls the ASIM for the device. The other components simply control the peripherals that give them their names (e.g. the DDR2 Memory Controller controls the DDR2 memory). Not shown in Figure 3.5 are the general purpose I/O modules for the LEDs.

This design has many advantages. First, it uses well-tested components from Xilinx including the DDR2 memory controller, the SPI and I2C controllers, and the MicroBlaze microprocessor. Second, the Embedded Development Kit (EDK) software from Xilinx provides users with a relatively simple way to design MicroBlaze-based systems for custom hardware. With a Xilinx Board Description (XBD) file, one can quickly create new projects that contain controllers and connections for all of the hardware devices on the FPGA board. Appendix B contains the XBD file for the CubeSat SDR system. Finally, using a MicroBlaze processor to control the system allows one to write much of the logic in the C programming language. VHDL, as its name implies, was designed to describe hardware. Complex control systems can be very difficult to design in VHDL, but those same systems might be relatively simple when described as C programs.

This proposed system does have two drawbacks. The majority of the components only work with the Xilinx toolset, so the system loses its platform independence. The Altera toolset has similar devices, such as the NIOS II processor, but these are not compatible with the Xilinx toolset. Thus, anyone wish to use the new USRP firmware in a larger system with Altera's tools could not use this proposed system. In addition, the USB2 controller is currently priced at \$14,000 [22], a prohibitive cost for smaller organizations. Xilinx does offer free trial licenses, but a full license is necessary to operate the system without the JTAG connection.

The proposed CubeSat SDR system should be compatible with GNU Radio with little, if any, modification. With proper configuration, the USB2 controller and the other devices in the system should be functionally equivalent to the original USRP design. To accomplish this, users should consider two things. First, GNU Radio attempts to reprogram the FX2 USB2 controller and the FPGA every time it runs an application. For the CubeSat SDR system, this behavior is not desirable. Since GNU Radio is open source software, it should be easy to bypass this functionality. Second, the USB2 controller should be configured for the same three endpoints that the original USRP used. Table 3.1 describes this configuration [40].

Table 3.1: USB endpoint configuration for the CubeSat SDR.

Endpoint	Function	DMA
0	Control, SPI, I2C	No
2	Tx Data to USRP	Yes
6	Rx Data from USRP	Yes

3.3 Chapter Summary

This chapter describes the differences between the USRP hardware and the CubeSat SDR system hardware. In addition, this chapter presents a proposal for a firmware that incorporates the new USRP adaptation into the overall CubeSat SDR system and explains the advantages and disadvantages of such a system. With the proposed firmware, the CubeSat SDR system should be compatible with the GNU Radio software, providing users with an easy way to test functionality.

Chapter 4

Proposed USRP Variant

This chapter introduces a new USRP firmware that is better suited to the needs of the CubeSat SDR system. This new firmware is functionally equivalent to the original USRP firmware, but does not rely on any platform-specific logic, benefits from VHDL's strong typing, and contains significantly more documentation. It is compatible with the original USRP system and the COSMIAC CubeSat SDR system.

4.1 Proposed USRP Firmware

While its advantages are numerous, the original USRP firmware has several drawbacks. First, it only works with Altera FPGAs and Altera tools. Second, it is written entirely in Verilog. Finally, it contains very little documentation. This chapter proposes a new USRP firmware that:

- *Works on Xilinx FPGAs.* In fact, this new USRP firmware works on any FPGA large enough to support it, including those from Altera. In addition, there are no proprietary components and so the design should work with any toolset.
- *Is written in VHDL.* The VHSIC Hardware Description Language (VHDL) is a standard for government, military, and other public sector projects. In addition, VHDL is a strongly typed language. That is, it uses explicit data types and conversion operations. As a result, VHDL offer an additional layer of safety in the design phase.

- *Contains extensive documentation.* One of the benefits of open source software is that other parties can modify and extend it to fit their own needs. However, working with another person's code can be very difficult when that code contains little or no documentation. The original USRP firmware contains extensive high-level documentation on various websites, but the actual Verilog contains almost none. The proposed USRP firmware contains extensive documentation in the VHDL files.

This new USRP firmware will provide for easier integration with other government and military applications, will allow future engineers to modify and extend it with ease, and will work on a larger collection of FPGA devices than the original. The USRP firmware contains almost all of the digital signal processing logic in the system as well as glue logic to control AD9862 mixed signal processor and the USB2 controller. These elements require a sufficiently powerful FPGA.

4.2 A New FPGA

The original USRP hardware uses an Altera Cyclone EP1C12Q240C8 FPGA. However, the firmware on this FPGA used 95% of the available resources [30]. Therefore, the engineers at COSMIAC opted to use a larger FPGA for the CubeSat SDR system. This new FPGA, the Xilinx Spartan3A-1400 (XCS1400A-5FG484), contains more logic cells, more internal random access memory (RAM), and more user I/O pins. Table 4.1 compares the two FPGAs [1] [3].

Table 4.1: Comparison of Cyclone EP1C12 and Spartan3A-1400.

	Cyclone EP1C12	Spartan3A-1400
Logic Elements	12,060	25,344[1]
RAM Bits	239,616	589,824
I/O Pins	173	375

[1]A Spartan3A "slice" consists of two 4-input lookup tables (LUTs) and 2 registers, while a Cyclone "logic element" contains only one of each. Thus, it is fair to say that one slice is worth approximately two logic elements. With this in mind, the Spartan3A-1400 would have 50,688 LEs.

Aside from providing additional resources, the largest affect of the switch in FPGA technology is the mandatory change in toolset. Altera supplies the Quartus II toolset for use with their FPGAs while Xilinx provides ISE. These toolsets provide similar functionality, but each has its own quirks. For example, each toolset provides tools to automatically generate VHDL through graphical user interfaces (GUI). However, these automatically generated files only work within their original environments. Unlike the original USRP, the CubeSat SDR system does not contain any platform-specific logic.

4.3 A New Language

The USRP firmware written by Ettus Research LLC used the Verilog hardware description language (HDL). As is true for many public sector projects, the sponsors of this thesis preferred the VHSIC HDL (VHDL). Therefore, the CubeSat SDR firmware is written in VHDL. Just as one might translate the text of a novel from German to English, once can translate Verilog logic descriptions into VHDL logic descriptions. This section describes the process of doing just that.

Much of the original Verilog code was relatively straightforward to translate into VHDL. Since the logic is not changed, oftentimes translation is a matter of replacing one keyword with another and accounting for syntaxual differences between the two languages. For example, consider the Verilog code in Listing 4.1.

This code describes a 4-to-1 multiplexer where the four inputs are *d1*, *d2*, *d3*, and *d4*, the select signals are *s(1:0)*, and the output is *q*. Listing 4.2 describes the same logic in VHDL. Note the similarities and differences between the two. Both begin by declaring their inputs and outputs and both contain a "case" statement to implement the multiplexer logic. The Verilog module uses "input" and "output" to classify ports, while VHDL uses "in" and "out". In VHDL, ports require a type specifier, such as as std_logic. Verilog does not require this, but it does require that we classify each port as a wire (connects two ports) or a reg (stores a value). In general, the VHDL code is more verbose.

In order to facilitate testing, each VHDL module is port-compatible with its Verilog counterpart. This required changing the port names of some Verilog modules (e.g. ports

Listing 4.1: 4-to-1 Multiplexer in Verilog.

```
module mux4to1(input wire d1, input wire d2, input wire d3, input wire
    d4,
    input wire [1:0] s, output reg q);

    always @(d1, d2, d3, d4, s)
    begin
        case(s)
            0: q = d1;
            1: q = d2;
            2: q = d3;
            3: q = d4;
        endcase
    end
endmodule
```

called *in* or *out*, since these are reserved keywords in VHDL). Additionally, it required modifying any modules that did not synthesize properly in the Xilinx toolset. Using this method, one can test a single VHDL module in the original Verilog system and incrementally build a complete VHDL system.

Some aspects of the translation were not as simple as replacing keywords and fixing syntax. The following subsections describe these challenges.

4.3.1 Header Files

Verilog supports header files; VHDL does not. However, VHDL does support package files. Package files may contain constants, functions, and other declarations and can be included in any VHDL source file. Thus, the CubeSat SDR system uses a series of package files to replace the configuration headers of the original USRP firmware. The primary package file, common_config.vhd, includes a list of other package files that enable different transmit and receive configurations on the FPGA. The user may uncomment any one of these configurations and the common_config.vhd package handles the details.

Listing 4.2: 4-to-1 Multiplexer in VHDL.

```
entity mux4to1 is
    port(d1 : in std_logic; d2 : in std_logic; d3 : in std_logic;
        d4 : in std_logic; s : in std_logic_vector(1 downto 0);
        q : out std_logic);
end mux4to1;

architecture Behavioral of mux4to1 is
begin
    process(d1, d2, d3, d4, s)
    begin
        case s is
            when "00" => q <= d1;
            when "01" => q <= d2;
            when "10" => q <= d3;
            when others => q <= d4;
        end case;
    end process;
end Behavioral;
```

One limitation of using package files to replace the headers is VHDL's lack of a preprocessor. Verilog has a preprocessor similar to the one used by ANSI C. It supports directives such as **'ifdef**, **'endif**, and **'include**. Consider Listing 4.3, which shows part of the common_config.vh header file. This header checks to see if TX_ON is defined (in another header file) and, if so, checks to see if it should configure one Tx channel or two. If TX_SINGLE is defined, then TX_EN_0 will also be defined and TX_CAP_NCHAN will have the value "001" (1). If TX_DUAL is defined, then both TX_EN_0 and TX_EN_1 will be defined and TX_CAP_NCHAN will have the value "010" (2). These definitions will be used later to determine how many transmit paths to build on the FPGA.

Unfortunately, VHDL does not have a preprocessor and package files do not handle conditional assignments like this. However, package files do support constants that are

Listing 4.3: Verilog configuration code (common_config.vh).

```
`ifdef TX_ON

 `ifdef TX_SINGLE
  `define TX_EN_0
  `define TX_CAP_NCHAN 3'd1
 `endif

 `ifdef TX_DUAL
  `define TX_EN_0
  `define TX_EN_1
  `define TX_CAP_NCHAN 3'd2
 `endif
`endif
```

defined by logic statements. Listing 4.4 demonstrates the same configuration in VHDL. Note that the constants TX_EN, TX_EN_0, and TX_EN_1 are always defined and that we use their values (*true* or *false*) to determine which Tx paths to build rather than relying on future **`ifdef** macros.

Listing 4.4: VHDL configuration code (common_config.vhd).

```
constant TX_EN : boolean := TX_ON;
constant TX_EN_0 : boolean := (TX_EN and (TX_SINGLE or TX_DUAL));
constant TX_EN_1 : boolean := (TX_EN and TX_DUAL);
constant TX_CAP_NCHAN : std_logic_vector(2 downto 0) := '0'
                                                 & conv_std_logic(TX_EN and
                                                     TX_DUAL)
                                                 & conv_std_logic(TX_EN and
                                                     TX_SINGLE));
```

4.3.2 Ternary Operators

Like C, Verilog supports a ternary operator, ?:. This operator is similar to if/else logic. For example, the statement '**assign** q = a ? b : c;' assigns the value of *b* to *q* if *a* is true, else it assigns the value of *c* to q. VHDL does not have such an operator. The original USRP firmware used the ternary operator nested three deep to create multiplexers. The easiest way to implement this functionality in VHDL is with if/else logic inside of a process statement. Thus, the Verilog code in Listing 4.5 becomes the VHDL in Listing 4.6.

Listing 4.5: Example of a Verilog ternary operator.

```
q = mux[2] ? (mux[1] ? (mux[0] ? d3
                               : d2)
                     : (mux[0] ? d1
                               : d0))
           : 1'b0;
```

Rather than replicating this code for each such multiplexer, the CubeSat SDR firmware contains new multiplexer components with equivalent logic. This results in cleaner code with less room for error at the cost of two additional source files (usrp_mux.vhd, adc_mux.vhd).

4.3.3 FIFO

The FIFO in the original USRP firmware has two problems. First, it does not synthesize in the Xilinx toolset. The FIFO was designed to work only in Altera's tools. Ettus Research included a vendor-independent FIFO with the USRP code, but it is not functionally equivalent to the original. Thus, the CubeSat SDR system has a new FIFO. This new FIFO is functionally equivalent to the Altera-specific FIFO and available in both VHDL and Verilog (see Listing A.21 for the VHDL version).

The second problem with the original FIFO is that it used a "read acknowledge" model, which outputs the first data line immediately and switches to the next line after the rising edge of the *rdclk* (read clock) signal. The problem with this model is that it does not provide a clock for reading data and, therefore, cannot be implemented in block RAM

Listing 4.6: VHDL replacement for ternary operators.

```
process(mux, d0, d1, d2, d3)
begin
    if(mux(2) = '1') then
        if(mux(1) = '1') then
            if(mux(0) = '1') then
                q <= d3;
            else
                q <= d2;
            end if;
        else
            if(mux(0) = '1') then
                q <= d1;
            else
                q <= d0;
            end if;
        end if;
    else
        q <= '0';
    end if;
end process;
```

(BRAM). Implementing the read ack FIFO in the FPGA fabric with LUTs is possible, but the two 4096 × 16-bit FIFOs used in the USRP require more LUTs than the Spartan3A-1400 provides. Instead, the new FIFO uses a "read request" model. In this model, the FIFO does not output any data until the rising edge of the *rdclk* signal. This FIFO can be implemented in BRAM and fits on the Spartan3A-1400.

Figure 4.1 shows a simulation of the read request FIFO. At 20 ns, the "write request" signal (wrreq) goes high, pushing the value 3855 into the FIFO. For the next three clock cycles, the simulation writes -3856, -13108, and 13107 into the FIFO. At 60 ns, the "read request" signal (rdreq) goes high and asks to read the first value pushed into the FIFO.

This value appears at 70 ns. For the next three clock cycles, the FIFO continues to output its stored values in the order that they were pushed. At 110 ns, the output ‘q’ is undefined because all values have been read from the FIFO.

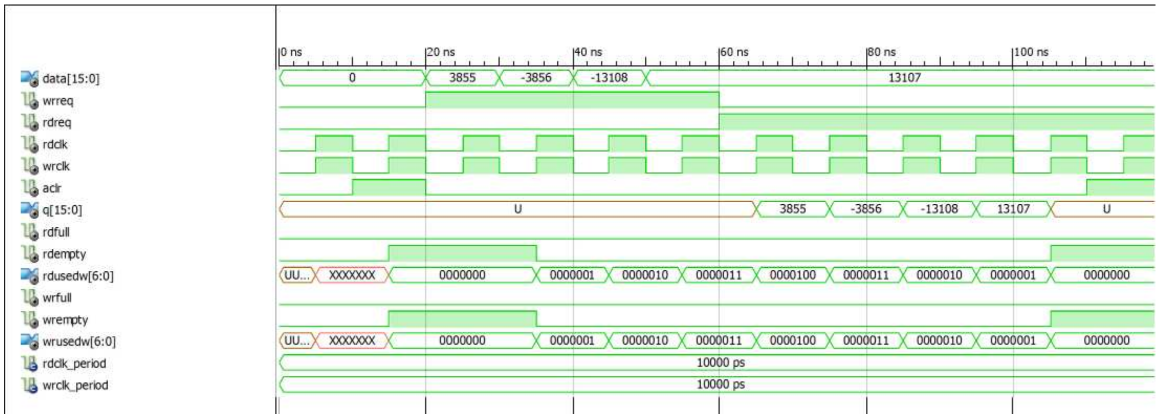


Figure 4.1: Simulation of a read request FIFO. Note that the data on bus ‘q’ is not available until one clock after the ‘rdreq’ signal is pulled high.

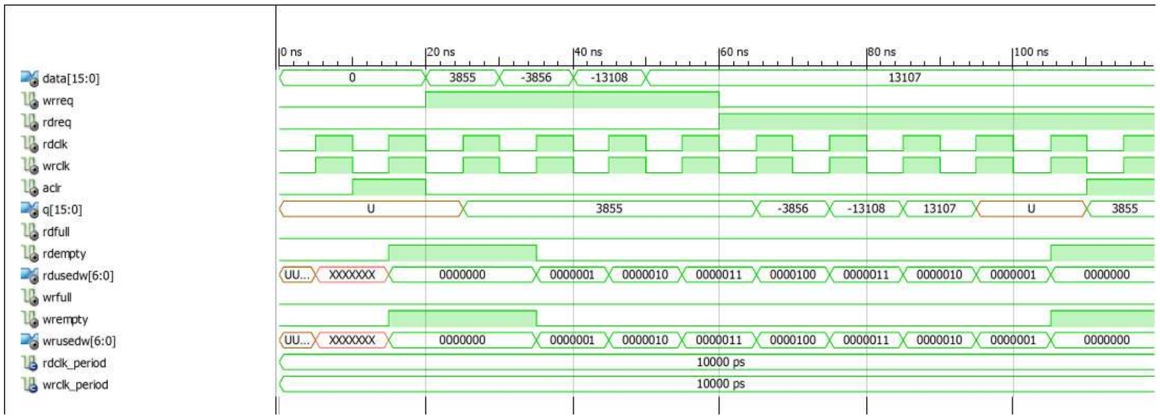


Figure 4.2: Simulation of a read acknowledgement FIFO. Note that the data bus ‘q’ is available one clock after the first write and that this output remains until one clock after the ‘rdreq’ signal is pulled high.

Figure 4.2 shows a simulation of the read ack FIFO. As in Figure 4.1, the simulation begins pushing four values into the system at 20 ns. However, notice that the first value is available on the next clock cycle (30 ns). This value remains on the output bus (q) until the ‘rdreq’ signal goes high, acknowledging that the output was read. Thus, outputs with this model are read one clock earlier than they are with the other model. The original USRP firmware used this second model, but supported the read request model as an option. Due to space and speed constraints, the read request model is used in the new firmware.

4.4 Synthesis Results

If the two USRP firmwares are logically equivalent, then their hardware usage should be very similar. Table 4.2 shows the hardware usage for each module in the USRP firmware as well as the maximum supported clock speed for that module. *Emphasized* modules have differences in hardware usage between the two firmware versions. These results were generated with the 64-bit version of Xilinx ISE 12.3 for the Spartan3A-1400FG484-5 FPGA and are the estimated values reported by that software. The Verilog firmware contains some minor modifications to make it compatible with the Xilinx synthesis tools.

As Table 4.2 shows, most of the modules have identical hardware usage and maximum clock rates. However, a small set of modules do not match. The *cic_interp*, *tx_chain*, *tx_buffer*, and *cic_dec_shifter* all have differences of only one slice of LUT. Since the synthesis outputs and simulations for these modules match, the differences are likely caused by the manner in which Xilinx XST interprets the two HDLs. The same is probably true for *rx_dcoffset*, which differs only in maximum clock speed. While the two HDLs are capable of describing the same logic, the synthesis tool is responsible for choosing how to map that logic. Thus, two modules may be functionally identical even though their hardware usage differs.

Other modules have more significant differences. In all of these cases, the synthesis tool finds the same quantities and types of logic elements and all tests (see Appendix C) show functional equivalence. However, it is possible to describe a logic function in more than one way and these different descriptions might be mapped differently by Xilinx XST. For example, the VHDL version of the *halfband_decim* module uses an accumulator instead of a simple adder. The functionality is the same and the source code very similar, but Xilinx XST infers an accumulator in the VHDL code and an adder in the Verilog code. Often, these differences propagate to higher level modules. In this case, the difference in *halfband_decim* propagates upward to the *rx_chain*.

Table 4.2: Hardware utilization for USRP modules.

Module	VHDL					Verilog				
	Flip Flops	Slices	LUTs	BRAMs	Max Clock (MHz)	Flip Flops	Slices	LUTs	BRAMs	Max Clock (MHz)
bustri	0	1	1	0	–	0	1	1	0	–
setting_reg	0	2	4	0	–	0	2	4	0	–
serial_io	52	137	258	0	167.972	52	137	258	0	167.972
fifo	40	23	44	1	284.953	40	23	44	1	284.953
fifo_4k_18	48	28	54	5	263.866	48	28	54	5	263.866
bidir_reg	0	0	16	0	–	0	0	16	0	–
io_pins	64	41	134	0	603.883	64	41	134	0	603.883
cic_int_shifter	0	81	156	0	–	0	81	156	0	–
sign_extend	0	0	0	0	–	0	0	0	0	–
cic_interp	383	284	437	0	192.569	383	285	437	0	192.569
tx_chain	766	565	842	0	192.569	766	566	842	0	192.569
tx_buffer	144	88	77	5	198.275	144	87	76	5	198.275
rx_buffer	191	191	287	5	120.049	191	171	301	5	120.049
coeff_rom	0	8	14	0	–	0	8	14	0	–
ram16_2sum	48	76	113	0	143.432	48	67	97	0	144.489
ram16	16	16	32	0	510.204	16	16	32	0	510.204
mult	0	1	1	0	–	0	1	1	0	–
acc	34	18	35	0	203.544	34	18	35	0	203.544
halfband_decim	118	169	325	0	138.383	118	150	291	0	139.367
cic_dec_shifter	0	152	263	0	–	0	151	262	0	—
cic_decim	580	379	616	0	177.055	580	381	601	0	177.055
cordic_stage	0	24	49	0	–	0	24	49	0	—
cordic	659	359	678	0	218.759	659	359	678	0	218.759
phase_acc	64	53	68	0	172.831	64	53	68	0	172.831
rx_chain	2087	1464	2577	0	138.383	2087	1455	2475	0	139.367
rx_dcoffset	32	56	106	0	84.72	32	56	106	0	74.384
rssi	52	55	104	0	143.981	52	55	104	0	143.981
adc_interface	533	573	1100	0	83.974	533	571	1096	0	73.492
strobe_gen	8	12	23	0	231.463	8	12	23	0	231.463
clk_divider	9	15	31	0	221.21	9	15	31	0	221.21
atr_delay	14	30	57	0	192.897	14	30	57	0	192.897
master_control	412	316	542	0	192.897	412	316	542	0	192.897

Each of the modules in Table 4.2 is a submodule of the *usrp_std* module. Table 4.3 presents three different implementations of this top-level module. The first implementation consists of only VHDL code and the second consists of only Verilog code. The final implementation uses a Verilog top-level module with purely VHDL submodules. This third implementation shows two things. First, the synthesis differences between the pure VHDL implementation and the hybrid implementation are minimal and likely a result of XST producing a different mapping with equivalent logic and timing. Second, it shows that the system is capable of supporting a mix of VHDL and Verilog modules.

Table 4.3: Hardware utilization for the USRP firmware.

Logic Type	VHDL		Verilog		Hybrid	
	No. Used	% used	No. Used	% used	No. Used	% used
Slices	3197	28	3,164	18	3,200	28
Flip Flops	4,068	18	4068	28	4,068	18
4-Input LUTs	5,467	24	5,360	23	5,467	24
I/O Buffers	173	46	173	46	173	46
Block RAMs	10	31	10	31	10	31
18×18 Multipliers	2	6	2	6	2	6
Clock Buffers	3	12	3	12	3	12

4.5 Chapter Summary

This chapter presented a new firmware for the CubeSat SDR system's USRP and discussed the challenges of targeting a different platform and using a different language, including the need to work around missing language features, to overcome toolset limitations, and to rewrite some modules. Finally, this chapter included some simulations and synthesis results to show the functional equivalence and hardware usage of the two different firmwares.

Chapter 5

Conclusions

5.1 Summary and Accomplishments

This thesis presented a flexible, plug and play software-defined radio system for CubeSat form-factor satellites. Based on the USRP, this new CubeSat SDR provides CubeSat engineers with an easy to use SDR that is compatible with the GNU Radio software and the Space Plug-and-Play Avionics (SPA) protocol.

The new adaptation of the USRP firmware is written in VHDL, instead of Verilog, and contains much more documentation than the original. Each module in the system is port-compatible with and functionally equivalent to its Verilog counterpart from the original USRP firmware. Thus, future users will be able to modify small parts of the system and test them with ease. Further, USRP users could choose to use the new VHDL firmware instead of the Verilog version if it better suits their needs. Like the original, this new firmware is free and open source.

The proposed CubeSat SDR system incorporates this firmware into a larger system that replaces the original USRP hardware with a design that fits into a 1U CubeSat. This new hardware maintains the interchangeable RF frontend design, allowing future users to adapt the system to various projects. The combination of hardware, software, and firmware described in this thesis results in a USRP-like system that is compatible with the GNU Radio software, a free SDR design and testing platform.

5.2 Future Work

The CubeSat SDR system could benefit from further research and design. Some possible areas of exploration include:

- *Migrate to the Spartan6 FPGA.* The current Spartan3A-1400 FPGA provides more resources than the original Cyclone FPGA, but it does not provide enough RAM bits to support the large FIFOs and the numerous peripheral controllers in the CubeSat SDR system. The Spartan6 FPGA provides more logic elements, RAM, and I/O capabilities. Further, it was designed as a drop-in replacement for the Spartan3. Migrating to the Spartan6 FPGA would be a relatively simple way to improve numerous aspects of the CubeSat SDR system.

- *Replace the USB2 controller.* The current USB2 controller, the USB3300, is significantly inferior to the original Cypress FX2 controller present on the USRP. As a result, a significant portion of the FPGA is used to supply this lost functionality. With the FX2, future users would have far more space on the FPGA for custom designs. In addition, they would not need to license the rather expensive USB2 IP core from Xilinx.

- *Improve the memory.* The CubeSat SDR system has three types of memory that the original USRP lacked. However, these memories are not ideal. The DDR2 memory controller is especially complicated and requires very specific timing constraints. Such controllers are often expensive, hard to use, or both. The NOR flash configuration memory is too small to fit the CubeSat SDR firmware. This memory should be larger.

- *Adapt the USRP2.* Instead of fixing the problems with the current USRP1-based system, respin the CubeSat as a USRP2 adaptation. The USRP2 was designed to work with Xilinx FPGAs and already include a MicroBlaze, memories, and many of the other components that the CubeSat SDR system requires. In addition, it uses Gigabit Ethernet instead of USB2, simplifying the controller design while simultaneously providing higher throughput. One downside to this approach is that the USRP2 is described in Verilog.

- *Eliminate GNU Radio.* Regardless of which platform the CubeSat SDR system is based on, GNU radio needs to go. GNU Radio runs on general-purpose personal computers. Such computers would not fit into a CubeSat. Therefore, the dependency on GNU Radio needs to go. Instead of using Python or an XML-based GUI to describe digital signal processing operations, a similar system that produces VHDL components and connects them together would add significant value to the CubeSat SDR system.

Appendix A

USRP Code

This appendix presents the VHDL source code for the CubeSat SDR system's USRP firmware, as described in Chapter 4. All of the code listed in this appendix is licensed under the GNU General Public License version 2 (GPL2).

Listing A.1: Accumulator module, used with the mult module to create a multiply accumulator (MAC). (acc.vhd)

```
-------------------------------------------------------------------------------
-- Accumulator
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--   Copyright (C) 2003 Matt Ettus
--
--   This program is free software; you can redistribute it and/or modify
--   it under the terms of the GNU General Public License as published by
--   the Free Software Foundation; either version 2 of the License, or
--   (at your option) any later version.
--
--   This program is distributed in the hope that it will be useful,
--   but WITHOUT ANY WARRANTY; without even the implied warranty of
--   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--   GNU General Public License for more details.
--
--   You should have received a copy of the GNU General Public License
--   along with this program; if not, write to the Free Software
--   Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
-------------------------------------------------------------------------------

```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

library WORK;
use WORK.FUNCS.ALL;

entity acc is
    port(
        clock           : in    std_logic;
        reset           : in    std_logic;
        clear           : in    std_logic;
        enable_in    : in    std_logic;
        enable_out   : out   std_logic;
        addend       : in    std_logic_vector(30 downto 0);
        sum          : out   std_logic_vector(33 downto 0)
    );
end acc;

architecture behavioral of acc is
    signal addend_t : signed(30 downto 0);
    signal sum_t : signed(33 downto 0);
begin
    addend_t <= conv_signed(conv_integer(addend), 31);
    sum <= conv_std_logic_vector(sum_t);

    -- Working with signed logic vectors in VHDL really, really sucks.
    -- SXT is sign extend, but it only works on std_logic_vector.  So,
    -- we need lots of conversion operators to make the synthesis tools
    -- happy.  In Verilog, this is much simpler!  For now, I wrote new
    -- conversion functions in the FUNCS package.  :)
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(reset = '1') then
                sum_t <= (others => '0');
            elsif(clear = '1') then
                sum_t <= conv_signed(sxt(conv_std_logic_vector(addend_t), 34));
            elsif(enable_in = '1') then
                sum_t <= sum_t + conv_signed(sxt(conv_std_logic_vector(addend_t), 34));
            end if;
        end if;
    end process;

    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            enable_out <= enable_in;
        end if;
    end process;
end behavioral;
```

Listing A.2: ADC interface module to control the AD9862 mixed signal processor. (adc_interface.vhd)

```
--------------------------------------------------------------------------------
```

```
-- ADC Interface
-- Last Modified: 20 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library WORK;
use WORK.FPGA_REGS_COMMON.ALL;
use WORK.FPGA_REGS_STANDARD.ALL;
use WORK.COMMON_CONFIG.ALL; -- FIXME Should we do this?

entity adc_interface is
    port(
        clock           : in    std_logic;
        reset           : in    std_logic;
        enable          : in    std_logic;
        serial_addr     : in    std_logic_vector(6 downto 0);
        serial_data     : in    std_logic_vector(31 downto 0);
        serial_strobe   : in    std_logic;
        rx_a_a          : in    std_logic_vector(11 downto 0);
        rx_b_a          : in    std_logic_vector(11 downto 0);
        rx_a_b          : in    std_logic_vector(11 downto 0);
        rx_b_b          : in    std_logic_vector(11 downto 0);
        rssi_0          : out   std_logic_vector(31 downto 0);
        rssi_1          : out   std_logic_vector(31 downto 0);
        rssi_2          : out   std_logic_vector(31 downto 0);
        rssi_3          : out   std_logic_vector(31 downto 0);
        ddc0_in_i       : out   std_logic_vector(15 downto 0);
        ddc0_in_q       : out   std_logic_vector(15 downto 0);
        ddc1_in_i       : out   std_logic_vector(15 downto 0);
        ddc1_in_q       : out   std_logic_vector(15 downto 0);
        ddc2_in_i       : out   std_logic_vector(15 downto 0);
        ddc2_in_q       : out   std_logic_vector(15 downto 0);
        ddc3_in_i       : out   std_logic_vector(15 downto 0);
        ddc3_in_q       : out   std_logic_vector(15 downto 0);
        rx_numchan      : out   std_logic_vector(3 downto 0)
    );
end adc_interface;

architecture behavioral of adc_interface is
    component setting_reg is
        generic(
            my_addr : integer := 0
        );

        port(
            clock   : in    std_logic;
            reset   : in    std_logic;
            strobe  : in    std_logic;
            addr    : in    std_logic_vector;
            d_in    : in    std_logic_vector(31 downto 0);
            d_out   : out   std_logic_vector(31 downto 0);
            changed : out   std_logic
        );
    end component;

    component rx_dcoffset is
        generic(
            MYADDR : integer := 0
        );

        port(
            clock           : in    std_logic;
            enable          : in    std_logic;
            reset           : in    std_logic;
            adc_in          : in    std_logic_vector(15 downto 0);
            adc_out         : out   std_logic_vector(15 downto 0);
            serial_addr     : in    std_logic_vector(6 downto 0);
            serial_data     : in    std_logic_vector(31 downto 0);
            serial_strobe   : in    std_logic
        );
    end component;

    component rssi is
        port(
            clock       : in    std_logic;
            reset       : in    std_logic;
            enable      : in    std_logic;
            adc         : in    std_logic_vector(11 downto 0);
            rssi        : out   std_logic_vector(15 downto 0);
            over_count  : out   std_logic_vector(15 downto 0)
        );
    end component;

    component adc_mux is
        port(
            clock           : in    std_logic;
            rx_realsignals  : in    std_logic;
            ddc_mux         : in    std_logic_vector(3 downto 0);
            adc3_corr       : in    std_logic_vector(15 downto 0);
            adc2_corr       : in    std_logic_vector(15 downto 0);
            adc1_corr       : in    std_logic_vector(15 downto 0);
            adc0_corr       : in    std_logic_vector(15 downto 0);
            ddc_i           : out   std_logic_vector(15 downto 0);
            ddc_q           : out   std_logic_vector(15 downto 0)
        );
    end component;

    -- Buffers
    signal adc0 : std_logic_vector(11 downto 0);
    signal adc1 : std_logic_vector(11 downto 0);
    signal adc2 : std_logic_vector(11 downto 0);
    signal adc3 : std_logic_vector(11 downto 0);

    signal dco_en : std_logic_vector(3 downto 0);
    signal adc0_corr : std_logic_vector(15 downto 0);
    signal adc1_corr : std_logic_vector(15 downto 0);
    signal adc2_corr : std_logic_vector(15 downto 0);
    signal adc3_corr : std_logic_vector(15 downto 0);

    signal ddc0mux : std_logic_vector(3 downto 0);
    signal ddc1mux : std_logic_vector(3 downto 0);
    signal ddc2mux : std_logic_vector(3 downto 0);
    signal ddc3mux : std_logic_vector(3 downto 0);

    signal rx_realsignals : std_logic;
begin
    -- Buffer at the inputs to the chip.
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            adc0 <= rx_a_a;
            adc1 <= rx_b_a;
            adc2 <= rx_a_b;
            adc3 <= rx_b_b;
        end if;
    end process;

    -- Then scale and subtract DC offset.
    sr_dco_en : setting_reg
        generic map(my_addr => FR_DC_OFFSET_CL_EN)
        port map(clock => clock, reset => reset, strobe => serial_strobe, addr => serial_addr,
            d_in => serial_data, d_out(31 downto 4) => OPEN, d_out(3 downto 0) => dco_en, changed
                => OPEN);

    rx_dcoffset0 : rx_dcoffset
        generic map(MYADDR => FR_ADC_OFFSET_0)
        port map(clock => clock, enable => dco_en(0), reset => reset, adc_in(15) => adc0(11),
            adc_in(14 downto 3) => adc0, adc_in(2 downto 0) => "000", adc_out => adc0_corr,
            serial_addr => serial_addr, serial_data => serial_data, serial_strobe =>
                serial_strobe);

    rx_dcoffset1 : rx_dcoffset
        generic map(MYADDR => FR_ADC_OFFSET_1)
        port map(clock => clock, enable => dco_en(1), reset => reset, adc_in(15) => adc1(11),
            adc_in(14 downto 3) => adc1, adc_in(2 downto 0) => "000", adc_out => adc1_corr,
            serial_addr => serial_addr, serial_data => serial_data, serial_strobe =>
                serial_strobe);

    rx_dcoffset2 : rx_dcoffset
        generic map(MYADDR => FR_ADC_OFFSET_2)
        port map(clock => clock, enable => dco_en(2), reset => reset, adc_in(15) => adc2(11),
            adc_in(14 downto 3) => adc2, adc_in(2 downto 0) => "000", adc_out => adc2_corr,
            serial_addr => serial_addr, serial_data => serial_data, serial_strobe =>
                serial_strobe);

    rx_dcoffset3 : rx_dcoffset
        generic map(MYADDR => FR_ADC_OFFSET_3)
        port map(clock => clock, enable => dco_en(3), reset => reset, adc_in(15) => adc3(11),
            adc_in(14 downto 3) => adc3, adc_in(2 downto 0) => "000", adc_out => adc3_corr,
            serial_addr => serial_addr, serial_data => serial_data, serial_strobe =>
                serial_strobe);

    -- Level sensing for AGC
    rssi_block_0 : rssi
        port map(clock => clock, reset => reset, enable => enable, adc => adc0, rssi => rssi_0(15
            downto 0),
            over_count => rssi_0(31 downto 16));

    rssi_block_1 : rssi
        port map(clock => clock, reset => reset, enable => enable, adc => adc1, rssi => rssi_1(15
            downto 0),
            over_count => rssi_1(31 downto 16));

    rssi_block_2 : rssi
        port map(clock => clock, reset => reset, enable => enable, adc => adc2, rssi => rssi_2(15
            downto 0),
            over_count => rssi_2(31 downto 16));

    rssi_block_3 : rssi
        port map(clock => clock, reset => reset, enable => enable, adc => adc3, rssi => rssi_3(15
            downto 0),
            over_count => rssi_3(31 downto 16));

    -- And MUX to the appropriate outputs.
    sr_rxmux : setting_reg
        generic map(my_addr => FR_RX_MUX)
        port map(clock => clock, reset => reset, strobe => serial_strobe, addr => serial_addr,
            d_in => serial_data, d_out(31 downto 20) => OPEN, d_out(19 downto 16) => ddc3mux,
            d_out(15 downto 12) => ddc2mux, d_out(11 downto 8) => ddc1mux, d_out(7 downto 4) =>
                ddc0mux,
            d_out(3) => rx_realsignals, d_out(2 downto 0) => rx_numchan(3 downto 1), changed =>
                OPEN);

    rx_numchan(0) <= '0';

    -- VHDL really needs to start using ternary operators!  ?: is awesome, this is ugly.
--  rx_en_0_t : if(RX_EN_0) generate
        adc_mux0 : adc_mux
            port map(clock => clock, rx_realsignals => rx_realsignals, ddc_mux => ddc0mux,
                adc3_corr => adc3_corr, adc2_corr => adc2_corr, adc1_corr => adc1_corr, adc0_corr
                    => adc0_corr,
                ddc_i => ddc0_in_i, ddc_q => ddc0_in_q);
--  end generate rx_en_0_t;

--  rx_en_0_f : if(not RX_EN_0) generate
--      ddc0_in_i <= (others => '0');
--      ddc0_in_q <= (others => '0');
--  end generate rx_en_0_f;

--  rx_en_1_t : if(RX_EN_1) generate
        adc_mux1 : adc_mux
            port map(clock => clock, rx_realsignals => rx_realsignals, ddc_mux => ddc1mux,
                adc3_corr => adc3_corr, adc2_corr => adc2_corr, adc1_corr => adc1_corr, adc0_corr
                    => adc0_corr,
                ddc_i => ddc1_in_i, ddc_q => ddc1_in_q);
--  end generate rx_en_1_t;

--  rx_en_1_f : if(not RX_EN_1) generate
--      ddc1_in_i <= (others => '0');
--      ddc1_in_q <= (others => '0');
--  end generate rx_en_1_f;

--  rx_en_2_t : if(RX_EN_2) generate
        adc_mux2 : adc_mux
            port map(clock => clock, rx_realsignals => rx_realsignals, ddc_mux => ddc2mux,
                adc3_corr => adc3_corr, adc2_corr => adc2_corr, adc1_corr => adc1_corr, adc0_corr
                    => adc0_corr,
                ddc_i => ddc2_in_i, ddc_q => ddc2_in_q);
--  end generate rx_en_2_t;

--  rx_en_2_f : if(not RX_EN_2) generate
--      ddc2_in_i <= (others => '0');
--      ddc2_in_q <= (others => '0');
--  end generate rx_en_2_f;

--  rx_en_3_t : if(RX_EN_3) generate
        adc_mux3 : adc_mux
            port map(clock => clock, rx_realsignals => rx_realsignals, ddc_mux => ddc3mux,
                adc3_corr => adc3_corr, adc2_corr => adc2_corr, adc1_corr => adc1_corr, adc0_corr
                    => adc0_corr,
                ddc_i => ddc3_in_i, ddc_q => ddc3_in_q);
--  end generate rx_en_3_t;

--  rx_en_3_f : if(not RX_EN_3) generate
--      ddc3_in_i <= (others => '0');
--      ddc3_in_q <= (others => '0');
--  end generate rx_en_3_f;
end behavioral;
```

Listing A.3: Multiplexer for the adc_interface module that replaces a large ternary expression from the original Verilog source. No Verilog version exists. (adc_mux.vhd)

```
----------------------------------------------------------------------------
-- Special MUX for ADC Signals
-- Last Modified: 20 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
```

```
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------

-- The original Verilog USRP1 contains no equivalent for this component.  In
-- Verilog, this logic is trivial to implement with the ternary ?: operator.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity adc_mux is
    port(
        clock           : in    std_logic;
        rx_realsignals  : in    std_logic;
        ddc_mux         : in    std_logic_vector(3 downto 0);
        adc3_corr       : in    std_logic_vector(15 downto 0);
        adc2_corr       : in    std_logic_vector(15 downto 0);
        adc1_corr       : in    std_logic_vector(15 downto 0);
        adc0_corr       : in    std_logic_vector(15 downto 0);
        ddc_i           : out   std_logic_vector(15 downto 0);
        ddc_q           : out   std_logic_vector(15 downto 0)
    );
end adc_mux;

architecture behavioral of adc_mux is
begin
    -- First, take care of ddc_i.  The equivalent Verilog statement is:
    --    ddc_i = ddc_mux[1] ? (ddc_mux[0] ? adc3_corr
    --                                     : adc2_corr)
    --                       : (ddc_mux[0] ? adc1_corr
    --                                     : adc0_corr);
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(ddc_mux(1) = '1') then
                if(ddc_mux(0) = '1') then
                    ddc_i <= adc3_corr;
                else
                    ddc_i <= adc2_corr;
                end if;
            else
                if(ddc_mux(0) = '1') then
                    ddc_i <= adc1_corr;
                else
```

```
                    ddc_i <= adc0_corr;
                end if;
            end if;
        end if;
    end process;

    -- Now, take care of ddc_q.  The equivalent Verilog statement is:
    --    ddc_q = rx_realsignals ? 16'b0
    --                           : ddc_mux[3] ? (ddc_mux[2] ? adc3_corr
    --                                                      : adc2_corr)
    --                                        : (ddc_mux[2] ? adc1_corr
    --                                                      : adc0_corr);
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(rx_realsignals = '1') then
                ddc_q <= (others => '0');
            else
                if(ddc_mux(3) = '1') then
                    if(ddc_mux(2) = '1') then
                        ddc_q <= adc3_corr;
                    else
                        ddc_q <= adc2_corr;
                    end if;
                else
                    if(ddc_mux(2) = '1') then
                        ddc_q <= adc1_corr;
                    else
                        ddc_q <= adc0_corr;
                    end if;
                end if;
            end if;
        end if;
    end process;
end behavioral;
```

Listing A.4: Auto transmit/receive switch that adds a configurable delay when switching from transmit to receive or from receive to transmit. (atr_delay.vhd)

```
--------------------------------------------------------------------------------
-- ATR Delay
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
```

```
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
-------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity atr_delay is
    port(
        clk_i         : in    std_logic;
        rst_i         : in    std_logic;
        ena_i         : in    std_logic;
        tx_empty_i  : in    std_logic;
        tx_delay_i  : in    std_logic_vector(11 downto 0);
        rx_delay_i  : in    std_logic_vector(11 downto 0);
        atr_tx_o      : out   std_logic
    );
end atr_delay;

-- Auto Transmit/Receive Switch.  Adds a delay when switching from Tx to Rx
-- or from Rx to Tx.  Configurable as a multiple of clock cycles.
architecture behavioral of atr_delay is
    signal state : std_logic_vector(3 downto 0);
    signal count : std_logic_vector(11 downto 0);

    -- The original Verilog forces one-hot encoding and I've kept that here.
    -- XST seems to prefer grey coding for both versions, though.
    constant ST_RX_DELAY    : std_logic_vector(3 downto 0) := "0001";
    constant ST_RX          : std_logic_vector(3 downto 0) := "0010";
    constant ST_TX_DELAY    : std_logic_vector(3 downto 0) := "0100";
    constant ST_TX          : std_logic_vector(3 downto 0) := "1000";
begin
    process(clk_i)
    begin
        if(clk_i'EVENT and clk_i = '1') then
            if(rst_i = '1' or ena_i = '0') then
                state <= ST_RX;
                count <= (others => '0');
            else
                case state is
                    -- Receiving State
                    when ST_RX =>
                        -- If there's something in the Tx buffer, go to Tx.
                        if(tx_empty_i = '0') then
                            state <= ST_TX_DELAY;
                            count <= tx_delay_i;
                        end if;
                    -- Rx->Tx Delay State
                    when ST_TX_DELAY =>
```

```
                        if(count = 0) then
                            state <= ST_TX;
                        else
                            count <= count - 1;
                        end if;
                    -- Transmitting State
                    when ST_TX =>
                        -- When the Tx buffer is empty, go back to Rx.
                        if(tx_empty_i = '1') then
                            state <= ST_RX_DELAY;
                            count <= rx_delay_i;
                        end if;
                    -- Tx->Rx Delay State
                    when ST_RX_DELAY =>
                        if(count = 0) then
                            state <= ST_RX;
                        else
                            count <= count - 1;
                        end if;
                    when others =>  -- Error
                        state <= ST_RX;
                        count <= (others => '0');
                end case;
            end if;
        end if;
    end process;

    atr_tx_o <= '1' when (state = ST_TX or state = ST_RX_DELAY) else '0';
end behavioral;
```

Listing A.5: A 16-bit tri-state bus with separate enable signals for each bit. (bidir_reg.vhd)

```
--------------------------------------------------------------------------------
-- Tri-State Register
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--   Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
---------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity bidir_reg is
    port(
        tristate    : inout std_logic_vector(15 downto 0);
        oe          : in        std_logic_vector(15 downto 0);
        reg_val : in        std_logic_vector(15 downto 0)
    );
end bidir_reg;

architecture behavioral of bidir_reg is
begin
    -- This is just another macro for a 16-bit tri-state bus.  This one
    -- has a separate enable for each bit, unlike the bus-wide enable
    -- in bustri.
    process(reg_val, oe)
    begin
        for i in 15 downto 0 loop
            if(oe(i) = '1') then
                tristate(i) <= reg_val(i);
            else
                tristate(i) <= 'Z';
            end if;
        end loop;
    end process;
end behavioral;
```

Listing A.6: Another 16-bit tri-state bus, but with only one bus-wide enable signal. (bustri.vhd)

```
---------------------------------------------------------------------------
-- Tri-state Bus Model
-- Last Modified: 20 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
-------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity bustri is
    port(
        data       : in    std_logic_vector(15 downto 0);
        enabledt   : in    std_logic;
        tridata : out   std_logic_vector(15 downto 0)
    );
end bustri;

architecture behavioral of bustri is
begin
    -- This is just a macro for a 16-bit tri-state bus.
    tridata <= data when enabledt = '1' else (others => 'Z');
end behavioral;
```

Listing A.7: CIC decimator module, part of the rx_chain. Decimates received data in baseband. (cic_decim.vhd)

```
-------------------------------------------------------------------------------
-- CIC Decimation
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
-------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- A CIC decimator is equivalent to a moving average filter.  Basically,
-- take N equivalent moving average filters (FIR filters), then separate
-- them so that the input goes through all integrators first and then all combs.
-- We can do this because both combs and integrators are LTI.  To think of a
-- moving average filter, imagine that at each step we add the newest sample
-- and remove the oldest sample, then divide by the number of stages in the
-- filter.  Something like this (divided by N).
--    y[n] = y[n-1] + x[n] - x[n-R]
entity cic_decim is
    generic(
        bw                      : integer := 16;    -- # of bits for input
        N                       : integer := 4; -- # of filter stages
        log2_of_max_rate    : integer := 7; -- log2 of max sampling rate
        maxbitgain          : integer := 28 -- N*log2_of_max_rate
    );

    port(
        clock           : in    std_logic;
        reset           : in    std_logic;
        enable      : in    std_logic;
        rate            : in    std_logic_vector(7 downto 0);
        strobe_in   : in    std_logic;
        strobe_out  : in    std_logic;
        signal_in   : in    std_logic_vector(bw-1 downto 0);
        signal_out  : out   std_logic_vector(bw-1 downto 0)
    );
end cic_decim;

architecture behavioral of cic_decim is
    component sign_extend is
        generic(
            bits_in     : integer := 0;
            bits_out    : integer := 0
        );

        port(
            d_in    : in    std_logic_vector(bits_in-1 downto 0);
            d_out   : out   std_logic_vector(bits_out-1 downto 0)
        );
    end component;

    component cic_dec_shifter is
        generic(
            bw : integer := 16;
            maxbitgain : integer := 28
        );

        port(
            rate        : in    std_logic_vector(7 downto 0);
            signal_in   : in    std_logic_vector(bw+maxbitgain-1 downto 0);
            signal_out  : out   std_logic_vector(bw-1 downto 0)
        );
    end component;

    type mem is array (integer range <>) of std_logic_vector(bw+maxbitgain-1 downto 0);
    signal integrator : mem(0 to N-1);
    signal differentiator : mem(0 to N-1);
    signal pipeline : mem(0 to N-1);

    signal signal_out_t : std_logic_vector(bw-1 downto 0);

    signal signal_in_ext : std_logic_vector(bw+maxbitgain-1 downto 0);

    signal signal_out_unreg : std_logic_vector(bw-1 downto 0);
    signal signal_out_unnorm : std_logic_vector(bw+maxbitgain-1 downto 0);

    signal sampler : std_logic_vector(bw+maxbitgain-1 downto 0);
begin
    -- Sign extend the input signal to account for bit gain in the filters.
    ext_input : sign_extend
        generic map( bits_in => bw, bits_out => (bw+maxbitgain))
        port map(d_in => signal_in, d_out => signal_in_ext);

    -- Integrator
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(reset = '1') then
                -- Clear the integrator.
                for i in 0 to N-1 loop
                    integrator(i) <= (others => '0');
                end loop;
            elsif((enable = '1') and (strobe_in = '1')) then
                -- Grab the next sample.
                integrator(0) <= integrator(0) + signal_in_ext;

                -- Integrate! The integrator adds the newest sample to the
                -- moving average.
                -- y[n] = y[n-1] + c[n]
                for i in 1 to N-1 loop
                    integrator(i) <= integrator(i) + integrator(i-1);
                end loop;
            end if;
        end if;
    end process;

    -- Comb Filter
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(reset = '1') then
                -- Clear the comb filters.
                sampler <= (others => '0');
                for i in 0 to N-1 loop
                    pipeline(i) <= (others => '0');
                    differentiator(i) <= (others => '0');
                end loop;
            elsif((enable = '1') and (strobe_out = '1')) then
                -- Grab the output of the integrator into the comb filter.
                sampler <= integrator(N-1);

                -- The comb filters remove the oldest samples from the
                -- moving average.
                -- c[n] = x[n] - x[n - R]
                differentiator(0) <= sampler;
                pipeline(0) <= sampler - differentiator(0);
                for i in 1 to N-1 loop
                    differentiator(i) <= pipeline(i-1);
                    pipeline(i) <= pipeline(i-1) - differentiator(i);
                end loop;
            end if;
        end if;
    end process;

    -- Normalize the output to 'bw' bits.
    signal_out_unnorm <= pipeline(N-1);

    cic_dec_shifter0 : cic_dec_shifter
        generic map(bw => bw)
        port map(rate => rate, signal_in => signal_out_unnorm, signal_out => signal_out_unreg);

    -- Register the outputs.
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            signal_out <= signal_out_unreg;
        end if;
    end process;
end behavioral;
```

Listing A.8: Decimation shifter to account for bitgain in the decimation process. (cic_dec_shifter.vhd)

```
--------------------------------------------------------------------------------
-- CIC Decimation Shifter
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
-- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
--------------------------------------------------------------------------------

-- This only works for N=4, max decimation rate of 128.
-- The signal "rate" is ONE LESS THAN the actual rate.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity cic_dec_shifter is
    generic(
        bw          : integer := 16;
        maxbitgain  : integer := 28
    );

    port(
        rate        : in    std_logic_vector(7 downto 0);
        signal_in   : in    std_logic_vector(bw+maxbitgain-1 downto 0);
        signal_out  : out   std_logic_vector(bw-1 downto 0)
    );
end cic_dec_shifter;

architecture behavioral of cic_dec_shifter is
    signal shift : std_logic_vector(4 downto 0);
    signal rate1 : std_logic_vector(7 downto 0);
begin
    -- I don't know why the original Verilog source uses rate-1 in the top
    -- module and then fixes it here.  But this is really the decimation
    -- rate.
    rate1 <= rate + 1;

    -- The actual bit gain is determined by the rate, where
    --    bitgain = N * log2(R)
    -- We fix N at 4 here and R is equal to 'rate'
    with conv_integer(rate1) select
        shift <= conv_std_logic_vector(8, 5) when 4,
                 conv_std_logic_vector(12, 5) when 8,
                 conv_std_logic_vector(16, 5) when 16,
                 conv_std_logic_vector(20, 5) when 32,
                 conv_std_logic_vector(24, 5) when 64,
                 conv_std_logic_vector(28, 5) when 128,
                 conv_std_logic_vector(10, 5) when 5,
                 conv_std_logic_vector(11, 5) when 6,
                 conv_std_logic_vector(12, 5) when 7,
                 conv_std_logic_vector(13, 5) when 9,
                 conv_std_logic_vector(14, 5) when 10 | 11,
                 conv_std_logic_vector(15, 5) when 12 | 13,
                 conv_std_logic_vector(16, 5) when 14 | 15,
                 conv_std_logic_vector(17, 5) when 17 | 18 | 19,
                 conv_std_logic_vector(18, 5) when 20 | 21 | 22,
                 conv_std_logic_vector(19, 5) when 23 | 24 | 25 | 26,
                 conv_std_logic_vector(20, 5) when 27 | 28 | 29 | 30 | 31,
                 conv_std_logic_vector(21, 5) when 33 | 34 | 35 | 36 | 37 | 38,
                 conv_std_logic_vector(22, 5) when 39 | 40 | 41 | 42 | 43 | 44 | 45,
                 conv_std_logic_vector(23, 5) when 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53,
                        conv_std_logic_vector(24, 5) when 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62
                            | 63,
                        conv_std_logic_vector(25, 5) when 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73
                            | 74 | 75 | 76,
                        conv_std_logic_vector(26, 5) when 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85
                            | 86 | 87 | 88 | 89 | 90,
                        conv_std_logic_vector(27, 5) when 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99
                            | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107,
                        conv_std_logic_vector(28, 5) when others;

    -- Wouldn't this be nice?  Xilinx ISE throws the following warning:
    --   Index value(s) does not match array range, simulation mismatch.
    --signal_out <= signal_in(conv_integer(shift)+bw-1 downto conv_integer(shift));

    -- So instead, we're stuck with this.
    signal_out <= signal_in(8+bw-1 downto 8) when shift = 8 else
                        signal_in(12+bw-1 downto 12) when shift = 12 else
                        signal_in(16+bw-1 downto 16) when shift = 16 else
                        signal_in(20+bw-1 downto 20) when shift = 20 else
                        signal_in(24+bw-1 downto 24) when shift = 24 else
                        signal_in(28+bw-1 downto 28) when shift = 28 else
                        signal_in(10+bw-1 downto 10) when shift = 10 else
                        signal_in(11+bw-1 downto 11) when shift = 11 else
                        signal_in(13+bw-1 downto 13) when shift = 13 else
                        signal_in(14+bw-1 downto 14) when shift = 14 else
                        signal_in(15+bw-1 downto 15) when shift = 15 else
                        signal_in(17+bw-1 downto 17) when shift = 17 else
                        signal_in(18+bw-1 downto 18) when shift = 18 else
                        signal_in(19+bw-1 downto 19) when shift = 19 else
                        signal_in(21+bw-1 downto 21) when shift = 21 else
                        signal_in(22+bw-1 downto 22) when shift = 22 else
                        signal_in(23+bw-1 downto 23) when shift = 23 else
                        signal_in(25+bw-1 downto 25) when shift = 25 else
                        signal_in(26+bw-1 downto 26) when shift = 26 else
                        signal_in(27+bw-1 downto 27) when shift = 27 else
                        signal_in(28+bw-1 downto 28);
end behavioral;
```

Listing A.9: CIC interpolator module, part of the tx_chain. Interpolates data in baseband before sending it to the AD9862 for transmission. (cic_interp.vhd)

```
--------------------------------------------------------------------------------
-- CIC Interpolation
-- Last Modified: 20 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
```

```vhdl
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- A CIC interpolator is equivalent to a moving average filter.  Basically,
-- take N equivalent moving average filters (FIR filters), then separate
-- them so that the input goes through all combs first and then all integrators.
-- We can do this because both combs and integrators are LTI.  To think of a
-- moving average filter, imagine that at each step we add the newest sample
-- and remove the oldest sample, then divide by the number of stages in the
-- filter.  Something like this (divided by N).
--    y[n] = y[n-1] + x[n] - x[n-R]
entity cic_interp is
    generic(
        bw                      : integer := 16;     -- # of bits for input
        N                       : integer := 4; -- # of filter stages
        log2_of_max_rate    : integer := 7; -- log2 of max sampling rate
        maxbitgain          : integer := 21 --(N-1)*log2_of_max_rate
    );

    port(
        clock           : in    std_logic;
        reset           : in    std_logic;
        enable      : in    std_logic;
        rate            : in    std_logic_vector(7 downto 0);
        strobe_in   : in    std_logic;
        strobe_out  : in    std_logic;
        signal_in   : in    std_logic_vector(bw-1 downto 0);
        signal_out  : out   std_logic_vector(bw-1 downto 0)
    );
end cic_interp;

architecture behavioral of cic_interp is
    component sign_extend is
        generic(
            bits_in : integer := 0;
            bits_out    : integer := 0
        );

        port(
            d_in    : in    std_logic_vector(bits_in-1 downto 0);
            d_out   : out   std_logic_vector(bits_out-1 downto 0)
        );
    end component;

    component cic_int_shifter is
        generic(
            bw            : integer := 16;
            maxbitgain    : integer := 21
        );

        port(
            rate          : in    std_logic_vector(7 downto 0);
            signal_in     : in    std_logic_vector(bw+maxbitgain-1 downto 0);
            signal_out    : out   std_logic_vector(bw-1 downto 0)
        );
    end component;

    type mem is array (integer range <>) of std_logic_vector(bw+maxbitgain-1 downto 0);
    signal integrator : mem(0 to N-1);
    signal differentiator : mem(0 to N-1);
    signal pipeline : mem(0 to N-1);

    signal signal_in_ext : std_logic_vector(bw+maxbitgain-1 downto 0);

    signal clear_me : std_logic;
    signal signal_out_unnorm : std_logic_vector(bw+maxbitgain-1 downto 0);
begin
    -- Sign extend the input signal to account for bit gain in the filters.
    ext_input : sign_extend
        generic map(bits_in => bw, bits_out => (bw+maxbitgain))
        port map(d_in => signal_in, d_out => signal_in_ext);

    -- A simple OR gate to reset the filters.
    clear_me <= '1' when ((reset = '1') or (enable = '0')) else '0';

    -- FIXME Note that this section has pipe and diff reversed.
    -- It still works, but is confusing.

    -- Comb Filter
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(clear_me = '1') then
                -- Clear the comb filters.
                for i in 0 to N-1 loop
                    differentiator(i) <= (others => '0');
                    pipeline(i) <= (others => '0');
                end loop;
            elsif((enable = '1') and (strobe_in = '1')) then
                -- The comb filters remove the oldest samples from the
                -- moving average.
                -- c[n] = x[n] - x[n - R]
                differentiator(0) <= signal_in_ext;
                pipeline(0) <= signal_in_ext - differentiator(0);
                for i in 1 to N-1 loop
                    differentiator(i) <= pipeline(i-1);
                    pipeline(i) <= pipeline(i-1) - differentiator(i);
                end loop;
            end if;
        end if;
    end process;

    -- Integrator
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(clear_me = '1') then
                -- Clear the integrator.
                for i in 0 to N-1 loop
                    integrator(i) <= (others => '0');
                end loop;
            elsif((enable = '1') and (strobe_out = '1')) then
                if(strobe_in = '1') then
                    -- Grab the output of the comb filters into the integrator.
                    integrator(0) <= integrator(0) + pipeline(N-1);
                end if;

                -- Integrate! The integrator adds the newest sample to the
                -- moving average.
                -- y[n] = y[n-1] + c[n]
                for i in 1 to N-1 loop
                    integrator(i) <= integrator(i) + integrator(i-1);
                end loop;
            end if;
        end if;
    end process;

    -- Normalize the output to 'bw' bits.
    signal_out_unnorm <= integrator(N-1);

    cic_int_shifter0 : cic_int_shifter
        generic map(bw => bw)
        port map(rate => rate, signal_in => signal_out_unnorm, signal_out => signal_out);
end behavioral;
```

Listing A.10: Interpolation shifter to account for bitgain in the interpolation process. (cic_int_shifter.vhd)

```
--------------------------------------------------------------------------------
-- CIC Interpolation Shifter
-- Last Modified: 20 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
-- GNU General Public License for more details.
--
-- You should have received a copy of the GNU General Public License
-- along with this program; if not, write to the Free Software
-- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
--------------------------------------------------------------------------------

-- This only works for N=4, max interp rate of 128.
-- Signal "rate" is ONE LESS THAN the actual rate.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity cic_int_shifter is
    generic(
        bw          : integer := 16;
        maxbitgain  : integer := 21
    );

    port(
        rate        : in    std_logic_vector(7 downto 0);
        signal_in   : in    std_logic_vector(bw+maxbitgain-1 downto 0);
        signal_out  : out   std_logic_vector(bw-1 downto 0)
    );
end cic_int_shifter;

architecture behavioral of cic_int_shifter is
    signal shift : std_logic_vector(4 downto 0);
    signal rate1 : std_logic_vector(7 downto 0);
begin
    -- I don't know why the original Verilog source uses rate-1 in the top
    -- module and then fixes it here. But this is really the interpolation
    -- rate.
    rate1 <= rate + 1;

    -- The actual bit gain is determined by the rate, where
    --    bitgain = (N-1) * log2(R)
    -- We fix N at 4 here and R is equal to 'rate'.
    with conv_integer(rate1) select
        shift <= conv_std_logic_vector(6, 5) when 4,
                   conv_std_logic_vector(9, 5) when 8,
                   conv_std_logic_vector(12, 5) when 16,
                   conv_std_logic_vector(15, 5) when 32,
                   conv_std_logic_vector(18, 5) when 64,
                   conv_std_logic_vector(21, 5) when 128,
                   conv_std_logic_vector(7, 5) when 5,
                   conv_std_logic_vector(8, 5) when 6,
                   conv_std_logic_vector(9, 5) when 7,
                   conv_std_logic_vector(10, 5) when 9 | 10,
                   conv_std_logic_vector(11, 5) when 11 | 12,
                   conv_std_logic_vector(12, 5) when 13 | 14 | 15,
                   conv_std_logic_vector(13, 5) when 17 | 18 | 19 | 20,
                   conv_std_logic_vector(14, 5) when 21 | 22 | 23 | 24 | 25,
                   conv_std_logic_vector(15, 5) when 26 | 27 | 28 | 29 | 30 | 31,
                        conv_std_logic_vector(16, 5) when 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40,
                        conv_std_logic_vector(17, 5) when 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49
                            | 50,
                        conv_std_logic_vector(18, 5) when 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59
                            | 60 | 61 | 62 | 63,
                        conv_std_logic_vector(19, 5) when 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73
                            | 74 | 75 | 76 | 77 | 78 | 79 | 80,
                        conv_std_logic_vector(20, 5) when 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89
                            | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 | 101,
                        conv_std_logic_vector(21, 5) when others;

        -- Wouldn't this be nice?  Xilinx ISE throws the following warning:
        --   Index value(s) does not match array range, simulation mismatch.
        --signal_out <= signal_in(conv_integer(shift)+bw-1 downto conv_integer(shift));

        -- So instead, we're stuck with this.
        signal_out <= signal_in(6+bw-1 downto 6) when shift = 6 else
                        signal_in(9+bw-1 downto 9) when shift = 9 else
                        signal_in(12+bw-1 downto 12) when shift = 12 else
                        signal_in(15+bw-1 downto 15) when shift = 15 else
                        signal_in(18+bw-1 downto 18) when shift = 18 else
                        signal_in(21+bw-1 downto 21) when shift = 21 else
                        signal_in(7+bw-1 downto 7) when shift = 7 else
                        signal_in(8+bw-1 downto 8) when shift = 8 else
                        signal_in(10+bw-1 downto 10) when shift = 10 else
                        signal_in(11+bw-1 downto 11) when shift = 11 else
                        signal_in(13+bw-1 downto 13) when shift = 13 else
                        signal_in(14+bw-1 downto 14) when shift = 14 else
                        signal_in(16+bw-1 downto 16) when shift = 16 else
                        signal_in(17+bw-1 downto 17) when shift = 17 else
                        signal_in(19+bw-1 downto 19) when shift = 19 else
                        signal_in(20+bw-1 downto 20) when shift = 20 else
                        signal_in(21+bw-1 downto 21);
end behavioral;
```

Listing A.11: A basic clock divider module. (clk_divider.vhd)

```
--------------------------------------------------------------------------------
-- Clock Divider
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.
--
-- You should have received a copy of the GNU General Public License
-- along with this program; if not, write to the Free Software
-- Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity clk_divider is
    port(
        reset      : in    std_logic;
        in_clk   : in    std_logic;
        out_clk  : out   std_logic;
        ratio      : in    std_logic_vector(7 downto 0)
    );
end clk_divider;

architecture behavioral of clk_divider is
    -- Original Verilog had [7:0], but it was wrong.
    signal counter : std_logic_vector(7 downto 0);
    signal out_clk_t : std_logic;
    signal count_tmp : std_logic;
begin
    out_clk <= out_clk_t;
    count_tmp <= '1' when ((ratio(0) = '1') and (out_clk_t = '1')) else '0';
    -- Use a down counter to count clocks between each toggle of out_clk.
    process(in_clk, reset)
    begin
        if(reset = '1') then
            counter <= (others => '0');
        elsif(in_clk'EVENT and in_clk = '1') then
            if(counter = 0) then
                -- If counter is zero here, we are diving in_clk by 2.
                -- This ugly notation forces XST to generate a carry adder like it should.
                counter <= conv_std_logic_vector(conv_integer(ratio(7 downto 1)) +
                     conv_integer(count_tmp) - 1, 8);
            else
                counter <= counter - "00000001";
            end if;
        end if;
    end process;

    -- Toggle out_clock when the counter hits zero.
    process(in_clk, reset)
    begin
        if(reset = '1') then
            out_clk_t <= '0';
        elsif(in_clk'EVENT and in_clk = '1') then
            if(counter = 0) then
                out_clk_t <= (not out_clk_t);
            end if;
        end if;
    end process;
end behavioral;
```

Listing A.12: Memory for storing the halfband filter coefficients. (coeff_rom.vhd)

```
----------------------------------------------------------------------------------
-- Coefficient ROM for the Halfband Decimation Filter
-- Last Modified: 14 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
----------------------------------------------------------------------------------

-- TODO:  Migrate this module to the new numeric_std library standard.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity coeff_rom is
    port(
        clock   : in    std_logic;
        addr    : in    std_logic_vector(2 downto 0);
        data    : out   std_logic_vector(15 downto 0)
    );
end coeff_rom;

architecture behavioral of coeff_rom is
    signal data_t : signed(15 downto 0);
begin
    data <= conv_std_logic_vector(data_t, 16);

    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
```

```
            case conv_integer(addr) is
                when 0 =>
                    data_t <= conv_signed(-49, 16);
                when 1 =>
                    data_t <= conv_signed(165, 16);
                when 2 =>
                    data_t <= conv_signed(-412, 16);
                when 3 =>
                    data_t <= conv_signed(873, 16);
                when 4 =>
                    data_t <= conv_signed(-1681, 16);
                when 5 =>
                    data_t <= conv_signed(3135, 16);
                when 6 =>
                    data_t <= conv_signed(-6282, 16);
                when 7 =>
                    data_t <= conv_signed(20628, 16);
                when others =>
            end case;
        end if;
    end process;
end behavioral;
```

Listing A.13: The next six files are configuration headers, implemented in VHDL as packages. This particular file contains information common to all USRP configurations. (common_config.vhd)

```
--------------------------------------------------------------------------------
-- USRP Common Configuration
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------

-- This is the common tail for standard configuration.  Uncomment a single
-- Tx/Rx configuration in the 'use' section.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

library WORK;
```

```
use WORK.FUNCS.ALL;

-- Configurations go here.  Uncomment only one configuration!
use WORK.COMMON_CONFIG_1RXHB_1TX.ALL;
--use WORK.COMMON_CONFIG_2RX_0TX.ALL;
--use WORK.COMMON_CONFIG_2RXHB_0TX.ALL;
--use WORK.COMMON_CONFIG_2RXHB_2TX.ALL;
--use WORK.COMMON_CONFIG_4RX_0TX.ALL;

-- You should not need to edit below this line!
package common_config is
    -- *ALL* of the remaining code should be conditionalized only in terms of:
    --   TX_EN, TX_EN_0, TX_EN_1, TX_EN_2, TX_EN_3, TX_CAP_NCHAN, TX_CAP_HB,
    --   RX_EN, RX_EN_0, RX_EN_1, RX_EN_2, RX_EN_3, RX_CAP_NCHAN, RX_CAP_HB,
    --   RX_NCO_EN, RX_CIC_EN

    constant TX_EN : boolean := TX_ON;
    constant TX_EN_0 : boolean := (TX_EN and (TX_SINGLE or TX_DUAL or TX_QUAD));
    constant TX_EN_1 : boolean := (TX_EN and (TX_DUAL or TX_QUAD));
    constant TX_EN_2 : boolean := (TX_EN and TX_QUAD);
    constant TX_EN_3 : boolean := (TX_EN and TX_QUAD);
    constant TX_CAP_HB : boolean := (TX_EN and TX_HB_ON);
    constant TX_CAP_NCHAN : std_logic_vector(2 downto 0) := (conv_std_logic(TX_EN and TX_QUAD)
                                                                                  &
                                                                                      conv_std_logic(TX_EN
                                                                                      and TX_DUAL)
                                                                                  &
                                                                                      conv_std_logic(TX_EN
                                                                                      and TX_SINGLE));

    constant RX_EN : boolean := RX_ON;
    constant RX_EN_0 : boolean := (RX_EN and (RX_SINGLE or RX_DUAL or RX_QUAD));
    constant RX_EN_1 : boolean := (RX_EN and (RX_DUAL or RX_QUAD));
    constant RX_EN_2 : boolean := (RX_EN and RX_QUAD);
    constant RX_EN_3 : boolean := (RX_EN and RX_QUAD);
    constant RX_CAP_HB : boolean := (RX_EN and RX_HB_ON);
    constant RX_CAP_NCHAN : std_logic_vector(2 downto 0) := (conv_std_logic(RX_EN and RX_QUAD)
                                                                                  &
                                                                                      conv_std_logic(RX_EN
                                                                                      and RX_DUAL)
                                                                                  &
                                                                                      conv_std_logic(RX_EN
                                                                                      and RX_SINGLE));

    constant RX_NCO_EN : boolean := (RX_EN and RX_NCO_ON);
    constant RX_CIC_EN : boolean := (RX_EN and RX_CIC_ON);
end common_config;
```

Listing A.14: Configuration for one transmit channel and one receive channel with halfband filter. (common_config_1rxhb_1tx.vhd)

```
-------------------------------------------------------------------------------
-- USRP Configuration with 1 Rx (Halfband) and 1 Tx
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
```

```
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
-------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

package common_config_1rxhb_1tx is
    -- If TX_ON is not defined, there is *no* transmit circuitry built.
    constant TX_ON : boolean := true;

    -- Define 1 and only 1 of TX_SINGLE, TX_DUAL, and TX_QUAD to respectively
    -- enable 1, 2, or 4 transmit channels.
    -- [Please note that only TX_SINGLE and TX_DUAL are currently valid.]
    constant TX_SINGLE : boolean := true;
    constant TX_DUAL : boolean := false;
    constant TX_QUAD : boolean := false;

    -- Define TX_HB_ON to enable the transmit halfband filter.
    -- [not implemented]
    constant TX_HB_ON : boolean := false;

    -- If RX_ON is not defined, there is *no* receive circuitry built.
    constant RX_ON : boolean := true;

    -- Define 1 and only 1 of RX_SINGLE, RX_DUAL, and TX_QUAD to respectively
    -- enable 1, 2, or 4 receive channels.
    constant RX_SINGLE : boolean := true;
    constant RX_DUAL : boolean := false;
    constant RX_QUAD : boolean := false;

    -- Define RX_HB_ON to enable the receive halfband filter.
    constant RX_HB_ON : boolean := true;

    -- Define RX_NCO_ON to enable the receive Numerical Controlled Osc.
    constant RX_NCO_ON : boolean := true;

    -- Define RX_CIC_ON to enable the receive Cascaded Integrator Comb filter.
    constant RX_CIC_ON : boolean := true;
end common_config_1rxhb_1tx;
```

Listing A.15: Configuration for zero transmit channels and two receive channel with half-band filters. (common_config_2rxhb_0tx.vhd)

```
--------------------------------------------------------------------------------
-- USRP Configuration with 2 Rx (Halfband) and 0 Tx
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

package common_config_2rxhb_0tx is
    -- If TX_ON is not defined, there is *no* transmit circuitry built.
    constant TX_ON : boolean := false;

    -- Define 1 and only 1 of TX_SINGLE, TX_DUAL, and TX_QUAD to respectively
    -- enable 1, 2, or 4 transmit channels.
    -- [Please note that only TX_SINGLE and TX_DUAL are currently valid.]
    constant TX_SINGLE : boolean := false;
    constant TX_DUAL : boolean := false;
    constant TX_QUAD : boolean := false;

    -- Define TX_HB_ON to enable the transmit halfband filter.
    -- [not implemented]
    constant TX_HB_ON : boolean := false;

    -- If RX_ON is not defined, there is *no* receive circuitry built.
    constant RX_ON : boolean := true;

    -- Define 1 and only 1 of RX_SINGLE, RX_DUAL, and TX_QUAD to respectively
    -- enable 1, 2, or 4 receive channels.
    constant RX_SINGLE : boolean := false;
    constant RX_DUAL : boolean := true;
    constant RX_QUAD : boolean := false;

    -- Define RX_HB_ON to enable the receive halfband filter.
    constant RX_HB_ON : boolean := true;

    -- Define RX_NCO_ON to enable the receive Numerical Controlled Osc.
    constant RX_NCO_ON : boolean := true;

    -- Define RX_CIC_ON to enable the receive Cascaded Integrator Comb filter.
    constant RX_CIC_ON : boolean := true;
end common_config_2rxhb_0tx;
```

Listing A.16: Configuration for two transmit channels and two receive channels with half-band filters. (common_config_2rxhb_2tx.vhd)

```
---------------------------------------------------------------------------------------
-- USRP Configuration with 2 Rx (Halfband) and 2 Tx
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
---------------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

package common_config_2rxhb_2tx is
    -- If TX_ON is not defined, there is *no* transmit circuitry built.
    constant TX_ON : boolean := true;

    -- Define 1 and only 1 of TX_SINGLE, TX_DUAL, and TX_QUAD to respectively
    -- enable 1, 2, or 4 transmit channels.
    -- [Please note that only TX_SINGLE and TX_DUAL are currently valid.]
    constant TX_SINGLE : boolean := false;
    constant TX_DUAL : boolean := true;
    constant TX_QUAD : boolean := false;

    -- Define TX_HB_ON to enable the transmit halfband filter.
    -- [not implemented]
    constant TX_HB_ON : boolean := false;

    -- If RX_ON is not defined, there is *no* receive circuitry built.
    constant RX_ON : boolean := true;

    -- Define 1 and only 1 of RX_SINGLE, RX_DUAL, and TX_QUAD to respectively
    -- enable 1, 2, or 4 receive channels.
    constant RX_SINGLE : boolean := false;
    constant RX_DUAL : boolean := true;
    constant RX_QUAD : boolean := false;

    -- Define RX_HB_ON to enable the receive halfband filter.
    constant RX_HB_ON : boolean := true;

    -- Define RX_NCO_ON to enable the receive Numerical Controlled Osc.
    constant RX_NCO_ON : boolean := true;

    -- Define RX_CIC_ON to enable the receive Cascaded Integrator Comb filter.
    constant RX_CIC_ON : boolean := true;
end common_config_2rxhb_2tx;
```

Listing A.17: Configuration for zero transmit channels and two receive channels with no halfband filters. (common_config_2rx_0tx.vhd)

```
----------------------------------------------------------------------------------
-- USRP Configuration with 2 Rx and 0 Tx
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
--
--   This program is free software; you can redistribute it and/or modify
--   it under the terms of the GNU General Public License as published by
--   the Free Software Foundation; either version 2 of the License, or
--   (at your option) any later version.
--
--   This program is distributed in the hope that it will be useful,
--   but WITHOUT ANY WARRANTY; without even the implied warranty of
--   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--   GNU General Public License for more details.
--
--   You should have received a copy of the GNU General Public License
--   along with this program; if not, write to the Free Software
--   Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

package common_config_2rx_0tx is
    -- If TX_ON is not defined, there is *no* transmit circuitry built.
    constant TX_ON : boolean := false;

    -- Define 1 and only 1 of TX_SINGLE, TX_DUAL, and TX_QUAD to respectively
    -- enable 1, 2, or 4 transmit channels.
    -- [Please note that only TX_SINGLE and TX_DUAL are currently valid.]
    constant TX_SINGLE : boolean := false;
    constant TX_DUAL : boolean := false;
    constant TX_QUAD : boolean := false;

    -- Define TX_HB_ON to enable the transmit halfband filter.
    -- [not implemented]
    constant TX_HB_ON : boolean := false;

    -- If RX_ON is not defined, there is *no* receive circuitry built.
    constant RX_ON : boolean := true;

    -- Define 1 and only 1 of RX_SINGLE, RX_DUAL, and TX_QUAD to respectively
    -- enable 1, 2, or 4 receive channels.
    constant RX_SINGLE : boolean := false;
    constant RX_DUAL : boolean := true;
    constant RX_QUAD : boolean := false;

    -- Define RX_HB_ON to enable the receive halfband filter.
    constant RX_HB_ON : boolean := false;

    -- Define RX_NCO_ON to enable the receive Numerical Controlled Osc.
    constant RX_NCO_ON : boolean := true;

    -- Define RX_CIC_ON to enable the receive Cascaded Integrator Comb filter.
    constant RX_CIC_ON : boolean := true;
end common_config_2rx_0tx;
```

Listing A.18: Configuration for zero transmit channels and four receive channels with no halfband filters. (common_config_4rx_0tx.vhd)

```
-------------------------------------------------------------------------------
-- USRP Configuration with 4 Rx and 0 Tx
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
-------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

package common_config_4rx_0tx is
    -- If TX_ON is not defined, there is *no* transmit circuitry built.
    constant TX_ON : boolean := false;

    -- Define 1 and only 1 of TX_SINGLE, TX_DUAL, and TX_QUAD to respectively
    -- enable 1, 2, or 4 transmit channels.
    -- [Please note that only TX_SINGLE and TX_DUAL are currently valid.]
    constant TX_SINGLE : boolean := false;
    constant TX_DUAL : boolean := false;
    constant TX_QUAD : boolean := false;

    -- Define TX_HB_ON to enable the transmit halfband filter.
    -- [not implemented]
    constant TX_HB_ON : boolean := false;

    -- If RX_ON is not defined, there is *no* receive circuitry built.
    constant RX_ON : boolean := true;

    -- Define 1 and only 1 of RX_SINGLE, RX_DUAL, and TX_QUAD to respectively
    -- enable 1, 2, or 4 receive channels.
    constant RX_SINGLE : boolean := false;
    constant RX_DUAL : boolean := false;
    constant RX_QUAD : boolean := true;

    -- Define RX_HB_ON to enable the receive halfband filter.
    constant RX_HB_ON : boolean := false;

    -- Define RX_NCO_ON to enable the receive Numerical Controlled Osc.
    constant RX_NCO_ON : boolean := true;

    -- Define RX_CIC_ON to enable the receive Cascaded Integrator Comb filter.
    constant RX_CIC_ON : boolean := true;
end common_config_4rx_0tx;
```

Listing A.19: Cordic numerically-controlled oscillator (NCO) module for the complex multiplier that translates received data from intermediate frequency to baseband. (cordic.vhd)

```
--------------------------------------------------------------------------------
-- Cordic
-- Last Modified: 20 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

```

```vhdl
library WORK;
use WORK.FUNCS.ALL;

-- This is the first stage in the Rx chain.  We move the signal from IF band
-- to baseband.
-- NOTE:  The Spartan3A 1400 actually has hardware multipliers and we might be
-- able to use them instead of CORDIC.
entity cordic is
    generic(
        bitwidth    : integer := 16;
        zwidth  : integer := 16
    );

    port(
        clock       : in    std_logic;
        reset       : in    std_logic;
        enable  : in    std_logic;
        xi          : in    std_logic_vector(bitwidth-1 downto 0);
        yi          : in    std_logic_vector(bitwidth-1 downto 0);
        xo          : out   std_logic_vector(bitwidth-1 downto 0);
        yo          : out   std_logic_vector(bitwidth-1 downto 0);
        zi          : in    std_logic_vector(zwidth-1 downto 0);
        zo          : out   std_logic_vector(zwidth-1 downto 0)
    );
end cordic;

architecture behavioral of cordic is
    -- FIXME Need to somehow loop to generate constants...
    constant STAGES : integer := 17;
    type const_rom is array(0 to STAGES-1) of integer;
    constant consts : const_rom := (
        8192, 4836, 2555, 1297, 651, 326, 163, 81, 41, 20, 10, 5, 3, 1, 1, 0, 0);

    component cordic_stage is
        generic(
            bitwidth    : integer := 16;
            zwidth  : integer := 16;
            shift       : integer := 1
        );

        port(
            clock       : in    std_logic;
            reset       : in    std_logic;
            enable  : in    std_logic;
            xi          : in    std_logic_vector(bitwidth-1 downto 0);
            yi          : in    std_logic_vector(bitwidth-1 downto 0);
            zi          : in    std_logic_vector(zwidth-1 downto 0);
            const       : in    std_logic_vector(zwidth-1 downto 0);
            xo          : out   std_logic_vector(bitwidth-1 downto 0);
            yo          : out   std_logic_vector(bitwidth-1 downto 0);
            zo          : out   std_logic_vector(zwidth-1 downto 0)
        );
    end component;

    -- FIXME This should probably be variable, too.
    type xy_vec is array(0 to 12) of std_logic_vector(bitwidth+1 downto 0);
    type z_vec is array(0 to 12) of std_logic_vector(zwidth-2 downto 0);
    signal x : xy_vec;
    signal y : xy_vec;
    signal z : z_vec;

    signal xi_ext : std_logic_vector(bitwidth+1 downto 0);
    signal yi_ext : std_logic_vector(bitwidth+1 downto 0);

    signal zi_t : std_logic_vector(1 downto 0);
begin
    -- Sign extend the inputs!
    xi_ext <= repeat(2, xi(bitwidth-1)) & xi;
    yi_ext <= repeat(2, yi(bitwidth-1)) & yi;

    -- Quadrant
    zi_t <= zi(zwidth-1 downto zwidth-2);

    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(reset = '1') then
                -- Clear the first stage inputs.
                x(0) <= (others => '0');
                y(0) <= (others => '0');
                z(0) <= (others => '0');
            elsif(enable = '1') then
                -- The first two bits are the quadrant, so take the rest here.
                -- Z is the phase.
                z(0) <= zi(zwidth-2 downto 0);

                -- X and Y are the I and Q components of the sample.
                case zi_t is
                    when "00" | "11" =>
                        x(0) <= xi_ext;
                        y(0) <= yi_ext;
                    when "01" | "10" =>
                        x(0) <= -xi_ext;
                        y(0) <= -yi_ext;
                    when others =>
                        null;
                end case;
            end if;
        end if;
    end process;

    -- FIXME loop should be variable, not strict 0:11.
    -- FIXME should be able to narrow zwidth?
    cordic_stages : for i in 0 to 11 generate
        cs : cordic_stage
            generic map(bitwidth+2, zwidth-1, i)
            port map(clock => clock, reset => reset, enable => enable, xi => x(i), yi => y(i), zi
                => z(i),
                const => conv_std_logic_vector(consts(i), zwidth-1), xo => x(i+1), yo => y(i+1),
                    zo => z(i+1));
    end generate cordic_stages;

    xo <= x(12)(bitwidth downto 1);
    yo <= y(12)(bitwidth downto 1);
    zo <= '0' & z(12);
end behavioral;
```

Listing A.20: One stage of the Cordic NCO. (cordic_stage.vhd)

```
-------------------------------------------------------------------------------
-- A Single Cordic Stage
-- Last Modified: 10 April 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--   Copyright (C) 2003 Matt Ettus
--
--   This program is free software; you can redistribute it and/or modify
--   it under the terms of the GNU General Public License as published by
--   the Free Software Foundation; either version 2 of the License, or
--   (at your option) any later version.
--
--   This program is distributed in the hope that it will be useful,
--   but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--   You should have received a copy of the GNU General Public License
--   along with this program; if not, write to the Free Software
--   Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
-------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library WORK;
use WORK.FUNCS.ALL;

entity cordic_stage is
    generic(
        bitwidth    : integer := 16;
        zwidth      : integer := 16;
        shift       : integer := 1
    );

    port(
        clock   : in    std_logic;
        reset   : in    std_logic;
        enable  : in    std_logic;
        xi      : in    std_logic_vector(bitwidth-1 downto 0);
        yi      : in    std_logic_vector(bitwidth-1 downto 0);
        zi      : in    std_logic_vector(zwidth-1 downto 0);
        const   : in    std_logic_vector(zwidth-1 downto 0);
        xo      : out   std_logic_vector(bitwidth-1 downto 0);
        yo      : out   std_logic_vector(bitwidth-1 downto 0);
        zo      : out   std_logic_vector(zwidth-1 downto 0)
    );
end cordic_stage;

architecture behavioral of cordic_stage is
    signal z_is_pos : std_logic;
begin
    z_is_pos <= (not zi(zwidth-1));

    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(reset = '1') then
                -- Clear the outputs on reset.
                xo <= (others => '0');
                yo <= (others => '0');
                zo <= (others => '0');
            elsif(enable = '1') then
                -- Basically, sign-extend and shift the inputs, then add/subtract as necessary
                -- to get the outputs.
                if(z_is_pos = '1') then
                    -- Z is positive, move counter-clockwise.
                    xo <= xi - (repeat(shift+1, yi(bitwidth-1)) & yi(bitwidth-2 downto shift));
                    yo <= yi + (repeat(shift+1, xi(bitwidth-1)) & xi(bitwidth-2 downto shift));
                    zo <= zi - const;
                else
                    -- Z is negative, move clockwise.
                    xo <= xi + (repeat(shift+1, yi(bitwidth-1)) & yi(bitwidth-2 downto shift));
                    yo <= yi - (repeat(shift+1, xi(bitwidth-1)) & xi(bitwidth-2 downto shift));
                    zo <= zi + const;
                end if;
            end if;
        end if;
    end process;
end behavioral;
```

Listing A.21: First-in-first-out (FIFO) buffer, used for both transmit and receive data between the USB2 port and the CIC filters. (fifo.vhd)

```
------------------------------------------------------------------------------
-- FIFO
-- Last Modified: 20 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity fifo is
    generic(
        width          : integer := 16;
        depth          : integer := 1024;
        addr_bits    : integer := 10
    );

    port(
        data        : in    std_logic_vector(width-1 downto 0);
        wrreq       : in    std_logic;
        rdreq       : in    std_logic;
        rdclk       : in    std_logic;
        wrclk       : in    std_logic;
        aclr        : in    std_logic;
        q           : out   std_logic_vector(width-1 downto 0);
        rdfull   : out   std_logic;
        rdempty  : out   std_logic;
        rdusedw  : out   std_logic_vector(addr_bits-1 downto 0);
        wrfull   : out   std_logic;
        wrempty  : out   std_logic;
        wrusedw  : out   std_logic_vector(addr_bits-1 downto 0)
    );
end fifo;

architecture behavioral of fifo is
    -- Set this to false for rd_ack, true for rd_req.  Choosing false here
    -- forces the tools to use LUTs for the FIFO since we'll have asynchronous
    -- read operations.  Choosing true uses BRAMs instead!  The original Verilog
    -- source had false, though.
    constant RD_REQ : boolean := true;

    -- The actual buffer
    type RAM is array(integer range <>) of std_logic_vector(width-1 downto 0);
    signal mem : RAM(0 to depth-1);

    -- Pointers for read and write
    signal rdptr : std_logic_vector(addr_bits-1 downto 0);
    signal wrptr : std_logic_vector(addr_bits-1 downto 0);

    -- Provide read access to outputs with temp signals!
    signal rdusedw_t : std_logic_vector(addr_bits-1 downto 0);
    signal wrusedw_t : std_logic_vector(addr_bits-1 downto 0);
begin
    -- Equate the temp signals with the real outputs.
    rdusedw <= rdusedw_t;
    wrusedw <= wrusedw_t;

    -- Write one line of data, increase the write pointer.
    process(wrclk, aclr)
    begin
        if(wrclk'EVENT and wrclk = '1') then
            if(aclr = '1') then
                -- Reset the pointer on aclr.
                wrptr <= (others => '0');
            elsif(wrreq = '1') then
                wrptr <= wrptr + 1;
                mem(conv_integer(wrptr)) <= data;
            end if;
        end if;
    end process;

    -- If using the read request model, output on request and increment
    -- the read pointer.
    rd_req_t : if(RD_REQ) generate
        process(rdclk, aclr)
        begin
            if(rdclk'EVENT and rdclk = '1') then
                if(aclr = '1') then
                    rdptr <= (others => '0');
                elsif(rdreq = '1') then
                    rdptr <= rdptr + 1;
                    q <= mem(conv_integer(rdptr));
                end if;
            end if;
        end process;
    end generate rd_req_t;

    -- If using the read ack model, always output and update the pointer on
    -- the clock (ack).
    rd_req_f : if(not RD_REQ) generate
        process(rdclk, aclr)
        begin
            if(rdclk'EVENT and rdclk = '1') then
                if(aclr = '1') then
                    -- Reset the pointer on aclr.
                    rdptr <= (others => '0');
                elsif(rdreq = '1') then
                    rdptr <= rdptr + 1;
                end if;
            end if;
        end process;

        q <= mem(conv_integer(rdptr));
    end generate rd_req_f;

    -- Honestly, I don't know why some of these exist.  The original Verilog
    -- just said "fix these".  As far as I can tell, they're never used.
    process(wrclk)
    begin
        if(wrclk'EVENT and wrclk = '1') then
            wrusedw_t <= conv_std_logic_vector(unsigned(wrptr - rdptr), wrusedw_t'LENGTH);
        end if;
    end process;

    process(rdclk)
    begin
        if(rdclk'EVENT and rdclk = '1') then
            rdusedw_t <= conv_std_logic_vector(unsigned(wrptr - rdptr), rdusedw_t'LENGTH);
        end if;
    end process;

    wrempty <= '1' when (wrusedw_t = 0) else '0';
    wrfull <= '1' when (wrusedw_t = (depth-1)) else '0';

    rdempty <= '1' when (rdusedw_t = 0) else '0';
    rdfull <= '1' when (rdusedw_t = (depth-1)) else '0';
end behavioral;
```

Listing A.22: A particular configuration of the FIFO module, 4096 lines of 18 bits each. (fifo_4k_18.vhd)

```
--------------------------------------------------------------------------------
-- FIFO (4k lines, 18-bit width)
-- Last Modified: 20 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--   Copyright (C) 2003 Matt Ettus
--
--   This program is free software; you can redistribute it and/or modify
--   it under the terms of the GNU General Public License as published by
--   the Free Software Foundation; either version 2 of the License, or
--   (at your option) any later version.
--
--   This program is distributed in the hope that it will be useful,
--   but WITHOUT ANY WARRANTY; without even the implied warranty of
--   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--   GNU General Public License for more details.
--
--   You should have received a copy of the GNU General Public License
--   along with this program; if not, write to the Free Software
--   Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fifo_4k_18 is
    port(
        data        : in    std_logic_vector(17 downto 0);
        wrreq       : in    std_logic;
        wrclk       : in    std_logic;
        wrfull  : out   std_logic;
        wrempty : out   std_logic;
        wrusedw : out   std_logic_vector(11 downto 0);
        q           : out   std_logic_vector(17 downto 0);
        rdreq       : in    std_logic;
        rdclk       : in    std_logic;
        rdfull  : out   std_logic;
        rdempty : out   std_logic;
        rdusedw : out   std_logic_vector(11 downto 0);
        aclr        : in    std_logic
    );
end fifo_4k_18;

architecture behavioral of fifo_4k_18 is
    component fifo is
        generic(
            width       : integer := 16;
            depth       : integer := 1024;
            addr_bits   : integer := 10
        );

        port(
            data        : in    std_logic_vector(width-1 downto 0);
            wrreq       : in    std_logic;
            rdreq       : in    std_logic;
            rdclk       : in    std_logic;
            wrclk       : in    std_logic;
            aclr        : in    std_logic;
            q           : out   std_logic_vector(width-1 downto 0);
            rdfull  : out   std_logic;
            rdempty : out   std_logic;
            rdusedw : out   std_logic_vector(addr_bits-1 downto 0);
            wrfull  : out   std_logic;
            wrempty : out   std_logic;
            wrusedw : out   std_logic_vector(addr_bits-1 downto 0)
        );
    end component;
begin
    fifo_4k : fifo
        generic map(width => 18, depth => 4096, addr_bits => 12)
        port map(data => data, wrreq => wrreq, rdreq => rdreq, rdclk => rdclk, wrclk => wrclk,
            aclr => aclr, q => q, rdfull => rdfull, rdempty => rdempty, rdusedw => rdusedw,
            wrfull => wrfull, wrempty => wrempty, wrusedw => wrusedw);
end behavioral;
```

Listing A.23: Defines some of the configurations registers for the USRP. (fpga_regs_common.vhd)

```
--------------------------------------------------------------------------------
-- Definitions Common for all FPGA Configurations (Registers 0-31)
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
---------------------------------------------------------------------------

-- This file is derived from the original Verilog file, which was automatically
-- generated.  Someone should update the GNU Radio code to automatically
-- generate VHDL files, too!
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

package fpga_regs_common is
    -- The FPGA needs to know the rate that samples are coming from and
    -- going to the A/D's and D/A's.  div = 128e6 / sample_rate
    constant FR_TX_SAMPLE_RATE_DIV : integer := 0;
    constant FR_RX_SAMPLE_RATE_DIV : integer := 1;

    -- 2 and 3 are defined in the ATR section

    -- Master enable and reset controls
    constant FR_MASTER_CTRL : integer := 4;

    -- i/o direction registers for pins that go to daughterboards.
    -- Setting the bit makes it an output from the FPGA to the d'board.
    -- top 16 is mask, low 16 is value
    constant FR_OE_0 : integer := 5;    -- slot 0
    constant FR_OE_1 : integer := 6;
    constant FR_OE_2 : integer := 7;
    constant FR_OE_3 : integer := 8;

    -- i/o registers for pins that go to daughterboards.
    -- top 16 is a mask, low 16 is value
    constant FR_IO_0 : integer := 9;    -- slot 0
    constant FR_IO_1 : integer := 10;
    constant FR_IO_2 : integer := 11;
    constant FR_IO_3 : integer := 12;

    constant FR_MODE : integer := 13;

    -- If the corresponding bit is set, internal FPGA debug circuitry
    -- controls the i/o pins for the associated bank of daughterboard
    -- i/o pins.  Typically used for debugging FPGA designs.
    constant FR_DEBUG_EN : integer := 14;

    -- If the corresponding bit is set, enable the automatic DC
    -- offset correction control loop.
    --
    -- The 4 low bits are significant:
    --
    --   ADC0 = (1 << 0)
    --   ADC1 = (1 << 1)
    --   ADC2 = (1 << 2)
    --   ADC3 = (1 << 3)
    --
    -- This control loop works if the attached daugherboard blocks DC.
    -- Currently all daughterboards do block DC.  This includes:
    -- basic rx, dbs_rx, tv_rx, flex_xxx_rx.
    -- DC Offset Control Loop Enable
    constant FR_DC_OFFSET_CL_EN : integer := 15;

    -- offset corrections for ADC's and DAC's (2's complement)
    constant FR_ADC_OFFSET_0 : integer := 16;
    constant FR_ADC_OFFSET_1 : integer := 17;
    constant FR_ADC_OFFSET_2 : integer := 18;
    constant FR_ADC_OFFSET_3 : integer := 19;

    -- ------------------------------------------------------------------------
    -- Automatic Transmit/Receive switching
    --
    -- If automatic transmit/receive (ATR) switching is enabled in the
    -- FR_ATR_CTL register, the presence or absence of data in the FPGA
    -- transmit fifo selects between two sets of values for each of the 4
    -- banks of daughterboard i/o pins.
    --
    -- Each daughterboard slot has 3 16-bit registers associated with it:
    --   FR_ATR_MASK_*, FR_ATR_TXVAL_* and FR_ATR_RXVAL_*
    --
    -- FR_ATR_MASK_{0,1,2,3}:
    --
    --   These registers determine which of the daugherboard i/o pins are
    --   affected by ATR switching.  If a bit in the mask is set, the
    --   corresponding i/o bit is controlled by ATR, else it's output
    --   value comes from the normal i/o pin output register:
    --   FR_IO_{0,1,2,3}.
    --
    -- FR_ATR_TXVAL_{0,1,2,3}:
    -- FR_ATR_RXVAL_{0,1,2,3}:
    --
    --   If the Tx fifo contains data, then the bits from TXVAL that are
    --   selected by MASK are output.  Otherwise, the bits from RXVAL that
    --   are selected by MASK are output.
    constant FR_ATR_MASK_0 : integer := 20; -- slot 0
    constant FR_ATR_TXVAL_0 : integer := 21;
    constant FR_ATR_RXVAL_0 : integer := 22;

    constant FR_ATR_MASK_1 : integer := 23; -- slot 1
    constant FR_ATR_TXVAL_1 : integer := 24;
    constant FR_ATR_RXVAL_1 : integer := 25;

    constant FR_ATR_MASK_2 : integer := 26; -- slot 2
    constant FR_ATR_TXVAL_2 : integer := 27;
    constant FR_ATR_RXVAL_2 : integer := 28;

    constant FR_ATR_MASK_3 : integer := 29; -- slot 3
    constant FR_ATR_TXVAL_3 : integer := 30;
    constant FR_ATR_RXVAL_3 : integer := 31;

    -- Clock ticks to delay rising and falling edge of T/R signal
    constant FR_ATR_TX_DELAY : integer := 2;
    constant FR_ATR_RX_DELAY : integer := 3;
end fpga_regs_common;
```

Listing A.24: Defines some of the configurations registers for the USRP. (fpga_regs_standard.vhd)

```
--------------------------------------------------------------------------------
-- Definitions for standard FPGA build (regs 32+) and custom (64-95).
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------

-- This file is derived from the original Verilog file, which was automatically
-- generated.  Someone should update the GNU Radio code to automatically
-- generate VHDL files, too!
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

package fpga_regs_standard is
```

```
    -- DDC / DUC
    constant FR_INTERP_RATE : integer := 32;    -- [1,1024]
    constant FR_DECIM_RATE : integer := 33; -- [1,256]

    -- DDC center freq
    constant FR_RX_FREQ_0 : integer := 34;
    constant FR_RX_FREQ_1 : integer := 35;
    constant FR_RX_FREQ_2 : integer := 36;
    constant FR_RX_FREQ_3 : integer := 37;

    -- See below for DDC Starting Phase

    ----------------------------------------------------------------------------
    --  configure FPGA Rx mux
    --
    --    3                   2                   1
    --  1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
    -- +-----------------------+-------+-------+-------+-------+-+-----+
    -- |      must be zero     | Q3| I3| Q2| I2| Q1| I1| Q0| I0|Z| NCH |
    -- +-----------------------+-------+-------+-------+-------+-+-----+
    --
    -- There are a maximum of 4 digital downconverters in the the FPGA.
    -- Each DDC has two 16-bit inputs, I and Q, and two 16-bit outputs, I & Q.
    --
    -- DDC I inputs are specified by the two bit fields I3, I2, I1 & I0
    --
    --   0 = DDC input is from ADC 0
    --   1 = DDC input is from ADC 1
    --   2 = DDC input is from ADC 2
    --   3 = DDC input is from ADC 3
    --
    -- If Z == 1, all DDC Q inputs are set to zero
    -- If Z == 0, DDC Q inputs are specified by the two bit fields Q3, Q2, Q1 & Q0
    --
    -- NCH specifies the number of complex channels that are sent across
    -- the USB.  The legal values are 1, 2 or 4, corresponding to 2, 4 or
    -- 8 16-bit values.
    constant FR_RX_MUX : integer := 38;

    ----------------------------------------------------------------------------
    --  configure FPGA Tx Mux.
    --
    --    3                   2                   1
    --  1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
    -- +-----------------------+-------+-------+-------+-------+-+-----+
    -- |                       | DAC3  | DAC2  | DAC1  |  DAC0 |0| NCH |
    -- +-----------------------------------------------+-------+-+-----+
    --
    -- NCH specifies the number of complex channels that are sent across
    -- the USB.  The legal values are 1 or 2, corresponding to 2 or 4
    -- 16-bit values.
    --
    -- There are two interpolators with complex inputs and outputs.
    -- There are four DACs.  (We use the DUC in each AD9862.)
    --
    -- Each 4-bit DACx field specifies the source for the DAC and
    -- whether or not that DAC is enabled.  Each subfield is coded
```

```
-- like this:
--
--     3 2 1 0
--    +-+-----+
--    |E|  N  |
--    +-+-----+
--
-- Where E is set if the DAC is enabled, and N specifies which
-- interpolator output is connected to this DAC.
--
--  N   which interp output
-- ---  -------------------
--  0   chan 0 I
--  1   chan 0 Q
--  2   chan 1 I
--  3   chan 1 Q
constant FR_TX_MUX : integer := 39;

-------------------------------------------------------------------------
-- REFCLK control
--
-- Control whether a reference clock is sent to the daughterboards,
-- and what frequency.  The refclk is sent on d'board i/o pin 0.
--
--    3                   2                   1
--  1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
-- +-----------------------------------------------+-+------------+
-- |             Reserved (Must be zero)           |E|   DIVISOR  |
-- +-----------------------------------------------+-+------------+
--
-- Bit 7  -- 1 turns on refclk, 0 allows IO use
-- Bits 6:0 Divider value
constant FR_TX_A_REFCLK : integer := 40;
constant FR_RX_A_REFCLK : integer := 41;
constant FR_TX_B_REFCLK : integer := 42;
constant FR_RX_B_REFCLK : integer := 43;

--------------------------------------------------------------------------
-- DDC Starting Phase
constant FR_RX_PHASE_0 : integer := 44;
constant FR_RX_PHASE_1 : integer := 45;
constant FR_RX_PHASE_2 : integer := 46;
constant FR_RX_PHASE_3 : integer := 47;

--------------------------------------------------------------------------
-- Tx data format control register
--
--    3                   2                   1
--  1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
-- +-------------------------------------------------------+-------+
-- |                   Reserved (Must be zero)             |  FMT  |
-- +-------------------------------------------------------+-------+
--
--  FMT values:
constant FR_TX_FORMAT : integer := 48;

--------------------------------------------------------------------------
```

```
-- Rx data format control register
--
--      3                    2                    1
--   1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
-- +-----------------------------------------+-+-+---------+-------+
-- |              Reserved (Must be zero)    |B|Q|  WIDTH  | SHIFT |
-- +-----------------------------------------+-+-+---------+-------+
--
--  FMT values:
constant FR_RX_FORMAT : integer := 49;

-- The valid combinations currently are:
--
--    B  Q  WIDTH  SHIFT
--    0  1    16     0
--    0  1     8     8

-- Possible future values of WIDTH = {4, 2, 1}
-- 12 takes a bit more work, since we need to know packet alignment.

---------------------------------------------------------------------------
-- FIXME register numbers 50 to 63 are available

---------------------------------------------------------------------------
-- Registers 64 to 95 are reserved for user custom FPGA builds.
-- The standard USRP software will not touch these.
constant FR_USER_0 : integer := 64;
constant FR_USER_1 : integer := 65;
constant FR_USER_2 : integer := 66;
constant FR_USER_3 : integer := 67;
constant FR_USER_4 : integer := 68;
constant FR_USER_5 : integer := 69;
constant FR_USER_6 : integer := 70;
constant FR_USER_7 : integer := 71;
constant FR_USER_8 : integer := 72;
constant FR_USER_9 : integer := 73;
constant FR_USER_10 : integer := 74;
constant FR_USER_11 : integer := 75;
constant FR_USER_12 : integer := 76;
constant FR_USER_13 : integer := 77;
constant FR_USER_14 : integer := 78;
constant FR_USER_15 : integer := 79;
constant FR_USER_16 : integer := 80;
constant FR_USER_17 : integer := 81;
constant FR_USER_18 : integer := 82;
constant FR_USER_19 : integer := 83;
constant FR_USER_20 : integer := 84;
constant FR_USER_21 : integer := 85;
constant FR_USER_22 : integer := 86;
constant FR_USER_23 : integer := 87;
constant FR_USER_24 : integer := 88;
constant FR_USER_25 : integer := 89;
constant FR_USER_26 : integer := 90;
constant FR_USER_27 : integer := 91;
constant FR_USER_28 : integer := 92;
constant FR_USER_29 : integer := 93;
constant FR_USER_30 : integer := 94;
constant FR_USER_31 : integer := 95;

--Registers needed for multi usrp master/slave configuration
--
--Rx Master/slave control register (FR_RX_MASTER_SLAVE = FR_USER_0)
--
constant FR_RX_MASTER_SLAVE : integer := 64;
constant bitnoFR_RX_SYNC : integer := 0;
constant bitnoFR_RX_SYNC_MASTER : integer := 1;
constant bitnoFR_RX_SYNC_SLAVE : integer := 2;

--Caution The master settings will output values on the io lines.
--They inheritely enable these lines as output. If you have a daughtercard which uses these
     lines also as output then you will burn your usrp and daughtercard.
--If you set the slave bits then your usrp won't do anything if you don't connect a master.
-- Rx Master/slave control register
--
-- The way this is supposed to be used is connecting a (short) 16pin flatcable from an rx
     daughterboard in RXA master io_rx[8..15] to slave io_rx[8..15] on RXA of slave usrp
-- This can be done with basic_rx boards or dbsrx boards
--dbsrx: connect master-J25 to slave-J25
--basic rx: connect J25 to slave-J25
--CAUTION: pay attention to the lineup of your connector.
--The red line (pin1) should be at the same side of the daughterboards on master and slave.
--If you turnaround the cable on one end you will burn your usrp.

--You cannot use a 16pin flatcable if you are using FLEX400 or FLEX2400 daughterboards, since
     these use a lot of the io pins.
--You can still link them but you must use only a 2pin or 1pin cable
--You can also use a 2-wire link. put a 2pin header on io[15],gnd of the master RXA
     daughterboard and connect it to io15,gnd of the slave RXA db.
--You can use a cable like the ones found with the leds on the mainbord of a PC.
--Make sure you don't twist the cable, otherwise you connect the sync output to ground.
--To be save you could also just use a single wire from master io[15] to slave io[15], but
     this is not optimal for signal integrity.


-- Since rx_io[0] can normally be used as a refclk and is not exported on all daughterboards
     this line
-- still has the refclk function if you use the master/slave setup (it is not touched by the
     master/slave settings).
-- The master/slave circuitry will only use io pin 15 and does not touch any of the other io
     pins.
constant bitnoFR_RX_SYNC_INPUT_IOPIN : integer := 15;
constant bmFR_RX_SYNC_INPUT_IOPIN : integer := 2**bitnoFR_RX_SYNC_INPUT_IOPIN; -- FIXME ?
--TODO the output pin is still hardcoded in the verilog code, make it listen to the following
     define
constant bitnoFR_RX_SYNC_OUTPUT_IOPIN : integer := 15;
constant bmFR_RX_SYNC_OUTPUT_IOPIN : integer := 2**bitnoFR_RX_SYNC_OUTPUT_IOPIN;

-- ======================================================================
-- READBACK Registers
-- ======================================================================
-- read back a-side i/o pins
constant FR_RB_IO_RX_A_IO_TX_A : integer := 1;
-- read back b-side i/o pins
constant FR_RB_IO_RX_B_IO_TX_B : integer := 2;
```

```

    ------------------------------------------------------------------------------
    -- FPGA Capability register
    --
    --      3                   2                   1
    --    1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
    -- +-----------------------------------------------+-+-----+-+-----+
    -- |                     Reserved (Must be zero)   |T|NDUC |R|NDDC |
    -- +-----------------------------------------------+-+-----+-+-----+
    --
    -- Bottom 4-bits are Rx capabilities
    -- Next   4-bits are Tx capabilities
    constant FR_RB_CAPS : integer := 3;
end fpga_regs_standard;
```

Listing A.25: This package contains a few functions that VHDL lacks, but which make code easier to read and write. (funcs.vhd)

```
--------------------------------------------------------------------------------
-- A collection of useful functions that should really be built into VHDL!
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------

-- This file should probably not exist.  I think that most of these functions
-- are available in numeric_std and numeric_extra, which we should eventually
-- use instead of std_logic_arith.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

package funcs is
    -- Mimics the Verilog replication operator.  For example, repeat(3, '1') returns "111".
    function repeat (N : natural; B : std_logic) return std_logic_vector;

    -- Converts a 'signed' logic vector into a 'std_logic_vector' logic vector of equal width.
    function conv_std_logic_vector(S: signed) return std_logic_vector;

    -- Converts a 'std_logic_vector' logic vector into a 'signed' logic vector of equal width.
    function conv_signed(S: std_logic_vector) return signed;

    -- Converts a 'boolean' to a 'std_logic' element, where 'true' = '1' and 'false' = '0'.
    function conv_std_logic(B: boolean) return std_logic;
end funcs;

package body funcs is
    function repeat(N: natural; B: std_logic)
        return std_logic_vector
    is
    begin
        return (N-1 downto 0 => B);
    end;

    function conv_std_logic_vector(S: signed)
        return std_logic_vector
    is
        variable tmp : std_logic_vector(S'RANGE);
    begin
        for i in S'RANGE loop
            tmp(i) := S(i);
        end loop;
        return tmp;
    end;

    function conv_signed(S: std_logic_vector)
        return signed
    is
        variable tmp : signed(S'RANGE);
    begin
        for i in S'RANGE loop
            tmp(i) := S(i);
        end loop;
        return tmp;
    end;

    function conv_std_logic(B: boolean)
        return std_logic
    is
    begin
        if(B) then
            return '1';
        else
            return '0';
        end if;
    end;
end funcs;
```

Listing A.26: Halfband decimation filter. Decimates baseband data by a factor of two. (halfband_decim.vhd)

```
-------------------------------------------------------------------------------
-- Halfband Decimator
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
```

```
--
--  USRP - Universal Software Radio Peripheral
--
--   Copyright (C) 2003 Matt Ettus
--
--   This program is free software; you can redistribute it and/or modify
--   it under the terms of the GNU General Public License as published by
--   the Free Software Foundation; either version 2 of the License, or
--   (at your option) any later version.
--
--   This program is distributed in the hope that it will be useful,
--   but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--   You should have received a copy of the GNU General Public License
--   along with this program; if not, write to the Free Software
--   Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
---------------------------------------------------------------------------------

-- This implements a 31-tap halfband filter that decimates by two.
-- The coefficients are symmetric, and with the exception of the middle tap,
-- every other coefficient is zero.  The middle section of taps looks like this:
--
--   ..., -1468, 0, 2950, 0, -6158, 0, 20585, 32768, 20585, 0, -6158, 0, 2950, 0, -1468, ...
--                                              |
--                                 middle tap -------+
--
-- See coeff_rom.vhd for the full set.  The taps are scaled relative to 32768,
-- thus the middle tap equals 1.0.  Not counting the middle tap, there are 8
-- non-zero taps on each side, and they are symmetric.  A naive implementation
-- requires a mulitply for each non-zero tap.  Because of symmetry, we can
-- replace 2 multiplies with 1 add and 1 multiply.  Thus, to compute each output
-- sample, we need to perform 8 multiplications.  Since the middle tap is 1.0,
-- we just add the corresponding delay line value.
--
-- About timing: We implement this with a single multiplier, so it takes
-- 8 cycles to compute a single output.  However, since we're decimating by two
-- we can accept a new input value every 4 cycles.  strobe_in is asserted when
-- there's a new input sample available.  Depending on the overall decimation
-- rate, strobe_in may be asserted less frequently than once every 4 clocks.
-- On the output side, we assert strobe_out when output contains a new sample.
--
-- Implementation: Every time strobe_in is asserted we store the new data into
-- the delay line.  We split the delay line into two components, one for the
-- even samples, and one for the odd samples.  ram16_odd is the delay line for
-- the odd samples.  This ram is written on each odd assertion of strobe_in, and
-- is read on each clock when we're computing the dot product.  ram16_even is
-- similar, although because it holds the even samples we must be able to read
-- two samples from different addresses at the same time, while writing the incoming
-- even samples. Thus it's "triple-ported".
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library WORK;
```

```
use WORK.REDUCE_PACK.ALL;
use WORK.FUNCS.ALL;

entity halfband_decim is
    port(
        clock           : in    std_logic;
        reset           : in    std_logic;
        enable      : in    std_logic;
        strobe_in   : in    std_logic;
        strobe_out  : out   std_logic;
        data_in     : in    std_logic_vector(15 downto 0);
        data_out        : out   std_logic_vector(15 downto 0);
        debugctrl   : out   std_logic_vector(15 downto 0)
    );
end halfband_decim;

architecture behavioral of halfband_decim is
    component coeff_rom is
        port(
            clock   : in    std_logic;
            addr    : in    std_logic_vector(2 downto 0);
            data    : out   std_logic_vector(15 downto 0)
        );
    end component;

    component ram16_2sum is
        port(
            clock       : in    std_logic;
            write_in    : in    std_logic;
            wr_addr : in    std_logic_vector(3 downto 0);
            wr_data : in    std_logic_vector(15 downto 0);
            rd_addr1    : in    std_logic_vector(3 downto 0);
            rd_addr2    : in    std_logic_vector(3 downto 0);
            sum     : out   std_logic_vector(15 downto 0)
        );
    end component;

    component ram16 is
        port(
            clock       : in    std_logic;
            write_in    : in    std_logic;
            wr_addr : in    std_logic_vector(3 downto 0);
            wr_data : in    std_logic_vector(15 downto 0);
            rd_addr : in    std_logic_vector(3 downto 0);
            rd_data : out   std_logic_vector(15 downto 0)
        );
    end component;

    component mult is
        port(
            clock           : in    std_logic;
            x               : in    std_logic_vector(15 downto 0);
            y               : in    std_logic_vector(15 downto 0);
            product     : out   std_logic_vector(30 downto 0);
            enable_in   : in    std_logic;
            enable_out  : out   std_logic
        );
    end component;

    component acc is
        port(
            clock           : in    std_logic;
            reset           : in    std_logic;
            clear           : in    std_logic;
            enable_in    : in    std_logic;
            enable_out   : out   std_logic;
            addend       : in    std_logic_vector(30 downto 0);
            sum          : out   std_logic_vector(33 downto 0)
        );
    end component;

    signal rd_addr1 : std_logic_vector(3 downto 0);
    signal rd_addr2 : std_logic_vector(3 downto 0);
    signal phase : std_logic_vector(3 downto 0);
    signal base_addr : std_logic_vector(3 downto 0);

    --signal mac_out : signed(15 downto 0);
    signal middle_data : signed(15 downto 0);
    signal sum : signed(15 downto 0);
    signal coeff : signed(15 downto 0);
    signal product : signed(30 downto 0);
    signal sum_even : signed(33 downto 0);

    -- VHDL doesn't support conversion operators on outputs, so we need
    -- std_logic_vector versions of the signed signals.
    --signal mac_out_t : std_logic_vector(15 downto 0);
    signal middle_data_t : std_logic_vector(15 downto 0);
    signal sum_t : std_logic_vector(15 downto 0);
    signal coeff_t : std_logic_vector(15 downto 0);
    signal product_t : std_logic_vector(30 downto 0);
    signal sum_even_t : std_logic_vector(33 downto 0);

    signal clear : std_logic;
    signal store_odd : std_logic;

    signal start : std_logic;
    signal start_d1 : std_logic;
    signal start_d2 : std_logic;
    signal start_d3 : std_logic;
    signal start_d4 : std_logic;
    signal start_d5 : std_logic;
    signal start_d6 : std_logic;
    signal start_d7 : std_logic;
    signal start_d8 : std_logic;
    signal start_d9 : std_logic;
    signal start_dA : std_logic;
    signal start_dB : std_logic;
    signal start_dC : std_logic;
    signal start_dD : std_logic;

    signal mult_en : std_logic;
    signal mult_en_pre : std_logic;

    signal latch_result : std_logic;
    signal acc_en : std_logic;

    signal dout : signed(33 downto 0);

    signal crom_addr : std_logic_vector(2 downto 0);
    signal write_in_even : std_logic;
    signal write_in_odd : std_logic;
    signal rd_addr_odd : std_logic_vector(3 downto 0);

    signal strobe_out_t : std_logic;
begin
    -- Flip between odd and even with each strobe.
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(reset = '1') then
                store_odd <= '0';
            elsif(strobe_in = '1') then
                store_odd <= (not store_odd);
            end if;
        end if;
    end process;

    start <= '1' when ((strobe_in = '1') and (store_odd = '1')) else '0';

    --
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(reset = '1') then
                base_addr <= (others => '0');
            elsif(start = '1') then
                base_addr <= base_addr + 1;
            end if;
        end if;
    end process;

    -- Phase control logic.
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(reset = '1') then
                phase <= conv_std_logic_vector(8, 4);
            elsif(start = '1') then
                phase <= conv_std_logic_vector(0, 4);
            elsif(phase /= 8) then
                phase <= phase + 1;
            end if;
        end if;
    end process;

    -- Recall that each signal is updated concurrently!
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            start_d1 <= start;
            start_d2 <= start_d1;
            start_d3 <= start_d2;
            start_d4 <= start_d3;
            start_d5 <= start_d4;
            start_d6 <= start_d5;
            start_d7 <= start_d6;
            start_d8 <= start_d7;
            start_d9 <= start_d8;
            start_dA <= start_d9;
            start_dB <= start_dA;
            start_dC <= start_dB;
            start_dD <= start_dC;
        end if;
    end process;

    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(phase /= 8) then
                mult_en_pre <= '1';
            else
                mult_en_pre <= '0';
            end if;

            mult_en <= mult_en_pre;
        end if;
    end process;

    -- These comments were in the original Verilog source.  I copied them here.
    clear <= start_d4;  -- was dC
    latch_result <= start_d4;   -- was dC
    strobe_out_t <= start_d5;   -- was dD

    -- The two values we read depend on the phase.
    process(phase, base_addr)
    begin
        case conv_integer(phase(2 downto 0)) is
            when 0 =>
                rd_addr1 <= base_addr + 0;
                rd_addr2 <= base_addr + 15;
            when 1 =>
                rd_addr1 <= base_addr + 1;
                rd_addr2 <= base_addr + 14;
            when 2 =>
                rd_addr1 <= base_addr + 2;
                rd_addr2 <= base_addr + 13;
            when 3 =>
                rd_addr1 <= base_addr + 3;
                rd_addr2 <= base_addr + 12;
            when 4 =>
                rd_addr1 <= base_addr + 4;
                rd_addr2 <= base_addr + 11;
            when 5 =>
                rd_addr1 <= base_addr + 5;
                rd_addr2 <= base_addr + 10;
            when 6 =>
                rd_addr1 <= base_addr + 6;
                rd_addr2 <= base_addr + 9;
            when 7 =>
                rd_addr1 <= base_addr + 7;
                rd_addr2 <= base_addr + 8;
            when others =>
                rd_addr1 <= base_addr + 0;
                rd_addr2 <= base_addr + 15;
        end case;
    end process;

    -- Equate the temporaries with the signed values.
    --mac_out <= conv_signed(mac_out_t);
    middle_data <= conv_signed(middle_data_t);
    sum <= conv_signed(sum_t);
    coeff <= conv_signed(coeff_t);
    product <= conv_signed(product_t);
    sum_even <= conv_signed(sum_even_t);

    crom_addr <= phase(2 downto 0) - 1;

    coeff_rom0 : coeff_rom
        port map(clock => clock, addr => crom_addr, data => coeff_t);

    write_in_even <= '1' when ((strobe_in = '1') and (store_odd = '0')) else '0';

    ram16_even : ram16_2sum
        port map(clock => clock, write_in => write_in_even, wr_addr => base_addr, wr_data =>
             data_in,
            rd_addr1 => rd_addr1, rd_addr2 => rd_addr2, sum => sum_t);

    write_in_odd <= '1' when ((strobe_in = '1') and (store_odd = '1')) else '0';
    rd_addr_odd <= base_addr + 6;

    ram16_odd : ram16   -- holds middle items
        port map(clock => clock, write_in => write_in_odd, wr_addr => base_addr, wr_data =>
             data_in,
            rd_addr => rd_addr_odd, rd_data => middle_data_t);

    mult0 : mult
        port map(clock => clock, x => coeff_t, y => sum_t,
            product => product_t, enable_in => mult_en, enable_out => acc_en);

    acc0 : acc
        port map(clock => clock, reset => reset, enable_in => acc_en, enable_out => OPEN, clear
             => clear,
            addend => product_t, sum => sum_even_t);

    dout <= sum_even + conv_signed((repeat(4, middle_data(15)) &
         conv_std_logic_vector(middle_data) & repeat(14, '0')));

    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(reset = '1') then
                data_out <= (others => '0');
            elsif(latch_result = '1') then
                if((dout(33) = '1') and (or_reduce(conv_std_logic_vector(dout(14 downto 0))) =
                     '1')) then
                    data_out <= conv_std_logic_vector(dout(30 downto 15) + 1);
                else
                    data_out <= conv_std_logic_vector(dout(30 downto 15));
                end if;
            end if;
        end if;
    end process;

    strobe_out <= strobe_out_t;
    debugctrl <= repeat(3, '0') & clock & reset & acc_en & mult_en & clear & latch_result &
        store_odd & strobe_in & strobe_out_t & phase;
end behavioral;
```

Listing A.27: Controller for the I/O pins that attach to the daughtercards. (io_pins.vhd)

```
--------------------------------------------------------------------------------
-- Tri-State I/O Pins for Daughtercard Headers
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library WORK;
use WORK.FPGA_REGS_COMMON.ALL;

entity io_pins is
    port(
        io_0                : inout std_logic_vector(15 downto 0);
        io_1                : inout std_logic_vector(15 downto 0);
        io_2                : inout std_logic_vector(15 downto 0);
        io_3              : inout std_logic_vector(15 downto 0);
        reg_0             : in    std_logic_vector(15 downto 0);
        reg_1             : in    std_logic_vector(15 downto 0);
        reg_2             : in    std_logic_vector(15 downto 0);
        reg_3             : in    std_logic_vector(15 downto 0);
        clock             : in    std_logic;
        rx_reset          : in    std_logic;
        tx_reset          : in    std_logic;
        serial_addr     : in      std_logic_vector(6 downto 0);
        serial_data     : in      std_logic_vector(31 downto 0);
        serial_strobe   : in      std_logic
    );
end io_pins;

architecture behavioral of io_pins is
    component bidir_reg is
        port(   tristate    : inout std_logic_vector(15 downto 0);
                oe          : in    std_logic_vector(15 downto 0);
                reg_val : in      std_logic_vector(15 downto 0)
        );
    end component;

    signal io_0_oe : std_logic_vector(15 downto 0);
    signal io_1_oe : std_logic_vector(15 downto 0);
    signal io_2_oe : std_logic_vector(15 downto 0);
    signal io_3_oe : std_logic_vector(15 downto 0);
begin
    bidir_reg_0 : bidir_reg
        port map( tristate => io_0, oe => io_0_oe, reg_val => reg_0);

    bidir_reg_1 : bidir_reg
        port map( tristate => io_1, oe => io_1_oe, reg_val => reg_1);

    bidir_reg_2 : bidir_reg
        port map( tristate => io_2, oe => io_2_oe, reg_val => reg_2);

    bidir_reg_3 : bidir_reg
        port map( tristate => io_3, oe => io_3_oe, reg_val => reg_3);

    -- The upper 16 bits are a mask for the lower 16 bits.  If a bit in the
    -- upper 16 bits is high, set the output bit to the value of the
    -- corresponding bit in the lower 16 bits.  Otherwise, do nothing.
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(serial_strobe = '1') then
                case conv_integer(serial_addr) is
                    when FR_OE_0 =>
                        io_0_oe <= ((io_0_oe and (not serial_data(31 downto 16))) or
                                    (serial_data(15 downto 0) and serial_data(31 downto 16)));
                    when FR_OE_1 =>
                        io_1_oe <= ((io_1_oe and (not serial_data(31 downto 16))) or
                                    (serial_data(15 downto 0) and serial_data(31 downto 16)));
                    when FR_OE_2 =>
                        io_2_oe <= ((io_2_oe and (not serial_data(31 downto 16))) or
                                    (serial_data(15 downto 0) and serial_data(31 downto 16)));
                    when FR_OE_3 =>
```

```
                            io_3_oe <= ((io_3_oe and (not serial_data(31 downto 16))) or
                                        (serial_data(15 downto 0) and serial_data(31 downto 16)));
                    when others =>
                        null;
                end case;
            end if;
        end if;
    end process;
end behavioral;
```

Listing A.28: Master control module. Provides enables, clocks, and serial data for the entire system. Also contains most of the configuration registers. (master_control.vhd)

```
--------------------------------------------------------------------------------
-- Master Control System (clocks, resets, enables, etc.)
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library WORK;
use WORK.FPGA_REGS_COMMON.ALL;
use WORK.FPGA_REGS_STANDARD.ALL;
use WORK.FUNCS.ALL;

entity master_control is
    port(
        master_clk          : in    std_logic;
        usbclk              : in    std_logic;
        serial_addr         : in    std_logic_vector(6 downto 0);
        serial_data         : in    std_logic_vector(31 downto 0);
        serial_strobe       : in    std_logic;
        tx_bus_reset        : out   std_logic;
        rx_bus_reset        : out   std_logic;
        tx_dsp_reset        : out   std_logic;
        rx_dsp_reset        : out   std_logic;
        enable_tx           : out   std_logic;
        enable_rx           : out   std_logic;
        interp_rate         : out   std_logic_vector(7 downto 0);
        decim_rate          : out   std_logic_vector(7 downto 0);
        tx_sample_strobe    : out   std_logic;
        strobe_interp       : out   std_logic;
        rx_sample_strobe    : out   std_logic;
        strobe_decim        : out   std_logic;
        tx_empty                : in    std_logic;
        debug_0             : in    std_logic_vector(15 downto 0);
        debug_1             : in    std_logic_vector(15 downto 0);
        debug_2             : in    std_logic_vector(15 downto 0);
        debug_3             : in    std_logic_vector(15 downto 0);
        reg_0                   : out   std_logic_vector(15 downto 0);
        reg_1                   : out   std_logic_vector(15 downto 0);
        reg_2                   : out   std_logic_vector(15 downto 0);
        reg_3                   : out   std_logic_vector(15 downto 0)
    );
end master_control;

architecture behavioral of master_control is
    component setting_reg is
        generic(
            my_addr : integer := 0
        );

        port(
            clock       : in    std_logic;
            reset       : in    std_logic;
            strobe  : in    std_logic;
            addr        : in    std_logic_vector;
            d_in        : in    std_logic_vector(31 downto 0);
            d_out       : out   std_logic_vector(31 downto 0);
            changed : out   std_logic
        );
    end component;

    component strobe_gen is
        port(
            clock           : in    std_logic;
            reset           : in    std_logic;
            enable      : in    std_logic;
            rate            : in    std_logic_vector(7 downto 0);
            strobe_in   : in    std_logic;
            strobe      : out   std_logic
        );
    end component;

    component clk_divider is
        port(
            reset       : in    std_logic;
            in_clk  : in    std_logic;
            out_clk : out   std_logic;
            ratio       : in    std_logic_vector(7 downto 0)
        );
    end component;

    component atr_delay is
        port(
            clk_i           : in    std_logic;
            rst_i           : in    std_logic;
            ena_i           : in    std_logic;
            tx_empty_i  : in    std_logic;
            tx_delay_i  : in    std_logic_vector(11 downto 0);
            rx_delay_i  : in    std_logic_vector(11 downto 0);
            atr_tx_o        : out   std_logic
        );
    end component;

    -- FIXME need a separate reset for all control settings
    signal master_controls : std_logic_vector(7 downto 0);

    -- Internals so we can read/write.
    signal tx_dsp_reset_t : std_logic;
    signal rx_dsp_reset_t : std_logic;
    signal dsp_reset_t : std_logic;
    signal enable_tx_t : std_logic;
    signal enable_rx_t : std_logic;

    signal interp_rate_t : std_logic_vector(7 downto 0);
    signal decim_rate_t : std_logic_vector(7 downto 0);
    signal tx_sample_strobe_t : std_logic;
    signal rx_sample_strobe_t : std_logic;

    -- USB reset signals
    signal tx_reset_bus_sync1 : std_logic;
    signal rx_reset_bus_sync1 : std_logic;
    signal tx_reset_bus_sync2 : std_logic;
    signal rx_reset_bus_sync2 : std_logic;

    -- Internal clocks
    signal txa_refclk : std_logic_vector(7 downto 0);
    signal rxa_refclk : std_logic_vector(7 downto 0);
    signal txb_refclk : std_logic_vector(7 downto 0);
    signal rxb_refclk : std_logic_vector(7 downto 0);

    signal txaclk : std_logic;
    signal rxaclk : std_logic;
    signal txbclk : std_logic;
    signal rxbclk : std_logic;

    -- ???
    signal debug_en : std_logic_vector(3 downto 0);
    signal txcvr_ctrl : std_logic_vector(3 downto 0);

    signal txcvr_txlines : std_logic_vector(31 downto 0);
    signal txcvr_rxlines : std_logic_vector(31 downto 0);

    -- Internal signals for daughtercard headers
    signal io_0_reg : std_logic_vector(15 downto 0);
    signal io_1_reg : std_logic_vector(15 downto 0);
    signal io_2_reg : std_logic_vector(15 downto 0);
    signal io_3_reg : std_logic_vector(15 downto 0);

    -- ATR stuff (auto transmit/receive switch)
    signal transmit_now : std_logic;
    signal atr_ctl : std_logic;
    signal atr_tx_delay : std_logic_vector(11 downto 0);
    signal atr_rx_delay : std_logic_vector(11 downto 0);
    signal atr_mask_0 : std_logic_vector(15 downto 0);
    signal atr_txval_0 : std_logic_vector(15 downto 0);
    signal atr_rxval_0 : std_logic_vector(15 downto 0);
    signal atr_mask_1 : std_logic_vector(15 downto 0);
    signal atr_txval_1 : std_logic_vector(15 downto 0);
    signal atr_rxval_1 : std_logic_vector(15 downto 0);
    signal atr_mask_2 : std_logic_vector(15 downto 0);
    signal atr_txval_2 : std_logic_vector(15 downto 0);
    signal atr_rxval_2 : std_logic_vector(15 downto 0);
    signal atr_mask_3 : std_logic_vector(15 downto 0);
    signal atr_txval_3 : std_logic_vector(15 downto 0);
    signal atr_rxval_3 : std_logic_vector(15 downto 0);

    signal atr_selected_0 : std_logic_vector(15 downto 0);
    signal atr_selected_1 : std_logic_vector(15 downto 0);
    signal atr_selected_2 : std_logic_vector(15 downto 0);
    signal atr_selected_3 : std_logic_vector(15 downto 0);

    -- More daughtercard header signals?
    signal io_0 : std_logic_vector(15 downto 0);
    signal io_1 : std_logic_vector(15 downto 0);
    signal io_2 : std_logic_vector(15 downto 0);
    signal io_3 : std_logic_vector(15 downto 0);
begin
    -- Assign internal signals to their outputs.
    enable_tx <= enable_tx_t;
    enable_rx <= enable_rx_t;
    tx_dsp_reset <= tx_dsp_reset_t;
    rx_dsp_reset <= rx_dsp_reset_t;
    dsp_reset_t <= (tx_dsp_reset_t or rx_dsp_reset_t);
    interp_rate <= interp_rate_t;
    decim_rate <= decim_rate_t;
    tx_sample_strobe <= tx_sample_strobe_t;
    rx_sample_strobe <= rx_sample_strobe_t;

    -- Master control assignments.
    sr_mstr_ctrl : setting_reg
        generic map(my_addr => FR_MASTER_CTRL)
        port map(clock => master_clk, reset => '0', strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 8) => OPEN,
            d_out(7 downto 0) => master_controls, changed => OPEN);

    -- Control signals 4:7 are not used.
    enable_tx_t <= master_controls(0);
    enable_rx_t <= master_controls(1);
    tx_dsp_reset_t <= master_controls(2);
    rx_dsp_reset_t <= master_controls(3);

    -- Strobe Generators
    sr_interp : setting_reg
        generic map(my_addr => FR_INTERP_RATE)
        port map(clock => master_clk, reset => tx_dsp_reset_t, strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 8) => OPEN,
            d_out(7 downto 0) => interp_rate_t, changed => OPEN);

    sr_decim : setting_reg
        generic map(my_addr => FR_DECIM_RATE)
        port map(clock => master_clk, reset => rx_dsp_reset_t, strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 8) => OPEN,
            d_out(7 downto 0) => decim_rate_t, changed => OPEN);

    rx_sample_strobe_t <= '1';

    da_strobe_gen : strobe_gen
        port map(clock => master_clk, reset => tx_dsp_reset_t, enable => enable_tx_t,
            rate => (others => '1'), strobe_in => '1', strobe => tx_sample_strobe_t);

    tx_strobe_gen : strobe_gen
        port map(clock => master_clk, reset => tx_dsp_reset_t, enable => enable_tx_t,
            rate => interp_rate_t, strobe_in => tx_sample_strobe_t, strobe => strobe_interp);

    decim_strobe_gen : strobe_gen
        port map(clock => master_clk, reset => rx_dsp_reset_t, enable => enable_rx_t,
            rate => decim_rate_t, strobe_in => rx_sample_strobe_t, strobe => strobe_decim);

    -- Reset syncs for bus (usbclk) side
    -- The RX bus side reset isn't used, the TX bus side one may not be needed.
    process(usbclk)
    begin
        if(usbclk'EVENT and usbclk = '1') then
            tx_reset_bus_sync1 <= tx_dsp_reset_t;
            rx_reset_bus_sync1 <= rx_dsp_reset_t;
            tx_reset_bus_sync2 <= tx_reset_bus_sync1;
            rx_reset_bus_sync2 <= rx_reset_bus_sync1;
        end if;
    end process;

    tx_bus_reset <= tx_reset_bus_sync2;
    rx_bus_reset <= rx_reset_bus_sync2;

    sr_txaref : setting_reg
        generic map(my_addr => FR_TX_A_REFCLK)
        port map(clock => master_clk, reset => tx_dsp_reset_t, strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 8) => OPEN,
            d_out(7 downto 0) => txa_refclk, changed => OPEN);

    sr_rxaref : setting_reg
        generic map(my_addr => FR_RX_A_REFCLK)
        port map(clock => master_clk, reset => rx_dsp_reset_t, strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 8) => OPEN,
            d_out(7 downto 0) => rxa_refclk, changed => OPEN);

    sr_txbref : setting_reg
        generic map(my_addr => FR_TX_B_REFCLK)
        port map(clock => master_clk, reset => tx_dsp_reset_t, strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 8) => OPEN,
            d_out(7 downto 0) => txb_refclk, changed => OPEN);

    sr_rxbref : setting_reg
        generic map(my_addr => FR_RX_B_REFCLK)
        port map(clock => master_clk, reset => rx_dsp_reset_t, strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 8) => OPEN,
            d_out(7 downto 0) => rxb_refclk, changed => OPEN);

    sr_debugen : setting_reg
        generic map(my_addr => FR_DEBUG_EN)
        port map(clock => master_clk, reset => dsp_reset_t, strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 4) => OPEN,
            d_out(3 downto 0) => debug_en, changed => OPEN);

    clk_div_0 : clk_divider
        port map(reset => tx_dsp_reset_t, in_clk => master_clk, out_clk => txaclk,
            ratio(7) => '0', ratio(6 downto 0) => txa_refclk(6 downto 0));

    clk_div_1 : clk_divider
        port map(reset => rx_dsp_reset_t, in_clk => master_clk, out_clk => rxaclk,
            ratio(7) => '0', ratio(6 downto 0) => rxa_refclk(6 downto 0));

    clk_div_2 : clk_divider
        port map(reset => tx_dsp_reset_t, in_clk => master_clk, out_clk => txbclk,
            ratio(7) => '0', ratio(6 downto 0) => txb_refclk(6 downto 0));

    clk_div_3 : clk_divider
        port map(reset => rx_dsp_reset_t, in_clk => master_clk, out_clk => rxbclk,
            ratio(7) => '0', ratio(6 downto 0) => rxb_refclk(6 downto 0));

    -- The upper 16 bits are a mask for the lower 16 bits.
    process(master_clk)
    begin
        if(master_clk'EVENT and master_clk = '1') then
            if(serial_strobe = '1') then
                case conv_integer(serial_addr) is
                    when FR_IO_0 =>
                        io_0_reg <= ((io_0_reg and (not serial_data(31 downto 16))) or
                                    (serial_data(15 downto 0) and serial_data(31 downto 16)));
                    when FR_IO_1 =>
                        io_1_reg <= ((io_1_reg and (not serial_data(31 downto 16))) or
                                    (serial_data(15 downto 0) and serial_data(31 downto 16)));
                    when FR_IO_2 =>
                        io_2_reg <= ((io_2_reg and (not serial_data(31 downto 16))) or
                                    (serial_data(15 downto 0) and serial_data(31 downto 16)));
                    when FR_IO_3 =>
                        io_3_reg <= ((io_3_reg and (not serial_data(31 downto 16))) or
                                    (serial_data(15 downto 0) and serial_data(31 downto 16)));
                    when others =>
                end case;
            end if;
        end if;
    end process;

    -- ATR stuff
    sr_atr_mask_0 : setting_reg
        generic map(my_addr => FR_ATR_MASK_0)
        port map(clock => master_clk, reset => '0', strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 16) => OPEN,
            d_out(15 downto 0) => atr_mask_0, changed => OPEN);

    sr_atr_txval_0 : setting_reg
        generic map(my_addr => FR_ATR_TXVAL_0)
        port map(clock => master_clk, reset => '0', strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 16) => OPEN,
            d_out(15 downto 0) => atr_txval_0, changed => OPEN);

    sr_atr_rxval_0 : setting_reg
        generic map(my_addr => FR_ATR_RXVAL_0)
        port map(clock => master_clk, reset => '0', strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 16) => OPEN,
            d_out(15 downto 0) => atr_rxval_0, changed => OPEN);

    sr_atr_mask_1 : setting_reg
        generic map(my_addr => FR_ATR_MASK_1)
        port map(clock => master_clk, reset => '0', strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 16) => OPEN,
            d_out(15 downto 0) => atr_mask_1, changed => OPEN);

    sr_atr_txval_1 : setting_reg
        generic map(my_addr => FR_ATR_TXVAL_1)
        port map(clock => master_clk, reset => '0', strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 16) => OPEN,
            d_out(15 downto 0) => atr_txval_1, changed => OPEN);

    sr_atr_rxval_1 : setting_reg
        generic map(my_addr => FR_ATR_RXVAL_1)
        port map(clock => master_clk, reset => '0', strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 16) => OPEN,
            d_out(15 downto 0) => atr_rxval_1, changed => OPEN);

    sr_atr_mask_2 : setting_reg
        generic map(my_addr => FR_ATR_MASK_2)
        port map(clock => master_clk, reset => '0', strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 16) => OPEN,
            d_out(15 downto 0) => atr_mask_2, changed => OPEN);

    sr_atr_txval_2 : setting_reg
        generic map(my_addr => FR_ATR_TXVAL_2)
        port map(clock => master_clk, reset => '0', strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 16) => OPEN,
            d_out(15 downto 0) => atr_txval_2, changed => OPEN);

    sr_atr_rxval_2 : setting_reg
        generic map(my_addr => FR_ATR_RXVAL_2)
        port map(clock => master_clk, reset => '0', strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 16) => OPEN,
            d_out(15 downto 0) => atr_rxval_2, changed => OPEN);

    sr_atr_mask_3 : setting_reg
        generic map(my_addr => FR_ATR_MASK_3)
        port map(clock => master_clk, reset => '0', strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 16) => OPEN,
            d_out(15 downto 0) => atr_mask_3, changed => OPEN);

    sr_atr_txval_3 : setting_reg
        generic map(my_addr => FR_ATR_TXVAL_3)
        port map(clock => master_clk, reset => '0', strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 16) => OPEN,
            d_out(15 downto 0) => atr_txval_3, changed => OPEN);

    sr_atr_rxval_3 : setting_reg
        generic map(my_addr => FR_ATR_RXVAL_3)
        port map(clock => master_clk, reset => '0', strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 16) => OPEN,
            d_out(15 downto 0) => atr_rxval_3, changed => OPEN);

--  sr_atr_ctl : setting_reg
--      generic map(my_addr => FR_ATR_CTL)
--      port map(clock => master_clk, reset => '0', strobe => serial_strobe,
--          addr => serial_addr, d_in => serial_data, d_out(31 downto 1) => OPEN,
--          d_out(0) => atr_ctl);

    sr_atr_tx_delay : setting_reg
        generic map(my_addr => FR_ATR_TX_DELAY)
        port map(clock => master_clk, reset => '0', strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 12) => OPEN,
            d_out(11 downto 0) => atr_tx_delay, changed => OPEN);

    sr_atr_rx_delay : setting_reg
        generic map(my_addr => FR_ATR_RX_DELAY)
        port map(clock => master_clk, reset => '0', strobe => serial_strobe,
            addr => serial_addr, d_in => serial_data, d_out(31 downto 12) => OPEN,
            d_out(11 downto 0) => atr_rx_delay, changed => OPEN);

    atr_ctl <= '1';

    atr_delay_0 : atr_delay
        port map(clk_i => master_clk, rst_i => tx_dsp_reset_t, ena_i => atr_ctl,
            tx_empty_i => tx_empty, tx_delay_i => atr_tx_delay, rx_delay_i => atr_rx_delay,
            atr_tx_o => transmit_now);

    atr_selected_0 <= atr_txval_0 when transmit_now = '1' else atr_rxval_0;
    atr_selected_1 <= atr_txval_1 when transmit_now = '1' else atr_rxval_1;
    atr_selected_2 <= atr_txval_2 when transmit_now = '1' else atr_rxval_2;
    atr_selected_3 <= atr_txval_3 when transmit_now = '1' else atr_rxval_3;

    io_0 <= (repeat(16, atr_ctl) and atr_mask_0 and atr_selected_0) or ((not (repeat(16, atr_ctl)
        and atr_mask_0)) and io_0_reg);
    io_1 <= (repeat(16, atr_ctl) and atr_mask_1 and atr_selected_1) or ((not (repeat(16, atr_ctl)
        and atr_mask_1)) and io_1_reg);
    io_2 <= (repeat(16, atr_ctl) and atr_mask_2 and atr_selected_2) or ((not (repeat(16, atr_ctl)
        and atr_mask_2)) and io_2_reg);
    io_3 <= (repeat(16, atr_ctl) and atr_mask_3 and atr_selected_3) or ((not (repeat(16, atr_ctl)
        and atr_mask_3)) and io_3_reg);

    reg_0 <= debug_0 when debug_en(0) = '1' else (io_0(15 downto 1) & txaclk) when txa_refclk(7)
        = '1' else io_0;
    reg_1 <= debug_1 when debug_en(1) = '1' else (io_1(15 downto 1) & rxaclk) when rxa_refclk(7)
        = '1' else io_1;
    reg_2 <= debug_2 when debug_en(2) = '1' else (io_2(15 downto 1) & txbclk) when txb_refclk(7)
        = '1' else io_2;
    reg_3 <= debug_3 when debug_en(3) = '1' else (io_3(15 downto 1) & rxbclk) when rxb_refclk(7)
        = '1' else io_3;
end behavioral;
```

Listing A.29: A 16x16 bit multiplier, used in conjunction with the acc module to create a MAC for the halfband filter. (mult.vhd)

```
--------------------------------------------------------------------------------
-- 16x16 Multiplier
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

library WORK;
use WORK.FUNCS.ALL;

entity mult is
    port(
        clock          : in    std_logic;
        x              : in    std_logic_vector(15 downto 0);
        y              : in    std_logic_vector(15 downto 0);
        product     : out   std_logic_vector(30 downto 0);
        enable_in   : in    std_logic;
        enable_out  : out   std_logic
    );
end mult;
```

```

architecture behavioral of mult is
    signal x_t : signed(15 downto 0);
    signal y_t : signed(15 downto 0);

    -- I wonder if the original Verilog was in error using 30:0.
    signal product_t : signed(31 downto 0);
begin
    x_t <= conv_signed(conv_integer(x), x_t'LENGTH);
    y_t <= conv_signed(conv_integer(y), y_t'LENGTH);
    product <= conv_std_logic_vector(product_t(30 downto 0), product'LENGTH);

    -- Multiple on each clock cycle, if enabled.
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(enable_in = '1') then
                product_t <= x_t * y_t;
            else
                product_t <= (others => '0');
            end if;
        end if;
    end process;

    -- I guess this lets us know if a result is valid?
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            enable_out <= enable_in;
        end if;
    end process;
end behavioral;
```

Listing A.30: Phase accumulator module for the NCO. (phase_acc.vhd)

```
-------------------------------------------------------------------------------
-- Basic Phase Accumulator (for DDS)
-- Last Modified: 20 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.
--
-- You should have received a copy of the GNU General Public License
-- along with this program; if not, write to the Free Software
-- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity phase_acc is
    generic(
        FREQADDR        : integer := 0;
        PHASEADDR   : integer := 0;
        resolution  : integer range 0 to 32 := 32
    );

    port(
        clk             : in    std_logic;
        reset               : in    std_logic;
        enable          : in    std_logic;
        strobe          : in    std_logic;
        serial_addr     : in    std_logic_vector(6 downto 0);
        serial_data     : in    std_logic_vector(31 downto 0);
        serial_strobe   : in    std_logic;
        phase               : out   std_logic_vector(resolution-1 downto 0)
    );
end phase_acc;

architecture behavioral of phase_acc is
    component setting_reg is
        generic(
            my_addr : integer := 0
        );

        port(
            clock       : in    std_logic;
            reset       : in    std_logic;
            strobe  : in    std_logic;
            addr        : in    std_logic_vector;
            d_in        : in    std_logic_vector(31 downto 0);
            d_out       : out   std_logic_vector(31 downto 0);
            changed : out   std_logic
        );
    end component;

    signal freq : std_logic_vector(resolution-1 downto 0);
    signal phase_t : std_logic_vector(resolution-1 downto 0);
begin
    phase <= phase_t;

    -- Note, d_out => freq only works when resolution = 32.  Else,
    -- you'll get a width mismatch.  Unfortunately, we can't use
    -- a generic value like resolution to wire part of d_out to OPEN
    -- because Xilinx throws an error, "Discrete range on slice is not
    -- locally static."
    sr_rxfreq0 : setting_reg
        generic map(my_addr => FREQADDR)
        port map(clock => clk, reset => '0', strobe => serial_strobe, addr => serial_addr,
            d_in => serial_data, d_out => freq, changed => OPEN);

    process(clk)
    begin
        if(clk'EVENT and clk = '1') then
            if(reset = '1') then
                -- Reset the accumulator.
                phase_t <= (others => '0');
            elsif((serial_strobe = '1') and (serial_addr = PHASEADDR)) then
                -- Initial phase comes from serial input.
                phase_t <= serial_data;
            elsif((enable = '1') and (strobe = '1')) then
                -- Accumulate on each strobe, where 'freq' is the center frequency.
                phase_t <= phase_t + freq;
            end if;
        end if;
    end process;
end behavioral;
```

Listing A.31: A 16-bit RAM used in the halfband filter. (ram16.vhd)

```
------------------------------------------------------------------------
-- RAM16
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
```

```vhdl
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ram16 is
    port(
        clock           : in    std_logic;
        write_in        : in    std_logic;
        wr_addr     : in    std_logic_vector(3 downto 0);
        wr_data     : in    std_logic_vector(15 downto 0);
        rd_addr     : in    std_logic_vector(3 downto 0);
        rd_data     : out   std_logic_vector(15 downto 0)
    );
end ram16;

architecture behavioral of ram16 is
    -- Create a 16x16 bit RAM.
    type ram is array(integer range <>) of std_logic_vector(15 downto 0);
    signal ram_array : ram(0 to 15);
begin
    -- Read from RAM on every clock.
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            rd_data <= ram_array(conv_integer(rd_addr));
        end if;
    end process;

    -- Write to RAM only when write_in is asserted.
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(write_in = '1') then
                ram_array(conv_integer(wr_addr)) <= wr_data;
            end if;
        end if;
    end process;
end behavioral;
```

Listing A.32: A 16-bit RAM with two read ports. This RAM sums the two read values together. (ram16_2sum.vhd)

```vhdl
--------------------------------------------------------------------------------
-- RAM16_2SUM
-- Last Modified: 14 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
```

```
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------

-- TODO:  Convert to the newer numeric_std library.
-- "Tri-port" RAM (2 reads, 1 write).  The output is the sum of the two read
-- values.  Sort of.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

entity ram16_2sum is
    port(
        clock       : in    std_logic;
        write_in    : in    std_logic;
        wr_addr : in    std_logic_vector(3 downto 0);
        wr_data : in    std_logic_vector(15 downto 0);
        rd_addr1    : in    std_logic_vector(3 downto 0);
        rd_addr2    : in    std_logic_vector(3 downto 0);
        sum     : out   std_logic_vector(15 downto 0)
    );
end ram16_2sum;

architecture behavioral of ram16_2sum is
    -- Create a 16x16 bit RAM.
    type ram is array (integer range <>) of signed(15 downto 0);
    signal ram_array : ram(0 to 15);

    signal a : signed(15 downto 0);
    signal b : signed(15 downto 0);
    signal sum_int : signed(16 downto 0);
begin
    -- Only write to RAM if write_in is high.
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(write_in = '1') then
                -- Isn't this horrible?  I think it's horrible.  VHDL needs to fix this.
                ram_array(conv_integer(conv_unsigned(conv_integer(wr_addr), wr_addr'LENGTH))) <=
                    conv_signed(conv_integer(wr_data), wr_data'LENGTH);
            end if;
        end if;
    end process;

    -- Read two values from RAM on every clock cycle.
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            a <= ram_array(conv_integer(conv_unsigned(conv_integer(rd_addr1), rd_addr1'LENGTH)));
            b <= ram_array(conv_integer(conv_unsigned(conv_integer(rd_addr2), rd_addr2'LENGTH)));
        end if;
    end process;

    -- Signed addition of the two read values.
    sum_int <= (a(15) & a) + (b(15) & b);

    -- Output the sum on every clock cycle.  If the sum is negative and odd, add 1.
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if((sum_int(16) = '1') and (sum_int(0) = '1')) then
                sum <= conv_std_logic_vector(conv_signed(sum_int(16 downto 1) + 1, sum'LENGTH),
                    sum'LENGTH);
            else
                sum <= conv_std_logic_vector(conv_signed(sum_int(16 downto 1), sum'LENGTH),
                    sum'LENGTH);
            end if;
        end if;
    end process;
end behavioral;
```

Listing A.33: Received signal strength indicator (RSSI) and over/underflow detector module. (rssi.vhd)

```
--------------------------------------------------------------------------------
-- Receive Signal Strength Indicator (RSSI)
-- Last Modified: 26 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```

entity rssi is
    port(
        clock       : in    std_logic;
        reset       : in    std_logic;
        enable      : in    std_logic;
        adc         : in    std_logic_vector(11 downto 0);
        rssi        : out   std_logic_vector(15 downto 0);
        over_count  : out   std_logic_vector(15 downto 0)
    );
end rssi;

architecture behavioral of rssi is
    signal over_hi : std_logic;
    signal over_lo : std_logic;
    signal over : std_logic;
    signal over_count_int : std_logic_vector(25 downto 0);
    signal over_tmp : std_logic_vector(25 downto 0);
    signal abs_adc : std_logic_vector(11 downto 0);
    signal rssi_int : std_logic_vector(25 downto 0);
begin
    -- Maximum value for 12 signed bits.
    over_hi <= '1' when (adc = 16#7FF#) else '0';

    -- Minimum value for 12 signed bits.
    over_lo <= '1' when (adc = 16#800#) else '0';

    -- Simple OR gate to determine overflows and underflows.
    over <= over_hi or over_lo;

    -- The original Verilog did this inside the process, but XST translates that
    -- differently.  This is based on the rssi logic.
    -- d26'65535 = b26'00000000001111111111111111.
    over_tmp <= "00000000001111111111111111" when over = '1' else
                "00000000000000000000000000";

    -- Track overflows and underflows from the ADCs.
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(reset = '1' or enable = '0') then
                -- Clear on reset.
                over_count_int <= (others => '0');
            else
                over_count_int <= over_count_int + over_tmp - over_count_int(25 downto 10);
            end if;
        end if;
    end process;

    over_count <= over_count_int(25 downto 10);

    -- Absolute value of adc (sort of).
    abs_adc <= (not adc) when adc(11) = '1' else adc;

    -- Track signal strength from ADCs.
    process(clock)
    begin
```

```
        if(clock'EVENT and clock = '1') then
            if(reset = '1' or enable = '0') then
                -- Clear on reset.
                rssi_int <= (others => '0');
            else
                rssi_int <= rssi_int + abs_adc - rssi_int(25 downto 10);
            end if;
        end if;
    end process;

    rssi <= rssi_int(25 downto 10);
end behavioral;
```

Listing A.34: Controls the FIFO buffer and USB interface on the receive side. (rx_buffer.vhd)

```
-------------------------------------------------------------------------------
-- Receive Buffer (USB <-> DSP)
-- Last Modified: 20 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
-------------------------------------------------------------------------------

-- Interface to Cypress FX2 bus.
-- A packet is 512 Bytes.  Each FIFO line is 2 bytes.
-- FIFO has 4096 lines.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library WORK;
use WORK.FPGA_REGS_STANDARD.ALL;
use WORK.REDUCE_PACK.ALL;

entity rx_buffer is
```

```
    port(
        -- Read/USB side
        usbclk          : in    std_logic;
        bus_reset       : in    std_logic;
        usbdata         : out   std_logic_vector(15 downto 0);
        RD              : in    std_logic;
        have_pkt_rdy    : out   std_logic;
        rx_overrun      : out   std_logic;
        clear_status    : in    std_logic;

        -- Write/DSP side
        rxclk           : in    std_logic;
        reset           : in    std_logic;  -- DSP-side reset, not for registers
        rxstrobe        : in    std_logic;
        channels        : in    std_logic_vector(3 downto 0);
        ch_0            : in    std_logic_vector(15 downto 0);
        ch_1            : in    std_logic_vector(15 downto 0);
        ch_2            : in    std_logic_vector(15 downto 0);
        ch_3            : in    std_logic_vector(15 downto 0);
        ch_4            : in    std_logic_vector(15 downto 0);
        ch_5            : in    std_logic_vector(15 downto 0);
        ch_6            : in    std_logic_vector(15 downto 0);
        ch_7            : in    std_logic_vector(15 downto 0);

        -- Settings, also using rxclk
        serial_addr     : in    std_logic_vector(6 downto 0);
        serial_data     : in    std_logic_vector(31 downto 0);
        serial_strobe   : in    std_logic;
        reset_regs      : in    std_logic;  -- Only resets regs
        debugbus        : out   std_logic_vector(31 downto 0)
    );
end rx_buffer;

architecture behavioral of rx_buffer is
    function round_8(in_val : std_logic_vector(15 downto 0))
        return std_logic_vector
    is
    begin
--      if(in_val(15) = '1' and or_reduce(in_val(7 downto 0)) = '1') then
--          return in_val(15 downto 8) + 1;
--      else
--          return in_val(15 downto 8);
--      end if;
        return in_val(15 downto 8) + (in_val(15) and or_reduce(in_val(7 downto 0)));
    end function;

    component setting_reg is
        generic(
            my_addr : integer := 0
        );

        port(
            clock   : in    std_logic;
            reset   : in    std_logic;
            strobe  : in    std_logic;
            addr    : in    std_logic_vector;
            d_in    : in    std_logic_vector(31 downto 0);
            d_out   : out   std_logic_vector(31 downto 0);
            changed : out   std_logic
        );
    end component;

    component fifo_4k_18 is
        port(
            data    : in    std_logic_vector(17 downto 0);
            wrreq   : in    std_logic;
            wrclk   : in    std_logic;
            wrfull  : out   std_logic;
            wrempty : out   std_logic;
            wrusedw : out   std_logic_vector(11 downto 0);
            q       : out   std_logic_vector(17 downto 0);
            rdreq   : in    std_logic;
            rdclk   : in    std_logic;
            rdfull  : out   std_logic;
            rdempty : out   std_logic;
            rdusedw : out   std_logic_vector(11 downto 0);
            aclr    : in    std_logic
        );
    end component;

    signal fifodata : std_logic_vector(15 downto 0);
    signal fifodata_8 : std_logic_vector(15 downto 0);
    signal fifodata_16 : std_logic_vector(15 downto 0);

    signal rxfifolevel : std_logic_vector(11 downto 0);
    signal rx_full : std_logic;

    signal bypass_hb : std_logic;
    signal want_q : std_logic;
    signal bitwidth : std_logic_vector(4 downto 0);
    signal bitshift : std_logic_vector(3 downto 0);

    signal read_count : std_logic_vector(8 downto 0);

    signal phase : std_logic_vector(3 downto 0);
    signal ch0_in : std_logic;
    signal ch0_out : std_logic;
    signal iq_out : std_logic;

    -- Temps
    signal have_pkt_rdy_t : std_logic;
    signal rx_overrun_t : std_logic;

    signal wr_req_t : std_logic;
    signal rd_req_t : std_logic;
    signal usbclk_t : std_logic;
    signal channels_h : std_logic_vector(3 downto 0);

    signal ch_0_reg : std_logic_vector(15 downto 0);
    signal ch_1_reg : std_logic_vector(15 downto 0);
    signal ch_2_reg : std_logic_vector(15 downto 0);
    signal ch_3_reg : std_logic_vector(15 downto 0);
    signal ch_4_reg : std_logic_vector(15 downto 0);
    signal ch_5_reg : std_logic_vector(15 downto 0);
    signal ch_6_reg : std_logic_vector(15 downto 0);
    signal ch_7_reg : std_logic_vector(15 downto 0);

    signal top : std_logic_vector(15 downto 0);
    signal bottom : std_logic_vector(15 downto 0);

    signal clear_status_dsp : std_logic;
    signal rx_overrun_dsp : std_logic;

    -- Temporary values for getting 8-bit fifodata.
--  signal top8 : std_logic_vector(7 downto 0);
--  signal bottom8 : std_logic_vector(7 downto 0);
begin
    have_pkt_rdy <= have_pkt_rdy_t;
    rx_overrun <= rx_overrun_t;

    sr_rxformat : setting_reg
        generic map(my_addr => FR_RX_FORMAT)
        port map(clock => rxclk, reset => reset_regs, strobe => serial_strobe, addr =>
            serial_addr,
            d_in => serial_data, d_out(31 downto 11) => OPEN, d_out(10) => bypass_hb, d_out(9) =>
                want_q,
            d_out(8 downto 4) => bitwidth, d_out(3 downto 0) => bitshift, changed => OPEN);

    -- USB read side of FIFO
    process(usbclk)
    begin
        if(usbclk'EVENT and usbclk = '0') then
            if(rxfifolevel >= "000100000000") then  -- 256
                have_pkt_rdy_t <= '1';
            else
                have_pkt_rdy_t <= '0';
            end if;
        end if;
    end process;

    -- 257 bug fix
    process(usbclk)
    begin
        if(usbclk'EVENT and usbclk = '0') then
            if(bus_reset = '1') then
                read_count <= (others => '0');
            elsif(RD = '1') then
                read_count <= read_count + 1;
            else
                read_count <= (others => '0');
            end if;
        end if;
    end process;

    -- FIFO
    ch0_in <= '1' when (phase = 1) else '0';
    wr_req_t <= '1' when ((rx_full = '0') and (phase /= 0)) else '0';
    rd_req_t <= '1' when ((RD = '1') and (read_count(8) = '0')) else '0';
    usbclk_t <= (not usbclk);
    rxfifo : fifo_4k_18
        port map(data(17) => ch0_in, data(16) => phase(0), data(15 downto 0) => fifodata,
            wrreq => wr_req_t, wrclk => rxclk, wrfull => rx_full, wrempty => OPEN, wrusedw =>
                OPEN,
            q(17) => ch0_out, q(16) => iq_out, q(15 downto 0) => usbdata, rdreq => rd_req_t,
            rdclk => usbclk_t, rdfull => OPEN, rdempty => OPEN, rdusedw => rxfifolevel, aclr =>
                reset);

    -- DSP write side of FIFO
    process(rxclk)
    begin
        if(rxclk'EVENT and rxclk = '1') then
            if(rxstrobe = '1') then
                ch_0_reg <= ch_0;
                ch_1_reg <= ch_1;
                ch_2_reg <= ch_2;
                ch_3_reg <= ch_3;
                ch_4_reg <= ch_4;
                ch_5_reg <= ch_5;
                ch_6_reg <= ch_6;
                ch_7_reg <= ch_7;
            end if;
        end if;
    end process;

    channels_h <= '0' & channels(3 downto 1);   -- channels >> 1;
    process(rxclk)
    begin
        if(rxclk'EVENT and rxclk = '1') then
            if(reset = '1') then
                phase <= (others => '0');
            elsif(phase = 0) then
                if(rxstrobe = '1') then
                    phase <= "0001";
                end if;
            elsif(rx_full = '0') then
                if(((bitwidth = 8) and (phase = channels_h)) or (phase = channels)) then
                    phase <= "0000";
                else
                    phase <= phase + 1;
                end if;
            end if;
        end if;
    end process;

    fifodata <= fifodata_8 when (bitwidth = 8) else fifodata_16;

--  top8 <= top(15 downto 8) + 1 when ((top(15) = '1') and (top(7 downto 0) /= "00000000")) else
--          top(15 downto 8);

--  bottom8 <= bottom(15 downto 8) + 1 when ((bottom(15) = '1') and (bottom(7 downto 0) /=
    "00000000")) else
--          bottom(15 downto 8);

    fifodata_8 <= round_8(top) & round_8(bottom);
    --fifodata_8 <= top8 & bottom8;

    process(phase, ch_0_reg, ch_1_reg, ch_2_reg, ch_3_reg, ch_4_reg, ch_5_reg, ch_6_reg, ch_7_reg)
    begin
```

```
        case conv_integer(phase) is
            when 1 =>
                bottom <= ch_0_reg;
                top <= ch_1_reg;
            when 2 =>
                bottom <= ch_2_reg;
                top <= ch_3_reg;
            when 3 =>
                bottom <= ch_4_reg;
                top <= ch_5_reg;
            when 4 =>
                top <= ch_6_reg;
                bottom <= ch_7_reg;
            when others =>
                top <= (others => '1');
                bottom <= (others => '1');
        end case;
    end process;

    process(phase, ch_0_reg, ch_1_reg, ch_2_reg, ch_3_reg, ch_4_reg, ch_5_reg, ch_6_reg, ch_7_reg)
    begin
        case conv_integer(phase) is
            when 1 =>
                fifodata_16 <= ch_0_reg;
            when 2 =>
                fifodata_16 <= ch_1_reg;
            when 3 =>
                fifodata_16 <= ch_2_reg;
            when 4 =>
                fifodata_16 <= ch_3_reg;
            when 5 =>
                fifodata_16 <= ch_4_reg;
            when 6 =>
                fifodata_16 <= ch_5_reg;
            when 7 =>
                fifodata_16 <= ch_6_reg;
            when 8 =>
                fifodata_16 <= ch_7_reg;
            when others =>
                fifodata_16 <= (others => '1');
        end case;
    end process;

    -- Detect overrun
    process(rxclk)
    begin
        if(rxclk'EVENT and rxclk = '1') then
            clear_status_dsp <= clear_status;
        end if;
    end process;

    process(usbclk)
    begin
        if(usbclk'EVENT and usbclk = '0') then
            rx_overrun_t <= rx_overrun_dsp;
        end if;
    end process;

    process(rxclk)
    begin
        if(rxclk'EVENT and rxclk = '1') then
            if(reset = '1') then
                rx_overrun_dsp <= '0';
            elsif((rxstrobe = '1') and (phase /= 0)) then
                rx_overrun_dsp <= '1';
            elsif(clear_status_dsp = '1') then
                rx_overrun_dsp <= '0';
            end if;
        end if;
    end process;

    -- Debug bus
    -- 15:0     rxclk   domain => TXA(15:0)
    -- 31:16    usbclk  domain => RXA(15:0)
    debugbus(0)             <= reset;
    debugbus(1)             <= reset_regs;
    debugbus(2)             <= rxstrobe;
    debugbus(6 downto 3)    <= channels;
    debugbus(7)             <= rx_full;
    debugbus(11 downto 8)   <= phase;
    debugbus(12)            <= ch0_in;
    debugbus(13)            <= clear_status_dsp;
    debugbus(14)            <= rx_overrun_dsp;
    debugbus(15)            <= rxclk;

    debugbus(16)            <= bus_reset;
    debugbus(17)            <= RD;
    debugbus(18)            <= have_pkt_rdy_t;
    debugbus(19)            <= rx_overrun_t;
    debugbus(20)            <= read_count(0);
    debugbus(21)            <= read_count(8);
    debugbus(22)            <= ch0_out;
    debugbus(23)            <= iq_out;
    debugbus(24)            <= clear_status;
    debugbus(30 downto 25)  <= "000000";
    debugbus(31)            <= usbclk;
end behavioral;
```

Listing A.35: Controls the IF to BB conversion and the decimation filters for received data. (rx_chain.vhd)

```
--------------------------------------------------------------------------------
-- Receive Chain
-- Last Modified: 20 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--   Copyright (C) 2003 Matt Ettus
--
```

```
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
----------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library WORK;
use WORK.COMMON_CONFIG.ALL;

entity rx_chain is
    generic(
        FREQADDR : integer := 0;
        PHASEADDR : integer := 0
    );

    port(
        clock               : in    std_logic;
        reset               : in    std_logic;
        enable              : in    std_logic;
        decim_rate          : in    std_logic_vector(7 downto 0);
        sample_strobe       : in    std_logic;
        decimator_strobe    : in    std_logic;
        hb_strobe           : out   std_logic;
        serial_addr         : in    std_logic_vector(6 downto 0);
        serial_data         : in    std_logic_vector(31 downto 0);
        serial_strobe       : in    std_logic;
        i_in                : in    std_logic_vector(15 downto 0);
        q_in                : in    std_logic_vector(15 downto 0);
        i_out               : out   std_logic_vector(15 downto 0);
        q_out               : out   std_logic_vector(15 downto 0);
        debugdata           : out   std_logic_vector(15 downto 0);
        debugctrl           : out   std_logic_vector(15 downto 0)
    );
end rx_chain;

architecture behavioral of rx_chain is
    component phase_acc is
        generic(
            FREQADDR     : integer := 0;
            PHASEADDR    : integer := 0;
            resolution   : integer range 0 to 32 := 32
        );

        port(
            clk             : in    std_logic;
            reset           : in    std_logic;
            enable          : in    std_logic;
            strobe          : in    std_logic;
            serial_addr     : in    std_logic_vector(6 downto 0);
            serial_data     : in    std_logic_vector(31 downto 0);
            serial_strobe   : in    std_logic;
            phase           : out   std_logic_vector(resolution-1 downto 0)
        );
    end component;

    component cordic is
        generic(
            bitwidth : integer := 16;
            zwidth : integer := 16
        );

        port(
            clock   : in    std_logic;
            reset   : in    std_logic;
            enable  : in    std_logic;
            xi      : in    std_logic_vector(bitwidth-1 downto 0);
            yi      : in    std_logic_vector(bitwidth-1 downto 0);
            xo      : out   std_logic_vector(bitwidth-1 downto 0);
            yo      : out   std_logic_vector(bitwidth-1 downto 0);
            zi      : in    std_logic_vector(zwidth-1 downto 0);
            zo      : out   std_logic_vector(zwidth-1 downto 0)
        );
    end component;

    component cic_decim is
        generic(
            bw : integer := 16;
            N : integer := 4;
            log2_of_max_rate : integer := 7;
            maxbitgain : integer := 28  -- N*log2_of_max_rate
        );

        port(
            clock       : in    std_logic;
            reset       : in    std_logic;
            enable      : in    std_logic;
            rate        : in    std_logic_vector(7 downto 0);
            strobe_in   : in    std_logic;
            strobe_out  : in    std_logic;
            signal_in   : in    std_logic_vector(bw-1 downto 0);
            signal_out  : out   std_logic_vector(bw-1 downto 0)
        );
    end component;

    component halfband_decim is
        port(
            clock       : in    std_logic;
            reset       : in    std_logic;
            enable      : in    std_logic;
            strobe_in   : in    std_logic;
            strobe_out   : out   std_logic;
            data_in      : in    std_logic_vector(15 downto 0);
            data_out     : out   std_logic_vector(15 downto 0);
            debugctrl    : out   std_logic_vector(15 downto 0)
        );
    end component;

    signal phase : std_logic_vector(31 downto 0);

    signal bb_i : std_logic_vector(15 downto 0);
    signal bb_q : std_logic_vector(15 downto 0);

    signal hb_in_i : std_logic_vector(15 downto 0);
    signal hb_in_q : std_logic_vector(15 downto 0);

    -- Enable write on inputs, read on outputs.
    signal sample_strobe_t : std_logic;
    signal decimator_strobe_t : std_logic;
begin
    debugdata <= hb_in_i;

    rx_nco_t : if(RX_NCO_EN) generate
        sample_strobe_t <= sample_strobe;

        rx_phase_acc : phase_acc
            generic map(FREQADDR => FREQADDR, PHASEADDR => PHASEADDR, resolution => 32)
            port map(clk => clock, reset => reset, enable => enable, serial_addr => serial_addr,
                serial_data => serial_data, serial_strobe => serial_strobe, strobe =>
                    sample_strobe_t,
                phase => phase);

        rx_cordic : cordic
            port map(clock => clock, reset => reset, enable => enable, xi => i_in, yi => q_in,
                zi => phase(31 downto 16), xo => bb_i, yo => bb_q, zo => OPEN);
    end generate rx_nco_t;

    rx_nco_f : if(not RX_NCO_EN) generate
        bb_i <= i_in;
        bb_q <= q_in;
        sample_strobe_t <= '1';
    end generate rx_nco_f;

    rx_cic_t : if(RX_CIC_EN) generate
        decimator_strobe_t <= decimator_strobe;

        cic_decim_i_0 : cic_decim
            port map(clock => clock, reset => reset, enable => enable, rate => decim_rate,
                strobe_in => sample_strobe, strobe_out => decimator_strobe_t, signal_in => bb_i,
                signal_out => hb_in_i);

        cic_decim_q_0 : cic_decim
            port map(clock => clock, reset => reset, enable => enable, rate => decim_rate,
                strobe_in => sample_strobe, strobe_out => decimator_strobe_t, signal_in => bb_q,
                signal_out => hb_in_q);
    end generate rx_cic_t;

    rx_cic_f : if(not RX_CIC_EN) generate
```

```
        hb_in_i <= bb_i;
        hb_in_q <= bb_q;
        decimator_strobe_t <= sample_strobe;
    end generate rx_cic_f;

    rx_hb_t : if(RX_CAP_HB) generate
        hbd_i_0 : halfband_decim
            port map(clock => clock, reset => reset, enable => enable, strobe_in =>
                 decimator_strobe,
                strobe_out => hb_strobe, data_in => hb_in_i, data_out => i_out, debugctrl =>
                     debugctrl);

        hbd_q_0 : halfband_decim
            port map(clock => clock, reset => reset, enable => enable, strobe_in =>
                 decimator_strobe,
                strobe_out => OPEN, data_in => hb_in_q, data_out => q_out, debugctrl => OPEN);
    end generate rx_hb_t;

    rx_hb_f : if(not RX_CAP_HB) generate
        i_out <= hb_in_i;
        q_out <= hb_in_q;
        hb_strobe <= decimator_strobe;
    end generate rx_hb_f;
end behavioral;
```

Listing A.36: Removes the codec-imposed DC offset from received data. (rx_dcoffset.vhd)

```
-----------------------------------------------------------------------------------------
-- Rx DC Offset
-- Last Modified: 20 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
-----------------------------------------------------------------------------------------

-- TODO: The reduce_pack license is probably not compatible with the GPL.
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

library WORK;
use WORK.REDUCE_PACK.ALL;

entity rx_dcoffset is
    generic(
        MYADDR  : integer := 0
    );

    port(
        clock              : in    std_logic;
        enable          : in    std_logic;
        reset              : in    std_logic;
        adc_in          : in    std_logic_vector(15 downto 0);
        adc_out         : out   std_logic_vector(15 downto 0);
        serial_addr     : in    std_logic_vector(6 downto 0);
        serial_data     : in    std_logic_vector(31 downto 0);
        serial_strobe   : in    std_logic
    );
end rx_dcoffset;

architecture behavioral of rx_dcoffset is
    signal adc_in_t : signed(15 downto 0);
    signal adc_out_t : signed(15 downto 0);
    signal serial_addr_t : unsigned(6 downto 0);
    signal serial_data_t : unsigned(31 downto 0);
    signal integrator : signed(31 downto 0);
    signal scaled_integrator : signed(15 downto 0);
begin
    -- Assign temporary signals to outputs, with conversions.
    adc_in_t <= conv_signed(conv_integer(adc_in), 16);
    serial_addr_t <= conv_unsigned(conv_integer(serial_addr), 7);
    serial_data_t <= conv_unsigned(conv_integer(serial_data), 32);
    adc_out <= conv_std_logic_vector(adc_out_t, 16);

    -- The original Verilog code used the or reduction operator, but this is equivalent to
    -- checking if the vector equals zero.
    scaled_integrator <= integrator(31 downto 16) + 1 when ((integrator(31) = '1') and
         (integrator(15 downto 0) /= 0)) else
                                integrator(31 downto 16);
    adc_out_t <= adc_in_t - scaled_integrator;

    -- FIXME do we need signed?
    -- FIXME What do we do when clipping?
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(reset = '1') then
                -- Clear on reset.
                integrator <= (others => '0');
            elsif(serial_strobe = '1' and (MYADDR = serial_addr_t)) then
                -- Read new value from the serial bus.
                -- This is ugly, but it forces XST to see a single 32-bit register instead of
                -- 32 1-bit registers.
                integrator <= conv_signed(serial_data_t(15 downto 0), 16) & "0000000000000000";
            elsif(enable = '1') then
                -- Otherwise, just keep accumulating.
                integrator <= integrator + adc_out_t;
            end if;
        end if;
    end process;
end behavioral;
```

Listing A.37: A very basic serial bus controller to receive SPI messages from the host, usually to conigure registers. (serial_io.vhd)

```
--------------------------------------------------------------------------------
-- Serial I/O Bus
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity serial_io is
    port(
        master_clk          : in    std_logic;
        serial_clock        : in    std_logic;
        serial_data_in      : in    std_logic;
        enable              : in    std_logic;
        reset                   : in    std_logic;
        serial_data_out : out   std_logic;
        serial_addr         : out   std_logic_vector(6 downto 0);
        serial_data         : out   std_logic_vector(31 downto 0);
        serial_strobe       : out   std_logic;
        readback_0          : in    std_logic_vector(31 downto 0);
        readback_1          : in    std_logic_vector(31 downto 0);
        readback_2          : in    std_logic_vector(31 downto 0);
        readback_3          : in    std_logic_vector(31 downto 0);
        readback_4          : in    std_logic_vector(31 downto 0);
        readback_5          : in    std_logic_vector(31 downto 0);
        readback_6          : in    std_logic_vector(31 downto 0);
        readback_7          : in    std_logic_vector(31 downto 0)
    );
end serial_io;

architecture behavioral of serial_io is
    signal is_read : std_logic;
    signal write_done : std_logic;
    signal ser_ctr : std_logic_vector(7 downto 0);
    signal serial_data_tmp : std_logic_vector(31 downto 0);
    signal serial_addr_tmp : std_logic_vector(6 downto 0);
    signal enable_d1 : std_logic;
    signal enable_d2 : std_logic;
begin
    serial_data_out <= serial_data_tmp(31) when is_read = '1' else 'Z';
    serial_data <= serial_data_tmp;
    serial_addr <= serial_addr_tmp;

    process(serial_clock, reset, enable)
    begin
        if(reset = '1') then
            ser_ctr <= (others => '0');
        elsif(enable = '0') then
            ser_ctr <= (others => '0');
        elsif(serial_clock'EVENT and serial_clock = '1') then
            -- A serial transaction is up to 40 bits.  The first bit is 1 for read and
            -- 0 for write.  The next seven bits are the actual register address.
            -- The final 32 bits are the value to write in a write operation.
            if(ser_ctr = 39) then
                ser_ctr <= (others => '0');
            else
                ser_ctr <= ser_ctr + 1;
            end if;
        end if;
    end process;

    process(serial_clock, reset, enable)
    begin
        if(reset = '1') then
            is_read <= '0';
        elsif(enable = '0') then
            is_read <= '0';
        elsif(serial_clock'EVENT and serial_clock = '1') then
            -- Check the first bit we shifted in just before we shift in the
            -- final address bit.
            if((ser_ctr = 7) and (serial_addr_tmp(6) = '1')) then
                is_read <= '1';
            end if;
        end if;
    end process;

```

```
process(serial_clock, enable, reset)
begin
    if(reset = '1') then
        serial_addr_tmp <= (others => '0');
        serial_data_tmp <= (others => '0');
        write_done <= '0';
    elsif(enable = '0') then
        --serial_addr_tmp <= (others => '0');
        --serial_data_tmp <= (others => '0');
        write_done <= '0';
    elsif(serial_clock'EVENT and serial_clock = '1') then
        -- On the final clock, indicate that we performed a write
        -- operation if we did not perform a read operation.
        if(is_read = '0' and ser_ctr = 39) then
            write_done <= '1';
        else
            write_done <= '0';
        end if;

        -- 8'b00001000 = 8'd8
        if(is_read = '1' and ser_ctr = "00001000") then
            -- After 8 shifts, we know that it's a read op and we have
            -- the seven address bits.  So, read the proper register.
            case serial_addr_tmp is
                when "0000001" =>
                    serial_data_tmp <= readback_0;
                when "0000010" =>
                    serial_data_tmp <= readback_1;
                when "0000011" =>
                    serial_data_tmp <= readback_2;
                when "0000100" =>
                    serial_data_tmp <= readback_3;
                when "0000101" =>
                    serial_data_tmp <= readback_4;
                when "0000110" =>
                    serial_data_tmp <= readback_5;
                when "0000111" =>
                    serial_data_tmp <= readback_6;
                when "0001000" =>
                    serial_data_tmp <= readback_7;
                when others =>
                    serial_data_tmp <= (others => '0');
            end case;
        elsif(ser_ctr >= "00001000") then
            -- On the 9th+ shift, write to the serial data because this is
            -- a write op.
            serial_data_tmp <= serial_data_tmp(30 downto 0) & serial_data_in;
        elsif(ser_ctr < "00001000") then
            -- For the first 8 shifts, collect the read/write bit and then the
            -- register address.  We temporarily store the read/write bit into
            -- the address vector.
            serial_addr_tmp <= serial_addr_tmp(5 downto 0) & serial_data_in;
        end if;
    end if;
end process;

process(master_clk)
```

```
    begin
        if(master_clk'EVENT and master_clk = '1') then
            enable_d1 <= enable;
            enable_d2 <= enable_d1;
        end if;
    end process;

    -- Assert serial strobe for one clock after the enable is asserted and
    -- then deasserted.
    serial_strobe <= (enable_d2 and (not enable_d1));
end behavioral;
```

Listing A.38: A configuration register module. (setting_reg.vhd)

```
--------------------------------------------------------------------------------
-- Setting Register
-- Last Modified: 20 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity setting_reg is
    generic(
        my_addr : integer := 0
    );

    port(
        clock       : in    std_logic;
        reset       : in    std_logic;
        strobe  : in    std_logic;
        addr     : in    std_logic_vector(6 downto 0);
        d_in     : in    std_logic_vector(31 downto 0);
        d_out    : out   std_logic_vector(31 downto 0);
        changed : out   std_logic
    );
end setting_reg;

architecture behavioral of setting_reg is
begin
    -- Each register in the system has a unique address, but all are connected
    -- to the same data bus and control signals.
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(reset = '1') then
                -- Clear on reset!
                d_out <= (others => '0');
                changed <= '0';
            else
                -- If the serial strobe is high (see serial_io) and our address is
                -- on the bus, write to this register.
                if(strobe = '1' and (my_addr = addr)) then
                    d_out <= d_in;
                    changed <= '1';
                else
                    changed <= '0';
                end if;
            end if;
        end if;
    end process;
end behavioral;
```

Listing A.39: A module to sign extend data. (sign_extend.vhd)

```
--------------------------------------------------------------------------------
-- Sign Extension
-- Last Modified: 20 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
```

```
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
----------------------------------------------------------------------------------

-- We probably don't need this module since VHDL has the sxt() function.  I
-- left it in place to keep the project as close to the Verilog as possible.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sign_extend is
    generic(
        bits_in : integer := 0;
        bits_out    : integer := 0
    );

    port(
        d_in    : in    std_logic_vector(bits_in-1 downto 0);
        d_out   : out   std_logic_vector(bits_out-1 downto 0)
    );
end sign_extend;

architecture behavioral of sign_extend is
begin
    -- Copy d_in to the lower bits_in bits.
    d_out(bits_in-1 downto 0) <= d_in;

    -- Fill the upper bits_out - bits_in bits with the MSB from d_in.
    d_out(bits_out-1 downto bits_in) <= (others => d_in(bits_in-1));
end behavioral;
```

Listing A.40: Generates strobes. These are used for read and write operations, decimation, and interpolation. (strobe_gen.vhd)

```
----------------------------------------------------------------------------------
-- Strobe Generator
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
-------------------------------------------------------------------------------

-- TODO:  The license for reduce_pack is probably not valid with the GPL.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library WORK;
use WORK.REDUCE_PACK.ALL;

entity strobe_gen is
    port(
        clock           : in    std_logic;
        reset           : in    std_logic;
        enable      : in    std_logic;
        rate            : in    std_logic_vector(7 downto 0);   -- 1 less than desired divide
            ratio
        strobe_in   : in    std_logic;
        strobe      : out   std_logic
    );
end strobe_gen;

architecture behavioral of strobe_gen is
    signal counter : std_logic_vector(7 downto 0);
begin
    -- Note that nor_reduce = 1 when counter = "00000000".
    strobe <= '1' when ((nor_reduce(counter) = '1') and (enable = '1') and (strobe_in = '1'))
         else '0';

    -- This is basically just a clock divider that uses a down counter.  For every
    -- 'rate' clocks that strobe_in is asserted, we assert strobe.
    process(clock)
    begin
        if(clock'EVENT and clock = '1') then
            if(reset = '1' or enable = '0') then
                counter <= (others => '0');
            elsif(strobe_in = '1') then
                if(counter = 0) then
                    counter <= rate;
                else
                    counter <= counter - 1;
                end if;
            end if;
        end if;
    end process;
end behavioral;
```

Listing A.41: Controls the FIFO buffer and USB interface on the transmit side. (tx_buffer.vhd)

```
-------------------------------------------------------------------------------
-- Transmit Buffer (DSP <-> USB)
-- Last Modified: 20 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
-------------------------------------------------------------------------------

-- Interface to Cypress FX2 bus.
-- A packet is 512 Bytes.  Each FIFO line is 2 bytes.
-- FIFO has 4096 lines.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity tx_buffer is
    port(
        -- USB side
        usbclk          : in    std_logic;
        bus_reset       : in    std_logic;  -- 257-Hack to fix FX2 bug
        usbdata         : in    std_logic_vector(15 downto 0);
        WR                  : in    std_logic;
        have_space      : out   std_logic;
        tx_underrun     : out   std_logic;
        clear_status    : in    std_logic;

        -- DSP side
        txclk               : in    std_logic;
        reset               : in    std_logic;  -- standard DSP-side reset
        channels            : in    std_logic_vector(3 downto 0);
        tx_i_0          : out   std_logic_vector(15 downto 0);
        tx_q_0          : out   std_logic_vector(15 downto 0);
        tx_i_1          : out   std_logic_vector(15 downto 0);
        tx_q_1          : out   std_logic_vector(15 downto 0);
        txstrobe            : in    std_logic;
        tx_empty            : out   std_logic;
        debugbus            : out   std_logic_vector(31 downto 0)
    );
end tx_buffer;

architecture behavioral of tx_buffer is
    component fifo_4k_18 is
        port(
            data    : in    std_logic_vector(17 downto 0);
            wrreq   : in    std_logic;
            wrclk   : in    std_logic;
            wrfull  : out   std_logic;
            wrempty : out   std_logic;
            wrusedw : out   std_logic_vector(11 downto 0);
            q       : out   std_logic_vector(17 downto 0);
            rdreq   : in    std_logic;
            rdclk   : in    std_logic;
            rdfull  : out   std_logic;
            rdempty : out   std_logic;
            rdusedw : out   std_logic_vector(11 downto 0);
            aclr    : in    std_logic
        );
    end component;

    signal txfifolevel : std_logic_vector(11 downto 0);
    signal fifodata : std_logic_vector(15 downto 0);
    signal rdreq : std_logic;
    signal phase : std_logic_vector(3 downto 0);
    signal sop_f : std_logic;
    signal iq_f : std_logic;
    signal sop : std_logic;

    -- USB side of the FIFO
    signal usbdata_reg : std_logic_vector(15 downto 0);
    signal wr_reg : std_logic;
    signal write_count : std_logic_vector(8 downto 0);

    -- Temporary for FIFO
    signal wr_req_t : std_logic;
    signal tx_empty_t : std_logic;
    signal have_space_t : std_logic;
    signal tx_underrun_t : std_logic;

    signal clear_status_dsp : std_logic;
    signal tx_underrun_dsp : std_logic;
begin
    tx_empty <= tx_empty_t;
    have_space <= have_space_t;
    tx_underrun <= tx_underrun_t;

    process(usbclk)
    begin
        if(usbclk'EVENT and usbclk = '1') then
            -- Be conservative! (4092 - 256)
            if(txfifolevel < "111011111100") then
                have_space_t <= '1';
            else
                have_space_t <= '0';
            end if;
        end if;
    end process;

    process(usbclk)
    begin
        if(usbclk'EVENT and usbclk = '1') then
            wr_reg <= WR;
            usbdata_reg <= usbdata;
        end if;
    end process;

    process(usbclk)
    begin
        if(usbclk'EVENT and usbclk = '1') then
            if(bus_reset = '1') then
                write_count <= (others => '0');
            elsif(wr_reg = '1') then
                write_count <= write_count + 1;
            else
                write_count <= (others => '0');
            end if;
        end if;
    end process;

    process(usbclk)
    begin
        if(usbclk'EVENT and usbclk = '1') then
            sop <= WR and (not wr_reg); -- edge detect
        end if;
    end process;

    -- FIFO
    wr_req_t <= (wr_reg and (not write_count(8)));
    txfifo : fifo_4k_18
        port map(data(17) => sop, data(16) => write_count(0), data(15 downto 0) => usbdata_reg,
            wrreq => wr_req_t, wrclk => usbclk, wrfull => OPEN, wrempty => OPEN,
            wrusedw => txfifolevel, q(17) => sop_f, q(16) => iq_f, q(15 downto 0) => fifodata,
            rdreq => rdreq, rdclk => txclk, rdfull => OPEN, rdempty => tx_empty_t, rdusedw =>
                OPEN,
            aclr => reset);

    -- DAC side of the FIFO
    process(txclk)
    begin
        if(txclk'EVENT and txclk = '1') then
            if(reset = '1') then
                tx_i_0 <= (others => '0');
                tx_q_0 <= (others => '0');
                tx_i_1 <= (others => '0');
                tx_q_1 <= (others => '0');
                phase <= "0000";
            elsif(phase = channels) then
                if(txstrobe = '1') then
                    phase <= "0000";
                end if;
            elsif(tx_empty_t = '0') then
                case phase is
                    when "0000" =>
                        tx_i_0 <= fifodata;
                    when "0001" =>
                        tx_q_0 <= fifodata;
                    when "0010" =>
                        tx_i_1 <= fifodata;
                    when "0011" =>
                        tx_q_1 <= fifodata;
                    when others =>
                        null;
                end case;

                phase <= phase + 1;
            end if;
        end if;
    end process;

    rdreq <= '1' when ((phase /= channels) and (tx_empty_t = '0')) else '0';

    -- Detect underruns, cross clock domains
    process(txclk)
    begin
        if(txclk'EVENT and txclk = '1') then
            clear_status_dsp <= clear_status;
        end if;
    end process;

    process(usbclk)
    begin
        if(usbclk'EVENT and usbclk = '1') then
            tx_underrun_t <= tx_underrun_dsp;
        end if;
    end process;

    process(txclk)
    begin
        if(txclk'EVENT and txclk = '1') then
            if(reset = '1') then
                tx_underrun_dsp <= '0';
            elsif(txstrobe = '1' and (phase /= channels)) then
                tx_underrun_dsp <= '1';
            elsif(clear_status_dsp = '1') then
                tx_underrun_dsp <= '0';
            end if;
        end if;
    end process;

    -- TX debug bus
    -- 15:0     txclk   domain => TXA(15:0)
    -- 31:16    usbclk  domain => RXA(15:0)
    debugbus(0)             <= reset;
    debugbus(1)             <= txstrobe;
    debugbus(2)             <= rdreq;
    debugbus(6 downto 3)    <= phase;
    debugbus(7)             <= tx_empty_t;
    debugbus(8)             <= tx_underrun_dsp;
    debugbus(9)             <= iq_f;
    debugbus(10)            <= sop_f;
    debugbus(14 downto 11)  <= "0000";
    debugbus(15)            <= txclk;

    debugbus(16)            <= bus_reset;
    debugbus(17)            <= WR;
    debugbus(18)            <= wr_reg;
    debugbus(19)            <= have_space_t;
    debugbus(20)            <= write_count(8);
    debugbus(21)            <= write_count(0);
    debugbus(22)            <= sop;
    debugbus(23)            <= tx_underrun_t;
    debugbus(30 downto 24)  <= "0000000";
    debugbus(31)            <= usbclk;
end behavioral;
```

Listing A.42: Controls the interpolation filters for transmit data. (tx_chain.vhd)

```
--------------------------------------------------------------------------------
-- Transmit Chain
-- Last Modified: 22 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------

-- The original Verilog source had partial support for cordic, but it was not
-- used and relied on outdated modules.  According to documentation, the
-- translation from baseband to the IF band is now done on the AD9862 codecs.
-- The 'freq' input is a legacy from the old cordic DUC.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tx_chain is
    port(
        clock                  : in    std_logic;
        reset                  : in    std_logic;
        enable              : in    std_logic;
        interp_rate         : in    std_logic_vector(7 downto 0);
        sample_strobe       : in    std_logic;
        interpolator_strobe : in    std_logic;
        freq                   : in    std_logic_vector(31 downto 0);
        i_in                   : in    std_logic_vector(15 downto 0);
        q_in                   : in    std_logic_vector(15 downto 0);
        i_out                  : out   std_logic_vector(15 downto 0);
        q_out                  : out   std_logic_vector(15 downto 0)
    );
end tx_chain;

architecture behavioral of tx_chain is
    component cic_interp is
        generic(
            bw                     : integer := 16;    -- # of bits for input
            N                      : integer := 4; -- # of filter stages
            log2_of_max_rate    : integer := 7; -- log2 of max sampling rate
            maxbitgain          : integer := 21 --(N-1)*log2_of_max_rate
        );

        port(
            clock          : in    std_logic;
            reset          : in    std_logic;
            enable      : in    std_logic;
            rate           : in    std_logic_vector(7 downto 0);
            strobe_in   : in    std_logic;
            strobe_out  : in    std_logic;
            signal_in   : in    std_logic_vector(bw-1 downto 0);
            signal_out  : out   std_logic_vector(bw-1 downto 0)
        );
    end component;

    signal bb_i : std_logic_vector(15 downto 0);
    signal bb_q : std_logic_vector(15 downto 0);
begin
    cic_interp_i : cic_interp
        port map(clock => clock, reset => reset, enable => enable, rate => interp_rate,
            strobe_in => interpolator_strobe, strobe_out => sample_strobe, signal_in => i_in,
            signal_out => bb_i);

    cic_interp_q : cic_interp
        port map(clock => clock, reset => reset, enable => enable, rate => interp_rate,
            strobe_in => interpolator_strobe, strobe_out => sample_strobe, signal_in => q_in,
            signal_out => bb_q);

    i_out <= bb_i;
    q_out <= bb_q;
end behavioral;
```

Listing A.43: Multiplexer for the usrp_std module that replaces a large ternary expression from the original Verilog source. No Verilog version exists. (usrp_mux.vhd)

```
-------------------------------------------------------------------------------
-- Special MUX for USRP Toplevel Signals
-- Last Modified: 20 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
-------------------------------------------------------------------------------

-- The original Verilog USRP1 contains no equivalent for this component.  In
-- Verilog, this logic is trivial to implement with the ternary ?: operator.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity usrp_mux is
    port(
        dac_mux     : in    std_logic_vector(3 downto 0);
        i_out_0     : in    std_logic_vector(15 downto 0);
        q_out_0     : in    std_logic_vector(15 downto 0);
        i_out_1     : in    std_logic_vector(15 downto 0);
        q_out_1     : in    std_logic_vector(15 downto 0);
        d_out           : out   std_logic_vector(15 downto 0)
    );
end usrp_mux;

architecture behavioral of usrp_mux is
begin
    -- Note that dac_mux(2) is intentionally not used.  I don't know why.
    -- The equivalent Verilog statement is:
    --  d_out = dac_mux[3] ? (dac_mux[1] ? (dac_mux[0] ? q_out_1
    --                                                  : i_out_1)
    --                                   : (dac_mux[0] ? q_out_0
    --                                                  : i_out_0))
    --                     : 16'b0;
    process(dac_mux, i_out_0, q_out_0, i_out_1, q_out_1)
    begin
        if(dac_mux(3) = '1') then
            if(dac_mux(1) = '1') then
                if(dac_mux(0) = '1') then
                    d_out <= q_out_1;
                else
                    d_out <= i_out_1;
                end if;
            else
```

```
                if(dac_mux(0) = '1') then
                    d_out <= q_out_0;
                else
                    d_out <= i_out_0;
                end if;
            end if;
        else
            d_out <= (others => '0');
        end if;
    end process;
end behavioral;
```

Listing A.44: The top-level module for the USRP. Connects the other components together. (usrp_std.vhd)

```
----------------------------------------------------------------------------------------------
-- USRP1 Standard Toplevel
-- Last Modified: 21 July 2010
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 COSMIAC
-- The copyright and license from the original Verilog implementation follows.
--
--
--  USRP - Universal Software Radio Peripheral
--
--  Copyright (C) 2003 Matt Ettus
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
----------------------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library WORK;
use WORK.COMMON_CONFIG.ALL;
use WORK.FPGA_REGS_COMMON.ALL;
use WORK.FPGA_REGS_STANDARD.ALL;

entity usrp_std is
    port(
        MYSTERY_SIGNAL   : out    std_logic;
        master_clk       : in     std_logic;

        -- SPI interface
        SCLK            : in    std_logic;
        SDI             : in    std_logic;
        SDO             : inout std_logic;
        SEN_FPGA        : in    std_logic;

        -- USB interface
        FX2_1           : in    std_logic;
        FX2_2           : out   std_logic;
        FX2_3           : out   std_logic;

        -- Codec interface
        rx_a_a          : in    std_logic_vector(11 downto 0);
        rx_b_a          : in    std_logic_vector(11 downto 0);
        rx_a_b          : in    std_logic_vector(11 downto 0);
        rx_b_b          : in    std_logic_vector(11 downto 0);

        tx_a            : out   std_logic_vector(13 downto 0);
        tx_b            : out   std_logic_vector(13 downto 0);

        TXSYNC_A        : out   std_logic;
        TXSYNC_B        : out   std_logic;

        -- USB interface
        usbclk          : in    std_logic;
        usbctl          : in    std_logic_vector(2 downto 0);
        usbrdy          : out   std_logic_vector(1 downto 0);
        usbdata         : inout std_logic_vector(15 downto 0);

        -- GPIOs to the daughtercard slots
        io_tx_a         : inout std_logic_vector(15 downto 0);
        io_tx_b         : inout std_logic_vector(15 downto 0);
        io_rx_a         : inout std_logic_vector(15 downto 0);
        io_rx_b         : inout std_logic_vector(15 downto 0)
    );
end usrp_std;

architecture behavioral of usrp_std is
    component bustri is
        port(
            data        : in    std_logic_vector(15 downto 0);
            enabledt    : in    std_logic;
            tridata     : out   std_logic_vector(15 downto 0)
        );
    end component;

    component tx_buffer is
        port(
            -- USB side
            usbclk          : in    std_logic;
            bus_reset       : in    std_logic;  -- 257-Hack to fix FX2 bug
            usbdata         : in    std_logic_vector(15 downto 0);
            WR              : in    std_logic;
            have_space      : out   std_logic;
            tx_underrun     : out   std_logic;
            clear_status    : in    std_logic;

            -- DSP side
            txclk          : in    std_logic;
            reset          : in    std_logic;  -- standard DSP-side reset
            channels       : in    std_logic_vector(3 downto 0);
            tx_i_0         : out   std_logic_vector(15 downto 0);
            tx_q_0         : out   std_logic_vector(15 downto 0);
            tx_i_1         : out   std_logic_vector(15 downto 0);
            tx_q_1         : out   std_logic_vector(15 downto 0);
            txstrobe       : in    std_logic;
            tx_empty       : out   std_logic;
            debugbus       : out   std_logic_vector(31 downto 0)
        );
    end component;

    component tx_chain is
        port(
            clock                : in    std_logic;
            reset                : in    std_logic;
            enable               : in    std_logic;
            interp_rate          : in    std_logic_vector(7 downto 0);
            sample_strobe        : in    std_logic;
            interpolator_strobe  : in    std_logic;
            freq                 : in    std_logic_vector(31 downto 0);
            i_in                 : in    std_logic_vector(15 downto 0);
            q_in                 : in    std_logic_vector(15 downto 0);
            i_out                : out   std_logic_vector(15 downto 0);
            q_out                : out   std_logic_vector(15 downto 0)
        );
    end component;

    component setting_reg is
        generic(
            my_addr : integer := 0
        );

        port(
            clock   : in    std_logic;
            reset   : in    std_logic;
            strobe  : in    std_logic;
            addr    : in    std_logic_vector;
            d_in    : in    std_logic_vector(31 downto 0);
            d_out   : out   std_logic_vector(31 downto 0);
            changed : out   std_logic
        );
    end component;

    component usrp_mux is
        port(
            dac_mux     : in    std_logic_vector(3 downto 0);
            i_out_0     : in    std_logic_vector(15 downto 0);
            q_out_0     : in    std_logic_vector(15 downto 0);
            i_out_1     : in    std_logic_vector(15 downto 0);
            q_out_1     : in    std_logic_vector(15 downto 0);
            d_out       : out   std_logic_vector(15 downto 0)
        );
    end component;
```

```

    component adc_interface is
        port(
            clock           : in    std_logic;
            reset           : in    std_logic;
            enable          : in    std_logic;
            serial_addr     : in    std_logic_vector(6 downto 0);
            serial_data     : in    std_logic_vector(31 downto 0);
            serial_strobe   : in    std_logic;
            rx_a_a          : in    std_logic_vector(11 downto 0);
            rx_b_a          : in    std_logic_vector(11 downto 0);
            rx_a_b          : in    std_logic_vector(11 downto 0);
            rx_b_b          : in    std_logic_vector(11 downto 0);
            rssi_0          : out   std_logic_vector(31 downto 0);
            rssi_1          : out   std_logic_vector(31 downto 0);
            rssi_2          : out   std_logic_vector(31 downto 0);
            rssi_3          : out   std_logic_vector(31 downto 0);
            ddc0_in_i       : out   std_logic_vector(15 downto 0);
            ddc0_in_q       : out   std_logic_vector(15 downto 0);
            ddc1_in_i       : out   std_logic_vector(15 downto 0);
            ddc1_in_q       : out   std_logic_vector(15 downto 0);
            ddc2_in_i       : out   std_logic_vector(15 downto 0);
            ddc2_in_q       : out   std_logic_vector(15 downto 0);
            ddc3_in_i       : out   std_logic_vector(15 downto 0);
            ddc3_in_q       : out   std_logic_vector(15 downto 0);
            rx_numchan      : out   std_logic_vector(3 downto 0)
        );
    end component;

    component rx_buffer is
        port(
            -- Read/USB side
            usbclk          : in    std_logic;
            bus_reset       : in    std_logic;
            usbdata         : out   std_logic_vector(15 downto 0);
            RD              : in    std_logic;
            have_pkt_rdy    : out   std_logic;
            rx_overrun      : out   std_logic;
            clear_status    : in    std_logic;

            -- Write/DSP side
            rxclk           : in    std_logic;
            reset           : in    std_logic;  -- DSP-side reset, not for registers
            rxstrobe        : in    std_logic;
            channels        : in    std_logic_vector(3 downto 0);
            ch_0            : in    std_logic_vector(15 downto 0);
            ch_1            : in    std_logic_vector(15 downto 0);
            ch_2            : in    std_logic_vector(15 downto 0);
            ch_3            : in    std_logic_vector(15 downto 0);
            ch_4            : in    std_logic_vector(15 downto 0);
            ch_5            : in    std_logic_vector(15 downto 0);
            ch_6            : in    std_logic_vector(15 downto 0);
            ch_7            : in    std_logic_vector(15 downto 0);

            -- Settings, also using rxclk
            serial_addr     : in    std_logic_vector(6 downto 0);
            serial_data     : in    std_logic_vector(31 downto 0);
            serial_strobe   : in    std_logic;
            reset_regs      : in    std_logic;  -- Only resets regs
            debugbus        : out   std_logic_vector(31 downto 0)
        );
    end component;

    component rx_chain is
        generic(
            FREQADDR : integer := 0;
            PHASEADDR : integer := 0
        );

        port(
            clock               : in    std_logic;
            reset               : in    std_logic;
            enable              : in    std_logic;
            decim_rate          : in    std_logic_vector(7 downto 0);
            sample_strobe       : in    std_logic;
            decimator_strobe    : in    std_logic;
            hb_strobe           : out   std_logic;
            serial_addr         : in    std_logic_vector(6 downto 0);
            serial_data         : in    std_logic_vector(31 downto 0);
            serial_strobe       : in    std_logic;
            i_in                : in    std_logic_vector(15 downto 0);
            q_in                : in    std_logic_vector(15 downto 0);
            i_out               : out   std_logic_vector(15 downto 0);
            q_out               : out   std_logic_vector(15 downto 0);
            debugdata           : out   std_logic_vector(15 downto 0);
            debugctrl           : out   std_logic_vector(15 downto 0)
        );
    end component;

    component serial_io is
        port(
            master_clk      : in    std_logic;
            serial_clock    : in    std_logic;
            serial_data_in  : in    std_logic;
            enable          : in    std_logic;
            reset           : in    std_logic;
            serial_data_out : out   std_logic;
            serial_addr     : out   std_logic_vector(6 downto 0);
            serial_data     : out   std_logic_vector(31 downto 0);
            serial_strobe   : out   std_logic;
            readback_0      : in    std_logic_vector(31 downto 0);
            readback_1      : in    std_logic_vector(31 downto 0);
            readback_2      : in    std_logic_vector(31 downto 0);
            readback_3      : in    std_logic_vector(31 downto 0);
            readback_4      : in    std_logic_vector(31 downto 0);
            readback_5      : in    std_logic_vector(31 downto 0);
            readback_6      : in    std_logic_vector(31 downto 0);
            readback_7      : in    std_logic_vector(31 downto 0)
        );
    end component;

    component master_control is
        port(
            master_clk          : in    std_logic;
            usbclk              : in    std_logic;
            serial_addr         : in    std_logic_vector(6 downto 0);
            serial_data         : in    std_logic_vector(31 downto 0);
            serial_strobe       : in    std_logic;
            tx_bus_reset        : out   std_logic;
            rx_bus_reset        : out   std_logic;
            tx_dsp_reset        : out   std_logic;
            rx_dsp_reset        : out   std_logic;
            enable_tx           : out   std_logic;
            enable_rx           : out   std_logic;
            interp_rate         : out   std_logic_vector(7 downto 0);
            decim_rate          : out   std_logic_vector(7 downto 0);
            tx_sample_strobe    : out   std_logic;
            strobe_interp       : out   std_logic;
            rx_sample_strobe    : out   std_logic;
            strobe_decim        : out   std_logic;
            tx_empty            : in    std_logic;
            debug_0             : in    std_logic_vector(15 downto 0);
            debug_1             : in    std_logic_vector(15 downto 0);
            debug_2             : in    std_logic_vector(15 downto 0);
            debug_3             : in    std_logic_vector(15 downto 0);
            reg_0               : out   std_logic_vector(15 downto 0);
            reg_1               : out   std_logic_vector(15 downto 0);
            reg_2               : out   std_logic_vector(15 downto 0);
            reg_3               : out   std_logic_vector(15 downto 0)
        );
    end component;

    component io_pins is
        port(
            io_0            : inout std_logic_vector(15 downto 0);
            io_1            : inout std_logic_vector(15 downto 0);
            io_2            : inout std_logic_vector(15 downto 0);
            io_3            : inout std_logic_vector(15 downto 0);
            reg_0           : in    std_logic_vector(15 downto 0);
            reg_1           : in    std_logic_vector(15 downto 0);
            reg_2           : in    std_logic_vector(15 downto 0);
            reg_3           : in    std_logic_vector(15 downto 0);
            clock           : in    std_logic;
            rx_reset        : in    std_logic;
            tx_reset        : in    std_logic;
            serial_addr     : in    std_logic_vector(6 downto 0);
            serial_data     : in    std_logic_vector(31 downto 0);
            serial_strobe   : in    std_logic
        );
    end component;

    signal debugdata : std_logic_vector(15 downto 0);
    signal debugctrl : std_logic_vector(15 downto 0);

    signal clk64 : std_logic;
    signal clk128 : std_logic;

    signal WR : std_logic;
    signal RD : std_logic;
    signal OE : std_logic;

    signal have_space : std_logic;
    signal have_pkt_rdy : std_logic;

    signal tx_underrun : std_logic;
    signal rx_overrun : std_logic;
    signal clear_status : std_logic;

    signal usbdata_out : std_logic_vector(15 downto 0);

    signal dac0mux : std_logic_vector(3 downto 0);
    signal dac1mux : std_logic_vector(3 downto 0);
    signal dac2mux : std_logic_vector(3 downto 0);
    signal dac3mux : std_logic_vector(3 downto 0);

    signal tx_realsignals : std_logic;
    signal rx_numchan : std_logic_vector(3 downto 0);
    signal tx_numchan : std_logic_vector(2 downto 0);

    signal interp_rate : std_logic_vector(7 downto 0);
    signal decim_rate : std_logic_vector(7 downto 0);

    signal tx_debugbus : std_logic_vector(31 downto 0);
    signal rx_debugbus : std_logic_vector(31 downto 0);

    signal enable_tx : std_logic;
    signal enable_rx : std_logic;

    signal tx_dsp_reset : std_logic;
    signal rx_dsp_reset : std_logic;
    signal tx_bus_reset : std_logic;
    signal rx_bus_reset : std_logic;

    signal settings : std_logic_vector(7 downto 0);

    signal ch0tx : std_logic_vector(15 downto 0);
    signal ch1tx : std_logic_vector(15 downto 0);
    signal ch2tx : std_logic_vector(15 downto 0);
    signal ch3tx : std_logic_vector(15 downto 0);

    signal ch0rx : std_logic_vector(15 downto 0);
    signal ch1rx : std_logic_vector(15 downto 0);
    signal ch2rx : std_logic_vector(15 downto 0);
    signal ch3rx : std_logic_vector(15 downto 0);
    signal ch4rx : std_logic_vector(15 downto 0);
    signal ch5rx : std_logic_vector(15 downto 0);
    signal ch6rx : std_logic_vector(15 downto 0);
    signal ch7rx : std_logic_vector(15 downto 0);

    -- TX
    signal i_out_0 : std_logic_vector(15 downto 0);
    signal i_out_1 : std_logic_vector(15 downto 0);
    signal q_out_0 : std_logic_vector(15 downto 0);
    signal q_out_1 : std_logic_vector(15 downto 0);

    signal bb_tx_i0 : std_logic_vector(15 downto 0);
    signal bb_tx_q0 : std_logic_vector(15 downto 0);
    signal bb_tx_i1 : std_logic_vector(15 downto 0);
    signal bb_tx_q1 : std_logic_vector(15 downto 0);

    signal strobe_interp : std_logic;
    signal tx_sample_strobe : std_logic;
    signal tx_empty : std_logic;

    signal serial_strobe : std_logic;
    signal serial_addr : std_logic_vector(6 downto 0);
    signal serial_data : std_logic_vector(31 downto 0);

    signal debug_counter : std_logic_vector(15 downto 0);
    signal loopback_i_0 : std_logic_vector(15 downto 0);
    signal loopback_q_0 : std_logic_vector(15 downto 0);

    signal tx_a_a : std_logic_vector(15 downto 0);
    signal tx_b_a : std_logic_vector(15 downto 0);
    signal tx_a_b : std_logic_vector(15 downto 0);
    signal tx_b_b : std_logic_vector(15 downto 0);

    signal txsync : std_logic;

    -- RX
    signal rx_sample_strobe : std_logic;
    signal strobe_decim : std_logic;
    signal hb_strobe : std_logic;

    signal bb_rx_i0 : std_logic_vector(15 downto 0);
    signal bb_rx_q0 : std_logic_vector(15 downto 0);
    signal bb_rx_i1 : std_logic_vector(15 downto 0);
    signal bb_rx_q1 : std_logic_vector(15 downto 0);
    signal bb_rx_i2 : std_logic_vector(15 downto 0);
    signal bb_rx_q2 : std_logic_vector(15 downto 0);
    signal bb_rx_i3 : std_logic_vector(15 downto 0);
    signal bb_rx_q3 : std_logic_vector(15 downto 0);

    signal loopback : std_logic;
    signal counter : std_logic;

    signal ddc0_in_i : std_logic_vector(15 downto 0);
    signal ddc0_in_q : std_logic_vector(15 downto 0);
    signal ddc1_in_i : std_logic_vector(15 downto 0);
    signal ddc1_in_q : std_logic_vector(15 downto 0);
    signal ddc2_in_i : std_logic_vector(15 downto 0);
    signal ddc2_in_q : std_logic_vector(15 downto 0);
    signal ddc3_in_i : std_logic_vector(15 downto 0);
    signal ddc3_in_q : std_logic_vector(15 downto 0);

    signal rssi_0 : std_logic_vector(31 downto 0);
    signal rssi_1 : std_logic_vector(31 downto 0);
    signal rssi_2 : std_logic_vector(31 downto 0);
    signal rssi_3 : std_logic_vector(31 downto 0);

    signal capabilities : std_logic_vector(31 downto 0);

    signal reg_0 : std_logic_vector(15 downto 0);
    signal reg_1 : std_logic_vector(15 downto 0);
    signal reg_2 : std_logic_vector(15 downto 0);
    signal reg_3 : std_logic_vector(15 downto 0);
begin
    MYSTERY_SIGNAL <= '0';

    -- USB STUFF -----------------------------------------------------------------
    WR <= usbctl(0);
    RD <= usbctl(1);
    OE <= usbctl(2);

    usbrdy(0) <= have_space;
    usbrdy(1) <= have_pkt_rdy;

    clear_status <= FX2_1;
    FX2_2 <= rx_overrun;
    FX2_3 <= tx_underrun;

    -- Tri-state bus macro
    bustri_0 : bustri
        port map(data => usbdata_out, enabledt => OE, tridata => usbdata);

    clk64 <= master_clk;

    -- TRANSMIT SIDE -------------------------------------------------------------
    tx_en_t : if(TX_EN) generate
        bb_tx_i0 <= ch0tx;
        bb_tx_q0 <= ch1tx;
        bb_tx_i1 <= ch2tx;
        bb_tx_q1 <= ch3tx;

        tx_buffer_0 : tx_buffer
            port map(usbclk => usbclk, bus_reset => tx_bus_reset, usbdata => usbdata, WR => WR,
                have_space => have_space, tx_underrun => tx_underrun, clear_status =>
                    clear_status,
                txclk => clk64, reset => tx_dsp_reset, channels(3 downto 1) => tx_numchan,
                channels(0) => '0', tx_i_0 => ch0tx, tx_q_0 => ch1tx, tx_i_1 => ch2tx, tx_q_1 =>
                    ch3tx,
                txstrobe => strobe_interp, tx_empty => tx_empty, debugbus => tx_debugbus);

        -- FIXME Should freq really be 0?
        tx_en_0_t : if(TX_EN_0) generate
            tx_chain_0 : tx_chain
                port map(clock => clk64, reset => tx_dsp_reset, enable => enable_tx,
                    interp_rate => interp_rate, sample_strobe => tx_sample_strobe,
                    interpolator_strobe => strobe_interp, freq => (others => '0'), i_in =>
                        bb_tx_i0,
                    q_in => bb_tx_q0, i_out => i_out_0, q_out => q_out_0);
        end generate tx_en_0_t;

        tx_en_0_f : if(not TX_EN_0) generate
            i_out_0 <= (others => '0');
            q_out_0 <= (others => '0');
        end generate tx_en_0_f;

        -- FIXME should freq really be 0?
        tx_en_1_t : if(TX_EN_1) generate
            tx_chain_1 : tx_chain
                port map(clock => clk64, reset => tx_dsp_reset, enable => enable_tx,
                  interp_rate => interp_rate, sample_strobe => tx_sample_strobe,
                  interpolator_strobe => strobe_interp, freq => (others => '0'), i_in =>
                      bb_tx_i1,
                  q_in => bb_tx_q1, i_out => i_out_1, q_out => q_out_1);
        end generate tx_en_1_t;

        tx_en_1_f : if(not TX_EN_1) generate
            i_out_1 <= (others => '0');
            q_out_1 <= (others => '0');
        end generate tx_en_1_f;

        sr_txmux : setting_reg
            generic map(my_addr => FR_TX_MUX)
            port map(clock => clk64, reset => tx_dsp_reset, strobe => serial_strobe, addr =>
                 serial_addr,
                d_in => serial_data, d_out(31 downto 20) => OPEN, d_out(19 downto 16) => dac3mux,
                d_out(15 downto 12) => dac2mux, d_out(11 downto 8) => dac1mux, d_out(7 downto 4)
                    => dac0mux,
                d_out(3) => tx_realsignals, d_out(2 downto 0) => tx_numchan, changed => OPEN);

        tx_mux_0 : usrp_mux
            port map(dac_mux => dac0mux, i_out_0 => i_out_0, q_out_0 => q_out_0, i_out_1 =>
                 i_out_1,
                q_out_1 => q_out_1, d_out => tx_a_a);

        tx_mux_1 : usrp_mux
            port map(dac_mux => dac1mux, i_out_0 => i_out_0, q_out_0 => q_out_0, i_out_1 =>
                 i_out_1,
                q_out_1 => q_out_1, d_out => tx_b_a);

        tx_mux_2 : usrp_mux
            port map(dac_mux => dac2mux, i_out_0 => i_out_0, q_out_0 => q_out_0, i_out_1 =>
                 i_out_1,
                q_out_1 => q_out_1, d_out => tx_a_b);

        tx_mux_3 : usrp_mux
            port map(dac_mux => dac3mux, i_out_0 => i_out_0, q_out_0 => q_out_0, i_out_1 =>
                 i_out_1,
                q_out_1 => q_out_1, d_out => tx_b_b);

        txsync <= tx_sample_strobe;
        TXSYNC_A <= txsync;
        TXSYNC_B <= txsync;

        tx_a <= tx_b_a(15 downto 2) when txsync = '1' else tx_a_a(15 downto 2);
        tx_b <= tx_b_b(15 downto 2) when txsync = '1' else tx_a_b(15 downto 2);
    end generate tx_en_t;

    tx_en_f : if(not TX_EN) generate
        -- TODO Assign signals!
    end generate tx_en_f;

    -- RECEIVE SIDE ---------------------------------------------------------------
    rx_en_t : if(RX_EN) generate
        loopback <= settings(0);
        counter <= settings(1);

        process(clk64)
        begin
            if(clk64'EVENT and clk64 = '1') then
                if(rx_dsp_reset = '1') then
                    debug_counter <= (others => '0');
                elsif(enable_rx /= '1') then
                    debug_counter <= (others => '0');
                elsif(hb_strobe = '1') then
                    debug_counter <= debug_counter + 2;
                end if;
            end if;
        end process;

        process(clk64)
        begin
            if(clk64'EVENT and clk64 = '1') then
                if(strobe_interp = '1') then
                    loopback_i_0 <= ch0tx;
                    loopback_q_0 <= ch1tx;
                end if;
            end if;
        end process;

        ch0rx <= debug_counter when (counter = '1') else loopback_i_0 when (loopback = '1') else
             bb_rx_i0;
        ch1rx <= debug_counter + 1 when (counter = '1') else loopback_q_0 when (loopback = '1')
             else bb_rx_q0;
        ch2rx <= bb_rx_i1;
        ch3rx <= bb_rx_q1;
        ch4rx <= bb_rx_i2;
        ch5rx <= bb_rx_q2;
        ch6rx <= bb_rx_i3;
        ch7rx <= bb_rx_q3;

        adc_interface_0 : adc_interface
            port map(clock => clk64, reset => rx_dsp_reset, enable => '1', serial_addr =>
                 serial_addr,
                serial_data => serial_data, serial_strobe => serial_strobe, rx_a_a => rx_a_a,
                rx_b_a => rx_b_a, rx_a_b => rx_a_b, rx_b_b => rx_b_b, rssi_0 => rssi_0,
                rssi_1 => rssi_1, rssi_2 => rssi_2, rssi_3 => rssi_3, ddc0_in_i => ddc0_in_i,
                ddc0_in_q => ddc0_in_q, ddc1_in_i => ddc1_in_i, ddc1_in_q => ddc1_in_q,
                ddc2_in_i => ddc2_in_i, ddc2_in_q => ddc2_in_q, ddc3_in_i => ddc3_in_i,
                ddc3_in_q => ddc3_in_q, rx_numchan => rx_numchan);

        rx_buffer_0 : rx_buffer
            port map(usbclk => usbclk, bus_reset => rx_bus_reset, reset => rx_dsp_reset,
                reset_regs => rx_dsp_reset, usbdata => usbdata_out, RD => RD, have_pkt_rdy =>
                     have_pkt_rdy,
                rx_overrun => rx_overrun, channels => rx_numchan, ch_0 => ch0rx, ch_1 => ch1rx,
                ch_2 => ch2rx, ch_3 => ch3rx, ch_4 => ch4rx, ch_5 => ch5rx, ch_6 => ch6rx, ch_7
                    => ch7rx,
                rxclk => clk64, rxstrobe => hb_strobe, clear_status => clear_status,
                serial_addr => serial_addr, serial_data => serial_data, serial_strobe =>
                    serial_strobe,
                debugbus => rx_debugbus);

        rx_en_0_t : if(RX_EN_0) generate
```

```
        rx_chain_0 : rx_chain
            generic map(FREQADDR => FR_RX_FREQ_0, PHASEADDR => FR_RX_PHASE_0)
            port map(clock => clk64, reset => '0', enable => enable_rx, decim_rate =>
                 decim_rate,
                sample_strobe => rx_sample_strobe, decimator_strobe => strobe_decim,
                hb_strobe => hb_strobe, serial_addr => serial_addr, serial_data =>
                     serial_data,
                serial_strobe => serial_strobe, i_in => ddc0_in_i, q_in => ddc0_in_q, i_out
                     => bb_rx_i0,
                q_out => bb_rx_q0, debugdata => debugdata, debugctrl => debugctrl);
    end generate rx_en_0_t;

    rx_en_0_f : if(not RX_EN_0) generate
        bb_rx_i0 <= (others => '0');
        bb_rx_q0 <= (others => '0');
    end generate rx_en_0_f;

    rx_en_1_t : if(RX_EN_1) generate
        rx_chain_1 : rx_chain
            generic map(FREQADDR => FR_RX_FREQ_1, PHASEADDR => FR_RX_PHASE_1)
            port map(clock => clk64, reset => '0', enable => enable_rx, decim_rate =>
                 decim_rate,
                sample_strobe => rx_sample_strobe, decimator_strobe => strobe_decim,
                hb_strobe => OPEN, serial_addr => serial_addr, serial_data => serial_data,
                serial_strobe => serial_strobe, i_in => ddc1_in_i, q_in => ddc1_in_q, i_out
                     => bb_rx_i1,
                q_out => bb_rx_q1, debugdata => OPEN, debugctrl => OPEN);
    end generate rx_en_1_t;

    rx_en_1_f : if(not RX_EN_1) generate
        bb_rx_i1 <= (others => '0');
        bb_rx_q1 <= (others => '0');
    end generate rx_en_1_f;

    rx_en_2_t : if(RX_EN_2) generate
        rx_chain_2 : rx_chain
            generic map(FREQADDR => FR_RX_FREQ_2, PHASEADDR => FR_RX_PHASE_2)
            port map(clock => clk64, reset => '0', enable => enable_rx, decim_rate =>
                 decim_rate,
                sample_strobe => rx_sample_strobe, decimator_strobe => strobe_decim,
                hb_strobe => OPEN, serial_addr => serial_addr, serial_data => serial_data,
                serial_strobe => serial_strobe, i_in => ddc2_in_i, q_in => ddc2_in_q, i_out
                     => bb_rx_i2,
                q_out => bb_rx_q2, debugdata => OPEN, debugctrl => OPEN);
    end generate rx_en_2_t;

    rx_en_2_f : if(not RX_EN_2) generate
        bb_rx_i2 <= (others => '0');
        bb_rx_q2 <= (others => '0');
    end generate rx_en_2_f;

    rx_en_3_t : if(RX_EN_3) generate
        rx_chain_3 : rx_chain
            generic map(FREQADDR => FR_RX_FREQ_3, PHASEADDR => FR_RX_PHASE_3)
            port map(clock => clk64, reset => '0', enable => enable_rx, decim_rate =>
                 decim_rate,
                sample_strobe => rx_sample_strobe, decimator_strobe => strobe_decim,
                hb_strobe => OPEN, serial_addr => serial_addr, serial_data => serial_data,
                serial_strobe => serial_strobe, i_in => ddc3_in_i, q_in => ddc3_in_q, i_out
                     => bb_rx_i3,
                q_out => bb_rx_q3, debugdata => OPEN, debugctrl => OPEN);
    end generate rx_en_3_t;

    rx_en_3_f : if(not RX_EN_3) generate
        bb_rx_i3 <= (others => '0');
        bb_rx_q3 <= (others => '0');
    end generate rx_en_3_f;
end generate rx_en_t;

rx_en_f : if(not RX_EN) generate
    -- TODO Assign signals!
end generate rx_en_f;

-- CONTROL SIGNALS -----------------------------------------------------------
capabilities(31 downto 8) <= (others => '0');
capabilities(7) <= '1' when TX_CAP_HB else '0';
capabilities(6 downto 4) <= TX_CAP_NCHAN;
capabilities(3) <= '1' when RX_CAP_HB else '0';
capabilities(2 downto 0) <= RX_CAP_NCHAN;

-- readback_3: 0xF0F0931A is greater than interger'high, so I used binary notation instead.
serial_io_0 : serial_io
    port map(master_clk => clk64, serial_clock => SCLK, serial_data_in => SDI, enable =>
        SEN_FPGA,
        reset => '0', serial_data_out => SDO, serial_addr => serial_addr, serial_data =>
                serial_data,
        serial_strobe => serial_strobe, readback_0(31 downto 16) => io_rx_a,
        readback_0(15 downto 0) => io_tx_a, readback_1(31 downto 16) => io_rx_b,
        readback_1(15 downto 0) => io_tx_b, readback_2 => capabilities,
        readback_3 => "11110000111100001001001100011010", readback_4 => rssi_0, readback_5 =>
                rssi_1,
        readback_6 => rssi_2, readback_7 => rssi_3);

master_control_0 : master_control
    port map(master_clk => clk64, usbclk => usbclk, serial_addr => serial_addr,
        serial_data => serial_data, serial_strobe => serial_strobe, tx_bus_reset =>
                tx_bus_reset,
        rx_bus_reset => rx_bus_reset, tx_dsp_reset => tx_dsp_reset, rx_dsp_reset =>
                rx_dsp_reset,
        enable_tx => enable_tx, enable_rx => enable_rx, interp_rate => interp_rate,
        decim_rate => decim_rate, tx_sample_strobe => tx_sample_strobe, strobe_interp =>
                strobe_interp,
        rx_sample_strobe => rx_sample_strobe, strobe_decim => strobe_decim, tx_empty =>
                tx_empty,
        debug_0 => tx_debugbus(15 downto 0), debug_1 => tx_debugbus(31 downto 16),
        debug_2 => rx_debugbus(15 downto 0), debug_3 => rx_debugbus(31 downto 16), reg_0 =>
                reg_0,
        reg_1 => reg_1, reg_2 => reg_2, reg_3 => reg_3);

io_pins_0 : io_pins
    port map(io_0 => io_tx_a, io_1 => io_rx_a, io_2 => io_tx_b, io_3 => io_rx_b, reg_0 =>
        reg_0,
        reg_1 => reg_1, reg_2 => reg_2, reg_3 => reg_3, clock => clk64, rx_reset =>
                rx_dsp_reset,
        tx_reset => tx_dsp_reset, serial_addr => serial_addr, serial_data => serial_data,
        serial_strobe => serial_strobe);

-- MISC SETTINGS ----------------------------------------------------------------
sr_misc : setting_reg
    generic map(my_addr => FR_MODE)
    port map(clock => clk64, reset => rx_dsp_reset, strobe => serial_strobe, addr =>
         serial_addr,
        d_in => serial_data, d_out(31 downto 8) => OPEN, d_out(7 downto 0) => settings,
        changed => OPEN);
end behavioral;
```

Appendix B

FPGA Board Description (XBD)

This appendix contains the Xilinx Board Definition [21] file for the COSMIAC FPGA board. This file is meant for use with Xilinx EDK. Please note that flash memory support is missing and that the DDR2 memory needs more information from the Memory Interface Generator (MIG).

Listing B.1: COSMIAC FPGA Board Description (XBD)

```
#-------------------------------------------------------------------------------
# Xilinx Board Definition for the SPA-U FPGA Board
# Last Modified: 13 April 2011
# XBD Author: Steve Olivieri
#
# Copyright (C) 2010 COSMIAC
#
#  This program is free software; you can redistribute it and/or modify
#  it under the terms of the GNU General Public License as published by
#  the Free Software Foundation; either version 2 of the License, or
#  (at your option) any later version.
#
#  This program is distributed in the hope that it will be useful,
#  but WITHOUT ANY WARRANTY; without even the implied warranty of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#  GNU General Public License for more details.
#
#  You should have received a copy of the GNU General Public License
#  along with this program; if not, write to the Free Software
#  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
#-------------------------------------------------------------------------------

### TODO ###
# USB Reset (H/L? Pin?)
# DDR2 Memory (MIG output)
# Flash Memory
# I2C Controller
# Fix SPI SS Bus (USRP, # of SS)

# Vendor Information
ATTRIBUTE VENDOR = COSMIAC
ATTRIBUTE SPEC_URL = www.cosmiac.com
ATTRIBUTE CONTACT_INFO_URL = http://www.cosmiac.com/

# Board Information
ATTRIBUTE NAME = SPA-U FPGA Board
ATTRIBUTE REVISION = B
ATTRIBUTE DESC = SPA-U FPGA Board
ATTRIBUTE LONG_DESC = 'The SPA-U FPGA Board utilizes a Xilinx Spartan-3A
XCS1400A-5FG484 device and has DDR2 memory, flash memory, USB2, I2C, SPI, RS232,
Ethernet, and GPIO capabilities.'

# Primary Clock (50MHz)
BEGIN IO_INTERFACE
  ATTRIBUTE IOTYPE = XIL_CLOCK_V1
  ATTRIBUTE INSTANCE = CLOCK_50MHZ

  PARAMETER CLK_FREQ = 50000000, IO_IS = clk_freq, RANGE = (50000000)

  PORT CLK = OSC_50_MHZ, IO_IS = ext_clk
END

# Second Clock (75MHz)
BEGIN IO_INTERFACE
  ATTRIBUTE IOTYPE = XIL_CLOCK_V1
  ATTRIBUTE INSTANCE = CLOCK_75MHZ

  PARAMETER CLK_FREQ = 75000000, IO_IS = clk_freq, RANGE = (75000000)

  PORT CLK = OSC_75_MHZ, IO_IS = ext_clk
END

# Third Clock (100MHz)
BEGIN IO_INTERFACE
  ATTRIBUTE IOTYPE = XIL_CLOCK_V1
  ATTRIBUTE INSTANCE = CLOCK_100MHZ

  PARAMETER CLK_FREQ = 100000000, IO_IS = clk_freq, RANGE = (100000000)

  PORT CLK = OSC_100_MHZ, IO_IS = ext_clk
END

# Reset Button
BEGIN IO_INTERFACE
  ATTRIBUTE IOTYPE = XIL_RESET_V1
  ATTRIBUTE INSTANCE = RESET

  PARAMETER RST_POLARITY = 1, IO_IS = polarity, VALUE_NOTE = Active High

  PORT RESET = CONN_RESET, IO_IS = ext_rst
END

# Primary UART
BEGIN IO_INTERFACE
```

```
  ATTRIBUTE IOTYPE = XIL_UART_V1
  ATTRIBUTE INSTANCE = RS232_PRIMARY

  PORT FROM_RS232 = CONN_FROM_RS232, IO_IS = serial_in
  PORT TO_RS232 = CONN_TO_RS232, IO_IS = serial_out , INITIALVAL = GND
END

# AT90 UART
BEGIN IO_INTERFACE
  ATTRIBUTE IOTYPE = XIL_UART_V1
  ATTRIBUTE INSTANCE = RS232_AT90

  PORT AT90_RXD1 = CONN_AT90_RXD1, IO_IS = serial_in
  PORT AT90_TXD1 = CONN_AT90_TXD1, IO_IS = serial_out , INITIALVAL = GND
END

# LEDs
BEGIN IO_INTERFACE
  ATTRIBUTE IOTYPE = XIL_GPIO_V1
  ATTRIBUTE INSTANCE = LEDs_8BIT

  PARAMETER num_bits = 8, IO_IS = num_bits
  PARAMETER is_dual = 0, IO_IS = is_dual
  PARAMETER bidir_data = 0, IO_IS = is_bidir
  PARAMETER all_inputs = 0, IO_IS = all_inputs

  PORT LED0 = CONN_LED0, IO_IS = gpio_data_out[0], INITIALVAL = VCC
  PORT LED1 = CONN_LED1, IO_IS = gpio_data_out[1], INITIALVAL = VCC
  PORT LED2 = CONN_LED2, IO_IS = gpio_data_out[2], INITIALVAL = VCC
  PORT LED3 = CONN_LED3, IO_IS = gpio_data_out[3], INITIALVAL = VCC
  PORT LED4 = CONN_LED4, IO_IS = gpio_data_out[4], INITIALVAL = VCC
  PORT LED5 = CONN_LED5, IO_IS = gpio_data_out[5], INITIALVAL = VCC
  PORT LED6 = CONN_LED6, IO_IS = gpio_data_out[6], INITIALVAL = VCC
  PORT LED7 = CONN_LED7, IO_IS = gpio_data_out[7], INITIALVAL = VCC
END

# Push Button
# The button is currently used for the system reset.
#BEGIN IO_INTERFACE
#  ATTRIBUTE IOTYPE    = XIL_GPIO_V1
#  ATTRIBUTE INSTANCE = PUSH_BUTTON
#
#  PARAMETER num_bits = 1, IO_IS = num_bits
#  PARAMETER is_dual = 0, IO_IS = is_dual
#  PARAMETER bidir_data = 0, IO_IS = is_bidir
#  PARAMETER all_inputs = 1, IO_IS = all_inputs
#
#  PORT BUTTON = CONN_BUTTON, IO_IS = gpio_data_in[0]
#END

# AT90 I/O Port A
BEGIN IO_INTERFACE
  ATTRIBUTE IOTYPE = XIL_GPIO_V1
  ATTRIBUTE INSTANCE = AT90_PA

  PARAMETER num_bits = 8, IO_IS = num_bits
  PARAMETER is_dual = 0, IO_IS = is_dual
  PARAMETER bidir_data = 0, IO_IS = is_bidir
  PARAMETER all_inputs = 0, IO_IS = all_inputs

  PORT AT_PA0 = CONN_AT_PA0, IO_IS = gpio_data_out[0], INITIALVAL = GND
  PORT AT_PA1 = CONN_AT_PA1, IO_IS = gpio_data_out[1], INITIALVAL = GND
  PORT AT_PA2 = CONN_AT_PA2, IO_IS = gpio_data_out[2], INITIALVAL = GND
  PORT AT_PA3 = CONN_AT_PA3, IO_IS = gpio_data_out[3], INITIALVAL = GND
  PORT AT_PA4 = CONN_AT_PA4, IO_IS = gpio_data_out[4], INITIALVAL = GND
  PORT AT_PA5 = CONN_AT_PA5, IO_IS = gpio_data_out[5], INITIALVAL = GND
  PORT AT_PA6 = CONN_AT_PA6, IO_IS = gpio_data_out[6], INITIALVAL = GND
  PORT AT_PA7 = CONN_AT_PA7, IO_IS = gpio_data_out[7], INITIALVAL = GND
END

# AT90 I/O Port C
BEGIN IO_INTERFACE
  ATTRIBUTE IOTYPE = XIL_GPIO_V1
  ATTRIBUTE INSTANCE = AT90_PC

  PARAMETER num_bits = 8, IO_IS = num_bits
  PARAMETER is_dual = 0, IO_IS = is_dual
  PARAMETER bidir_data = 0, IO_IS = is_bidir
  PARAMETER all_inputs = 0, IO_IS = all_inputs

  PORT AT_PC0 = CONN_AT_PC0, IO_IS = gpio_data_out[0], INITIALVAL = GND
  PORT AT_PC1 = CONN_AT_PC1, IO_IS = gpio_data_out[1], INITIALVAL = GND
  PORT AT_PC2 = CONN_AT_PC2, IO_IS = gpio_data_out[2], INITIALVAL = GND
  PORT AT_PC3 = CONN_AT_PC3, IO_IS = gpio_data_out[3], INITIALVAL = GND
  PORT AT_PC4 = CONN_AT_PC4, IO_IS = gpio_data_out[4], INITIALVAL = GND
  PORT AT_PC5 = CONN_AT_PC5, IO_IS = gpio_data_out[5], INITIALVAL = GND
  PORT AT_PC6 = CONN_AT_PC6, IO_IS = gpio_data_out[6], INITIALVAL = GND
  PORT AT_PC7 = CONN_AT_PC7, IO_IS = gpio_data_out[7], INITIALVAL = GND
END

# AT90 I/O Port F
BEGIN IO_INTERFACE
  ATTRIBUTE IOTYPE = XIL_GPIO_V1
  ATTRIBUTE INSTANCE = AT90_PF

  PARAMETER num_bits = 4, IO_IS = num_bits
  PARAMETER is_dual = 0, IO_IS = is_dual
  PARAMETER bidir_data = 0, IO_IS = is_bidir
  PARAMETER all_inputs = 0, IO_IS = all_inputs

  PORT AT_PF0 = CONN_AT_PF0, IO_IS = gpio_data_out[0], INITIALVAL = GND
  PORT AT_PF1 = CONN_AT_PF1, IO_IS = gpio_data_out[1], INITIALVAL = GND
  PORT AT_PF2 = CONN_AT_PF2, IO_IS = gpio_data_out[2], INITIALVAL = GND
  PORT AT_PF3 = CONN_AT_PF3, IO_IS = gpio_data_out[3], INITIALVAL = GND
END

# Ethernet
BEGIN IO_INTERFACE
  ATTRIBUTE IOTYPE    = XIL_ETHERNET_V1
  ATTRIBUTE INSTANCE = ETHERNET

  PORT E_COL = CONN_E_COL, IO_IS = ETH_COL
  PORT E_CRS = CONN_E_CRS, IO_IS = ETH_CRS
  PORT E_MDC = CONN_E_MDC, IO_IS = ETH_MDC
  PORT E_MDIO = CONN_E_MDIO, IO_IS = ETH_MDIO
  PORT E_NRST = CONN_E_NRST, IO_IS = PHY_RESETn

  PORT E_RX_CLK = CONN_E_RX_CLK, IO_IS = ETH_RXC
  PORT E_RX_DV = CONN_E_RX_DV, IO_IS = ETH_RXDV
  PORT E_RXD0 = CONN_E_RXD0, IO_IS = ETH_RXD[0]
  PORT E_RXD1 = CONN_E_RXD1, IO_IS = ETH_RXD[1]
  PORT E_RXD2 = CONN_E_RXD2, IO_IS = ETH_RXD[2]
  PORT E_RXD3 = CONN_E_RXD3, IO_IS = ETH_RXD[3]
  PORT E_RX_ER = CONN_E_RX_ER, IO_IS = ETH_RXER

  PORT E_TX_CLK = CONN_E_TX_CLK, IO_IS = ETH_TXC
  PORT E_TX_EN = CONN_E_TX_EN, IO_IS = ETH_TXEN
  PORT E_TXD0 = CONN_E_TXD0, IO_IS = ETH_TXD[0]
  PORT E_TXD1 = CONN_E_TXD1, IO_IS = ETH_TXD[1]
  PORT E_TXD2 = CONN_E_TXD2, IO_IS = ETH_TXD[2]
  PORT E_TXD3 = CONN_E_TXD3, IO_IS = ETH_TXD[3]
  PORT E_TX_ER = CONN_E_TX_ER, IO_IS = ETH_TXER
END

# USB Controller
BEGIN IO_INTERFACE
  ATTRIBUTE IOTYPE = XIL_USB2DEVICE_V4
  ATTRIBUTE INSTANCE = USB_CONTROLLER

  PARAMETER C_INCLUDE_DMA = 1, IO_IS = C_INCLUDE_DMA
  PARAMETER C_PHY_RESET_TYPE = 1, IO_IS = C_PHY_RESET_TYPE

  PORT USB_CLKOUT = CONN_USB_CLKOUT, IO_IS = ULPI_Clock
  PORT USB_DIR = CONN_USB_DIR, IO_IS = ULPI_Dir
  PORT USB_NXT = CONN_USB_NXT, IO_IS = ULPI_Next
  PORT USB_STP = CONN_USB_STP, IO_IS = ULPI_Stop
#  PORT USB_RESET = CONN_USB_RESET, IO_IS = ULPI_Reset
  PORT USB_RESET = net_gnd, IO_IS = ULPI_Reset

  PORT USB_D0 = CONN_USB_D0, IO_IS = ULPI_Data[0]
  PORT USB_D1 = CONN_USB_D1, IO_IS = ULPI_Data[1]
  PORT USB_D2 = CONN_USB_D2, IO_IS = ULPI_Data[2]
  PORT USB_D3 = CONN_USB_D3, IO_IS = ULPI_Data[3]
  PORT USB_D4 = CONN_USB_D4, IO_IS = ULPI_Data[4]
  PORT USB_D5 = CONN_USB_D5, IO_IS = ULPI_Data[5]
  PORT USB_D6 = CONN_USB_D6, IO_IS = ULPI_Data[6]
  PORT USB_D7 = CONN_USB_D7, IO_IS = ULPI_Data[7]
END

# DDR2 Memory
BEGIN IO_INTERFACE
  ATTRIBUTE IOTYPE = XIL_MEMORY_V1
  ATTRIBUTE INSTANCE = DDR2_SDRAM

  # Controller Settings
  PARAMETER C_NUM_PORTS = 1, IO_IS = C_NUM_PORTS

  # Micron MT47H128M16HG-3-IT:A
  PARAMETER C_MEM_TYPE = DDR2, IO_IS = C_MEM_TYPE
  PARAMETER C_MEM_PARTNO="CUSTOM", IO_IS=C_MEM_PARTNO

```

```
# 256MB
PARAMETER C_BASEADDR = 0x00000000 , IO_IS = C_BASEADDR
PARAMETER C_HIGHADDR = 0x0fffffff , IO_IS = C_HIGHADDR

# Memory/DIMM Settings
PARAMETER C_MEM_ADDR_WIDTH = 14, IO_IS = C_MEM_ADDR_WIDTH
PARAMETER C_MEM_BANKADDR_WDITh = 3, IO_IS = C_MEM_BANKADDR_WIDTH
PARAMETER C_MEM_DM_WIDTH = 2, IO_IS = C_MEM_DM_WIDTH
PARAMETER C_MEM_DQS_WIDTH = 2, IO_IS = C_MEM_DQS_WIDTH
PARAMETER C_MEM_DATA_WIDTH = 16, IO_IS = C_MEM_DATA_WIDTH
PARAMETER C_DDR2_DQSN_ENABLE = 1, IO_IS = C_DDR2_DQSN_ENABLE
PARAMETER C_MPMC_CLK0_PERIOD_PS = 3000, IO_IS = C_MPMC_CLK0_PERIOD_PS      # ps

# Memory Part Settings
PARAMETER C_MEM_PART_DATA_DEPTH = 128, IO_IS = C_MEM_PART_DATA_DEPTH
PARAMETER C_MEM_PART_DATA_WIDTH = 16, IO_IS = C_MEM_PART_DATA_WIDTH
PARAMETER C_MEM_PART_NUM_BANK_BITS = 3, IO_IS = C_MEM_PART_NUM_BANK_BITS
PARAMETER C_MEM_PART_NUM_ROW_BITS = 14, IO_IS = C_MEM_PART_NUM_ROW_BITS
PARAMETER C_MEM_PART_NUM_COL_BITS = 10, IO_IS = C_MEM_PART_NUM_COL_BITS

# Memory Timing Settings
PARAMETER C_MEM_PART_CAS_A_FMAX = 667. IO_IS = C_MEM_PART_CAS_A_FMAX  # MHz
PARAMETER C_MEM_PART_CAS_A = 5, IO_IS = C_MEM_PART_CAS_A
PARAMETER C_MEM_PART_TRRD = 10000, IO_IS = C_MEM_PART_TRRD     # ps
PARAMETER C_MEM_PART_TMRD = 2, IO_IS = C_MEM_PART_TMRD     # tCK
PARAMETER C_MEM_PART_TRCD = 15000, IO_IS = C_MEM_PART_TRCD     # ps
PARAMETER C_MEM_PART_TCCD = 2, IO_IS = C_MEM_PART_TCCD     # tCK
PARAMETER C_MEM_PART_TWR = 15000, IO_IS = C_MEM_PART_TWR  # ps
PARAMETER C_MEM_PART_TWTR = 7500, IO_IS = C_MEM_PART_TWTR # ps
PARAMETER C_MEM_PART_TREFI = 3900000, IO_IS = C_MEM_PART_TREFI     # ps
PARAMETER C_MEM_PART_TRFC = 195000, IO_IS = C_MEM_PART_TRFC    # ps
PARAMETER C_MEM_PART_TRP = 15000, IO_IS = C_MEM_PART_TRP  # ps
PARAMETER C_MEM_PART_TRC = 60000, IO_IS = C_MEM_PART_TRC  # ps
PARAMETER C_MEM_PART_TRASMAX = 70000000, IO_IS = C_MEM_PART_TRASMAX    # ps
PARAMETER C_MEM_PART_TRAS = 40000, IO_IS = C_MEM_PART_TRAS     # ps

PORT DDR_ODT = ddr2_ddr_odt_0 , IO_IS = ddr2_odt

PORT DDR_Addr_0 = ddr2_ddr_addr_0 , IO_IS = ddr2_address[0]
PORT DDR_Addr_1 = ddr2_ddr_addr_1 , IO_IS = ddr2_address[1]
PORT DDR_Addr_2 = ddr2_ddr_addr_2 , IO_IS = ddr2_address[2]
PORT DDR_Addr_3 = ddr2_ddr_addr_3 , IO_IS = ddr2_address[3]
PORT DDR_Addr_4 = ddr2_ddr_addr_4 , IO_IS = ddr2_address[4]
PORT DDR_Addr_5 = ddr2_ddr_addr_5 , IO_IS = ddr2_address[5]
PORT DDR_Addr_6 = ddr2_ddr_addr_6 , IO_IS = ddr2_address[6]
PORT DDR_Addr_7 = ddr2_ddr_addr_7 , IO_IS = ddr2_address[7]
PORT DDR_Addr_8 = ddr2_ddr_addr_8 , IO_IS = ddr2_address[8]
PORT DDR_Addr_9 = ddr2_ddr_addr_9 , IO_IS = ddr2_address[9]
PORT DDR_Addr_10 = ddr2_ddr_addr_10 , IO_IS = ddr2_address[10]
PORT DDR_Addr_11 = ddr2_ddr_addr_11 , IO_IS = ddr2_address[11]
PORT DDR_Addr_12 = ddr2_ddr_addr_12 , IO_IS = ddr2_address[12]

PORT DDR_BankAddr_0 = ddr2_ddr_bankaddr_0 , IO_IS = ddr2_BankAddr[0]
PORT DDR_BankAddr_1 = ddr2_ddr_bankaddr_1 , IO_IS = ddr2_BankAddr[1]

PORT DDR_CASn = ddr2_ddr_casn , IO_IS = ddr2_col_addr_select
PORT DDR_RASn = ddr2_ddr_rasn , IO_IS = ddr2_row_addr_select
```

```
  PORT DDR_CKE = ddr2_ddr_cke, IO_IS = ddr2_clk_enable
  PORT DDR_CSn = ddr2_ddr_csn, IO_IS = ddr2_chip_select
  PORT DDR_WEn = ddr2_ddr_wen, IO_IS = ddr2_write_enable

  PORT DDR_CLK = ddr2_clk, IO_IS = ddr2_clk
  PORT DDR_CLK_n = ddr2_clk_n, IO_IS = ddr2_clk_n

  PORT DDR_DM_0 = ddr2_ddr_dm_0, IO_IS = ddr2_data_mask[0]
  PORT DDR_DM_1 = ddr2_ddr_dm_1, IO_IS = ddr2_data_mask[1]
  PORT DDR_DQS_0 = ddr2_ddr_dqs_0, IO_IS = ddr2_data_strobe[0]
  PORT DDR_DQS_1 = ddr2_ddr_dqs_1, IO_IS = ddr2_data_strobe[1]
  PORT DDR_DQSn_0 = ddr2_ddr_dqsn_0, IO_IS = ddr2_data_strobe_n[0]
  PORT DDR_DQSn_1 = ddr2_ddr_dqsn_1, IO_IS = ddr2_data_strobe_n[1]

  PORT DDR_DQ_0 =  ddr2_ddr_dq_0 ,   IO_IS = ddr2_data[0]
  PORT DDR_DQ_1 =  ddr2_ddr_dq_1 ,   IO_IS = ddr2_data[1]
  PORT DDR_DQ_2 =  ddr2_ddr_dq_2 ,   IO_IS = ddr2_data[2]
  PORT DDR_DQ_3 =  ddr2_ddr_dq_3 ,   IO_IS = ddr2_data[3]
  PORT DDR_DQ_4 =  ddr2_ddr_dq_4 ,   IO_IS = ddr2_data[4]
  PORT DDR_DQ_5 =  ddr2_ddr_dq_5 ,   IO_IS = ddr2_data[5]
  PORT DDR_DQ_6 =  ddr2_ddr_dq_6 ,   IO_IS = ddr2_data[6]
  PORT DDR_DQ_7 =  ddr2_ddr_dq_7 ,   IO_IS = ddr2_data[7]
  PORT DDR_DQ_8 =  ddr2_ddr_dq_8 ,   IO_IS = ddr2_data[8]
  PORT DDR_DQ_9 =  ddr2_ddr_dq_9 ,   IO_IS = ddr2_data[9]
  PORT DDR_DQ_10 = ddr2_ddr_dq_10,   IO_IS = ddr2_data[10]
  PORT DDR_DQ_11 = ddr2_ddr_dq_11,   IO_IS = ddr2_data[11]
  PORT DDR_DQ_12 = ddr2_ddr_dq_12,   IO_IS = ddr2_data[12]
  PORT DDR_DQ_13 = ddr2_ddr_dq_13,   IO_IS = ddr2_data[13]
  PORT DDR_DQ_14 = ddr2_ddr_dq_14,   IO_IS = ddr2_data[14]
  PORT DDR_DQ_15 = ddr2_ddr_dq_15,   IO_IS = ddr2_data[15]

  PORT DDR_DQS_DIV_I = ddr2_ddr_dqs_div_i, IO_IS = ddr2_dqs_div_i
  PORT DDR_DQS_DIV_O = ddr2_ddr_dqs_div_o, IO_IS = ddr2_dqs_div_o
END

# SPI Controller
BEGIN IO_INTERFACE
  ATTRIBUTE IOTYPE = XIL_SPI_V1
  ATTRIBUTE INSTANCE = SPI_CONTROLLER

  PARAMETER C_FIFO_EXIST = 1, IO_IS=fifo_exist
  PARAMETER C_SCK_RATIO = 32, IO_IS=clk_freq
  PARAMETER C_NUM_TRANSFER_BITS = 8, IO_IS = num_transfer_bits
#  PARAMETER C_NUM_SS_BITS = 4, IO_IS = ss_bits
  PARAMETER C_NUM_SS_BITS = 2, IO_IS = ss_bits

  PORT SPISEL = net_vcc, IO_IS = slave_select_n
  PORT SEN_SPI_RX = CONN_SEN_SPI_RX, IO_IS = data_in
  PORT SEN_SPI_TX = CONN_SEN_SPI_TX, IO_IS = data_out
  PORT SEN_SPI_CLK = CONN_SEN_SPI_CLK, IO_IS = clk_out

  PORT SEN_SS_0 = CONN_SEN_SS_0, IO_IS = slave_select[0]
  PORT SEN_SS_1 = CONN_SEN_SS_1, IO_IS = slave_select[1]          # SEN_OE_IO
#  PORT SEN_SS_2 = CONN_SEN_SS_2, IO_IS = slave_select[2]
#  PORT SEN_SS_3 = CONN_SEN_SS_3, IO_IS = slave_select[3]
END

```

```
# FPGA Definition
BEGIN FPGA
  ATTRIBUTE INSTANCE       = fpga_0
  ATTRIBUTE FAMILY         = spartan3a
  ATTRIBUTE DEVICE         = xc3s1400a
  ATTRIBUTE PACKAGE        = fg484
  ATTRIBUTE SPEED_GRADE    = -5
  ATTRIBUTE JTAG_POSITION = 2

  ### CLOCKS ###
  PORT CLOCK          = CONN_OSC_50_MHZ    , UCF_NET_STRING = ( "LOC = V12", "IOSTANDARD = LCVMOS33"
      )
  PORT CLOCK          = CONN_OSC_75_MHZ    , UCF_NET_STRING = ( "LOC = AA12", "IOSTANDARD =
      LCVMOS33", "CLOCK_DEDICATED_ROUTE = FALSE" )
  PORT CLOCK          = CONN_OSC_100_MHZ   , UCF_NET_STRING = ( "LOC = C12", "IOSTANDARD =
      LCVMOS33", "CLOCK_DEDICATED_ROUTE = FALSE" )

  ### RESET ### Used the south button
  PORT RESET          = CONN_RESET         , UCF_NET_STRING = ( "LOC = U7", "IOSTANDARD = LVCMOS33",
      "PULLDOWN" )

  ### UARTS ###
  PORT FROM_RS232     = CONN_FROM_RS232    , UCF_NET_STRING = ( "LOC = R15", "IOSTANDARD = LVCMOS33"
      )
  PORT TO_RS232       = CONN_TO_RS232      , UCF_NET_STRING = ( "LOC = Y22", "IOSTANDARD = LVCMOS33"
      )

  PORT AT90_RXD1      = CONN_AT90_RXD1     , UCF_NET_STRING = ( "LOC = D18", "IOSTANDARD = LVCMOS33"
      )
  PORT AT90_TXD1      = CONN_AT90_TXD1     , UCF_NET_STRING = ( "LOC = D19", "IOSTANDARD = LVCMOS33"
      )

  ### LEDS ###
  PORT LED0     = CONN_LED0     , UCF_NET_STRING = ( "LOC = AA19", "IOSTANDARD = LVCMOS33" )
  PORT LED1     = CONN_LED1     , UCF_NET_STRING = ( "LOC = AB19", "IOSTANDARD = LVCMOS33" )
  PORT LED2     = CONN_LED2     , UCF_NET_STRING = ( "LOC = V17", "IOSTANDARD = LVCMOS33" )
  PORT LED3     = CONN_LED3     , UCF_NET_STRING = ( "LOC = W18", "IOSTANDARD = LVCMOS33" )
  PORT LED4     = CONN_LED4     , UCF_NET_STRING = ( "LOC = W17", "IOSTANDARD = LVCMOS33" )
  PORT LED5     = CONN_LED5     , UCF_NET_STRING = ( "LOC = Y18", "IOSTANDARD = LVCMOS33" )
  PORT LED6     = CONN_LED6     , UCF_NET_STRING = ( "LOC = AA21", "IOSTANDARD = LVCMOS33" )
  PORT LED7     = CONN_LED7     , UCF_NET_STRING = ( "LOC = AB21", "IOSTANDARD = LVCMOS33" )

  ### PUSH BUTTON ###
#  PORT BUTTON        = CONN_BUTTON        , UCF_NET_STRING = ( "LOC = U7", "IOSTANDARD = LVCMOS33",
    "PULLDOWN" )

  ### AT90 I/O Ports ###
  PORT AT_PA0         = CONN_AT_PA0        , UCF_NET_STRING = ( "LOC = A6", "IOSTANDARD = LVCMOS33" )
  PORT AT_PA1         = CONN_AT_PA1        , UCF_NET_STRING = ( "LOC = A7", "IOSTANDARD = LVCMOS33" )
  PORT AT_PA2         = CONN_AT_PA2        , UCF_NET_STRING = ( "LOC = C7", "IOSTANDARD = LVCMOS33" )
  PORT AT_PA3         = CONN_AT_PA3        , UCF_NET_STRING = ( "LOC = D7", "IOSTANDARD = LVCMOS33" )
  PORT AT_PA4         = CONN_AT_PA4        , UCF_NET_STRING = ( "LOC = A5", "IOSTANDARD = LVCMOS33" )
  PORT AT_PA5         = CONN_AT_PA5        , UCF_NET_STRING = ( "LOC = B6", "IOSTANDARD = LVCMOS33" )
  PORT AT_PA6         = CONN_AT_PA6        , UCF_NET_STRING = ( "LOC = D6", "IOSTANDARD = LVCMOS33" )
  PORT AT_PA7         = CONN_AT_PA7        , UCF_NET_STRING = ( "LOC = C6", "IOSTANDARD = LVCMOS33" )

  PORT AT_PC0        = CONN_AT_PC0        , UCF_NET_STRING = ( "LOC = C10" , "IOSTANDARD = LVCMOS33"
      )
  PORT AT_PC1        = CONN_AT_PC1        , UCF_NET_STRING = ( "LOC = A10" , "IOSTANDARD = LVCMOS33"
      )
  PORT AT_PC2        = CONN_AT_PC2        , UCF_NET_STRING = ( "LOC = A8" , "IOSTANDARD = LVCMOS33" )
  PORT AT_PC3        = CONN_AT_PC3        , UCF_NET_STRING = ( "LOC = A9" , "IOSTANDARD = LVCMOS33" )
  PORT AT_PC4        = CONN_AT_PC4        , UCF_NET_STRING = ( "LOC = E10" , "IOSTANDARD = LVCMOS33"
      )
  PORT AT_PC5        = CONN_AT_PC5        , UCF_NET_STRING = ( "LOC = D10" , "IOSTANDARD = LVCMOS33"
      )
  PORT AT_PC6        = CONN_AT_PC6        , UCF_NET_STRING = ( "LOC = C9" , "IOSTANDARD = LVCMOS33" )
  PORT AT_PC7        = CONN_AT_PC7        , UCF_NET_STRING = ( "LOC = B9" , "IOSTANDARD = LVCMOS33" )

  PORT AT_PF0        = CONN_AT_PF0        , UCF_NET_STRING = ( "LOC = B17" , "IOSTANDARD = LVCMOS33"
      )
  PORT AT_PF1        = CONN_AT_PF1        , UCF_NET_STRING = ( "LOC = A17" , "IOSTANDARD = LVCMOS33"
      )
  PORT AT_PF2        = CONN_AT_PF2        , UCF_NET_STRING = ( "LOC = D15" , "IOSTANDARD = LVCMOS33"
      )
  PORT AT_PF3        = CONN_AT_PF3        , UCF_NET_STRING = ( "LOC = C15" , "IOSTANDARD = LVCMOS33"
      )

  ### Ethernet ###
  PORT E_COL         = CONN_E_COL         , UCF_NET_STRING = ( "LOC = AB3" , "IOSTANDARD =
      LVCMOS33" , "PULLDOWN" )
  PORT E_CRS         = CONN_E_CRS         , UCF_NET_STRING = ( "LOC = AA4" , "IOSTANDARD =
      LVCMOS33" , "PULLDOWN" )
  PORT E_MDC         = CONN_E_MDC         , UCF_NET_STRING = ( "LOC = AB4" , "IOSTANDARD = LVCMOS33"
      )
  PORT E_MDIO        = CONN_E_MDIO        , UCF_NET_STRING = ( "LOC = AB5" , "IOSTANDARD = LVCMOS33"
      )
  PORT E_NRST        = CONN_E_NRST        , UCF_NET_STRING = ( "LOC = Y5" , "IOSTANDARD = LVCMOS33" )
  PORT E_RX_CLK      = CONN_E_RX_CLK      , UCF_NET_STRING = ( "LOC = W7" , "IOSTANDARD = LVCMOS33" )
  PORT E_RX_DV       = CONN_E_RX_DV       , UCF_NET_STRING = ( "LOC = Y7" , "IOSTANDARD = LVCMOS33" )
  PORT E_RXD0        = CONN_E_RXD0        , UCF_NET_STRING = ( "LOC = AB7" , "IOSTANDARD =
      LVCMOS33" , "PULLUP" )
  PORT E_RXD1        = CONN_E_RXD1        , UCF_NET_STRING = ( "LOC = W6" , "IOSTANDARD = LVCMOS33" ,
      "PULLUP" )
  PORT E_RXD2        = CONN_E_RXD2        , UCF_NET_STRING = ( "LOC = Y6" , "IOSTANDARD = LVCMOS33" ,
      "PULLUP" )
  PORT E_RXD3        = CONN_E_RXD3        , UCF_NET_STRING = ( "LOC = AA6" , "IOSTANDARD =
      LVCMOS33" , "PULLUP" )
  PORT E_RX_ER       = CONN_E_RX_ER       , UCF_NET_STRING = ( "LOC = AB8" , "IOSTANDARD = LVCMOS33"
      )
  PORT E_TC_CLK      = CONN_E_TX_CLK      , UCF_NET_STRING = ( "LOC = AA8" , "IOSTANDARD = LVCMOS33"
      )
  PORT E_TX_EN       = CONN_E_TX_EN       , UCF_NET_STRING = ( "LOC = AB6" , "IOSTANDARD = LVCMOS33"
      )
  PORT E_TXD0        = CONN_E_TXD0        , UCF_NET_STRING = ( "LOC = AB10" , "IOSTANDARD =
      LVCMOS33" )
  PORT E_TXD1        = CONN_E_TXD1        , UCF_NET_STRING = ( "LOC = AA10" , "IOSTANDARD =
      LVCMOS33" )
  PORT E_TXD2        = CONN_E_TXD2        , UCF_NET_STRING = ( "LOC = Y10" , "IOSTANDARD = LVCMOS33"
      )
  PORT E_TXD3        = CONN_E_TXD3        , UCF_NET_STRING = ( "LOC = V10" , "IOSTANDARD = LVCMOS33"
      )
```

```
  PORT E_TX_ER        = CONN_E_TX_ER        , UCF_NET_STRING = ( "LOC = AA3", "IOSTANDARD = LVCMOS33", "PULLUP" )

  ### USB ###
  PORT USB_CLKOUT     = CONN_USB_CLKOUT     , UCF_NET_STRING = ( "LOC = E13", "IOSTANDARD = LVCMOS33", "PULLDOWN" )
  PORT USB_DIR        = CONN_USB_DIR        , UCF_NET_STRING = ( "LOC = C13", "IOSTANDARD = LVCMOS33", "PULLDOWN" )
  PORT USB_NXT        = CONN_USB_NXT        , UCF_NET_STRING = ( "LOC = D13", "IOSTANDARD = LVCMOS33", "PULLDOWN" )
  PORT USB_STP        = CONN_USB_STP        , UCF_NET_STRING = ( "LOC = F13", "IOSTANDARD = LVCMOS33", "PULLDOWN" )
# PORT USB_RESET      = CONN_USB_RESET      , UCF_NET_STRING = ( "LOC = ", "IOSTANDARD = LVCMOS33", "PULLDOWN", "TIG" )
  PORT USB_D0         = CONN_USB_D0         , UCF_NET_STRING = ( "LOC = B20", "IOSTANDARD = LVCMOS33", "PULLDOWN" )
  PORT USB_D1         = CONN_USB_D1         , UCF_NET_STRING = ( "LOC = A10", "IOSTANDARD = LVCMOS33", "PULLDOWN" )
  PORT USB_D2         = CONN_USB_D2         , UCF_NET_STRING = ( "LOC = E15", "IOSTANDARD = LVCMOS33", "PULLDOWN" )
  PORT USB_D3         = CONN_USB_D3         , UCF_NET_STRING = ( "LOC = F15", "IOSTANDARD = LVCMOS33", "PULLDOWN" )
  PORT USB_D4         = CONN_USB_D4         , UCF_NET_STRING = ( "LOC = C18", "IOSTANDARD = LVCMOS33", "PULLDOWN" )
  PORT USB_D5         = CONN_USB_D5         , UCF_NET_STRING = ( "LOC = A18", "IOSTANDARD = LVCMOS33", "PULLDOWN" )
  PORT USB_D6         = CONN_USB_D6         , UCF_NET_STRING = ( "LOC = B19", "IOSTANDARD = LVCMOS33", "PULLDOWN" )
  PORT USB_D7         = CONN_USB_D7         , UCF_NET_STRING = ( "LOC = A19", "IOSTANDARD = LVCMOS33", "PULLDOWN" )

  ### SPI ###
  PORT SEN_SPI_RX     = CONN_SEN_SPI_RX     , UCF_NET_STRING = ( "LOC = M18", "IOSTANDARD = LVTTL" )
  PORT SEN_SPI_TX     = CONN_SEN_SPI_TX     , UCF_NET_STRING = ( "LOC = L22", "IOSTANDARD = LVTTL" )
  PORT SEN_SPI_CLK    = CONN_SEN_SPI_CLK    , UCF_NET_STRING = ( "LOC = K17", "IOSTANDARD = LVTTL" )
  PORT SEN_SS_0       = CONN_SEN_SS_0       , UCF_NET_STRING = ( "LOC = T19", "IOSTANDARD = LVTTL" ) # FIXME
  PORT SEN_SS_1       = CONN_SEN_SS_1       , UCF_NET_STRING = ( "LOC = M22", "IOSTANDARD = LVTTL" )

  ### DDR2 (FPGA Section)
  PORT DDR2_ODT            = ddr2_ddr_odt_0      , UCF_NET_STRING=("LOC=P1", "IOSTANDARD = SSTL18_I")
  PORT DDR2_DDR_ADDR0      = ddr2_ddr_addr_0     , UCF_NET_STRING=("LOC=R2", "IOSTANDARD = SSTL18_I")
  PORT DDR2_DDR_ADDR1      = ddr2_ddr_addr_1     , UCF_NET_STRING=("LOC=T4", "IOSTANDARD = SSTL18_I")
  PORT DDR2_DDR_ADDR2      = ddr2_ddr_addr_2     , UCF_NET_STRING=("LOC=R1", "IOSTANDARD = SSTL18_I")
  PORT DDR2_DDR_ADDR3      = ddr2_ddr_addr_3     , UCF_NET_STRING=("LOC=U3", "IOSTANDARD = SSTL18_I")
  PORT DDR2_DDR_ADDR4      = ddr2_ddr_addr_4     , UCF_NET_STRING=("LOC=U2", "IOSTANDARD = SSTL18_I")
  PORT DDR2_DDR_ADDR5      = ddr2_ddr_addr_5     , UCF_NET_STRING=("LOC=U4", "IOSTANDARD = SSTL18_I")
  PORT DDR2_DDR_ADDR6      = ddr2_ddr_addr_6     , UCF_NET_STRING=("LOC=U1", "IOSTANDARD = SSTL18_I")
  PORT DDR2_DDR_ADDR7      = ddr2_ddr_addr_7     , UCF_NET_STRING=("LOC=Y1", "IOSTANDARD = SSTL18_I")
  PORT DDR2_DDR_ADDR8      = ddr2_ddr_addr_8     , UCF_NET_STRING=("LOC=W1", "IOSTANDARD = SSTL18_I")
  PORT DDR2_DDR_ADDR9      = ddr2_ddr_addr_9     , UCF_NET_STRING=("LOC=W2", "IOSTANDARD = SSTL18_I")
  PORT DDR2_DDR_ADDR10     = ddr2_ddr_addr_10    , UCF_NET_STRING=("LOC=T3", "IOSTANDARD = SSTL18_I")
  PORT DDR2_DDR_ADDR11     = ddr2_ddr_addr_11    , UCF_NET_STRING=("LOC=V1", "IOSTANDARD = SSTL18_I")
  PORT DDR2_DDR_ADDR12     = ddr2_ddr_addr_12    , UCF_NET_STRING=("LOC=Y2", "IOSTANDARD = SSTL18_I")

  PORT DDR2_DDR_BANKADDR0 = ddr2_ddr_bankaddr_0, UCF_NET_STRING=("LOC=P3", "IOSTANDARD = SSTL18_I")
  PORT DDR2_DDR_BANKADDR1 = ddr2_ddr_bankaddr_1 , UCF_NET_STRING=("LOC=R3" , "IOSTANDARD = SSTL18_I")

  PORT DDR2_DDR_CASN     = ddr2_ddr_casn      , UCF_NET_STRING=("LOC=M4" , "IOSTANDARD = SSTL18_I")
  PORT DDR2_DDR_CKE      = ddr2_ddr_cke       , UCF_NET_STRING=("LOC=N3" , "IOSTANDARD = SSTL18_I")
  PORT DDR2_DDR_CSN      = ddr2_ddr_csn       , UCF_NET_STRING=("LOC=M5" , "IOSTANDARD = SSTL18_I")
  PORT DDR2_DDR_RASN     = ddr2_ddr_rasn      , UCF_NET_STRING=("LOC=M3" , "IOSTANDARD = SSTL18_I")
  PORT DDR2_DDR_WEN      = ddr2_ddr_wen       , UCF_NET_STRING=("LOC=N4" , "IOSTANDARD = SSTL18_I")
  PORT DDR2_DDR_CLK      = ddr2_clk           , UCF_NET_STRING=("LOC=M1" , "IOSTANDARD = DIFF_SSTL18_I")
  PORT DDR2_DDR_CLKn     = ddr2_clk_n         , UCF_NET_STRING=("LOC=M2" , "IOSTANDARD = DIFF_SSTL18_I")

  PORT DDR2_DDR_DM0      = ddr2_ddr_dm_0      , UCF_NET_STRING=("LOC=J3" , "IOSTANDARD = SSTL18_I")
  PORT DDR2_DDR_DM1      = ddr2_ddr_dm_1      , UCF_NET_STRING=("LOC=E3" , "IOSTANDARD = SSTL18_I")

  PORT DDR2_DQS_DIV_I    = ddr2_ddr_dqs_div_i     , UCF_NET_STRING=("LOC=H4" , "IOSTANDARD = SSTL18_II")
  PORT DDR2_DQS_DIV_O    = ddr2_ddr_dqs_div_o     , UCF_NET_STRING=("LOC=H3" , "IOSTANDARD = SSTL18_II")

  PORT DDR2_DDR_DQS0     = ddr2_ddr_dqs_0     , UCF_NET_STRING=("LOC=K3" , "IOSTANDARD = DIFF_SSTL18_II")
  PORT DDR2_DDR_DQS1     = ddr2_ddr_dqs_1     , UCF_NET_STRING=("LOC=K6" , "IOSTANDARD = DIFF_SSTL18_II")

  PORT DDR2_DDR_DQSn0    = ddr2_ddr_dqsn_0    , UCF_NET_STRING=("LOC=K2" , "IOSTANDARD = DIFF_SSTL18_II")
  PORT DDR2_DDR_DQSn1    = ddr2_ddr_dqsn_1    , UCF_NET_STRING=("LOC=J5" , "IOSTANDARD = DIFF_SSTL18_II")

  PORT DDR2_DDR_DQ0      =  ddr2_ddr_dq_0     , UCF_NET_STRING=("LOC=H1" , "IOSTANDARD = SSTL18_II")
  PORT DDR2_DDR_DQ1      =  ddr2_ddr_dq_1     , UCF_NET_STRING=("LOC=K5" , "IOSTANDARD = SSTL18_II")
  PORT DDR2_DDR_DQ2      =  ddr2_ddr_dq_2     , UCF_NET_STRING=("LOC=K1" , "IOSTANDARD = SSTL18_II")
  PORT DDR2_DDR_DQ3      =  ddr2_ddr_dq_3     , UCF_NET_STRING=("LOC=L3" , "IOSTANDARD = SSTL18_II")
  PORT DDR2_DDR_DQ4      =  ddr2_ddr_dq_4     , UCF_NET_STRING=("LOC=L5" , "IOSTANDARD = SSTL18_II")
  PORT DDR2_DDR_DQ5      =  ddr2_ddr_dq_5     , UCF_NET_STRING=("LOC=L1" , "IOSTANDARD = SSTL18_II")
  PORT DDR2_DDR_DQ6      =  ddr2_ddr_dq_6     , UCF_NET_STRING=("LOC=K4" , "IOSTANDARD = SSTL18_II")
  PORT DDR2_DDR_DQ7      =  ddr2_ddr_dq_7     , UCF_NET_STRING=("LOC=H2" , "IOSTANDARD = SSTL18_II")
  PORT DDR2_DDR_DQ8      =  ddr2_ddr_dq_8     , UCF_NET_STRING=("LOC=F2" , "IOSTANDARD = SSTL18_II")
  PORT DDR2_DDR_DQ9      =  ddr2_ddr_dq_9     , UCF_NET_STRING=("LOC=G4" , "IOSTANDARD = SSTL18_II")
  PORT DDR2_DDR_DQ10     =  ddr2_ddr_dq_10    , UCF_NET_STRING=("LOC=G1" , "IOSTANDARD = SSTL18_II")
  PORT DDR2_DDR_DQ11     =  ddr2_ddr_dq_11    , UCF_NET_STRING=("LOC=H6" , "IOSTANDARD = SSTL18_II")
  PORT DDR2_DDR_DQ12     =  ddr2_ddr_dq_12    , UCF_NET_STRING=("LOC=H5" , "IOSTANDARD = SSTL18_II")
  PORT DDR2_DDR_DQ13     =  ddr2_ddr_dq_13    , UCF_NET_STRING=("LOC=F1" , "IOSTANDARD = SSTL18_II")
  PORT DDR2_DDR_DQ14     =  ddr2_ddr_dq_14    , UCF_NET_STRING=("LOC=G3" , "IOSTANDARD = SSTL18_II")
  PORT DDR2_DDR_DQ15     =  ddr2_ddr_dq_15    , UCF_NET_STRING=("LOC=F3" , "IOSTANDARD = SSTL18_II")
END
```

Appendix C

USRP Test Benches

This appendix contains the test code for the CubeSat SDR system. Each test bench is called TB_<module>.vhd, where <module> is the name of the entity under test. The code listed in this appendix is licensed under the GPL2.

Listing C.1: TB_TB_acc.vhd

```
--------------------------------------------------------------------------------
-- Test Bench for acc
-- Last Modified: 22 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2011 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

entity TB_acc is
end TB_acc;
```

```vhdl

architecture behavioral of TB_acc is
    -- Component declaration for the Unit Under Test (UUT)
    component acc is
        port(
            clock         : in    std_logic;
            reset         : in    std_logic;
            clear         : in    std_logic;
            enable_in    : in    std_logic;
            enable_out   : out   std_logic;
            addend       : in    std_logic_vector(30 downto 0);
            sum          : out   std_logic_vector(33 downto 0)
        );
    end component acc;

    -- Inputs
    signal clock : std_logic := '0';
    signal reset : std_logic := '0';
    signal clear : std_logic := '0';
    signal enable_in : std_logic := '0';
    signal addend : std_logic_vector(30 downto 0) := (others => '0');

    -- Outputs
    signal enable_out : std_logic;
    signal sum : std_logic_vector(33 downto 0);

    -- Clock Period
    constant clock_period : time := 10ns;
begin
    -- Instantiate the UUT.
    UUT : acc
        port map(clock => clock, reset => reset, clear => clear, enable_in => enable_in,
            enable_out => enable_out, addend => addend, sum => sum);

    -- Clock Process
    clock_process : process
    begin
        clock <= '0';
        wait for clock_period / 2;
        clock <= '1';
        wait for clock_period / 2;
    end process clock_process;

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for one clock period.
        wait for clock_period;

        -- Assert the reset.
        reset <= '1';
        wait for clock_period;

        -- Clear the accumulator.
        reset <= '0';
        clear <= '1';
        addend <= (others => '0');
        wait for clock_period;

        -- Add some values.
        clear <= '0';
        enable_in <= '1';
        for i in 1 to 15 loop
            addend <= conv_std_logic_vector(i, addend'LENGTH);
            wait for clock_period;
        end loop;

        -- Clear the accumulator.
        enable_in <= '0';
        clear <= '1';
        addend <= (others => '0');
        wait for clock_period;

        -- Add some negative values.
        clear <= '0';
        enable_in <= '1';
        for i in -1 downto -15 loop
            addend <= conv_std_logic_vector(i, addend'LENGTH);
            wait for clock_period;
        end loop;

        -- Clear the accumulator.
        enable_in <= '0';
        clear <= '1';
        addend <= (others => '0');
        wait for clock_period;

        -- Test overflow.
        clear <= '0';
        enable_in <= '1';
        for i in 0 to 8 loop
            addend <= "01111111111111111111111111111111";
            wait for clock_period;
        end loop;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.2: TB_adc_mux.vhd

```
--------------------------------------------------------------------------------
-- Test Bench for adc_mux
-- Last Modified: 14 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2011 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
```

```
--
-- This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.
--
-- You should have received a copy of the GNU General Public License
-- along with this program; if not, write to the Free Software
-- Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TB_adc_mux is
end TB_adc_mux;

architecture behavioral of TB_adc_mux is
    -- Component declaration for the Unit Under Test (UUT)
    component adc_mux is
        port(
            clock              : in    std_logic;
            rx_realsignals  : in    std_logic;
            ddc_mux          : in    std_logic_vector(3 downto 0);
            adc3_corr        : in    std_logic_vector(15 downto 0);
            adc2_corr        : in    std_logic_vector(15 downto 0);
            adc1_corr        : in    std_logic_vector(15 downto 0);
            adc0_corr        : in    std_logic_vector(15 downto 0);
            ddc_i              : out   std_logic_vector(15 downto 0);
            ddc_q              : out   std_logic_vector(15 downto 0)
        );
    end component adc_mux;

    -- Inputs
    signal clock : std_logic := '0';
    signal rx_realsignals : std_logic := '0';
    signal ddc_mux : std_logic_vector(3 downto 0) := (others => '0');
    signal adc3_corr : std_logic_vector(15 downto 0) := (others => '0');
    signal adc2_corr : std_logic_vector(15 downto 0) := (others => '0');
    signal adc1_corr : std_logic_vector(15 downto 0) := (others => '0');
    signal adc0_corr : std_logic_vector(15 downto 0) := (others => '0');

    -- Outputs
    signal ddc_i : std_logic_vector(15 downto 0);
    signal ddc_q : std_logic_vector(15 downto 0);

    -- Clock Period
    constant clock_period : time := 10ns;
begin
    -- Instantiate the UUT.
    UUT : adc_mux
        port map(clock => clock, rx_realsignals => rx_realsignals, ddc_mux => ddc_mux,
            adc3_corr => adc3_corr, adc2_corr => adc2_corr, adc1_corr => adc1_corr,
            adc0_corr => adc0_corr, ddc_i => ddc_i, ddc_q => ddc_q);

    -- Clock Process
```

```vhdl
    clock_process : process
    begin
        clock <= '0';
        wait for clock_period / 2;
        clock <= '1';
        wait for clock_period / 2;
    end process clock_process;

    -- Stimulus Process
    stim_proc : process
    begin
        -- Assign some values to the registers.
        adc0_corr <= "0001000100010001";
        adc1_corr <= "0010001000100010";
        adc2_corr <= "0011001100110011";
        adc3_corr <= "0100010001000100";

        -- Try every possible input for ddc_mux with rx_realsignals low.
        rx_realsignals <= '0';
        for i in 0 to 15 loop
            ddc_mux <= conv_std_logic_vector(i, ddc_mux'LENGTH);
            wait for clock_period;
        end loop;

        -- Try every possible input for ddc_mux with rx_realsignals high.
        rx_realsignals <= '1';
        for i in 0 to 15 loop
            ddc_mux <= conv_std_logic_vector(i, ddc_mux'LENGTH);
            wait for clock_period;
        end loop;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.3: TB_atr_delay.vhd

```vhdl
------------------------------------------------------------------------------
-- Test Bench for atr_delay
-- Last Modified: 14 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TB_atr_delay is
end TB_atr_delay;

architecture behavioral of TB_atr_delay is
    -- Component declaration for the Unit Under Test (UUT)
    component atr_delay is
        port(
            clk_i          : in    std_logic;
            rst_i          : in    std_logic;
            ena_i          : in    std_logic;
            tx_empty_i  : in    std_logic;
            tx_delay_i  : in    std_logic_vector(11 downto 0);
            rx_delay_i  : in    std_logic_vector(11 downto 0);
            atr_tx_o        : out   std_logic
        );
    end component atr_delay;

    -- Inputs
    signal clk_i : std_logic := '0';
    signal rst_i : std_logic := '0';
    signal ena_i : std_logic := '0';
    signal tx_empty_i : std_logic := '0';
    signal tx_delay_i : std_logic_vector(11 downto 0) := (others => '0');
    signal rx_delay_i : std_logic_vector(11 downto 0) := (others => '0');

    -- Outputs
    signal atr_tx_o : std_logic;

    -- Clock Period
   constant clk_i_period : time := 10ns;
begin
    -- Instantiate the UUT.
    UUT : atr_delay
        port map(clk_i => clk_i, rst_i => rst_i, ena_i => ena_i, tx_empty_i => tx_empty_i,
            tx_delay_i => tx_delay_i, rx_delay_i => rx_delay_i, atr_tx_o => atr_tx_o);

    -- Clock Process
    clk_i_process : process
    begin
        clk_i <= '0';
        wait for clk_i_period / 2;
        clk_i <= '1';
        wait for clk_i_period / 2;
    end process clk_i_process;

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for one clock period.
        wait for clk_i_period;

        -- Assert the reset.
        rst_i <= '1';
        wait for clk_i_period;

        -- Deassert the reset, assert the enable.
        rst_i <= '0';
        ena_i <= '1';

        -- Delay values of 2 and 3.
        tx_delay_i <= "000000000010";
        rx_delay_i <= "000000000011";

        -- Because tx_empty_i is low, we should switch to TX_DELAY and then TX.
        -- We'll stay in TX until we assert tx_empty_i.
        wait for 10 * clk_i_period;

        -- Now, assert tx_empty_i.  The atr_tx_o signal should remain high for
        -- rx_delay while we're in the RX_DELAY state and then it should go low
        -- when we reach RX.
        tx_empty_i <= '1';
        wait for 10 * clk_i_period;

        -- Let's get back to the TX state so we can test the reset again.  We'll
        -- change the delay values to 4 and 7, as well.
        tx_empty_i <= '0';
        tx_delay_i <= "000000000100";
        rx_delay_i <= "000000000111";
        wait for 10 * clk_i_period;

        -- Assert the reset.  This should put us back to RX.
        rst_i <= '1';
        wait for 10 * clk_i_period;

        -- Delay until TX again.
        rst_i <= '0';
        wait for 10 * clk_i_period;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.4: TB_bidir_reg.vhd

```
--------------------------------------------------------------------------------
-- Test Bench for bidir_reg
-- Last Modified: 14 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
```

```
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TB_bidir_reg is
end TB_bidir_reg;

architecture behavioral of TB_bidir_reg is
    -- Component declaration for the Unit Under Test (UUT)
    component bidir_reg is
        port(
            tristate    : inout std_logic_vector(15 downto 0);
            oe          : in        std_logic_vector(15 downto 0);
            reg_val : in        std_logic_vector(15 downto 0)
        );
    end component bidir_reg;

    -- Inputs
    signal oe : std_logic_vector(15 downto 0) := (others => '0');
    signal reg_val : std_logic_vector(15 downto 0) := (others => '0');

    -- Bidirectionals
    signal tristate : std_logic_vector(15 downto 0);
begin
    -- Instantiate the UUT.
    UUT : bidir_reg
        port map(tristate => tristate, oe => oe, reg_val => reg_val);

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for 10ns.
        wait for 10ns;

        -- Keep the enables off, but change the input values.
        oe <= (others => '0');
        reg_val <= "1100001111000011";
        wait for 10ns;

        -- Turn on all enables.
        oe <= (others => '1');
        wait for 10ns;

        -- Change some of the inputs while leaving all enables on.
        reg_val <= "0000000011111111";
        wait for 10ns;

        -- Turn off some enables, leave others on.
        oe <= "1010101001010101";
        wait for 10ns;

        -- Change some of the inputs again.
        reg_val <= "1111000011110000";
        wait for 10ns;

        -- Disable all outputs.
        oe <= (others => '0');
        wait for 10ns;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.5: TB_bustri.vhd

```
---------------------------------------------------------------------------
-- Test Bench for bustri
-- Last Modified: 13 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
---------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TB_bustri is
end TB_bustri;

architecture behavioral of TB_bustri is
    -- Component declaration for the Unit Under Test (UUT)
    component bustri
        port(
            data      : in    std_logic_vector(15 downto 0);
            enabledt  : in    std_logic;
            tridata : out   std_logic_vector(15 downto 0)
        );
    end component bustri;

    -- Inputs
    signal data : std_logic_vector(15 downto 0) := (others => '0');
    signal enabledt : std_logic := '0';

    -- Outputs
    signal tridata : std_logic_vector(15 downto 0);
begin
    -- Instantiate the UUT.
    UUT : bustri
        port map(data => data, enabledt => enabledt, tridata => tridata);

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for 10ns.
        wait for 10ns;

        -- Set data, turn off output.
        data <= "1010010111110000";
        enabledt <= '0';
        wait for 10ns;

        -- Turn on output.
        enabledt <= '1';
        wait for 10ns;

        -- Change data with output still enabled.
        data <= "0000111101011010";
        wait for 10ns;

        -- Disable output.
        enabledt <= '0';
        wait for 10ns;

        -- Change data with output disabled.
        data <= "0101010110101010";
        wait for 10ns;

        -- Enable output.
        enabledt <= '1';
        wait for 10ns;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.6: TB_cic_decim.vhd

```
--------------------------------------------------------------------------------
-- Test Bench for cic_decim
-- Last Modified: 22 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2011 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TB_cic_decim is
end TB_cic_decim;

architecture behavioral of TB_cic_decim is
    -- Component declaration for the Unit Under Test (UUT)
    component cic_decim is
        generic(
            bw                  : integer := 16;    -- # of bits for input
            N                   : integer := 4; -- # of filter stages
            log2_of_max_rate    : integer := 7; -- log2 of max sampling rate
            maxbitgain          : integer := 28 -- N*log2_of_max_rate
        );

        port(
            clock       : in    std_logic;
            reset       : in    std_logic;
            enable      : in    std_logic;
            rate        : in    std_logic_vector(7 downto 0);
            strobe_in   : in    std_logic;
            strobe_out  : in    std_logic;
            signal_in   : in    std_logic_vector(bw-1 downto 0);
            signal_out  : out   std_logic_vector(bw-1 downto 0)
        );
    end component cic_decim;

    -- Inputs
    signal clock : std_logic := '0';
    signal reset : std_logic := '0';
    signal enable : std_logic := '0';
    signal rate : std_logic_vector(7 downto 0) := (others => '0');
    signal strobe_in : std_logic := '0';
    signal strobe_out : std_logic := '0';
    signal signal_in : std_logic_vector(15 downto 0) := (others => '0');

    -- Outputs
    signal signal_out : std_logic_vector(15 downto 0);

    -- Clock Period
    constant clock_period : time := 10ns;
begin
```

```
    -- Instantiate the UUT.
    UUT : cic_decim
        port map(clock => clock, reset => reset, enable => enable, rate => rate,
            strobe_in => strobe_in, strobe_out => strobe_out, signal_in => signal_in,
            signal_out => signal_out);

    -- Clock Process
    clock_process : process
    begin
        clock <= '0';
        wait for clock_period / 2;
        clock <= '1';
        wait for clock_period / 2;
    end process clock_process;

    -- Strobe Process
    strobe_proc : process
    begin
        wait for 7 * clock_period;
        strobe_out <= '1';
        wait for clock_period;
        strobe_out <= '0';
    end process strobe_proc;

    -- We have a sample on every clock, so just keep strobe_out enabled.
    strobe_in <= '1';

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for one clock period.
        wait for clock_period;

        -- Assert the reset.
        reset <= '1';
        wait for clock_period;

        -- Enable the filter.
        reset <= '0';
        enable <= '1';

        -- Set the rate to 7.  This is actually a rate of 32 because of the
        -- codecs that we use.  The formula is (rate+1)*4.
        rate <= conv_std_logic_vector(7, rate'LENGTH);

        -- Test values of 1, max, min, and -1.
        signal_in <= "0000000000000001";
        wait for 100 * clock_period;
        signal_in <= "0111111111111111";
        wait for 100 * clock_period;
        signal_in <= "1000000000000000";
        wait for 100 * clock_period;
        signal_in <= "1111111111111111";
        wait for 100 * clock_period;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.7: TB_cic_dec_shifter.vhd

```
--------------------------------------------------------------------------------
-- Test Bench for cic_dec_shifter
-- Last Modified: 22 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2011 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TB_cic_dec_shifter is
end TB_cic_dec_shifter;

architecture behavioral of TB_cic_dec_shifter is
    -- Component declaration for the Unit Under Test (UUT)
    component cic_dec_shifter is
        generic(
            bw            : integer := 16;
            maxbitgain    : integer := 28
        );

        port(
            rate          : in    std_logic_vector(7 downto 0);
            signal_in     : in    std_logic_vector(bw+maxbitgain-1 downto 0);
            signal_out    : out   std_logic_vector(bw-1 downto 0)
        );
    end component cic_dec_shifter;

    -- Inputs
    signal rate : std_logic_vector(7 downto 0) := (others => '0');
    signal signal_in : std_logic_vector(43 downto 0) := (others => '0');

    -- Outputs
    signal signal_out : std_logic_vector(15 downto 0);
begin
```

```
    -- Instantiate the UUT.
    UUT : cic_dec_shifter
        generic map(bw => 16, maxbitgain => 28)
        port map(rate => rate, signal_in => signal_in, signal_out => signal_out);

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for 10ns.
        wait for 10ns;

        -- Set signal_in to a constant value (0x0123456789AB).
        signal_in <= "000100100011010001010110011110001001101010 11";

        for i in 4 to 127 loop
            rate <= conv_std_logic_vector(i, rate'LENGTH);
            wait for 10ns;
        end loop;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.8: TB_cic_interp.vhd

```
--------------------------------------------------------------------------------
-- Test Bench for cic_interp
-- Last Modified: 21 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2011 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TB_cic_interp is
end TB_cic_interp;

-- This test bench is inspired by the justinterp_tb.v test included with the
-- original USRP source code.
architecture behavioral of TB_cic_interp is
    -- Component declaration for the Unit Under Test (UUT)
    component cic_interp is
        generic(
            bw                  : integer := 16;    -- # of bits for input
            N                   : integer := 4; -- # of filter stages
            log2_of_max_rate    : integer := 7; -- log2 of max sampling rate
            maxbitgain          : integer := 21 --(N-1)*log2_of_max_rate
        );

        port(
            clock       : in    std_logic;
            reset       : in    std_logic;
            enable      : in    std_logic;
            rate        : in    std_logic_vector(7 downto 0);
            strobe_in   : in    std_logic;
            strobe_out  : in    std_logic;
            signal_in   : in    std_logic_vector(bw-1 downto 0);
            signal_out  : out   std_logic_vector(bw-1 downto 0)
        );
    end component cic_interp;

    -- Inputs
    signal clock : std_logic := '0';
    signal reset : std_logic := '0';
    signal enable : std_logic := '0';
    signal rate : std_logic_vector(7 downto 0) := (others => '0');
    signal strobe_in : std_logic := '0';
    signal strobe_out : std_logic := '0';
    signal signal_in : std_logic_vector(15 downto 0) := (others => '0');

    -- Outputs
    signal signal_out : std_logic_vector(15 downto 0);

   -- Clock Period
    constant clock_period : time := 10ns;
begin
    -- Instantiate the UUT.
    UUT : cic_interp
        port map(clock => clock, reset => reset, enable => enable, rate => rate,
            strobe_in => strobe_in, strobe_out => strobe_out, signal_in => signal_in,
            signal_out => signal_out);

    -- Clock Process
    clock_process : process
    begin
        clock <= '0';
        wait for clock_period / 2;
        clock <= '1';
        wait for clock_period / 2;
    end process clock_process;

    -- Strobe Process
    strobe_proc : process
    begin
        wait for 7 * clock_period;
        strobe_in <= '1';
        wait for clock_period;
        strobe_in <= '0';
    end process strobe_proc;

    -- We want a sample on every clock, so just keep strobe_out enabled.
    strobe_out <= '1';

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for one clock period.
        wait for clock_period;

        -- Assert the reset.
        reset <= '1';
        wait for clock_period;

        -- Enable the filter.
        reset <= '0';
        enable <= '1';

        -- Set the rate to 7.  This is actually a rate of 32 because of the
        -- codecs that we use.  The formula is (rate+1)*4.
        rate <= conv_std_logic_vector(7, rate'LENGTH);

        -- Test values of 1, max, min, and -1.
        signal_in <= "0000000000000001";
        wait for 100 * clock_period;
        signal_in <= "0111111111111111";
        wait for 100 * clock_period;
        signal_in <= "1000000000000000";
        wait for 100 * clock_period;
        signal_in <= "1111111111111111";
        wait for 100 * clock_period;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.9: TB_cic_int_shifter.vhd

```
------------------------------------------------------------------------------
-- Test Bench for cic_int_shifter
-- Last Modified: 20 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2011 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TB_cic_int_shifter is
end TB_cic_int_shifter;

architecture behavioral of TB_cic_int_shifter is
    -- Component declaration for the Unit Under Test (UUT)
    component cic_int_shifter is
        generic(
            bw              : integer := 16;
            maxbitgain  : integer := 21
        );

        port(
            rate            : in    std_logic_vector(7 downto 0);
            signal_in   : in    std_logic_vector(bw+maxbitgain-1 downto 0);
            signal_out  : out   std_logic_vector(bw-1 downto 0)
        );
    end component cic_int_shifter;

    -- Inputs
    signal rate : std_logic_vector(7 downto 0) := (others => '0');
    signal signal_in : std_logic_vector(36 downto 0) := (others => '0');

    -- Outputs
    signal signal_out : std_logic_vector(15 downto 0);
begin
    -- Instantiate the UUT.
    UUT : cic_int_shifter
        generic map(bw => 16, maxbitgain => 21)
        port map(rate => rate, signal_in => signal_in, signal_out => signal_out);

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for 10ns.
        wait for 10ns;

        -- Set signal_in to a constant value (0x0123456789).
        signal_in <= "0000100100011010001010110011110001001";

        -- Test each sample rate.  The minimum possible value is 4.
        for i in 4 to 127 loop
            rate <= conv_std_logic_vector(i, rate'LENGTH);
            wait for 10ns;
        end loop;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.10: TB_clk_divider.vhd

```
--------------------------------------------------------------------------------
-- Test Bench for clk_divider
-- Last Modified: 14 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TB_clk_divider is
end TB_clk_divider;

architecture behavioral of TB_clk_divider is
    -- Component declaration for the Unit Under Test (UUT)
    component clk_divider is
        port(
            reset       : in    std_logic;
            in_clk  : in    std_logic;
            out_clk : out   std_logic;
            ratio       : in    std_logic_vector(7 downto 0)
        );
    end component clk_divider;

    -- Inputs
    signal reset : std_logic := '0';
    signal in_clk : std_logic := '0';
    signal ratio : std_logic_vector(7 downto 0) := (others => '0');

    -- Outputs
    signal out_clk : std_logic;

   -- Clock Period
    constant in_clk_period : time := 10ns;
```

```
begin
    -- Instantiate the UUT.
    UUT : clk_divider
        port map(reset => reset, in_clk => in_clk, out_clk => out_clk, ratio => ratio);

    -- Clock Process
    in_clk_process : process
    begin
        in_clk <= '0';
        wait for in_clk_period / 2;
        in_clk <= '1';
        wait for in_clk_period / 2;
    end process in_clk_process;

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for one clock period.
        wait for in_clk_period;

        -- Assert the reset.
        reset <= '1';
        wait for in_clk_period;

        -- Deassert the reset.  Set the ratio to 2.
        reset <= '0';
        ratio <= "00000010";
        wait for 8 * in_clk_period;

        -- Assert the reset.
        reset <= '1';
        wait for 4 * in_clk_period;

        -- New ratio of 3.
        reset <= '0';
        ratio <= "00000011";
        wait for 12 * in_clk_period;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.11: TB_coeff_rom.vhd

```
------------------------------------------------------------------------------
-- Test Bench for coeff_rom
-- Last Modified: 14 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
```

```
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
-------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TB_coeff_rom is
end TB_coeff_rom;

architecture behavioral of TB_coeff_rom is
    -- Component declaration for the Unit Under Test (UUT)
    component coeff_rom is
        port(
            clock   : in    std_logic;
            addr    : in    std_logic_vector(2 downto 0);
            data    : out   std_logic_vector(15 downto 0)
        );
    end component coeff_rom;

    -- Inputs
    signal clock : std_logic := '0';
    signal addr : std_logic_vector(2 downto 0) := (others => '0');

    -- Outputs
    signal data : std_logic_vector(15 downto 0);

    -- Clock Period
    constant clock_period : time := 10ns;
begin
    -- Instantiate the UUT.
    UUT : coeff_rom
        port map(clock => clock, addr => addr, data => data);

    -- Clock Process
    clock_process : process
    begin
        clock <= '0';
        wait for clock_period / 2;
        clock <= '1';
        wait for clock_period / 2;
    end process clock_process;

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for one clock period.
        wait for clock_period;

        -- Test each possible address.
        for i in 0 to 7 loop
            addr <= conv_std_logic_vector(i, addr'LENGTH);
            wait for clock_period;
        end loop;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.12: TB_cordic.vhd

```
--------------------------------------------------------------------------------
-- Test Bench for cordic
-- Last Modified: 23 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2011 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TB_cordic is
end TB_cordic;

architecture behavioral of TB_cordic is
    -- Component declaration for the Unit Under Test (UUT)
    component cordic is
        generic(
            bitwidth    : integer := 16;
            zwidth  : integer := 16
        );

        port(
            clock       : in    std_logic;
            reset       : in    std_logic;
            enable  : in    std_logic;
            xi          : in    std_logic_vector(bitwidth-1 downto 0);
            yi          : in    std_logic_vector(bitwidth-1 downto 0);
            xo          : out   std_logic_vector(bitwidth-1 downto 0);
            yo          : out   std_logic_vector(bitwidth-1 downto 0);
            zi          : in    std_logic_vector(zwidth-1 downto 0);
            zo          : out   std_logic_vector(zwidth-1 downto 0)
        );
    end component cordic;

    -- Inputs
    signal clock : std_logic := '0';
    signal reset : std_logic := '0';
    signal enable : std_logic := '0';
    signal xi : std_logic_vector(15 downto 0) := (others => '0');
    signal yi : std_logic_vector(15 downto 0) := (others => '0');
    signal zi : std_logic_vector(15 downto 0) := (others => '0');

    -- Outputs
    signal xo : std_logic_vector(15 downto 0);
    signal yo : std_logic_vector(15 downto 0);
    signal zo : std_logic_vector(15 downto 0);

    -- Clock Period
    constant clock_period : time := 10ns;
begin
    -- Instantiate the UUT.
    UUT : cordic
        generic map(bitwidth => 16, zwidth => 16)
        port map(clock => clock, reset => reset, enable => enable, xi => xi,
            yi => yi, xo => xo, yo => yo, zi => zi, zo => zo);

   -- Clock Process
    clock_process : process
    begin
        clock <= '0';
        wait for clock_period / 2;
        clock <= '1';
        wait for clock_period / 2;
    end process clock_process;

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for one clock period.
        wait for clock_period;

        -- Assert the reset.
        reset <= '1';
        wait for clock_period;

        -- Enable the unit.
        reset <= '0';
        enable <= '1';
        wait for clock_period;

        --
        for i in 0 to 255 loop
            xi <= conv_std_logic_vector(i mod 2, xi'LENGTH);
            yi <= conv_std_logic_vector(2*(i mod 2), yi'LENGTH);
            zi(13 downto 0) <= "01111111111111";
            zi(15 downto 14) <= conv_std_logic_vector(i mod 4, 2);
            wait for clock_period;
        end loop;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.13: TB_cordic_stage.vhd

```
--------------------------------------------------------------------------------
-- Test Bench for cordic_stage
-- Last Modified: 23 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2011 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TB_cordic_stage is
end TB_cordic_stage;

architecture behavioral of TB_cordic_stage is
    -- Component declaration for the Unit Under Test (UUT)
    component cordic_stage is
        generic(
            bitwidth    : integer := 16;
            zwidth  : integer := 16;
            shift       : integer := 1
        );

        port(
            clock       : in    std_logic;
            reset       : in    std_logic;
            enable  : in    std_logic;
            xi          : in    std_logic_vector(bitwidth-1 downto 0);
            yi          : in    std_logic_vector(bitwidth-1 downto 0);
            zi          : in    std_logic_vector(zwidth-1 downto 0);
            const       : in    std_logic_vector(zwidth-1 downto 0);
            xo          : out   std_logic_vector(bitwidth-1 downto 0);
            yo          : out   std_logic_vector(bitwidth-1 downto 0);
            zo          : out   std_logic_vector(zwidth-1 downto 0)
        );
    end component cordic_stage;

    -- Inputs
    signal clock : std_logic := '0';
    signal reset : std_logic := '0';
    signal enable : std_logic := '0';
    signal xi : std_logic_vector(15 downto 0) := (others => '0');
    signal yi : std_logic_vector(15 downto 0) := (others => '0');
    signal zi : std_logic_vector(15 downto 0) := (others => '0');
    signal const : std_logic_vector(15 downto 0) := (others => '0');

    -- Outputs
    signal xo : std_logic_vector(15 downto 0);
    signal yo : std_logic_vector(15 downto 0);
    signal zo : std_logic_vector(15 downto 0);

    -- Clock Period
    constant clock_period : time := 10ns;
begin
    -- Instantiate the UUT.
    UUT : cordic_stage
        generic map(bitwidth => 16, zwidth => 16, shift => 3)
        port map(clock => clock, reset => reset, enable => enable, xi => xi,
            yi => yi, zi => zi, const => const, xo => xo, yo => yo, zo => zo);

    -- Clock Process
    clock_process : process
    begin
        clock <= '0';
        wait for clock_period / 2;
        clock <= '1';
        wait for clock_period / 2;
    end process clock_process;

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for one clock period.
        wait for clock_period;

        -- Assert the reset.
        reset <= '1';
        wait for clock_period;

        -- Enable the unit.
        reset <= '0';
        enable <= '1';

        -- Use one of the constants from cordic.vhd to test.
        const <= conv_std_logic_vector(1297, const'LENGTH);

        -- Test +x, +y, +z.
        xi <= conv_std_logic_vector(300, xi'LENGTH);
        yi <= conv_std_logic_vector(250, yi'LENGTH);
        zi <= conv_std_logic_vector(2000, zi'LENGTH);
        wait for clock_period;

        -- Test +x, +y, -z.
        xi <= conv_std_logic_vector(300, xi'LENGTH);
        yi <= conv_std_logic_vector(250, yi'LENGTH);
        zi <= conv_std_logic_vector(-2000, zi'LENGTH);
        wait for clock_period;

        -- Test +x, -y, +z.
        xi <= conv_std_logic_vector(300, xi'LENGTH);
        yi <= conv_std_logic_vector(-250, yi'LENGTH);
        zi <= conv_std_logic_vector(2000, zi'LENGTH);
        wait for clock_period;

        -- Test +x, -y, -z.
        xi <= conv_std_logic_vector(300, xi'LENGTH);
        yi <= conv_std_logic_vector(-250, yi'LENGTH);
        zi <= conv_std_logic_vector(-2000, zi'LENGTH);
        wait for clock_period;

        -- Test -x, +y, +z.
        xi <= conv_std_logic_vector(-300, xi'LENGTH);
        yi <= conv_std_logic_vector(250, yi'LENGTH);
        zi <= conv_std_logic_vector(2000, zi'LENGTH);
        wait for clock_period;

        -- Test -x, +y, -z.
        xi <= conv_std_logic_vector(-300, xi'LENGTH);
        yi <= conv_std_logic_vector(250, yi'LENGTH);
        zi <= conv_std_logic_vector(-2000, zi'LENGTH);
        wait for clock_period;

        -- Test -x, -y, +z.
        xi <= conv_std_logic_vector(-300, xi'LENGTH);
        yi <= conv_std_logic_vector(-250, yi'LENGTH);
        zi <= conv_std_logic_vector(2000, zi'LENGTH);
        wait for clock_period;

        -- Test -x, -y, -z.
        xi <= conv_std_logic_vector(-300, xi'LENGTH);
        yi <= conv_std_logic_vector(-250, yi'LENGTH);
        zi <= conv_std_logic_vector(-2000, zi'LENGTH);
        wait for clock_period;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.14: TB_fifo.vhd

```
--------------------------------------------------------------------------------
```

```
-- Test Bench for fifo
-- Last Modified: 19 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TB_fifo is
end TB_fifo;

architecture behavioral of TB_fifo is
    -- Component declaration for the Unit Under Test (UUT)
    component fifo is
        generic(
            width           : integer := 16;
            depth           : integer := 1024;
            addr_bits   : integer := 10
        );

        port(
            data        : in    std_logic_vector(width-1 downto 0);
            wrreq       : in    std_logic;
            rdreq       : in    std_logic;
            rdclk       : in    std_logic;
            wrclk       : in    std_logic;
            aclr        : in    std_logic;
            q           : out   std_logic_vector(width-1 downto 0);
            rdfull  : out   std_logic;
            rdempty : out   std_logic;
            rdusedw : out   std_logic_vector(addr_bits-1 downto 0);
            wrfull  : out   std_logic;
            wrempty : out   std_logic;
            wrusedw : out   std_logic_vector(addr_bits-1 downto 0)
        );
    end component fifo;

    -- Inputs
    signal data : std_logic_vector(15 downto 0) := (others => '0');
    signal wrreq : std_logic := '0';
    signal rdreq : std_logic := '0';
    signal rdclk : std_logic := '0';
    signal wrclk : std_logic := '0';
    signal aclr : std_logic := '0';

    -- Outputs
    signal q : std_logic_vector(15 downto 0);
    signal rdfull : std_logic;
    signal rdempty : std_logic;
    signal rdusedw : std_logic_vector(6 downto 0);
    signal wrfull : std_logic;
    signal wrempty : std_logic;
    signal wrusedw : std_logic_vector(6 downto 0);

    -- Clock Period
    constant rdclk_period : time := 10ns;
    constant wrclk_period : time := 10ns;
begin
    -- Instantiate the UUT.
   UUT : fifo
        generic map(width => 16, depth => 128, addr_bits => 7)
        port map(data => data, wrreq => wrreq, rdreq => rdreq, rdclk => rdclk,
            wrclk => wrclk, aclr => aclr, q => q, rdfull => rdfull, rdempty => rdempty,
            rdusedw => rdusedw, wrfull => wrfull, wrempty => wrempty, wrusedw => wrusedw);

    -- Clock Process (rdclk)
    rdclk_process : process
    begin
        rdclk <= '0';
        wait for rdclk_period / 2;
        rdclk <= '1';
        wait for rdclk_period / 2;
    end process rdclk_process;

    wrclk_process : process
    begin
        wrclk <= '0';
        wait for wrclk_period / 2;
        wrclk <= '1';
        wait for wrclk_period / 2;
    end process wrclk_process;

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for one clock period.
        wait for rdclk_period;

        -- Assert the reset.
        aclr <= '1';
        wait for rdclk_period;
        aclr <= '0';

        -- Write a few items into the FIFO.
        data <= "0000111100001111";
        wrreq <= '1';
        wait for wrclk_period;

        data <= "1111000011110000";
        wrreq <= '1';
        wait for wrclk_period;

        data <= "1100110011001100";
        wrreq <= '1';
        wait for wrclk_period;

        data <= "0011001100110011";
        wrreq <= '1';
        wait for wrclk_period;

        -- Read back from the FIFO.
        wrreq <= '0';
        rdreq <= '1';
        wait for 4 * rdclk_period;

        -- What if we read extra?
        rdreq <= '1';
        wait for rdclk_period;

        -- Send a clear to the FIFO.
        aclr <= '1';
        wait for rdclk_period;

        -- Now, completely fill the FIFO.
        aclr <= '0';
        rdreq <= '0';
        for i in 0 to 127 loop
            data <= conv_std_logic_vector(i, 16);
            wrreq <= '1';
            wait for wrclk_period;
        end loop;

        -- Wait an extra clock to examine the output signals.
        wrreq <= '0';
        wait for rdclk_period;

        -- Write one extra value to the FIFO.  What happens?
        data <= "1111111111111111";
        wrreq <= '1';
        wait for wrclk_period;

        -- Read back the entire FIFO.
        wrreq <= '0';
        for i in 0 to 127 loop
            rdreq <= '1';
            wait for rdclk_period;
        end loop;

        -- Write a few values back into the FIFO.
        rdreq <= '0';
        for i in 0 to 3 loop
            data <= conv_std_logic_vector(i, 16);
            wrreq <= '1';
            wait for wrclk_period;
        end loop;

        -- Now, read and write simultaneously for 10 clocks.
        for i in 0 to 9 loop
            data <= conv_std_logic_vector(i + 10, 16);
            wrreq <= '1';
            rdreq <= '1';
            wait for wrclk_period;
        end loop;

        -- Send a clear signal.
        aclr <= '1';
        wait for rdclk_period;

        -- What if we read now?
        rdreq <= '1';
        wait for 3 * rdclk_period;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.15: TB_io_pins.vhd

```
--------------------------------------------------------------------------------
-- Test Bench for io_pins
-- Last Modified: 14 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library WORK;
use WORK.FPGA_REGS_COMMON.ALL;

entity TB_io_pins is
end TB_io_pins;
```

```

architecture behavioral of TB_io_pins is
    -- Component declaration for the Unit Under Test (UUT)
    component io_pins is
        port(
            io_0              : inout std_logic_vector(15 downto 0);
            io_1              : inout std_logic_vector(15 downto 0);
            io_2              : inout std_logic_vector(15 downto 0);
            io_3              : inout std_logic_vector(15 downto 0);
            reg_0             : in      std_logic_vector(15 downto 0);
            reg_1             : in      std_logic_vector(15 downto 0);
            reg_2             : in      std_logic_vector(15 downto 0);
            reg_3             : in      std_logic_vector(15 downto 0);
            clock             : in      std_logic;
            rx_reset          : in      std_logic;
            tx_reset          : in      std_logic;
            serial_addr     : in      std_logic_vector(6 downto 0);
            serial_data     : in      std_logic_vector(31 downto 0);
            serial_strobe   : in      std_logic
        );
    end component io_pins;

    -- Inputs
    signal reg_0 : std_logic_vector(15 downto 0) := (others => '0');
    signal reg_1 : std_logic_vector(15 downto 0) := (others => '0');
    signal reg_2 : std_logic_vector(15 downto 0) := (others => '0');
    signal reg_3 : std_logic_vector(15 downto 0) := (others => '0');
    signal clock : std_logic := '0';
    signal rx_reset : std_logic := '0';
    signal tx_reset : std_logic := '0';
    signal serial_addr : std_logic_vector(6 downto 0) := (others => '0');
    signal serial_data : std_logic_vector(31 downto 0) := (others => '0');
    signal serial_strobe : std_logic := '0';

    -- Bidirectionals
    signal io_0 : std_logic_vector(15 downto 0);
    signal io_1 : std_logic_vector(15 downto 0);
    signal io_2 : std_logic_vector(15 downto 0);
    signal io_3 : std_logic_vector(15 downto 0);

    -- Clock Period
    constant clock_period : time := 10ns;
begin
    -- Instantiate the UUT.
    UUT : io_pins
        port map(io_0 => io_0, io_1 => io_1, io_2 => io_2, io_3 => io_3,
            reg_0 => reg_0, reg_1 => reg_1, reg_2 => reg_2, reg_3 => reg_3,
            clock => clock, rx_reset => rx_reset, tx_reset => tx_reset,
            serial_addr => serial_addr, serial_data => serial_data,
            serial_strobe => serial_strobe);

    -- Clock Process
    clock_process : process
    begin
        clock <= '0';
        wait for clock_period / 2;
        clock <= '1';
        wait for clock_period / 2;
    end process clock_process;

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for one clock period.
        wait for clock_period;

        -- Set some initial values for the input registers.
        reg_0 <= "0001000100010001";
        reg_1 <= "0010001000100010";
        reg_2 <= "0011001100110011";
        reg_3 <= "0100010001000100";

        -- Enable all all outputs for each register.
        serial_data <= (others => '1');

        serial_addr <= conv_std_logic_vector(FR_OE_0, serial_addr'LENGTH);
        serial_strobe <= '1';
        wait for clock_period;
        serial_strobe <= '0';
        wait for clock_period;

        serial_addr <= conv_std_logic_vector(FR_OE_1, serial_addr'LENGTH);
        serial_strobe <= '1';
        wait for clock_period;
        serial_strobe <= '0';
        wait for clock_period;

        serial_addr <= conv_std_logic_vector(FR_OE_2, serial_addr'LENGTH);
        serial_strobe <= '1';
        wait for clock_period;
        serial_strobe <= '0';
        wait for clock_period;

        serial_addr <= conv_std_logic_vector(FR_OE_3, serial_addr'LENGTH);
        serial_strobe <= '1';
        wait for clock_period;
        serial_strobe <= '0';
        wait for clock_period;

        -- Change the values of each register.
        reg_0 <= "1111111111111111";
        wait for clock_period;

        reg_1 <= "0000000000000000";
        wait for clock_period;

        reg_2 <= "1111000011110000";
        wait for clock_period;

        reg_3 <= "0000111100001111";
        wait for clock_period;

        -- Change the output masks for each register.
        serial_data <= "11111111000000000000000000000000";
        serial_addr <= conv_std_logic_vector(FR_OE_0, serial_addr'LENGTH);
        serial_strobe <= '1';
        wait for clock_period;
        serial_strobe <= '0';
        wait for clock_period;

        serial_data <= "11111111111111110000000000000000";
        serial_addr <= conv_std_logic_vector(FR_OE_1, serial_addr'LENGTH);
        serial_strobe <= '1';
        wait for clock_period;
        serial_strobe <= '0';
        wait for clock_period;

        serial_data <= "11110000111100000000000000000000";
        serial_addr <= conv_std_logic_vector(FR_OE_2, serial_addr'LENGTH);
        serial_strobe <= '1';
        wait for clock_period;
        serial_strobe <= '0';
        wait for clock_period;

        serial_data <= "11111111111111111111111111111111";
        serial_addr <= conv_std_logic_vector(FR_OE_3, serial_addr'LENGTH);
        serial_strobe <= '1';
        wait for clock_period;
        serial_strobe <= '0';
        wait for clock_period;

        -- The two reset signals are never used in io_pins, but we'll test them
        -- anyway.
        tx_reset <= '1';
        wait for clock_period;

        rx_reset <= '1';
        wait for clock_period;

        tx_reset <= '0';
        wait for clock_period;

        rx_reset <= '0';
        wait for clock_period;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.16: TB_mult.vhd

```
--------------------------------------------------------------------------------
-- Test Bench for mult
-- Last Modified: 15 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TB_mult is
end TB_mult;

architecture behavioral of TB_mult is
    -- Component declaration for the Unit Under Test (UUT)
    component mult is
        port(
            clock           : in    std_logic;
            x               : in    std_logic_vector(15 downto 0);
            y               : in    std_logic_vector(15 downto 0);
            product     : out   std_logic_vector(30 downto 0);
            enable_in   : in    std_logic;
            enable_out  : out   std_logic
        );
    end component mult;

    -- Inputs
    signal clock : std_logic := '0';
    signal x : std_logic_vector(15 downto 0) := (others => '0');
    signal y : std_logic_vector(15 downto 0) := (others => '0');
    signal enable_in : std_logic := '0';

    -- Outputs
    signal product : std_logic_vector(30 downto 0);
    signal enable_out : std_logic;

    -- Clock Period
    constant clock_period : time := 10ns;
begin
    -- Instantiate the UUT.
    UUT : mult
        port map(clock => clock, x => x, y => y, product => product,
            enable_in => enable_in, enable_out => enable_out);

    -- Clock Process
    clock_process : process
    begin
        clock <= '0';
        wait for clock_period / 2;
        clock <= '1';
        wait for clock_period / 2;
    end process clock_process;

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for one clock period.
        wait for clock_period;

        -- Enable multiplications.
        enable_in <= '1';

        -- Multiply two positive numbers.
        x <= "0000000000001111";
        y <= "0000000000001111";
        wait for clock_period;

        -- Multiply one positive and one negative number.
        x <= "0000000000001100";
        y <= "1111111111111000";
        wait for clock_period;

        -- Multiply two negative numbers.
        x <= "1111111111111100";
        y <= "1111111111111010";
        wait for clock_period;

        -- Multiply by zero.
        x <= "0000000000000000";
        y <= "0000000000001111";
        wait for clock_period;

        -- Ensure nothing happens when enable_in is low.
        enable_in <= '0';
        wait for 3 * clock_period;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.17: TB_phase_acc.vhd

```
--------------------------------------------------------------------------------
-- Test Bench for phase_acc
-- Last Modified: 23 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2011 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```vhdl
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library WORK;
use WORK.FPGA_REGS_STANDARD.ALL;

entity TB_phase_acc is
end TB_phase_acc;

architecture behavioral of TB_phase_acc is
    -- Component declaration for the Unit Under Test (UUT)
    component phase_acc is
        generic(
            FREQADDR        : integer := 0;
            PHASEADDR   : integer := 0;
            resolution  : integer range 0 to 32 := 32
        );

        port(
            clk             : in    std_logic;
            reset               : in    std_logic;
            enable          : in    std_logic;
            strobe          : in    std_logic;
            serial_addr     : in    std_logic_vector(6 downto 0);
            serial_data     : in    std_logic_vector(31 downto 0);
            serial_strobe   : in    std_logic;
            phase               : out   std_logic_vector(resolution-1 downto 0)
        );
    end component phase_acc;

    -- Inputs
    signal clk : std_logic := '0';
    signal reset : std_logic := '0';
    signal enable : std_logic := '0';
    signal strobe : std_logic := '0';
    signal serial_addr : std_logic_vector(6 downto 0) := (others => '0');
    signal serial_data : std_logic_vector(31 downto 0) := (others => '0');
    signal serial_strobe : std_logic := '0';

    -- Outputs
    signal phase : std_logic_vector(31 downto 0);

    -- Clock Period
    constant clk_period : time := 10ns;
begin
    -- Instantiate the UUT.
    UUT : phase_acc
        generic map(FREQADDR => FR_RX_FREQ_0, PHASEADDR => FR_RX_PHASE_0)
        port map(clk => clk, reset => reset, enable => enable, strobe => strobe,
            serial_addr => serial_addr, serial_data => serial_data,
            serial_strobe => serial_strobe, phase => phase);

    -- Clock Process
    clk_process : process
    begin
        clk <= '0';
        wait for clk_period / 2;
        clk <= '1';
        wait for clk_period / 2;
    end process clk_process;

    -- Strobe Process
    -- We'll just use a 1/4 strobe rate for testing purposes. In practice, this
    -- value will be determined by the decimation rate.
    strobe_process : process
    begin
        strobe <= '0';
        wait for 3 * clk_period;
        strobe <= '1';
        wait for clk_period;
    end process strobe_process;

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for one clock period.
        wait for clk_period;

        -- Assert the reset.
        reset <= '1';
        wait for clk_period;

        -- Set the frequency to 45.
        serial_addr <= conv_std_logic_vector(FR_RX_FREQ_0, serial_addr'LENGTH);
        serial_data <= conv_std_logic_vector(45, serial_data'LENGTH);
        serial_strobe <= '1';
        wait for clk_period;

        -- Enable the unit.
        reset <= '0';
        enable <= '1';
        serial_strobe <= '0';
        wait for 3 * clk_period;

        -- Set a new starting phase of 90.
        serial_addr <= conv_std_logic_vector(FR_RX_PHASE_0, serial_addr'LENGTH);
        serial_data <= conv_std_logic_vector(90, serial_data'LENGTH);
        serial_strobe <= '1';
        wait for clk_period;

        -- Accumulate for a while.
        serial_strobe <= '0';
        wait for 10 * clk_period;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.18: TB_ram16.vhd

```
--------------------------------------------------------------------------------
-- Test Bench for ram16
-- Last Modified: 14 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TB_ram16 is
end TB_ram16;

architecture behavioral of TB_ram16 is
    -- Component declaration for the Unit Under Test (UUT)
    component ram16 is
        port(
            clock           : in    std_logic;
            write_in        : in    std_logic;
            wr_addr     : in    std_logic_vector(3 downto 0);
            wr_data     : in    std_logic_vector(15 downto 0);
            rd_addr     : in    std_logic_vector(3 downto 0);
            rd_data     : out   std_logic_vector(15 downto 0)
        );
    end component ram16;

    -- Inputs
    signal clock : std_logic := '0';
    signal write_in : std_logic := '0';
    signal wr_addr : std_logic_vector(3 downto 0) := (others => '0');
    signal wr_data : std_logic_vector(15 downto 0) := (others => '0');
    signal rd_addr : std_logic_vector(3 downto 0) := (others => '0');

    -- Outputs
    signal rd_data : std_logic_vector(15 downto 0);

    -- Clock Period
    constant clock_period : time := 10ns;
begin
    -- Instantiate the UUT.
    UUT : ram16
        port map(clock => clock, write_in => write_in, wr_addr => wr_addr,
            wr_data => wr_data, rd_addr => rd_addr, rd_data => rd_data);

    -- Clock Process
    clock_process : process
    begin
        clock <= '0';
        wait for clock_period / 2;
        clock <= '1';
        wait for clock_period / 2;
    end process;

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for one clock period.
        wait for clock_period;

        -- Write to all 16 memory locations.
        write_in <= '1';
        for i in 0 to 15 loop
            wr_addr <= conv_std_logic_vector(i, wr_addr'LENGTH);
            wr_data <= conv_std_logic_vector(15 - i, wr_data'LENGTH);
            wait for clock_period;
        end loop;

        -- Now read back from each memory location.
        write_in <= '0';
        for i in 0 to 15 loop
            rd_addr <= conv_std_logic_vector(i, rd_addr'LENGTH);
            wait for clock_period;
        end loop;

        -- Make sure that we can't write when write_in is low.
        wr_addr <= "0100";
        wr_data <= "0000000011111111";
        wait for clock_period;

        rd_addr <= "0100";
        wait for clock_period;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.19: TB_ram16_2sum.vhd

```
--------------------------------------------------------------------------------
```

```
-- Test Bench for ram16_2sum
-- Last Modified: 14 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
-------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TB_ram16_2sum is
end TB_ram16_2sum;

architecture behavioral of TB_ram16_2sum is
    -- Component declaration for the Unit Under Test (UUT)
    component ram16_2sum is
        port(
            clock       : in    std_logic;
            write_in    : in    std_logic;
            wr_addr : in    std_logic_vector(3 downto 0);
            wr_data : in    std_logic_vector(15 downto 0);
            rd_addr1    : in    std_logic_vector(3 downto 0);
            rd_addr2    : in    std_logic_vector(3 downto 0);
            sum     : out   std_logic_vector(15 downto 0)
        );
    end component ram16_2sum;

    -- Inputs
    signal clock : std_logic := '0';
    signal write_in : std_logic := '0';
    signal wr_addr : std_logic_vector(3 downto 0) := (others => '0');
    signal wr_data : std_logic_vector(15 downto 0) := (others => '0');
    signal rd_addr1 : std_logic_vector(3 downto 0) := (others => '0');
    signal rd_addr2 : std_logic_vector(3 downto 0) := (others => '0');

    -- Outputs
   signal sum : std_logic_vector(15 downto 0);

    -- Clock Period
    constant clock_period : time := 10ns;
begin
    -- Instantiate the UUT.
```

```
UUT : ram16_2sum
    port map(clock => clock, write_in => write_in, wr_addr => wr_addr,
        wr_data => wr_data, rd_addr1 => rd_addr1, rd_addr2 => rd_addr2,
        sum => sum);

-- Clock Process
clock_process : process
begin
    clock <= '0';
    wait for clock_period / 2;
    clock <= '1';
    wait for clock_period / 2;
end process;

-- Stimulus Process
stim_proc : process
begin
    -- Do nothing for one clock period.
    wait for clock_period;

    -- Write to all 16 memory locations.
    write_in <= '1';
    for i in 0 to 15 loop
        wr_addr <= conv_std_logic_vector(i, wr_addr'LENGTH);
        wr_data <= conv_std_logic_vector(conv_integer(conv_signed(-i + 7, wr_data'LENGTH)),
            wr_data'LENGTH);
        wait for clock_period;
    end loop;

    -- Now read back from each memory location.
    write_in <= '0';
    for i in 0 to 15 loop
        rd_addr1 <= conv_std_logic_vector(i, rd_addr1'LENGTH);
        rd_addr2 <= conv_std_logic_vector(15 - i, rd_addr2'LENGTH);
        wait for clock_period;
    end loop;

    -- Make sure that we can't write when write_in is low.
    wr_data <= "0000000011111111";
    wait for 2 * clock_period;

    -- Finally, test the 'add 1' part.
    write_in <= '1';
    wr_addr <= "0010";
    wr_data <= "1111000000001111";
    wait for clock_period;

    wr_addr <= "0001";
    wr_data <= "0000000000000000";
    wait for clock_period;

    write_in <= '0';
    rd_addr1 <= "0010";
    rd_addr2 <= "0001";
    wait for 3 * clock_period;

    -- End test.
```

```
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.20: TB_rssi.vhd

```
---------------------------------------------------------------------------
-- Test Bench for rssi
-- Last Modified: 26 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2011 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
---------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TB_rssi is
end TB_rssi;

architecture behavioral of TB_rssi is
    -- Component declaration for the Unit Under Test (UUT)
    component rssi is
        port(
            clock       : in    std_logic;
            reset       : in    std_logic;
            enable      : in    std_logic;
            adc         : in    std_logic_vector(11 downto 0);
            rssi        : out   std_logic_vector(15 downto 0);
            over_count  : out   std_logic_vector(15 downto 0)
        );
    end component rssi;

    -- Inputs
    signal clock : std_logic := '0';
    signal reset : std_logic := '0';
    signal enable : std_logic := '0';
    signal adc : std_logic_vector(11 downto 0) := (others => '0');

    -- Outputs
    signal rssi_t : std_logic_vector(15 downto 0);
    signal over_count : std_logic_vector(15 downto 0);

    -- Clock Period
    constant clock_period : time := 10ns;
begin
    -- Instantiate the UUT.
    UUT : rssi
        port map(clock => clock, reset => reset, enable => enable, adc => adc,
            rssi => rssi_t, over_count => over_count);

    -- Clock Process
    clock_process : process
    begin
        clock <= '0';
        wait for clock_period / 2;
        clock <= '1';
        wait for clock_period / 2;
    end process clock_process;

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for one clock period.
        wait for clock_period;

        -- Assert the reset.
        reset <= '1';
        wait for clock_period;

        -- Enable the unit.
        reset <= '0';
        enable <= '1';
        wait for clock_period;

        -- Test some non-error values.  These are signed.
        for i in 0 to 19 loop
            adc <= conv_std_logic_vector(i * 200, adc'LENGTH);
            wait for clock_period;
            adc <= conv_std_logic_vector(-(i * 200), adc'LENGTH);
            wait for clock_period;
        end loop;

        -- Signal overflows.
        for i in 0 to 3 loop
            adc <= "011111111111";
            wait for clock_period;
        end loop;

        -- Signal underflows.
        for i in 0 to 3 loop
            adc <= "111111111111";
            wait for clock_period;
        end loop;

        -- Test the reset again.
        reset <= '1';
        wait for clock_period;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.21: TB_rx_dcoffset.vhd

```
------------------------------------------------------------------------------------------
-- Test Bench for rx_dcoffset
-- Last Modified: 28 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2011 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
------------------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

library WORK;
use WORK.FPGA_REGS_COMMON.ALL;

entity TB_rx_dcoffset is
end TB_rx_dcoffset;

architecture behavioral of TB_rx_dcoffset is
    -- Component declaration for the Unit Under Test (UUT)
    component rx_dcoffset is
        generic(
            MYADDR  : integer := 0
        );

        port(
            clock           : in    std_logic;
            enable          : in    std_logic;
            reset           : in    std_logic;
            adc_in          : in    std_logic_vector(15 downto 0);
            adc_out         : out   std_logic_vector(15 downto 0);
            serial_addr     : in    std_logic_vector(6 downto 0);
            serial_data     : in    std_logic_vector(31 downto 0);
            serial_strobe   : in    std_logic
        );
    end component rx_dcoffset;

    -- Inputs
    signal clock : std_logic := '0';
    signal enable : std_logic := '0';
    signal reset : std_logic := '0';
    signal adc_in : std_logic_vector(15 downto 0) := (others => '0');
    signal serial_addr : std_logic_vector(6 downto 0) := (others => '0');
    signal serial_data : std_logic_vector(31 downto 0) := (others => '0');
    signal serial_strobe : std_logic := '0';

    -- Outputs
    signal adc_out : std_logic_vector(15 downto 0);

    -- Clock Period
    constant clock_period : time := 10ns;
begin
    -- Instantiate the UUT.
    UUT : rx_dcoffset
        generic map(MYADDR => FR_ADC_OFFSET_0)
        port map(clock => clock, enable => enable, reset => reset, adc_in => adc_in,
            adc_out => adc_out, serial_addr => serial_addr, serial_data => serial_data,
            serial_strobe => serial_strobe);

    -- Clock Process
    clock_process : process
    begin
        clock <= '0';
        wait for clock_period / 2;
        clock <= '1';
        wait for clock_period / 2;
    end process clock_process;

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for one clock period.
        wait for clock_period;

        -- Assert the reset.
        reset <= '1';
        wait for clock_period;

        -- Enable the unit.
        reset <= '0';
        enable <= '1';
        wait for clock_period;

        -- Set up the serial bus.
        serial_addr <= conv_std_logic_vector(FR_ADC_OFFSET_0, serial_addr'LENGTH);
        serial_data <= "00000000111111110000000000000000";
        serial_strobe <= '1';
        wait for clock_period;

        -- Add some more samples.
        serial_strobe <= '0';
        for i in 0 to 15 loop
            adc_in <= "0111000011110000";
            wait for clock_period;
        end loop;

        -- Try a negative sample or two!
        for i in 0 to 3 loop
            adc_in <= "1000000000000011";
            wait for clock_period;
        end loop;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.22: TB_serial_io.vhd

```
--------------------------------------------------------------------------------
-- Test Bench for serial_io
-- Last Modified: 14 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TB_serial_io is
end TB_serial_io;

architecture behavioral of TB_serial_io is
    -- Component declaration for the Unit Under Test (UUT)
    component serial_io is
        port(
            master_clk      : in    std_logic;
            serial_clock    : in    std_logic;
            serial_data_in  : in    std_logic;
            enable          : in    std_logic;
            reset               : in    std_logic;
            serial_data_out : out   std_logic;
            serial_addr     : out   std_logic_vector(6 downto 0);
            serial_data         : out   std_logic_vector(31 downto 0);
            serial_strobe       : out   std_logic;
            readback_0          : in    std_logic_vector(31 downto 0);
            readback_1          : in    std_logic_vector(31 downto 0);
            readback_2          : in    std_logic_vector(31 downto 0);
            readback_3          : in    std_logic_vector(31 downto 0);
            readback_4          : in    std_logic_vector(31 downto 0);
            readback_5          : in    std_logic_vector(31 downto 0);
            readback_6          : in    std_logic_vector(31 downto 0);
            readback_7          : in    std_logic_vector(31 downto 0)
        );
    end component serial_io;

    -- Inputs
    signal master_clk : std_logic := '0';
    signal serial_clock : std_logic := '0';
    signal serial_data_in : std_logic := '0';
    signal enable : std_logic := '0';
    signal reset : std_logic := '0';
    signal readback_0 : std_logic_vector(31 downto 0) := (others => '0');
    signal readback_1 : std_logic_vector(31 downto 0) := (others => '0');
    signal readback_2 : std_logic_vector(31 downto 0) := (others => '0');
    signal readback_3 : std_logic_vector(31 downto 0) := (others => '0');
    signal readback_4 : std_logic_vector(31 downto 0) := (others => '0');
    signal readback_5 : std_logic_vector(31 downto 0) := (others => '0');
    signal readback_6 : std_logic_vector(31 downto 0) := (others => '0');
    signal readback_7 : std_logic_vector(31 downto 0) := (others => '0');

    -- Outputs
    signal serial_data_out : std_logic;
    signal serial_addr : std_logic_vector(6 downto 0);
    signal serial_data : std_logic_vector(31 downto 0);
    signal serial_strobe : std_logic;

    -- Clock Periods
    constant master_clk_period : time := 10ns;
    constant serial_clock_period : time := 10ns;

    -- Constants for operation instructions.
    constant read_reg0 : std_logic_vector(7 downto 0) := "10000001";
    constant read_reg1 : std_logic_vector(7 downto 0) := "10000010";
    constant read_reg2 : std_logic_vector(7 downto 0) := "10000011";
    constant read_reg3 : std_logic_vector(7 downto 0) := "10000100";
    constant read_reg4 : std_logic_vector(7 downto 0) := "10000101";
    constant read_reg5 : std_logic_vector(7 downto 0) := "10000110";
    constant read_reg6 : std_logic_vector(7 downto 0) := "10000111";
    constant read_reg7 : std_logic_vector(7 downto 0) := "10001000";
    constant write_1 : std_logic_vector(39 downto 0) :=
        "0000000011001100110011001100110011001100";
    constant write_2 : std_logic_vector(39 downto 0) :=
        "0000000000110011001100110011001100110011";
begin
    -- Instantiate the UUT.
    UUT : serial_io
        port map(master_clk => master_clk, serial_clock => serial_clock,
            serial_data_in => serial_data_in, enable => enable, reset => reset,
            serial_data_out => serial_data_out, serial_addr => serial_addr,
            serial_data => serial_data, serial_strobe => serial_strobe,
            readback_0 => readback_0, readback_1 => readback_1,
            readback_2 => readback_2, readback_3 => readback_3,
            readback_4 => readback_4, readback_5 => readback_5,
            readback_6 => readback_6, readback_7 => readback_7);

    -- Clock Process (master_clk)
    master_clk_process : process
    begin
        master_clk <= '0';
        wait for master_clk_period / 2;
        master_clk <= '1';
        wait for master_clk_period / 2;
    end process master_clk_process;

    -- Clock Process (serial_clock)
    serial_clock_process : process
    begin
        serial_clock <= '0';
        wait for serial_clock_period / 2;
        serial_clock <= '1';
        wait for serial_clock_period / 2;
    end process serial_clock_process;

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for one clock period.
        wait for master_clk_period;

        -- Set values for the readback registers.
        readback_0 <= "00000000000000000000000000000000";
        readback_1 <= "00010001000100010001000100010001";
        readback_2 <= "00100010001000100010001000100010";
        readback_3 <= "00110011001100110011001100110011";
        readback_4 <= "01000100010001000100010001000100";
        readback_5 <= "01010101010101010101010101010101";
        readback_6 <= "01100110011001100110011001100110";
        readback_7 <= "01110111011101110111011101110111";

        -- Enable the reset.
        reset <= '1';
        wait for master_clk_period;

        -- Disable the reset.
        reset <= '0';

        -- Each operation is 40 cycles.  First, we test reading of all eight
        -- readback registers.
        enable <= '1';
        for i in read_reg0'RANGE loop
            serial_data_in <= read_reg0(i);
            wait for serial_clock_period;
        end loop;

        wait for master_clk_period;
        enable <= '0';
```

```
        wait for master_clk_period;
        enable <= '1';

        for i in read_reg1'RANGE loop
            serial_data_in <= read_reg1(i);
            wait for serial_clock_period;
        end loop;

        wait for master_clk_period;
        enable <= '0';
        wait for master_clk_period;
        enable <= '1';

        for i in read_reg2'RANGE loop
            serial_data_in <= read_reg2(i);
            wait for serial_clock_period;
        end loop;

        wait for master_clk_period;
        enable <= '0';
        wait for master_clk_period;
        enable <= '1';

        for i in read_reg3'RANGE loop
            serial_data_in <= read_reg3(i);
            wait for serial_clock_period;
        end loop;

        wait for master_clk_period;
        enable <= '0';
        wait for master_clk_period;
        enable <= '1';

        for i in read_reg4'RANGE loop
            serial_data_in <= read_reg4(i);
            wait for serial_clock_period;
        end loop;

        wait for master_clk_period;
        enable <= '0';
        wait for master_clk_period;
        enable <= '1';

        for i in read_reg5'RANGE loop
            serial_data_in <= read_reg5(i);
            wait for serial_clock_period;
        end loop;

        wait for master_clk_period;
        enable <= '0';
        wait for master_clk_period;
        enable <= '1';

        for i in read_reg6'RANGE loop
            serial_data_in <= read_reg6(i);
            wait for serial_clock_period;
        end loop;

        wait for master_clk_period;
        enable <= '0';
        wait for master_clk_period;
        enable <= '1';

        for i in read_reg7'RANGE loop
            serial_data_in <= read_reg7(i);
            wait for serial_clock_period;
        end loop;

        wait for master_clk_period;
        enable <= '0';
        wait for master_clk_period;
        enable <= '1';

        -- Now, perform two write tests.
        for i in write_1'RANGE loop
            serial_data_in <= write_1(i);
            wait for serial_clock_period;
        end loop;

        wait for master_clk_period;
        enable <= '0';
        wait for master_clk_period;
        enable <= '1';

        for i in write_2'RANGE loop
            serial_data_in <= write_2(i);
            wait for serial_clock_period;
        end loop;

        wait for master_clk_period;
        enable <= '0';
        wait for master_clk_period;

        -- Disable the module while altering serial_in.
        enable <= '0';
        serial_data_in <= '1';
        wait for serial_clock_period;
        serial_data_in <= '0';
        wait for serial_clock_period;
        serial_data_in <= '1';
        wait for serial_clock_period;

        -- Enable the reset.
        reset <= '1';
        wait for master_clk_period;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.23: TB_setting_reg.vhd

```
----------------------------------------------------------------------------------
-- Test Bench for setting_reg
-- Last Modified: 13 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TB_setting_reg is
end TB_setting_reg;

architecture behavioral of TB_setting_reg is
    -- Component declaration for the Unit Under Test (UUT)
    component setting_reg is
        generic(
            my_addr : integer := 0
        );

        port(
            clock       : in    std_logic;
            reset       : in    std_logic;
            strobe   : in    std_logic;
            addr        : in    std_logic_vector(6 downto 0);
            d_in        : in    std_logic_vector(31 downto 0);
            d_out       : out   std_logic_vector(31 downto 0);
            changed : out   std_logic
        );
    end component setting_reg;

    -- Inputs
    signal clock : std_logic := '0';
    signal reset : std_logic := '0';
    signal strobe : std_logic := '0';
    signal addr : std_logic_vector(6 downto 0) := (others => '0');
    signal d_in : std_logic_vector(31 downto 0) := (others => '0');

    -- Outputs
    signal d_out1 : std_logic_vector(31 downto 0);
    signal d_out2 : std_logic_vector(31 downto 0);
    signal d_out3 : std_logic_vector(31 downto 0);
    signal changed1 : std_logic;
    signal changed2 : std_logic;
    signal changed3 : std_logic;

    -- Clock period
    constant clock_period : time := 10ns;
begin
    -- Instantiate the UUTs.
    UUT1 : setting_reg
        generic map(my_addr => 1)
        port map(clock => clock, reset => reset, strobe => strobe, addr => addr,
            d_in => d_in, d_out => d_out1, changed => changed1);

    UUT2 : setting_reg
        generic map(my_addr => 2)
        port map(clock => clock, reset => reset, strobe => strobe, addr => addr,
            d_in => d_in, d_out => d_out2, changed => changed2);

    UUT3 : setting_reg
        generic map(my_addr => 3)
        port map(clock => clock, reset => reset, strobe => strobe, addr => addr,
            d_in => d_in, d_out => d_out3, changed => changed3);

    -- Clock Process
    clock_process : process
    begin
        clock <= '0';
        wait for clock_period / 2;
        clock <= '1';
        wait for clock_period / 2;
    end process clock_process;

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for one clock.
        wait for clock_period;

        -- Enable reset.
        reset <= '1';
        wait for clock_period;

        -- Change values while reset is asserted.
        addr <= "0000001";
        d_in <= "11110000110000111010010110010110";
        strobe <= '1';
        wait for clock_period;

        -- Disable reset.
        reset <= '0';
        strobe <= '0';
        wait for clock_period;

        -- Assert the strobe.
        strobe <= '1';
        wait for clock_period;

        -- Deassert the strobe.
        strobe <= '0';
        wait for clock_period;

        -- Try three sequential writes to different registers.
        addr <= "0000001";
        d_in <= "00010001000100010001000100010001";
        strobe <= '1';
        wait for clock_period;

        addr <= "0000010";
        d_in <= "00100010001000100010001000100010";
        strobe <= '1';
        wait for clock_period;

        addr <= "0000011";
        d_in <= "00110011001100110011001100110011";
        strobe <= '1';
        wait for clock_period;

        -- Try to write to a register that does not exist.
        addr <= "0000100";
        d_in <= "01000100010001000100010001000100";
        strobe <= '1';
        wait for clock_period;

        -- Valid parameters, no strobe.
        addr <= "0000001";
        d_in <= "11111111111111111111111111111111";
        strobe <= '0';
        wait for clock_period;

        -- Enable the reset.
        reset <= '1';
        wait for clock_period;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.24: TB_sign_extend.vhd

```
-------------------------------------------------------------------------------
-- Test Bench for sign_extend
-- Last Modified: 14 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```vhdl
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TB_sign_extend is
end TB_sign_extend;

architecture behavioral of TB_sign_extend is
    -- Component declaration for the Unit Under Test (UUT)
    component sign_extend is
        generic(
            bits_in : integer := 0;
            bits_out    : integer := 0
        );

        port(
            d_in    : in    std_logic_vector(bits_in-1 downto 0);
            d_out   : out   std_logic_vector(bits_out-1 downto 0)
        );
    end component sign_extend;

    -- Bus widths for the UUTs.
    constant bin1 : integer := 4;
    constant bout1 : integer := 6;

    constant bin2 : integer := 2;
    constant bout2 : integer := 5;

    constant bin3 : integer := 4;
    constant bout3 : integer := 4;

    -- Inputs
    signal d_in1 : std_logic_vector(bin1-1 downto 0) := (others => '0');
    signal d_in2 : std_logic_vector(bin2-1 downto 0) := (others => '0');
    signal d_in3 : std_logic_vector(bin3-1 downto 0) := (others => '0');

    -- Outputs
    signal d_out1 : std_logic_vector(bout1-1 downto 0);
    signal d_out2 : std_logic_vector(bout2-1 downto 0);
    signal d_out3 : std_logic_vector(bout3-1 downto 0);
begin
    -- Instantiate the UUTs.
    UUT1 : sign_extend
        generic map(bits_in => bin1, bits_out => bout1)
        port map(d_in => d_in1, d_out => d_out1);

    UUT2 : sign_extend
        generic map(bits_in => bin2, bits_out => bout2)
        port map(d_in => d_in2, d_out => d_out2);

    UUT3 : sign_extend
        generic map(bits_in => bin3, bits_out => bout3)
        port map(d_in => d_in3, d_out => d_out3);

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for 10ns.
        wait for 10ns;

        -- Set all possible input values.
        for i in 0 to 15 loop
            d_in1 <= conv_std_logic_vector(i, d_in1'LENGTH);
            d_in2 <= conv_std_logic_vector(i, d_in2'LENGTH);
            d_in3 <= conv_std_logic_vector(i, d_in3'LENGTH);
            wait for 10ns;
        end loop;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.25: TB_strobe_gen.vhd

```
--------------------------------------------------------------------------------
-- Test Bench for strobe_gen
-- Last Modified: 14 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TB_strobe_gen is
end TB_strobe_gen;

architecture behavioral of TB_strobe_gen is
    -- Component declaration for the Unit Under Test (UUT)
    component strobe_gen is
```

```
        port(
            clock           : in    std_logic;
            reset           : in    std_logic;
            enable      : in    std_logic;
            rate            : in    std_logic_vector(7 downto 0);  -- 1 less than desired divide
                ratio
            strobe_in   : in    std_logic;
            strobe      : out   std_logic
        );
    end component strobe_gen;

    -- Inputs
    signal clock : std_logic := '0';
    signal reset : std_logic := '0';
    signal enable : std_logic := '0';
    signal rate : std_logic_vector(7 downto 0) := (others => '0');
    signal strobe_in : std_logic := '0';

    -- Outputs
    signal strobe : std_logic;

    -- Clock Period
    constant clock_period : time := 10ns;
begin
    -- Instantitate the UUT.
    UUT : strobe_gen
        port map(clock => clock, reset => reset, enable => enable, rate => rate,
            strobe_in => strobe_in, strobe => strobe);

    -- Clock Process
    clock_process : process
    begin
        clock <= '0';
        wait for clock_period / 2;
        clock <= '1';
        wait for clock_period / 2;
    end process clock_process;

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for one clock period.
        wait for clock_period;

        -- Assert the reset.
        reset <= '1';
        wait for clock_period;

        -- Set the rate to 16.
        rate <= "00010000";

        -- Deassert the reset, assert the enable.
        reset <= '0';
        enable <= '1';
        wait for clock_period;

        -- As long as strobe_in is not asserted, nothing will happen.
        -- Test this by waiting 16 clocks.
        wait for 16 * clock_period;

        -- Now, assert strobe_in.  We'll just leave it that way for easier
        -- testing, but normal operation might include periods of time between
        -- strobe_in assertions.
        strobe_in <= '1';
        wait for 16 * clock_period;

        -- Do it again.
        wait for 16 * clock_period;

        -- Disable.
        enable <= '0';
        wait for clock_period;

        -- Set a new rate, assert enable.  Wait for eight strobes.
        rate <= "00000010";
        enable <= '1';
        wait for 16 * clock_period;

        -- Assert the reset without disabling.
        reset <= '1';
        wait for clock_period;

        -- Deassert the reset to continue operation.
        reset <= '0';
        wait for 4 * clock_period;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.26: TB_tx_chain.vhd

```
--------------------------------------------------------------------------------
-- Test Bench for tx_chain
-- Last Modified: 22 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2011 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
```

```
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TB_tx_chain is
end TB_tx_chain;

architecture behavioral of TB_tx_chain is
    -- Component declaration for the Unit Under Test (UUT)
    component tx_chain is
        port(
            clock                   : in    std_logic;
            reset                   : in    std_logic;
            enable              : in    std_logic;
            interp_rate         : in    std_logic_vector(7 downto 0);
            sample_strobe       : in    std_logic;
            interpolator_strobe : in    std_logic;
            freq                    : in    std_logic_vector(31 downto 0);
            i_in                    : in    std_logic_vector(15 downto 0);
            q_in                    : in    std_logic_vector(15 downto 0);
            i_out                   : out   std_logic_vector(15 downto 0);
            q_out                   : out   std_logic_vector(15 downto 0)
        );
    end component tx_chain;

    -- Inputs
    signal clock : std_logic := '0';
    signal reset : std_logic := '0';
    signal enable : std_logic := '0';
    signal interp_rate : std_logic_vector(7 downto 0) := (others => '0');
    signal sample_strobe : std_logic := '0';
    signal interpolator_strobe : std_logic := '0';
    signal freq : std_logic_vector(31 downto 0) := (others => '0');
    signal i_in : std_logic_vector(15 downto 0) := (others => '0');
    signal q_in : std_logic_vector(15 downto 0) := (others => '0');

    -- Outputs
    signal i_out : std_logic_vector(15 downto 0);
    signal q_out : std_logic_vector(15 downto 0);

    -- Clock Period
    constant clock_period : time := 10ns;
begin
    -- Instantiate the UUT.
    UUT : tx_chain
        port map(clock => clock, reset => reset, enable => enable,
            interp_rate => interp_rate, sample_strobe => sample_strobe,
            interpolator_strobe => interpolator_strobe, freq => freq,
            i_in => i_in, q_in => q_in, i_out => i_out, q_out => q_out);

    -- Clock Process
    clock_process : process
    begin
        clock <= '0';
        wait for clock_period / 2;
        clock <= '1';
        wait for clock_period / 2;
    end process clock_process;

    -- Strobe Process
    strobe_proc : process
    begin
        wait for 7 * clock_period;
        interpolator_strobe <= '1';
        wait for clock_period;
        interpolator_strobe <= '0';
    end process strobe_proc;

    -- We want a sample on every clock, so just keep sample_strobe enabled.
    sample_strobe <= '1';

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for one clock cycle.
        wait for clock_period;

        -- Assert the reset.
        reset <= '1';
        wait for clock_period;

        -- Enable the filter.
        reset <= '0';
        enable <= '1';

        -- Set the rate to 7.  This is actually a rate of 32 because of the
        -- codecs that we use.  The formula is (rate+1)*4.
        interp_rate <= conv_std_logic_vector(7, interp_rate'LENGTH);

        -- Test values of 1, max, min, and -1.  Q will go in reverse.
        i_in <= "0000000000000001";
        q_in <= "1111111111111111";
        wait for 100 * clock_period;
        i_in <= "0111111111111111";
        q_in <= "1000000000000000";
        wait for 100 * clock_period;
        i_in <= "1000000000000000";
        q_in <= "0111111111111111";
        wait for 100 * clock_period;
        i_in <= "1111111111111111";
        q_in <= "0000000000000001";
        wait for 100 * clock_period;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Listing C.27: TB_usrp_mux.vhd

```
--------------------------------------------------------------------------------
```

```
-- Test Bench for usrp_mux
-- Last Modified: 14 February 2011
-- VHDL Author: Steve Olivieri
--
-- Copyright (C) 2010 Worcester Polytechnic Institute
--
--  This program is free software; you can redistribute it and/or modify
--  it under the terms of the GNU General Public License as published by
--  the Free Software Foundation; either version 2 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program; if not, write to the Free Software
--  Foundation, Inc., 51 Franklin Street, Boston, MA  02110-1301  USA
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TB_usrp_mux is
end TB_usrp_mux;

architecture behavioral of TB_usrp_mux is
    -- Component declaration for the Unit Under Test (UUT)
    component usrp_mux is
        port(
            dac_mux     : in    std_logic_vector(3 downto 0);
            i_out_0     : in    std_logic_vector(15 downto 0);
            q_out_0     : in    std_logic_vector(15 downto 0);
            i_out_1     : in    std_logic_vector(15 downto 0);
            q_out_1     : in    std_logic_vector(15 downto 0);
            d_out           : out   std_logic_vector(15 downto 0)
        );
    end component usrp_mux;

    -- Inputs
    signal dac_mux : std_logic_vector(3 downto 0) := (others => '0');
    signal i_out_0 : std_logic_vector(15 downto 0) := (others => '0');
    signal q_out_0 : std_logic_vector(15 downto 0) := (others => '0');
    signal i_out_1 : std_logic_vector(15 downto 0) := (others => '0');
    signal q_out_1 : std_logic_vector(15 downto 0) := (others => '0');

    -- Outputs
    signal d_out : std_logic_vector(15 downto 0);
begin
    -- Instantiate the UUT.
    UUT : usrp_mux
        port map(dac_mux => dac_mux, i_out_0 => i_out_0, q_out_0 => q_out_0,
            i_out_1 => i_out_1, q_out_1 => q_out_1, d_out => d_out);

    -- Stimulus Process
    stim_proc : process
    begin
        -- Do nothing for 10ns.
        wait for 10ns;

        -- Set data inputs.
        i_out_0 <= "0000000000000001";
        q_out_0 <= "0000000000000010";
        i_out_1 <= "0000000000000011";
        q_out_1 <= "0000000000000100";

        -- Provide every possible combination for dac_mux.
        for i in 0 to 15 loop
            dac_mux <= conv_std_logic_vector(i, dac_mux'LENGTH);
            wait for 10ns;
        end loop;

        -- End test.
        wait;
    end process stim_proc;
end behavioral;
```

Bibliography

[1] *Altera Cyclone FPGA Family Data Sheet v1.5*, 2008. Available from http://www.altera.com/literature/hb/cyc/cyc_c5v1_01.pdf.

[2] *Spartan-3 Generation FPGA User Guide v1.7*, 2010. Available from http://www.xilinx.com/support/documentation/user_guides/ug331.pdf.

[3] *Xilinx Spartan-3A FPGA Family: Data Sheet v2.0*, 2010. Available from http://www.xilinx.com/support/documentation/data_sheets/ds529.pdf.

[4] AAUSAT II CubeSat. Available from http://aausatii.space.aau.dk/eng/, April 2011.

[5] CanX-1 CubeSat. Available from http://www.utias-sfl.net/nanosatellites/CanX1/, April 2011.

[6] COSMIAC CubeSat FPGA Board. Available from http://www.cosmiac.org/formfactor.html, April 2011.

[7] COSMIAC CubeSat SDR Photograph. Available from http://www.cosmiac.org/images/CCRBwSDR.JPG, April 2011.

[8] CubeSat Photographs. Available from http://www.cubesat.org/index.php/media/pictures, April 2011.

[9] GeneSat-1 CubeSat. Available from https://directory.eoportal.org/presentations/129/13064.html, April 2011.

[10] GNU Radio. Available from http://gnuradio.org/redmine/wiki/gnuradio, April 2011.

[11] Intel Core i7-990X Processor Extreme Edition. Available from http://ark.intel.com/Product.aspx?id=52585, April 2011.

[12] Lyrtech Small Form Factor SDR Development Platforms. Available from http://www.lyrtech.com/products/sff_sdr_development_platforms.php, April 2011.

[13] Moore's Law Timeline. Available from http://download.intel.com/pressroom/kits/events/moores_law_40th/MLTimeline.pdf, April 2011.

[14] Mouser — Lyrtech SFF SDR Development Platform. Availabe from http://www.mouser.com/Lyrtech/, April 2011.

[15] SpaceWire. Available from http://www.spacewire.esa.int/content/Home/HomeIntro.php, April 2011.

[16] Universal Software Radio Peripheral. Available from http://www.ettus.com/products, April 2011.

[17] University of Tokyo CubeSat Mission. Available from http://www.space.t.u-tokyo.ac.jp/cubesat/mission/index-e.html, April 2011.

[18] USRP and Daughtercard Schematics. Available from http://code.ettus.com/redmine/ettus/projects/public/documents, April 2011.

[19] Vulcan Wireless SDR Products. Available from http://www.vulcanwireless.com/products/, April 2011.

[20] Wireless Open-Access Research Platform. Available from http://warp.rice.edu/index.php, April 2011.

[21] *Xilinx Platform Specification Format Reference Manual, EDK 13.1*, 2011. Available from http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/psf_rm.pdf.

[22] Xilinx USB 2.0 Device. Available from `http://www.xilinx.com/products/intellectual-property/xps_usb2_device.htm`, April 2011.

[23] Josh Blum. GNU Radio Companion. Available from `http://www.joshknows.com/grc`, April 2011.

[24] Vanu Bose. On the Softer Side: A Software Driven Approach to SDR Design. *COTS Journal*, January, 2004. Available from `http://www.cotsjournalonline.com/articles/view/100056`.

[25] Elaine Caday-Eames. Small Box, Big Potential. Available from `http://www.boeing.com/news/frontiers/archive/2006/october/i_ids02.pdf`, April 2011.

[26] California Polytechnic State University. *CubeSat Design Specification Rev. 12*, 2009. Available from `http://www.cubesat.org/images/developers/cds_rev12.pdf`.

[27] Chen Chang, John Wawrzyned, and Robert W. Brodersen. BEE2: A High-End Reconfigurable Computing System. *IEEE Design & Test of Computers*, March-April:115–125, 2005. Available from `http://bee2.eecs.berkeley.edu/papers/BEE2_chang_ieee.pdf`.

[28] Leonard David. Cubesats: Tiny Spacecraft, Huge Payoffs. Available from `http://www.space.com/308-cubesats-tiny-spacecraft-huge-payoffs.html`, 2004.

[29] Wireless Innovation Forum. What is Software Defined Radio. Available from `http://data.memberclicks.com/site/sdf/SoftwareDefinedRadio.pdf`, April 2011.

[30] Firas Abbas Hamza. *The USRP Under 1.5x Magnifying Lens! v1.0*, 2008. Available from `http://gnuradio.org/redmine/attachments/129/USRP_Documentation.pdf`.

[31] Bryan Klofas. A Survey of CubeSat Communication Subsystems. Presented at the 2008 CubeSat Developers Workshop at California Polytechnic State Institute, April 2008.

[32] Michael Joseph Leferman. Rapid Prototyping for Software Define Radio Experimentation. Master's thesis, Worcester Polytechnic Institute, February 2010. Available from http://www.wpi.edu/Pubs/ETD/Available/etd-012010-150837/.

[33] Jim Lyke, Don Fronterhouse, Scott Cannon, Denise Lanza, and Wheaton Tony Byers. Space Plug-and-Play Avionics. In *3rd Responsive Space Conference*, Los Angeles, CA, April 2005. AIAA.

[34] Kevin Lynaugh and Matt Davis. Software Defined Radios for Small Satellites. Presented at the ReSpace/MAPLD 2010 Conference in Albuquerque, New Mexico, November 2010.

[35] Christopher J. McNutt, Robert Vick, Harry Whiting, and Jim Lyke. Modular Nanosatellites—(Plug-and-Play) PnP CubeSat. In *7th Responsive Space Conference*, Los Angeles, CA, April 2009. AIAA.

[36] G. J. Minden, J. B. Evans, L. Searl, D. DePardo, V. R. Petty, R. Raijbanshi, T. Newman, Q. Chen, F. Weidling, J. Guffey, D. Datla, B. Barker, M. Peck, B. Cordill, A. M. Wyglinski, and A. Agah. KUAR: A Flexible Software-Defined Radio Development Platform. In *2nd Dynamic Spectrum Access Networks Symposium*, pages 428–439. IEEE, April 2007. Available from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4221524.

[37] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), 1965. Available from http://download.intel.com/museum/Moores_Law/Articles-Press_Releases/Gordon_Moore_1965_Article.pdf.

[38] Steven Olivieri, Alexander M. Wyglinski, L. Howard Pollard, Jim Aarestad, and Craig Kief. Responsive Satellite Communications via FPGA-Based Software-Defined Radio for SPA-U Compatible Platforms. Presented at the ReSpace/MAPLD 2010 Conference in Albuquerque, New Mexico, November 2010.

[39] Joseph Palmer. The Firehose Adaptive Software Defined Radio. Presented at the ReSpace/MAPLD 2010 Conference in Albuquerque, New Mexico, November 2010.

[40] Kalen Watermeyer. Design of a hardware platform for narrow-band Software Defined Radio applications. Master's thesis, University of Cape Town, January 2007. Available from `http://rrsg.uct.ac.za/theses/msc_theses/kwatermeyer_thesis.pdf`.

[41] Alexander M. Wyglinski, Maziar Nekovee, and Thomas Hou. *Cognitive Radio Communications and Networks: Principles and Practice*. Academic Press, 2009.

[42] Gerald Youngblood. A Software-Defined Radio for the Masses, Part 1. *QEX*, Jul/Aug:13–21, 2002. Available from `http://www.arrl.org/files/file/Technology/tis/info/pdf/020708qex013.pdf`.