

2011-04-27

Modular FPGA-Based Software Defined Radio for CubeSats

Steven J. Olivieri
Worcester Polytechnic Institute

Follow this and additional works at: <https://digitalcommons.wpi.edu/etd-theses>

Repository Citation

Olivieri, Steven J., "Modular FPGA-Based Software Defined Radio for CubeSats" (2011). *Masters Theses (All Theses, All Years)*. 375.
<https://digitalcommons.wpi.edu/etd-theses/375>

This thesis is brought to you for free and open access by Digital WPI. It has been accepted for inclusion in Masters Theses (All Theses, All Years) by an authorized administrator of Digital WPI. For more information, please contact wpi-etd@wpi.edu.

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	vi
List of Tables	vii
List of Listings	x
1 Introduction	1
1.1 Research Motivation	1
1.2 Current State of the Art	3
1.3 Research Objectives	4
1.4 Thesis Contributions	5
1.5 Thesis Organization	5
2 Software-Defined Radio and CubeSats	6
2.1 Software-Defined Radio	6
2.2 Field-Programmable Gate Arrays	9
2.3 GNU Radio and the USRP	11
2.3.1 GNU Radio	12
2.3.2 USRP	14
2.4 CubeSats	19
2.4.1 Space Plug-and-Play Avionics (SPA)	20
2.5 Chapter Summary	22
3 COSMIAC CubeSat SDR System	23
3.1 Hardware	23
3.2 Software	27
3.3 Chapter Summary	30
4 Proposed USRP Variant	31
4.1 Proposed USRP Firmware	31
4.2 A New FPGA	32
4.3 A New Language	33
4.3.1 Header Files	34
4.3.2 Ternary Operators	37

4.3.3	FIFO	37
4.4	Synthesis Results	40
4.5	Chapter Summary	42
5	Conclusions	43
5.1	Summary and Accomplishments	43
5.2	Future Work	44
A	USRP Code	46
B	FPGA Board Description (XBD)	160
C	USRP Test Benches	171
	Bibliography	231

List of Figures

1.1	A FPGA swapping modulation schemes.	2
2.1	Block diagram of a typical software-defined radio.	7
2.2	Illustration of a FPGA fabric.	10
2.3	Illustration of a Spartan3A slice.	11
2.4	Data flow through the USRP Platform.	12
2.5	GNU Radio Companion WBFM Receiver.	13
2.6	A USRP and RF daughtercards.	15
2.7	Block diagram of the USRP.	16
2.8	Data flow through the USRP.	17
2.9	Tx data flow through the USRP.	18
2.10	Rx data flow through the USRP.	18
2.11	A photograph of three CubeSats.	20
2.12	Organization of an XTEDs.	21
3.1	Block diagram of the COSMIAC FPGA board.	25
3.2	A photograph of the COSMIAC FPGA board.	26
3.3	A photograph of the COSMIAC CubeSat SDR system.	26
3.4	Block diagram of the COSMIAC RF daughtercard.	27
3.5	Block diagram of the proposed CubeSat SDR firmware.	28
4.1	Simulation of a read request FIFO.	39
4.2	Simulation of a read acknowledgement FIFO.	39

List of Tables

2.1	Comparison of microelectronic platforms for SDR.	8
2.2	Some examples of SDR implementations.	9
3.1	USB endpoint configuration for the CubeSat SDR.	30
4.1	Comparison of Cyclone EP1C12 and Spartan3A-1400.	32
4.2	Hardware utilization for USRP modules.	41
4.3	Hardware utilization for the USRP firmware.	42

List of Listings

4.1	4-to-1 Multiplexer in Verilog.	34
4.2	4-to-1 Multiplexer in VHDL.	35
4.3	Verilog configuration code (common_config.vh).	36
4.4	VHDL configuration code (common_config.vhd).	36
4.5	Example of a Verilog ternary operator.	37
4.6	VHDL replacement for ternary operators.	38
A.1	acc.vhd	46
A.2	adc_interface.vhd	47
A.3	adc_mux.vhd	52
A.4	atr_delay.vhd	54
A.5	bidir_reg.vhd	56
A.6	bustri.vhd	57
A.7	cic_decim.vhd	58
A.8	cic_dec_shifter.vhd	61
A.9	cic_interp.vhd	63
A.10	cic_int_shifter.vhd	66
A.11	clk_divider.vhd	68
A.12	coeff_rom.vhd	70
A.13	common_config.vhd	71
A.14	common_config_1rxhb_1tx.vhd	72
A.15	common_config_2rxhb_0tx.vhd	73
A.16	common_config_2rxhb_2tx.vhd	75
A.17	common_config_2rx_0tx.vhd	76
A.18	common_config_4rx_0tx.vhd	77
A.19	cordic.vhd	78
A.20	cordic_stage.vhd	81
A.21	fifo.vhd	82
A.22	fifo_4k_18.vhd	85
A.23	fpga_regs_common.vhd	86
A.24	fpga_regs_standard.vhd	89
A.25	funcs.vhd	94
A.26	halfband_decim.vhd	95
A.27	io_pins.vhd	102

A.28	master_control.vhd	104
A.29	mult.vhd	112
A.30	phase_acc.vhd	113
A.31	ram16.vhd	115
A.32	ram16_2sum.vhd	116
A.33	rsi.vhd	118
A.34	rx_buffer.vhd	120
A.35	rx_chain.vhd	126
A.36	rx_dcoffset.vhd	130
A.37	serial_io.vhd	132
A.38	setting_reg.vhd	135
A.39	sign_extend.vhd	136
A.40	strobe_gen.vhd	137
A.41	tx_buffer.vhd	138
A.42	tx_chain.vhd	143
A.43	usrp_mux.vhd	144
A.44	usrp_std.vhd	146
B.1	COSMIAC FPGA Board Description (XBD)	160
C.1	TB_TB_acc.vhd	171
C.2	TB_adc_mux.vhd	173
C.3	TB_atr_delay.vhd	175
C.4	TB_bidir_reg.vhd	177
C.5	TB_bustri.vhd	179
C.6	TB_cic_decim.vhd	180
C.7	TB_cic_dec_shifter.vhd	183
C.8	TB_cic_interp.vhd	184
C.9	TB_cic_int_shifter.vhd	186
C.10	TB_clk_divider.vhd	188
C.11	TB_coeff_rom.vhd	189
C.12	TB_cordic.vhd	191
C.13	TB_cordic_stage.vhd	193
C.14	TB_fifo.vhd	195
C.15	TB_io_pins.vhd	199
C.16	TB_mult.vhd	202
C.17	TB_phase_acc.vhd	204
C.18	TB_ram16.vhd	207
C.19	TB_ram16_2sum.vhd	208
C.20	TB_rssi.vhd	211
C.21	TB_rx_dcoffset.vhd	213
C.22	TB_serial_io.vhd	215
C.23	TB_setting_reg.vhd	219
C.24	TB_sign_extend.vhd	222
C.25	TB_strobe_gen.vhd	224

C.26 TB_tx_chain.vhd	226
C.27 TB_usrp_mux.vhd	228

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 primarily 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.

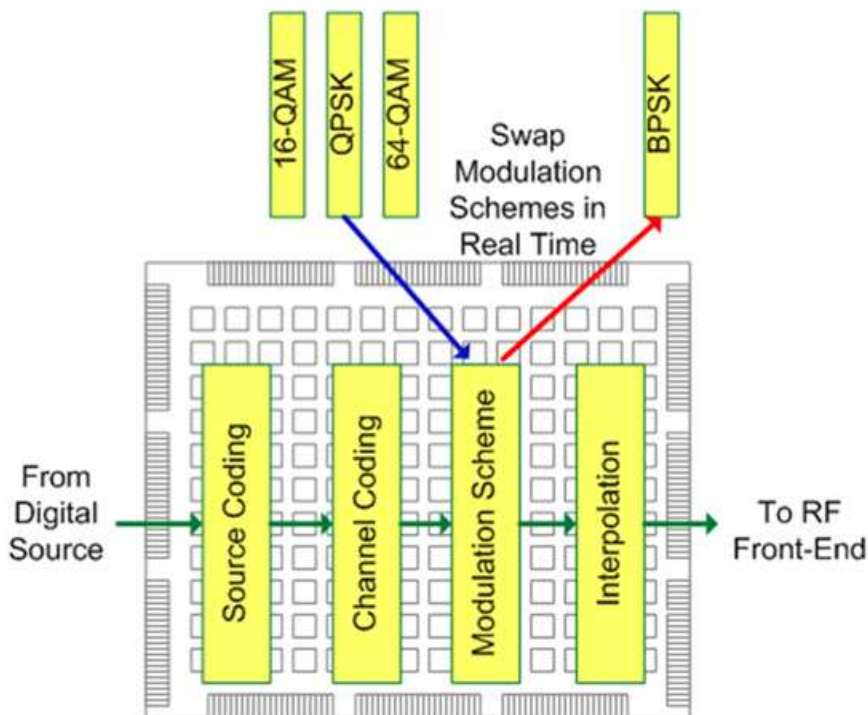


Figure 1.1: A FPGA swapping modulation schemes. The different modulation schemes are stored in memory or in the FPGA logic elements and connected as needed. While not shown here, the other components in a FPGA-based SDR could also be reconfigurable.

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 VH-SIC 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

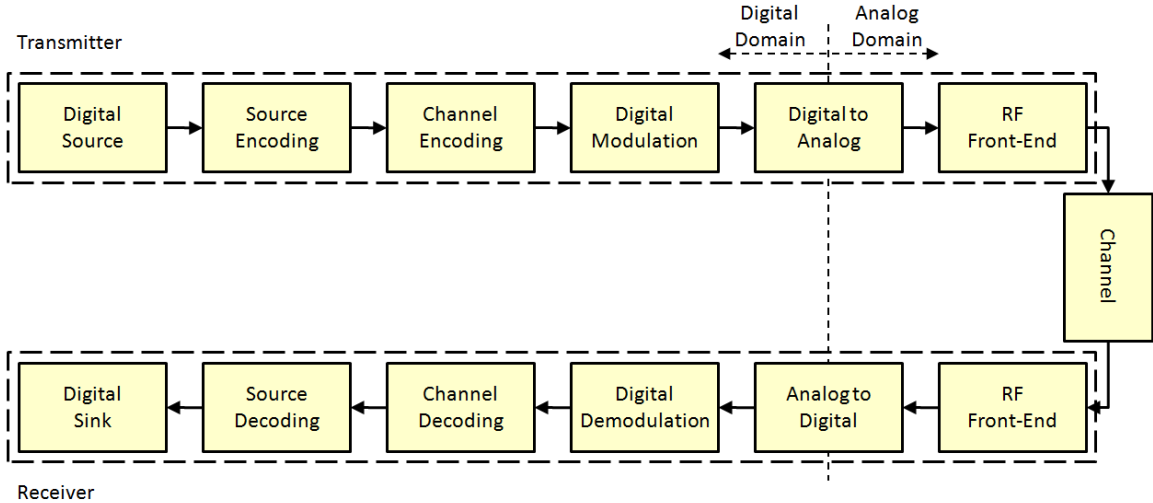


Figure 2.1: Block diagram of a typical software-defined radio. Everything in the digital domain is performed in software while everything in the analog domain remains hardware.

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 addition, 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.

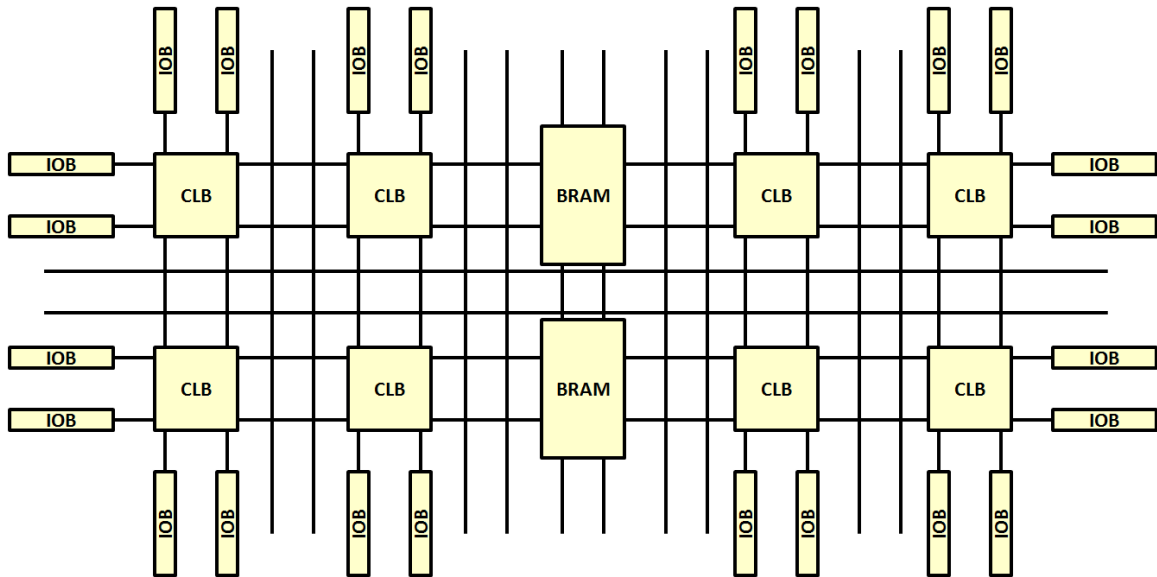


Figure 2.2: Illustration of a FPGA fabric. Each of the intersections can be programmed to connect different IOB, BRAM, and CLB elements together.

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 16×1 memory.

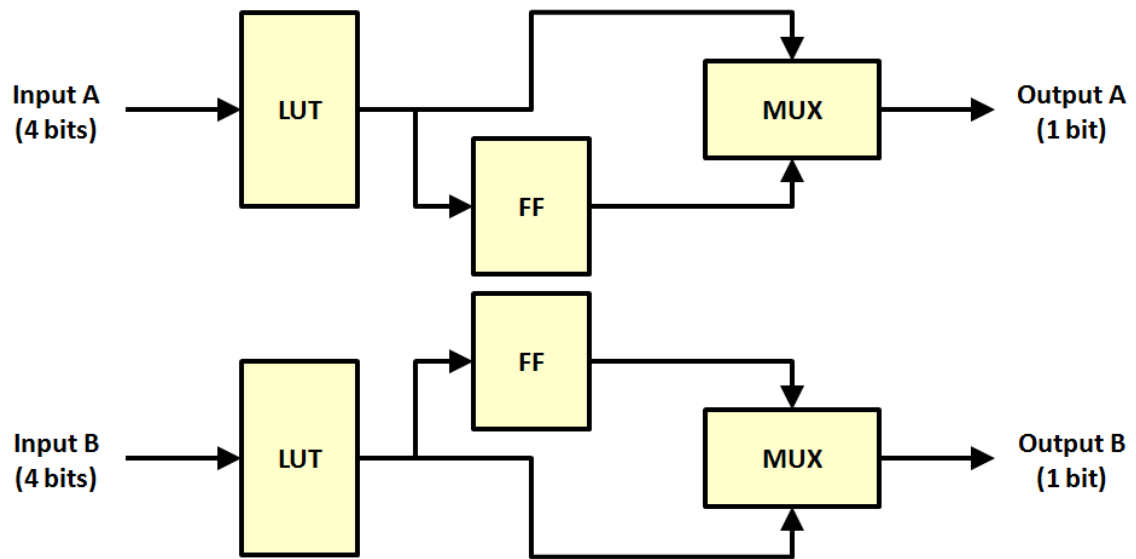


Figure 2.3: Illustration of a Spartan3A slice. Slices contain additional logic gates and signals that enable faster data paths, such as XOR gates and bypass signals.

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.

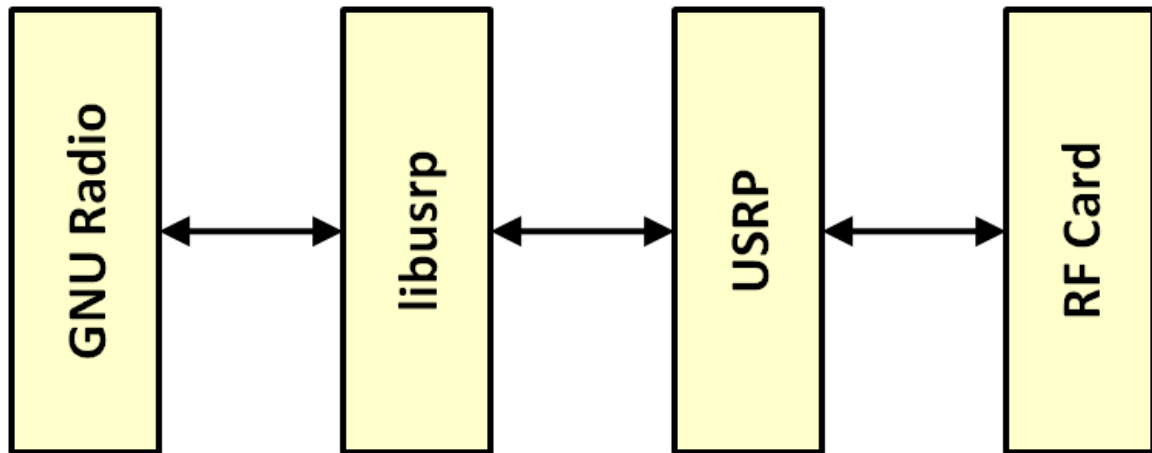


Figure 2.4: Data flow through the USRP platform. GNU Radio and `libusrp` reside on a general-purpose computer, while the USRP and its RF daughtercards are custom radio hardware.

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].

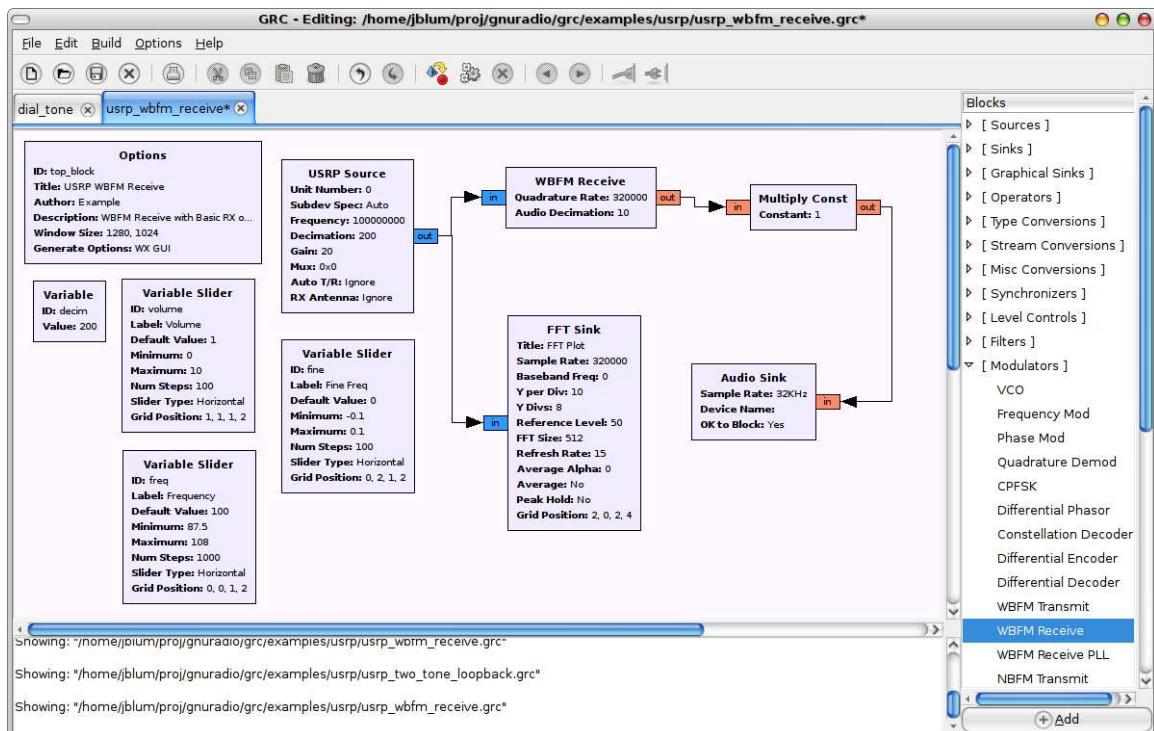


Figure 2.5: A GNU Radio Companion flow graph for a wideband FM receiver. The arrows indicate the flow of data from the USRP to the sound card [23].

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

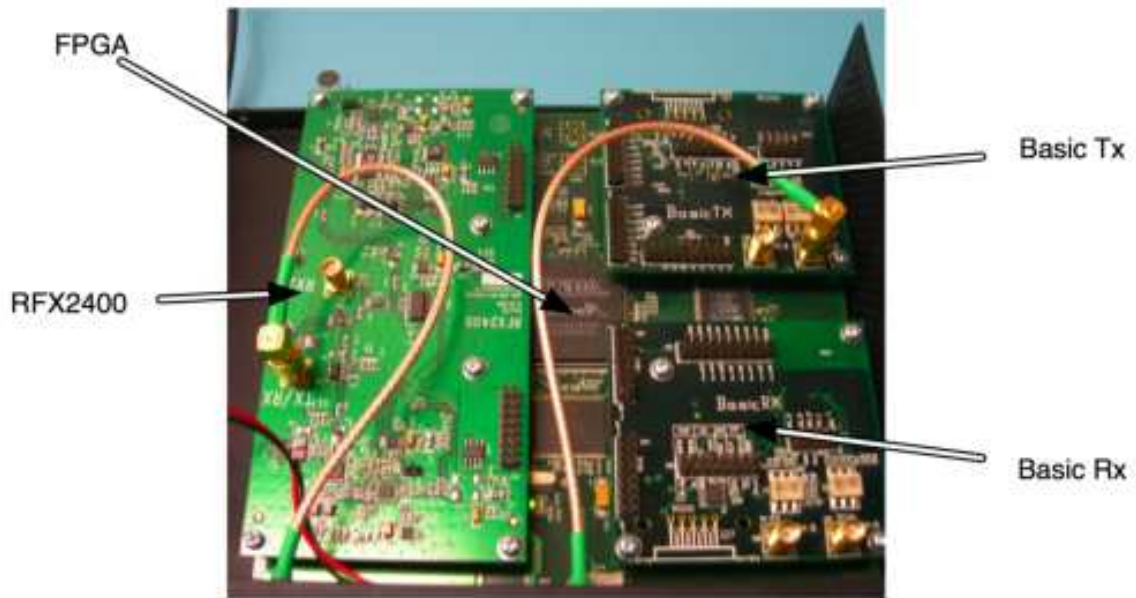


Figure 2.6: A USRP and RF daughtercards. The RFX2400 card contains one Tx RF frontend and one Rx RF frontend. The USRP supports up to four total RF frontends simultaneously with a maximum of two Tx and four Rx [32].

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

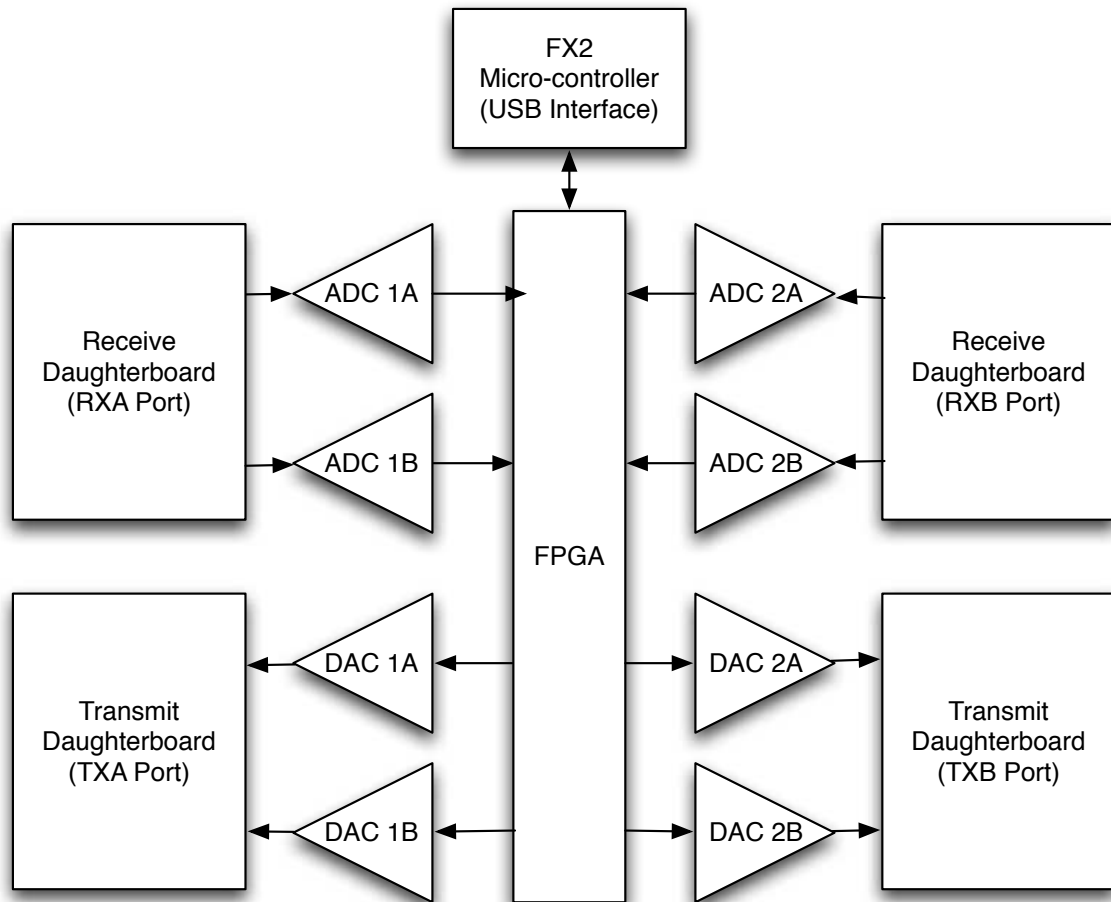


Figure 2.7: Block diagram of the USRP. This particular USRP has two Tx boards and two Rx boards, but the system could support four Rx boards instead [41].

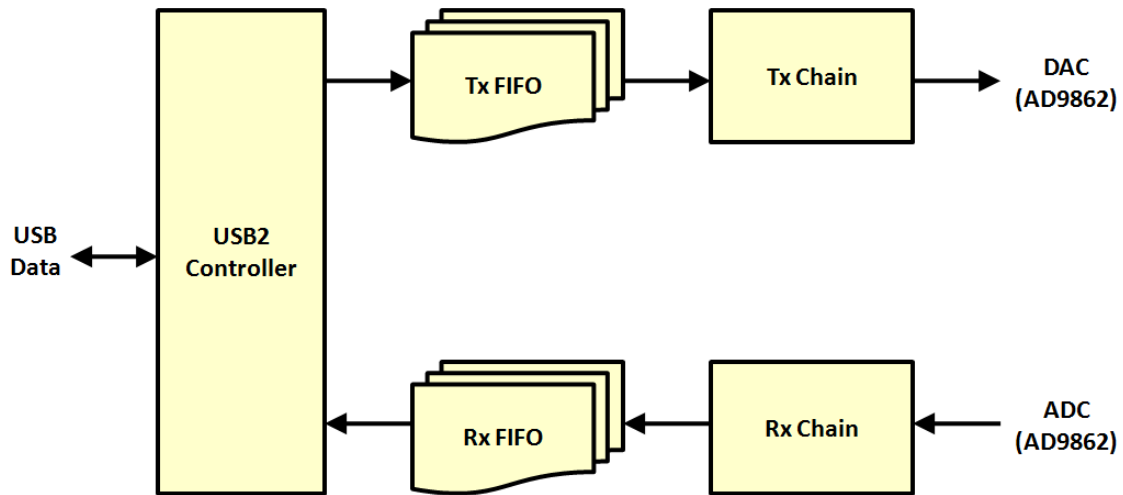


Figure 2.8: Data flow through the USRP. GNU Radio passes data to and from the USRP via the USB2 controller and the USRP handles the remaining signal processing.

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

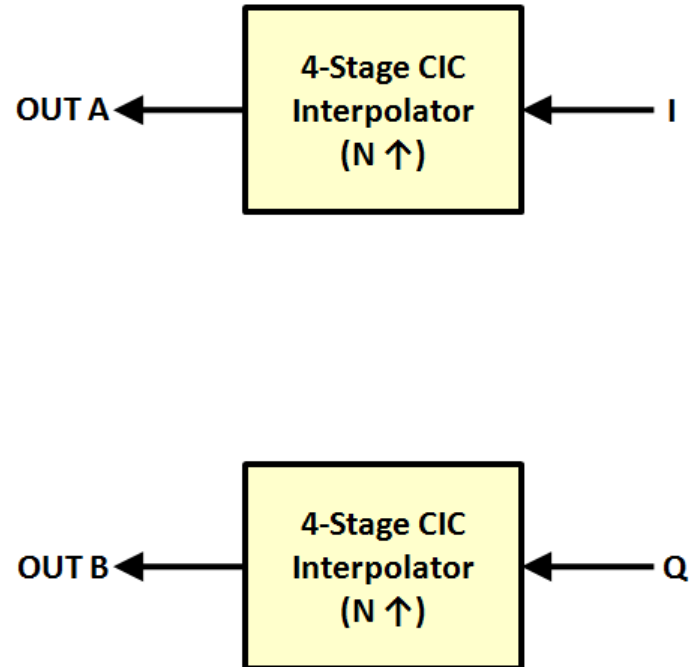


Figure 2.9: Tx data flow through the USRP. The remainder of the digital upconversion process is handled by the AD9862.

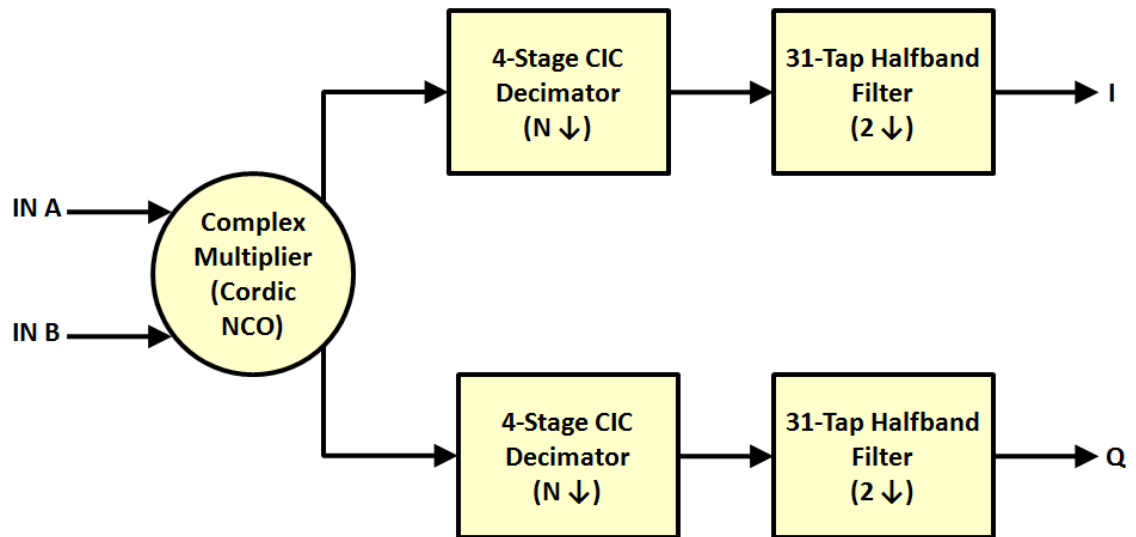


Figure 2.10: Rx data flow through the USRP. The AD9862 does not provide a Cordic NCO on the receive path, so the FPGA does the conversion from IF band to baseband internally.

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 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 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.

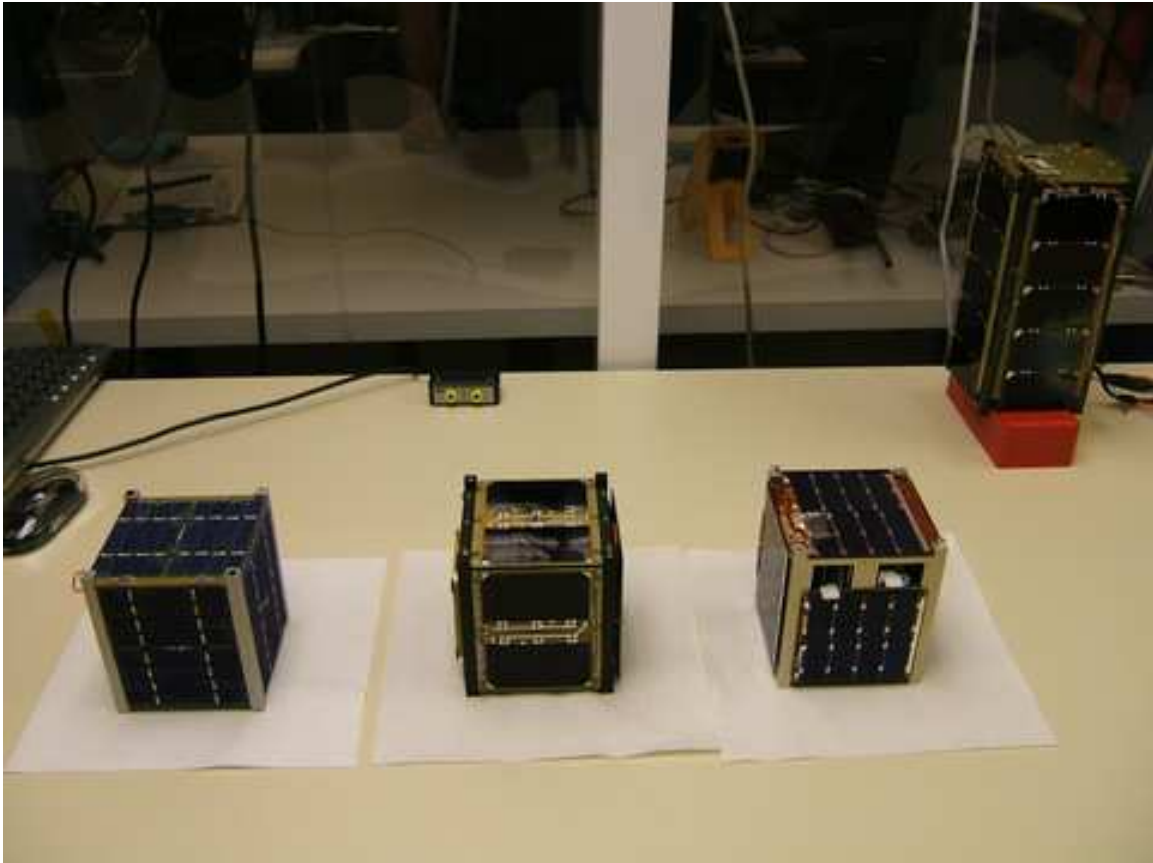


Figure 2.11: A photograph of three CubeSats at California Polytechnic State University. These satellites are ready for integration into the launcher system, called P-POD [8].

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].

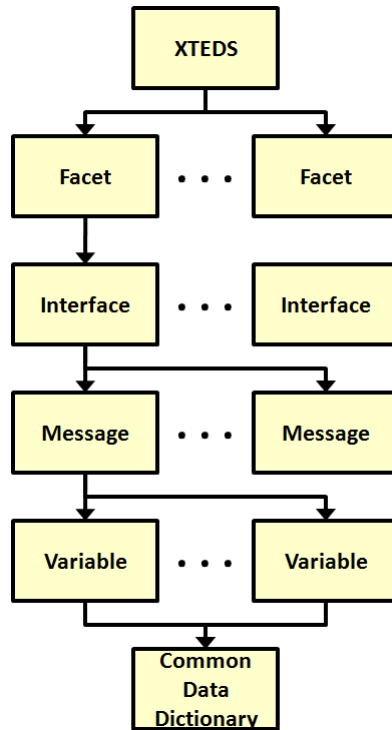


Figure 2.12: Organization of an XTEDs. Like any other XML document, an XTEDs combines user-defined properties and values to describe something. XTEDs are composed of “atoms” drawn from a common data dictionary (CCD). These atoms compose variables, variables compose message, and so on to form an XTEDs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

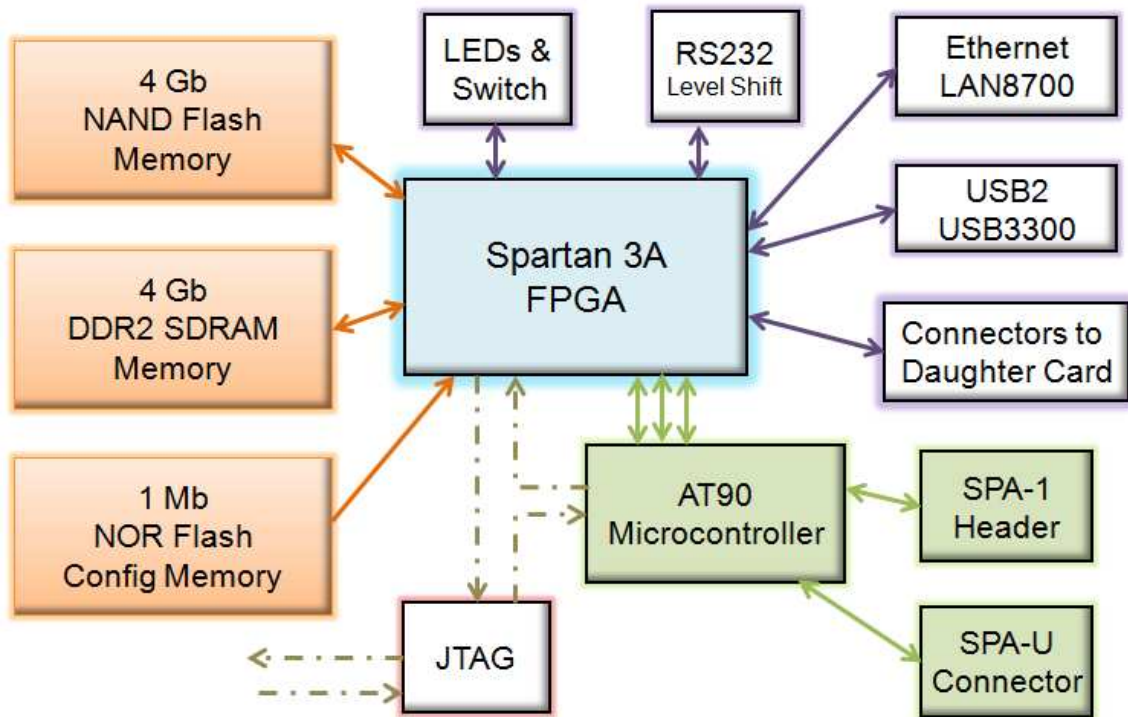


Figure 3.1: Block diagram of the COSMIAC FPGA board. Note the lack of AD9862 processors, which reside instead on the RF daughtercards (see Figure 3.4. This design choice enables CubeSat engineers to design non-SDR systems with the same primary FPGA board [38].

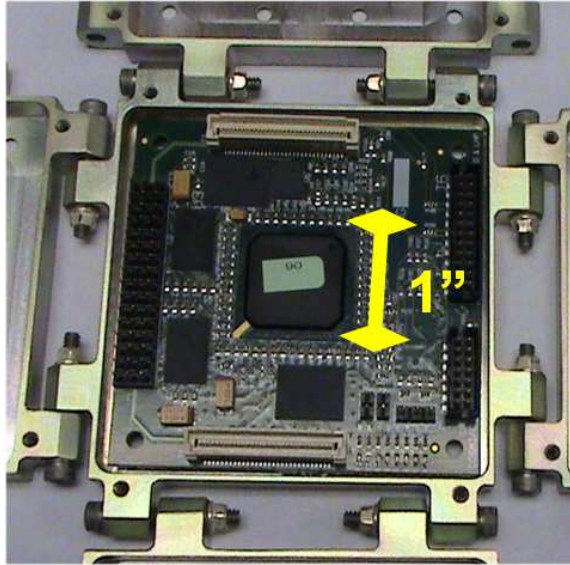


Figure 3.2: A photograph of the COSMIAC FPGA board. The board sits on one side of the CubeSat enclosure and four others flank it. The RF card attaches to the two white connectors on the top and bottom of the board [38].

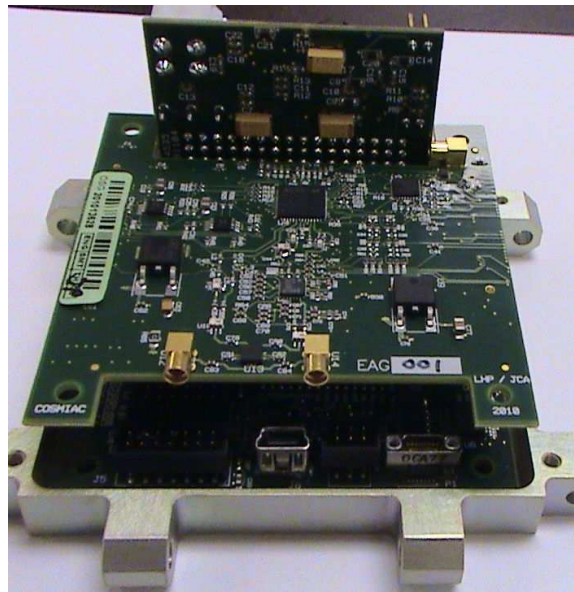


Figure 3.3: A photograph of the COSMIAC CubeSat SDR system. The bottom board is the FPGA board while the one on top is the RF daughtercard. The perpendicular board in the back provides power to the system. All three boards fit into a 1U CubeSat [7].

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].

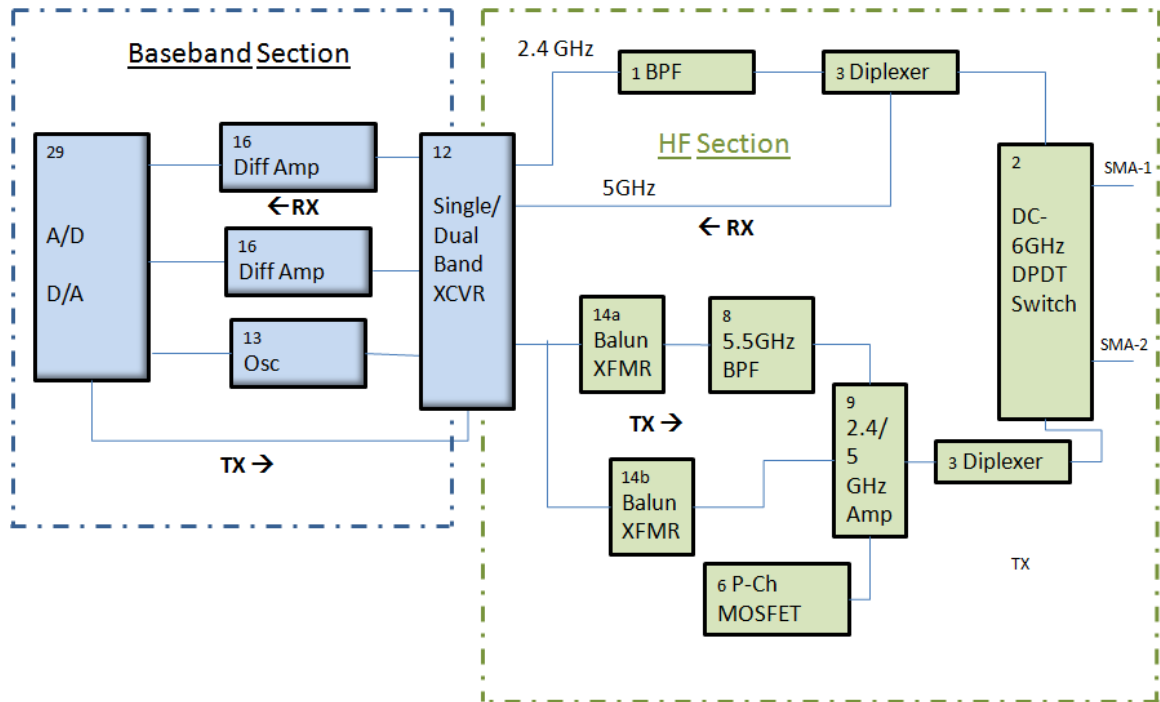


Figure 3.4: Block diagram of the COSMIAC RF daughtercard. The AD9862 mixed-signal processors were moved to the daughtercard to make room for other devices on the primary FPGA board [38].

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

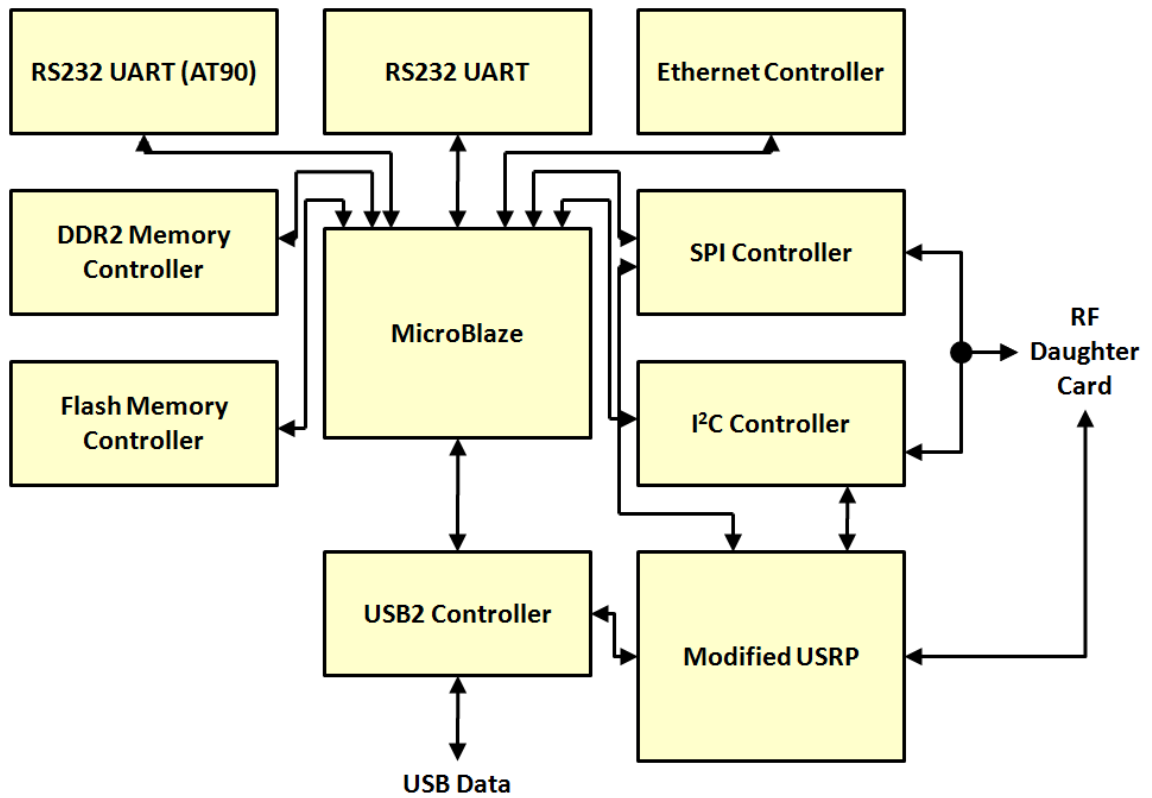


Figure 3.5: Block diagram of the proposed CubeSat SDR firmware. The MicroBlaze processor acts as the central control for all of the various communications blocks.

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 a 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

¹A 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 `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.

```

1 module mux4to1(input wire d1, input wire d2, input wire d3, input wire
   d4,
2   input wire [1:0] s, output reg q);
3
4   always @(d1, d2, d3, d4, s)
5   begin
6       case(s)
7           0: q = d1;
8           1: q = d2;
9           2: q = d3;
10          3: q = d4;
11       endcase
12   end
13 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.

```

1  entity mux4to1 is
2      port(d1 : in std_logic; d2 : in std_logic; d3 : in std_logic;
3          d4 : in std_logic; s : in std_logic_vector(1 downto 0);
4          q : out std_logic);
5  end mux4to1;
6
7  architecture Behavioral of mux4to1 is
8  begin
9      process(d1, d2, d3, d4, s)
10     begin
11         case s is
12             when "00" => q <= d1;
13             when "01" => q <= d2;
14             when "10" => q <= d3;
15             when others => q <= d4;
16         end case;
17     end process;
18 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).

```

1  'ifdef TX_ON
2
3  'ifdef TX_SINGLE
4    'define TX_EN_0
5    'define TX_CAP_NCHAN 3'd1
6  'endif
7
8  'ifdef TX_DUAL
9    'define TX_EN_0
10   'define TX_EN_1
11   'define TX_CAP_NCHAN 3'd2
12  'endif
13 '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).

```

1  constant TX_EN : boolean := TX_ON;
2  constant TX_EN_0 : boolean := (TX_EN and (TX_SINGLE or TX_DUAL));
3  constant TX_EN_1 : boolean := (TX_EN and TX_DUAL);
4  constant TX_CAP_NCHAN : std_logic_vector(2 downto 0) := '0'
5
6                                     & conv_std_logic(TX_EN and
7                                     TX_DUAL)
6                                     & conv_std_logic(TX_EN and
7                                     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.

```

1 q = mux[2] ? (mux[1] ? (mux[0] ? d3
2                               : d2)
3                               : (mux[0] ? d1
4                               : d0))
5       : 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.

```

1  process(mux, d0, d1, d2, d3)
2  begin
3      if(mux(2) = '1') then
4          if(mux(1) = '1') then
5              if(mux(0) = '1') then
6                  q <= d3;
7              else
8                  q <= d2;
9              end if;
10         else
11             if(mux(0) = '1') then
12                 q <= d1;
13             else
14                 q <= d0;
15             end if;
16         end if;
17     else
18         q <= '0';
19     end if;
20 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.

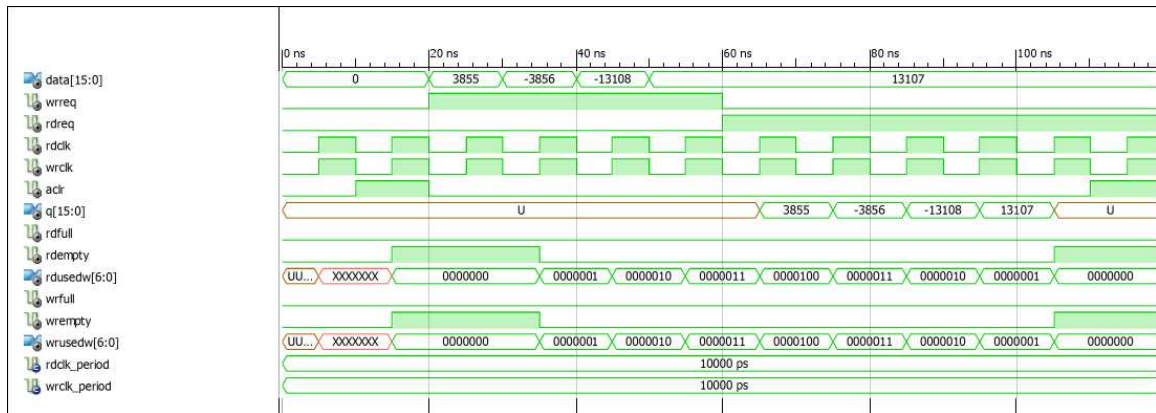


Figure 4.1: Simulation of a read request FIFO. Note that the data on bus 'q' is not available until one clock after the 'rdreq' signal is pulled high.

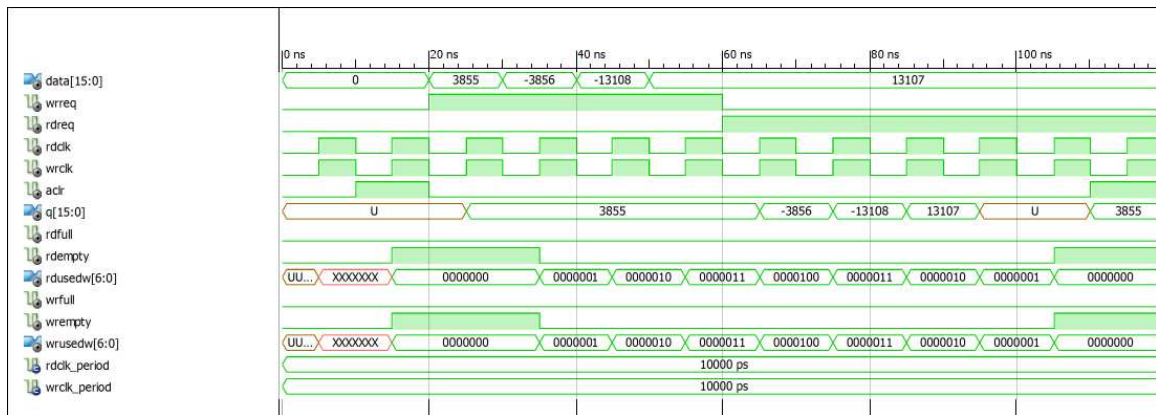


Figure 4.2: Simulation of a read acknowledgement FIFO. Note that the data bus 'q' is available one clock after the first write and that this output remains until one clock after the 'rdreq' signal is pulled high.

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	–
<i>cic_interp</i>	383	284	437	0	192.569	383	285	437	0	192.569
<i>tx_chain</i>	766	565	842	0	192.569	766	566	842	0	192.569
<i>tx_buffer</i>	144	88	77	5	198.275	144	87	76	5	198.275
<i>rx_buffer</i>	191	191	287	5	120.049	191	171	301	5	120.049
coeff_rom	0	8	14	0	–	0	8	14	0	–
<i>ram16_2sum</i>	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
<i>halfband_decim</i>	118	169	325	0	138.383	118	150	291	0	139.367
<i>cic_dec_shifter</i>	0	152	263	0	–	0	151	262	0	–
<i>cic_decim</i>	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
<i>rx_chain</i>	2087	1464	2577	0	138.383	2087	1455	2475	0	139.367
<i>rx_dcoffset</i>	32	56	106	0	84.72	32	56	106	0	74.384
rss_i	52	55	104	0	143.981	52	55	104	0	143.981
<i>adc_interface</i>	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)

```

1 -----
2 -- Accumulator
3 -- Last Modified: 21 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28

```

```

29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;
31 use IEEE.STD_LOGIC_ARITH.ALL;
32 use IEEE.STD_LOGIC_SIGNED.ALL;
33
34 library WORK;
35 use WORK.FUNCS.ALL;
36
37 entity acc is
38     port(
39         clock          : in    std_logic;
40         reset          : in    std_logic;
41         clear          : in    std_logic;
42         enable_in     : in    std_logic;
43         enable_out    : out   std_logic;
44         addend        : in    std_logic_vector(30 downto 0);
45         sum           : out   std_logic_vector(33 downto 0)
46     );
47 end acc;
48
49 architecture behavioral of acc is
50     signal addend_t : signed(30 downto 0);
51     signal sum_t    : signed(33 downto 0);
52 begin
53     addend_t <= conv_signed(conv_integer(addend), 31);
54     sum <= conv_std_logic_vector(sum_t);
55
56     -- Working with signed logic vectors in VHDL really, really sucks.
57     -- SXT is sign extend, but it only works on std_logic_vector. So,
58     -- we need lots of conversion operators to make the synthesis tools
59     -- happy. In Verilog, this is much simpler! For now, I wrote new
60     -- conversion functions in the FUNCS package. :)
61     process(clock)
62     begin
63         if(clock 'EVENT and clock = '1') then
64             if(reset = '1') then
65                 sum_t <= (others => '0');
66             elsif(clear = '1') then
67                 sum_t <= conv_signed(sxt(conv_std_logic_vector(addend_t), 34));
68             elsif(enable_in = '1') then
69                 sum_t <= sum_t + conv_signed(sxt(conv_std_logic_vector(addend_t), 34));
70             end if;
71         end if;
72     end process;
73
74     process(clock)
75     begin
76         if(clock 'EVENT and clock = '1') then
77             enable_out <= enable_in;
78         end if;
79     end process;
80 end behavioral;

```

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

```

2  -- ADC Interface
3  -- Last Modified: 20 July 2010
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2010 COSMIAC
7  -- The copyright and license from the original Verilog implementation follows.
8  --
9  --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;
31 use IEEE.STD_LOGIC_ARITH.ALL;
32 use IEEE.STD_LOGIC_UNSIGNED.ALL;
33
34 library WORK;
35 use WORK.FPGA_REGS_COMMON.ALL;
36 use WORK.FPGA_REGS_STANDARD.ALL;
37 use WORK.COMMON_CONFIG.ALL; -- FIXME Should we do this?
38
39 entity adc_interface is
40     port(
41         clock          : in   std_logic;
42         reset          : in   std_logic;
43         enable         : in   std_logic;
44         serial_addr    : in   std_logic_vector(6 downto 0);
45         serial_data    : in   std_logic_vector(31 downto 0);
46         serial_strobe  : in   std_logic;
47         rx_a_a         : in   std_logic_vector(11 downto 0);
48         rx_b_a         : in   std_logic_vector(11 downto 0);
49         rx_a_b         : in   std_logic_vector(11 downto 0);
50         rx_b_b         : in   std_logic_vector(11 downto 0);
51         rssi_0         : out  std_logic_vector(31 downto 0);
52         rssi_1         : out  std_logic_vector(31 downto 0);
53         rssi_2         : out  std_logic_vector(31 downto 0);
54         rssi_3         : out  std_logic_vector(31 downto 0);
55         ddc0_in_i      : out  std_logic_vector(15 downto 0);
56         ddc0_in_q      : out  std_logic_vector(15 downto 0);
57         ddc1_in_i      : out  std_logic_vector(15 downto 0);
58         ddc1_in_q      : out  std_logic_vector(15 downto 0);

```

```

59         ddc2_in_i      : out   std_logic_vector(15 downto 0);
60         ddc2_in_q      : out   std_logic_vector(15 downto 0);
61         ddc3_in_i      : out   std_logic_vector(15 downto 0);
62         ddc3_in_q      : out   std_logic_vector(15 downto 0);
63         rx_numchan     : out   std_logic_vector(3  downto 0)
64     );
65 end adc_interface;
66
67 architecture behavioral of adc_interface is
68     component setting_reg is
69         generic(
70             my_addr : integer := 0
71         );
72
73         port(
74             clock    : in    std_logic;
75             reset    : in    std_logic;
76             strobe   : in    std_logic;
77             addr     : in    std_logic_vector;
78             d_in     : in    std_logic_vector(31 downto 0);
79             d_out    : out   std_logic_vector(31 downto 0);
80             changed  : out   std_logic
81         );
82     end component;
83
84     component rx_dc_offset is
85         generic(
86             MYADDR : integer := 0
87         );
88
89         port(
90             clock      : in    std_logic;
91             enable     : in    std_logic;
92             reset      : in    std_logic;
93             adc_in     : in    std_logic_vector(15 downto 0);
94             adc_out    : out   std_logic_vector(15 downto 0);
95             serial_addr : in    std_logic_vector(6  downto 0);
96             serial_data : in    std_logic_vector(31 downto 0);
97             serial_strobe : in   std_logic
98         );
99     end component;
100
101     component rssi is
102         port(
103             clock      : in    std_logic;
104             reset      : in    std_logic;
105             enable     : in    std_logic;
106             adc        : in    std_logic_vector(11 downto 0);
107             rssi       : out   std_logic_vector(15 downto 0);
108             over_count : out   std_logic_vector(15 downto 0)
109         );
110     end component;
111
112     component adc_mux is
113         port(
114             clock      : in    std_logic;
115             rx_realsignals : in   std_logic;

```

```

116         ddc_mux      : in      std_logic_vector(3 downto 0);
117         adc3_corr    : in      std_logic_vector(15 downto 0);
118         adc2_corr    : in      std_logic_vector(15 downto 0);
119         adc1_corr    : in      std_logic_vector(15 downto 0);
120         adc0_corr    : in      std_logic_vector(15 downto 0);
121         ddc_i        : out     std_logic_vector(15 downto 0);
122         ddc_q        : out     std_logic_vector(15 downto 0)
123     );
124 end component;
125
126 -- Buffers
127 signal adc0 : std_logic_vector(11 downto 0);
128 signal adc1 : std_logic_vector(11 downto 0);
129 signal adc2 : std_logic_vector(11 downto 0);
130 signal adc3 : std_logic_vector(11 downto 0);
131
132 signal dco_en : std_logic_vector(3 downto 0);
133 signal adc0_corr : std_logic_vector(15 downto 0);
134 signal adc1_corr : std_logic_vector(15 downto 0);
135 signal adc2_corr : std_logic_vector(15 downto 0);
136 signal adc3_corr : std_logic_vector(15 downto 0);
137
138 signal ddc0mux : std_logic_vector(3 downto 0);
139 signal ddc1mux : std_logic_vector(3 downto 0);
140 signal ddc2mux : std_logic_vector(3 downto 0);
141 signal ddc3mux : std_logic_vector(3 downto 0);
142
143 signal rx_realsignals : std_logic;
144 begin
145     -- Buffer at the inputs to the chip.
146     process(clock)
147     begin
148         if(clock 'EVENT and clock = '1') then
149             adc0 <= rx_a_a;
150             adc1 <= rx_b_a;
151             adc2 <= rx_a_b;
152             adc3 <= rx_b_b;
153         end if;
154     end process;
155
156     -- Then scale and subtract DC offset.
157     sr_dco_en : setting_reg
158         generic map(my_addr => FR_DC_OFFSET_CL_EN)
159         port map(clock => clock, reset => reset, strobe => serial_strobe, addr => serial_addr,
160             d_in => serial_data, d_out(31 downto 4) => OPEN, d_out(3 downto 0) => dco_en, changed
161             => OPEN);
162
163     rx_dcoffset0 : rx_dcoffset
164         generic map(MYADDR => FR_ADC_OFFSET_0)
165         port map(clock => clock, enable => dco_en(0), reset => reset, adc_in(15) => adc0(11),
166             adc_in(14 downto 3) => adc0, adc_in(2 downto 0) => "000", adc_out => adc0_corr,
167             serial_addr => serial_addr, serial_data => serial_data, serial_strobe =>
168             serial_strobe);
169
170     rx_dcoffset1 : rx_dcoffset
171         generic map(MYADDR => FR_ADC_OFFSET_1)
172         port map(clock => clock, enable => dco_en(1), reset => reset, adc_in(15) => adc1(11),

```

```

171         adc_in(14 downto 3) => adc1, adc_in(2 downto 0) => "000", adc_out => adc1_corr ,
172         serial_addr => serial_addr , serial_data => serial_data , serial_strobe =>
            serial_strobe);
173
174 rx_dcoffset2 : rx_dcoffset
175     generic map(MYADDR => FR_ADC_OFFSET_2)
176     port map(clock => clock, enable => dco_en(2), reset => reset , adc_in(15) => adc2(11),
177             adc_in(14 downto 3) => adc2, adc_in(2 downto 0) => "000", adc_out => adc2_corr ,
178             serial_addr => serial_addr , serial_data => serial_data , serial_strobe =>
            serial_strobe);
179
180 rx_dcoffset3 : rx_dcoffset
181     generic map(MYADDR => FR_ADC_OFFSET_3)
182     port map(clock => clock, enable => dco_en(3), reset => reset , adc_in(15) => adc3(11),
183             adc_in(14 downto 3) => adc3, adc_in(2 downto 0) => "000", adc_out => adc3_corr ,
184             serial_addr => serial_addr , serial_data => serial_data , serial_strobe =>
            serial_strobe);
185
186 -- Level sensing for AGC
187 rssi_block_0 : rssi
188     port map(clock => clock, reset => reset, enable => enable, adc => adc0, rssi => rssi_0(15
            downto 0),
189             over_count => rssi_0(31 downto 16));
190
191 rssi_block_1 : rssi
192     port map(clock => clock, reset => reset, enable => enable, adc => adc1, rssi => rssi_1(15
            downto 0),
193             over_count => rssi_1(31 downto 16));
194
195 rssi_block_2 : rssi
196     port map(clock => clock, reset => reset, enable => enable, adc => adc2, rssi => rssi_2(15
            downto 0),
197             over_count => rssi_2(31 downto 16));
198
199 rssi_block_3 : rssi
200     port map(clock => clock, reset => reset, enable => enable, adc => adc3, rssi => rssi_3(15
            downto 0),
201             over_count => rssi_3(31 downto 16));
202
203 -- And MUX to the appropriate outputs.
204 sr_rxmux : setting_reg
205     generic map(my_addr => FR_RX_MUX)
206     port map(clock => clock, reset => reset, strobe => serial_strobe, addr => serial_addr ,
207             d_in => serial_data , d_out(31 downto 20) => OPEN, d_out(19 downto 16) => ddc3mux,
208             d_out(15 downto 12) => ddc2mux, d_out(11 downto 8) => ddc1mux, d_out(7 downto 4) =>
            ddc0mux,
209             d_out(3) => rx_realsignals , d_out(2 downto 0) => rx_numchan(3 downto 1), changed =>
            OPEN);
210
211 rx_numchan(0) <= '0';
212
213 -- VHDL really needs to start using ternary operators! ?: is awesome, this is ugly.
214 -- rx_en_0_t : if (RX_EN_0) generate
215     adc_mux0 : adc_mux
216     port map(clock => clock, rx_realsignals => rx_realsignals , ddc_mux => ddc0mux,
217             adc3_corr => adc3_corr , adc2_corr => adc2_corr , adc1_corr => adc1_corr , adc0_corr
            => adc0_corr ,

```

```

218             ddc_i => ddc0_in_i , ddc_q => ddc0_in_q);
219 -- end generate rx_en_0_t;
220
221 -- rx_en_0_f : if(not RX_EN_0) generate
222 --     ddc0_in_i <= (others => '0');
223 --     ddc0_in_q <= (others => '0');
224 -- end generate rx_en_0_f;
225
226 -- rx_en_1_t : if(RX_EN_1) generate
227     adc_mux1 : adc_mux
228     port map(clock => clock , rx_realsignals => rx_realsignals , ddc_mux => ddc1mux ,
229             adc3_corr => adc3_corr , adc2_corr => adc2_corr , adc1_corr => adc1_corr , adc0_corr
230             => adc0_corr ,
231             ddc_i => ddc1_in_i , ddc_q => ddc1_in_q);
232 -- end generate rx_en_1_t;
233
234 -- rx_en_1_f : if(not RX_EN_1) generate
235 --     ddc1_in_i <= (others => '0');
236 --     ddc1_in_q <= (others => '0');
237 -- end generate rx_en_1_f;
238
239 -- rx_en_2_t : if(RX_EN_2) generate
240     adc_mux2 : adc_mux
241     port map(clock => clock , rx_realsignals => rx_realsignals , ddc_mux => ddc2mux ,
242             adc3_corr => adc3_corr , adc2_corr => adc2_corr , adc1_corr => adc1_corr , adc0_corr
243             => adc0_corr ,
244             ddc_i => ddc2_in_i , ddc_q => ddc2_in_q);
245 -- end generate rx_en_2_t;
246
247 -- rx_en_2_f : if(not RX_EN_2) generate
248 --     ddc2_in_i <= (others => '0');
249 --     ddc2_in_q <= (others => '0');
250 -- end generate rx_en_2_f;
251
252 -- rx_en_3_t : if(RX_EN_3) generate
253     adc_mux3 : adc_mux
254     port map(clock => clock , rx_realsignals => rx_realsignals , ddc_mux => ddc3mux ,
255             adc3_corr => adc3_corr , adc2_corr => adc2_corr , adc1_corr => adc1_corr , adc0_corr
256             => adc0_corr ,
257             ddc_i => ddc3_in_i , ddc_q => ddc3_in_q);
258 -- end generate rx_en_3_t;
259
260 -- rx_en_3_f : if(not RX_EN_3) generate
261 --     ddc3_in_i <= (others => '0');
262 --     ddc3_in_q <= (others => '0');
263 -- end generate rx_en_3_f;
264
265 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`)

```

1 -----
2 -- Special MUX for ADC Signals
3 -- Last Modified: 20 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC

```

```

7  --
8  -- This program is free software; you can redistribute it and/or modify
9  -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22
23 -- The original Verilog USRP1 contains no equivalent for this component. In
24 -- Verilog, this logic is trivial to implement with the ternary ?: operator.
25 library IEEE;
26 use IEEE.STD_LOGIC_1164.ALL;
27 use IEEE.STD_LOGIC_ARITH.ALL;
28 use IEEE.STD_LOGIC_UNSIGNED.ALL;
29
30 entity adc_mux is
31     port(
32         clock          : in    std_logic;
33         rx_realsignals : in    std_logic;
34         ddc_mux        : in    std_logic_vector(3 downto 0);
35         adc3_corr      : in    std_logic_vector(15 downto 0);
36         adc2_corr      : in    std_logic_vector(15 downto 0);
37         adc1_corr      : in    std_logic_vector(15 downto 0);
38         adc0_corr      : in    std_logic_vector(15 downto 0);
39         ddc_i          : out   std_logic_vector(15 downto 0);
40         ddc_q          : out   std_logic_vector(15 downto 0)
41     );
42 end adc_mux;
43
44 architecture behavioral of adc_mux is
45 begin
46     -- First, take care of ddc_i. The equivalent Verilog statement is:
47     -- ddc_i = ddc_mux[1] ? (ddc_mux[0] ? adc3_corr
48     --                               : adc2_corr)
49     --           : (ddc_mux[0] ? adc1_corr
50     --                               : adc0_corr);
51     process(clock)
52     begin
53         if(clock'EVENT and clock = '1') then
54             if(ddc_mux(1) = '1') then
55                 if(ddc_mux(0) = '1') then
56                     ddc_i <= adc3_corr;
57                 else
58                     ddc_i <= adc2_corr;
59                 end if;
60             else
61                 if(ddc_mux(0) = '1') then
62                     ddc_i <= adc1_corr;
63                 else

```



```

64             ddc_i <= adc0_corr;
65         end if;
66     end if;
67 end if;
68 end process;
69
70 -- Now, take care of ddc_q. The equivalent Verilog statement is:
71 -- ddc_q = rx_realsignals ? 16'b0
72 --             : ddc_mux[3] ? (ddc_mux[2] ? adc3_corr
73 --                               : adc2_corr)
74 --             : (ddc_mux[2] ? adc1_corr
75 --                               : adc0_corr);
76 process(clock)
77 begin
78     if(clock 'EVENT and clock = '1') then
79         if(rx_realsignals = '1') then
80             ddc_q <= (others => '0');
81         else
82             if(ddc_mux(3) = '1') then
83                 if(ddc_mux(2) = '1') then
84                     ddc_q <= adc3_corr;
85                 else
86                     ddc_q <= adc2_corr;
87                 end if;
88             else
89                 if(ddc_mux(2) = '1') then
90                     ddc_q <= adc1_corr;
91                 else
92                     ddc_q <= adc0_corr;
93                 end if;
94             end if;
95         end if;
96     end if;
97 end process;
98 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)

```

1 -----
2 -- ATR Delay
3 -- Last Modified: 21 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --

```

```

19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;
31 use IEEE.STD_LOGIC_ARITH.ALL;
32 use IEEE.STD_LOGIC_UNSIGNED.ALL;
33
34 entity atr_delay is
35     port(
36         clk_i          : in    std_logic;
37         rst_i          : in    std_logic;
38         ena_i          : in    std_logic;
39         tx_empty_i    : in    std_logic;
40         tx_delay_i    : in    std_logic_vector(11 downto 0);
41         rx_delay_i    : in    std_logic_vector(11 downto 0);
42         atr_tx_o      : out   std_logic
43     );
44 end atr_delay;
45
46 -- Auto Transmit/Receive Switch. Adds a delay when switching from Tx to Rx
47 -- or from Rx to Tx. Configurable as a multiple of clock cycles.
48 architecture behavioral of atr_delay is
49     signal state : std_logic_vector(3 downto 0);
50     signal count : std_logic_vector(11 downto 0);
51
52     -- The original Verilog forces one-hot encoding and I've kept that here.
53     -- XST seems to prefer grey coding for both versions, though.
54     constant ST_RX_DELAY : std_logic_vector(3 downto 0) := "0001";
55     constant ST_RX      : std_logic_vector(3 downto 0) := "0010";
56     constant ST_TX_DELAY : std_logic_vector(3 downto 0) := "0100";
57     constant ST_TX      : std_logic_vector(3 downto 0) := "1000";
58 begin
59     process(clk_i)
60     begin
61         if (clk_i'EVENT and clk_i = '1') then
62             if (rst_i = '1' or ena_i = '0') then
63                 state <= ST_RX;
64                 count <= (others => '0');
65             else
66                 case state is
67                     -- Receiving State
68                     when ST_RX =>
69                         -- If there's something in the Tx buffer, go to Tx.
70                         if (tx_empty_i = '0') then
71                             state <= ST_TX_DELAY;
72                             count <= tx_delay_i;
73                         end if;
74                         -- Rx->Tx Delay State
75                     when ST_TX_DELAY =>

```

```

76         if(count = 0) then
77             state <= ST_TX;
78         else
79             count <= count - 1;
80         end if;
81     -- Transmitting State
82     when ST_TX =>
83         -- When the Tx buffer is empty, go back to Rx.
84         if(tx_empty_i = '1') then
85             state <= ST_RX_DELAY;
86             count <= rx_delay_i;
87         end if;
88     -- Tx->Rx Delay State
89     when ST_RX_DELAY =>
90         if(count = 0) then
91             state <= ST_RX;
92         else
93             count <= count - 1;
94         end if;
95     when others => -- Error
96         state <= ST_RX;
97         count <= (others => '0');
98     end case;
99     end if;
100 end if;
101 end process;
102
103 atr_tx_o <= '1' when (state = ST_TX or state = ST_RX_DELAY) else '0';
104 end behavioral;

```

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

```

1 -----
2 -- Tri-State Register
3 -- Last Modified: 21 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software

```

```

26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;
31 use IEEE.STD_LOGIC_ARITH.ALL;
32 use IEEE.STD_LOGIC_UNSIGNED.ALL;
33
34 entity bidir_reg is
35     port(
36         tristate    : inout std_logic_vector(15 downto 0);
37         oe          : in      std_logic_vector(15 downto 0);
38         reg_val     : in      std_logic_vector(15 downto 0)
39     );
40 end bidir_reg;
41
42 architecture behavioral of bidir_reg is
43 begin
44     -- This is just another macro for a 16-bit tri-state bus. This one
45     -- has a separate enable for each bit, unlike the bus-wide enable
46     -- in bustri.
47     process(reg_val, oe)
48     begin
49         for i in 15 downto 0 loop
50             if(oe(i) = '1') then
51                 tristate(i) <= reg_val(i);
52             else
53                 tristate(i) <= 'Z';
54             end if;
55         end loop;
56     end process;
57 end behavioral;

```

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

```

1 -----
2 -- Tri-state Bus Model
3 -- Last Modified: 20 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

```

```

22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;
31
32 entity bustri is
33     port(
34         data      : in    std_logic_vector(15 downto 0);
35         enabledt  : in    std_logic;
36         tridata   : out   std_logic_vector(15 downto 0)
37     );
38 end bustri;
39
40 architecture behavioral of bustri is
41 begin
42     -- This is just a macro for a 16-bit tri-state bus.
43     tridata <= data when enabledt = '1' else (others => 'Z');
44 end behavioral;

```

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

```

1 -----
2 -- CIC Decimation
3 -- Last Modified: 21 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;

```

```

31 use IEEE.STD_LOGIC_ARITH.ALL;
32 use IEEE.STD_LOGIC_UNSIGNED.ALL;
33
34 -- A CIC decimator is equivalent to a moving average filter. Basically,
35 -- take N equivalent moving average filters (FIR filters), then separate
36 -- them so that the input goes through all integrators first and then all combs.
37 -- We can do this because both combs and integrators are LTI. To think of a
38 -- moving average filter, imagine that at each step we add the newest sample
39 -- and remove the oldest sample, then divide by the number of stages in the
40 -- filter. Something like this (divided by N).
41 --  $y[n] = y[n-1] + x[n] - x[n-R]$ 
42 entity cic_decim is
43     generic(
44         bw          : integer := 16;    -- # of bits for input
45         N           : integer := 4;    -- # of filter stages
46         log2_of_max_rate : integer := 7; -- log2 of max sampling rate
47         maxbitgain   : integer := 28 --  $N * \log_2 \text{of\_max\_rate}$ 
48     );
49
50     port(
51         clock      : in    std_logic;
52         reset      : in    std_logic;
53         enable     : in    std_logic;
54         rate       : in    std_logic_vector(7 downto 0);
55         strobe_in  : in    std_logic;
56         strobe_out : in    std_logic;
57         signal_in  : in    std_logic_vector(bw-1 downto 0);
58         signal_out : out   std_logic_vector(bw-1 downto 0)
59     );
60 end cic_decim;
61
62 architecture behavioral of cic_decim is
63     component sign_extend is
64         generic(
65             bits_in  : integer := 0;
66             bits_out : integer := 0
67         );
68
69         port(
70             d_in   : in    std_logic_vector(bits_in-1 downto 0);
71             d_out  : out   std_logic_vector(bits_out-1 downto 0)
72         );
73     end component;
74
75     component cic_dec_shifter is
76         generic(
77             bw : integer := 16;
78             maxbitgain : integer := 28
79         );
80
81         port(
82             rate      : in    std_logic_vector(7 downto 0);
83             signal_in : in    std_logic_vector(bw+maxbitgain-1 downto 0);
84             signal_out : out   std_logic_vector(bw-1 downto 0)
85         );
86     end component;
87

```

```

88     type mem is array (integer range <>) of std_logic_vector(bw+maxbitgain-1 downto 0);
89     signal integrator : mem(0 to N-1);
90     signal differentiator : mem(0 to N-1);
91     signal pipeline : mem(0 to N-1);
92
93     signal signal_out_t : std_logic_vector(bw-1 downto 0);
94
95     signal signal_in_ext : std_logic_vector(bw+maxbitgain-1 downto 0);
96
97     signal signal_out_unreg : std_logic_vector(bw-1 downto 0);
98     signal signal_out_unnorm : std_logic_vector(bw+maxbitgain-1 downto 0);
99
100    signal sampler : std_logic_vector(bw+maxbitgain-1 downto 0);
101    begin
102    -- Sign extend the input signal to account for bit gain in the filters.
103    ext_input : sign_extend
104        generic map(bits_in => bw, bits_out => (bw+maxbitgain))
105        port map(d_in => signal_in, d_out => signal_in_ext);
106
107    -- Integrator
108    process(clock)
109    begin
110        if(clock 'EVENT and clock = '1') then
111            if(reset = '1') then
112                -- Clear the integrator.
113                for i in 0 to N-1 loop
114                    integrator(i) <= (others => '0');
115                end loop;
116            elsif((enable = '1') and (strobe_in = '1')) then
117                -- Grab the next sample.
118                integrator(0) <= integrator(0) + signal_in_ext;
119
120                -- Integrate! The integrator adds the newest sample to the
121                -- moving average.
122                --  $y[n] = y[n-1] + c[n]$ 
123                for i in 1 to N-1 loop
124                    integrator(i) <= integrator(i) + integrator(i-1);
125                end loop;
126            end if;
127        end if;
128    end process;
129
130    -- Comb Filter
131    process(clock)
132    begin
133        if(clock 'EVENT and clock = '1') then
134            if(reset = '1') then
135                -- Clear the comb filters.
136                sampler <= (others => '0');
137                for i in 0 to N-1 loop
138                    pipeline(i) <= (others => '0');
139                    differentiator(i) <= (others => '0');
140                end loop;
141            elsif((enable = '1') and (strobe_out = '1')) then
142                -- Grab the output of the integrator into the comb filter.
143                sampler <= integrator(N-1);
144

```

```

145         -- The comb filters remove the oldest samples from the
146         -- moving average.
147         --  $c[n] = x[n] - x[n - R]$ 
148         differentiator(0) <= sampler;
149         pipeline(0) <= sampler - differentiator(0);
150         for i in 1 to N-1 loop
151             differentiator(i) <= pipeline(i-1);
152             pipeline(i) <= pipeline(i-1) - differentiator(i);
153         end loop;
154     end if;
155 end if;
156 end process;
157
158 -- Normalize the output to 'bw' bits.
159 signal_out_unnorm <= pipeline(N-1);
160
161 cic_dec_shifter0 : cic_dec_shifter
162     generic map(bw => bw)
163     port map(rate => rate, signal_in => signal_in, signal_out => signal_out_unreg);
164
165 -- Register the outputs.
166 process(clock)
167 begin
168     if(clock'EVENT and clock = '1') then
169         signal_out <= signal_out_unreg;
170     end if;
171 end process;
172 end behavioral;

```

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

```

1 -----
2 -- CIC Decimation Shifter
3 -- Last Modified: 21 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software

```



```

26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 -- This only works for N=4, max decimation rate of 128.
30 -- The signal "rate" is ONE LESS THAN the actual rate.
31 library IEEE;
32 use IEEE.STD_LOGIC_1164.ALL;
33 use IEEE.STD_LOGIC_ARITH.ALL;
34 use IEEE.STD_LOGIC_UNSIGNED.ALL;
35
36 entity cic_dec_shifter is
37     generic(
38         bw          : integer := 16;
39         maxbitgain  : integer := 28
40     );
41
42     port(
43         rate        : in   std_logic_vector(7 downto 0);
44         signal_in   : in   std_logic_vector(bw+maxbitgain-1 downto 0);
45         signal_out  : out  std_logic_vector(bw-1 downto 0)
46     );
47 end cic_dec_shifter;
48
49 architecture behavioral of cic_dec_shifter is
50     signal shift : std_logic_vector(4 downto 0);
51     signal ratel : std_logic_vector(7 downto 0);
52 begin
53     -- I don't know why the original Verilog source uses rate-1 in the top
54     -- module and then fixes it here. But this is really the decimation
55     -- rate.
56     ratel <= rate + 1;
57
58     -- The actual bit gain is determined by the rate, where
59     -- bitgain = N * log2(R)
60     -- We fix N at 4 here and R is equal to 'rate'
61     with conv_integer(ratel) select
62         shift <= conv_std_logic_vector(8, 5) when 4,
63             conv_std_logic_vector(12, 5) when 8,
64             conv_std_logic_vector(16, 5) when 16,
65             conv_std_logic_vector(20, 5) when 32,
66             conv_std_logic_vector(24, 5) when 64,
67             conv_std_logic_vector(28, 5) when 128,
68             conv_std_logic_vector(10, 5) when 5,
69             conv_std_logic_vector(11, 5) when 6,
70             conv_std_logic_vector(12, 5) when 7,
71             conv_std_logic_vector(13, 5) when 9,
72             conv_std_logic_vector(14, 5) when 10 | 11,
73             conv_std_logic_vector(15, 5) when 12 | 13,
74             conv_std_logic_vector(16, 5) when 14 | 15,
75             conv_std_logic_vector(17, 5) when 17 | 18 | 19,
76             conv_std_logic_vector(18, 5) when 20 | 21 | 22,
77             conv_std_logic_vector(19, 5) when 23 | 24 | 25 | 26,
78             conv_std_logic_vector(20, 5) when 27 | 28 | 29 | 30 | 31,
79             conv_std_logic_vector(21, 5) when 33 | 34 | 35 | 36 | 37 | 38,
80             conv_std_logic_vector(22, 5) when 39 | 40 | 41 | 42 | 43 | 44 | 45,
81             conv_std_logic_vector(23, 5) when 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53,

```

```

82         conv_std_logic_vector(24, 5) when 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62
           | 63,
83         conv_std_logic_vector(25, 5) when 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73
           | 74 | 75 | 76,
84         conv_std_logic_vector(26, 5) when 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85
           | 86 | 87 | 88 | 89 | 90,
85         conv_std_logic_vector(27, 5) when 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99
           | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107,
86         conv_std_logic_vector(28, 5) when others;
87
88     -- Wouldn't this be nice? Xilinx ISE throws the following warning:
89     --   Index value(s) does not match array range, simulation mismatch.
90     -- signal_out <= signal_in(conv_integer(shift)+bw-1 downto conv_integer(shift));
91
92     -- So instead, we're stuck with this.
93     signal_out <= signal_in(8+bw-1 downto 8) when shift = 8 else
94         signal_in(12+bw-1 downto 12) when shift = 12 else
95         signal_in(16+bw-1 downto 16) when shift = 16 else
96         signal_in(20+bw-1 downto 20) when shift = 20 else
97         signal_in(24+bw-1 downto 24) when shift = 24 else
98         signal_in(28+bw-1 downto 28) when shift = 28 else
99         signal_in(10+bw-1 downto 10) when shift = 10 else
100        signal_in(11+bw-1 downto 11) when shift = 11 else
101        signal_in(13+bw-1 downto 13) when shift = 13 else
102        signal_in(14+bw-1 downto 14) when shift = 14 else
103        signal_in(15+bw-1 downto 15) when shift = 15 else
104        signal_in(17+bw-1 downto 17) when shift = 17 else
105        signal_in(18+bw-1 downto 18) when shift = 18 else
106        signal_in(19+bw-1 downto 19) when shift = 19 else
107        signal_in(21+bw-1 downto 21) when shift = 21 else
108        signal_in(22+bw-1 downto 22) when shift = 22 else
109        signal_in(23+bw-1 downto 23) when shift = 23 else
110        signal_in(25+bw-1 downto 25) when shift = 25 else
111        signal_in(26+bw-1 downto 26) when shift = 26 else
112        signal_in(27+bw-1 downto 27) when shift = 27 else
113        signal_in(28+bw-1 downto 28);
114 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)

```

1 -----
2 -- CIC Interpolation
3 -- Last Modified: 20 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or

```

```

17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;
31 use IEEE.STD_LOGIC_ARITH.ALL;
32 use IEEE.STD_LOGIC_UNSIGNED.ALL;
33
34 -- A CIC interpolator is equivalent to a moving average filter. Basically,
35 -- take N equivalent moving average filters (FIR filters), then separate
36 -- them so that the input goes through all combs first and then all integrators.
37 -- We can do this because both combs and integrators are LTI. To think of a
38 -- moving average filter, imagine that at each step we add the newest sample
39 -- and remove the oldest sample, then divide by the number of stages in the
40 -- filter. Something like this (divided by N).
41 --  $y[n] = y[n-1] + x[n] - x[n-R]$ 
42 entity cic_interp is
43   generic(
44     bw          : integer := 16; -- # of bits for input
45     N           : integer := 4; -- # of filter stages
46     log2_of_max_rate : integer := 7; -- log2 of max sampling rate
47     maxbitgain   : integer := 21 -- (N-1)*log2_of_max_rate
48   );
49
50   port(
51     clock      : in    std_logic;
52     reset      : in    std_logic;
53     enable     : in    std_logic;
54     rate       : in    std_logic_vector(7 downto 0);
55     strobe_in  : in    std_logic;
56     strobe_out : in    std_logic;
57     signal_in  : in    std_logic_vector(bw-1 downto 0);
58     signal_out : out   std_logic_vector(bw-1 downto 0)
59   );
60 end cic_interp;
61
62 architecture behavioral of cic_interp is
63   component sign_extend is
64     generic(
65       bits_in : integer := 0;
66       bits_out : integer := 0
67     );
68
69     port(
70       d_in  : in    std_logic_vector(bits_in-1 downto 0);
71       d_out : out   std_logic_vector(bits_out-1 downto 0)
72     );
73   end component;

```

```

74
75 component cic_int_shifter is
76     generic(
77         bw            : integer := 16;
78         maxbitgain   : integer := 21
79     );
80
81     port(
82         rate          : in    std_logic_vector(7 downto 0);
83         signal_in     : in    std_logic_vector(bw+maxbitgain-1 downto 0);
84         signal_out    : out   std_logic_vector(bw-1 downto 0)
85     );
86 end component;
87
88 type mem is array (integer range <>) of std_logic_vector(bw+maxbitgain-1 downto 0);
89 signal integrator : mem(0 to N-1);
90 signal differentiator : mem(0 to N-1);
91 signal pipeline : mem(0 to N-1);
92
93 signal signal_in_ext : std_logic_vector(bw+maxbitgain-1 downto 0);
94
95 signal clear_me : std_logic;
96 signal signal_out_unnorm : std_logic_vector(bw+maxbitgain-1 downto 0);
97 begin
98     -- Sign extend the input signal to account for bit gain in the filters.
99     ext_input : sign_extend
100         generic map(bits_in => bw, bits_out => (bw+maxbitgain))
101         port map(d_in => signal_in, d_out => signal_in_ext);
102
103     -- A simple OR gate to reset the filters.
104     clear_me <= '1' when ((reset = '1') or (enable = '0')) else '0';
105
106     -- FIXME Note that this section has pipe and diff reversed.
107     -- It still works, but is confusing.
108
109     -- Comb Filter
110     process(clock)
111     begin
112         if(clock 'EVENT and clock = '1') then
113             if(clear_me = '1') then
114                 -- Clear the comb filters.
115                 for i in 0 to N-1 loop
116                     differentiator(i) <= (others => '0');
117                     pipeline(i) <= (others => '0');
118                 end loop;
119                 elsif((enable = '1') and (strobe_in = '1')) then
120                     -- The comb filters remove the oldest samples from the
121                     -- moving average.
122                     --  $c[n] = x[n] - x[n - R]$ 
123                     differentiator(0) <= signal_in_ext;
124                     pipeline(0) <= signal_in_ext - differentiator(0);
125                     for i in 1 to N-1 loop
126                         differentiator(i) <= pipeline(i-1);
127                         pipeline(i) <= pipeline(i-1) - differentiator(i);
128                     end loop;
129                 end if;
130             end if;

```

```

131     end process;
132
133     -- Integrator
134     process(clock)
135     begin
136         if(clock 'EVENT and clock = '1') then
137             if(clear_me = '1') then
138                 -- Clear the integrator.
139                 for i in 0 to N-1 loop
140                     integrator(i) <= (others => '0');
141                 end loop;
142             elsif((enable = '1') and (strobe_out = '1')) then
143                 if(strobe_in = '1') then
144                     -- Grab the output of the comb filters into the integrator.
145                     integrator(0) <= integrator(0) + pipeline(N-1);
146                 end if;
147
148                 -- Integrate! The integrator adds the newest sample to the
149                 -- moving average.
150                 --  $y[n] = y[n-1] + c[n]$ 
151                 for i in 1 to N-1 loop
152                     integrator(i) <= integrator(i) + integrator(i-1);
153                 end loop;
154             end if;
155         end if;
156     end process;
157
158     -- Normalize the output to 'bw' bits.
159     signal_out_unnorm <= integrator(N-1);
160
161     cic_int_shifter0 : cic_int_shifter
162         generic map(bw => bw)
163         port map(rate => rate, signal_in => signal_out_unnorm, signal_out => signal_out);
164 end behavioral;

```

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

```

1  -----
2  -- CIC Interpolation Shifter
3  -- Last Modified: 20 July 2010
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2010 COSMIAC
7  -- The copyright and license from the original Verilog implementation follows.
8  --
9  --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,

```

```

20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 -- This only works for N=4, max interp rate of 128.
30 -- Signal "rate" is ONE LESS THAN the actual rate.
31 library IEEE;
32 use IEEE.STD_LOGIC_1164.ALL;
33 use IEEE.STD_LOGIC_ARITH.ALL;
34 use IEEE.STD_LOGIC_UNSIGNED.ALL;
35
36 entity cic_int_shifter is
37     generic(
38         bw          : integer := 16;
39         maxbitgain  : integer := 21
40     );
41
42     port(
43         rate        : in    std_logic_vector(7 downto 0);
44         signal_in   : in    std_logic_vector(bw+maxbitgain-1 downto 0);
45         signal_out  : out   std_logic_vector(bw-1 downto 0)
46     );
47 end cic_int_shifter;
48
49 architecture behavioral of cic_int_shifter is
50     signal shift : std_logic_vector(4 downto 0);
51     signal ratel : std_logic_vector(7 downto 0);
52 begin
53     -- I don't know why the original Verilog source uses rate-1 in the top
54     -- module and then fixes it here. But this is really the interpolation
55     -- rate.
56     ratel <= rate + 1;
57
58     -- The actual bit gain is determined by the rate, where
59     -- bitgain = (N-1) * log2(R)
60     -- We fix N at 4 here and R is equal to 'rate'.
61     with conv_integer(ratel) select
62         shift <= conv_std_logic_vector(6, 5) when 4,
63             conv_std_logic_vector(9, 5) when 8,
64             conv_std_logic_vector(12, 5) when 16,
65             conv_std_logic_vector(15, 5) when 32,
66             conv_std_logic_vector(18, 5) when 64,
67             conv_std_logic_vector(21, 5) when 128,
68             conv_std_logic_vector(7, 5) when 5,
69             conv_std_logic_vector(8, 5) when 6,
70             conv_std_logic_vector(9, 5) when 7,
71             conv_std_logic_vector(10, 5) when 9 | 10,
72             conv_std_logic_vector(11, 5) when 11 | 12,
73             conv_std_logic_vector(12, 5) when 13 | 14 | 15,
74             conv_std_logic_vector(13, 5) when 17 | 18 | 19 | 20,
75             conv_std_logic_vector(14, 5) when 21 | 22 | 23 | 24 | 25,
76             conv_std_logic_vector(15, 5) when 26 | 27 | 28 | 29 | 30 | 31,

```

```

77         conv_std_logic_vector(16, 5) when 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40,
78         conv_std_logic_vector(17, 5) when 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49
          | 50,
79         conv_std_logic_vector(18, 5) when 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59
          | 60 | 61 | 62 | 63,
80         conv_std_logic_vector(19, 5) when 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73
          | 74 | 75 | 76 | 77 | 78 | 79 | 80,
81         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,
82         conv_std_logic_vector(21, 5) when others;
83
84     -- Wouldn't this be nice? Xilinx ISE throws the following warning:
85     --   Index value(s) does not match array range, simulation mismatch.
86     -- signal_out <= signal_in(conv_integer(shift)+bw-1 downto conv_integer(shift));
87
88     -- So instead, we're stuck with this.
89     signal_out <= signal_in(6+bw-1 downto 6) when shift = 6 else
90         signal_in(9+bw-1 downto 9) when shift = 9 else
91         signal_in(12+bw-1 downto 12) when shift = 12 else
92         signal_in(15+bw-1 downto 15) when shift = 15 else
93         signal_in(18+bw-1 downto 18) when shift = 18 else
94         signal_in(21+bw-1 downto 21) when shift = 21 else
95         signal_in(7+bw-1 downto 7) when shift = 7 else
96         signal_in(8+bw-1 downto 8) when shift = 8 else
97         signal_in(10+bw-1 downto 10) when shift = 10 else
98         signal_in(11+bw-1 downto 11) when shift = 11 else
99         signal_in(13+bw-1 downto 13) when shift = 13 else
100        signal_in(14+bw-1 downto 14) when shift = 14 else
101        signal_in(16+bw-1 downto 16) when shift = 16 else
102        signal_in(17+bw-1 downto 17) when shift = 17 else
103        signal_in(19+bw-1 downto 19) when shift = 19 else
104        signal_in(20+bw-1 downto 20) when shift = 20 else
105        signal_in(21+bw-1 downto 21);
106 end behavioral;

```

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

```

1  -----
2  -- Clock Divider
3  -- Last Modified: 21 July 2010
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2010 COSMIAC
7  -- The copyright and license from the original Verilog implementation follows.
8  --
9  --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,

```

```

20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;
31 use IEEE.STD_LOGIC_ARITH.ALL;
32 use IEEE.STD_LOGIC_UNSIGNED.ALL;
33
34 entity clk_divider is
35     port(
36         reset      : in    std_logic;
37         in_clk     : in    std_logic;
38         out_clk    : out   std_logic;
39         ratio      : in    std_logic_vector(7 downto 0)
40     );
41 end clk_divider;
42
43 architecture behavioral of clk_divider is
44     -- Original Verilog had [7:0], but it was wrong.
45     signal counter : std_logic_vector(7 downto 0);
46     signal out_clk_t : std_logic;
47     signal count_tmp : std_logic;
48 begin
49     out_clk <= out_clk_t;
50     count_tmp <= '1' when ((ratio(0) = '1') and (out_clk_t = '1')) else '0';
51     -- Use a down counter to count clocks between each toggle of out_clk.
52     process(in_clk, reset)
53     begin
54         if(reset = '1') then
55             counter <= (others => '0');
56         elsif(in_clk 'EVENT and in_clk = '1') then
57             if(counter = 0) then
58                 -- If counter is zero here, we are diving in_clk by 2.
59                 -- This ugly notation forces XST to generate a carry adder like it should.
60                 counter <= conv_std_logic_vector(conv_integer(ratio(7 downto 1)) +
61                     conv_integer(count_tmp) - 1, 8);
62             else
63                 counter <= counter - "00000001";
64             end if;
65         end if;
66     end process;
67     -- Toggle out_clock when the counter hits zero.
68     process(in_clk, reset)
69     begin
70         if(reset = '1') then
71             out_clk_t <= '0';
72         elsif(in_clk 'EVENT and in_clk = '1') then
73             if(counter = 0) then
74                 out_clk_t <= (not out_clk_t);
75             end if;

```



```

76     end if;
77     end process;
78 end behavioral;

```

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

```

1  -----
2  -- Coefficient ROM for the Halfband Decimation Filter
3  -- Last Modified: 14 February 2011
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2010 COSMIAC
7  -- The copyright and license from the original Verilog implementation follows.
8  --
9  --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 -- TODO: Migrate this module to the new numeric_std library standard.
30 library IEEE;
31 use IEEE.STD_LOGIC_1164.ALL;
32 use IEEE.STD_LOGIC_ARITH.ALL;
33 use IEEE.STD_LOGIC_UNSIGNED.ALL;
34
35 entity coeff_rom is
36     port(
37         clock    : in    std_logic;
38         addr     : in    std_logic_vector(2 downto 0);
39         data     : out   std_logic_vector(15 downto 0)
40     );
41 end coeff_rom;
42
43 architecture behavioral of coeff_rom is
44     signal data_t : signed(15 downto 0);
45 begin
46     data <= conv_std_logic_vector(data_t, 16);
47
48     process(clock)
49     begin
50         if(clock 'EVENT and clock = '1') then

```

```

51         case conv_integer(addr) is
52             when 0 =>
53                 data_t <= conv_signed(-49, 16);
54             when 1 =>
55                 data_t <= conv_signed(165, 16);
56             when 2 =>
57                 data_t <= conv_signed(-412, 16);
58             when 3 =>
59                 data_t <= conv_signed(873, 16);
60             when 4 =>
61                 data_t <= conv_signed(-1681, 16);
62             when 5 =>
63                 data_t <= conv_signed(3135, 16);
64             when 6 =>
65                 data_t <= conv_signed(-6282, 16);
66             when 7 =>
67                 data_t <= conv_signed(20628, 16);
68             when others =>
69         end case;
70     end if;
71 end process;
72 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)

```

1 -----
2 -- USRP Common Configuration
3 -- Last Modified: 21 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 --
8 -- This program is free software; you can redistribute it and/or modify
9 -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22
23 -- This is the common tail for standard configuration. Uncomment a single
24 -- Tx/Rx configuration in the 'use' section.
25 library IEEE;
26 use IEEE.STD_LOGIC_1164.ALL;
27 use IEEE.STD_LOGIC_ARITH.ALL;
28
29 library WORK;

```

```

30 use WORK.FUNCS.ALL;
31
32 -- Configurations go here. Uncomment only one configuration!
33 use WORK.COMMON_CONFIG_1RXHB_1TX.ALL;
34 --use WORK.COMMON_CONFIG_2RX_0TX.ALL;
35 --use WORK.COMMON_CONFIG_2RXHB_0TX.ALL;
36 --use WORK.COMMON_CONFIG_2RXHB_2TX.ALL;
37 --use WORK.COMMON_CONFIG_4RX_0TX.ALL;
38
39 -- You should not need to edit below this line!
40 package common_config is
41     -- *ALL* of the remaining code should be conditionalized only in terms of:
42     -- TX_EN, TX_EN_0, TX_EN_1, TX_EN_2, TX_EN_3, TX_CAP_NCHAN, TX_CAP_HB,
43     -- RX_EN, RX_EN_0, RX_EN_1, RX_EN_2, RX_EN_3, RX_CAP_NCHAN, RX_CAP_HB,
44     -- RX_NCO_EN, RX_CIC_EN
45
46     constant TX_EN : boolean := TX_ON;
47     constant TX_EN_0 : boolean := (TX_EN and (TX_SINGLE or TX_DUAL or TX_QUAD));
48     constant TX_EN_1 : boolean := (TX_EN and (TX_DUAL or TX_QUAD));
49     constant TX_EN_2 : boolean := (TX_EN and TX_QUAD);
50     constant TX_EN_3 : boolean := (TX_EN and TX_QUAD);
51     constant TX_CAP_HB : boolean := (TX_EN and TX_HB_ON);
52     constant TX_CAP_NCHAN : std_logic_vector(2 downto 0) := (conv_std_logic(TX_EN and TX_QUAD)
53                                     &
54                                     conv_std_logic(TX_EN
55                                     and TX_DUAL)
56                                     &
57                                     conv_std_logic(TX_EN
58                                     and TX_SINGLE));
59
60     constant RX_EN : boolean := RX_ON;
61     constant RX_EN_0 : boolean := (RX_EN and (RX_SINGLE or RX_DUAL or RX_QUAD));
62     constant RX_EN_1 : boolean := (RX_EN and (RX_DUAL or RX_QUAD));
63     constant RX_EN_2 : boolean := (RX_EN and RX_QUAD);
64     constant RX_EN_3 : boolean := (RX_EN and RX_QUAD);
65     constant RX_CAP_HB : boolean := (RX_EN and RX_HB_ON);
66     constant RX_CAP_NCHAN : std_logic_vector(2 downto 0) := (conv_std_logic(RX_EN and RX_QUAD)
67                                     &
68                                     conv_std_logic(RX_EN
69                                     and RX_DUAL)
70                                     &
71                                     conv_std_logic(RX_EN
72                                     and RX_SINGLE));
73
74     constant RX_NCO_EN : boolean := (RX_EN and RX_NCO_ON);
75     constant RX_CIC_EN : boolean := (RX_EN and RX_CIC_ON);
76 end common_config;

```

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

```

1 -----
2 -- USRP Configuration with 1 Rx (Halfband) and 1 Tx
3 -- Last Modified: 21 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC

```

```

7  --
8  --  This program is free software; you can redistribute it and/or modify
9  --  it under the terms of the GNU General Public License as published by
10 --  the Free Software Foundation; either version 2 of the License, or
11 --  (at your option) any later version.
12 --
13 --  This program is distributed in the hope that it will be useful,
14 --  but WITHOUT ANY WARRANTY; without even the implied warranty of
15 --  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
16 --  GNU General Public License for more details.
17 --
18 --  You should have received a copy of the GNU General Public License
19 --  along with this program; if not, write to the Free Software
20 --  Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22
23 library IEEE;
24 use IEEE.STD_LOGIC_1164.ALL;
25 use IEEE.STD_LOGIC_ARITH.ALL;
26
27 package common_config_lrxhb_ltx is
28     -- If TX_ON is not defined, there is *no* transmit circuitry built.
29     constant TX_ON : boolean := true;
30
31     -- Define 1 and only 1 of TX_SINGLE, TX_DUAL, and TX_QUAD to respectively
32     -- enable 1, 2, or 4 transmit channels.
33     -- [Please note that only TX_SINGLE and TX_DUAL are currently valid.]
34     constant TX_SINGLE : boolean := true;
35     constant TX_DUAL : boolean := false;
36     constant TX_QUAD : boolean := false;
37
38     -- Define TX_HB_ON to enable the transmit halfband filter.
39     -- [not implemented]
40     constant TX_HB_ON : boolean := false;
41
42     -- If RX_ON is not defined, there is *no* receive circuitry built.
43     constant RX_ON : boolean := true;
44
45     -- Define 1 and only 1 of RX_SINGLE, RX_DUAL, and TX_QUAD to respectively
46     -- enable 1, 2, or 4 receive channels.
47     constant RX_SINGLE : boolean := true;
48     constant RX_DUAL : boolean := false;
49     constant RX_QUAD : boolean := false;
50
51     -- Define RX_HB_ON to enable the receive halfband filter.
52     constant RX_HB_ON : boolean := true;
53
54     -- Define RX_NCO_ON to enable the receive Numerical Controlled Osc.
55     constant RX_NCO_ON : boolean := true;
56
57     -- Define RX_CIC_ON to enable the receive Cascaded Integrator Comb filter.
58     constant RX_CIC_ON : boolean := true;
59 end common_config_lrxhb_ltx;

```

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

```

1 -----
2 -- USRP Configuration with 2 Rx (Halfband) and 0 Tx
3 -- Last Modified: 21 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 --
8 -- This program is free software; you can redistribute it and/or modify
9 -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22
23 library IEEE;
24 use IEEE.STD_LOGIC_1164.ALL;
25 use IEEE.STD_LOGIC_ARITH.ALL;
26
27 package common_config_2rxhb_0tx is
28     -- If TX_ON is not defined, there is *no* transmit circuitry built.
29     constant TX_ON : boolean := false;
30
31     -- Define 1 and only 1 of TX_SINGL, TX_DUAL, and TX_QUAD to respectively
32     -- enable 1, 2, or 4 transmit channels.
33     -- [Please note that only TX_SINGL and TX_DUAL are currently valid.]
34     constant TX_SINGL : boolean := false;
35     constant TX_DUAL : boolean := false;
36     constant TX_QUAD : boolean := false;
37
38     -- Define TX_HB_ON to enable the transmit halfband filter.
39     -- [not implemented]
40     constant TX_HB_ON : boolean := false;
41
42     -- If RX_ON is not defined, there is *no* receive circuitry built.
43     constant RX_ON : boolean := true;
44
45     -- Define 1 and only 1 of RX_SINGL, RX_DUAL, and TX_QUAD to respectively
46     -- enable 1, 2, or 4 receive channels.
47     constant RX_SINGL : boolean := false;
48     constant RX_DUAL : boolean := true;
49     constant RX_QUAD : boolean := false;
50
51     -- Define RX_HB_ON to enable the receive halfband filter.
52     constant RX_HB_ON : boolean := true;
53
54     -- Define RX_NCO_ON to enable the receive Numerical Controlled Osc.
55     constant RX_NCO_ON : boolean := true;
56
57     -- Define RX_CIC_ON to enable the receive Cascaded Integrator Comb filter.

```

```

58     constant RX_CIC_ON : boolean := true;
59 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)

```

1  -----
2  -- USRP Configuration with 2 Rx (Halfband) and 2 Tx
3  -- Last Modified: 21 July 2010
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2010 COSMIAC
7  --
8  -- This program is free software; you can redistribute it and/or modify
9  -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22
23 library IEEE;
24 use IEEE.STD_LOGIC_1164.ALL;
25 use IEEE.STD_LOGIC_ARITH.ALL;
26
27 package common_config_2rxhb_2tx is
28     -- If TX_ON is not defined, there is *no* transmit circuitry built.
29     constant TX_ON : boolean := true;
30
31     -- Define 1 and only 1 of TX_SINGLE, TX_DUAL, and TX_QUAD to respectively
32     -- enable 1, 2, or 4 transmit channels.
33     -- [Please note that only TX_SINGLE and TX_DUAL are currently valid.]
34     constant TX_SINGLE : boolean := false;
35     constant TX_DUAL : boolean := true;
36     constant TX_QUAD : boolean := false;
37
38     -- Define TX_HB_ON to enable the transmit halfband filter.
39     -- [not implemented]
40     constant TX_HB_ON : boolean := false;
41
42     -- If RX_ON is not defined, there is *no* receive circuitry built.
43     constant RX_ON : boolean := true;
44
45     -- Define 1 and only 1 of RX_SINGLE, RX_DUAL, and TX_QUAD to respectively
46     -- enable 1, 2, or 4 receive channels.
47     constant RX_SINGLE : boolean := false;
48     constant RX_DUAL : boolean := true;
49     constant RX_QUAD : boolean := false;
50
51     -- Define RX_HB_ON to enable the receive halfband filter.

```

```

52     constant RX_HB_ON : boolean := true;
53
54     -- Define RX_NCO_ON to enable the receive Numerical Controlled Osc.
55     constant RX_NCO_ON : boolean := true;
56
57     -- Define RX_CIC_ON to enable the receive Cascaded Integrator Comb filter.
58     constant RX_CIC_ON : boolean := true;
59 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)

```

1  -----
2  -- USRP Configuration with 2 Rx and 0 Tx
3  -- Last Modified: 21 July 2010
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2010 COSMIAC
7  --
8  -- This program is free software; you can redistribute it and/or modify
9  -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22
23 library IEEE;
24 use IEEE.STD_LOGIC_1164.ALL;
25 use IEEE.STD_LOGIC_ARITH.ALL;
26
27 package common_config_2rx_0tx is
28     -- If TX_ON is not defined, there is *no* transmit circuitry built.
29     constant TX_ON : boolean := false;
30
31     -- Define 1 and only 1 of TX_SINGLE, TX_DUAL, and TX_QUAD to respectively
32     -- enable 1, 2, or 4 transmit channels.
33     -- [Please note that only TX_SINGLE and TX_DUAL are currently valid.]
34     constant TX_SINGLE : boolean := false;
35     constant TX_DUAL : boolean := false;
36     constant TX_QUAD : boolean := false;
37
38     -- Define TX_HB_ON to enable the transmit halfband filter.
39     -- [not implemented]
40     constant TX_HB_ON : boolean := false;
41
42     -- If RX_ON is not defined, there is *no* receive circuitry built.
43     constant RX_ON : boolean := true;
44
45     -- Define 1 and only 1 of RX_SINGLE, RX_DUAL, and TX_QUAD to respectively

```

```

46  -- enable 1, 2, or 4 receive channels.
47  constant RX_SINGLE : boolean := false;
48  constant RX_DUAL   : boolean := true;
49  constant RX_QUAD   : boolean := false;
50
51  -- Define RX_HB_ON to enable the receive halfband filter.
52  constant RX_HB_ON  : boolean := false;
53
54  -- Define RX_NCO_ON to enable the receive Numerical Controlled Osc.
55  constant RX_NCO_ON : boolean := true;
56
57  -- Define RX_CIC_ON to enable the receive Cascaded Integrator Comb filter.
58  constant RX_CIC_ON : boolean := true;
59  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)

```

1  -----
2  -- USRP Configuration with 4 Rx and 0 Tx
3  -- Last Modified: 21 July 2010
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2010 COSMIAC
7  --
8  -- This program is free software; you can redistribute it and/or modify
9  -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22
23 library IEEE;
24 use IEEE.STD_LOGIC_1164.ALL;
25 use IEEE.STD_LOGIC_ARITH.ALL;
26
27 package common_config_4rx_0tx is
28   -- If TX_ON is not defined, there is *no* transmit circuitry built.
29   constant TX_ON : boolean := false;
30
31   -- Define 1 and only 1 of TX_SINGLE, TX_DUAL, and TX_QUAD to respectively
32   -- enable 1, 2, or 4 transmit channels.
33   -- [Please note that only TX_SINGLE and TX_DUAL are currently valid.]
34   constant TX_SINGLE : boolean := false;
35   constant TX_DUAL   : boolean := false;
36   constant TX_QUAD   : boolean := false;
37
38   -- Define TX_HB_ON to enable the transmit halfband filter.
39   -- [not implemented]

```



```

40     constant TX_HB_ON : boolean := false;
41
42     -- If RX_ON is not defined, there is *no* receive circuitry built.
43     constant RX_ON : boolean := true;
44
45     -- Define 1 and only 1 of RX_SINGLE, RX_DUAL, and TX_QUAD to respectively
46     -- enable 1, 2, or 4 receive channels.
47     constant RX_SINGLE : boolean := false;
48     constant RX_DUAL : boolean := false;
49     constant RX_QUAD : boolean := true;
50
51     -- Define RX_HB_ON to enable the receive halfband filter.
52     constant RX_HB_ON : boolean := false;
53
54     -- Define RX_NCO_ON to enable the receive Numerical Controlled Osc.
55     constant RX_NCO_ON : boolean := true;
56
57     -- Define RX_CIC_ON to enable the receive Cascaded Integrator Comb filter.
58     constant RX_CIC_ON : boolean := true;
59 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)

```

1 -----
2 -- Cordic
3 -- Last Modified: 20 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;
31 use IEEE.STD_LOGIC_ARITH.ALL;
32 use IEEE.STD_LOGIC_SIGNED.ALL;
33

```

```

34 library WORK;
35 use WORK.FUNCS.ALL;
36
37 -- This is the first stage in the Rx chain. We move the signal from IF band
38 -- to baseband.
39 -- NOTE: The Spartan3A 1400 actually has hardware multipliers and we might be
40 -- able to use them instead of CORDIC.
41 entity cordic is
42     generic(
43         bitwidth    : integer := 16;
44         zwidth     : integer := 16
45     );
46
47     port(
48         clock       : in    std_logic;
49         reset       : in    std_logic;
50         enable      : in    std_logic;
51         xi         : in    std_logic_vector(bitwidth-1 downto 0);
52         yi         : in    std_logic_vector(bitwidth-1 downto 0);
53         xo         : out   std_logic_vector(bitwidth-1 downto 0);
54         yo         : out   std_logic_vector(bitwidth-1 downto 0);
55         zi         : in    std_logic_vector(zwidth-1 downto 0);
56         zo         : out   std_logic_vector(zwidth-1 downto 0)
57     );
58 end cordic;
59
60 architecture behavioral of cordic is
61     -- FIXME Need to somehow loop to generate constants...
62     constant STAGES : integer := 17;
63     type const_rom is array(0 to STAGES-1) of integer;
64     constant consts : const_rom := (
65         8192, 4836, 2555, 1297, 651, 326, 163, 81, 41, 20, 10, 5, 3, 1, 1, 0, 0);
66
67     component cordic_stage is
68         generic(
69             bitwidth    : integer := 16;
70             zwidth     : integer := 16;
71             shift       : integer := 1
72         );
73
74         port(
75             clock       : in    std_logic;
76             reset       : in    std_logic;
77             enable      : in    std_logic;
78             xi         : in    std_logic_vector(bitwidth-1 downto 0);
79             yi         : in    std_logic_vector(bitwidth-1 downto 0);
80             zi         : in    std_logic_vector(zwidth-1 downto 0);
81             const      : in    std_logic_vector(zwidth-1 downto 0);
82             xo         : out   std_logic_vector(bitwidth-1 downto 0);
83             yo         : out   std_logic_vector(bitwidth-1 downto 0);
84             zo         : out   std_logic_vector(zwidth-1 downto 0)
85         );
86     end component;
87
88     -- FIXME This should probably be variable, too.
89     type xy_vec is array(0 to 12) of std_logic_vector(bitwidth+1 downto 0);
90     type z_vec is array(0 to 12) of std_logic_vector(zwidth-2 downto 0);

```

```

91     signal x : xy_vec;
92     signal y : xy_vec;
93     signal z : z_vec;
94
95     signal xi_ext : std_logic_vector(bitwidth+1 downto 0);
96     signal yi_ext : std_logic_vector(bitwidth+1 downto 0);
97
98     signal zi_t : std_logic_vector(1 downto 0);
99 begin
100    -- Sign extend the inputs!
101    xi_ext <= repeat(2, xi(bitwidth-1)) & xi;
102    yi_ext <= repeat(2, yi(bitwidth-1)) & yi;
103
104    -- Quadrant
105    zi_t <= zi(zwidth-1 downto zwidth-2);
106
107    process(clock)
108    begin
109        if(clock 'EVENT and clock = '1') then
110            if(reset = '1') then
111                -- Clear the first stage inputs.
112                x(0) <= (others => '0');
113                y(0) <= (others => '0');
114                z(0) <= (others => '0');
115            elsif(enable = '1') then
116                -- The first two bits are the quadrant, so take the rest here.
117                -- Z is the phase.
118                z(0) <= zi(zwidth-2 downto 0);
119
120                -- X and Y are the I and Q components of the sample.
121                case zi_t is
122                    when "00" | "11" =>
123                        x(0) <= xi_ext;
124                        y(0) <= yi_ext;
125                    when "01" | "10" =>
126                        x(0) <= -xi_ext;
127                        y(0) <= -yi_ext;
128                    when others =>
129                        null;
130                end case;
131            end if;
132        end if;
133    end process;
134
135    -- FIXME loop should be variable, not strict 0:11.
136    -- FIXME should be able to narrow zwidth?
137    cordic_stages : for i in 0 to 11 generate
138        cs : cordic_stage
139            generic map(bitwidth+2, zwidth-1, i)
140            port map(clock => clock, reset => reset, enable => enable, xi => x(i), yi => y(i), zi
141                => z(i),
142                const => conv_std_logic_vector(consts(i), zwidth-1), xo => x(i+1), yo => y(i+1),
143                zo => z(i+1));
144    end generate cordic_stages;
145
146    xo <= x(12)(bitwidth downto 1);
147    yo <= y(12)(bitwidth downto 1);

```

```

146     zo <= '0' & z(12);
147 end behavioral;

```

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

```

1 -----
2  -- A Single Cordic Stage
3  -- Last Modified: 10 April 2011
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2010 COSMIAC
7  -- The copyright and license from the original Verilog implementation follows.
8  --
9  --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;
31 use IEEE.STD_LOGIC_ARITH.ALL;
32 use IEEE.STD_LOGIC_UNSIGNED.ALL;
33
34 library WORK;
35 use WORK.FUNCS.ALL;
36
37 entity cordic_stage is
38     generic(
39         bitwidth    : integer := 16;
40         zwidth      : integer := 16;
41         shift       : integer := 1
42     );
43
44     port(
45         clock      : in    std_logic;
46         reset      : in    std_logic;
47         enable     : in    std_logic;
48         xi         : in    std_logic_vector(bitwidth-1 downto 0);
49         yi         : in    std_logic_vector(bitwidth-1 downto 0);
50         zi         : in    std_logic_vector(zwidth-1 downto 0);
51         const      : in    std_logic_vector(zwidth-1 downto 0);

```

```

52     xo      : out   std_logic_vector(bitwidth-1 downto 0);
53     yo      : out   std_logic_vector(bitwidth-1 downto 0);
54     zo      : out   std_logic_vector(zwidth-1 downto 0)
55 );
56 end cordic_stage;
57
58 architecture behavioral of cordic_stage is
59     signal z_is_pos : std_logic;
60 begin
61     z_is_pos <= (not zi(zwidth-1));
62
63     process(clock)
64     begin
65         if(clock 'EVENT and clock = '1') then
66             if(reset = '1') then
67                 -- Clear the outputs on reset.
68                 xo <= (others => '0');
69                 yo <= (others => '0');
70                 zo <= (others => '0');
71             elsif(enable = '1') then
72                 -- Basically, sign-extend and shift the inputs, then add/subtract as necessary
73                 -- to get the outputs.
74                 if(z_is_pos = '1') then
75                     -- Z is positive, move counter-clockwise.
76                     xo <= xi - (repeat(shift+1, yi(bitwidth-1)) & yi(bitwidth-2 downto shift));
77                     yo <= yi + (repeat(shift+1, xi(bitwidth-1)) & xi(bitwidth-2 downto shift));
78                     zo <= zi - const;
79                 else
80                     -- Z is negative, move clockwise.
81                     xo <= xi + (repeat(shift+1, yi(bitwidth-1)) & yi(bitwidth-2 downto shift));
82                     yo <= yi - (repeat(shift+1, xi(bitwidth-1)) & xi(bitwidth-2 downto shift));
83                     zo <= zi + const;
84                 end if;
85             end if;
86         end if;
87     end process;
88 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)

```

1 -----
2 -- FIFO
3 -- Last Modified: 20 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or

```

```

17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;
31 use IEEE.STD_LOGIC_ARITH.ALL;
32 use IEEE.STD_LOGIC_UNSIGNED.ALL;
33
34 entity fifo is
35     generic(
36         width          : integer := 16;
37         depth          : integer := 1024;
38         addr_bits      : integer := 10
39     );
40
41     port(
42         data           : in    std_logic_vector(width-1 downto 0);
43         wrreq          : in    std_logic;
44         rdreq          : in    std_logic;
45         rdclk          : in    std_logic;
46         wrclk          : in    std_logic;
47         aclr           : in    std_logic;
48         q              : out   std_logic_vector(width-1 downto 0);
49         rdfull         : out   std_logic;
50         rdempty        : out   std_logic;
51         rdusedw        : out   std_logic_vector(addr_bits-1 downto 0);
52         wrfull         : out   std_logic;
53         wrempty        : out   std_logic;
54         wrusedw        : out   std_logic_vector(addr_bits-1 downto 0)
55     );
56 end fifo;
57
58 architecture behavioral of fifo is
59     -- Set this to false for rd_ack, true for rd_req. Choosing false here
60     -- forces the tools to use LUTs for the FIFO since we'll have asynchronous
61     -- read operations. Choosing true uses BRAMs instead! The original Verilog
62     -- source had false, though.
63     constant RD_REQ : boolean := true;
64
65     -- The actual buffer
66     type RAM is array(integer range <>) of std_logic_vector(width-1 downto 0);
67     signal mem : RAM(0 to depth-1);
68
69     -- Pointers for read and write
70     signal rdptr : std_logic_vector(addr_bits-1 downto 0);
71     signal wrptr : std_logic_vector(addr_bits-1 downto 0);
72
73     -- Provide read access to outputs with temp signals!

```

```

74     signal rdusedw_t : std_logic_vector(addr_bits-1 downto 0);
75     signal wrusedw_t : std_logic_vector(addr_bits-1 downto 0);
76 begin
77     -- Equate the temp signals with the real outputs.
78     rdusedw <= rdusedw_t;
79     wrusedw <= wrusedw_t;
80
81     -- Write one line of data, increase the write pointer.
82     process(wrclk, aclr)
83     begin
84         if(wrclk'EVENT and wrclk = '1') then
85             if(aclr = '1') then
86                 -- Reset the pointer on aclr.
87                 wrptr <= (others => '0');
88             elsif(wrreq = '1') then
89                 wrptr <= wrptr + 1;
90                 mem(conv_integer(wrptr)) <= data;
91             end if;
92         end if;
93     end process;
94
95     -- If using the read request model, output on request and increment
96     -- the read pointer.
97     rd_req_t : if(RD_REQ) generate
98         process(rdclk, aclr)
99         begin
100             if(rdclk'EVENT and rdclk = '1') then
101                 if(aclr = '1') then
102                     rdptr <= (others => '0');
103                 elsif(rdreq = '1') then
104                     rdptr <= rdptr + 1;
105                     q <= mem(conv_integer(rdptr));
106                 end if;
107             end if;
108         end process;
109     end generate rd_req_t;
110
111     -- If using the read ack model, always output and update the pointer on
112     -- the clock (ack).
113     rd_req_f : if(not RD_REQ) generate
114         process(rdclk, aclr)
115         begin
116             if(rdclk'EVENT and rdclk = '1') then
117                 if(aclr = '1') then
118                     -- Reset the pointer on aclr.
119                     rdptr <= (others => '0');
120                 elsif(rdreq = '1') then
121                     rdptr <= rdptr + 1;
122                 end if;
123             end if;
124         end process;
125
126         q <= mem(conv_integer(rdptr));
127     end generate rd_req_f;
128
129     -- Honestly, I don't know why some of these exist. The original Verilog
130     -- just said "fix these". As far as I can tell, they're never used.

```

```

131     process(wrclk)
132     begin
133         if(wrclk'EVENT and wrclk = '1') then
134             wrusedw_t <= conv_std_logic_vector(unsigned(wrptr - rdptr), wrusedw_t'LENGTH);
135         end if;
136     end process;
137
138     process(rdclk)
139     begin
140         if(rdclk'EVENT and rdclk = '1') then
141             rdusedw_t <= conv_std_logic_vector(unsigned(wrptr - rdptr), rdusedw_t'LENGTH);
142         end if;
143     end process;
144
145     wrempty <= '1' when (wrusedw_t = 0) else '0';
146     wrfull <= '1' when (wrusedw_t = (depth-1)) else '0';
147
148     rdempty <= '1' when (rdusedw_t = 0) else '0';
149     rdfull <= '1' when (rdusedw_t = (depth-1)) else '0';
150 end behavioral;

```

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

```

1 -----
2 -- FIFO (4k lines , 18-bit width)
3 -- Last Modified: 20 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;
31
32 entity fifo_4k_18 is
33     port(

```



```

34     data      : in    std_logic_vector(17 downto 0);
35     wrreq     : in    std_logic;
36     wrclk     : in    std_logic;
37     wrfull    : out   std_logic;
38     wrempty   : out   std_logic;
39     wrusedw   : out   std_logic_vector(11 downto 0);
40     q         : out   std_logic_vector(17 downto 0);
41     rdreq     : in    std_logic;
42     rdclk     : in    std_logic;
43     rdfull    : out   std_logic;
44     rdempty   : out   std_logic;
45     rdusedw   : out   std_logic_vector(11 downto 0);
46     aclr      : in    std_logic
47 );
48 end fifo_4k_18;
49
50 architecture behavioral of fifo_4k_18 is
51     component fifo is
52         generic(
53             width      : integer := 16;
54             depth      : integer := 1024;
55             addr_bits  : integer := 10
56         );
57
58         port(
59             data      : in    std_logic_vector(width-1 downto 0);
60             wrreq     : in    std_logic;
61             rdreq     : in    std_logic;
62             rdclk     : in    std_logic;
63             wrclk     : in    std_logic;
64             aclr      : in    std_logic;
65             q         : out   std_logic_vector(width-1 downto 0);
66             rdfull    : out   std_logic;
67             rdempty   : out   std_logic;
68             rdusedw   : out   std_logic_vector(addr_bits-1 downto 0);
69             wrfull    : out   std_logic;
70             wrempty   : out   std_logic;
71             wrusedw   : out   std_logic_vector(addr_bits-1 downto 0)
72         );
73     end component;
74 begin
75     fifo_4k : fifo
76         generic map(width => 18, depth => 4096, addr_bits => 12)
77         port map(data => data, wrreq => wrreq, rdreq => rdreq, rdclk => rdclk, wrclk => wrclk,
78             aclr => aclr, q => q, rdfull => rdfull, rdempty => rdempty, rdusedw => rdusedw,
79             wrfull => wrfull, wrempty => wrempty, wrusedw => wrusedw);
80 end behavioral;

```

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

```

1 -----
2 -- Definitions Common for all FPGA Configurations (Registers 0-31)
3 -- Last Modified: 21 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC

```

```

7  -- The copyright and license from the original Verilog implementation follows.
8  --
9  --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 -- This file is derived from the original Verilog file, which was automatically
30 -- generated. Someone should update the GNU Radio code to automatically
31 -- generate VHDL files, too!
32 library IEEE;
33 use IEEE.STD_LOGIC_1164.ALL;
34 use IEEE.STD_LOGIC_ARITH.ALL;
35 use IEEE.STD_LOGIC_UNSIGNED.ALL;
36
37 package fpga_regs_common is
38     -- The FPGA needs to know the rate that samples are coming from and
39     -- going to the A/D's and D/A's. div = 128e6 / sample_rate
40     constant FR_TX_SAMPLE_RATE_DIV : integer := 0;
41     constant FR_RX_SAMPLE_RATE_DIV : integer := 1;
42
43     -- 2 and 3 are defined in the ATR section
44
45     -- Master enable and reset controls
46     constant FR_MASTER_CTRL : integer := 4;
47
48     -- i/o direction registers for pins that go to daughterboards.
49     -- Setting the bit makes it an output from the FPGA to the d'board.
50     -- top 16 is mask, low 16 is value
51     constant FR_OE_0 : integer := 5;    -- slot 0
52     constant FR_OE_1 : integer := 6;
53     constant FR_OE_2 : integer := 7;
54     constant FR_OE_3 : integer := 8;
55
56     -- i/o registers for pins that go to daughterboards.
57     -- top 16 is a mask, low 16 is value
58     constant FR_IO_0 : integer := 9;    -- slot 0
59     constant FR_IO_1 : integer := 10;
60     constant FR_IO_2 : integer := 11;
61     constant FR_IO_3 : integer := 12;
62
63     constant FR_MODE : integer := 13;

```

```

64
65  -- If the corresponding bit is set, internal FPGA debug circuitry
66  -- controls the i/o pins for the associated bank of daughterboard
67  -- i/o pins. Typically used for debugging FPGA designs.
68  constant FR_DEBUG_EN : integer := 14;
69
70  -- If the corresponding bit is set, enable the automatic DC
71  -- offset correction control loop.
72  --
73  -- The 4 low bits are significant:
74  --
75  --   ADC0 = (1 << 0)
76  --   ADC1 = (1 << 1)
77  --   ADC2 = (1 << 2)
78  --   ADC3 = (1 << 3)
79  --
80  -- This control loop works if the attached daughterboard blocks DC.
81  -- Currently all daughterboards do block DC. This includes:
82  -- basic rx, dbs_rx, tv_rx, flex_xxx_rx.
83  -- DC Offset Control Loop Enable
84  constant FR_DC_OFFSET_CL_EN : integer := 15;
85
86  -- offset corrections for ADC's and DAC's (2's complement)
87  constant FR_ADC_OFFSET_0 : integer := 16;
88  constant FR_ADC_OFFSET_1 : integer := 17;
89  constant FR_ADC_OFFSET_2 : integer := 18;
90  constant FR_ADC_OFFSET_3 : integer := 19;
91
92  -----
93  -- Automatic Transmit/Receive switching
94  --
95  -- If automatic transmit/receive (ATR) switching is enabled in the
96  -- FR_ATR_CTL register, the presence or absence of data in the FPGA
97  -- transmit fifo selects between two sets of values for each of the 4
98  -- banks of daughterboard i/o pins.
99  --
100  -- Each daughterboard slot has 3 16-bit registers associated with it:
101  --   FR_ATR_MASK_*, FR_ATR_TXVAL_* and FR_ATR_RXVAL_*
102  --
103  -- FR_ATR_MASK_{0,1,2,3}:
104  --
105  -- These registers determine which of the daughterboard i/o pins are
106  -- affected by ATR switching. If a bit in the mask is set, the
107  -- corresponding i/o bit is controlled by ATR, else it's output
108  -- value comes from the normal i/o pin output register:
109  --   FR_IO_{0,1,2,3}.
110  --
111  -- FR_ATR_TXVAL_{0,1,2,3}:
112  -- FR_ATR_RXVAL_{0,1,2,3}:
113  --
114  -- If the Tx fifo contains data, then the bits from TXVAL that are
115  -- selected by MASK are output. Otherwise, the bits from RXVAL that
116  -- are selected by MASK are output.
117  constant FR_ATR_MASK_0 : integer := 20; -- slot 0
118  constant FR_ATR_TXVAL_0 : integer := 21;
119  constant FR_ATR_RXVAL_0 : integer := 22;
120

```

```

121     constant FR_ATR_MASK_1 : integer := 23; -- slot 1
122     constant FR_ATR_TXVAL_1 : integer := 24;
123     constant FR_ATR_RXVAL_1 : integer := 25;
124
125     constant FR_ATR_MASK_2 : integer := 26; -- slot 2
126     constant FR_ATR_TXVAL_2 : integer := 27;
127     constant FR_ATR_RXVAL_2 : integer := 28;
128
129     constant FR_ATR_MASK_3 : integer := 29; -- slot 3
130     constant FR_ATR_TXVAL_3 : integer := 30;
131     constant FR_ATR_RXVAL_3 : integer := 31;
132
133     -- Clock ticks to delay rising and falling edge of T/R signal
134     constant FR_ATR_TX_DELAY : integer := 2;
135     constant FR_ATR_RX_DELAY : integer := 3;
136 end fpga_regs_common;

```

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

```

1 -----
2 -- Definitions for standard FPGA build (regs 32+) and custom (64-95).
3 -- Last Modified: 21 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 -- This file is derived from the original Verilog file, which was automatically
30 -- generated. Someone should update the GNU Radio code to automatically
31 -- generate VHDL files, too!
32 library IEEE;
33 use IEEE.STD_LOGIC_1164.ALL;
34 use IEEE.STD_LOGIC_ARITH.ALL;
35 use IEEE.STD_LOGIC_UNSIGNED.ALL;
36
37 package fpga_regs_standard is

```

```

38  -- DDC / DUC
39  constant FR_INTERP_RATE : integer := 32;  -- [1,1024]
40  constant FR_DECIM_RATE : integer := 33;  -- [1,256]
41
42  -- DDC center freq
43  constant FR_RX_FREQ_0 : integer := 34;
44  constant FR_RX_FREQ_1 : integer := 35;
45  constant FR_RX_FREQ_2 : integer := 36;
46  constant FR_RX_FREQ_3 : integer := 37;
47
48  -- See below for DDC Starting Phase
49
50  -----
51  --  configure FPGA Rx mux
52  --
53  --      3                2                1
54  --  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
55  --  +-----+-----+-----+-----+-----+
56  --  |           must be zero           | Q3| I3| Q2| I2| Q1| I1| Q0| I0|Z| NCH |
57  --  +-----+-----+-----+-----+-----+
58  --
59  --  There are a maximum of 4 digital downconverters in the the FPGA.
60  --  Each DDC has two 16-bit inputs, I and Q, and two 16-bit outputs, I & Q.
61  --
62  --  DDC I inputs are specified by the two bit fields I3, I2, I1 & I0
63  --
64  --  0 = DDC input is from ADC 0
65  --  1 = DDC input is from ADC 1
66  --  2 = DDC input is from ADC 2
67  --  3 = DDC input is from ADC 3
68  --
69  --  If Z == 1, all DDC Q inputs are set to zero
70  --  If Z == 0, DDC Q inputs are specified by the two bit fields Q3, Q2, Q1 & Q0
71  --
72  --  NCH specifies the number of complex channels that are sent across
73  --  the USB. The legal values are 1, 2 or 4, corresponding to 2, 4 or
74  --  8 16-bit values.
75  constant FR_RX_MUX : integer := 38;
76
77  -----
78  --  configure FPGA Tx Mux.
79  --
80  --      3                2                1
81  --  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
82  --  +-----+-----+-----+-----+-----+
83  --  |           | DAC3 | DAC2 | DAC1 | DAC0 |0| NCH |
84  --  +-----+-----+-----+-----+-----+
85  --
86  --  NCH specifies the number of complex channels that are sent across
87  --  the USB. The legal values are 1 or 2, corresponding to 2 or 4
88  --  16-bit values.
89  --
90  --  There are two interpolators with complex inputs and outputs.
91  --  There are four DACs. (We use the DUC in each AD9862.)
92  --
93  --  Each 4-bit DACx field specifies the source for the DAC and
94  --  whether or not that DAC is enabled. Each subfield is coded

```

```

95  -- like this:
96  --
97  --   3 2 1 0
98  -- +-----+
99  -- |E| N |
100 -- +-----+
101 --
102 -- Where E is set if the DAC is enabled, and N specifies which
103 -- interpolator output is connected to this DAC.
104 --
105 -- N which interp output
106 -- ----
107 -- 0 chan 0 I
108 -- 1 chan 0 Q
109 -- 2 chan 1 I
110 -- 3 chan 1 Q
111 constant FR_TX_MUX : integer := 39;
112
113 -----
114 -- REFCLK control
115 --
116 -- Control whether a reference clock is sent to the daughterboards,
117 -- and what frequency. The refclk is sent on d'board i/o pin 0.
118 --
119 --   3           2           1
120 -- 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
121 -- +-----+-----+-----+
122 -- |           Reserved (Must be zero)           |E| DIVISOR |
123 -- +-----+-----+-----+
124 --
125 -- Bit 7 -- 1 turns on refclk, 0 allows IO use
126 -- Bits 6:0 Divider value
127 constant FR_TX_A_REFCLK : integer := 40;
128 constant FR_RX_A_REFCLK : integer := 41;
129 constant FR_TX_B_REFCLK : integer := 42;
130 constant FR_RX_B_REFCLK : integer := 43;
131
132 -----
133 -- DDC Starting Phase
134 constant FR_RX_PHASE_0 : integer := 44;
135 constant FR_RX_PHASE_1 : integer := 45;
136 constant FR_RX_PHASE_2 : integer := 46;
137 constant FR_RX_PHASE_3 : integer := 47;
138
139 -----
140 -- Tx data format control register
141 --
142 --   3           2           1
143 -- 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
144 -- +-----+-----+-----+
145 -- |           Reserved (Must be zero)           | FMT |
146 -- +-----+-----+-----+
147 --
148 -- FMT values:
149 constant FR_TX_FORMAT : integer := 48;
150
151 -----

```

```

152 -- Rx data format control register
153 --
154 --      3                2                1
155 -- 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
156 -- +-----+-----+-----+-----+
157 -- |          Reserved (Must be zero)          |B|Q| WIDTH | SHIFT |
158 -- +-----+-----+-----+-----+
159 --
160 -- FMT values :
161 constant FR_RX_FORMAT : integer := 49;
162
163 -- The valid combinations currently are :
164 --
165 --   B  Q  WIDTH  SHIFT
166 --   0  1   16    0
167 --   0  1    8    8
168
169 -- Possible future values of WIDTH = {4, 2, 1}
170 -- 12 takes a bit more work, since we need to know packet alignment.
171
172 -----
173 -- FIXME register numbers 50 to 63 are available
174
175 -----
176 -- Registers 64 to 95 are reserved for user custom FPGA builds.
177 -- The standard USRP software will not touch these.
178 constant FR_USER_0 : integer := 64;
179 constant FR_USER_1 : integer := 65;
180 constant FR_USER_2 : integer := 66;
181 constant FR_USER_3 : integer := 67;
182 constant FR_USER_4 : integer := 68;
183 constant FR_USER_5 : integer := 69;
184 constant FR_USER_6 : integer := 70;
185 constant FR_USER_7 : integer := 71;
186 constant FR_USER_8 : integer := 72;
187 constant FR_USER_9 : integer := 73;
188 constant FR_USER_10 : integer := 74;
189 constant FR_USER_11 : integer := 75;
190 constant FR_USER_12 : integer := 76;
191 constant FR_USER_13 : integer := 77;
192 constant FR_USER_14 : integer := 78;
193 constant FR_USER_15 : integer := 79;
194 constant FR_USER_16 : integer := 80;
195 constant FR_USER_17 : integer := 81;
196 constant FR_USER_18 : integer := 82;
197 constant FR_USER_19 : integer := 83;
198 constant FR_USER_20 : integer := 84;
199 constant FR_USER_21 : integer := 85;
200 constant FR_USER_22 : integer := 86;
201 constant FR_USER_23 : integer := 87;
202 constant FR_USER_24 : integer := 88;
203 constant FR_USER_25 : integer := 89;
204 constant FR_USER_26 : integer := 90;
205 constant FR_USER_27 : integer := 91;
206 constant FR_USER_28 : integer := 92;
207 constant FR_USER_29 : integer := 93;
208 constant FR_USER_30 : integer := 94;

```

```

209     constant FR_USER_31 : integer := 95;
210
211     --Registers needed for multi usrp master/slave configuration
212     --
213     --Rx Master/slave control register (FR_RX_MASTER_SLAVE = FR_USER_0)
214     --
215     constant FR_RX_MASTER_SLAVE : integer := 64;
216     constant bitnoFR_RX_SYNC : integer := 0;
217     constant bitnoFR_RX_SYNC_MASTER : integer := 1;
218     constant bitnoFR_RX_SYNC_SLAVE : integer := 2;
219
220     --Caution The master settings will output values on the io lines.
221     --They inheritely enable these lines as output. If you have a daughtercard which uses these
222     --lines also as output then you will burn your usrp and daughtercard.
223     --If you set the slave bits then your usrp won't do anything if you don't connect a master.
224     -- Rx Master/slave control register
225     --
226     -- The way this is supposed to be used is connecting a (short) 16pin flatcable from an rx
227     -- daughterboard in RXA master io_rx[8..15] to slave io_rx[8..15] on RXA of slave usrp
228     -- This can be done with basic_rx boards or dbrsx boards
229     --dbrsx: connect master-J25 to slave-J25
230     --basic rx: connect J25 to slave-J25
231     --CAUTION: pay attention to the lineup of your connector.
232     --The red line (pin1) should be at the same side of the daughterboards on master and slave.
233     --If you turnaround the cable on one end you will burn your usrp.
234
235     --You cannot use a 16pin flatcable if you are using FLEX400 or FLEX2400 daughterboards, since
236     --these use a lot of the io pins.
237     --You can still link them but you must use only a 2pin or 1pin cable
238     --You can also use a 2-wire link. put a 2pin header on io[15],gnd of the master RXA
239     --daughterboard and connect it to io15,gnd of the slave RXA db.
240     --You can use a cable like the ones found with the leds on the mainbord of a PC.
241     --Make sure you don't twist the cable, otherwise you connect the sync output to ground.
242     --To be save you could also just use a single wire from master io[15] to slave io[15], but
243     --this is not optimal for signal integrity.
244
245     -- Since rx_io[0] can normally be used as a refclk and is not exported on all daughterboards
246     -- this line
247     -- still has the refclk function if you use the master/slave setup (it is not touched by the
248     -- master/slave settings).
249     -- The master/slave circuitry will only use io pin 15 and does not touch any of the other io
250     -- pins.
251     constant bitnoFR_RX_SYNC_INPUT_IOPIN : integer := 15;
252     constant bmFR_RX_SYNC_INPUT_IOPIN : integer := 2**bitnoFR_RX_SYNC_INPUT_IOPIN; -- FIXME ?
253     --TODO the output pin is still hardcoded in the verilog code, make it listen to the following
254     --define
255     constant bitnoFR_RX_SYNC_OUTPUT_IOPIN : integer := 15;
256     constant bmFR_RX_SYNC_OUTPUT_IOPIN : integer := 2**bitnoFR_RX_SYNC_OUTPUT_IOPIN;
257
258     -- =====
259     -- READBACK Registers
260     -- =====
261     -- read back a-side i/o pins
262     constant FR_RB_IO_RX_A_IO_TX_A : integer := 1;
263     -- read back b-side i/o pins
264     constant FR_RB_IO_RX_B_IO_TX_B : integer := 2;

```



```

257
258 -----
259 -- FPGA Capability register
260 --
261 --           3           2           1
262 -- 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
263 -- +-----+-----+-----+
264 -- |                               |T|NDUC |R|NDDC |
265 -- +-----+-----+-----+
266 --
267 -- Bottom 4-bits are Rx capabilities
268 -- Next 4-bits are Tx capabilities
269 constant FR_RB_CAPS : integer := 3;
270 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)

```

1 -----
2 -- A collection of useful functions that should really be built into VHDL!
3 -- Last Modified: 21 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 --
8 -- This program is free software; you can redistribute it and/or modify
9 -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22
23 -- This file should probably not exist. I think that most of these functions
24 -- are available in numeric_std and numeric_extra, which we should eventually
25 -- use instead of std_logic_arith.
26 library IEEE;
27 use IEEE.STD_LOGIC_1164.ALL;
28 use IEEE.STD_LOGIC_ARITH.ALL;
29
30 package funcs is
31     -- Mimics the Verilog replication operator. For example, repeat(3, '1') returns "111".
32     function repeat (N : natural; B : std_logic) return std_logic_vector;
33
34     -- Converts a 'signed' logic vector into a 'std_logic_vector' logic vector of equal width.
35     function conv_std_logic_vector(S: signed) return std_logic_vector;
36
37     -- Converts a 'std_logic_vector' logic vector into a 'signed' logic vector of equal width.
38     function conv_signed(S: std_logic_vector) return signed;
39

```

```

40     -- Converts a 'boolean' to a 'std_logic' element, where 'true' = '1' and 'false' = '0'.
41     function conv_std_logic(B: boolean) return std_logic;
42 end funcns;
43
44 package body funcns is
45     function repeat(N: natural; B: std_logic)
46         return std_logic_vector
47     is
48     begin
49         return (N-1 downto 0 => B);
50     end;
51
52     function conv_std_logic_vector(S: signed)
53         return std_logic_vector
54     is
55     variable tmp : std_logic_vector(S'RANGE);
56     begin
57         for i in S'RANGE loop
58             tmp(i) := S(i);
59         end loop;
60         return tmp;
61     end;
62
63     function conv_signed(S: std_logic_vector)
64         return signed
65     is
66     variable tmp : signed(S'RANGE);
67     begin
68         for i in S'RANGE loop
69             tmp(i) := S(i);
70         end loop;
71         return tmp;
72     end;
73
74     function conv_std_logic(B: boolean)
75         return std_logic
76     is
77     begin
78         if(B) then
79             return '1';
80         else
81             return '0';
82         end if;
83     end;
84 end funcns;

```

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

```

1 -----
2 -- Halfband Decimator
3 -- Last Modified: 21 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --

```

```

9  --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28 --
29 -- This implements a 31-tap halfband filter that decimates by two.
30 -- The coefficients are symmetric, and with the exception of the middle tap,
31 -- every other coefficient is zero. The middle section of taps looks like this:
32 --
33 -- ..., -1468, 0, 2950, 0, -6158, 0, 20585, 32768, 20585, 0, -6158, 0, 2950, 0, -1468, ...
34 --
35 --
36 --
37 -- See coeff_rom.vhd for the full set. The taps are scaled relative to 32768,
38 -- thus the middle tap equals 1.0. Not counting the middle tap, there are 8
39 -- non-zero taps on each side, and they are symmetric. A naive implementation
40 -- requires a multiply for each non-zero tap. Because of symmetry, we can
41 -- replace 2 multiplies with 1 add and 1 multiply. Thus, to compute each output
42 -- sample, we need to perform 8 multiplications. Since the middle tap is 1.0,
43 -- we just add the corresponding delay line value.
44 --
45 -- About timing: We implement this with a single multiplier, so it takes
46 -- 8 cycles to compute a single output. However, since we're decimating by two
47 -- we can accept a new input value every 4 cycles. strobe_in is asserted when
48 -- there's a new input sample available. Depending on the overall decimation
49 -- rate, strobe_in may be asserted less frequently than once every 4 clocks.
50 -- On the output side, we assert strobe_out when output contains a new sample.
51 --
52 -- Implementation: Every time strobe_in is asserted we store the new data into
53 -- the delay line. We split the delay line into two components, one for the
54 -- even samples, and one for the odd samples. ram16_odd is the delay line for
55 -- the odd samples. This ram is written on each odd assertion of strobe_in, and
56 -- is read on each clock when we're computing the dot product. ram16_even is
57 -- similar, although because it holds the even samples we must be able to read
58 -- two samples from different addresses at the same time, while writing the incoming
59 -- even samples. Thus it's "triple-ported".
60 library IEEE;
61 use IEEE.STD_LOGIC_1164.ALL;
62 use IEEE.STD_LOGIC_ARITH.ALL;
63 use IEEE.STD_LOGIC_UNSIGNED.ALL;
64
65 library WORK;

```

```

66 use WORK.REDUCE_PACK.ALL;
67 use WORK.FUNCS.ALL;
68
69 entity halfband_decim is
70     port(
71         clock          : in    std_logic;
72         reset          : in    std_logic;
73         enable         : in    std_logic;
74         strobe_in      : in    std_logic;
75         strobe_out     : out   std_logic;
76         data_in        : in    std_logic_vector(15 downto 0);
77         data_out       : out   std_logic_vector(15 downto 0);
78         debugctrl     : out   std_logic_vector(15 downto 0)
79     );
80 end halfband_decim;
81
82 architecture behavioral of halfband_decim is
83     component coeff_rom is
84         port(
85             clock      : in    std_logic;
86             addr       : in    std_logic_vector(2 downto 0);
87             data       : out   std_logic_vector(15 downto 0)
88         );
89     end component;
90
91     component ram16_2sum is
92         port(
93             clock      : in    std_logic;
94             write_in   : in    std_logic;
95             wr_addr    : in    std_logic_vector(3 downto 0);
96             wr_data    : in    std_logic_vector(15 downto 0);
97             rd_addr1   : in    std_logic_vector(3 downto 0);
98             rd_addr2   : in    std_logic_vector(3 downto 0);
99             sum        : out   std_logic_vector(15 downto 0)
100        );
101     end component;
102
103     component ram16 is
104         port(
105             clock      : in    std_logic;
106             write_in   : in    std_logic;
107             wr_addr    : in    std_logic_vector(3 downto 0);
108             wr_data    : in    std_logic_vector(15 downto 0);
109             rd_addr    : in    std_logic_vector(3 downto 0);
110             rd_data    : out   std_logic_vector(15 downto 0)
111        );
112     end component;
113
114     component mult is
115         port(
116             clock      : in    std_logic;
117             x           : in    std_logic_vector(15 downto 0);
118             y           : in    std_logic_vector(15 downto 0);
119             product     : out   std_logic_vector(30 downto 0);
120             enable_in   : in    std_logic;
121             enable_out  : out   std_logic
122        );

```

```

123     end component;
124
125     component acc is
126         port(
127             clock          : in    std_logic;
128             reset          : in    std_logic;
129             clear          : in    std_logic;
130             enable_in     : in    std_logic;
131             enable_out    : out   std_logic;
132             addend        : in    std_logic_vector(30 downto 0);
133             sum           : out   std_logic_vector(33 downto 0)
134         );
135     end component;
136
137     signal rd_addr1 : std_logic_vector(3 downto 0);
138     signal rd_addr2 : std_logic_vector(3 downto 0);
139     signal phase : std_logic_vector(3 downto 0);
140     signal base_addr : std_logic_vector(3 downto 0);
141
142     -- signal mac_out : signed(15 downto 0);
143     signal middle_data : signed(15 downto 0);
144     signal sum : signed(15 downto 0);
145     signal coeff : signed(15 downto 0);
146     signal product : signed(30 downto 0);
147     signal sum_even : signed(33 downto 0);
148
149     -- VHDL doesn't support conversion operators on outputs, so we need
150     -- std_logic_vector versions of the signed signals.
151     -- signal mac_out_t : std_logic_vector(15 downto 0);
152     signal middle_data_t : std_logic_vector(15 downto 0);
153     signal sum_t : std_logic_vector(15 downto 0);
154     signal coeff_t : std_logic_vector(15 downto 0);
155     signal product_t : std_logic_vector(30 downto 0);
156     signal sum_even_t : std_logic_vector(33 downto 0);
157
158     signal clear : std_logic;
159     signal store_odd : std_logic;
160
161     signal start : std_logic;
162     signal start_d1 : std_logic;
163     signal start_d2 : std_logic;
164     signal start_d3 : std_logic;
165     signal start_d4 : std_logic;
166     signal start_d5 : std_logic;
167     signal start_d6 : std_logic;
168     signal start_d7 : std_logic;
169     signal start_d8 : std_logic;
170     signal start_d9 : std_logic;
171     signal start_dA : std_logic;
172     signal start_dB : std_logic;
173     signal start_dC : std_logic;
174     signal start_dD : std_logic;
175
176     signal mult_en : std_logic;
177     signal mult_en_pre : std_logic;
178
179     signal latch_result : std_logic;

```

```

180     signal acc_en : std_logic;
181
182     signal dout : signed(33 downto 0);
183
184     signal crom_addr : std_logic_vector(2 downto 0);
185     signal write_in_even : std_logic;
186     signal write_in_odd : std_logic;
187     signal rd_addr_odd : std_logic_vector(3 downto 0);
188
189     signal strobe_out_t : std_logic;
190 begin
191     -- Flip between odd and even with each strobe.
192     process(clock)
193     begin
194         if(clock 'EVENT and clock = '1') then
195             if(reset = '1') then
196                 store_odd <= '0';
197             elsif(strobe_in = '1') then
198                 store_odd <= (not store_odd);
199             end if;
200         end if;
201     end process;
202
203     start <= '1' when ((strobe_in = '1') and (store_odd = '1')) else '0';
204
205     --
206     process(clock)
207     begin
208         if(clock 'EVENT and clock = '1') then
209             if(reset = '1') then
210                 base_addr <= (others => '0');
211             elsif(start = '1') then
212                 base_addr <= base_addr + 1;
213             end if;
214         end if;
215     end process;
216
217     -- Phase control logic.
218     process(clock)
219     begin
220         if(clock 'EVENT and clock = '1') then
221             if(reset = '1') then
222                 phase <= conv_std_logic_vector(8, 4);
223             elsif(start = '1') then
224                 phase <= conv_std_logic_vector(0, 4);
225             elsif(phase /= 8) then
226                 phase <= phase + 1;
227             end if;
228         end if;
229     end process;
230
231     -- Recall that each signal is updated concurrently!
232     process(clock)
233     begin
234         if(clock 'EVENT and clock = '1') then
235             start_d1 <= start;
236             start_d2 <= start_d1;

```

```

237         start_d3 <= start_d2;
238         start_d4 <= start_d3;
239         start_d5 <= start_d4;
240         start_d6 <= start_d5;
241         start_d7 <= start_d6;
242         start_d8 <= start_d7;
243         start_d9 <= start_d8;
244         start_dA <= start_d9;
245         start_dB <= start_dA;
246         start_dC <= start_dB;
247         start_dD <= start_dC;
248     end if;
249 end process;
250
251 process(clock)
252 begin
253     if(clock 'EVENT and clock = '1') then
254         if(phase /= 8) then
255             mult_en_pre <= '1';
256         else
257             mult_en_pre <= '0';
258         end if;
259
260         mult_en <= mult_en_pre;
261     end if;
262 end process;
263
264 -- These comments were in the original Verilog source. I copied them here.
265 clear <= start_d4; -- was dC
266 latch_result <= start_d4; -- was dC
267 strobe_out_t <= start_d5; -- was dD
268
269 -- The two values we read depend on the phase.
270 process(phase, base_addr)
271 begin
272     case conv_integer(phase(2 downto 0)) is
273         when 0 =>
274             rd_addr1 <= base_addr + 0;
275             rd_addr2 <= base_addr + 15;
276         when 1 =>
277             rd_addr1 <= base_addr + 1;
278             rd_addr2 <= base_addr + 14;
279         when 2 =>
280             rd_addr1 <= base_addr + 2;
281             rd_addr2 <= base_addr + 13;
282         when 3 =>
283             rd_addr1 <= base_addr + 3;
284             rd_addr2 <= base_addr + 12;
285         when 4 =>
286             rd_addr1 <= base_addr + 4;
287             rd_addr2 <= base_addr + 11;
288         when 5 =>
289             rd_addr1 <= base_addr + 5;
290             rd_addr2 <= base_addr + 10;
291         when 6 =>
292             rd_addr1 <= base_addr + 6;
293             rd_addr2 <= base_addr + 9;

```

```

294         when 7 =>
295             rd_addr1 <= base_addr + 7;
296             rd_addr2 <= base_addr + 8;
297         when others =>
298             rd_addr1 <= base_addr + 0;
299             rd_addr2 <= base_addr + 15;
300     end case;
301 end process;
302
303 -- Equate the temporaries with the signed values.
304 --mac_out <= conv_signed(mac_out_t);
305 middle_data <= conv_signed(middle_data_t);
306 sum <= conv_signed(sum_t);
307 coeff <= conv_signed(coeff_t);
308 product <= conv_signed(product_t);
309 sum_even <= conv_signed(sum_even_t);
310
311 crom_addr <= phase(2 downto 0) - 1;
312
313 coeff_rom0 : coeff_rom
314     port map(clock => clock, addr => crom_addr, data => coeff_t);
315
316 write_in_even <= '1' when ((strobe_in = '1') and (store_odd = '0')) else '0';
317
318 ram16_even : ram16_2sum
319     port map(clock => clock, write_in => write_in_even, wr_addr => base_addr, wr_data =>
320         data_in,
321         rd_addr1 => rd_addr1, rd_addr2 => rd_addr2, sum => sum_t);
322
323 write_in_odd <= '1' when ((strobe_in = '1') and (store_odd = '1')) else '0';
324 rd_addr_odd <= base_addr + 6;
325
326 ram16_odd : ram16 -- holds middle items
327     port map(clock => clock, write_in => write_in_odd, wr_addr => base_addr, wr_data =>
328         data_in,
329         rd_addr => rd_addr_odd, rd_data => middle_data_t);
330
331 mult0 : mult
332     port map(clock => clock, x => coeff_t, y => sum_t,
333         product => product_t, enable_in => mult_en, enable_out => acc_en);
334
335 acc0 : acc
336     port map(clock => clock, reset => reset, enable_in => acc_en, enable_out => OPEN, clear
337         => clear,
338         addend => product_t, sum => sum_even_t);
339
340 dout <= sum_even + conv_signed((repeat(4, middle_data(15)) &
341     conv_std_logic_vector(middle_data) & repeat(14, '0')));
342
343 process(clock)
344 begin
345     if (clock 'EVENT and clock = '1') then
346         if (reset = '1') then
347             data_out <= (others => '0');
348         elsif (latch_result = '1') then
349             if ((dout(33) = '1') and (or_reduce(conv_std_logic_vector(dout(14 downto 0))) =
350                 '1')) then

```



```

346         data_out <= conv_std_logic_vector(dout(30 downto 15) + 1);
347     else
348         data_out <= conv_std_logic_vector(dout(30 downto 15));
349     end if;
350     end if;
351     end if;
352 end process;
353
354     strobe_out <= strobe_out_t;
355     debugctrl <= repeat(3, '0') & clock & reset & acc_en & mult_en & clear & latch_result &
        store_odd & strobe_in & strobe_out_t & phase;
356 end behavioral;

```

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

```

1 -----
2 -- Tri-State I/O Pins for Daughtercard Headers
3 -- Last Modified: 21 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;
31 use IEEE.STD_LOGIC_ARITH.ALL;
32 use IEEE.STD_LOGIC_UNSIGNED.ALL;
33
34 library WORK;
35 use WORK.FPGA_REGS_COMMON.ALL;
36
37 entity io_pins is
38     port(
39         io_0          : inout std_logic_vector(15 downto 0);
40         io_1          : inout std_logic_vector(15 downto 0);
41         io_2          : inout std_logic_vector(15 downto 0);

```

```

42     io_3           : inout std_logic_vector(15 downto 0);
43     reg_0          : in     std_logic_vector(15 downto 0);
44     reg_1          : in     std_logic_vector(15 downto 0);
45     reg_2          : in     std_logic_vector(15 downto 0);
46     reg_3          : in     std_logic_vector(15 downto 0);
47     clock          : in     std_logic;
48     rx_reset       : in     std_logic;
49     tx_reset       : in     std_logic;
50     serial_addr    : in     std_logic_vector(6  downto 0);
51     serial_data    : in     std_logic_vector(31 downto 0);
52     serial_strobe  : in     std_logic
53 );
54 end io_pins;
55
56 architecture behavioral of io_pins is
57     component bidir_reg is
58         port(   tristate   : inout std_logic_vector(15 downto 0);
59               oe          : in     std_logic_vector(15 downto 0);
60               reg_val     : in     std_logic_vector(15 downto 0)
61         );
62     end component;
63
64     signal io_0_oe : std_logic_vector(15 downto 0);
65     signal io_1_oe : std_logic_vector(15 downto 0);
66     signal io_2_oe : std_logic_vector(15 downto 0);
67     signal io_3_oe : std_logic_vector(15 downto 0);
68 begin
69     bidir_reg_0 : bidir_reg
70         port map( tristate => io_0, oe => io_0_oe, reg_val => reg_0);
71
72     bidir_reg_1 : bidir_reg
73         port map( tristate => io_1, oe => io_1_oe, reg_val => reg_1);
74
75     bidir_reg_2 : bidir_reg
76         port map( tristate => io_2, oe => io_2_oe, reg_val => reg_2);
77
78     bidir_reg_3 : bidir_reg
79         port map( tristate => io_3, oe => io_3_oe, reg_val => reg_3);
80
81     -- The upper 16 bits are a mask for the lower 16 bits. If a bit in the
82     -- upper 16 bits is high, set the output bit to the value of the
83     -- corresponding bit in the lower 16 bits. Otherwise, do nothing.
84     process(clock)
85     begin
86         if(clock'EVENT and clock = '1') then
87             if(serial_strobe = '1') then
88                 case conv_integer(serial_addr) is
89                     when FR_OE_0 =>
90                         io_0_oe <= ((io_0_oe and (not serial_data(31 downto 16))) or
91                                     (serial_data(15 downto 0) and serial_data(31 downto 16)));
92                     when FR_OE_1 =>
93                         io_1_oe <= ((io_1_oe and (not serial_data(31 downto 16))) or
94                                     (serial_data(15 downto 0) and serial_data(31 downto 16)));
95                     when FR_OE_2 =>
96                         io_2_oe <= ((io_2_oe and (not serial_data(31 downto 16))) or
97                                     (serial_data(15 downto 0) and serial_data(31 downto 16)));
98                     when FR_OE_3 =>

```

```

99             io_3_oe <= ((io_3_oe and (not serial_data(31 downto 16))) or
100                (serial_data(15 downto 0) and serial_data(31 downto 16)));
101             when others =>
102                 null;
103             end case;
104         end if;
105     end if;
106 end process;
107 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)

```

1 -----
2 -- Master Control System (clocks, resets, enables, etc.)
3 -- Last Modified: 21 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;
31 use IEEE.STD_LOGIC_ARITH.ALL;
32 use IEEE.STD_LOGIC_UNSIGNED.ALL;
33
34 library WORK;
35 use WORK.FPGA_REGS_COMMON.ALL;
36 use WORK.FPGA_REGS_STANDARD.ALL;
37 use WORK.FUNCS.ALL;
38
39 entity master_control is
40     port(
41         master_clk      : in    std_logic;
42         usbclk          : in    std_logic;
43         serial_addr     : in    std_logic_vector(6 downto 0);
44         serial_data     : in    std_logic_vector(31 downto 0);

```

```

45     serial_strobe      : in    std_logic;
46     tx_bus_reset      : out   std_logic;
47     rx_bus_reset      : out   std_logic;
48     tx_dsp_reset      : out   std_logic;
49     rx_dsp_reset      : out   std_logic;
50     enable_tx         : out   std_logic;
51     enable_rx         : out   std_logic;
52     interp_rate       : out   std_logic_vector(7 downto 0);
53     decim_rate        : out   std_logic_vector(7 downto 0);
54     tx_sample_strobe  : out   std_logic;
55     strobe_interp     : out   std_logic;
56     rx_sample_strobe  : out   std_logic;
57     strobe_decim      : out   std_logic;
58     tx_empty          : in    std_logic;
59     debug_0           : in    std_logic_vector(15 downto 0);
60     debug_1           : in    std_logic_vector(15 downto 0);
61     debug_2           : in    std_logic_vector(15 downto 0);
62     debug_3           : in    std_logic_vector(15 downto 0);
63     reg_0             : out   std_logic_vector(15 downto 0);
64     reg_1             : out   std_logic_vector(15 downto 0);
65     reg_2             : out   std_logic_vector(15 downto 0);
66     reg_3             : out   std_logic_vector(15 downto 0);
67 );
68 end master_control;
69
70 architecture behavioral of master_control is
71     component setting_reg is
72         generic(
73             my_addr : integer := 0
74         );
75
76         port(
77             clock      : in    std_logic;
78             reset      : in    std_logic;
79             strobe     : in    std_logic;
80             addr       : in    std_logic_vector;
81             d_in       : in    std_logic_vector(31 downto 0);
82             d_out      : out   std_logic_vector(31 downto 0);
83             changed    : out   std_logic
84         );
85     end component;
86
87     component strobe_gen is
88         port(
89             clock      : in    std_logic;
90             reset      : in    std_logic;
91             enable     : in    std_logic;
92             rate       : in    std_logic_vector(7 downto 0);
93             strobe_in  : in    std_logic;
94             strobe     : out   std_logic
95         );
96     end component;
97
98     component clk_divider is
99         port(
100            reset      : in    std_logic;
101            in_clk     : in    std_logic;

```

```

102         out_clk : out   std_logic;
103         ratio    : in    std_logic_vector(7 downto 0)
104     );
105 end component;
106
107 component atr_delay is
108     port(
109         clk_i      : in    std_logic;
110         rst_i      : in    std_logic;
111         ena_i      : in    std_logic;
112         tx_empty_i : in    std_logic;
113         tx_delay_i : in    std_logic_vector(11 downto 0);
114         rx_delay_i : in    std_logic_vector(11 downto 0);
115         atr_tx_o   : out   std_logic
116     );
117 end component;
118
119 -- FIXME need a separate reset for all control settings
120 signal master_controls : std_logic_vector(7 downto 0);
121
122 -- Internals so we can read/write.
123 signal tx_dsp_reset_t : std_logic;
124 signal rx_dsp_reset_t : std_logic;
125 signal dsp_reset_t   : std_logic;
126 signal enable_tx_t   : std_logic;
127 signal enable_rx_t   : std_logic;
128
129 signal interp_rate_t : std_logic_vector(7 downto 0);
130 signal decim_rate_t  : std_logic_vector(7 downto 0);
131 signal tx_sample_strobe_t : std_logic;
132 signal rx_sample_strobe_t : std_logic;
133
134 -- USB reset signals
135 signal tx_reset_bus_sync1 : std_logic;
136 signal rx_reset_bus_sync1 : std_logic;
137 signal tx_reset_bus_sync2 : std_logic;
138 signal rx_reset_bus_sync2 : std_logic;
139
140 -- Internal clocks
141 signal txa_refclk : std_logic_vector(7 downto 0);
142 signal rxa_refclk : std_logic_vector(7 downto 0);
143 signal txb_refclk : std_logic_vector(7 downto 0);
144 signal rxb_refclk : std_logic_vector(7 downto 0);
145
146 signal txack : std_logic;
147 signal rxack : std_logic;
148 signal txbclk : std_logic;
149 signal rxbclk : std_logic;
150
151 -- ???
152 signal debug_en : std_logic_vector(3 downto 0);
153 signal txcvr_ctrl : std_logic_vector(3 downto 0);
154
155 signal txcvr_txlines : std_logic_vector(31 downto 0);
156 signal txcvr_rxlines : std_logic_vector(31 downto 0);
157
158 -- Internal signals for daughtercard headers

```

```

159     signal io_0_reg : std_logic_vector(15 downto 0);
160     signal io_1_reg : std_logic_vector(15 downto 0);
161     signal io_2_reg : std_logic_vector(15 downto 0);
162     signal io_3_reg : std_logic_vector(15 downto 0);
163
164     -- ATR stuff (auto transmit/receive switch)
165     signal transmit_now : std_logic;
166     signal atr_ctl : std_logic;
167     signal atr_tx_delay : std_logic_vector(11 downto 0);
168     signal atr_rx_delay : std_logic_vector(11 downto 0);
169     signal atr_mask_0 : std_logic_vector(15 downto 0);
170     signal atr_txval_0 : std_logic_vector(15 downto 0);
171     signal atr_rxval_0 : std_logic_vector(15 downto 0);
172     signal atr_mask_1 : std_logic_vector(15 downto 0);
173     signal atr_txval_1 : std_logic_vector(15 downto 0);
174     signal atr_rxval_1 : std_logic_vector(15 downto 0);
175     signal atr_mask_2 : std_logic_vector(15 downto 0);
176     signal atr_txval_2 : std_logic_vector(15 downto 0);
177     signal atr_rxval_2 : std_logic_vector(15 downto 0);
178     signal atr_mask_3 : std_logic_vector(15 downto 0);
179     signal atr_txval_3 : std_logic_vector(15 downto 0);
180     signal atr_rxval_3 : std_logic_vector(15 downto 0);
181
182     signal atr_selected_0 : std_logic_vector(15 downto 0);
183     signal atr_selected_1 : std_logic_vector(15 downto 0);
184     signal atr_selected_2 : std_logic_vector(15 downto 0);
185     signal atr_selected_3 : std_logic_vector(15 downto 0);
186
187     -- More daughtercard header signals?
188     signal io_0 : std_logic_vector(15 downto 0);
189     signal io_1 : std_logic_vector(15 downto 0);
190     signal io_2 : std_logic_vector(15 downto 0);
191     signal io_3 : std_logic_vector(15 downto 0);
192 begin
193     -- Assign internal signals to their outputs.
194     enable_tx <= enable_tx_t;
195     enable_rx <= enable_rx_t;
196     tx_dsp_reset <= tx_dsp_reset_t;
197     rx_dsp_reset <= rx_dsp_reset_t;
198     dsp_reset_t <= (tx_dsp_reset_t or rx_dsp_reset_t);
199     interp_rate <= interp_rate_t;
200     decim_rate <= decim_rate_t;
201     tx_sample_strobe <= tx_sample_strobe_t;
202     rx_sample_strobe <= rx_sample_strobe_t;
203
204     -- Master control assignments.
205     sr_mstr_ctrl : setting_reg
206         generic map(my_addr => FR_MASTER_CTRL)
207         port map(clock => master_clk, reset => '0', strobe => serial_strobe,
208             addr => serial_addr, d_in => serial_data, d_out(31 downto 8) => OPEN,
209             d_out(7 downto 0) => master_controls, changed => OPEN);
210
211     -- Control signals 4:7 are not used.
212     enable_tx_t <= master_controls(0);
213     enable_rx_t <= master_controls(1);
214     tx_dsp_reset_t <= master_controls(2);
215     rx_dsp_reset_t <= master_controls(3);

```

```

216
217 -- Strobe Generators
218 sr_interp : setting_reg
219     generic map(my_addr => FR_INTERP_RATE)
220     port map(clock => master_clk, reset => tx_dsp_reset_t, strobe => serial_strobe,
221             addr => serial_addr, d_in => serial_data, d_out(31 downto 8) => OPEN,
222             d_out(7 downto 0) => interp_rate_t, changed => OPEN);
223
224 sr_decim : setting_reg
225     generic map(my_addr => FR_DECIM_RATE)
226     port map(clock => master_clk, reset => rx_dsp_reset_t, strobe => serial_strobe,
227             addr => serial_addr, d_in => serial_data, d_out(31 downto 8) => OPEN,
228             d_out(7 downto 0) => decim_rate_t, changed => OPEN);
229
230 rx_sample_strobe_t <= '1';
231
232 da_strobe_gen : strobe_gen
233     port map(clock => master_clk, reset => tx_dsp_reset_t, enable => enable_tx_t,
234             rate => (others => '1'), strobe_in => '1', strobe => tx_sample_strobe_t);
235
236 tx_strobe_gen : strobe_gen
237     port map(clock => master_clk, reset => tx_dsp_reset_t, enable => enable_tx_t,
238             rate => interp_rate_t, strobe_in => tx_sample_strobe_t, strobe => strobe_interp);
239
240 decim_strobe_gen : strobe_gen
241     port map(clock => master_clk, reset => rx_dsp_reset_t, enable => enable_rx_t,
242             rate => decim_rate_t, strobe_in => rx_sample_strobe_t, strobe => strobe_decim);
243
244 -- Reset syncs for bus (usbclk) side
245 -- The RX bus side reset isn't used, the TX bus side one may not be needed.
246 process(usbclk)
247 begin
248     if(usbclk 'EVENT and usbclk = '1') then
249         tx_reset_bus_sync1 <= tx_dsp_reset_t;
250         rx_reset_bus_sync1 <= rx_dsp_reset_t;
251         tx_reset_bus_sync2 <= tx_reset_bus_sync1;
252         rx_reset_bus_sync2 <= rx_reset_bus_sync1;
253     end if;
254 end process;
255
256 tx_bus_reset <= tx_reset_bus_sync2;
257 rx_bus_reset <= rx_reset_bus_sync2;
258
259 sr_txaref : setting_reg
260     generic map(my_addr => FR_TX_A_REFCLK)
261     port map(clock => master_clk, reset => tx_dsp_reset_t, strobe => serial_strobe,
262             addr => serial_addr, d_in => serial_data, d_out(31 downto 8) => OPEN,
263             d_out(7 downto 0) => txa_refclk, changed => OPEN);
264
265 sr_rxaref : setting_reg
266     generic map(my_addr => FR_RX_A_REFCLK)
267     port map(clock => master_clk, reset => rx_dsp_reset_t, strobe => serial_strobe,
268             addr => serial_addr, d_in => serial_data, d_out(31 downto 8) => OPEN,
269             d_out(7 downto 0) => rxa_refclk, changed => OPEN);
270
271 sr_txbref : setting_reg
272     generic map(my_addr => FR_TX_B_REFCLK)

```

```

273     port map(clock => master_clk, reset => tx_dsp_reset_t, strobe => serial_strobe,
274             addr => serial_addr, d_in => serial_data, d_out(31 downto 8) => OPEN,
275             d_out(7 downto 0) => txb_refclk, changed => OPEN);
276
277 sr_rxbref : setting_reg
278     generic map(my_addr => FR_RX_B_REFCLK)
279     port map(clock => master_clk, reset => rx_dsp_reset_t, strobe => serial_strobe,
280             addr => serial_addr, d_in => serial_data, d_out(31 downto 8) => OPEN,
281             d_out(7 downto 0) => rxb_refclk, changed => OPEN);
282
283 sr_debugen : setting_reg
284     generic map(my_addr => FR_DEBUG_EN)
285     port map(clock => master_clk, reset => dsp_reset_t, strobe => serial_strobe,
286             addr => serial_addr, d_in => serial_data, d_out(31 downto 4) => OPEN,
287             d_out(3 downto 0) => debug_en, changed => OPEN);
288
289 clk_div_0 : clk_divider
290     port map(reset => tx_dsp_reset_t, in_clk => master_clk, out_clk => txack,
291             ratio(7) => '0', ratio(6 downto 0) => txa_refclk(6 downto 0));
292
293 clk_div_1 : clk_divider
294     port map(reset => rx_dsp_reset_t, in_clk => master_clk, out_clk => rxack,
295             ratio(7) => '0', ratio(6 downto 0) => rxa_refclk(6 downto 0));
296
297 clk_div_2 : clk_divider
298     port map(reset => tx_dsp_reset_t, in_clk => master_clk, out_clk => txbclk,
299             ratio(7) => '0', ratio(6 downto 0) => txb_refclk(6 downto 0));
300
301 clk_div_3 : clk_divider
302     port map(reset => rx_dsp_reset_t, in_clk => master_clk, out_clk => rxbclk,
303             ratio(7) => '0', ratio(6 downto 0) => rxb_refclk(6 downto 0));
304
305 -- The upper 16 bits are a mask for the lower 16 bits.
306 process(master_clk)
307 begin
308     if(master_clk'EVENT and master_clk = '1') then
309         if(serial_strobe = '1') then
310             case conv_integer(serial_addr) is
311                 when FR_IO_0 =>
312                     io_0_reg <= ((io_0_reg and (not serial_data(31 downto 16))) or
313                                 (serial_data(15 downto 0) and serial_data(31 downto 16)));
314                 when FR_IO_1 =>
315                     io_1_reg <= ((io_1_reg and (not serial_data(31 downto 16))) or
316                                 (serial_data(15 downto 0) and serial_data(31 downto 16)));
317                 when FR_IO_2 =>
318                     io_2_reg <= ((io_2_reg and (not serial_data(31 downto 16))) or
319                                 (serial_data(15 downto 0) and serial_data(31 downto 16)));
320                 when FR_IO_3 =>
321                     io_3_reg <= ((io_3_reg and (not serial_data(31 downto 16))) or
322                                 (serial_data(15 downto 0) and serial_data(31 downto 16)));
323                 when others =>
324                     end case;
325             end if;
326         end if;
327     end process;
328
329 -- ATR stuff

```



```

330 sr_atr_mask_0 : setting_reg
331     generic map(my_addr => FR_ATR_MASK_0)
332     port map(clock => master_clk, reset => '0', strobe => serial_strobe,
333             addr => serial_addr, d_in => serial_data, d_out(31 downto 16) => OPEN,
334             d_out(15 downto 0) => atr_mask_0, changed => OPEN);
335
336 sr_atr_txval_0 : setting_reg
337     generic map(my_addr => FR_ATR_TXVAL_0)
338     port map(clock => master_clk, reset => '0', strobe => serial_strobe,
339             addr => serial_addr, d_in => serial_data, d_out(31 downto 16) => OPEN,
340             d_out(15 downto 0) => atr_txval_0, changed => OPEN);
341
342 sr_atr_rxval_0 : setting_reg
343     generic map(my_addr => FR_ATR_RXVAL_0)
344     port map(clock => master_clk, reset => '0', strobe => serial_strobe,
345             addr => serial_addr, d_in => serial_data, d_out(31 downto 16) => OPEN,
346             d_out(15 downto 0) => atr_rxval_0, changed => OPEN);
347
348 sr_atr_mask_1 : setting_reg
349     generic map(my_addr => FR_ATR_MASK_1)
350     port map(clock => master_clk, reset => '0', strobe => serial_strobe,
351             addr => serial_addr, d_in => serial_data, d_out(31 downto 16) => OPEN,
352             d_out(15 downto 0) => atr_mask_1, changed => OPEN);
353
354 sr_atr_txval_1 : setting_reg
355     generic map(my_addr => FR_ATR_TXVAL_1)
356     port map(clock => master_clk, reset => '0', strobe => serial_strobe,
357             addr => serial_addr, d_in => serial_data, d_out(31 downto 16) => OPEN,
358             d_out(15 downto 0) => atr_txval_1, changed => OPEN);
359
360 sr_atr_rxval_1 : setting_reg
361     generic map(my_addr => FR_ATR_RXVAL_1)
362     port map(clock => master_clk, reset => '0', strobe => serial_strobe,
363             addr => serial_addr, d_in => serial_data, d_out(31 downto 16) => OPEN,
364             d_out(15 downto 0) => atr_rxval_1, changed => OPEN);
365
366 sr_atr_mask_2 : setting_reg
367     generic map(my_addr => FR_ATR_MASK_2)
368     port map(clock => master_clk, reset => '0', strobe => serial_strobe,
369             addr => serial_addr, d_in => serial_data, d_out(31 downto 16) => OPEN,
370             d_out(15 downto 0) => atr_mask_2, changed => OPEN);
371
372 sr_atr_txval_2 : setting_reg
373     generic map(my_addr => FR_ATR_TXVAL_2)
374     port map(clock => master_clk, reset => '0', strobe => serial_strobe,
375             addr => serial_addr, d_in => serial_data, d_out(31 downto 16) => OPEN,
376             d_out(15 downto 0) => atr_txval_2, changed => OPEN);
377
378 sr_atr_rxval_2 : setting_reg
379     generic map(my_addr => FR_ATR_RXVAL_2)
380     port map(clock => master_clk, reset => '0', strobe => serial_strobe,
381             addr => serial_addr, d_in => serial_data, d_out(31 downto 16) => OPEN,
382             d_out(15 downto 0) => atr_rxval_2, changed => OPEN);
383
384 sr_atr_mask_3 : setting_reg
385     generic map(my_addr => FR_ATR_MASK_3)
386     port map(clock => master_clk, reset => '0', strobe => serial_strobe,

```

```

387         addr => serial_addr , d_in => serial_data , d_out(31 downto 16) => OPEN,
388         d_out(15 downto 0) => atr_mask_3 , changed => OPEN);
389
390 sr_atr_txval_3 : setting_reg
391     generic map(my_addr => FR_ATR_TXVAL_3)
392     port map(clock => master_clk , reset => '0' , strobe => serial_strobe ,
393             addr => serial_addr , d_in => serial_data , d_out(31 downto 16) => OPEN,
394             d_out(15 downto 0) => atr_txval_3 , changed => OPEN);
395
396 sr_atr_rxval_3 : setting_reg
397     generic map(my_addr => FR_ATR_RXVAL_3)
398     port map(clock => master_clk , reset => '0' , strobe => serial_strobe ,
399             addr => serial_addr , d_in => serial_data , d_out(31 downto 16) => OPEN,
400             d_out(15 downto 0) => atr_rxval_3 , changed => OPEN);
401
402 -- sr_atr_ctl : setting_reg
403 --     generic map(my_addr => FR_ATR_CTL)
404 --     port map(clock => master_clk , reset => '0' , strobe => serial_strobe ,
405 --             addr => serial_addr , d_in => serial_data , d_out(31 downto 1) => OPEN,
406 --             d_out(0) => atr_ctl);
407
408 sr_atr_tx_delay : setting_reg
409     generic map(my_addr => FR_ATR_TX_DELAY)
410     port map(clock => master_clk , reset => '0' , strobe => serial_strobe ,
411             addr => serial_addr , d_in => serial_data , d_out(31 downto 12) => OPEN,
412             d_out(11 downto 0) => atr_tx_delay , changed => OPEN);
413
414 sr_atr_rx_delay : setting_reg
415     generic map(my_addr => FR_ATR_RX_DELAY)
416     port map(clock => master_clk , reset => '0' , strobe => serial_strobe ,
417             addr => serial_addr , d_in => serial_data , d_out(31 downto 12) => OPEN,
418             d_out(11 downto 0) => atr_rx_delay , changed => OPEN);
419
420 atr_ctl <= '1';
421
422 atr_delay_0 : atr_delay
423     port map(clk_i => master_clk , rst_i => tx_dsp_reset_t , ena_i => atr_ctl ,
424             tx_empty_i => tx_empty , tx_delay_i => atr_tx_delay , rx_delay_i => atr_rx_delay ,
425             atr_tx_o => transmit_now);
426
427 atr_selected_0 <= atr_txval_0 when transmit_now = '1' else atr_rxval_0;
428 atr_selected_1 <= atr_txval_1 when transmit_now = '1' else atr_rxval_1;
429 atr_selected_2 <= atr_txval_2 when transmit_now = '1' else atr_rxval_2;
430 atr_selected_3 <= atr_txval_3 when transmit_now = '1' else atr_rxval_3;
431
432 io_0 <= (repeat(16, atr_ctl) and atr_mask_0 and atr_selected_0) or ((not (repeat(16, atr_ctl)
433     and atr_mask_0)) and io_0_reg);
434 io_1 <= (repeat(16, atr_ctl) and atr_mask_1 and atr_selected_1) or ((not (repeat(16, atr_ctl)
435     and atr_mask_1)) and io_1_reg);
436 io_2 <= (repeat(16, atr_ctl) and atr_mask_2 and atr_selected_2) or ((not (repeat(16, atr_ctl)
437     and atr_mask_2)) and io_2_reg);
438 io_3 <= (repeat(16, atr_ctl) and atr_mask_3 and atr_selected_3) or ((not (repeat(16, atr_ctl)
439     and atr_mask_3)) and io_3_reg);
440
441 reg_0 <= debug_0 when debug_en(0) = '1' else (io_0(15 downto 1) & txack) when txa_refclk(7)
442     = '1' else io_0;

```

```

438     reg_1 <= debug_1 when debug_en(1) = '1' else (io_1(15 downto 1) & rxack) when rxa_refclk(7)
         = '1' else io_1;
439     reg_2 <= debug_2 when debug_en(2) = '1' else (io_2(15 downto 1) & txbclk) when txb_refclk(7)
         = '1' else io_2;
440     reg_3 <= debug_3 when debug_en(3) = '1' else (io_3(15 downto 1) & rxbclk) when rxb_refclk(7)
         = '1' else io_3;
441 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)

```

1  -----
2  -- 16x16 Multiplier
3  -- Last Modified: 21 July 2010
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2010 COSMIAC
7  -- The copyright and license from the original Verilog implementation follows.
8  --
9  --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;
31 use IEEE.STD_LOGIC_ARITH.ALL;
32 use IEEE.STD_LOGIC_SIGNED.ALL;
33
34 library WORK;
35 use WORK.FUNCS.ALL;
36
37 entity mult is
38     port(
39         clock          : in    std_logic;
40         x              : in    std_logic_vector(15 downto 0);
41         y              : in    std_logic_vector(15 downto 0);
42         product        : out   std_logic_vector(30 downto 0);
43         enable_in      : in    std_logic;
44         enable_out     : out   std_logic
45     );
46 end mult;

```

```

47
48 architecture behavioral of mult is
49     signal x_t : signed(15 downto 0);
50     signal y_t : signed(15 downto 0);
51
52     -- I wonder if the original Verilog was in error using 30:0.
53     signal product_t : signed(31 downto 0);
54 begin
55     x_t <= conv_signed(conv_integer(x), x_t'LENGTH);
56     y_t <= conv_signed(conv_integer(y), y_t'LENGTH);
57     product <= conv_std_logic_vector(product_t(30 downto 0), product'LENGTH);
58
59     -- Multiple on each clock cycle, if enabled.
60     process(clock)
61     begin
62         if(clock'EVENT and clock = '1') then
63             if(enable_in = '1') then
64                 product_t <= x_t * y_t;
65             else
66                 product_t <= (others => '0');
67             end if;
68         end if;
69     end process;
70
71     -- I guess this lets us know if a result is valid?
72     process(clock)
73     begin
74         if(clock'EVENT and clock = '1') then
75             enable_out <= enable_in;
76         end if;
77     end process;
78 end behavioral;

```

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

```

1 -----
2 -- Basic Phase Accumulator (for DDS)
3 -- Last Modified: 20 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

```

```

22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;
31 use IEEE.STD_LOGIC_ARITH.ALL;
32 use IEEE.STD_LOGIC_UNSIGNED.ALL;
33
34 entity phase_acc is
35     generic(
36         FREQADDR      : integer := 0;
37         PHASEADDR     : integer := 0;
38         resolution    : integer range 0 to 32 := 32
39     );
40
41     port(
42         clk           : in    std_logic;
43         reset        : in    std_logic;
44         enable       : in    std_logic;
45         strobe       : in    std_logic;
46         serial_addr  : in    std_logic_vector(6 downto 0);
47         serial_data  : in    std_logic_vector(31 downto 0);
48         serial_strobe : in    std_logic;
49         phase        : out   std_logic_vector(resolution-1 downto 0)
50     );
51 end phase_acc;
52
53 architecture behavioral of phase_acc is
54     component setting_reg is
55         generic(
56             my_addr : integer := 0
57         );
58
59         port(
60             clock      : in    std_logic;
61             reset      : in    std_logic;
62             strobe     : in    std_logic;
63             addr       : in    std_logic_vector;
64             d_in       : in    std_logic_vector(31 downto 0);
65             d_out      : out   std_logic_vector(31 downto 0);
66             changed   : out   std_logic
67         );
68     end component;
69
70     signal freq : std_logic_vector(resolution-1 downto 0);
71     signal phase_t : std_logic_vector(resolution-1 downto 0);
72 begin
73     phase <= phase_t;
74
75     -- Note, d_out => freq only works when resolution = 32. Else,
76     -- you'll get a width mismatch. Unfortunately, we can't use
77     -- a generic value like resolution to wire part of d_out to OPEN
78     -- because Xilinx throws an error, "Discrete range on slice is not

```

```

79  -- locally static."
80  sr_rxfreq0 : setting_reg
81  generic map(my_addr => FREQADDR)
82  port map(clock => clk, reset => '0', strobe => serial_strobe, addr => serial_addr,
83  d_in => serial_data, d_out => freq, changed => OPEN);
84
85  process(clk)
86  begin
87  if(clk 'EVENT and clk = '1') then
88  if(reset = '1') then
89  -- Reset the accumulator.
90  phase_t <= (others => '0');
91  elsif((serial_strobe = '1') and (serial_addr = PHASEADDR)) then
92  -- Initial phase comes from serial input.
93  phase_t <= serial_data;
94  elsif((enable = '1') and (strobe = '1')) then
95  -- Accumulate on each strobe, where 'freq' is the center frequency.
96  phase_t <= phase_t + freq;
97  end if;
98  end if;
99  end process;
100 end behavioral;

```

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

```

1  -----
2  -- RAM16
3  -- Last Modified: 21 July 2010
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2010 COSMIAC
7  -- The copyright and license from the original Verilog implementation follows.
8  --
9  --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;
31 use IEEE.STD_LOGIC_ARITH.ALL;

```

```

32 use IEEE.STD_LOGIC_UNSIGNED.ALL;
33
34 entity ram16 is
35     port(
36         clock          : in    std_logic;
37         write_in       : in    std_logic;
38         wr_addr        : in    std_logic_vector(3 downto 0);
39         wr_data        : in    std_logic_vector(15 downto 0);
40         rd_addr        : in    std_logic_vector(3 downto 0);
41         rd_data        : out   std_logic_vector(15 downto 0)
42     );
43 end ram16;
44
45 architecture behavioral of ram16 is
46     -- Create a 16x16 bit RAM.
47     type ram is array(integer range <>) of std_logic_vector(15 downto 0);
48     signal ram_array : ram(0 to 15);
49 begin
50     -- Read from RAM on every clock.
51     process(clock)
52     begin
53         if(clock 'EVENT and clock = '1') then
54             rd_data <= ram_array(conv_integer(rd_addr));
55         end if;
56     end process;
57
58     -- Write to RAM only when write_in is asserted.
59     process(clock)
60     begin
61         if(clock 'EVENT and clock = '1') then
62             if(write_in = '1') then
63                 ram_array(conv_integer(wr_addr)) <= wr_data;
64             end if;
65         end if;
66     end process;
67 end behavioral;

```

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

```

1 -----
2 -- RAM16_2SUM
3 -- Last Modified: 14 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.

```

```

18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 -- TODO: Convert to the newer numeric_std library.
30 -- "Tri-port" RAM (2 reads, 1 write). The output is the sum of the two read
31 -- values. Sort of.
32 library IEEE;
33 use IEEE.STD_LOGIC_1164.ALL;
34 use IEEE.STD_LOGIC_ARITH.ALL;
35 use IEEE.STD_LOGIC_SIGNED.ALL;
36
37 entity ram16_2sum is
38     port(
39         clock      : in    std_logic;
40         write_in   : in    std_logic;
41         wr_addr   : in    std_logic_vector(3 downto 0);
42         wr_data   : in    std_logic_vector(15 downto 0);
43         rd_addr1  : in    std_logic_vector(3 downto 0);
44         rd_addr2  : in    std_logic_vector(3 downto 0);
45         sum       : out   std_logic_vector(15 downto 0)
46     );
47 end ram16_2sum;
48
49 architecture behavioral of ram16_2sum is
50     -- Create a 16x16 bit RAM.
51     type ram is array (integer range <>) of signed(15 downto 0);
52     signal ram_array : ram(0 to 15);
53
54     signal a : signed(15 downto 0);
55     signal b : signed(15 downto 0);
56     signal sum_int : signed(16 downto 0);
57 begin
58     -- Only write to RAM if write_in is high.
59     process(clock)
60     begin
61         if(clock 'EVENT and clock = '1') then
62             if(write_in = '1') then
63                 -- Isn't this horrible? I think it's horrible. VHDL needs to fix this.
64                 ram_array(conv_integer(conv_unsigned(conv_integer(wr_addr), wr_addr'LENGTH))) <=
65                     conv_signed(conv_integer(wr_data), wr_data'LENGTH);
66             end if;
67         end if;
68     end process;
69
70     -- Read two values from RAM on every clock cycle.
71     process(clock)
72     begin
73         if(clock 'EVENT and clock = '1') then
74             a <= ram_array(conv_integer(conv_unsigned(conv_integer(rd_addr1), rd_addr1'LENGTH)));

```



```

74         b <= ram_array(conv_integer(conv_unsigned(conv_integer(rd_addr2), rd_addr2'LENGTH)));
75     end if;
76 end process;
77
78 -- Signed addition of the two read values.
79 sum_int <= (a(15) & a) + (b(15) & b);
80
81 -- Output the sum on every clock cycle. If the sum is negative and odd, add 1.
82 process(clock)
83 begin
84     if(clock'EVENT and clock = '1') then
85         if((sum_int(16) = '1') and (sum_int(0) = '1')) then
86             sum <= conv_std_logic_vector(conv_signed(sum_int(16 downto 1) + 1, sum'LENGTH),
87                 sum'LENGTH);
88         else
89             sum <= conv_std_logic_vector(conv_signed(sum_int(16 downto 1), sum'LENGTH),
90                 sum'LENGTH);
91         end if;
92     end if;
93 end process;
94 end behavioral;

```

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

```

1 -----
2 -- Receive Signal Strength Indicator (RSSI)
3 -- Last Modified: 26 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;
31 use IEEE.STD_LOGIC_ARITH.ALL;
32 use IEEE.STD_LOGIC_UNSIGNED.ALL;

```

```

33
34 entity rssi is
35     port(
36         clock          : in    std_logic;
37         reset          : in    std_logic;
38         enable         : in    std_logic;
39         adc            : in    std_logic_vector(11 downto 0);
40         rssi          : out   std_logic_vector(15 downto 0);
41         over_count    : out   std_logic_vector(15 downto 0)
42     );
43 end rssi;
44
45 architecture behavioral of rssi is
46     signal over_hi : std_logic;
47     signal over_lo : std_logic;
48     signal over : std_logic;
49     signal over_count_int : std_logic_vector(25 downto 0);
50     signal over_tmp : std_logic_vector(25 downto 0);
51     signal abs_adc : std_logic_vector(11 downto 0);
52     signal rssi_int : std_logic_vector(25 downto 0);
53 begin
54     -- Maximum value for 12 signed bits.
55     over_hi <= '1' when (adc = 16#7FF#) else '0';
56
57     -- Minimum value for 12 signed bits.
58     over_lo <= '1' when (adc = 16#800#) else '0';
59
60     -- Simple OR gate to determine overflows and underflows.
61     over <= over_hi or over_lo;
62
63     -- The original Verilog did this inside the process, but XST translates that
64     -- differently. This is based on the rssi logic.
65     -- d26'65535 = b26'00000000011111111111111111111111.
66     over_tmp <= "0000000000111111111111111111" when over = '1' else
67         "00000000000000000000000000000000";
68
69     -- Track overflows and underflows from the ADCs.
70     process(clock)
71     begin
72         if(clock 'EVENT and clock = '1') then
73             if(reset = '1' or enable = '0') then
74                 -- Clear on reset.
75                 over_count_int <= (others => '0');
76             else
77                 over_count_int <= over_count_int + over_tmp - over_count_int(25 downto 10);
78             end if;
79         end if;
80     end process;
81
82     over_count <= over_count_int(25 downto 10);
83
84     -- Absolute value of adc (sort of).
85     abs_adc <= (not adc) when adc(11) = '1' else adc;
86
87     -- Track signal strength from ADCs.
88     process(clock)
89     begin

```

```

90     if(clock 'EVENT and clock = '1') then
91         if(reset = '1' or enable = '0') then
92             -- Clear on reset.
93             rssi_int <= (others => '0');
94         else
95             rssi_int <= rssi_int + abs_adc - rssi_int(25 downto 10);
96         end if;
97     end if;
98 end process;
99
100 rssi <= rssi_int(25 downto 10);
101 end behavioral;

```

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

```

1  -----
2  -- Receive Buffer (USB <-> DSP)
3  -- Last Modified: 20 July 2010
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2010 COSMIAC
7  -- The copyright and license from the original Verilog implementation follows.
8  --
9  --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27  -----
28
29 -- Interface to Cypress FX2 bus.
30 -- A packet is 512 Bytes. Each FIFO line is 2 bytes.
31 -- FIFO has 4096 lines.
32 library IEEE;
33 use IEEE.STD_LOGIC_1164.ALL;
34 use IEEE.STD_LOGIC_ARITH.ALL;
35 use IEEE.STD_LOGIC_UNSIGNED.ALL;
36
37 library WORK;
38 use WORK.FPGA_REGS_STANDARD.ALL;
39 use WORK.REDUCE_PACK.ALL;
40
41 entity rx_buffer is

```

```

42     port(
43         -- Read/USB side
44         usbclk      : in    std_logic;
45         bus_reset   : in    std_logic;
46         usbdata     : out   std_logic_vector(15 downto 0);
47         RD          : in    std_logic;
48         have_pkt_rdy : out   std_logic;
49         rx_oversrun : out   std_logic;
50         clear_status : in    std_logic;
51
52         -- Write/DSP side
53         rxclk       : in    std_logic;
54         reset       : in    std_logic; -- DSP-side reset, not for registers
55         rxstrobe    : in    std_logic;
56         channels    : in    std_logic_vector(3 downto 0);
57         ch_0        : in    std_logic_vector(15 downto 0);
58         ch_1        : in    std_logic_vector(15 downto 0);
59         ch_2        : in    std_logic_vector(15 downto 0);
60         ch_3        : in    std_logic_vector(15 downto 0);
61         ch_4        : in    std_logic_vector(15 downto 0);
62         ch_5        : in    std_logic_vector(15 downto 0);
63         ch_6        : in    std_logic_vector(15 downto 0);
64         ch_7        : in    std_logic_vector(15 downto 0);
65
66         -- Settings, also using rxclk
67         serial_addr  : in    std_logic_vector(6 downto 0);
68         serial_data  : in    std_logic_vector(31 downto 0);
69         serial_strobe : in    std_logic;
70         reset_regs   : in    std_logic; -- Only resets regs
71         debugbus     : out   std_logic_vector(31 downto 0)
72     );
73 end rx_buffer;
74
75 architecture behavioral of rx_buffer is
76     function round_8(in_val : std_logic_vector(15 downto 0))
77         return std_logic_vector
78     is
79     begin
80         -- if(in_val(15) = '1' and or_reduce(in_val(7 downto 0)) = '1') then
81         --     return in_val(15 downto 8) + 1;
82         -- else
83         --     return in_val(15 downto 8);
84         -- end if;
85         return in_val(15 downto 8) + (in_val(15) and or_reduce(in_val(7 downto 0)));
86     end function;
87
88     component setting_reg is
89         generic(
90             my_addr : integer := 0
91         );
92
93         port(
94             clock : in    std_logic;
95             reset : in    std_logic;
96             strobe : in    std_logic;
97             addr  : in    std_logic_vector;
98             d_in  : in    std_logic_vector(31 downto 0);

```

```

99         d_out   : out   std_logic_vector(31 downto 0);
100         changed : out   std_logic
101     );
102 end component;
103
104 component fifo_4k_18 is
105     port(
106         data      : in     std_logic_vector(17 downto 0);
107         wrreq     : in     std_logic;
108         wrclk     : in     std_logic;
109         wrfull    : out    std_logic;
110         wrempty   : out    std_logic;
111         wrusedw   : out    std_logic_vector(11 downto 0);
112         q         : out    std_logic_vector(17 downto 0);
113         rdreq     : in     std_logic;
114         rdclk     : in     std_logic;
115         rdfull    : out    std_logic;
116         rdempty   : out    std_logic;
117         rdusedw   : out    std_logic_vector(11 downto 0);
118         aclr     : in     std_logic
119     );
120 end component;
121
122 signal fifodata : std_logic_vector(15 downto 0);
123 signal fifodata_8 : std_logic_vector(15 downto 0);
124 signal fifodata_16 : std_logic_vector(15 downto 0);
125
126 signal rxfifolevel : std_logic_vector(11 downto 0);
127 signal rx_full : std_logic;
128
129 signal bypass_hb : std_logic;
130 signal want_q : std_logic;
131 signal bitwidth : std_logic_vector(4 downto 0);
132 signal bitshift : std_logic_vector(3 downto 0);
133
134 signal read_count : std_logic_vector(8 downto 0);
135
136 signal phase : std_logic_vector(3 downto 0);
137 signal ch0_in : std_logic;
138 signal ch0_out : std_logic;
139 signal iq_out : std_logic;
140
141 -- Temps
142 signal have_pkt_rdy_t : std_logic;
143 signal rx_overrun_t : std_logic;
144
145 signal wr_req_t : std_logic;
146 signal rd_req_t : std_logic;
147 signal usbclk_t : std_logic;
148 signal channels_h : std_logic_vector(3 downto 0);
149
150 signal ch_0_reg : std_logic_vector(15 downto 0);
151 signal ch_1_reg : std_logic_vector(15 downto 0);
152 signal ch_2_reg : std_logic_vector(15 downto 0);
153 signal ch_3_reg : std_logic_vector(15 downto 0);
154 signal ch_4_reg : std_logic_vector(15 downto 0);
155 signal ch_5_reg : std_logic_vector(15 downto 0);

```

```

156     signal ch_6_reg : std_logic_vector(15 downto 0);
157     signal ch_7_reg : std_logic_vector(15 downto 0);
158
159     signal top : std_logic_vector(15 downto 0);
160     signal bottom : std_logic_vector(15 downto 0);
161
162     signal clear_status_dsp : std_logic;
163     signal rx_overnun_dsp : std_logic;
164
165     -- Temporary values for getting 8-bit fifodata.
166     -- signal top8 : std_logic_vector(7 downto 0);
167     -- signal bottom8 : std_logic_vector(7 downto 0);
168     begin
169         have_pkt_rdy <= have_pkt_rdy_t;
170         rx_overnun <= rx_overnun_t;
171
172         sr_rxformat : setting_reg
173             generic map(my_addr => FR_RX_FORMAT)
174             port map(clock => rxclk, reset => reset_regs, strobe => serial_strobe, addr =>
175                 serial_addr,
176                 d_in => serial_data, d_out(31 downto 11) => OPEN, d_out(10) => bypass_hb, d_out(9) =>
177                 want_q,
178                 d_out(8 downto 4) => bitwidth, d_out(3 downto 0) => bitshift, changed => OPEN);
179
180         -- USB read side of FIFO
181         process(usbclk)
182             begin
183                 if(usbclk 'EVENT and usbclk = '0') then
184                     if(rx_fifolevel >= "000100000000") then -- 256
185                         have_pkt_rdy_t <= '1';
186                     else
187                         have_pkt_rdy_t <= '0';
188                     end if;
189                 end if;
190             end process;
191
192         -- 257 bug fix
193         process(usbclk)
194             begin
195                 if(usbclk 'EVENT and usbclk = '0') then
196                     if(bus_reset = '1') then
197                         read_count <= (others => '0');
198                     elsif(RD = '1') then
199                         read_count <= read_count + 1;
200                     else
201                         read_count <= (others => '0');
202                     end if;
203                 end if;
204             end process;
205
206         -- FIFO
207         ch0_in <= '1' when (phase = 1) else '0';
208         wr_req_t <= '1' when ((rx_full = '0') and (phase /= 0)) else '0';
209         rd_req_t <= '1' when ((RD = '1') and (read_count(8) = '0')) else '0';
210         usbclk_t <= (not usbclk);
211         rxfifo : fifo_4k_18
212             port map(data(17) => ch0_in, data(16) => phase(0), data(15 downto 0) => fifodata,

```

```

211         wrreq => wr_req_t, wrclk => rxclk, wrfull => rx_full, wrempty => OPEN, wrusedw =>
                OPEN,
212         q(17) => ch0_out, q(16) => iq_out, q(15 downto 0) => usbddata, rdreq => rd_req_t,
213         rdclk => usbcclk_t, rdfull => OPEN, rdempty => OPEN, rdusedw => rxfifolevel, aclr =>
                reset);

214
215     -- DSP write side of FIFO
216     process(rxclk)
217     begin
218         if(rxclk'EVENT and rxclk = '1') then
219             if(rxstrobe = '1') then
220                 ch_0_reg <= ch_0;
221                 ch_1_reg <= ch_1;
222                 ch_2_reg <= ch_2;
223                 ch_3_reg <= ch_3;
224                 ch_4_reg <= ch_4;
225                 ch_5_reg <= ch_5;
226                 ch_6_reg <= ch_6;
227                 ch_7_reg <= ch_7;
228             end if;
229         end if;
230     end process;
231
232     channels_h <= '0' & channels(3 downto 1); -- channels >> 1;
233     process(rxclk)
234     begin
235         if(rxclk'EVENT and rxclk = '1') then
236             if(reset = '1') then
237                 phase <= (others => '0');
238             elsif(phase = 0) then
239                 if(rxstrobe = '1') then
240                     phase <= "0001";
241                 end if;
242             elsif(rx_full = '0') then
243                 if(((bitwidth = 8) and (phase = channels_h)) or (phase = channels)) then
244                     phase <= "0000";
245                 else
246                     phase <= phase + 1;
247                 end if;
248             end if;
249         end if;
250     end process;
251
252     fifodata <= fifodata_8 when (bitwidth = 8) else fifodata_16;
253
254     -- top8 <= top(15 downto 8) + 1 when ((top(15) = '1') and (top(7 downto 0) /= "00000000")) else
255     --     top(15 downto 8);
256
257     -- bottom8 <= bottom(15 downto 8) + 1 when ((bottom(15) = '1') and (bottom(7 downto 0) /=
258     --     "00000000")) else
259     --     bottom(15 downto 8);
260
261     fifodata_8 <= round_8(top) & round_8(bottom);
262     --fifodata_8 <= top8 & bottom8;
263
264     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)
265     begin

```

```

265     case conv_integer(phase) is
266         when 1 =>
267             bottom <= ch_0_reg;
268             top <= ch_1_reg;
269         when 2 =>
270             bottom <= ch_2_reg;
271             top <= ch_3_reg;
272         when 3 =>
273             bottom <= ch_4_reg;
274             top <= ch_5_reg;
275         when 4 =>
276             top <= ch_6_reg;
277             bottom <= ch_7_reg;
278         when others =>
279             top <= (others => '1');
280             bottom <= (others => '1');
281     end case;
282 end process;
283
284 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)
285 begin
286     case conv_integer(phase) is
287         when 1 =>
288             fifodata_16 <= ch_0_reg;
289         when 2 =>
290             fifodata_16 <= ch_1_reg;
291         when 3 =>
292             fifodata_16 <= ch_2_reg;
293         when 4 =>
294             fifodata_16 <= ch_3_reg;
295         when 5 =>
296             fifodata_16 <= ch_4_reg;
297         when 6 =>
298             fifodata_16 <= ch_5_reg;
299         when 7 =>
300             fifodata_16 <= ch_6_reg;
301         when 8 =>
302             fifodata_16 <= ch_7_reg;
303         when others =>
304             fifodata_16 <= (others => '1');
305     end case;
306 end process;
307
308 -- Detect overrun
309 process(rxclk)
310 begin
311     if(rxclk'EVENT and rxclk = '1') then
312         clear_status_dsp <= clear_status;
313     end if;
314 end process;
315
316 process(usbclock)
317 begin
318     if(usbclock'EVENT and usbclock = '0') then
319         rx_overrun_t <= rx_overrun_dsp;
320     end if;
321 end process;

```



```

322
323   process(rxclk)
324   begin
325       if(rxclk 'EVENT and rxclk = '1') then
326           if(reset = '1') then
327               rx_ouerrun_dsp <= '0';
328           elsif((rxstrobe = '1') and (phase /= 0)) then
329               rx_ouerrun_dsp <= '1';
330           elsif(clear_status_dsp = '1') then
331               rx_ouerrun_dsp <= '0';
332           end if;
333       end if;
334   end process;
335
336   -- Debug bus
337   -- 15:0   rxclk   domain => TXA(15:0)
338   -- 31:16  usbclk  domain => RXA(15:0)
339   debugbus(0)          <= reset;
340   debugbus(1)          <= reset_regs;
341   debugbus(2)          <= rxstrobe;
342   debugbus(6 downto 3) <= channels;
343   debugbus(7)          <= rx_full;
344   debugbus(11 downto 8) <= phase;
345   debugbus(12)         <= ch0_in;
346   debugbus(13)         <= clear_status_dsp;
347   debugbus(14)         <= rx_ouerrun_dsp;
348   debugbus(15)         <= rxclk;
349
350   debugbus(16)         <= bus_reset;
351   debugbus(17)         <= RD;
352   debugbus(18)         <= have_pkt_rdy_t;
353   debugbus(19)         <= rx_ouerrun_t;
354   debugbus(20)         <= read_count(0);
355   debugbus(21)         <= read_count(8);
356   debugbus(22)         <= ch0_out;
357   debugbus(23)         <= iq_out;
358   debugbus(24)         <= clear_status;
359   debugbus(30 downto 25) <= "000000";
360   debugbus(31)         <= usbclk;
361 end behavioral;

```

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

```

1 -----
2 -- Receive Chain
3 -- Last Modified: 20 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --

```

```

14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;
31 use IEEE.STD_LOGIC_ARITH.ALL;
32 use IEEE.STD_LOGIC_UNSIGNED.ALL;
33
34 library WORK;
35 use WORK.COMMON_CONFIG.ALL;
36
37 entity rx_chain is
38     generic(
39         FREQADDR : integer := 0;
40         PHASEADDR : integer := 0
41     );
42
43     port(
44         clock          : in    std_logic;
45         reset          : in    std_logic;
46         enable         : in    std_logic;
47         decim_rate     : in    std_logic_vector(7 downto 0);
48         sample_strobe  : in    std_logic;
49         decimator_strobe : in    std_logic;
50         hb_strobe      : out   std_logic;
51         serial_addr    : in    std_logic_vector(6 downto 0);
52         serial_data    : in    std_logic_vector(31 downto 0);
53         serial_strobe  : in    std_logic;
54         i_in           : in    std_logic_vector(15 downto 0);
55         q_in           : in    std_logic_vector(15 downto 0);
56         i_out          : out   std_logic_vector(15 downto 0);
57         q_out          : out   std_logic_vector(15 downto 0);
58         debugdata     : out   std_logic_vector(15 downto 0);
59         debugctrl     : out   std_logic_vector(15 downto 0)
60     );
61 end rx_chain;
62
63 architecture behavioral of rx_chain is
64     component phase_acc is
65         generic(
66             FREQADDR : integer := 0;
67             PHASEADDR : integer := 0;
68             resolution : integer range 0 to 32 := 32
69         );
70

```

```

71     port(
72         clk           : in    std_logic;
73         reset        : in    std_logic;
74         enable       : in    std_logic;
75         strobe       : in    std_logic;
76         serial_addr  : in    std_logic_vector(6 downto 0);
77         serial_data  : in    std_logic_vector(31 downto 0);
78         serial_strobe : in    std_logic;
79         phase        : out   std_logic_vector(resolution-1 downto 0)
80     );
81 end component;
82
83 component cordic is
84     generic(
85         bitwidth : integer := 16;
86         zwidth  : integer := 16
87     );
88
89     port(
90         clock   : in    std_logic;
91         reset   : in    std_logic;
92         enable  : in    std_logic;
93         xi      : in    std_logic_vector(bitwidth-1 downto 0);
94         yi      : in    std_logic_vector(bitwidth-1 downto 0);
95         xo      : out   std_logic_vector(bitwidth-1 downto 0);
96         yo      : out   std_logic_vector(bitwidth-1 downto 0);
97         zi      : in    std_logic_vector(zwidth-1 downto 0);
98         zo      : out   std_logic_vector(zwidth-1 downto 0)
99     );
100 end component;
101
102 component cic_decim is
103     generic(
104         bw : integer := 16;
105         N  : integer := 4;
106         log2_of_max_rate : integer := 7;
107         maxbitgain : integer := 28 --  $N * \log_2 \text{of } \text{max\_rate}$ 
108     );
109
110     port(
111         clock       : in    std_logic;
112         reset       : in    std_logic;
113         enable      : in    std_logic;
114         rate        : in    std_logic_vector(7 downto 0);
115         strobe_in   : in    std_logic;
116         strobe_out  : in    std_logic;
117         signal_in   : in    std_logic_vector(bw-1 downto 0);
118         signal_out  : out   std_logic_vector(bw-1 downto 0)
119     );
120 end component;
121
122 component halfband_decim is
123     port(
124         clock       : in    std_logic;
125         reset       : in    std_logic;
126         enable      : in    std_logic;
127         strobe_in   : in    std_logic;

```

```

128         strobe_out  : out   std_logic;
129         data_in     : in    std_logic_vector(15 downto 0);
130         data_out    : out   std_logic_vector(15 downto 0);
131         debugctrl   : out   std_logic_vector(15 downto 0)
132     );
133 end component;
134
135 signal phase : std_logic_vector(31 downto 0);
136
137 signal bb_i : std_logic_vector(15 downto 0);
138 signal bb_q : std_logic_vector(15 downto 0);
139
140 signal hb_in_i : std_logic_vector(15 downto 0);
141 signal hb_in_q : std_logic_vector(15 downto 0);
142
143 -- Enable write on inputs, read on outputs.
144 signal sample_strobe_t : std_logic;
145 signal decimator_strobe_t : std_logic;
146 begin
147     debugdata <= hb_in_i;
148
149     rx_nco_t : if(RX_NCO_EN) generate
150         sample_strobe_t <= sample_strobe;
151
152         rx_phase_acc : phase_acc
153             generic map(FREQADDR => FREQADDR, PHASEADDR => PHASEADDR, resolution => 32)
154             port map(clk => clock, reset => reset, enable => enable, serial_addr => serial_addr,
155                 serial_data => serial_data, serial_strobe => serial_strobe, strobe =>
156                     sample_strobe_t,
157                     phase => phase);
158
159         rx_cordic : cordic
160             port map(clock => clock, reset => reset, enable => enable, xi => i_in, yi => q_in,
161                 zi => phase(31 downto 16), xo => bb_i, yo => bb_q, zo => OPEN);
162     end generate rx_nco_t;
163
164     rx_nco_f : if(not RX_NCO_EN) generate
165         bb_i <= i_in;
166         bb_q <= q_in;
167         sample_strobe_t <= '1';
168     end generate rx_nco_f;
169
170     rx_cic_t : if(RX_CIC_EN) generate
171         decimator_strobe_t <= decimator_strobe;
172
173         cic_decim_i_0 : cic_decim
174             port map(clock => clock, reset => reset, enable => enable, rate => decim_rate,
175                 strobe_in => sample_strobe, strobe_out => decimator_strobe_t, signal_in => bb_i,
176                 signal_out => hb_in_i);
177
178         cic_decim_q_0 : cic_decim
179             port map(clock => clock, reset => reset, enable => enable, rate => decim_rate,
180                 strobe_in => sample_strobe, strobe_out => decimator_strobe_t, signal_in => bb_q,
181                 signal_out => hb_in_q);
182     end generate rx_cic_t;
183
184     rx_cic_f : if(not RX_CIC_EN) generate

```

```

184     hb_in_i <= bb_i;
185     hb_in_q <= bb_q;
186     decimator_strobe_t <= sample_strobe;
187 end generate rx_cic_f;
188
189 rx_hb_t : if (RX_CAP_HB) generate
190     hbd_i_0 : halfband_decim
191     port map(clock => clock, reset => reset, enable => enable, strobe_in =>
192         decimator_strobe,
193         strobe_out => hb_strobe, data_in => hb_in_i, data_out => i_out, debugctrl =>
194         debugctrl);
195
196     hbd_q_0 : halfband_decim
197     port map(clock => clock, reset => reset, enable => enable, strobe_in =>
198         decimator_strobe,
199         strobe_out => OPEN, data_in => hb_in_q, data_out => q_out, debugctrl => OPEN);
200 end generate rx_hb_t;
201
202 rx_hb_f : if (not RX_CAP_HB) generate
203     i_out <= hb_in_i;
204     q_out <= hb_in_q;
205     hb_strobe <= decimator_strobe;
206 end generate rx_hb_f;
207 end behavioral;

```

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

```

1 -----
2 -- Rx DC Offset
3 -- Last Modified: 20 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 -- TODO: The reduce_pack license is probably not compatible with the GPL.

```

```

30 library IEEE;
31 use IEEE.STD_LOGIC_1164.ALL;
32 use IEEE.STD_LOGIC_ARITH.ALL;
33 use IEEE.STD_LOGIC_SIGNED.ALL;
34
35 library WORK;
36 use WORK.REDUCE_PACK.ALL;
37
38 entity rx_dcoffset is
39     generic(
40         MYADDR : integer := 0
41     );
42
43     port(
44         clock          : in    std_logic;
45         enable         : in    std_logic;
46         reset          : in    std_logic;
47         adc_in         : in    std_logic_vector(15 downto 0);
48         adc_out        : out   std_logic_vector(15 downto 0);
49         serial_addr    : in    std_logic_vector(6  downto 0);
50         serial_data    : in    std_logic_vector(31 downto 0);
51         serial_strobe  : in    std_logic
52     );
53 end rx_dcoffset;
54
55 architecture behavioral of rx_dcoffset is
56     signal adc_in_t : signed(15 downto 0);
57     signal adc_out_t : signed(15 downto 0);
58     signal serial_addr_t : unsigned(6  downto 0);
59     signal serial_data_t : unsigned(31 downto 0);
60     signal integrator : signed(31  downto 0);
61     signal scaled_integrator : signed(15 downto 0);
62 begin
63     -- Assign temporary signals to outputs, with conversions.
64     adc_in_t <= conv_signed(conv_integer(adc_in), 16);
65     serial_addr_t <= conv_unsigned(conv_integer(serial_addr), 7);
66     serial_data_t <= conv_unsigned(conv_integer(serial_data), 32);
67     adc_out <= conv_std_logic_vector(adc_out_t, 16);
68
69     -- The original Verilog code used the or reduction operator, but this is equivalent to
70     -- checking if the vector equals zero.
71     scaled_integrator <= integrator(31 downto 16) + 1 when ((integrator(31) = '1') and
72         (integrator(15 downto 0) /= 0)) else
73         integrator(31 downto 16);
74     adc_out_t <= adc_in_t - scaled_integrator;
75
76     -- FIXME do we need signed?
77     -- FIXME What do we do when clipping?
78     process(clock)
79     begin
80         if(clock'EVENT and clock = '1') then
81             if(reset = '1') then
82                 -- Clear on reset.
83                 integrator <= (others => '0');
84             elsif(serial_strobe = '1' and (MYADDR = serial_addr_t)) then
85                 -- Read new value from the serial bus.
86                 -- This is ugly, but it forces XST to see a single 32-bit register instead of

```

```

86         -- 32 1-bit registers.
87         integrator <= conv_signed(serial_data_t(15 downto 0), 16) & "0000000000000000";
88         elsif(enable = '1') then
89             -- Otherwise, just keep accumulating.
90             integrator <= integrator + adc_out_t;
91         end if;
92     end if;
93 end process;
94 end behavioral;

```

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

```

1 -----
2 -- Serial I/O Bus
3 -- Last Modified: 21 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;
31 use IEEE.STD_LOGIC_ARITH.ALL;
32 use IEEE.STD_LOGIC_UNSIGNED.ALL;
33
34 entity serial_io is
35     port(
36         master_clk      : in    std_logic;
37         serial_clock    : in    std_logic;
38         serial_data_in  : in    std_logic;
39         enable          : in    std_logic;
40         reset           : in    std_logic;
41         serial_data_out : out   std_logic;
42         serial_addr     : out   std_logic_vector(6 downto 0);
43         serial_data     : out   std_logic_vector(31 downto 0);
44         serial_strobe   : out   std_logic;

```

```

45     readback_0      : in    std_logic_vector(31 downto 0);
46     readback_1      : in    std_logic_vector(31 downto 0);
47     readback_2      : in    std_logic_vector(31 downto 0);
48     readback_3      : in    std_logic_vector(31 downto 0);
49     readback_4      : in    std_logic_vector(31 downto 0);
50     readback_5      : in    std_logic_vector(31 downto 0);
51     readback_6      : in    std_logic_vector(31 downto 0);
52     readback_7      : in    std_logic_vector(31 downto 0)
53 );
54 end serial_io;
55
56 architecture behavioral of serial_io is
57     signal is_read : std_logic;
58     signal write_done : std_logic;
59     signal ser_ctr : std_logic_vector(7 downto 0);
60     signal serial_data_tmp : std_logic_vector(31 downto 0);
61     signal serial_addr_tmp : std_logic_vector(6 downto 0);
62     signal enable_d1 : std_logic;
63     signal enable_d2 : std_logic;
64 begin
65     serial_data_out <= serial_data_tmp(31) when is_read = '1' else 'Z';
66     serial_data <= serial_data_tmp;
67     serial_addr <= serial_addr_tmp;
68
69     process(serial_clock , reset , enable)
70     begin
71         if(reset = '1') then
72             ser_ctr <= (others => '0');
73         elsif(enable = '0') then
74             ser_ctr <= (others => '0');
75         elsif(serial_clock 'EVENT and serial_clock = '1') then
76             -- A serial transaction is up to 40 bits. The first bit is 1 for read and
77             -- 0 for write. The next seven bits are the actual register address.
78             -- The final 32 bits are the value to write in a write operation.
79             if(ser_ctr = 39) then
80                 ser_ctr <= (others => '0');
81             else
82                 ser_ctr <= ser_ctr + 1;
83             end if;
84         end if;
85     end process;
86
87     process(serial_clock , reset , enable)
88     begin
89         if(reset = '1') then
90             is_read <= '0';
91         elsif(enable = '0') then
92             is_read <= '0';
93         elsif(serial_clock 'EVENT and serial_clock = '1') then
94             -- Check the first bit we shifted in just before we shift in the
95             -- final address bit.
96             if((ser_ctr = 7) and (serial_addr_tmp(6) = '1')) then
97                 is_read <= '1';
98             end if;
99         end if;
100    end process;
101

```



```

102 process(serial_clock , enable , reset)
103 begin
104     if(reset = '1') then
105         serial_addr_tmp <= (others => '0');
106         serial_data_tmp <= (others => '0');
107         write_done <= '0';
108     elsif(enable = '0') then
109         --serial_addr_tmp <= (others => '0');
110         --serial_data_tmp <= (others => '0');
111         write_done <= '0';
112     elsif(serial_clock 'EVENT and serial_clock = '1') then
113         -- On the final clock , indicate that we performed a write
114         -- operation if we did not perform a read operation.
115         if(is_read = '0' and ser_ctr = 39) then
116             write_done <= '1';
117         else
118             write_done <= '0';
119         end if;
120
121         -- 8'b00001000 = 8'd8
122         if(is_read = '1' and ser_ctr = "00001000") then
123             -- After 8 shifts , we know that it's a read op and we have
124             -- the seven address bits. So, read the proper register.
125             case serial_addr_tmp is
126                 when "0000001" =>
127                     serial_data_tmp <= readback_0;
128                 when "0000010" =>
129                     serial_data_tmp <= readback_1;
130                 when "0000011" =>
131                     serial_data_tmp <= readback_2;
132                 when "0000100" =>
133                     serial_data_tmp <= readback_3;
134                 when "0000101" =>
135                     serial_data_tmp <= readback_4;
136                 when "0000110" =>
137                     serial_data_tmp <= readback_5;
138                 when "0000111" =>
139                     serial_data_tmp <= readback_6;
140                 when "0001000" =>
141                     serial_data_tmp <= readback_7;
142                 when others =>
143                     serial_data_tmp <= (others => '0');
144             end case;
145         elsif(ser_ctr >= "00001000") then
146             -- On the 9th+ shift , write to the serial data because this is
147             -- a write op.
148             serial_data_tmp <= serial_data_tmp(30 downto 0) & serial_data_in;
149         elsif(ser_ctr < "00001000") then
150             -- For the first 8 shifts , collect the read/write bit and then the
151             -- register address. We temporarily store the read/write bit into
152             -- the address vector.
153             serial_addr_tmp <= serial_addr_tmp(5 downto 0) & serial_data_in;
154         end if;
155     end if;
156 end process;
157
158 process(master_clk)

```

```

159     begin
160         if(master_clk 'EVENT and master_clk = '1') then
161             enable_d1 <= enable;
162             enable_d2 <= enable_d1;
163         end if;
164     end process;
165
166     -- Assert serial strobe for one clock after the enable is asserted and
167     -- then deasserted.
168     serial_strobe <= (enable_d2 and (not enable_d1));
169 end behavioral;

```

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

```

1 -----
2 -- Setting Register
3 -- Last Modified: 20 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;
31 use IEEE.STD_LOGIC_ARITH.ALL;
32 use IEEE.STD_LOGIC_UNSIGNED.ALL;
33
34 entity setting_reg is
35     generic(
36         my_addr : integer := 0
37     );
38
39     port(
40         clock      : in    std_logic;
41         reset      : in    std_logic;
42         strobe     : in    std_logic;

```

```

43     addr      : in    std_logic_vector(6 downto 0);
44     d_in      : in    std_logic_vector(31 downto 0);
45     d_out     : out   std_logic_vector(31 downto 0);
46     changed  : out   std_logic
47   );
48   end setting_reg;
49
50   architecture behavioral of setting_reg is
51   begin
52     -- Each register in the system has a unique address, but all are connected
53     -- to the same data bus and control signals.
54     process(clock)
55     begin
56       if(clock 'EVENT and clock = '1') then
57         if(reset = '1') then
58           -- Clear on reset!
59           d_out <= (others => '0');
60           changed <= '0';
61         else
62           -- If the serial strobe is high (see serial_io) and our address is
63           -- on the bus, write to this register.
64           if(strobe = '1' and (my_addr = addr)) then
65             d_out <= d_in;
66             changed <= '1';
67           else
68             changed <= '0';
69           end if;
70         end if;
71       end if;
72     end process;
73   end behavioral;

```

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

```

1  -----
2  -- Sign Extension
3  -- Last Modified: 20 July 2010
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2010 COSMIAC
7  -- The copyright and license from the original Verilog implementation follows.
8  --
9  --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.

```

```

23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 -- We probably don't need this module since VHDL has the sxt() function. I
30 -- left it in place to keep the project as close to the Verilog as possible.
31 library IEEE;
32 use IEEE.STD_LOGIC_1164.ALL;
33 use IEEE.STD_LOGIC_ARITH.ALL;
34 use IEEE.STD_LOGIC_UNSIGNED.ALL;
35
36 entity sign_extend is
37     generic(
38         bits_in : integer := 0;
39         bits_out  : integer := 0
40     );
41
42     port(
43         d_in   : in   std_logic_vector(bits_in-1 downto 0);
44         d_out  : out  std_logic_vector(bits_out-1 downto 0)
45     );
46 end sign_extend;
47
48 architecture behavioral of sign_extend is
49 begin
50     -- Copy d_in to the lower bits_in bits.
51     d_out(bits_in-1 downto 0) <= d_in;
52
53     -- Fill the upper bits_out - bits_in bits with the MSB from d_in.
54     d_out(bits_out-1 downto bits_in) <= (others => d_in(bits_in-1));
55 end behavioral;

```

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

```

1 -----
2 -- Strobe Generator
3 -- Last Modified: 21 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of

```

```

21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 -- TODO: The license for reduce_pack is probably not valid with the GPL.
30 library IEEE;
31 use IEEE.STD_LOGIC_1164.ALL;
32 use IEEE.STD_LOGIC_ARITH.ALL;
33 use IEEE.STD_LOGIC_UNSIGNED.ALL;
34
35 library WORK;
36 use WORK.REDUCE_PACK.ALL;
37
38 entity strobe_gen is
39     port(
40         clock          : in    std_logic;
41         reset          : in    std_logic;
42         enable         : in    std_logic;
43         rate           : in    std_logic_vector(7 downto 0); -- 1 less than desired divide
44                             ratio
45         strobe_in      : in    std_logic;
46         strobe         : out   std_logic
47     );
48 end strobe_gen;
49
50 architecture behavioral of strobe_gen is
51     signal counter : std_logic_vector(7 downto 0);
52     begin
53         -- Note that nor_reduce = 1 when counter = "00000000".
54         strobe <= '1' when ((nor_reduce(counter) = '1') and (enable = '1') and (strobe_in = '1'))
55             else '0';
56
57         -- This is basically just a clock divider that uses a down counter. For every
58         -- 'rate' clocks that strobe_in is asserted, we assert strobe.
59         process(clock)
60         begin
61             if(clock 'EVENT and clock = '1') then
62                 if(reset = '1' or enable = '0') then
63                     counter <= (others => '0');
64                 elsif(strobe_in = '1') then
65                     if(counter = 0) then
66                         counter <= rate;
67                     else
68                         counter <= counter - 1;
69                     end if;
70                 end if;
71             end if;
72         end process;
73     end behavioral;

```

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

```

1 -----
2 -- Transmit Buffer (DSP <-> USB)
3 -- Last Modified: 20 July 2010
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 -- Interface to Cypress FX2 bus.
30 -- A packet is 512 Bytes. Each FIFO line is 2 bytes.
31 -- FIFO has 4096 lines.
32 library IEEE;
33 use IEEE.STD_LOGIC_1164.ALL;
34 use IEEE.STD_LOGIC_ARITH.ALL;
35 use IEEE.STD_LOGIC_UNSIGNED.ALL;
36
37 entity tx_buffer is
38     port(
39         -- USB side
40         usbclk          : in    std_logic;
41         bus_reset       : in    std_logic; -- 257-Hack to fix FX2 bug
42         usbdata         : in    std_logic_vector(15 downto 0);
43         WR              : in    std_logic;
44         have_space      : out   std_logic;
45         tx_underrun     : out   std_logic;
46         clear_status    : in    std_logic;
47
48         -- DSP side
49         txclk           : in    std_logic;
50         reset           : in    std_logic; -- standard DSP-side reset
51         channels        : in    std_logic_vector(3 downto 0);
52         tx_i_0         : out   std_logic_vector(15 downto 0);
53         tx_q_0         : out   std_logic_vector(15 downto 0);
54         tx_i_1         : out   std_logic_vector(15 downto 0);
55         tx_q_1         : out   std_logic_vector(15 downto 0);
56         txstrobe       : in    std_logic;
57         tx_empty       : out   std_logic;

```

```

58         debugbus          : out   std_logic_vector(31 downto 0)
59     );
60 end tx_buffer;
61
62 architecture behavioral of tx_buffer is
63     component fifo_4k_18 is
64         port(
65             data      : in   std_logic_vector(17 downto 0);
66             wrreq     : in   std_logic;
67             wrclk     : in   std_logic;
68             wrfull    : out  std_logic;
69             wrempty   : out  std_logic;
70             wrusedw   : out  std_logic_vector(11 downto 0);
71             q         : out  std_logic_vector(17 downto 0);
72             rdreq     : in   std_logic;
73             rdclk     : in   std_logic;
74             rdfull    : out  std_logic;
75             rdempty   : out  std_logic;
76             rdusedw   : out  std_logic_vector(11 downto 0);
77             aclr      : in   std_logic
78         );
79     end component;
80
81     signal txfifolevel : std_logic_vector(11 downto 0);
82     signal fifodata    : std_logic_vector(15 downto 0);
83     signal rdreq       : std_logic;
84     signal phase       : std_logic_vector(3 downto 0);
85     signal sop_f       : std_logic;
86     signal iq_f        : std_logic;
87     signal sop         : std_logic;
88
89     -- USB side of the FIFO
90     signal usbdata_reg : std_logic_vector(15 downto 0);
91     signal wr_reg      : std_logic;
92     signal write_count : std_logic_vector(8 downto 0);
93
94     -- Temporary for FIFO
95     signal wr_req_t    : std_logic;
96     signal tx_empty_t  : std_logic;
97     signal have_space_t : std_logic;
98     signal tx_underrun_t : std_logic;
99
100    signal clear_status_dsp : std_logic;
101    signal tx_underrun_dsp  : std_logic;
102 begin
103     tx_empty <= tx_empty_t;
104     have_space <= have_space_t;
105     tx_underrun <= tx_underrun_t;
106
107     process(usbclk)
108     begin
109         if(usbclk 'EVENT and usbclk = '1') then
110             -- Be conservative! (4092 - 256)
111             if(txfifolevel < "111011111100") then
112                 have_space_t <= '1';
113             else
114                 have_space_t <= '0';

```

```

115         end if;
116     end if;
117 end process;
118
119 process(usbclk)
120 begin
121     if(usbclk 'EVENT and usbclk = '1') then
122         wr_reg <= WR;
123         usbdata_reg <= usbdata;
124     end if;
125 end process;
126
127 process(usbclk)
128 begin
129     if(usbclk 'EVENT and usbclk = '1') then
130         if(bus_reset = '1') then
131             write_count <= (others => '0');
132         elsif(wr_reg = '1') then
133             write_count <= write_count + 1;
134         else
135             write_count <= (others => '0');
136         end if;
137     end if;
138 end process;
139
140 process(usbclk)
141 begin
142     if(usbclk 'EVENT and usbclk = '1') then
143         sop <= WR and (not wr_reg); -- edge detect
144     end if;
145 end process;
146
147 -- FIFO
148 wr_req_t <= (wr_reg and (not write_count(8)));
149 txfifo : fifo_4k_18
150     port map(data(17) => sop, data(16) => write_count(0), data(15 downto 0) => usbdata_reg,
151             wrreq => wr_req_t, wrclk => usbclk, wrfull => OPEN, wrempty => OPEN,
152             wrusedw => txfifolevel, q(17) => sop_f, q(16) => iq_f, q(15 downto 0) => fifodata,
153             rdreq => rdreq, rdclk => txclk, rdfull => OPEN, rdempty => tx_empty_t, rdusedw =>
154                 OPEN,
155             aclr => reset);
156
157 -- DAC side of the FIFO
158 process(txclk)
159 begin
160     if(txclk 'EVENT and txclk = '1') then
161         if(reset = '1') then
162             tx_i_0 <= (others => '0');
163             tx_q_0 <= (others => '0');
164             tx_i_1 <= (others => '0');
165             tx_q_1 <= (others => '0');
166             phase <= "0000";
167         elsif(phase = channels) then
168             if(txstrobe = '1') then
169                 phase <= "0000";
170             end if;
171         elsif(tx_empty_t = '0') then

```



```

171         case phase is
172             when "0000" =>
173                 tx_i_0 <= fifodata;
174             when "0001" =>
175                 tx_q_0 <= fifodata;
176             when "0010" =>
177                 tx_i_1 <= fifodata;
178             when "0011" =>
179                 tx_q_1 <= fifodata;
180             when others =>
181                 null;
182         end case;
183
184         phase <= phase + 1;
185     end if;
186 end if;
187 end process;
188
189 rdreq <= '1' when ((phase /= channels) and (tx_empty_t = '0')) else '0';
190
191 -- Detect underruns, cross clock domains
192 process(txclk)
193 begin
194     if(txclk 'EVENT and txclk = '1') then
195         clear_status_dsp <= clear_status;
196     end if;
197 end process;
198
199 process(usbclock)
200 begin
201     if(usbclock 'EVENT and usbclock = '1') then
202         tx_underrun_t <= tx_underrun_dsp;
203     end if;
204 end process;
205
206 process(txclk)
207 begin
208     if(txclk 'EVENT and txclk = '1') then
209         if(reset = '1') then
210             tx_underrun_dsp <= '0';
211         elsif(txstrobe = '1' and (phase /= channels)) then
212             tx_underrun_dsp <= '1';
213         elsif(clear_status_dsp = '1') then
214             tx_underrun_dsp <= '0';
215         end if;
216     end if;
217 end process;
218
219 -- TX debug bus
220 -- 15:0    txclk    domain => TXA(15:0)
221 -- 31:16  usbclock domain => RXA(15:0)
222 debugbus(0)      <= reset;
223 debugbus(1)      <= txstrobe;
224 debugbus(2)      <= rdreq;
225 debugbus(6 downto 3) <= phase;
226 debugbus(7)      <= tx_empty_t;
227 debugbus(8)      <= tx_underrun_dsp;

```

```

228     debugbus(9)           <= iq_f;
229     debugbus(10)          <= sop_f;
230     debugbus(14 downto 11) <= "0000";
231     debugbus(15)          <= txclk;
232
233     debugbus(16)           <= bus_reset;
234     debugbus(17)           <= WR;
235     debugbus(18)           <= wr_reg;
236     debugbus(19)           <= have_space_t;
237     debugbus(20)           <= write_count(8);
238     debugbus(21)           <= write_count(0);
239     debugbus(22)           <= sop;
240     debugbus(23)           <= tx_underrun_t;
241     debugbus(30 downto 24) <= "0000000";
242     debugbus(31)           <= usbclk;
243 end behavioral;

```

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

```

1 -----
2 -- Transmit Chain
3 -- Last Modified: 22 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 COSMIAC
7 -- The copyright and license from the original Verilog implementation follows.
8 --
9 --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 -- The original Verilog source had partial support for cordic, but it was not
30 -- used and relied on outdated modules. According to documentation, the
31 -- translation from baseband to the IF band is now done on the AD9862 codecs.
32 -- The 'freq' input is a legacy from the old cordic DUC.
33 library IEEE;
34 use IEEE.STD_LOGIC_1164.ALL;
35
36 entity tx_chain is
37     port(

```

```

38     clock                : in    std_logic;
39     reset                : in    std_logic;
40     enable               : in    std_logic;
41     interp_rate         : in    std_logic_vector(7 downto 0);
42     sample_strobe       : in    std_logic;
43     interpolator_strobe : in    std_logic;
44     freq                : in    std_logic_vector(31 downto 0);
45     i_in                : in    std_logic_vector(15 downto 0);
46     q_in                : in    std_logic_vector(15 downto 0);
47     i_out               : out   std_logic_vector(15 downto 0);
48     q_out               : out   std_logic_vector(15 downto 0);
49 );
50 end tx_chain;
51
52 architecture behavioral of tx_chain is
53     component cic_interp is
54         generic(
55             bw                : integer := 16;  -- # of bits for input
56             N                 : integer := 4;  -- # of filter stages
57             log2_of_max_rate  : integer := 7;  -- log2 of max sampling rate
58             maxbitgain       : integer := 21 -- (N-1)*log2_of_max_rate
59         );
60
61         port(
62             clock            : in    std_logic;
63             reset           : in    std_logic;
64             enable          : in    std_logic;
65             rate            : in    std_logic_vector(7 downto 0);
66             strobe_in       : in    std_logic;
67             strobe_out      : in    std_logic;
68             signal_in      : in    std_logic_vector(bw-1 downto 0);
69             signal_out      : out   std_logic_vector(bw-1 downto 0)
70         );
71     end component;
72
73     signal bb_i : std_logic_vector(15 downto 0);
74     signal bb_q : std_logic_vector(15 downto 0);
75 begin
76     cic_interp_i : cic_interp
77         port map(clock => clock, reset => reset, enable => enable, rate => interp_rate,
78             strobe_in => interpolator_strobe, strobe_out => sample_strobe, signal_in => i_in,
79             signal_out => bb_i);
80
81     cic_interp_q : cic_interp
82         port map(clock => clock, reset => reset, enable => enable, rate => interp_rate,
83             strobe_in => interpolator_strobe, strobe_out => sample_strobe, signal_in => q_in,
84             signal_out => bb_q);
85
86     i_out <= bb_i;
87     q_out <= bb_q;
88 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)

```

1
2 -- Special MUX for USRP Toplevel Signals

```

```

3  -- Last Modified: 20 July 2010
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2010 COSMIAC
7  --
8  -- This program is free software; you can redistribute it and/or modify
9  -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22
23 -- The original Verilog USRP1 contains no equivalent for this component. In
24 -- Verilog, this logic is trivial to implement with the ternary ?: operator.
25 library IEEE;
26 use IEEE.STD_LOGIC_1164.ALL;
27 use IEEE.STD_LOGIC_ARITH.ALL;
28 use IEEE.STD_LOGIC_UNSIGNED.ALL;
29
30 entity usrp_mux is
31     port(
32         dac_mux      : in    std_logic_vector(3 downto 0);
33         i_out_0      : in    std_logic_vector(15 downto 0);
34         q_out_0      : in    std_logic_vector(15 downto 0);
35         i_out_1      : in    std_logic_vector(15 downto 0);
36         q_out_1      : in    std_logic_vector(15 downto 0);
37         d_out        : out   std_logic_vector(15 downto 0)
38     );
39 end usrp_mux;
40
41 architecture behavioral of usrp_mux is
42 begin
43     -- Note that dac_mux(2) is intentionally not used. I don't know why.
44     -- The equivalent Verilog statement is:
45     -- d_out = dac_mux[3] ? (dac_mux[1] ? (dac_mux[0] ? q_out_1
46     --                                     : i_out_1)
47     --                : (dac_mux[0] ? q_out_0
48     --                  : i_out_0))
49     --                : 16'b0;
50     process(dac_mux, i_out_0, q_out_0, i_out_1, q_out_1)
51     begin
52         if(dac_mux(3) = '1') then
53             if(dac_mux(1) = '1') then
54                 if(dac_mux(0) = '1') then
55                     d_out <= q_out_1;
56                 else
57                     d_out <= i_out_1;
58                 end if;
59             else

```

```

60         if(dac_mux(0) = '1') then
61             d_out <= q_out_0;
62         else
63             d_out <= i_out_0;
64         end if;
65     end if;
66 else
67     d_out <= (others => '0');
68 end if;
69 end process;
70 end behavioral;

```

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

```

1  -----
2  -- USRP1 Standard Toplevel
3  -- Last Modified: 21 July 2010
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2010 COSMIAC
7  -- The copyright and license from the original Verilog implementation follows.
8  --
9  --
10 -- USRP - Universal Software Radio Peripheral
11 --
12 -- Copyright (C) 2003 Matt Ettus
13 --
14 -- This program is free software; you can redistribute it and/or modify
15 -- it under the terms of the GNU General Public License as published by
16 -- the Free Software Foundation; either version 2 of the License, or
17 -- (at your option) any later version.
18 --
19 -- This program is distributed in the hope that it will be useful,
20 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
21 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 -- GNU General Public License for more details.
23 --
24 -- You should have received a copy of the GNU General Public License
25 -- along with this program; if not, write to the Free Software
26 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
27 -----
28
29 library IEEE;
30 use IEEE.STD_LOGIC_1164.ALL;
31 use IEEE.STD_LOGIC_ARITH.ALL;
32 use IEEE.STD_LOGIC_UNSIGNED.ALL;
33
34 library WORK;
35 use WORK.COMMON_CONFIG.ALL;
36 use WORK.FPGA_REGS_COMMON.ALL;
37 use WORK.FPGA_REGS_STANDARD.ALL;
38
39 entity usrp_std is
40     port(
41         MYSTERY_SIGNAL : out  std_logic;
42         master_clk      : in   std_logic;

```

```

43
44     -- SPI interface
45     SCLK           : in    std_logic;
46     SDI            : in    std_logic;
47     SDO            : inout  std_logic;
48     SEN_FPGA       : in    std_logic;
49
50     -- USB interface
51     FX2_1          : in    std_logic;
52     FX2_2          : out   std_logic;
53     FX2_3          : out   std_logic;
54
55     -- Codec interface
56     rx_a_a         : in    std_logic_vector(11 downto 0);
57     rx_b_a         : in    std_logic_vector(11 downto 0);
58     rx_a_b         : in    std_logic_vector(11 downto 0);
59     rx_b_b         : in    std_logic_vector(11 downto 0);
60
61     tx_a           : out   std_logic_vector(13 downto 0);
62     tx_b           : out   std_logic_vector(13 downto 0);
63
64     TXSYNC_A       : out   std_logic;
65     TXSYNC_B       : out   std_logic;
66
67     -- USB interface
68     usbclk         : in    std_logic;
69     usbctl         : in    std_logic_vector(2 downto 0);
70     usbrdy         : out   std_logic_vector(1 downto 0);
71     usbdata        : inout  std_logic_vector(15 downto 0);
72
73     -- GPIOs to the daughtercard slots
74     io_tx_a        : inout  std_logic_vector(15 downto 0);
75     io_tx_b        : inout  std_logic_vector(15 downto 0);
76     io_rx_a        : inout  std_logic_vector(15 downto 0);
77     io_rx_b        : inout  std_logic_vector(15 downto 0)
78 );
79 end usrp_std;
80
81 architecture behavioral of usrp_std is
82     component bustri is
83     port(
84         data        : in    std_logic_vector(15 downto 0);
85         enabledt    : in    std_logic;
86         tridata     : out   std_logic_vector(15 downto 0)
87     );
88 end component;
89
90     component tx_buffer is
91     port(
92         -- USB side
93         usbclk      : in    std_logic;
94         bus_reset   : in    std_logic; -- 257-Hack to fix FX2 bug
95         usbdata     : in    std_logic_vector(15 downto 0);
96         WR          : in    std_logic;
97         have_space  : out   std_logic;
98         tx_underrun : out   std_logic;
99         clear_status : in    std_logic;

```

```

100
101     -- DSP side
102     txclk          : in    std_logic;
103     reset          : in    std_logic; -- standard DSP-side reset
104     channels       : in    std_logic_vector(3 downto 0);
105     tx_i_0         : out   std_logic_vector(15 downto 0);
106     tx_q_0         : out   std_logic_vector(15 downto 0);
107     tx_i_1         : out   std_logic_vector(15 downto 0);
108     tx_q_1         : out   std_logic_vector(15 downto 0);
109     txstrobe       : in    std_logic;
110     tx_empty       : out   std_logic;
111     debugbus       : out   std_logic_vector(31 downto 0)
112 );
113 end component;
114
115 component tx_chain is
116     port(
117         clock          : in    std_logic;
118         reset          : in    std_logic;
119         enable         : in    std_logic;
120         interp_rate    : in    std_logic_vector(7 downto 0);
121         sample_strobe  : in    std_logic;
122         interpolator_strobe : in    std_logic;
123         freq           : in    std_logic_vector(31 downto 0);
124         i_in           : in    std_logic_vector(15 downto 0);
125         q_in           : in    std_logic_vector(15 downto 0);
126         i_out          : out   std_logic_vector(15 downto 0);
127         q_out          : out   std_logic_vector(15 downto 0)
128     );
129 end component;
130
131 component setting_reg is
132     generic(
133         my_addr : integer := 0
134     );
135
136     port(
137         clock : in    std_logic;
138         reset : in    std_logic;
139         strobe : in    std_logic;
140         addr : in    std_logic_vector;
141         d_in : in    std_logic_vector(31 downto 0);
142         d_out : out   std_logic_vector(31 downto 0);
143         changed : out   std_logic
144     );
145 end component;
146
147 component usrp_mux is
148     port(
149         dac_mux : in    std_logic_vector(3 downto 0);
150         i_out_0 : in    std_logic_vector(15 downto 0);
151         q_out_0 : in    std_logic_vector(15 downto 0);
152         i_out_1 : in    std_logic_vector(15 downto 0);
153         q_out_1 : in    std_logic_vector(15 downto 0);
154         d_out : out   std_logic_vector(15 downto 0)
155     );
156 end component;

```

```

157
158     component adc_interface is
159         port(
160             clock          : in    std_logic;
161             reset           : in    std_logic;
162             enable         : in    std_logic;
163             serial_addr     : in    std_logic_vector(6 downto 0);
164             serial_data     : in    std_logic_vector(31 downto 0);
165             serial_strobe  : in    std_logic;
166             rx_a_a         : in    std_logic_vector(11 downto 0);
167             rx_b_a         : in    std_logic_vector(11 downto 0);
168             rx_a_b         : in    std_logic_vector(11 downto 0);
169             rx_b_b         : in    std_logic_vector(11 downto 0);
170             rssi_0         : out   std_logic_vector(31 downto 0);
171             rssi_1         : out   std_logic_vector(31 downto 0);
172             rssi_2         : out   std_logic_vector(31 downto 0);
173             rssi_3         : out   std_logic_vector(31 downto 0);
174             ddc0_in_i     : out   std_logic_vector(15 downto 0);
175             ddc0_in_q     : out   std_logic_vector(15 downto 0);
176             ddc1_in_i     : out   std_logic_vector(15 downto 0);
177             ddc1_in_q     : out   std_logic_vector(15 downto 0);
178             ddc2_in_i     : out   std_logic_vector(15 downto 0);
179             ddc2_in_q     : out   std_logic_vector(15 downto 0);
180             ddc3_in_i     : out   std_logic_vector(15 downto 0);
181             ddc3_in_q     : out   std_logic_vector(15 downto 0);
182             rx_numchan    : out   std_logic_vector(3 downto 0)
183         );
184     end component;
185
186     component rx_buffer is
187         port(
188             -- Read/USB side
189             usbclk        : in    std_logic;
190             bus_reset     : in    std_logic;
191             usbdata       : out   std_logic_vector(15 downto 0);
192             RD            : in    std_logic;
193             have_pkt_rdy  : out   std_logic;
194             rx_overrun    : out   std_logic;
195             clear_status  : in    std_logic;
196
197             -- Write/DSP side
198             rxclk         : in    std_logic;
199             reset         : in    std_logic; -- DSP-side reset, not for registers
200             rxstrobe     : in    std_logic;
201             channels      : in    std_logic_vector(3 downto 0);
202             ch_0          : in    std_logic_vector(15 downto 0);
203             ch_1          : in    std_logic_vector(15 downto 0);
204             ch_2          : in    std_logic_vector(15 downto 0);
205             ch_3          : in    std_logic_vector(15 downto 0);
206             ch_4          : in    std_logic_vector(15 downto 0);
207             ch_5          : in    std_logic_vector(15 downto 0);
208             ch_6          : in    std_logic_vector(15 downto 0);
209             ch_7          : in    std_logic_vector(15 downto 0);
210
211             -- Settings, also using rxclk
212             serial_addr   : in    std_logic_vector(6 downto 0);
213             serial_data   : in    std_logic_vector(31 downto 0);

```



```

214         serial_strobe   : in    std_logic;
215         reset_regs     : in    std_logic; -- Only resets regs
216         debugbus       : out   std_logic_vector(31 downto 0)
217     );
218 end component;
219
220 component rx_chain is
221     generic(
222         FREQADDR : integer := 0;
223         PHASEADDR : integer := 0
224     );
225
226     port(
227         clock           : in    std_logic;
228         reset           : in    std_logic;
229         enable          : in    std_logic;
230         decim_rate      : in    std_logic_vector(7 downto 0);
231         sample_strobe  : in    std_logic;
232         decimator_strobe : in    std_logic;
233         hb_strobe      : out   std_logic;
234         serial_addr    : in    std_logic_vector(6 downto 0);
235         serial_data     : in    std_logic_vector(31 downto 0);
236         serial_strobe  : in    std_logic;
237         i_in           : in    std_logic_vector(15 downto 0);
238         q_in           : in    std_logic_vector(15 downto 0);
239         i_out          : out   std_logic_vector(15 downto 0);
240         q_out          : out   std_logic_vector(15 downto 0);
241         debugdata      : out   std_logic_vector(15 downto 0);
242         debugctrl      : out   std_logic_vector(15 downto 0)
243     );
244 end component;
245
246 component serial_io is
247     port(
248         master_clk      : in    std_logic;
249         serial_clock    : in    std_logic;
250         serial_data_in  : in    std_logic;
251         enable          : in    std_logic;
252         reset           : in    std_logic;
253         serial_data_out : out   std_logic;
254         serial_addr     : out   std_logic_vector(6 downto 0);
255         serial_data     : out   std_logic_vector(31 downto 0);
256         serial_strobe  : out   std_logic;
257         readback_0     : in    std_logic_vector(31 downto 0);
258         readback_1     : in    std_logic_vector(31 downto 0);
259         readback_2     : in    std_logic_vector(31 downto 0);
260         readback_3     : in    std_logic_vector(31 downto 0);
261         readback_4     : in    std_logic_vector(31 downto 0);
262         readback_5     : in    std_logic_vector(31 downto 0);
263         readback_6     : in    std_logic_vector(31 downto 0);
264         readback_7     : in    std_logic_vector(31 downto 0)
265     );
266 end component;
267
268 component master_control is
269     port(
270         master_clk      : in    std_logic;

```

```

271         usbclk           : in    std_logic;
272         serial_addr      : in    std_logic_vector(6 downto 0);
273         serial_data       : in    std_logic_vector(31 downto 0);
274         serial_strobe    : in    std_logic;
275         tx_bus_reset     : out   std_logic;
276         rx_bus_reset     : out   std_logic;
277         tx_dsp_reset     : out   std_logic;
278         rx_dsp_reset     : out   std_logic;
279         enable_tx        : out   std_logic;
280         enable_rx        : out   std_logic;
281         interp_rate      : out   std_logic_vector(7 downto 0);
282         decim_rate       : out   std_logic_vector(7 downto 0);
283         tx_sample_strobe : out   std_logic;
284         strobe_interp    : out   std_logic;
285         rx_sample_strobe : out   std_logic;
286         strobe_decim     : out   std_logic;
287         tx_empty         : in    std_logic;
288         debug_0          : in    std_logic_vector(15 downto 0);
289         debug_1          : in    std_logic_vector(15 downto 0);
290         debug_2          : in    std_logic_vector(15 downto 0);
291         debug_3          : in    std_logic_vector(15 downto 0);
292         reg_0            : out   std_logic_vector(15 downto 0);
293         reg_1            : out   std_logic_vector(15 downto 0);
294         reg_2            : out   std_logic_vector(15 downto 0);
295         reg_3            : out   std_logic_vector(15 downto 0);
296     );
297 end component;
298
299 component io_pins is
300     port(
301         io_0              : inout std_logic_vector(15 downto 0);
302         io_1              : inout std_logic_vector(15 downto 0);
303         io_2              : inout std_logic_vector(15 downto 0);
304         io_3              : inout std_logic_vector(15 downto 0);
305         reg_0             : in    std_logic_vector(15 downto 0);
306         reg_1             : in    std_logic_vector(15 downto 0);
307         reg_2             : in    std_logic_vector(15 downto 0);
308         reg_3             : in    std_logic_vector(15 downto 0);
309         clock             : in    std_logic;
310         rx_reset          : in    std_logic;
311         tx_reset          : in    std_logic;
312         serial_addr       : in    std_logic_vector(6 downto 0);
313         serial_data       : in    std_logic_vector(31 downto 0);
314         serial_strobe     : in    std_logic
315     );
316 end component;
317
318 signal debugdata : std_logic_vector(15 downto 0);
319 signal debugctrl : std_logic_vector(15 downto 0);
320
321 signal clk64 : std_logic;
322 signal clk128 : std_logic;
323
324 signal WR : std_logic;
325 signal RD : std_logic;
326 signal OE : std_logic;
327

```

```

328     signal have_space : std_logic;
329     signal have_pkt_rdy : std_logic;
330
331     signal tx_underrun : std_logic;
332     signal rx_overrun : std_logic;
333     signal clear_status : std_logic;
334
335     signal usbdata_out : std_logic_vector(15 downto 0);
336
337     signal dac0mux : std_logic_vector(3 downto 0);
338     signal dac1mux : std_logic_vector(3 downto 0);
339     signal dac2mux : std_logic_vector(3 downto 0);
340     signal dac3mux : std_logic_vector(3 downto 0);
341
342     signal tx_realsignals : std_logic;
343     signal rx_numchan : std_logic_vector(3 downto 0);
344     signal tx_numchan : std_logic_vector(2 downto 0);
345
346     signal interp_rate : std_logic_vector(7 downto 0);
347     signal decim_rate : std_logic_vector(7 downto 0);
348
349     signal tx_debugbus : std_logic_vector(31 downto 0);
350     signal rx_debugbus : std_logic_vector(31 downto 0);
351
352     signal enable_tx : std_logic;
353     signal enable_rx : std_logic;
354
355     signal tx_dsp_reset : std_logic;
356     signal rx_dsp_reset : std_logic;
357     signal tx_bus_reset : std_logic;
358     signal rx_bus_reset : std_logic;
359
360     signal settings : std_logic_vector(7 downto 0);
361
362     signal ch0tx : std_logic_vector(15 downto 0);
363     signal ch1tx : std_logic_vector(15 downto 0);
364     signal ch2tx : std_logic_vector(15 downto 0);
365     signal ch3tx : std_logic_vector(15 downto 0);
366
367     signal ch0rx : std_logic_vector(15 downto 0);
368     signal ch1rx : std_logic_vector(15 downto 0);
369     signal ch2rx : std_logic_vector(15 downto 0);
370     signal ch3rx : std_logic_vector(15 downto 0);
371     signal ch4rx : std_logic_vector(15 downto 0);
372     signal ch5rx : std_logic_vector(15 downto 0);
373     signal ch6rx : std_logic_vector(15 downto 0);
374     signal ch7rx : std_logic_vector(15 downto 0);
375
376     -- TX
377     signal i_out_0 : std_logic_vector(15 downto 0);
378     signal i_out_1 : std_logic_vector(15 downto 0);
379     signal q_out_0 : std_logic_vector(15 downto 0);
380     signal q_out_1 : std_logic_vector(15 downto 0);
381
382     signal bb_tx_i0 : std_logic_vector(15 downto 0);
383     signal bb_tx_q0 : std_logic_vector(15 downto 0);
384     signal bb_tx_i1 : std_logic_vector(15 downto 0);

```

```

385     signal bb_tx_q1 : std_logic_vector(15 downto 0);
386
387     signal strobe_interp : std_logic;
388     signal tx_sample_strobe : std_logic;
389     signal tx_empty : std_logic;
390
391     signal serial_strobe : std_logic;
392     signal serial_addr : std_logic_vector(6 downto 0);
393     signal serial_data : std_logic_vector(31 downto 0);
394
395     signal debug_counter : std_logic_vector(15 downto 0);
396     signal loopback_i_0 : std_logic_vector(15 downto 0);
397     signal loopback_q_0 : std_logic_vector(15 downto 0);
398
399     signal tx_a_a : std_logic_vector(15 downto 0);
400     signal tx_b_a : std_logic_vector(15 downto 0);
401     signal tx_a_b : std_logic_vector(15 downto 0);
402     signal tx_b_b : std_logic_vector(15 downto 0);
403
404     signal txsync : std_logic;
405
406     -- RX
407     signal rx_sample_strobe : std_logic;
408     signal strobe_decim : std_logic;
409     signal hb_strobe : std_logic;
410
411     signal bb_rx_i0 : std_logic_vector(15 downto 0);
412     signal bb_rx_q0 : std_logic_vector(15 downto 0);
413     signal bb_rx_i1 : std_logic_vector(15 downto 0);
414     signal bb_rx_q1 : std_logic_vector(15 downto 0);
415     signal bb_rx_i2 : std_logic_vector(15 downto 0);
416     signal bb_rx_q2 : std_logic_vector(15 downto 0);
417     signal bb_rx_i3 : std_logic_vector(15 downto 0);
418     signal bb_rx_q3 : std_logic_vector(15 downto 0);
419
420     signal loopback : std_logic;
421     signal counter : std_logic;
422
423     signal ddc0_in_i : std_logic_vector(15 downto 0);
424     signal ddc0_in_q : std_logic_vector(15 downto 0);
425     signal ddc1_in_i : std_logic_vector(15 downto 0);
426     signal ddc1_in_q : std_logic_vector(15 downto 0);
427     signal ddc2_in_i : std_logic_vector(15 downto 0);
428     signal ddc2_in_q : std_logic_vector(15 downto 0);
429     signal ddc3_in_i : std_logic_vector(15 downto 0);
430     signal ddc3_in_q : std_logic_vector(15 downto 0);
431
432     signal rssi_0 : std_logic_vector(31 downto 0);
433     signal rssi_1 : std_logic_vector(31 downto 0);
434     signal rssi_2 : std_logic_vector(31 downto 0);
435     signal rssi_3 : std_logic_vector(31 downto 0);
436
437     signal capabilities : std_logic_vector(31 downto 0);
438
439     signal reg_0 : std_logic_vector(15 downto 0);
440     signal reg_1 : std_logic_vector(15 downto 0);
441     signal reg_2 : std_logic_vector(15 downto 0);

```

```

442     signal reg_3 : std_logic_vector(15 downto 0);
443 begin
444     MYSTERY_SIGNAL <= '0';
445
446     -- USB STUFF -----
447     WR <= usbctl(0);
448     RD <= usbctl(1);
449     OE <= usbctl(2);
450
451     usbrdy(0) <= have_space;
452     usbrdy(1) <= have_pkt_rdy;
453
454     clear_status <= FX2_1;
455     FX2_2 <= rx_ouerrun;
456     FX2_3 <= tx_underrun;
457
458     -- Tri-state bus macro
459     bustri_0 : bustri
460         port map(data => usbdata_out, enabledt => OE, tridata => usbdata);
461
462     clk64 <= master_clk;
463
464     -- TRANSMIT SIDE -----
465     tx_en_t : if(TX_EN) generate
466         bb_tx_i0 <= ch0tx;
467         bb_tx_q0 <= chltx;
468         bb_tx_i1 <= ch2tx;
469         bb_tx_q1 <= ch3tx;
470
471         tx_buffer_0 : tx_buffer
472             port map(usbclk => usbclk, bus_reset => tx_bus_reset, usbdata => usbdata, WR => WR,
473                 have_space => have_space, tx_underrun => tx_underrun, clear_status =>
474                     clear_status,
475                 txclk => clk64, reset => tx_dsp_reset, channels(3 downto 1) => tx_numchan,
476                 channels(0) => '0', tx_i_0 => ch0tx, tx_q_0 => chltx, tx_i_1 => ch2tx, tx_q_1 =>
477                     ch3tx,
478                 txstrobe => strobe_interp, tx_empty => tx_empty, debugbus => tx_debugbus);
479
480         -- FIXME Should freq really be 0?
481         tx_en_0_t : if(TX_EN_0) generate
482             tx_chain_0 : tx_chain
483                 port map(clock => clk64, reset => tx_dsp_reset, enable => enable_tx,
484                     interp_rate => interp_rate, sample_strobe => tx_sample_strobe,
485                     interpolator_strobe => strobe_interp, freq => (others => '0'), i_in =>
486                         bb_tx_i0,
487                     q_in => bb_tx_q0, i_out => i_out_0, q_out => q_out_0);
488         end generate tx_en_0_t;
489
490         tx_en_0_f : if(not TX_EN_0) generate
491             i_out_0 <= (others => '0');
492             q_out_0 <= (others => '0');
493         end generate tx_en_0_f;
494
495         -- FIXME should freq really be 0?
496         tx_en_1_t : if(TX_EN_1) generate
497             tx_chain_1 : tx_chain
498                 port map(clock => clk64, reset => tx_dsp_reset, enable => enable_tx,

```

```

496         interp_rate => interp_rate , sample_strobe => tx_sample_strobe ,
497         interpolator_strobe => strobe_interp , freq => (others => '0') , i_in =>
           bb_tx_il ,
498         q_in => bb_tx_q1 , i_out => i_out_1 , q_out => q_out_1);
499     end generate tx_en_1_t;
500
501     tx_en_1_f : if(not TX_EN_1) generate
502         i_out_1 <= (others => '0');
503         q_out_1 <= (others => '0');
504     end generate tx_en_1_f;
505
506     sr_txmux : setting_reg
507         generic map(my_addr => FR_TX_MUX)
508         port map(clock => clk64 , reset => tx_dsp_reset , strobe => serial_strobe , addr =>
           serial_addr ,
509         d_in => serial_data , d_out(31 downto 20) => OPEN , d_out(19 downto 16) => dac3mux ,
510         d_out(15 downto 12) => dac2mux , d_out(11 downto 8) => dac1mux , d_out(7 downto 4)
           => dac0mux ,
511         d_out(3) => tx_realsignals , d_out(2 downto 0) => tx_numchan , changed => OPEN);
512
513     tx_mux_0 : usrp_mux
514         port map(dac_mux => dac0mux , i_out_0 => i_out_0 , q_out_0 => q_out_0 , i_out_1 =>
           i_out_1 ,
515         q_out_1 => q_out_1 , d_out => tx_a_a);
516
517     tx_mux_1 : usrp_mux
518         port map(dac_mux => dac1mux , i_out_0 => i_out_0 , q_out_0 => q_out_0 , i_out_1 =>
           i_out_1 ,
519         q_out_1 => q_out_1 , d_out => tx_b_a);
520
521     tx_mux_2 : usrp_mux
522         port map(dac_mux => dac2mux , i_out_0 => i_out_0 , q_out_0 => q_out_0 , i_out_1 =>
           i_out_1 ,
523         q_out_1 => q_out_1 , d_out => tx_a_b);
524
525     tx_mux_3 : usrp_mux
526         port map(dac_mux => dac3mux , i_out_0 => i_out_0 , q_out_0 => q_out_0 , i_out_1 =>
           i_out_1 ,
527         q_out_1 => q_out_1 , d_out => tx_b_b);
528
529     txsync <= tx_sample_strobe;
530     TXSYNC_A <= txsync;
531     TXSYNC_B <= txsync;
532
533     tx_a <= tx_b_a(15 downto 2) when txsync = '1' else tx_a_a(15 downto 2);
534     tx_b <= tx_b_b(15 downto 2) when txsync = '1' else tx_a_b(15 downto 2);
535     end generate tx_en_t;
536
537     tx_en_f : if(not TX_EN) generate
538         -- TODO Assign signals!
539     end generate tx_en_f;
540
541     -- RECEIVE SIDE -----
542     rx_en_t : if(RX_EN) generate
543         loopback <= settings(0);
544         counter <= settings(1);
545

```

```

546     process(clk64)
547     begin
548         if(clk64'EVENT and clk64 = '1') then
549             if(rx_dsp_reset = '1') then
550                 debug_counter <= (others => '0');
551             elsif(enable_rx /= '1') then
552                 debug_counter <= (others => '0');
553             elsif(hb_strobe = '1') then
554                 debug_counter <= debug_counter + 2;
555             end if;
556         end if;
557     end process;
558
559     process(clk64)
560     begin
561         if(clk64'EVENT and clk64 = '1') then
562             if(strobe_interp = '1') then
563                 loopback_i_0 <= ch0tx;
564                 loopback_q_0 <= chltx;
565             end if;
566         end if;
567     end process;
568
569     ch0rx <= debug_counter when (counter = '1') else loopback_i_0 when (loopback = '1') else
570         bb_rx_i0;
571     chlrx <= debug_counter + 1 when (counter = '1') else loopback_q_0 when (loopback = '1')
572         else bb_rx_q0;
573     ch2rx <= bb_rx_i1;
574     ch3rx <= bb_rx_q1;
575     ch4rx <= bb_rx_i2;
576     ch5rx <= bb_rx_q2;
577     ch6rx <= bb_rx_i3;
578     ch7rx <= bb_rx_q3;
579
580     adc_interface_0 : adc_interface
581     port map(clock => clk64, reset => rx_dsp_reset, enable => '1', serial_addr =>
582         serial_addr,
583         serial_data => serial_data, serial_strobe => serial_strobe, rx_a_a => rx_a_a,
584         rx_b_a => rx_b_a, rx_a_b => rx_a_b, rx_b_b => rx_b_b, rssi_0 => rssi_0,
585         rssi_1 => rssi_1, rssi_2 => rssi_2, rssi_3 => rssi_3, ddc0_in_i => ddc0_in_i,
586         ddc0_in_q => ddc0_in_q, ddcl_in_i => ddcl_in_i, ddcl_in_q => ddcl_in_q,
587         ddc2_in_i => ddc2_in_i, ddc2_in_q => ddc2_in_q, ddc3_in_i => ddc3_in_i,
588         ddc3_in_q => ddc3_in_q, rx_numchan => rx_numchan);
589
590     rx_buffer_0 : rx_buffer
591     port map(usbclk => usbclk, bus_reset => rx_bus_reset, reset => rx_dsp_reset,
592         reset_regs => rx_dsp_reset, usbd_data => usbd_data_out, RD => RD, have_pkt_rdy =>
593         have_pkt_rdy,
594         rx_ouerrun => rx_ouerrun, channels => rx_numchan, ch_0 => ch0rx, ch_1 => chlrx,
595         ch_2 => ch2rx, ch_3 => ch3rx, ch_4 => ch4rx, ch_5 => ch5rx, ch_6 => ch6rx, ch_7
596         => ch7rx,
597         rxclk => clk64, rxstrobe => hb_strobe, clear_status => clear_status,
598         serial_addr => serial_addr, serial_data => serial_data, serial_strobe =>
599         serial_strobe,
600         debugbus => rx_debugbus);
601
602     rx_en_0_t : if(RX_EN_0) generate

```

```

597     rx_chain_0 : rx_chain
598         generic map(FREQADDR => FR_RX_FREQ_0, PHASEADDR => FR_RX_PHASE_0)
599         port map(clock => clk64, reset => '0', enable => enable_rx, decim_rate =>
600             decim_rate,
601             sample_strobe => rx_sample_strobe, decimator_strobe => strobe_decim,
602             hb_strobe => hb_strobe, serial_addr => serial_addr, serial_data =>
603                 serial_data,
604             serial_strobe => serial_strobe, i_in => ddc0_in_i, q_in => ddc0_in_q, i_out
605                 => bb_rx_i0,
606             q_out => bb_rx_q0, debugdata => debugdata, debugctrl => debugctrl);
607     end generate rx_en_0_t;
608
609     rx_en_0_f : if(not RX_EN_0) generate
610         bb_rx_i0 <= (others => '0');
611         bb_rx_q0 <= (others => '0');
612     end generate rx_en_0_f;
613
614     rx_en_1_t : if(RX_EN_1) generate
615         rx_chain_1 : rx_chain
616             generic map(FREQADDR => FR_RX_FREQ_1, PHASEADDR => FR_RX_PHASE_1)
617             port map(clock => clk64, reset => '0', enable => enable_rx, decim_rate =>
618                 decim_rate,
619                 sample_strobe => rx_sample_strobe, decimator_strobe => strobe_decim,
620                 hb_strobe => OPEN, serial_addr => serial_addr, serial_data => serial_data,
621                 serial_strobe => serial_strobe, i_in => ddc1_in_i, q_in => ddc1_in_q, i_out
622                     => bb_rx_i1,
623                 q_out => bb_rx_q1, debugdata => OPEN, debugctrl => OPEN);
624     end generate rx_en_1_t;
625
626     rx_en_1_f : if(not RX_EN_1) generate
627         bb_rx_i1 <= (others => '0');
628         bb_rx_q1 <= (others => '0');
629     end generate rx_en_1_f;
630
631     rx_en_2_t : if(RX_EN_2) generate
632         rx_chain_2 : rx_chain
633             generic map(FREQADDR => FR_RX_FREQ_2, PHASEADDR => FR_RX_PHASE_2)
634             port map(clock => clk64, reset => '0', enable => enable_rx, decim_rate =>
635                 decim_rate,
636                 sample_strobe => rx_sample_strobe, decimator_strobe => strobe_decim,
637                 hb_strobe => OPEN, serial_addr => serial_addr, serial_data => serial_data,
638                 serial_strobe => serial_strobe, i_in => ddc2_in_i, q_in => ddc2_in_q, i_out
639                     => bb_rx_i2,
640                 q_out => bb_rx_q2, debugdata => OPEN, debugctrl => OPEN);
641     end generate rx_en_2_t;
642
643     rx_en_2_f : if(not RX_EN_2) generate
644         bb_rx_i2 <= (others => '0');
645         bb_rx_q2 <= (others => '0');
646     end generate rx_en_2_f;
647
648     rx_en_3_t : if(RX_EN_3) generate
649         rx_chain_3 : rx_chain
650             generic map(FREQADDR => FR_RX_FREQ_3, PHASEADDR => FR_RX_PHASE_3)
651             port map(clock => clk64, reset => '0', enable => enable_rx, decim_rate =>
652                 decim_rate,
653                 sample_strobe => rx_sample_strobe, decimator_strobe => strobe_decim,

```



```

646         hb_strobe => OPEN, serial_addr => serial_addr , serial_data => serial_data ,
647         serial_strobe => serial_strobe , i_in => ddc3_in_i , q_in => ddc3_in_q , i_out
           => bb_rx_i3 ,
648         q_out => bb_rx_q3 , debugdata => OPEN, debugctrl => OPEN);
649     end generate rx_en_3_t;
650
651     rx_en_3_f : if(not RX_EN3) generate
652         bb_rx_i3 <= (others => '0');
653         bb_rx_q3 <= (others => '0');
654     end generate rx_en_3_f;
655 end generate rx_en_t;
656
657 rx_en_f : if(not RX_EN) generate
658     -- TODO Assign signals!
659 end generate rx_en_f;
660
661 -- CONTROL SIGNALS -----
662 capabilities(31 downto 8) <= (others => '0');
663 capabilities(7) <= '1' when TX_CAP_HB else '0';
664 capabilities(6 downto 4) <= TX_CAP_NCHAN;
665 capabilities(3) <= '1' when RX_CAP_HB else '0';
666 capabilities(2 downto 0) <= RX_CAP_NCHAN;
667
668 -- readback_3: 0xF0F0931A is greater than interger 'high', so I used binary notation instead.
669 serial_io_0 : serial_io
670     port map(master_clk => clk64 , serial_clock => SCLK, serial_data_in => SDI, enable =>
           SEN_FPGA,
671     reset => '0', serial_data_out => SDO, serial_addr => serial_addr , serial_data =>
           serial_data ,
672     serial_strobe => serial_strobe , readback_0(31 downto 16) => io_rx_a ,
673     readback_0(15 downto 0) => io_tx_a , readback_1(31 downto 16) => io_rx_b ,
674     readback_1(15 downto 0) => io_tx_b , readback_2 => capabilities ,
675     readback_3 => "11110000111100001001001100011010" , readback_4 => rssi_0 , readback_5 =>
           rssi_1 ,
676     readback_6 => rssi_2 , readback_7 => rssi_3);
677
678 master_control_0 : master_control
679     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 ,
681     rx_bus_reset => rx_bus_reset , tx_dsp_reset => tx_dsp_reset , rx_dsp_reset =>
           rx_dsp_reset ,
682     enable_tx => enable_tx , enable_rx => enable_rx , interp_rate => interp_rate ,
683     decim_rate => decim_rate , tx_sample_strobe => tx_sample_strobe , strobe_interp =>
           strobe_interp ,
684     rx_sample_strobe => rx_sample_strobe , strobe_decim => strobe_decim , tx_empty =>
           tx_empty ,
685     debug_0 => tx_debugbus(15 downto 0) , debug_1 => tx_debugbus(31 downto 16) ,
686     debug_2 => rx_debugbus(15 downto 0) , debug_3 => rx_debugbus(31 downto 16) , reg_0 =>
           reg_0 ,
687     reg_1 => reg_1 , reg_2 => reg_2 , reg_3 => reg_3);
688
689 io_pins_0 : io_pins
690     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 ,
691     reg_1 => reg_1 , reg_2 => reg_2 , reg_3 => reg_3 , clock => clk64 , rx_reset =>
           rx_dsp_reset ,

```

```
692         tx_reset => tx_dsp_reset , serial_addr => serial_addr , serial_data => serial_data ,
693         serial_strobe => serial_strobe);
694
695 -- MISC SETTINGS -----
696 sr_misc : setting_reg
697     generic map(my_addr => FRMODE)
698     port map(clock => clk64 , reset => rx_dsp_reset , strobe => serial_strobe , addr =>
        serial_addr ,
699         d_in => serial_data , d_out(31 downto 8) => OPEN, d_out(7 downto 0) => settings ,
700         changed => OPEN);
701 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)

```

1  #-----
2  # Xilinx Board Definition for the SPA-U FPGA Board
3  # Last Modified: 13 April 2011
4  # XBD Author: Steve Olivieri
5  #
6  # Copyright (C) 2010 COSMIAC
7  #
8  # This program is free software; you can redistribute it and/or modify
9  # it under the terms of the GNU General Public License as published by
10 # the Free Software Foundation; either version 2 of the License, or
11 # (at your option) any later version.
12 #
13 # This program is distributed in the hope that it will be useful,
14 # but WITHOUT ANY WARRANTY; without even the implied warranty of
15 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 # GNU General Public License for more details.
17 #
18 # You should have received a copy of the GNU General Public License
19 # along with this program; if not, write to the Free Software
20 # Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 #-----
22
23 ### TODO ###
24 # USB Reset (H/L? Pin?)
25 # DDR2 Memory (MIG output)
26 # Flash Memory
27 # I2C Controller

```

```

28 # Fix SPI SS Bus (USRP, # of SS)
29
30 # Vendor Information
31 ATTRIBUTE VENDOR = COSMIAC
32 ATTRIBUTE SPEC_URL = www.cosmiac.com
33 ATTRIBUTE CONTACT_INFO_URL = http://www.cosmiac.com/
34
35 # Board Information
36 ATTRIBUTE NAME = SPA-U FPGA Board
37 ATTRIBUTE REVISION = B
38 ATTRIBUTE DESC = SPA-U FPGA Board
39 ATTRIBUTE LONG_DESC = 'The SPA-U FPGA Board utilizes a Xilinx Spartan-3A
40 XCS1400A-5FG484 device and has DDR2 memory, flash memory, USB2, I2C, SPI, RS232,
41 Ethernet, and GPIO capabilities.'
42
43 # Primary Clock (50MHz)
44 BEGIN IO_INTERFACE
45     ATTRIBUTE IOTYPE = XIL_CLOCK_V1
46     ATTRIBUTE INSTANCE = CLOCK_50MHZ
47
48     PARAMETER CLK_FREQ = 50000000, IO_IS = clk_freq, RANGE = (50000000)
49
50     PORT CLK = OSC_50_MHZ, IO_IS = ext_clk
51 END
52
53 # Second Clock (75MHz)
54 BEGIN IO_INTERFACE
55     ATTRIBUTE IOTYPE = XIL_CLOCK_V1
56     ATTRIBUTE INSTANCE = CLOCK_75MHZ
57
58     PARAMETER CLK_FREQ = 75000000, IO_IS = clk_freq, RANGE = (75000000)
59
60     PORT CLK = OSC_75_MHZ, IO_IS = ext_clk
61 END
62
63 # Third Clock (100MHz)
64 BEGIN IO_INTERFACE
65     ATTRIBUTE IOTYPE = XIL_CLOCK_V1
66     ATTRIBUTE INSTANCE = CLOCK_100MHZ
67
68     PARAMETER CLK_FREQ = 100000000, IO_IS = clk_freq, RANGE = (100000000)
69
70     PORT CLK = OSC_100_MHZ, IO_IS = ext_clk
71 END
72
73 # Reset Button
74 BEGIN IO_INTERFACE
75     ATTRIBUTE IOTYPE = XIL_RESET_V1
76     ATTRIBUTE INSTANCE = RESET
77
78     PARAMETER RST_POLARITY = 1, IO_IS = polarity, VALUE_NOTE = Active High
79
80     PORT RESET = CONN_RESET, IO_IS = ext_rst
81 END
82
83 # Primary UART
84 BEGIN IO_INTERFACE

```

```

85     ATTRIBUTE IOTYPE = XIL_UART_V1
86     ATTRIBUTE INSTANCE = RS232_PRIMARY
87
88     PORT FROM_RS232 = CONN_FROM_RS232, IO_IS = serial_in
89     PORT TO_RS232 = CONN_TO_RS232, IO_IS = serial_out, INITIALVAL = GND
90 END
91
92 # AT90 UART
93 BEGIN IO_INTERFACE
94     ATTRIBUTE IOTYPE = XIL_UART_V1
95     ATTRIBUTE INSTANCE = RS232_AT90
96
97     PORT AT90_RXD1 = CONN_AT90_RXD1, IO_IS = serial_in
98     PORT AT90_TXD1 = CONN_AT90_TXD1, IO_IS = serial_out, INITIALVAL = GND
99 END
100
101 # LEDs
102 BEGIN IO_INTERFACE
103     ATTRIBUTE IOTYPE = XIL_GPIO_V1
104     ATTRIBUTE INSTANCE = LEDs_8BIT
105
106     PARAMETER num_bits = 8, IO_IS = num_bits
107     PARAMETER is_dual = 0, IO_IS = is_dual
108     PARAMETER bidir_data = 0, IO_IS = is_bidir
109     PARAMETER all_inputs = 0, IO_IS = all_inputs
110
111     PORT LED0 = CONN_LED0, IO_IS = gpio_data_out[0], INITIALVAL = VCC
112     PORT LED1 = CONN_LED1, IO_IS = gpio_data_out[1], INITIALVAL = VCC
113     PORT LED2 = CONN_LED2, IO_IS = gpio_data_out[2], INITIALVAL = VCC
114     PORT LED3 = CONN_LED3, IO_IS = gpio_data_out[3], INITIALVAL = VCC
115     PORT LED4 = CONN_LED4, IO_IS = gpio_data_out[4], INITIALVAL = VCC
116     PORT LED5 = CONN_LED5, IO_IS = gpio_data_out[5], INITIALVAL = VCC
117     PORT LED6 = CONN_LED6, IO_IS = gpio_data_out[6], INITIALVAL = VCC
118     PORT LED7 = CONN_LED7, IO_IS = gpio_data_out[7], INITIALVAL = VCC
119 END
120
121 # Push Button
122 # The button is currently used for the system reset.
123 #BEGIN IO_INTERFACE
124 # ATTRIBUTE IOTYPE = XIL_GPIO_V1
125 # ATTRIBUTE INSTANCE = PUSH_BUTTON
126 #
127 # PARAMETER num_bits = 1, IO_IS = num_bits
128 # PARAMETER is_dual = 0, IO_IS = is_dual
129 # PARAMETER bidir_data = 0, IO_IS = is_bidir
130 # PARAMETER all_inputs = 1, IO_IS = all_inputs
131 #
132 # PORT BUTTON = CONN_BUTTON, IO_IS = gpio_data_in[0]
133 #END
134
135 # AT90 I/O Port A
136 BEGIN IO_INTERFACE
137     ATTRIBUTE IOTYPE = XIL_GPIO_V1
138     ATTRIBUTE INSTANCE = AT90_PA
139
140     PARAMETER num_bits = 8, IO_IS = num_bits
141     PARAMETER is_dual = 0, IO_IS = is_dual

```

```

142 PARAMETER bidir_data = 0, IO_IS = is_bidir
143 PARAMETER all_inputs = 0, IO_IS = all_inputs
144
145 PORT AT_PA0 = CONN_AT_PA0, IO_IS = gpio_data_out[0], INITIALVAL = GND
146 PORT AT_PA1 = CONN_AT_PA1, IO_IS = gpio_data_out[1], INITIALVAL = GND
147 PORT AT_PA2 = CONN_AT_PA2, IO_IS = gpio_data_out[2], INITIALVAL = GND
148 PORT AT_PA3 = CONN_AT_PA3, IO_IS = gpio_data_out[3], INITIALVAL = GND
149 PORT AT_PA4 = CONN_AT_PA4, IO_IS = gpio_data_out[4], INITIALVAL = GND
150 PORT AT_PA5 = CONN_AT_PA5, IO_IS = gpio_data_out[5], INITIALVAL = GND
151 PORT AT_PA6 = CONN_AT_PA6, IO_IS = gpio_data_out[6], INITIALVAL = GND
152 PORT AT_PA7 = CONN_AT_PA7, IO_IS = gpio_data_out[7], INITIALVAL = GND
153 END
154
155 # AT90 I/O Port C
156 BEGIN IO_INTERFACE
157     ATTRIBUTE IOTYPE = XIL_GPIO_V1
158     ATTRIBUTE INSTANCE = AT90_PC
159
160     PARAMETER num_bits = 8, IO_IS = num_bits
161     PARAMETER is_dual = 0, IO_IS = is_dual
162     PARAMETER bidir_data = 0, IO_IS = is_bidir
163     PARAMETER all_inputs = 0, IO_IS = all_inputs
164
165     PORT AT_PC0 = CONN_AT_PC0, IO_IS = gpio_data_out[0], INITIALVAL = GND
166     PORT AT_PC1 = CONN_AT_PC1, IO_IS = gpio_data_out[1], INITIALVAL = GND
167     PORT AT_PC2 = CONN_AT_PC2, IO_IS = gpio_data_out[2], INITIALVAL = GND
168     PORT AT_PC3 = CONN_AT_PC3, IO_IS = gpio_data_out[3], INITIALVAL = GND
169     PORT AT_PC4 = CONN_AT_PC4, IO_IS = gpio_data_out[4], INITIALVAL = GND
170     PORT AT_PC5 = CONN_AT_PC5, IO_IS = gpio_data_out[5], INITIALVAL = GND
171     PORT AT_PC6 = CONN_AT_PC6, IO_IS = gpio_data_out[6], INITIALVAL = GND
172     PORT AT_PC7 = CONN_AT_PC7, IO_IS = gpio_data_out[7], INITIALVAL = GND
173 END
174
175 # AT90 I/O Port F
176 BEGIN IO_INTERFACE
177     ATTRIBUTE IOTYPE = XIL_GPIO_V1
178     ATTRIBUTE INSTANCE = AT90_PF
179
180     PARAMETER num_bits = 4, IO_IS = num_bits
181     PARAMETER is_dual = 0, IO_IS = is_dual
182     PARAMETER bidir_data = 0, IO_IS = is_bidir
183     PARAMETER all_inputs = 0, IO_IS = all_inputs
184
185     PORT AT_PF0 = CONN_AT_PF0, IO_IS = gpio_data_out[0], INITIALVAL = GND
186     PORT AT_PF1 = CONN_AT_PF1, IO_IS = gpio_data_out[1], INITIALVAL = GND
187     PORT AT_PF2 = CONN_AT_PF2, IO_IS = gpio_data_out[2], INITIALVAL = GND
188     PORT AT_PF3 = CONN_AT_PF3, IO_IS = gpio_data_out[3], INITIALVAL = GND
189 END
190
191 # Ethernet
192 BEGIN IO_INTERFACE
193     ATTRIBUTE IOTYPE = XIL_ETHERNET_V1
194     ATTRIBUTE INSTANCE = ETHERNET
195
196     PORT E_COL = CONN_E_COL, IO_IS = ETH_COL
197     PORT E_CRS = CONN_E_CRS, IO_IS = ETH_CRS
198     PORT E_MDC = CONN_E_MDC, IO_IS = ETH_MDC

```

```

199 PORT E_MDIO = CONN_E_MDIO, IO_IS = ETH_MDIO
200 PORT E_NRST = CONN_E_NRST, IO_IS = PHY_RESETh
201
202 PORT E_RX_CLK = CONN_E_RX_CLK, IO_IS = ETH_RXC
203 PORT E_RX_DV = CONN_E_RX_DV, IO_IS = ETH_RXDV
204 PORT E_RXD0 = CONN_E_RXD0, IO_IS = ETH_RXD[0]
205 PORT E_RXD1 = CONN_E_RXD1, IO_IS = ETH_RXD[1]
206 PORT E_RXD2 = CONN_E_RXD2, IO_IS = ETH_RXD[2]
207 PORT E_RXD3 = CONN_E_RXD3, IO_IS = ETH_RXD[3]
208 PORT E_RX_ER = CONN_E_RX_ER, IO_IS = ETH_RXER
209
210 PORT E_TX_CLK = CONN_E_TX_CLK, IO_IS = ETH_TXC
211 PORT E_TX_EN = CONN_E_TX_EN, IO_IS = ETH_TXEN
212 PORT E_TXD0 = CONN_E_TXD0, IO_IS = ETH_TXD[0]
213 PORT E_TXD1 = CONN_E_TXD1, IO_IS = ETH_TXD[1]
214 PORT E_TXD2 = CONN_E_TXD2, IO_IS = ETH_TXD[2]
215 PORT E_TXD3 = CONN_E_TXD3, IO_IS = ETH_TXD[3]
216 PORT E_TX_ER = CONN_E_TX_ER, IO_IS = ETH_TXER
217 END
218
219 # USB Controller
220 BEGIN IO_INTERFACE
221 ATTRIBUTE IOTYPE = XIL_USB2DEVICE_V4
222 ATTRIBUTE INSTANCE = USB_CONTROLLER
223
224 PARAMETER C_INCLUDE_DMA = 1, IO_IS = C_INCLUDE_DMA
225 PARAMETER C_PHY_RESET_TYPE = 1, IO_IS = C_PHY_RESET_TYPE
226
227 PORT USB_CLKOUT = CONN_USB_CLKOUT, IO_IS = ULPI_Clock
228 PORT USB_DIR = CONN_USB_DIR, IO_IS = ULPI_Dir
229 PORT USB_NXT = CONN_USB_NXT, IO_IS = ULPI_Next
230 PORT USB_STP = CONN_USB_STP, IO_IS = ULPI_Stop
231 # PORT USB_RESET = CONN_USB_RESET, IO_IS = ULPI_Reset
232 PORT USB_RESET = net_gnd, IO_IS = ULPI_Reset
233
234 PORT USB_D0 = CONN_USB_D0, IO_IS = ULPI_Data[0]
235 PORT USB_D1 = CONN_USB_D1, IO_IS = ULPI_Data[1]
236 PORT USB_D2 = CONN_USB_D2, IO_IS = ULPI_Data[2]
237 PORT USB_D3 = CONN_USB_D3, IO_IS = ULPI_Data[3]
238 PORT USB_D4 = CONN_USB_D4, IO_IS = ULPI_Data[4]
239 PORT USB_D5 = CONN_USB_D5, IO_IS = ULPI_Data[5]
240 PORT USB_D6 = CONN_USB_D6, IO_IS = ULPI_Data[6]
241 PORT USB_D7 = CONN_USB_D7, IO_IS = ULPI_Data[7]
242 END
243
244 # DDR2 Memory
245 BEGIN IO_INTERFACE
246 ATTRIBUTE IOTYPE = XIL_MEMORY_V1
247 ATTRIBUTE INSTANCE = DDR2_SDRAM
248
249 # Controller Settings
250 PARAMETER C_NUM_PORTS = 1, IO_IS = C_NUM_PORTS
251
252 # Micron MT47H128M16HG-3-IT:A
253 PARAMETER C_MEM_TYPE = DDR2, IO_IS = C_MEM_TYPE
254 PARAMETER C_MEM_PARTNO="CUSTOM", IO_IS=C_MEM_PARTNO
255

```

```

256 # 256MB
257 PARAMETER C_BASEADDR = 0x00000000, IO_IS = C_BASEADDR
258 PARAMETER C_HIGHADDR = 0x0fffffff, IO_IS = C_HIGHADDR
259
260 # Memory/DIMM Settings
261 PARAMETER C_MEM_ADDR_WIDTH = 14, IO_IS = C_MEM_ADDR_WIDTH
262 PARAMETER C_MEM_BANK_ADDR_WIDTH = 3, IO_IS = C_MEM_BANK_ADDR_WIDTH
263 PARAMETER C_MEM_DM_WIDTH = 2, IO_IS = C_MEM_DM_WIDTH
264 PARAMETER C_MEM_DQS_WIDTH = 2, IO_IS = C_MEM_DQS_WIDTH
265 PARAMETER C_MEM_DATA_WIDTH = 16, IO_IS = C_MEM_DATA_WIDTH
266 PARAMETER C_DDR2_DQSN_ENABLE = 1, IO_IS = C_DDR2_DQSN_ENABLE
267 PARAMETER C_MPMC_CLK0_PERIOD_PS = 3000, IO_IS = C_MPMC_CLK0_PERIOD_PS # ps
268
269 # Memory Part Settings
270 PARAMETER C_MEM_PART_DATA_DEPTH = 128, IO_IS = C_MEM_PART_DATA_DEPTH
271 PARAMETER C_MEM_PART_DATA_WIDTH = 16, IO_IS = C_MEM_PART_DATA_WIDTH
272 PARAMETER C_MEM_PART_NUM_BANK_BITS = 3, IO_IS = C_MEM_PART_NUM_BANK_BITS
273 PARAMETER C_MEM_PART_NUM_ROW_BITS = 14, IO_IS = C_MEM_PART_NUM_ROW_BITS
274 PARAMETER C_MEM_PART_NUM_COL_BITS = 10, IO_IS = C_MEM_PART_NUM_COL_BITS
275
276 # Memory Timing Settings
277 PARAMETER C_MEM_PART_CAS_A_FMAX = 667, IO_IS = C_MEM_PART_CAS_A_FMAX # MHz
278 PARAMETER C_MEM_PART_CAS_A = 5, IO_IS = C_MEM_PART_CAS_A
279 PARAMETER C_MEM_PART_TTRD = 10000, IO_IS = C_MEM_PART_TTRD # ps
280 PARAMETER C_MEM_PART_TM RD = 2, IO_IS = C_MEM_PART_TM RD # tCK
281 PARAMETER C_MEM_PART_TRCD = 15000, IO_IS = C_MEM_PART_TRCD # ps
282 PARAMETER C_MEM_PART_TCCD = 2, IO_IS = C_MEM_PART_TCCD # tCK
283 PARAMETER C_MEM_PART_TWR = 15000, IO_IS = C_MEM_PART_TWR # ps
284 PARAMETER C_MEM_PART_TWTR = 7500, IO_IS = C_MEM_PART_TWTR # ps
285 PARAMETER C_MEM_PART_TREFI = 3900000, IO_IS = C_MEM_PART_TREFI # ps
286 PARAMETER C_MEM_PART_TRFC = 195000, IO_IS = C_MEM_PART_TRFC # ps
287 PARAMETER C_MEM_PART_TRP = 15000, IO_IS = C_MEM_PART_TRP # ps
288 PARAMETER C_MEM_PART_TRC = 60000, IO_IS = C_MEM_PART_TRC # ps
289 PARAMETER C_MEM_PART_TRASMAX = 70000000, IO_IS = C_MEM_PART_TRASMAX # ps
290 PARAMETER C_MEM_PART_TRAS = 40000, IO_IS = C_MEM_PART_TRAS # ps
291
292 PORT DDR_ODT = ddr2_ddr_odt_0, IO_IS = ddr2_odt
293
294 PORT DDR_Addr_0 = ddr2_ddr_addr_0, IO_IS = ddr2_address [0]
295 PORT DDR_Addr_1 = ddr2_ddr_addr_1, IO_IS = ddr2_address [1]
296 PORT DDR_Addr_2 = ddr2_ddr_addr_2, IO_IS = ddr2_address [2]
297 PORT DDR_Addr_3 = ddr2_ddr_addr_3, IO_IS = ddr2_address [3]
298 PORT DDR_Addr_4 = ddr2_ddr_addr_4, IO_IS = ddr2_address [4]
299 PORT DDR_Addr_5 = ddr2_ddr_addr_5, IO_IS = ddr2_address [5]
300 PORT DDR_Addr_6 = ddr2_ddr_addr_6, IO_IS = ddr2_address [6]
301 PORT DDR_Addr_7 = ddr2_ddr_addr_7, IO_IS = ddr2_address [7]
302 PORT DDR_Addr_8 = ddr2_ddr_addr_8, IO_IS = ddr2_address [8]
303 PORT DDR_Addr_9 = ddr2_ddr_addr_9, IO_IS = ddr2_address [9]
304 PORT DDR_Addr_10 = ddr2_ddr_addr_10, IO_IS = ddr2_address [10]
305 PORT DDR_Addr_11 = ddr2_ddr_addr_11, IO_IS = ddr2_address [11]
306 PORT DDR_Addr_12 = ddr2_ddr_addr_12, IO_IS = ddr2_address [12]
307
308 PORT DDR_BankAddr_0 = ddr2_ddr_bankaddr_0, IO_IS = ddr2_BankAddr [0]
309 PORT DDR_BankAddr_1 = ddr2_ddr_bankaddr_1, IO_IS = ddr2_BankAddr [1]
310
311 PORT DDR_CASn = ddr2_ddr_casn, IO_IS = ddr2_col_addr_select
312 PORT DDR_RASn = ddr2_ddr_rasn, IO_IS = ddr2_row_addr_select

```



```

313 PORT DDR_CKE = ddr2_ddr_cke , IO_IS = ddr2_clk_enable
314 PORT DDR_CSn = ddr2_ddr_csn , IO_IS = ddr2_chip_select
315 PORT DDR_WEn = ddr2_ddr_wen , IO_IS = ddr2_write_enable
316
317 PORT DDR_CLK = ddr2_clk , IO_IS = ddr2_clk
318 PORT DDR_CLK_n = ddr2_clk_n , IO_IS = ddr2_clk_n
319
320 PORT DDR_DM0 = ddr2_ddr_dm_0 , IO_IS = ddr2_data_mask [0]
321 PORT DDR_DM1 = ddr2_ddr_dm_1 , IO_IS = ddr2_data_mask [1]
322 PORT DDR_DQS0 = ddr2_ddr_dqs_0 , IO_IS = ddr2_data_strobe [0]
323 PORT DDR_DQS1 = ddr2_ddr_dqs_1 , IO_IS = ddr2_data_strobe [1]
324 PORT DDR_DQSn_0 = ddr2_ddr_dqsn_0 , IO_IS = ddr2_data_strobe_n [0]
325 PORT DDR_DQSn_1 = ddr2_ddr_dqsn_1 , IO_IS = ddr2_data_strobe_n [1]
326
327 PORT DDR_DQ0 = ddr2_ddr_dq_0 , IO_IS = ddr2_data [0]
328 PORT DDR_DQ1 = ddr2_ddr_dq_1 , IO_IS = ddr2_data [1]
329 PORT DDR_DQ2 = ddr2_ddr_dq_2 , IO_IS = ddr2_data [2]
330 PORT DDR_DQ3 = ddr2_ddr_dq_3 , IO_IS = ddr2_data [3]
331 PORT DDR_DQ4 = ddr2_ddr_dq_4 , IO_IS = ddr2_data [4]
332 PORT DDR_DQ5 = ddr2_ddr_dq_5 , IO_IS = ddr2_data [5]
333 PORT DDR_DQ6 = ddr2_ddr_dq_6 , IO_IS = ddr2_data [6]
334 PORT DDR_DQ7 = ddr2_ddr_dq_7 , IO_IS = ddr2_data [7]
335 PORT DDR_DQ8 = ddr2_ddr_dq_8 , IO_IS = ddr2_data [8]
336 PORT DDR_DQ9 = ddr2_ddr_dq_9 , IO_IS = ddr2_data [9]
337 PORT DDR_DQ10 = ddr2_ddr_dq_10 , IO_IS = ddr2_data [10]
338 PORT DDR_DQ11 = ddr2_ddr_dq_11 , IO_IS = ddr2_data [11]
339 PORT DDR_DQ12 = ddr2_ddr_dq_12 , IO_IS = ddr2_data [12]
340 PORT DDR_DQ13 = ddr2_ddr_dq_13 , IO_IS = ddr2_data [13]
341 PORT DDR_DQ14 = ddr2_ddr_dq_14 , IO_IS = ddr2_data [14]
342 PORT DDR_DQ15 = ddr2_ddr_dq_15 , IO_IS = ddr2_data [15]
343
344 PORT DDR_DQS_DIV_I = ddr2_ddr_dqs_div_i , IO_IS = ddr2_dqs_div_i
345 PORT DDR_DQS_DIV_O = ddr2_ddr_dqs_div_o , IO_IS = ddr2_dqs_div_o
346 END
347
348 # SPI Controller
349 BEGIN IO_INTERFACE
350 ATTRIBUTE IOTYPE = XIL_SPI_V1
351 ATTRIBUTE INSTANCE = SPI_CONTROLLER
352
353 PARAMETER C_FIFO_EXIST = 1 , IO_IS=fifo_exist
354 PARAMETER C_SCK_RATIO = 32 , IO_IS=clk_freq
355 PARAMETER C_NUM_TRANSFER_BITS = 8 , IO_IS = num_transfer_bits
356 # PARAMETER C_NUM_SS_BITS = 4 , IO_IS = ss_bits
357 PARAMETER C_NUM_SS_BITS = 2 , IO_IS = ss_bits
358
359 PORT SPISEL = net_vcc , IO_IS = slave_select_n
360 PORT SEN_SPI_RX = CONN_SEN_SPI_RX , IO_IS = data_in
361 PORT SEN_SPI_TX = CONN_SEN_SPI_TX , IO_IS = data_out
362 PORT SEN_SPI_CLK = CONN_SEN_SPI_CLK , IO_IS = clk_out
363
364 PORT SEN_SS_0 = CONN_SEN_SS_0 , IO_IS = slave_select [0]
365 PORT SEN_SS_1 = CONN_SEN_SS_1 , IO_IS = slave_select [1] # SEN_OE_IO
366 # PORT SEN_SS_2 = CONN_SEN_SS_2 , IO_IS = slave_select [2]
367 # PORT SEN_SS_3 = CONN_SEN_SS_3 , IO_IS = slave_select [3]
368 END
369

```

```

370 # FPGA Definition
371 BEGIN FPGA
372     ATTRIBUTE INSTANCE      = fpga_0
373     ATTRIBUTE FAMILY        = spartan3a
374     ATTRIBUTE DEVICE        = xc3s1400a
375     ATTRIBUTE PACKAGE       = fg484
376     ATTRIBUTE SPEED_GRADE   = -5
377     ATTRIBUTE JTAG_POSITION = 2
378
379     ### CLOCKS ###
380     PORT CLOCK              = CONN_OSC_50_MHZ , UCF_NET_STRING = ( "LOC = V12", "IOSTANDARD = LVCMOS33"
381     )
381     PORT CLOCK              = CONN_OSC_75_MHZ , UCF_NET_STRING = ( "LOC = AA12", "IOSTANDARD =
382     LVCMOS33", "CLOCK_DEDICATED_ROUTE = FALSE" )
382     PORT CLOCK              = CONN_OSC_100_MHZ , UCF_NET_STRING = ( "LOC = C12", "IOSTANDARD =
383     LVCMOS33", "CLOCK_DEDICATED_ROUTE = FALSE" )
383
384     ### RESET ### Used the south button
385     PORT RESET              = CONN_RESET      , UCF_NET_STRING = ( "LOC = U7", "IOSTANDARD = LVCMOS33",
386     "PULLDOWN" )
386
387     ### UARTS ###
388     PORT FROM_RS232        = CONN_FROM_RS232 , UCF_NET_STRING = ( "LOC = R15", "IOSTANDARD = LVCMOS33"
389     )
389     PORT TO_RS232          = CONN_TO_RS232   , UCF_NET_STRING = ( "LOC = Y22", "IOSTANDARD = LVCMOS33"
390     )
390
391     PORT AT90_RXD1         = CONN_AT90_RXD1  , UCF_NET_STRING = ( "LOC = D18", "IOSTANDARD = LVCMOS33"
392     )
392     PORT AT90_TXD1         = CONN_AT90_TXD1  , UCF_NET_STRING = ( "LOC = D19", "IOSTANDARD = LVCMOS33"
393     )
393
394     ### LEDS ###
395     PORT LED0              = CONN_LED0       , UCF_NET_STRING = ( "LOC = AA19", "IOSTANDARD = LVCMOS33" )
396     PORT LED1              = CONN_LED1       , UCF_NET_STRING = ( "LOC = AB19", "IOSTANDARD = LVCMOS33" )
397     PORT LED2              = CONN_LED2       , UCF_NET_STRING = ( "LOC = V17", "IOSTANDARD = LVCMOS33" )
398     PORT LED3              = CONN_LED3       , UCF_NET_STRING = ( "LOC = W18", "IOSTANDARD = LVCMOS33" )
399     PORT LED4              = CONN_LED4       , UCF_NET_STRING = ( "LOC = W17", "IOSTANDARD = LVCMOS33" )
400     PORT LED5              = CONN_LED5       , UCF_NET_STRING = ( "LOC = Y18", "IOSTANDARD = LVCMOS33" )
401     PORT LED6              = CONN_LED6       , UCF_NET_STRING = ( "LOC = AA21", "IOSTANDARD = LVCMOS33" )
402     PORT LED7              = CONN_LED7       , UCF_NET_STRING = ( "LOC = AB21", "IOSTANDARD = LVCMOS33" )
403
404     ### PUSH BUTTON ###
405     # PORT BUTTON          = CONN_BUTTON     , UCF_NET_STRING = ( "LOC = U7", "IOSTANDARD = LVCMOS33",
406     "PULLDOWN" )
406
407     ### AT90 I/O Ports ###
408     PORT AT_PA0            = CONN_AT_PA0     , UCF_NET_STRING = ( "LOC = A6", "IOSTANDARD = LVCMOS33" )
409     PORT AT_PA1            = CONN_AT_PA1     , UCF_NET_STRING = ( "LOC = A7", "IOSTANDARD = LVCMOS33" )
410     PORT AT_PA2            = CONN_AT_PA2     , UCF_NET_STRING = ( "LOC = C7", "IOSTANDARD = LVCMOS33" )
411     PORT AT_PA3            = CONN_AT_PA3     , UCF_NET_STRING = ( "LOC = D7", "IOSTANDARD = LVCMOS33" )
412     PORT AT_PA4            = CONN_AT_PA4     , UCF_NET_STRING = ( "LOC = A5", "IOSTANDARD = LVCMOS33" )
413     PORT AT_PA5            = CONN_AT_PA5     , UCF_NET_STRING = ( "LOC = B6", "IOSTANDARD = LVCMOS33" )
414     PORT AT_PA6            = CONN_AT_PA6     , UCF_NET_STRING = ( "LOC = D6", "IOSTANDARD = LVCMOS33" )
415     PORT AT_PA7            = CONN_AT_PA7     , UCF_NET_STRING = ( "LOC = C6", "IOSTANDARD = LVCMOS33" )
416

```

```

417 PORT AT_PC0      = CONN_AT_PC0      , UCF_NET_STRING = ( "LOC = C10" , "IOSTANDARD = LVCMOS33"
    )
418 PORT AT_PC1      = CONN_AT_PC1      , UCF_NET_STRING = ( "LOC = A10" , "IOSTANDARD = LVCMOS33"
    )
419 PORT AT_PC2      = CONN_AT_PC2      , UCF_NET_STRING = ( "LOC = A8" , "IOSTANDARD = LVCMOS33" )
420 PORT AT_PC3      = CONN_AT_PC3      , UCF_NET_STRING = ( "LOC = A9" , "IOSTANDARD = LVCMOS33" )
421 PORT AT_PC4      = CONN_AT_PC4      , UCF_NET_STRING = ( "LOC = E10" , "IOSTANDARD = LVCMOS33"
    )
422 PORT AT_PC5      = CONN_AT_PC5      , UCF_NET_STRING = ( "LOC = D10" , "IOSTANDARD = LVCMOS33"
    )
423 PORT AT_PC6      = CONN_AT_PC6      , UCF_NET_STRING = ( "LOC = C9" , "IOSTANDARD = LVCMOS33" )
424 PORT AT_PC7      = CONN_AT_PC7      , UCF_NET_STRING = ( "LOC = B9" , "IOSTANDARD = LVCMOS33" )
425
426 PORT AT_PF0      = CONN_AT_PF0      , UCF_NET_STRING = ( "LOC = B17" , "IOSTANDARD = LVCMOS33"
    )
427 PORT AT_PF1      = CONN_AT_PF1      , UCF_NET_STRING = ( "LOC = A17" , "IOSTANDARD = LVCMOS33"
    )
428 PORT AT_PF2      = CONN_AT_PF2      , UCF_NET_STRING = ( "LOC = D15" , "IOSTANDARD = LVCMOS33"
    )
429 PORT AT_PF3      = CONN_AT_PF3      , UCF_NET_STRING = ( "LOC = C15" , "IOSTANDARD = LVCMOS33"
    )
430
431 ### Ethernet ###
432 PORT E_COL       = CONN_E_COL       , UCF_NET_STRING = ( "LOC = AB3" , "IOSTANDARD =
    LVCMOS33" , "PULLDOWN" )
433 PORT E_CRCS      = CONN_E_CRCS      , UCF_NET_STRING = ( "LOC = AA4" , "IOSTANDARD =
    LVCMOS33" , "PULLDOWN" )
434 PORT E_MDC       = CONN_E_MDC       , UCF_NET_STRING = ( "LOC = AB4" , "IOSTANDARD = LVCMOS33"
    )
435 PORT E_MDIO      = CONN_E_MDIO      , UCF_NET_STRING = ( "LOC = AB5" , "IOSTANDARD = LVCMOS33"
    )
436 PORT E_NRST      = CONN_E_NRST      , UCF_NET_STRING = ( "LOC = Y5" , "IOSTANDARD = LVCMOS33" )
437 PORT E_RX_CLK    = CONN_E_RX_CLK    , UCF_NET_STRING = ( "LOC = W7" , "IOSTANDARD = LVCMOS33" )
438 PORT E_RX_DV     = CONN_E_RX_DV     , UCF_NET_STRING = ( "LOC = Y7" , "IOSTANDARD = LVCMOS33" )
439 PORT E_RXD0      = CONN_E_RXD0      , UCF_NET_STRING = ( "LOC = AB7" , "IOSTANDARD =
    LVCMOS33" , "PULLUP" )
440 PORT E_RXD1      = CONN_E_RXD1      , UCF_NET_STRING = ( "LOC = W6" , "IOSTANDARD = LVCMOS33" ,
    "PULLUP" )
441 PORT E_RXD2      = CONN_E_RXD2      , UCF_NET_STRING = ( "LOC = Y6" , "IOSTANDARD = LVCMOS33" ,
    "PULLUP" )
442 PORT E_RXD3      = CONN_E_RXD3      , UCF_NET_STRING = ( "LOC = AA6" , "IOSTANDARD =
    LVCMOS33" , "PULLUP" )
443 PORT E_RX_ER     = CONN_E_RX_ER     , UCF_NET_STRING = ( "LOC = AB8" , "IOSTANDARD = LVCMOS33"
    )
444 PORT E_TC_CLK    = CONN_E_TX_CLK    , UCF_NET_STRING = ( "LOC = AA8" , "IOSTANDARD = LVCMOS33"
    )
445 PORT E_TX_EN     = CONN_E_TX_EN     , UCF_NET_STRING = ( "LOC = AB6" , "IOSTANDARD = LVCMOS33"
    )
446 PORT E_TXD0      = CONN_E_TXD0      , UCF_NET_STRING = ( "LOC = AB10" , "IOSTANDARD =
    LVCMOS33" )
447 PORT E_TXD1      = CONN_E_TXD1      , UCF_NET_STRING = ( "LOC = AA10" , "IOSTANDARD =
    LVCMOS33" )
448 PORT E_TXD2      = CONN_E_TXD2      , UCF_NET_STRING = ( "LOC = Y10" , "IOSTANDARD = LVCMOS33"
    )
449 PORT E_TXD3      = CONN_E_TXD3      , UCF_NET_STRING = ( "LOC = V10" , "IOSTANDARD = LVCMOS33"
    )

```

```

450 PORT E_TX_ER      = CONN_E_TX_ER      , UCF_NET_STRING = ( "LOC = AA3" , "IOSTANDARD =
      LVCMOS33" , "PULLUP" )
451
452 ### USB ###
453 PORT USB_CLKOUT   = CONN_USB_CLKOUT   , UCF_NET_STRING = ( "LOC = E13" , "IOSTANDARD =
      LVCMOS33" , "PULLDOWN" )
454 PORT USB_DIR      = CONN_USB_DIR      , UCF_NET_STRING = ( "LOC = C13" , "IOSTANDARD =
      LVCMOS33" , "PULLDOWN" )
455 PORT USB_NXT      = CONN_USB_NXT      , UCF_NET_STRING = ( "LOC = D13" , "IOSTANDARD =
      LVCMOS33" , "PULLDOWN" )
456 PORT USB_STP      = CONN_USB_STP      , UCF_NET_STRING = ( "LOC = F13" , "IOSTANDARD =
      LVCMOS33" , "PULLDOWN" )
457 # PORT USB_RESET  = CONN_USB_RESET    , UCF_NET_STRING = ( "LOC = " , "IOSTANDARD = LVCMOS33" ,
      "PULLDOWN" , "TIG" )
458 PORT USB_D0       = CONN_USB_D0       , UCF_NET_STRING = ( "LOC = B20" , "IOSTANDARD =
      LVCMOS33" , "PULLDOWN" )
459 PORT USB_D1       = CONN_USB_D1       , UCF_NET_STRING = ( "LOC = A10" , "IOSTANDARD =
      LVCMOS33" , "PULLDOWN" )
460 PORT USB_D2       = CONN_USB_D2       , UCF_NET_STRING = ( "LOC = E15" , "IOSTANDARD =
      LVCMOS33" , "PULLDOWN" )
461 PORT USB_D3       = CONN_USB_D3       , UCF_NET_STRING = ( "LOC = F15" , "IOSTANDARD =
      LVCMOS33" , "PULLDOWN" )
462 PORT USB_D4       = CONN_USB_D4       , UCF_NET_STRING = ( "LOC = C18" , "IOSTANDARD =
      LVCMOS33" , "PULLDOWN" )
463 PORT USB_D5       = CONN_USB_D5       , UCF_NET_STRING = ( "LOC = A18" , "IOSTANDARD =
      LVCMOS33" , "PULLDOWN" )
464 PORT USB_D6       = CONN_USB_D6       , UCF_NET_STRING = ( "LOC = B19" , "IOSTANDARD =
      LVCMOS33" , "PULLDOWN" )
465 PORT USB_D7       = CONN_USB_D7       , UCF_NET_STRING = ( "LOC = A19" , "IOSTANDARD =
      LVCMOS33" , "PULLDOWN" )
466
467 ### SPI ###
468 PORT SEN_SPL_RX   = CONN_SEN_SPL_RX   , UCF_NET_STRING = ( "LOC = M18" , "IOSTANDARD = LVTTL" )
469 PORT SEN_SPL_TX   = CONN_SEN_SPL_TX   , UCF_NET_STRING = ( "LOC = L22" , "IOSTANDARD = LVTTL" )
470 PORT SEN_SPL_CLK  = CONN_SEN_SPL_CLK  , UCF_NET_STRING = ( "LOC = K17" , "IOSTANDARD = LVTTL" )
471 PORT SEN_SS_0     = CONN_SEN_SS_0     , UCF_NET_STRING = ( "LOC = T19" , "IOSTANDARD = LVTTL" )
      # FIXME
472 PORT SEN_SS_1     = CONN_SEN_SS_1     , UCF_NET_STRING = ( "LOC = M22" , "IOSTANDARD = LVTTL" )
473
474 ### DDR2 (FPGA Section)
475 PORT DDR2_ODT     = ddr2_dds_odt_0    , UCF_NET_STRING=( "LOC=P1" , "IOSTANDARD = SSTL18_I" )
476 PORT DDR2_DDR_ADDR0 = ddr2_dds_addr_0 , UCF_NET_STRING=( "LOC=R2" , "IOSTANDARD = SSTL18_I" )
477 PORT DDR2_DDR_ADDR1 = ddr2_dds_addr_1 , UCF_NET_STRING=( "LOC=T4" , "IOSTANDARD = SSTL18_I" )
478 PORT DDR2_DDR_ADDR2 = ddr2_dds_addr_2 , UCF_NET_STRING=( "LOC=R1" , "IOSTANDARD = SSTL18_I" )
479 PORT DDR2_DDR_ADDR3 = ddr2_dds_addr_3 , UCF_NET_STRING=( "LOC=U3" , "IOSTANDARD = SSTL18_I" )
480 PORT DDR2_DDR_ADDR4 = ddr2_dds_addr_4 , UCF_NET_STRING=( "LOC=U2" , "IOSTANDARD = SSTL18_I" )
481 PORT DDR2_DDR_ADDR5 = ddr2_dds_addr_5 , UCF_NET_STRING=( "LOC=U4" , "IOSTANDARD = SSTL18_I" )
482 PORT DDR2_DDR_ADDR6 = ddr2_dds_addr_6 , UCF_NET_STRING=( "LOC=U1" , "IOSTANDARD = SSTL18_I" )
483 PORT DDR2_DDR_ADDR7 = ddr2_dds_addr_7 , UCF_NET_STRING=( "LOC=Y1" , "IOSTANDARD = SSTL18_I" )
484 PORT DDR2_DDR_ADDR8 = ddr2_dds_addr_8 , UCF_NET_STRING=( "LOC=W1" , "IOSTANDARD = SSTL18_I" )
485 PORT DDR2_DDR_ADDR9 = ddr2_dds_addr_9 , UCF_NET_STRING=( "LOC=W2" , "IOSTANDARD = SSTL18_I" )
486 PORT DDR2_DDR_ADDR10 = ddr2_dds_addr_10 , UCF_NET_STRING=( "LOC=T3" , "IOSTANDARD = SSTL18_I" )
487 PORT DDR2_DDR_ADDR11 = ddr2_dds_addr_11 , UCF_NET_STRING=( "LOC=V1" , "IOSTANDARD = SSTL18_I" )
488 PORT DDR2_DDR_ADDR12 = ddr2_dds_addr_12 , UCF_NET_STRING=( "LOC=Y2" , "IOSTANDARD = SSTL18_I" )
489
490 PORT DDR2_DDR_BANKADDR0 = ddr2_dds_bankaddr_0 , UCF_NET_STRING=( "LOC=P3" , "IOSTANDARD =
      SSTL18_I" )

```

```

491 PORT DDR2_DDR_BANKADDR1 = ddr2_ddr_bankaddr_1 , UCF_NET_STRING=( "LOC=R3" , "IOSTANDARD =
      SSTL18_I" )
492
493 PORT DDR2_DDR_CASN      = ddr2_ddr_casn      , UCF_NET_STRING=( "LOC=M4" , "IOSTANDARD = SSTL18_I" )
494 PORT DDR2_DDR_CKE      = ddr2_ddr_cke      , UCF_NET_STRING=( "LOC=N3" , "IOSTANDARD = SSTL18_I" )
495 PORT DDR2_DDR_CSN      = ddr2_ddr_csn      , UCF_NET_STRING=( "LOC=M5" , "IOSTANDARD = SSTL18_I" )
496 PORT DDR2_DDR_RASN     = ddr2_ddr_rasn     , UCF_NET_STRING=( "LOC=M3" , "IOSTANDARD = SSTL18_I" )
497 PORT DDR2_DDR_WEN      = ddr2_ddr_wen     , UCF_NET_STRING=( "LOC=N4" , "IOSTANDARD = SSTL18_I" )
498 PORT DDR2_DDR_CLK      = ddr2_clk        , UCF_NET_STRING=( "LOC=M1" , "IOSTANDARD =
      DIFF_SSTL18_I" )
499 PORT DDR2_DDR_CLKn     = ddr2_clk_n      , UCF_NET_STRING=( "LOC=M2" , "IOSTANDARD =
      DIFF_SSTL18_I" )
500
501 PORT DDR2_DDR_DM0      = ddr2_ddr_dm_0    , UCF_NET_STRING=( "LOC=J3" , "IOSTANDARD = SSTL18_I" )
502 PORT DDR2_DDR_DM1      = ddr2_ddr_dm_1    , UCF_NET_STRING=( "LOC=E3" , "IOSTANDARD = SSTL18_I" )
503
504 PORT DDR2_DQS_DIV_I    = ddr2_ddr_dqs_div_i , UCF_NET_STRING=( "LOC=H4" , "IOSTANDARD =
      SSTL18_II" )
505 PORT DDR2_DQS_DIV_O    = ddr2_ddr_dqs_div_o , UCF_NET_STRING=( "LOC=H3" , "IOSTANDARD =
      SSTL18_II" )
506
507 PORT DDR2_DDR_DQS0     = ddr2_ddr_dqs_0    , UCF_NET_STRING=( "LOC=K3" , "IOSTANDARD =
      DIFF_SSTL18_II" )
508 PORT DDR2_DDR_DQS1     = ddr2_ddr_dqs_1    , UCF_NET_STRING=( "LOC=K6" , "IOSTANDARD =
      DIFF_SSTL18_II" )
509
510 PORT DDR2_DDR_DQSn0    = ddr2_ddr_dqsn_0   , UCF_NET_STRING=( "LOC=K2" , "IOSTANDARD =
      DIFF_SSTL18_II" )
511 PORT DDR2_DDR_DQSn1    = ddr2_ddr_dqsn_1   , UCF_NET_STRING=( "LOC=J5" , "IOSTANDARD =
      DIFF_SSTL18_II" )
512
513 PORT DDR2_DDR_DQ0      = ddr2_ddr_dq_0    , UCF_NET_STRING=( "LOC=H1" , "IOSTANDARD = SSTL18_II" )
514 PORT DDR2_DDR_DQ1      = ddr2_ddr_dq_1    , UCF_NET_STRING=( "LOC=K5" , "IOSTANDARD = SSTL18_II" )
515 PORT DDR2_DDR_DQ2      = ddr2_ddr_dq_2    , UCF_NET_STRING=( "LOC=K1" , "IOSTANDARD = SSTL18_II" )
516 PORT DDR2_DDR_DQ3      = ddr2_ddr_dq_3    , UCF_NET_STRING=( "LOC=L3" , "IOSTANDARD = SSTL18_II" )
517 PORT DDR2_DDR_DQ4      = ddr2_ddr_dq_4    , UCF_NET_STRING=( "LOC=L5" , "IOSTANDARD = SSTL18_II" )
518 PORT DDR2_DDR_DQ5      = ddr2_ddr_dq_5    , UCF_NET_STRING=( "LOC=L1" , "IOSTANDARD = SSTL18_II" )
519 PORT DDR2_DDR_DQ6      = ddr2_ddr_dq_6    , UCF_NET_STRING=( "LOC=K4" , "IOSTANDARD = SSTL18_II" )
520 PORT DDR2_DDR_DQ7      = ddr2_ddr_dq_7    , UCF_NET_STRING=( "LOC=H2" , "IOSTANDARD = SSTL18_II" )
521 PORT DDR2_DDR_DQ8      = ddr2_ddr_dq_8    , UCF_NET_STRING=( "LOC=F2" , "IOSTANDARD = SSTL18_II" )
522 PORT DDR2_DDR_DQ9      = ddr2_ddr_dq_9    , UCF_NET_STRING=( "LOC=G4" , "IOSTANDARD = SSTL18_II" )
523 PORT DDR2_DDR_DQ10     = ddr2_ddr_dq_10   , UCF_NET_STRING=( "LOC=G1" , "IOSTANDARD = SSTL18_II" )
524 PORT DDR2_DDR_DQ11     = ddr2_ddr_dq_11   , UCF_NET_STRING=( "LOC=H6" , "IOSTANDARD = SSTL18_II" )
525 PORT DDR2_DDR_DQ12     = ddr2_ddr_dq_12   , UCF_NET_STRING=( "LOC=H5" , "IOSTANDARD = SSTL18_II" )
526 PORT DDR2_DDR_DQ13     = ddr2_ddr_dq_13   , UCF_NET_STRING=( "LOC=F1" , "IOSTANDARD = SSTL18_II" )
527 PORT DDR2_DDR_DQ14     = ddr2_ddr_dq_14   , UCF_NET_STRING=( "LOC=G3" , "IOSTANDARD = SSTL18_II" )
528 PORT DDR2_DDR_DQ15     = ddr2_ddr_dq_15   , UCF_NET_STRING=( "LOC=F3" , "IOSTANDARD = SSTL18_II" )
529 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

```

1 -----
2 -- Test Bench for acc
3 -- Last Modified: 22 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2011 Worcester Polytechnic Institute
7 --
8 -- This program is free software; you can redistribute it and/or modify
9 -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_SIGNED.ALL;
26
27 entity TB_acc is
28 end TB_acc;
```

```

29
30 architecture behavioral of TB_acc is
31     -- Component declaration for the Unit Under Test (UUT)
32     component acc is
33         port(
34             clock          : in    std_logic;
35             reset          : in    std_logic;
36             clear          : in    std_logic;
37             enable_in     : in    std_logic;
38             enable_out    : out   std_logic;
39             addend        : in    std_logic_vector(30 downto 0);
40             sum           : out   std_logic_vector(33 downto 0)
41         );
42     end component acc;
43
44     -- Inputs
45     signal clock : std_logic := '0';
46     signal reset : std_logic := '0';
47     signal clear : std_logic := '0';
48     signal enable_in : std_logic := '0';
49     signal addend : std_logic_vector(30 downto 0) := (others => '0');
50
51     -- Outputs
52     signal enable_out : std_logic;
53     signal sum : std_logic_vector(33 downto 0);
54
55     -- Clock Period
56     constant clock_period : time := 10ns;
57 begin
58     -- Instantiate the UUT.
59     UUT : acc
60         port map(clock => clock, reset => reset, clear => clear, enable_in => enable_in,
61                 enable_out => enable_out, addend => addend, sum => sum);
62
63     -- Clock Process
64     clock_process : process
65     begin
66         clock <= '0';
67         wait for clock_period / 2;
68         clock <= '1';
69         wait for clock_period / 2;
70     end process clock_process;
71
72     -- Stimulus Process
73     stim_proc : process
74     begin
75         -- Do nothing for one clock period.
76         wait for clock_period;
77
78         -- Assert the reset.
79         reset <= '1';
80         wait for clock_period;
81
82         -- Clear the accumulator.
83         reset <= '0';
84         clear <= '1';
85         addend <= (others => '0');

```

```

86     wait for clock_period;
87
88     -- Add some values.
89     clear <= '0';
90     enable_in <= '1';
91     for i in 1 to 15 loop
92         addend <= conv_std_logic_vector(i, addend'LENGTH);
93         wait for clock_period;
94     end loop;
95
96     -- Clear the accumulator.
97     enable_in <= '0';
98     clear <= '1';
99     addend <= (others => '0');
100    wait for clock_period;
101
102    -- Add some negative values.
103    clear <= '0';
104    enable_in <= '1';
105    for i in -1 downto -15 loop
106        addend <= conv_std_logic_vector(i, addend'LENGTH);
107        wait for clock_period;
108    end loop;
109
110    -- Clear the accumulator.
111    enable_in <= '0';
112    clear <= '1';
113    addend <= (others => '0');
114    wait for clock_period;
115
116    -- Test overflow.
117    clear <= '0';
118    enable_in <= '1';
119    for i in 0 to 8 loop
120        addend <= "01111111111111111111111111111111";
121        wait for clock_period;
122    end loop;
123
124    -- End test.
125    wait;
126 end process stim_proc;
127 end behavioral;

```

Listing C.2: TB_adc_mux.vhd

```

1 -----
2 -- Test Bench for adc_mux
3 -- Last Modified: 14 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2011 Worcester Polytechnic Institute
7 --
8 -- This program is free software; you can redistribute it and/or modify
9 -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.

```



```

12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_UNSIGNED.ALL;
26
27 entity TB_adc_mux is
28 end TB_adc_mux;
29
30 architecture behavioral of TB_adc_mux is
31     -- Component declaration for the Unit Under Test (UUT)
32     component adc_mux is
33     port(
34         clock           : in   std_logic;
35         rx_realsignals  : in   std_logic;
36         ddc_mux         : in   std_logic_vector(3 downto 0);
37         adc3_corr      : in   std_logic_vector(15 downto 0);
38         adc2_corr      : in   std_logic_vector(15 downto 0);
39         adc1_corr      : in   std_logic_vector(15 downto 0);
40         adc0_corr      : in   std_logic_vector(15 downto 0);
41         ddc_i           : out  std_logic_vector(15 downto 0);
42         ddc_q           : out  std_logic_vector(15 downto 0)
43     );
44     end component adc_mux;
45
46     -- Inputs
47     signal clock : std_logic := '0';
48     signal rx_realsignals : std_logic := '0';
49     signal ddc_mux : std_logic_vector(3 downto 0) := (others => '0');
50     signal adc3_corr : std_logic_vector(15 downto 0) := (others => '0');
51     signal adc2_corr : std_logic_vector(15 downto 0) := (others => '0');
52     signal adc1_corr : std_logic_vector(15 downto 0) := (others => '0');
53     signal adc0_corr : std_logic_vector(15 downto 0) := (others => '0');
54
55     -- Outputs
56     signal ddc_i : std_logic_vector(15 downto 0);
57     signal ddc_q : std_logic_vector(15 downto 0);
58
59     -- Clock Period
60     constant clock_period : time := 10ns;
61 begin
62     -- Instantiate the UUT.
63     UUT : adc_mux
64         port map(clock => clock, rx_realsignals => rx_realsignals, ddc_mux => ddc_mux,
65                 adc3_corr => adc3_corr, adc2_corr => adc2_corr, adc1_corr => adc1_corr,
66                 adc0_corr => adc0_corr, ddc_i => ddc_i, ddc_q => ddc_q);
67
68     -- Clock Process

```

```

69     clock_process : process
70     begin
71         clock <= '0';
72         wait for clock_period / 2;
73         clock <= '1';
74         wait for clock_period / 2;
75     end process clock_process;
76
77     -- Stimulus Process
78     stim_proc : process
79     begin
80         -- Assign some values to the registers.
81         adc0_corr <= "0001000100010001";
82         adc1_corr <= "0010001000100010";
83         adc2_corr <= "0011001100110011";
84         adc3_corr <= "0100010001000100";
85
86         -- Try every possible input for ddc_mux with rx_realsignals low.
87         rx_realsignals <= '0';
88         for i in 0 to 15 loop
89             ddc_mux <= conv_std_logic_vector(i, ddc_mux'LENGTH);
90             wait for clock_period;
91         end loop;
92
93         -- Try every possible input for ddc_mux with rx_realsignals high.
94         rx_realsignals <= '1';
95         for i in 0 to 15 loop
96             ddc_mux <= conv_std_logic_vector(i, ddc_mux'LENGTH);
97             wait for clock_period;
98         end loop;
99
100        -- End test.
101        wait;
102    end process stim_proc;
103 end behavioral;

```

Listing C.3: TB_atr_delay.vhd

```

1 -----
2 -- Test Bench for atr_delay
3 -- Last Modified: 14 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 Worcester Polytechnic Institute
7 --
8 -- This program is free software; you can redistribute it and/or modify
9 -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License

```

```

19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 entity TB_atr_delay is
26 end TB_atr_delay;
27
28 architecture behavioral of TB_atr_delay is
29     -- Component declaration for the Unit Under Test (UUT)
30     component atr_delay is
31         port(
32             clk_i          : in    std_logic;
33             rst_i          : in    std_logic;
34             ena_i          : in    std_logic;
35             tx_empty_i    : in    std_logic;
36             tx_delay_i    : in    std_logic_vector(11 downto 0);
37             rx_delay_i    : in    std_logic_vector(11 downto 0);
38             atr_tx_o      : out   std_logic
39         );
40     end component atr_delay;
41
42     -- Inputs
43     signal clk_i : std_logic := '0';
44     signal rst_i : std_logic := '0';
45     signal ena_i : std_logic := '0';
46     signal tx_empty_i : std_logic := '0';
47     signal tx_delay_i : std_logic_vector(11 downto 0) := (others => '0');
48     signal rx_delay_i : std_logic_vector(11 downto 0) := (others => '0');
49
50     -- Outputs
51     signal atr_tx_o : std_logic;
52
53     -- Clock Period
54     constant clk_i_period : time := 10ns;
55 begin
56     -- Instantiate the UUT.
57     UUT : atr_delay
58         port map(clk_i => clk_i, rst_i => rst_i, ena_i => ena_i, tx_empty_i => tx_empty_i,
59             tx_delay_i => tx_delay_i, rx_delay_i => rx_delay_i, atr_tx_o => atr_tx_o);
60
61     -- Clock Process
62     clk_i_process : process
63     begin
64         clk_i <= '0';
65         wait for clk_i_period / 2;
66         clk_i <= '1';
67         wait for clk_i_period / 2;
68     end process clk_i_process;
69
70     -- Stimulus Process
71     stim_proc : process
72     begin
73         -- Do nothing for one clock period.
74         wait for clk_i_period;
75

```

```

76      -- Assert the reset.
77      rst_i <= '1';
78      wait for clk_i_period;
79
80      -- Deassert the reset, assert the enable.
81      rst_i <= '0';
82      ena_i <= '1';
83
84      -- Delay values of 2 and 3.
85      tx_delay_i <= "00000000010";
86      rx_delay_i <= "00000000011";
87
88      -- Because tx_empty_i is low, we should switch to TX_DELAY and then TX.
89      -- We'll stay in TX until we assert tx_empty_i.
90      wait for 10 * clk_i_period;
91
92      -- Now, assert tx_empty_i. The atr_tx_o signal should remain high for
93      -- rx_delay while we're in the RX_DELAY state and then it should go low
94      -- when we reach RX.
95      tx_empty_i <= '1';
96      wait for 10 * clk_i_period;
97
98      -- Let's get back to the TX state so we can test the reset again. We'll
99      -- change the delay values to 4 and 7, as well.
100     tx_empty_i <= '0';
101     tx_delay_i <= "00000000100";
102     rx_delay_i <= "00000000111";
103     wait for 10 * clk_i_period;
104
105     -- Assert the reset. This should put us back to RX.
106     rst_i <= '1';
107     wait for 10 * clk_i_period;
108
109     -- Delay until TX again.
110     rst_i <= '0';
111     wait for 10 * clk_i_period;
112
113     -- End test.
114     wait;
115 end process stim_proc;
116 end behavioral;

```

Listing C.4: TB_bidir_reg.vhd

```

1 -----
2 -- Test Bench for bidir_reg
3 -- Last Modified: 14 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 Worcester Polytechnic Institute
7 --
8 -- This program is free software; you can redistribute it and/or modify
9 -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --

```

```

13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 entity TB_bidir_reg is
26 end TB_bidir_reg;
27
28 architecture behavioral of TB_bidir_reg is
29     -- Component declaration for the Unit Under Test (UUT)
30     component bidir_reg is
31         port(
32             tristate    : inout std_logic_vector(15 downto 0);
33             oe           : in    std_logic_vector(15 downto 0);
34             reg_val      : in    std_logic_vector(15 downto 0)
35         );
36     end component bidir_reg;
37
38     -- Inputs
39     signal oe : std_logic_vector(15 downto 0) := (others => '0');
40     signal reg_val : std_logic_vector(15 downto 0) := (others => '0');
41
42     -- Bidirectionals
43     signal tristate : std_logic_vector(15 downto 0);
44 begin
45     -- Instantiate the UUT.
46     UUT : bidir_reg
47         port map(tristate => tristate, oe => oe, reg_val => reg_val);
48
49     -- Stimulus Process
50     stim_proc : process
51     begin
52         -- Do nothing for 10ns.
53         wait for 10ns;
54
55         -- Keep the enables off, but change the input values.
56         oe <= (others => '0');
57         reg_val <= "1100001111000011";
58         wait for 10ns;
59
60         -- Turn on all enables.
61         oe <= (others => '1');
62         wait for 10ns;
63
64         -- Change some of the inputs while leaving all enables on.
65         reg_val <= "0000000011111111";
66         wait for 10ns;
67
68         -- Turn off some enables, leave others on.
69         oe <= "1010101001010101";

```

```

70     wait for 10ns;
71
72     -- Change some of the inputs again.
73     reg_val <= "1111000011110000";
74     wait for 10ns;
75
76     -- Disable all outputs.
77     oe <= (others => '0');
78     wait for 10ns;
79
80     -- End test.
81     wait;
82 end process stim_proc;
83 end behavioral;

```

Listing C.5: TB_bustri.vhd

```

1  -----
2  -- Test Bench for bustri
3  -- Last Modified: 13 February 2011
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2010 Worcester Polytechnic Institute
7  --
8  -- This program is free software; you can redistribute it and/or modify
9  -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 entity TB_bustri is
26 end TB_bustri;
27
28 architecture behavioral of TB_bustri is
29     -- Component declaration for the Unit Under Test (UUT)
30     component bustri
31     port(
32         data      : in   std_logic_vector(15 downto 0);
33         enabledt  : in   std_logic;
34         tridata  : out  std_logic_vector(15 downto 0)
35     );
36 end component bustri;
37
38 -- Inputs
39 signal data : std_logic_vector(15 downto 0) := (others => '0');

```

```

40     signal enabledt : std_logic := '0';
41
42     -- Outputs
43     signal tridata : std_logic_vector(15 downto 0);
44 begin
45     -- Instantiate the UUT.
46     UUT : bustri
47         port map(data => data, enabledt => enabledt, tridata => tridata);
48
49     -- Stimulus Process
50     stim_proc : process
51     begin
52         -- Do nothing for 10ns.
53         wait for 10ns;
54
55         -- Set data, turn off output.
56         data <= "1010010111110000";
57         enabledt <= '0';
58         wait for 10ns;
59
60         -- Turn on output.
61         enabledt <= '1';
62         wait for 10ns;
63
64         -- Change data with output still enabled.
65         data <= "0000111101011010";
66         wait for 10ns;
67
68         -- Disable output.
69         enabledt <= '0';
70         wait for 10ns;
71
72         -- Change data with output disabled.
73         data <= "0101010110101010";
74         wait for 10ns;
75
76         -- Enable output.
77         enabledt <= '1';
78         wait for 10ns;
79
80         -- End test.
81         wait;
82     end process stim_proc;
83 end behavioral;

```

Listing C.6: TB_cic_decim.vhd

```

1
2 -- Test Bench for cic_decim
3 -- Last Modified: 22 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2011 Worcester Polytechnic Institute
7 --
8 -- This program is free software; you can redistribute it and/or modify
9 -- it under the terms of the GNU General Public License as published by

```

```

10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_UNSIGNED.ALL;
26
27 entity TB_cic_decim is
28 end TB_cic_decim;
29
30 architecture behavioral of TB_cic_decim is
31 -- Component declaration for the Unit Under Test (UUT)
32 component cic_decim is
33     generic(
34         bw                : integer := 16;    -- # of bits for input
35         N                  : integer := 4;    -- # of filter stages
36         log2_of_max_rate  : integer := 7;    -- log2 of max sampling rate
37         maxbitgain        : integer := 28 -- N*log2_of_max_rate
38     );
39
40     port(
41         clock              : in    std_logic;
42         reset              : in    std_logic;
43         enable             : in    std_logic;
44         rate               : in    std_logic_vector(7 downto 0);
45         strobe_in         : in    std_logic;
46         strobe_out        : in    std_logic;
47         signal_in         : in    std_logic_vector(bw-1 downto 0);
48         signal_out        : out   std_logic_vector(bw-1 downto 0)
49     );
50 end component cic_decim;
51
52 -- Inputs
53 signal clock : std_logic := '0';
54 signal reset : std_logic := '0';
55 signal enable : std_logic := '0';
56 signal rate : std_logic_vector(7 downto 0) := (others => '0');
57 signal strobe_in : std_logic := '0';
58 signal strobe_out : std_logic := '0';
59 signal signal_in : std_logic_vector(15 downto 0) := (others => '0');
60
61 -- Outputs
62 signal signal_out : std_logic_vector(15 downto 0);
63
64 -- Clock Period
65 constant clock_period : time := 10ns;
66 begin

```



```

67  -- Instantiate the UUT.
68  UUT : cic_decim
69      port map(clock => clock, reset => reset, enable => enable, rate => rate,
70              strobe_in => strobe_in, strobe_out => strobe_out, signal_in => signal_in,
71              signal_out => signal_out);
72
73  -- Clock Process
74  clock_process : process
75  begin
76      clock <= '0';
77      wait for clock_period / 2;
78      clock <= '1';
79      wait for clock_period / 2;
80  end process clock_process;
81
82  -- Strobe Process
83  strobe_proc : process
84  begin
85      wait for 7 * clock_period;
86      strobe_out <= '1';
87      wait for clock_period;
88      strobe_out <= '0';
89  end process strobe_proc;
90
91  -- We have a sample on every clock, so just keep strobe_out enabled.
92  strobe_in <= '1';
93
94  -- Stimulus Process
95  stim_proc : process
96  begin
97      -- Do nothing for one clock period.
98      wait for clock_period;
99
100     -- Assert the reset.
101     reset <= '1';
102     wait for clock_period;
103
104     -- Enable the filter.
105     reset <= '0';
106     enable <= '1';
107
108     -- Set the rate to 7. This is actually a rate of 32 because of the
109     -- codecs that we use. The formula is (rate+1)*4.
110     rate <= conv_std_logic_vector(7, rate'LENGTH);
111
112     -- Test values of 1, max, min, and -1.
113     signal_in <= "0000000000000001";
114     wait for 100 * clock_period;
115     signal_in <= "0111111111111111";
116     wait for 100 * clock_period;
117     signal_in <= "1000000000000000";
118     wait for 100 * clock_period;
119     signal_in <= "1111111111111111";
120     wait for 100 * clock_period;
121
122     -- End test.
123     wait;

```

```

124     end process stim_proc;
125 end behavioral;

```

Listing C.7: TB_cic_dec_shifter.vhd

```

1  -----
2  -- Test Bench for cic_dec_shifter
3  -- Last Modified: 22 February 2011
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2011 Worcester Polytechnic Institute
7  --
8  -- This program is free software; you can redistribute it and/or modify
9  -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_UNSIGNED.ALL;
26
27 entity TB_cic_dec_shifter is
28 end TB_cic_dec_shifter;
29
30 architecture behavioral of TB_cic_dec_shifter is
31     -- Component declaration for the Unit Under Test (UUT)
32     component cic_dec_shifter is
33         generic(
34             bw          : integer := 16;
35             maxbitgain  : integer := 28
36         );
37
38         port(
39             rate        : in    std_logic_vector(7 downto 0);
40             signal_in   : in    std_logic_vector(bw+maxbitgain-1 downto 0);
41             signal_out  : out   std_logic_vector(bw-1 downto 0)
42         );
43     end component cic_dec_shifter;
44
45     -- Inputs
46     signal rate : std_logic_vector(7 downto 0) := (others => '0');
47     signal signal_in : std_logic_vector(43 downto 0) := (others => '0');
48
49     -- Outputs
50     signal signal_out : std_logic_vector(15 downto 0);
51 begin

```

```

52     -- Instantiate the UUT.
53     UUT : cic_dec_shifter
54         generic map(bw => 16, maxbitgain => 28)
55         port map(rate => rate, signal_in => signal_in, signal_out => signal_out);
56
57     -- Stimulus Process
58     stim_proc : process
59     begin
60         -- Do nothing for 10ns.
61         wait for 10ns;
62
63         -- Set signal_in to a constant value (0x0123456789AB).
64         signal_in <= "00010010001101000101011001111000100110101011";
65
66         for i in 4 to 127 loop
67             rate <= conv_std_logic_vector(i, rate'LENGTH);
68             wait for 10ns;
69         end loop;
70
71         -- End test.
72         wait;
73     end process stim_proc;
74 end behavioral;

```

Listing C.8: TB_cic_interp.vhd

```

1 -----
2 -- Test Bench for cic_interp
3 -- Last Modified: 21 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2011 Worcester Polytechnic Institute
7 --
8 -- This program is free software; you can redistribute it and/or modify
9 -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_UNSIGNED.ALL;
26
27 entity TB_cic_interp is
28 end TB_cic_interp;
29
30 -- This test bench is inspired by the justinterp_tb.v test included with the

```

```

31 -- original USRP source code.
32 architecture behavioral of TB_cic_interp is
33     -- Component declaration for the Unit Under Test (UUT)
34     component cic_interp is
35         generic(
36             bw                : integer := 16;    -- # of bits for input
37             N                  : integer := 4;    -- # of filter stages
38             log2_of_max_rate   : integer := 7;    -- log2 of max sampling rate
39             maxbitgain         : integer := 21 --(N-1)*log2_of_max_rate
40         );
41
42     port(
43         clock                : in    std_logic;
44         reset                 : in    std_logic;
45         enable                : in    std_logic;
46         rate                  : in    std_logic_vector(7 downto 0);
47         strobe_in             : in    std_logic;
48         strobe_out            : in    std_logic;
49         signal_in             : in    std_logic_vector(bw-1 downto 0);
50         signal_out            : out   std_logic_vector(bw-1 downto 0)
51     );
52 end component cic_interp;
53
54 -- Inputs
55 signal clock : std_logic := '0';
56 signal reset : std_logic := '0';
57 signal enable : std_logic := '0';
58 signal rate : std_logic_vector(7 downto 0) := (others => '0');
59 signal strobe_in : std_logic := '0';
60 signal strobe_out : std_logic := '0';
61 signal signal_in : std_logic_vector(15 downto 0) := (others => '0');
62
63 -- Outputs
64 signal signal_out : std_logic_vector(15 downto 0);
65
66 -- Clock Period
67 constant clock_period : time := 10ns;
68 begin
69     -- Instantiate the UUT.
70     UUT : cic_interp
71         port map(clock => clock, reset => reset, enable => enable, rate => rate,
72                 strobe_in => strobe_in, strobe_out => strobe_out, signal_in => signal_in,
73                 signal_out => signal_out);
74
75     -- Clock Process
76     clock_process : process
77     begin
78         clock <= '0';
79         wait for clock_period / 2;
80         clock <= '1';
81         wait for clock_period / 2;
82     end process clock_process;
83
84     -- Strobe Process
85     strobe_proc : process
86     begin
87         wait for 7 * clock_period;

```

```

88     strobe_in <= '1';
89     wait for clock_period;
90     strobe_in <= '0';
91 end process strobe_proc;
92
93 -- We want a sample on every clock, so just keep strobe_out enabled.
94 strobe_out <= '1';
95
96 -- Stimulus Process
97 stim_proc : process
98 begin
99     -- Do nothing for one clock period.
100    wait for clock_period;
101
102    -- Assert the reset.
103    reset <= '1';
104    wait for clock_period;
105
106    -- Enable the filter.
107    reset <= '0';
108    enable <= '1';
109
110    -- Set the rate to 7. This is actually a rate of 32 because of the
111    -- codecs that we use. The formula is (rate+1)*4.
112    rate <= conv_std_logic_vector(7, rate'LENGTH);
113
114    -- Test values of 1, max, min, and -1.
115    signal_in <= "0000000000000001";
116    wait for 100 * clock_period;
117    signal_in <= "0111111111111111";
118    wait for 100 * clock_period;
119    signal_in <= "1000000000000000";
120    wait for 100 * clock_period;
121    signal_in <= "1111111111111111";
122    wait for 100 * clock_period;
123
124    -- End test.
125    wait;
126 end process stim_proc;
127 end behavioral;

```

Listing C.9: TB_cic_int_shifter.vhd

```

1 -----
2 -- Test Bench for cic_int_shifter
3 -- Last Modified: 20 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2011 Worcester Polytechnic Institute
7 --
8 -- This program is free software; you can redistribute it and/or modify
9 -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,

```

```

14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_UNSIGNED.ALL;
26
27 entity TB_cic_int_shifter is
28 end TB_cic_int_shifter;
29
30 architecture behavioral of TB_cic_int_shifter is
31     -- Component declaration for the Unit Under Test (UUT)
32     component cic_int_shifter is
33         generic(
34             bw          : integer := 16;
35             maxbitgain  : integer := 21
36         );
37
38         port(
39             rate        : in    std_logic_vector(7 downto 0);
40             signal_in   : in    std_logic_vector(bw+maxbitgain-1 downto 0);
41             signal_out  : out   std_logic_vector(bw-1 downto 0)
42         );
43     end component cic_int_shifter;
44
45     -- Inputs
46     signal rate : std_logic_vector(7 downto 0) := (others => '0');
47     signal signal_in : std_logic_vector(36 downto 0) := (others => '0');
48
49     -- Outputs
50     signal signal_out : std_logic_vector(15 downto 0);
51 begin
52     -- Instantiate the UUT.
53     UUT : cic_int_shifter
54         generic map(bw => 16, maxbitgain => 21)
55         port map(rate => rate, signal_in => signal_in, signal_out => signal_out);
56
57     -- Stimulus Process
58     stim_proc : process
59     begin
60         -- Do nothing for 10ns.
61         wait for 10ns;
62
63         -- Set signal_in to a constant value (0x0123456789).
64         signal_in <= "0000100100011010001010110011110001001";
65
66         -- Test each sample rate. The minimum possible value is 4.
67         for i in 4 to 127 loop
68             rate <= conv_std_logic_vector(i, rate'LENGTH);
69             wait for 10ns;
70         end loop;

```

```

71
72     -- End test.
73     wait;
74     end process stim_proc;
75 end behavioral;

```

Listing C.10: TB_clk_divider.vhd

```

1  -----
2  -- Test Bench for clk_divider
3  -- Last Modified: 14 February 2011
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2010 Worcester Polytechnic Institute
7  --
8  -- This program is free software; you can redistribute it and/or modify
9  -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 entity TB_clk_divider is
26 end TB_clk_divider;
27
28 architecture behavioral of TB_clk_divider is
29     -- Component declaration for the Unit Under Test (UUT)
30     component clk_divider is
31         port(
32             reset      : in   std_logic;
33             in_clk     : in   std_logic;
34             out_clk    : out  std_logic;
35             ratio      : in   std_logic_vector(7 downto 0)
36         );
37     end component clk_divider;
38
39     -- Inputs
40     signal reset : std_logic := '0';
41     signal in_clk : std_logic := '0';
42     signal ratio : std_logic_vector(7 downto 0) := (others => '0');
43
44     -- Outputs
45     signal out_clk : std_logic;
46
47     -- Clock Period
48     constant in_clk_period : time := 10ns;

```

```

49 begin
50   -- Instantiate the UUT.
51   UUT : clk_divider
52     port map(reset => reset , in_clk => in_clk , out_clk => out_clk , ratio => ratio);
53
54   -- Clock Process
55   in_clk_process : process
56   begin
57     in_clk <= '0';
58     wait for in_clk_period / 2;
59     in_clk <= '1';
60     wait for in_clk_period / 2;
61   end process in_clk_process;
62
63   -- Stimulus Process
64   stim_proc : process
65   begin
66     -- Do nothing for one clock period.
67     wait for in_clk_period;
68
69     -- Assert the reset.
70     reset <= '1';
71     wait for in_clk_period;
72
73     -- Deassert the reset. Set the ratio to 2.
74     reset <= '0';
75     ratio <= "00000010";
76     wait for 8 * in_clk_period;
77
78     -- Assert the reset.
79     reset <= '1';
80     wait for 4 * in_clk_period;
81
82     -- New ratio of 3.
83     reset <= '0';
84     ratio <= "00000011";
85     wait for 12 * in_clk_period;
86
87     -- End test.
88     wait;
89   end process stim_proc;
90 end behavioral;

```

Listing C.11: TB_coeff_rom.vhd

```

1 -----
2 -- Test Bench for coeff_rom
3 -- Last Modified: 14 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 Worcester Polytechnic Institute
7 --
8 -- This program is free software; you can redistribute it and/or modify
9 -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.

```



```

12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_UNSIGNED.ALL;
26
27 entity TB_coeff_rom is
28 end TB_coeff_rom;
29
30 architecture behavioral of TB_coeff_rom is
31     -- Component declaration for the Unit Under Test (UUT)
32     component coeff_rom is
33     port(
34         clock    : in    std_logic;
35         addr     : in    std_logic_vector(2 downto 0);
36         data     : out   std_logic_vector(15 downto 0)
37     );
38     end component coeff_rom;
39
40     -- Inputs
41     signal clock : std_logic := '0';
42     signal addr  : std_logic_vector(2 downto 0) := (others => '0');
43
44     -- Outputs
45     signal data : std_logic_vector(15 downto 0);
46
47     -- Clock Period
48     constant clock_period : time := 10ns;
49 begin
50     -- Instantiate the UUT.
51     UUT : coeff_rom
52         port map(clock => clock, addr => addr, data => data);
53
54     -- Clock Process
55     clock_process : process
56     begin
57         clock <= '0';
58         wait for clock_period / 2;
59         clock <= '1';
60         wait for clock_period / 2;
61     end process clock_process;
62
63     -- Stimulus Process
64     stim_proc : process
65     begin
66         -- Do nothing for one clock period.
67         wait for clock_period;
68

```

```

69     -- Test each possible address.
70     for i in 0 to 7 loop
71         addr <= conv_std_logic_vector(i, addr'LENGTH);
72         wait for clock_period;
73     end loop;
74
75     -- End test.
76     wait;
77 end process stim_proc;
78 end behavioral;

```

Listing C.12: TB_cordic.vhd

```

1  -----
2  -- Test Bench for cordic
3  -- Last Modified: 23 February 2011
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2011 Worcester Polytechnic Institute
7  --
8  -- This program is free software; you can redistribute it and/or modify
9  -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_UNSIGNED.ALL;
26
27 entity TB_cordic is
28 end TB_cordic;
29
30 architecture behavioral of TB_cordic is
31     -- Component declaration for the Unit Under Test (UUT)
32     component cordic is
33         generic(
34             bitwidth    : integer := 16;
35             zwidth      : integer := 16
36         );
37
38     port(
39         clock          : in    std_logic;
40         reset          : in    std_logic;
41         enable         : in    std_logic;
42         xi              : in    std_logic_vector(bitwidth-1 downto 0);
43         yi              : in    std_logic_vector(bitwidth-1 downto 0);

```

```

44         xo         : out   std_logic_vector(bitwidth-1 downto 0);
45         yo         : out   std_logic_vector(bitwidth-1 downto 0);
46         zi         : in    std_logic_vector(zwidth-1 downto 0);
47         zo         : out   std_logic_vector(zwidth-1 downto 0)
48     );
49     end component cordic;
50
51     -- Inputs
52     signal clock : std_logic := '0';
53     signal reset : std_logic := '0';
54     signal enable : std_logic := '0';
55     signal xi : std_logic_vector(15 downto 0) := (others => '0');
56     signal yi : std_logic_vector(15 downto 0) := (others => '0');
57     signal zi : std_logic_vector(15 downto 0) := (others => '0');
58
59     -- Outputs
60     signal xo : std_logic_vector(15 downto 0);
61     signal yo : std_logic_vector(15 downto 0);
62     signal zo : std_logic_vector(15 downto 0);
63
64     -- Clock Period
65     constant clock_period : time := 10ns;
66 begin
67     -- Instantiate the UUT.
68     UUT : cordic
69         generic map(bitwidth => 16, zwidth => 16)
70         port map(clock => clock, reset => reset, enable => enable, xi => xi,
71             yi => yi, xo => xo, yo => yo, zi => zi, zo => zo);
72
73     -- Clock Process
74     clock_process : process
75     begin
76         clock <= '0';
77         wait for clock_period / 2;
78         clock <= '1';
79         wait for clock_period / 2;
80     end process clock_process;
81
82     -- Stimulus Process
83     stim_proc : process
84     begin
85         -- Do nothing for one clock period.
86         wait for clock_period;
87
88         -- Assert the reset.
89         reset <= '1';
90         wait for clock_period;
91
92         -- Enable the unit.
93         reset <= '0';
94         enable <= '1';
95         wait for clock_period;
96
97         --
98         for i in 0 to 255 loop
99             xi <= conv_std_logic_vector(i mod 2, xi'LENGTH);
100            yi <= conv_std_logic_vector(2*(i mod 2), yi'LENGTH);

```

```

101         zi(13 downto 0) <= "01111111111111";
102         zi(15 downto 14) <= conv_std_logic_vector(i mod 4, 2);
103         wait for clock_period;
104     end loop;
105
106     -- End test.
107     wait;
108 end process stim_proc;
109 end behavioral;

```

Listing C.13: TB_cordic_stage.vhd

```

1 -----
2 -- Test Bench for cordic_stage
3 -- Last Modified: 23 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2011 Worcester Polytechnic Institute
7 --
8 -- This program is free software; you can redistribute it and/or modify
9 -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_UNSIGNED.ALL;
26
27 entity TB_cordic_stage is
28 end TB_cordic_stage;
29
30 architecture behavioral of TB_cordic_stage is
31     -- Component declaration for the Unit Under Test (UUT)
32     component cordic_stage is
33         generic(
34             bitwidth    : integer := 16;
35             zwidth      : integer := 16;
36             shift        : integer := 1
37         );
38
39         port(
40             clock        : in    std_logic;
41             reset        : in    std_logic;
42             enable       : in    std_logic;
43             xi           : in    std_logic_vector(bitwidth-1 downto 0);
44             yi           : in    std_logic_vector(bitwidth-1 downto 0);

```

```

45         zi      : in   std_logic_vector(zwidth-1 downto 0);
46         const   : in   std_logic_vector(zwidth-1 downto 0);
47         xo      : out  std_logic_vector(bitwidth-1 downto 0);
48         yo      : out  std_logic_vector(bitwidth-1 downto 0);
49         zo      : out  std_logic_vector(zwidth-1 downto 0)
50     );
51     end component cordic_stage;
52
53     -- Inputs
54     signal clock : std_logic := '0';
55     signal reset : std_logic := '0';
56     signal enable : std_logic := '0';
57     signal xi : std_logic_vector(15 downto 0) := (others => '0');
58     signal yi : std_logic_vector(15 downto 0) := (others => '0');
59     signal zi : std_logic_vector(15 downto 0) := (others => '0');
60     signal const : std_logic_vector(15 downto 0) := (others => '0');
61
62     -- Outputs
63     signal xo : std_logic_vector(15 downto 0);
64     signal yo : std_logic_vector(15 downto 0);
65     signal zo : std_logic_vector(15 downto 0);
66
67     -- Clock Period
68     constant clock_period : time := 10ns;
69     begin
70         -- Instantiate the UUT.
71         UUT : cordic_stage
72             generic map(bitwidth => 16, zwidth => 16, shift => 3)
73             port map(clock => clock, reset => reset, enable => enable, xi => xi,
74                    yi => yi, zi => zi, const => const, xo => xo, yo => yo, zo => zo);
75
76     -- Clock Process
77     clock_process : process
78     begin
79         clock <= '0';
80         wait for clock_period / 2;
81         clock <= '1';
82         wait for clock_period / 2;
83     end process clock_process;
84
85     -- Stimulus Process
86     stim_proc : process
87     begin
88         -- Do nothing for one clock period.
89         wait for clock_period;
90
91         -- Assert the reset.
92         reset <= '1';
93         wait for clock_period;
94
95         -- Enable the unit.
96         reset <= '0';
97         enable <= '1';
98
99         -- Use one of the constants from cordic.vhd to test.
100        const <= conv_std_logic_vector(1297, const'LENGTH);
101

```

```

102      -- Test +x, +y, +z.
103      xi <= conv_std_logic_vector(300, xi'LENGTH);
104      yi <= conv_std_logic_vector(250, yi'LENGTH);
105      zi <= conv_std_logic_vector(2000, zi'LENGTH);
106      wait for clock_period;
107
108      -- Test +x, +y, -z.
109      xi <= conv_std_logic_vector(300, xi'LENGTH);
110      yi <= conv_std_logic_vector(250, yi'LENGTH);
111      zi <= conv_std_logic_vector(-2000, zi'LENGTH);
112      wait for clock_period;
113
114      -- Test +x, -y, +z.
115      xi <= conv_std_logic_vector(300, xi'LENGTH);
116      yi <= conv_std_logic_vector(-250, yi'LENGTH);
117      zi <= conv_std_logic_vector(2000, zi'LENGTH);
118      wait for clock_period;
119
120      -- Test +x, -y, -z.
121      xi <= conv_std_logic_vector(300, xi'LENGTH);
122      yi <= conv_std_logic_vector(-250, yi'LENGTH);
123      zi <= conv_std_logic_vector(-2000, zi'LENGTH);
124      wait for clock_period;
125
126      -- Test -x, +y, +z.
127      xi <= conv_std_logic_vector(-300, xi'LENGTH);
128      yi <= conv_std_logic_vector(250, yi'LENGTH);
129      zi <= conv_std_logic_vector(2000, zi'LENGTH);
130      wait for clock_period;
131
132      -- Test -x, +y, -z.
133      xi <= conv_std_logic_vector(-300, xi'LENGTH);
134      yi <= conv_std_logic_vector(250, yi'LENGTH);
135      zi <= conv_std_logic_vector(-2000, zi'LENGTH);
136      wait for clock_period;
137
138      -- Test -x, -y, +z.
139      xi <= conv_std_logic_vector(-300, xi'LENGTH);
140      yi <= conv_std_logic_vector(-250, yi'LENGTH);
141      zi <= conv_std_logic_vector(2000, zi'LENGTH);
142      wait for clock_period;
143
144      -- Test -x, -y, -z.
145      xi <= conv_std_logic_vector(-300, xi'LENGTH);
146      yi <= conv_std_logic_vector(-250, yi'LENGTH);
147      zi <= conv_std_logic_vector(-2000, zi'LENGTH);
148      wait for clock_period;
149
150      -- End test.
151      wait;
152  end process stim_proc;
153 end behavioral;

```

Listing C.14: TB_fifo.vhd

```

2  -- Test Bench for fifo
3  -- Last Modified: 19 February 2011
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2010 Worcester Polytechnic Institute
7  --
8  -- This program is free software; you can redistribute it and/or modify
9  -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_UNSIGNED.ALL;
26
27 entity TB_fifo is
28 end TB_fifo;
29
30 architecture behavioral of TB_fifo is
31     -- Component declaration for the Unit Under Test (UUT)
32     component fifo is
33         generic(
34             width          : integer := 16;
35             depth          : integer := 1024;
36             addr_bits     : integer := 10
37         );
38
39         port(
40             data          : in   std_logic_vector(width-1 downto 0);
41             wrreq         : in   std_logic;
42             rdreq         : in   std_logic;
43             rdclk         : in   std_logic;
44             wrclk         : in   std_logic;
45             aclr          : in   std_logic;
46             q              : out  std_logic_vector(width-1 downto 0);
47             rdfull        : out  std_logic;
48             rdempty       : out  std_logic;
49             rdusedw       : out  std_logic_vector(addr_bits-1 downto 0);
50             wrfull        : out  std_logic;
51             wrempty       : out  std_logic;
52             wrusedw       : out  std_logic_vector(addr_bits-1 downto 0)
53         );
54     end component fifo;
55
56     -- Inputs
57     signal data : std_logic_vector(15 downto 0) := (others => '0');
58     signal wrreq : std_logic := '0';

```

```

59     signal rdreq : std_logic := '0';
60     signal rdclk : std_logic := '0';
61     signal wrclk : std_logic := '0';
62     signal aclr : std_logic := '0';
63
64     -- Outputs
65     signal q : std_logic_vector(15 downto 0);
66     signal rdfull : std_logic;
67     signal rdempty : std_logic;
68     signal rdusedw : std_logic_vector(6 downto 0);
69     signal wrfull : std_logic;
70     signal wrempty : std_logic;
71     signal wrusedw : std_logic_vector(6 downto 0);
72
73     -- Clock Period
74     constant rdclk_period : time := 10ns;
75     constant wrclk_period : time := 10ns;
76 begin
77     -- Instantiate the UUT.
78     UUT : fifo
79         generic map(width => 16, depth => 128, addr_bits => 7)
80         port map(data => data, wrreq => wrreq, rdreq => rdreq, rdclk => rdclk,
81                wrclk => wrclk, aclr => aclr, q => q, rdfull => rdfull, rdempty => rdempty,
82                rdusedw => rdusedw, wrfull => wrfull, wrempty => wrempty, wrusedw => wrusedw);
83
84     -- Clock Process (rdclk)
85     rdclk_process : process
86     begin
87         rdclk <= '0';
88         wait for rdclk_period / 2;
89         rdclk <= '1';
90         wait for rdclk_period / 2;
91     end process rdclk_process;
92
93     wrclk_process : process
94     begin
95         wrclk <= '0';
96         wait for wrclk_period / 2;
97         wrclk <= '1';
98         wait for wrclk_period / 2;
99     end process wrclk_process;
100
101     -- Stimulus Process
102     stim_proc : process
103     begin
104         -- Do nothing for one clock period.
105         wait for rdclk_period;
106
107         -- Assert the reset.
108         aclr <= '1';
109         wait for rdclk_period;
110         aclr <= '0';
111
112         -- Write a few items into the FIFO.
113         data <= "0000111100001111";
114         wrreq <= '1';
115         wait for wrclk_period;

```



```

116
117     data <= "1111000011110000";
118     wrreq <= '1';
119     wait for wrclk_period;
120
121     data <= "1100110011001100";
122     wrreq <= '1';
123     wait for wrclk_period;
124
125     data <= "0011001100110011";
126     wrreq <= '1';
127     wait for wrclk_period;
128
129     -- Read back from the FIFO.
130     wrreq <= '0';
131     rdreq <= '1';
132     wait for 4 * rdclk_period;
133
134     -- What if we read extra?
135     rdreq <= '1';
136     wait for rdclk_period;
137
138     -- Send a clear to the FIFO.
139     aclr <= '1';
140     wait for rdclk_period;
141
142     -- Now, completely fill the FIFO.
143     aclr <= '0';
144     rdreq <= '0';
145     for i in 0 to 127 loop
146         data <= conv_std_logic_vector(i, 16);
147         wrreq <= '1';
148         wait for wrclk_period;
149     end loop;
150
151     -- Wait an extra clock to examine the output signals.
152     wrreq <= '0';
153     wait for rdclk_period;
154
155     -- Write one extra value to the FIFO. What happens?
156     data <= "1111111111111111";
157     wrreq <= '1';
158     wait for wrclk_period;
159
160     -- Read back the entire FIFO.
161     wrreq <= '0';
162     for i in 0 to 127 loop
163         rdreq <= '1';
164         wait for rdclk_period;
165     end loop;
166
167     -- Write a few values back into the FIFO.
168     rdreq <= '0';
169     for i in 0 to 3 loop
170         data <= conv_std_logic_vector(i, 16);
171         wrreq <= '1';
172         wait for wrclk_period;

```

```

173     end loop;
174
175     -- Now, read and write simultaneously for 10 clocks.
176     for i in 0 to 9 loop
177         data <= conv_std_logic_vector(i + 10, 16);
178         wrreq <= '1';
179         rdreq <= '1';
180         wait for wrclk_period;
181     end loop;
182
183     -- Send a clear signal.
184     aclr <= '1';
185     wait for rdclk_period;
186
187     -- What if we read now?
188     rdreq <= '1';
189     wait for 3 * rdclk_period;
190
191     -- End test.
192     wait;
193 end process stim_proc;
194 end behavioral;

```

Listing C.15: TB_io_pins.vhd

```

1  -----
2  -- Test Bench for io_pins
3  -- Last Modified: 14 February 2011
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2010 Worcester Polytechnic Institute
7  --
8  -- This program is free software; you can redistribute it and/or modify
9  -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_UNSIGNED.ALL;
26
27 library WORK;
28 use WORK.FPGA_REGS_COMMON.ALL;
29
30 entity TB_io_pins is
31 end TB_io_pins;

```

```

32
33 architecture behavioral of TB_io_pins is
34     -- Component declaration for the Unit Under Test (UUT)
35     component io_pins is
36         port(
37             io_0           : inout std_logic_vector(15 downto 0);
38             io_1           : inout std_logic_vector(15 downto 0);
39             io_2           : inout std_logic_vector(15 downto 0);
40             io_3           : inout std_logic_vector(15 downto 0);
41             reg_0          : in      std_logic_vector(15 downto 0);
42             reg_1          : in      std_logic_vector(15 downto 0);
43             reg_2          : in      std_logic_vector(15 downto 0);
44             reg_3          : in      std_logic_vector(15 downto 0);
45             clock          : in      std_logic;
46             rx_reset      : in      std_logic;
47             tx_reset      : in      std_logic;
48             serial_addr   : in      std_logic_vector(6 downto 0);
49             serial_data   : in      std_logic_vector(31 downto 0);
50             serial_strobe : in      std_logic
51         );
52     end component io_pins;
53
54     -- Inputs
55     signal reg_0 : std_logic_vector(15 downto 0) := (others => '0');
56     signal reg_1 : std_logic_vector(15 downto 0) := (others => '0');
57     signal reg_2 : std_logic_vector(15 downto 0) := (others => '0');
58     signal reg_3 : std_logic_vector(15 downto 0) := (others => '0');
59     signal clock : std_logic := '0';
60     signal rx_reset : std_logic := '0';
61     signal tx_reset : std_logic := '0';
62     signal serial_addr : std_logic_vector(6 downto 0) := (others => '0');
63     signal serial_data : std_logic_vector(31 downto 0) := (others => '0');
64     signal serial_strobe : std_logic := '0';
65
66     -- Bidirectionals
67     signal io_0 : std_logic_vector(15 downto 0);
68     signal io_1 : std_logic_vector(15 downto 0);
69     signal io_2 : std_logic_vector(15 downto 0);
70     signal io_3 : std_logic_vector(15 downto 0);
71
72     -- Clock Period
73     constant clock_period : time := 10ns;
74 begin
75     -- Instantiate the UUT.
76     UUT : io_pins
77         port map(io_0 => io_0, io_1 => io_1, io_2 => io_2, io_3 => io_3,
78             reg_0 => reg_0, reg_1 => reg_1, reg_2 => reg_2, reg_3 => reg_3,
79             clock => clock, rx_reset => rx_reset, tx_reset => tx_reset,
80             serial_addr => serial_addr, serial_data => serial_data,
81             serial_strobe => serial_strobe);
82
83     -- Clock Process
84     clock_process : process
85     begin
86         clock <= '0';
87         wait for clock_period / 2;
88         clock <= '1';

```

```

89         wait for clock_period / 2;
90     end process clock_process;
91
92     -- Stimulus Process
93     stim_proc : process
94     begin
95         -- Do nothing for one clock period.
96         wait for clock_period;
97
98         -- Set some initial values for the input registers.
99         reg_0 <= "0001000100010001";
100        reg_1 <= "0010001000100010";
101        reg_2 <= "0011001100110011";
102        reg_3 <= "0100010001000100";
103
104        -- Enable all all outputs for each register.
105        serial_data <= (others => '1');
106
107        serial_addr <= conv_std_logic_vector(FR_OE_0, serial_addr'LENGTH);
108        serial_strobe <= '1';
109        wait for clock_period;
110        serial_strobe <= '0';
111        wait for clock_period;
112
113        serial_addr <= conv_std_logic_vector(FR_OE_1, serial_addr'LENGTH);
114        serial_strobe <= '1';
115        wait for clock_period;
116        serial_strobe <= '0';
117        wait for clock_period;
118
119        serial_addr <= conv_std_logic_vector(FR_OE_2, serial_addr'LENGTH);
120        serial_strobe <= '1';
121        wait for clock_period;
122        serial_strobe <= '0';
123        wait for clock_period;
124
125        serial_addr <= conv_std_logic_vector(FR_OE_3, serial_addr'LENGTH);
126        serial_strobe <= '1';
127        wait for clock_period;
128        serial_strobe <= '0';
129        wait for clock_period;
130
131        -- Change the values of each register.
132        reg_0 <= "1111111111111111";
133        wait for clock_period;
134
135        reg_1 <= "0000000000000000";
136        wait for clock_period;
137
138        reg_2 <= "1111000011110000";
139        wait for clock_period;
140
141        reg_3 <= "0000111100001111";
142        wait for clock_period;
143
144        -- Change the output masks for each register.
145        serial_data <= "11111111000000000000000000000000";

```

```

146     serial_addr <= conv_std_logic_vector(FR_OE_0, serial_addr 'LENGTH);
147     serial_strobe <= '1';
148     wait for clock_period;
149     serial_strobe <= '0';
150     wait for clock_period;
151
152     serial_data <= "1111111111111111100000000000000000";
153     serial_addr <= conv_std_logic_vector(FR_OE_1, serial_addr 'LENGTH);
154     serial_strobe <= '1';
155     wait for clock_period;
156     serial_strobe <= '0';
157     wait for clock_period;
158
159     serial_data <= "1111000011110000000000000000000000";
160     serial_addr <= conv_std_logic_vector(FR_OE_2, serial_addr 'LENGTH);
161     serial_strobe <= '1';
162     wait for clock_period;
163     serial_strobe <= '0';
164     wait for clock_period;
165
166     serial_data <= "1111111111111111111111111111111111";
167     serial_addr <= conv_std_logic_vector(FR_OE_3, serial_addr 'LENGTH);
168     serial_strobe <= '1';
169     wait for clock_period;
170     serial_strobe <= '0';
171     wait for clock_period;
172
173     -- The two reset signals are never used in io_pins, but we'll test them
174     -- anyway.
175     tx_reset <= '1';
176     wait for clock_period;
177
178     rx_reset <= '1';
179     wait for clock_period;
180
181     tx_reset <= '0';
182     wait for clock_period;
183
184     rx_reset <= '0';
185     wait for clock_period;
186
187     -- End test.
188     wait;
189 end process stim_proc;
190 end behavioral;

```

Listing C.16: TB_mult.vhd

```

1 -----
2 -- Test Bench for mult
3 -- Last Modified: 15 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 Worcester Polytechnic Institute
7 --
8 -- This program is free software; you can redistribute it and/or modify

```

```

9  --  it under the terms of the GNU General Public License as published by
10 --  the Free Software Foundation; either version 2 of the License, or
11 --  (at your option) any later version.
12 --
13 --  This program is distributed in the hope that it will be useful,
14 --  but WITHOUT ANY WARRANTY; without even the implied warranty of
15 --  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 --  GNU General Public License for more details.
17 --
18 --  You should have received a copy of the GNU General Public License
19 --  along with this program; if not, write to the Free Software
20 --  Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 entity TB_mult is
26 end TB_mult;
27
28 architecture behavioral of TB_mult is
29     -- Component declaration for the Unit Under Test (UUT)
30     component mult is
31         port(
32             clock          : in    std_logic;
33             x               : in    std_logic_vector(15 downto 0);
34             y               : in    std_logic_vector(15 downto 0);
35             product        : out   std_logic_vector(30 downto 0);
36             enable_in      : in    std_logic;
37             enable_out     : out   std_logic
38         );
39     end component mult;
40
41     -- Inputs
42     signal clock : std_logic := '0';
43     signal x : std_logic_vector(15 downto 0) := (others => '0');
44     signal y : std_logic_vector(15 downto 0) := (others => '0');
45     signal enable_in : std_logic := '0';
46
47     -- Outputs
48     signal product : std_logic_vector(30 downto 0);
49     signal enable_out : std_logic;
50
51     -- Clock Period
52     constant clock_period : time := 10ns;
53 begin
54     -- Instantiate the UUT.
55     UUT : mult
56         port map(clock => clock, x => x, y => y, product => product,
57             enable_in => enable_in, enable_out => enable_out);
58
59     -- Clock Process
60     clock_process : process
61     begin
62         clock <= '0';
63         wait for clock_period / 2;
64         clock <= '1';
65         wait for clock_period / 2;

```

```

66     end process clock_process;
67
68     -- Stimulus Process
69     stim_proc : process
70     begin
71         -- Do nothing for one clock period.
72         wait for clock_period;
73
74         -- Enable multiplications.
75         enable_in <= '1';
76
77         -- Multiply two positive numbers.
78         x <= "0000000000001111";
79         y <= "0000000000001111";
80         wait for clock_period;
81
82         -- Multiply one positive and one negative number.
83         x <= "0000000000001100";
84         y <= "1111111111110000";
85         wait for clock_period;
86
87         -- Multiply two negative numbers.
88         x <= "1111111111111000";
89         y <= "1111111111110100";
90         wait for clock_period;
91
92         -- Multiply by zero.
93         x <= "0000000000000000";
94         y <= "0000000000001111";
95         wait for clock_period;
96
97         -- Ensure nothing happens when enable_in is low.
98         enable_in <= '0';
99         wait for 3 * clock_period;
100
101         -- End test.
102         wait;
103     end process stim_proc;
104 end behavioral;

```

Listing C.17: TB_phase_acc.vhd

```

1 -----
2 -- Test Bench for phase_acc
3 -- Last Modified: 23 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2011 Worcester Polytechnic Institute
7 --
8 -- This program is free software; you can redistribute it and/or modify
9 -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of

```

```

15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_UNSIGNED.ALL;
26
27 library WORK;
28 use WORK.FPGA_REGS_STANDARD.ALL;
29
30 entity TB_phase_acc is
31 end TB_phase_acc;
32
33 architecture behavioral of TB_phase_acc is
34     -- Component declaration for the Unit Under Test (UUT)
35     component phase_acc is
36         generic(
37             FREQADDR      : integer := 0;
38             PHASEADDR     : integer := 0;
39             resolution    : integer range 0 to 32 := 32
40         );
41
42         port(
43             clk           : in    std_logic;
44             reset        : in    std_logic;
45             enable       : in    std_logic;
46             strobe       : in    std_logic;
47             serial_addr  : in    std_logic_vector(6 downto 0);
48             serial_data  : in    std_logic_vector(31 downto 0);
49             serial_strobe : in    std_logic;
50             phase       : out   std_logic_vector(resolution-1 downto 0)
51         );
52     end component phase_acc;
53
54     -- Inputs
55     signal clk : std_logic := '0';
56     signal reset : std_logic := '0';
57     signal enable : std_logic := '0';
58     signal strobe : std_logic := '0';
59     signal serial_addr : std_logic_vector(6 downto 0) := (others => '0');
60     signal serial_data : std_logic_vector(31 downto 0) := (others => '0');
61     signal serial_strobe : std_logic := '0';
62
63     -- Outputs
64     signal phase : std_logic_vector(31 downto 0);
65
66     -- Clock Period
67     constant clk_period : time := 10ns;
68 begin
69     -- Instantiate the UUT.
70     UUT : phase_acc
71         generic map(FREQADDR => FR_RX_FREQ_0, PHASEADDR => FR_RX_PHASE_0)

```



```

72     port map(clk => clk, reset => reset, enable => enable, strobe => strobe,
73             serial_addr => serial_addr, serial_data => serial_data,
74             serial_strobe => serial_strobe, phase => phase);
75
76     -- Clock Process
77     clk_process : process
78     begin
79         clk <= '0';
80         wait for clk_period / 2;
81         clk <= '1';
82         wait for clk_period / 2;
83     end process clk_process;
84
85     -- Strobe Process
86     -- We'll just use a 1/4 strobe rate for testing purposes. In practice, this
87     -- value will be determined by the decimation rate.
88     strobe_process : process
89     begin
90         strobe <= '0';
91         wait for 3 * clk_period;
92         strobe <= '1';
93         wait for clk_period;
94     end process strobe_process;
95
96     -- Stimulus Process
97     stim_proc : process
98     begin
99         -- Do nothing for one clock period.
100        wait for clk_period;
101
102        -- Assert the reset.
103        reset <= '1';
104        wait for clk_period;
105
106        -- Set the frequency to 45.
107        serial_addr <= conv_std_logic_vector(FR_RX_FREQ_0, serial_addr'LENGTH);
108        serial_data <= conv_std_logic_vector(45, serial_data'LENGTH);
109        serial_strobe <= '1';
110        wait for clk_period;
111
112        -- Enable the unit.
113        reset <= '0';
114        enable <= '1';
115        serial_strobe <= '0';
116        wait for 3 * clk_period;
117
118        -- Set a new starting phase of 90.
119        serial_addr <= conv_std_logic_vector(FR_RX_PHASE_0, serial_addr'LENGTH);
120        serial_data <= conv_std_logic_vector(90, serial_data'LENGTH);
121        serial_strobe <= '1';
122        wait for clk_period;
123
124        -- Accumulate for a while.
125        serial_strobe <= '0';
126        wait for 10 * clk_period;
127
128        -- End test.

```

```

129         wait;
130     end process stim_proc;
131 end behavioral;

```

Listing C.18: TB_ram16.vhd

```

1 -----
2 -- Test Bench for ram16
3 -- Last Modified: 14 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 Worcester Polytechnic Institute
7 --
8 -- This program is free software; you can redistribute it and/or modify
9 -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_UNSIGNED.ALL;
26
27 entity TB_ram16 is
28 end TB_ram16;
29
30 architecture behavioral of TB_ram16 is
31     -- Component declaration for the Unit Under Test (UUT)
32     component ram16 is
33         port(
34             clock          : in    std_logic;
35             write_in       : in    std_logic;
36             wr_addr        : in    std_logic_vector(3 downto 0);
37             wr_data        : in    std_logic_vector(15 downto 0);
38             rd_addr        : in    std_logic_vector(3 downto 0);
39             rd_data        : out   std_logic_vector(15 downto 0)
40         );
41     end component ram16;
42
43     -- Inputs
44     signal clock : std_logic := '0';
45     signal write_in : std_logic := '0';
46     signal wr_addr : std_logic_vector(3 downto 0) := (others => '0');
47     signal wr_data : std_logic_vector(15 downto 0) := (others => '0');
48     signal rd_addr : std_logic_vector(3 downto 0) := (others => '0');
49
50     -- Outputs

```

```

51     signal rd_data : std_logic_vector(15 downto 0);
52
53     -- Clock Period
54     constant clock_period : time := 10ns;
55 begin
56     -- Instantiate the UUT.
57     UUT : ram16
58         port map(clock => clock, write_in => write_in, wr_addr => wr_addr,
59                 wr_data => wr_data, rd_addr => rd_addr, rd_data => rd_data);
60
61     -- Clock Process
62     clock_process : process
63     begin
64         clock <= '0';
65         wait for clock_period / 2;
66         clock <= '1';
67         wait for clock_period / 2;
68     end process;
69
70     -- Stimulus Process
71     stim_proc : process
72     begin
73         -- Do nothing for one clock period.
74         wait for clock_period;
75
76         -- Write to all 16 memory locations.
77         write_in <= '1';
78         for i in 0 to 15 loop
79             wr_addr <= conv_std_logic_vector(i, wr_addr'LENGTH);
80             wr_data <= conv_std_logic_vector(15 - i, wr_data'LENGTH);
81             wait for clock_period;
82         end loop;
83
84         -- Now read back from each memory location.
85         write_in <= '0';
86         for i in 0 to 15 loop
87             rd_addr <= conv_std_logic_vector(i, rd_addr'LENGTH);
88             wait for clock_period;
89         end loop;
90
91         -- Make sure that we can't write when write_in is low.
92         wr_addr <= "0100";
93         wr_data <= "0000000011111111";
94         wait for clock_period;
95
96         rd_addr <= "0100";
97         wait for clock_period;
98
99         -- End test.
100        wait;
101    end process stim_proc;
102 end behavioral;

```

Listing C.19: TB_ram16_2sum.vhd

```

2  -- Test Bench for ram16_2sum
3  -- Last Modified: 14 February 2011
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2010 Worcester Polytechnic Institute
7  --
8  -- This program is free software; you can redistribute it and/or modify
9  -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_UNSIGNED.ALL;
26
27 entity TB_ram16_2sum is
28 end TB_ram16_2sum;
29
30 architecture behavioral of TB_ram16_2sum is
31     -- Component declaration for the Unit Under Test (UUT)
32     component ram16_2sum is
33         port(
34             clock      : in    std_logic;
35             write_in   : in    std_logic;
36             wr_addr    : in    std_logic_vector(3 downto 0);
37             wr_data    : in    std_logic_vector(15 downto 0);
38             rd_addr1   : in    std_logic_vector(3 downto 0);
39             rd_addr2   : in    std_logic_vector(3 downto 0);
40             sum        : out   std_logic_vector(15 downto 0)
41         );
42     end component ram16_2sum;
43
44     -- Inputs
45     signal clock : std_logic := '0';
46     signal write_in : std_logic := '0';
47     signal wr_addr : std_logic_vector(3 downto 0) := (others => '0');
48     signal wr_data : std_logic_vector(15 downto 0) := (others => '0');
49     signal rd_addr1 : std_logic_vector(3 downto 0) := (others => '0');
50     signal rd_addr2 : std_logic_vector(3 downto 0) := (others => '0');
51
52     -- Outputs
53     signal sum : std_logic_vector(15 downto 0);
54
55     -- Clock Period
56     constant clock_period : time := 10ns;
57 begin
58     -- Instantiate the UUT.

```

```

59     UUT : ram16_2sum
60     port map(clock => clock , write_in => write_in , wr_addr => wr_addr ,
61             wr_data => wr_data , rd_addr1 => rd_addr1 , rd_addr2 => rd_addr2 ,
62             sum => sum);
63
64     -- Clock Process
65     clock_process : process
66     begin
67         clock <= '0';
68         wait for clock_period / 2;
69         clock <= '1';
70         wait for clock_period / 2;
71     end process;
72
73     -- Stimulus Process
74     stim_proc : process
75     begin
76         -- Do nothing for one clock period.
77         wait for clock_period;
78
79         -- Write to all 16 memory locations.
80         write_in <= '1';
81         for i in 0 to 15 loop
82             wr_addr <= conv_std_logic_vector(i, wr_addr'LENGTH);
83             wr_data <= conv_std_logic_vector(conv_integer(conv_signed(-i + 7, wr_data'LENGTH)),
84             wr_data'LENGTH);
85             wait for clock_period;
86         end loop;
87
88         -- Now read back from each memory location.
89         write_in <= '0';
90         for i in 0 to 15 loop
91             rd_addr1 <= conv_std_logic_vector(i, rd_addr1'LENGTH);
92             rd_addr2 <= conv_std_logic_vector(15 - i, rd_addr2'LENGTH);
93             wait for clock_period;
94         end loop;
95
96         -- Make sure that we can't write when write_in is low.
97         wr_data <= "0000000011111111";
98         wait for 2 * clock_period;
99
100        -- Finally, test the 'add 1' part.
101        write_in <= '1';
102        wr_addr <= "0010";
103        wr_data <= "1111000000001111";
104        wait for clock_period;
105
106        wr_addr <= "0001";
107        wr_data <= "0000000000000000";
108        wait for clock_period;
109
110        write_in <= '0';
111        rd_addr1 <= "0010";
112        rd_addr2 <= "0001";
113        wait for 3 * clock_period;
114
115        -- End test.

```

```

115     wait;
116     end process stim_proc;
117 end behavioral;

```

Listing C.20: TB_rssi.vhd

```

1 -----
2 -- Test Bench for rssi
3 -- Last Modified: 26 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2011 Worcester Polytechnic Institute
7 --
8 -- This program is free software; you can redistribute it and/or modify
9 -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_UNSIGNED.ALL;
26
27 entity TB_rssi is
28 end TB_rssi;
29
30 architecture behavioral of TB_rssi is
31     -- Component declaration for the Unit Under Test (UUT)
32     component rssi is
33         port(
34             clock      : in   std_logic;
35             reset      : in   std_logic;
36             enable     : in   std_logic;
37             adc        : in   std_logic_vector(11 downto 0);
38             rssi       : out  std_logic_vector(15 downto 0);
39             over_count : out  std_logic_vector(15 downto 0)
40         );
41     end component rssi;
42
43     -- Inputs
44     signal clock : std_logic := '0';
45     signal reset : std_logic := '0';
46     signal enable : std_logic := '0';
47     signal adc : std_logic_vector(11 downto 0) := (others => '0');
48
49     -- Outputs
50     signal rssi_t : std_logic_vector(15 downto 0);

```

```

51     signal over_count : std_logic_vector(15 downto 0);
52
53     -- Clock Period
54     constant clock_period : time := 10ns;
55 begin
56     -- Instantiate the UUT.
57     UUT : rssi
58         port map(clock => clock, reset => reset, enable => enable, adc => adc,
59                 rssi => rssi_t, over_count => over_count);
60
61     -- Clock Process
62     clock_process : process
63     begin
64         clock <= '0';
65         wait for clock_period / 2;
66         clock <= '1';
67         wait for clock_period / 2;
68     end process clock_process;
69
70     -- Stimulus Process
71     stim_proc : process
72     begin
73         -- Do nothing for one clock period.
74         wait for clock_period;
75
76         -- Assert the reset.
77         reset <= '1';
78         wait for clock_period;
79
80         -- Enable the unit.
81         reset <= '0';
82         enable <= '1';
83         wait for clock_period;
84
85         -- Test some non-error values. These are signed.
86         for i in 0 to 19 loop
87             adc <= conv_std_logic_vector(i * 200, adc'LENGTH);
88             wait for clock_period;
89             adc <= conv_std_logic_vector(-(i * 200), adc'LENGTH);
90             wait for clock_period;
91         end loop;
92
93         -- Signal overflows.
94         for i in 0 to 3 loop
95             adc <= "011111111111";
96             wait for clock_period;
97         end loop;
98
99         -- Signal underflows.
100        for i in 0 to 3 loop
101            adc <= "111111111111";
102            wait for clock_period;
103        end loop;
104
105        -- Test the reset again.
106        reset <= '1';
107        wait for clock_period;

```

```

108
109     -- End test.
110     wait;
111     end process stim_proc;
112 end behavioral;

```

Listing C.21: TB_rx_dcoffset.vhd

```

1  -----
2  -- Test Bench for rx_dcoffset
3  -- Last Modified: 28 February 2011
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2011 Worcester Polytechnic Institute
7  --
8  -- This program is free software; you can redistribute it and/or modify
9  -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_SIGNED.ALL;
26
27 library WORK;
28 use WORK.FPGA_REGS_COMMON.ALL;
29
30 entity TB_rx_dcoffset is
31 end TB_rx_dcoffset;
32
33 architecture behavioral of TB_rx_dcoffset is
34     -- Component declaration for the Unit Under Test (UUT)
35     component rx_dcoffset is
36         generic(
37             MYADDR : integer := 0
38         );
39
40     port(
41         clock          : in    std_logic;
42         enable         : in    std_logic;
43         reset          : in    std_logic;
44         adc_in         : in    std_logic_vector(15 downto 0);
45         adc_out        : out   std_logic_vector(15 downto 0);
46         serial_addr    : in    std_logic_vector(6  downto 0);
47         serial_data    : in    std_logic_vector(31 downto 0);
48         serial_strobe  : in    std_logic

```



```

49     );
50     end component rx_dcoffset;
51
52     -- Inputs
53     signal clock : std_logic := '0';
54     signal enable : std_logic := '0';
55     signal reset : std_logic := '0';
56     signal adc_in : std_logic_vector(15 downto 0) := (others => '0');
57     signal serial_addr : std_logic_vector(6 downto 0) := (others => '0');
58     signal serial_data : std_logic_vector(31 downto 0) := (others => '0');
59     signal serial_strobe : std_logic := '0';
60
61     -- Outputs
62     signal adc_out : std_logic_vector(15 downto 0);
63
64     -- Clock Period
65     constant clock_period : time := 10ns;
66 begin
67     -- Instantiate the UUT.
68     UUT : rx_dcoffset
69         generic map(MYADDR => FR_ADC_OFFSET_0)
70         port map(clock => clock, enable => enable, reset => reset, adc_in => adc_in,
71             adc_out => adc_out, serial_addr => serial_addr, serial_data => serial_data,
72             serial_strobe => serial_strobe);
73
74     -- Clock Process
75     clock_process : process
76     begin
77         clock <= '0';
78         wait for clock_period / 2;
79         clock <= '1';
80         wait for clock_period / 2;
81     end process clock_process;
82
83     -- Stimulus Process
84     stim_proc : process
85     begin
86         -- Do nothing for one clock period.
87         wait for clock_period;
88
89         -- Assert the reset.
90         reset <= '1';
91         wait for clock_period;
92
93         -- Enable the unit.
94         reset <= '0';
95         enable <= '1';
96         wait for clock_period;
97
98         -- Set up the serial bus.
99         serial_addr <= conv_std_logic_vector(FR_ADC_OFFSET_0, serial_addr'LENGTH);
100        serial_data <= "00000000111111110000000000000000";
101        serial_strobe <= '1';
102        wait for clock_period;
103
104        -- Add some more samples.
105        serial_strobe <= '0';

```

```

106     for i in 0 to 15 loop
107         adc_in <= "0111000011110000";
108         wait for clock_period;
109     end loop;
110
111     -- Try a negative sample or two!
112     for i in 0 to 3 loop
113         adc_in <= "1000000000000011";
114         wait for clock_period;
115     end loop;
116
117     -- End test.
118     wait;
119 end process stim_proc;
120 end behavioral;

```

Listing C.22: TB_serial_io.vhd

```

1 -----
2 -- Test Bench for serial_io
3 -- Last Modified: 14 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 Worcester Polytechnic Institute
7 --
8 -- This program is free software; you can redistribute it and/or modify
9 -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 entity TB_serial_io is
26 end TB_serial_io;
27
28 architecture behavioral of TB_serial_io is
29     -- Component declaration for the Unit Under Test (UUT)
30     component serial_io is
31         port(
32             master_clk           : in    std_logic;
33             serial_clock         : in    std_logic;
34             serial_data_in       : in    std_logic;
35             enable                : in    std_logic;
36             reset                 : in    std_logic;
37             serial_data_out      : out   std_logic;
38             serial_addr          : out   std_logic_vector(6 downto 0);

```

```

39         serial_data      : out   std_logic_vector(31 downto 0);
40         serial_strobe    : out   std_logic;
41         readback_0       : in    std_logic_vector(31 downto 0);
42         readback_1       : in    std_logic_vector(31 downto 0);
43         readback_2       : in    std_logic_vector(31 downto 0);
44         readback_3       : in    std_logic_vector(31 downto 0);
45         readback_4       : in    std_logic_vector(31 downto 0);
46         readback_5       : in    std_logic_vector(31 downto 0);
47         readback_6       : in    std_logic_vector(31 downto 0);
48         readback_7       : in    std_logic_vector(31 downto 0)
49     );
50 end component serial_io;
51
52 -- Inputs
53 signal master_clk : std_logic := '0';
54 signal serial_clock : std_logic := '0';
55 signal serial_data_in : std_logic := '0';
56 signal enable : std_logic := '0';
57 signal reset : std_logic := '0';
58 signal readback_0 : std_logic_vector(31 downto 0) := (others => '0');
59 signal readback_1 : std_logic_vector(31 downto 0) := (others => '0');
60 signal readback_2 : std_logic_vector(31 downto 0) := (others => '0');
61 signal readback_3 : std_logic_vector(31 downto 0) := (others => '0');
62 signal readback_4 : std_logic_vector(31 downto 0) := (others => '0');
63 signal readback_5 : std_logic_vector(31 downto 0) := (others => '0');
64 signal readback_6 : std_logic_vector(31 downto 0) := (others => '0');
65 signal readback_7 : std_logic_vector(31 downto 0) := (others => '0');
66
67 -- Outputs
68 signal serial_data_out : std_logic;
69 signal serial_addr : std_logic_vector(6 downto 0);
70 signal serial_data : std_logic_vector(31 downto 0);
71 signal serial_strobe : std_logic;
72
73 -- Clock Periods
74 constant master_clk_period : time := 10ns;
75 constant serial_clock_period : time := 10ns;
76
77 -- Constants for operation instructions.
78 constant read_reg0 : std_logic_vector(7 downto 0) := "10000001";
79 constant read_reg1 : std_logic_vector(7 downto 0) := "10000010";
80 constant read_reg2 : std_logic_vector(7 downto 0) := "10000011";
81 constant read_reg3 : std_logic_vector(7 downto 0) := "10000100";
82 constant read_reg4 : std_logic_vector(7 downto 0) := "10000101";
83 constant read_reg5 : std_logic_vector(7 downto 0) := "10000110";
84 constant read_reg6 : std_logic_vector(7 downto 0) := "10000111";
85 constant read_reg7 : std_logic_vector(7 downto 0) := "10001000";
86 constant write_1 : std_logic_vector(39 downto 0) :=
87     "00000000110011001100110011001100110011001100";
88 constant write_2 : std_logic_vector(39 downto 0) :=
89     "00000000001100110011001100110011001100110011";
90
91 begin
92     -- Instantiate the UUT.
93     UUT : serial_io
94         port map(master_clk => master_clk, serial_clock => serial_clock,
95                 serial_data_in => serial_data_in, enable => enable, reset => reset,
96                 serial_data_out => serial_data_out, serial_addr => serial_addr,

```

```

94         serial_data => serial_data , serial_strobe => serial_strobe ,
95         readback_0 => readback_0 , readback_1 => readback_1 ,
96         readback_2 => readback_2 , readback_3 => readback_3 ,
97         readback_4 => readback_4 , readback_5 => readback_5 ,
98         readback_6 => readback_6 , readback_7 => readback_7);
99
100     -- Clock Process (master_clk)
101     master_clk_process : process
102     begin
103         master_clk <= '0';
104         wait for master_clk_period / 2;
105         master_clk <= '1';
106         wait for master_clk_period / 2;
107     end process master_clk_process;
108
109     -- Clock Process (serial_clock)
110     serial_clock_process : process
111     begin
112         serial_clock <= '0';
113         wait for serial_clock_period / 2;
114         serial_clock <= '1';
115         wait for serial_clock_period / 2;
116     end process serial_clock_process;
117
118     -- Stimulus Process
119     stim_proc : process
120     begin
121         -- Do nothing for one clock period.
122         wait for master_clk_period;
123
124         -- Set values for the readback registers.
125         readback_0 <= "00000000000000000000000000000000";
126         readback_1 <= "00010001000100010001000100010001";
127         readback_2 <= "00100010001000100010001000100010";
128         readback_3 <= "00110011001100110011001100110011";
129         readback_4 <= "01000100010001000100010001000100";
130         readback_5 <= "01010101010101010101010101010101";
131         readback_6 <= "01100110011001100110011001100110";
132         readback_7 <= "01110111011101110111011101110111";
133
134         -- Enable the reset.
135         reset <= '1';
136         wait for master_clk_period;
137
138         -- Disable the reset.
139         reset <= '0';
140
141         -- Each operation is 40 cycles. First, we test reading of all eight
142         -- readback registers.
143         enable <= '1';
144         for i in read_reg0'RANGE loop
145             serial_data_in <= read_reg0(i);
146             wait for serial_clock_period;
147         end loop;
148
149         wait for master_clk_period;
150         enable <= '0';

```

```
151     wait for master_clk_period;
152     enable <= '1';
153
154     for i in read_reg1 'RANGE loop
155         serial_data_in <= read_reg1(i);
156         wait for serial_clock_period;
157     end loop;
158
159     wait for master_clk_period;
160     enable <= '0';
161     wait for master_clk_period;
162     enable <= '1';
163
164     for i in read_reg2 'RANGE loop
165         serial_data_in <= read_reg2(i);
166         wait for serial_clock_period;
167     end loop;
168
169     wait for master_clk_period;
170     enable <= '0';
171     wait for master_clk_period;
172     enable <= '1';
173
174     for i in read_reg3 'RANGE loop
175         serial_data_in <= read_reg3(i);
176         wait for serial_clock_period;
177     end loop;
178
179     wait for master_clk_period;
180     enable <= '0';
181     wait for master_clk_period;
182     enable <= '1';
183
184     for i in read_reg4 'RANGE loop
185         serial_data_in <= read_reg4(i);
186         wait for serial_clock_period;
187     end loop;
188
189     wait for master_clk_period;
190     enable <= '0';
191     wait for master_clk_period;
192     enable <= '1';
193
194     for i in read_reg5 'RANGE loop
195         serial_data_in <= read_reg5(i);
196         wait for serial_clock_period;
197     end loop;
198
199     wait for master_clk_period;
200     enable <= '0';
201     wait for master_clk_period;
202     enable <= '1';
203
204     for i in read_reg6 'RANGE loop
205         serial_data_in <= read_reg6(i);
206         wait for serial_clock_period;
207     end loop;
```

```

208
209     wait for master_clk_period;
210     enable <= '0';
211     wait for master_clk_period;
212     enable <= '1';
213
214     for i in read_reg7 'RANGE loop
215         serial_data_in <= read_reg7(i);
216         wait for serial_clock_period;
217     end loop;
218
219     wait for master_clk_period;
220     enable <= '0';
221     wait for master_clk_period;
222     enable <= '1';
223
224     -- Now, perform two write tests.
225     for i in write_1 'RANGE loop
226         serial_data_in <= write_1(i);
227         wait for serial_clock_period;
228     end loop;
229
230     wait for master_clk_period;
231     enable <= '0';
232     wait for master_clk_period;
233     enable <= '1';
234
235     for i in write_2 'RANGE loop
236         serial_data_in <= write_2(i);
237         wait for serial_clock_period;
238     end loop;
239
240     wait for master_clk_period;
241     enable <= '0';
242     wait for master_clk_period;
243
244     -- Disable the module while altering serial_in.
245     enable <= '0';
246     serial_data_in <= '1';
247     wait for serial_clock_period;
248     serial_data_in <= '0';
249     wait for serial_clock_period;
250     serial_data_in <= '1';
251     wait for serial_clock_period;
252
253     -- Enable the reset.
254     reset <= '1';
255     wait for master_clk_period;
256
257     -- End test.
258     wait;
259 end process stim_proc;
260 end behavioral;

```

Listing C.23: TB_setting_reg.vhd

```

1 -----
2 -- Test Bench for setting_reg
3 -- Last Modified: 13 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 Worcester Polytechnic Institute
7 --
8 -- This program is free software; you can redistribute it and/or modify
9 -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 entity TB_setting_reg is
26 end TB_setting_reg;
27
28 architecture behavioral of TB_setting_reg is
29     -- Component declaration for the Unit Under Test (UUT)
30     component setting_reg is
31         generic(
32             my_addr : integer := 0
33         );
34
35         port(
36             clock      : in    std_logic;
37             reset      : in    std_logic;
38             strobe     : in    std_logic;
39             addr       : in    std_logic_vector(6 downto 0);
40             d_in       : in    std_logic_vector(31 downto 0);
41             d_out      : out   std_logic_vector(31 downto 0);
42             changed   : out   std_logic
43         );
44     end component setting_reg;
45
46     -- Inputs
47     signal clock : std_logic := '0';
48     signal reset : std_logic := '0';
49     signal strobe : std_logic := '0';
50     signal addr : std_logic_vector(6 downto 0) := (others => '0');
51     signal d_in : std_logic_vector(31 downto 0) := (others => '0');
52
53     -- Outputs
54     signal d_out1 : std_logic_vector(31 downto 0);
55     signal d_out2 : std_logic_vector(31 downto 0);
56     signal d_out3 : std_logic_vector(31 downto 0);
57     signal changed1 : std_logic;

```

```

58     signal changed2 : std_logic;
59     signal changed3 : std_logic;
60
61     -- Clock period
62     constant clock_period : time := 10ns;
63 begin
64     -- Instantiate the UUTs.
65     UUT1 : setting_reg
66         generic map(my_addr => 1)
67         port map(clock => clock, reset => reset, strobe => strobe, addr => addr,
68             d_in => d_in, d_out => d_out1, changed => changed1);
69
70     UUT2 : setting_reg
71         generic map(my_addr => 2)
72         port map(clock => clock, reset => reset, strobe => strobe, addr => addr,
73             d_in => d_in, d_out => d_out2, changed => changed2);
74
75     UUT3 : setting_reg
76         generic map(my_addr => 3)
77         port map(clock => clock, reset => reset, strobe => strobe, addr => addr,
78             d_in => d_in, d_out => d_out3, changed => changed3);
79
80     -- Clock Process
81     clock_process : process
82     begin
83         clock <= '0';
84         wait for clock_period / 2;
85         clock <= '1';
86         wait for clock_period / 2;
87     end process clock_process;
88
89     -- Stimulus Process
90     stim_proc : process
91     begin
92         -- Do nothing for one clock.
93         wait for clock_period;
94
95         -- Enable reset.
96         reset <= '1';
97         wait for clock_period;
98
99         -- Change values while reset is asserted.
100        addr <= "0000001";
101        d_in <= "11110000110000111010010110010110";
102        strobe <= '1';
103        wait for clock_period;
104
105        -- Disable reset.
106        reset <= '0';
107        strobe <= '0';
108        wait for clock_period;
109
110        -- Assert the strobe.
111        strobe <= '1';
112        wait for clock_period;
113
114        -- Deassert the strobe.

```



```

115     strobe <= '0';
116     wait for clock_period;
117
118     -- Try three sequential writes to different registers.
119     addr <= "0000001";
120     d_in <= "00010001000100010001000100010001";
121     strobe <= '1';
122     wait for clock_period;
123
124     addr <= "0000010";
125     d_in <= "00100010001000100010001000100010";
126     strobe <= '1';
127     wait for clock_period;
128
129     addr <= "0000011";
130     d_in <= "00110011001100110011001100110011";
131     strobe <= '1';
132     wait for clock_period;
133
134     -- Try to write to a register that does not exist.
135     addr <= "0000100";
136     d_in <= "01000100010001000100010001000100";
137     strobe <= '1';
138     wait for clock_period;
139
140     -- Valid parameters, no strobe.
141     addr <= "0000001";
142     d_in <= "1111111111111111111111111111111111";
143     strobe <= '0';
144     wait for clock_period;
145
146     -- Enable the reset.
147     reset <= '1';
148     wait for clock_period;
149
150     -- End test.
151     wait;
152     end process stim_proc;
153 end behavioral;

```

Listing C.24: TB_sign_extend.vhd

```

1 -----
2 -- Test Bench for sign_extend
3 -- Last Modified: 14 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 Worcester Polytechnic Institute
7 --
8 -- This program is free software; you can redistribute it and/or modify
9 -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of

```

```

15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_UNSIGNED.ALL;
26
27 entity TB_sign_extend is
28 end TB_sign_extend;
29
30 architecture behavioral of TB_sign_extend is
31     -- Component declaration for the Unit Under Test (UUT)
32     component sign_extend is
33         generic(
34             bits_in : integer := 0;
35             bits_out  : integer := 0
36         );
37
38         port(
39             d_in   : in   std_logic_vector(bits_in-1 downto 0);
40             d_out  : out  std_logic_vector(bits_out-1 downto 0)
41         );
42     end component sign_extend;
43
44     -- Bus widths for the UUTs.
45     constant bin1 : integer := 4;
46     constant bout1 : integer := 6;
47
48     constant bin2 : integer := 2;
49     constant bout2 : integer := 5;
50
51     constant bin3 : integer := 4;
52     constant bout3 : integer := 4;
53
54     -- Inputs
55     signal d_in1 : std_logic_vector(bin1-1 downto 0) := (others => '0');
56     signal d_in2 : std_logic_vector(bin2-1 downto 0) := (others => '0');
57     signal d_in3 : std_logic_vector(bin3-1 downto 0) := (others => '0');
58
59     -- Outputs
60     signal d_out1 : std_logic_vector(bout1-1 downto 0);
61     signal d_out2 : std_logic_vector(bout2-1 downto 0);
62     signal d_out3 : std_logic_vector(bout3-1 downto 0);
63 begin
64     -- Instantiate the UUTs.
65     UUT1 : sign_extend
66         generic map(bits_in => bin1, bits_out => bout1)
67         port map(d_in => d_in1, d_out => d_out1);
68
69     UUT2 : sign_extend
70         generic map(bits_in => bin2, bits_out => bout2)
71         port map(d_in => d_in2, d_out => d_out2);

```

```

72
73     UUT3 : sign_extend
74         generic map(bits_in => bin3, bits_out => bout3)
75         port map(d_in => d_in3, d_out => d_out3);
76
77     -- Stimulus Process
78     stim_proc : process
79     begin
80         -- Do nothing for 10ns.
81         wait for 10ns;
82
83         -- Set all possible input values.
84         for i in 0 to 15 loop
85             d_in1 <= conv_std_logic_vector(i, d_in1'LENGTH);
86             d_in2 <= conv_std_logic_vector(i, d_in2'LENGTH);
87             d_in3 <= conv_std_logic_vector(i, d_in3'LENGTH);
88             wait for 10ns;
89         end loop;
90
91         -- End test.
92         wait;
93     end process stim_proc;
94 end behavioral;

```

Listing C.25: TB_strobe_gen.vhd

```

1 -----
2 -- Test Bench for strobe_gen
3 -- Last Modified: 14 February 2011
4 -- VHDL Author: Steve Olivieri
5 --
6 -- Copyright (C) 2010 Worcester Polytechnic Institute
7 --
8 -- This program is free software; you can redistribute it and/or modify
9 -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 entity TB_strobe_gen is
26 end TB_strobe_gen;
27
28 architecture behavioral of TB_strobe_gen is
29     -- Component declaration for the Unit Under Test (UUT)
30     component strobe_gen is

```

```

31     port(
32         clock          : in   std_logic;
33         reset          : in   std_logic;
34         enable         : in   std_logic;
35         rate           : in   std_logic_vector(7 downto 0);  -- 1 less than desired divide
36         ratio
37         strobe_in      : in   std_logic;
38         strobe         : out  std_logic
39     );
40
41     end component strobe_gen;
42
43     -- Inputs
44     signal clock : std_logic := '0';
45     signal reset : std_logic := '0';
46     signal enable : std_logic := '0';
47     signal rate : std_logic_vector(7 downto 0) := (others => '0');
48     signal strobe_in : std_logic := '0';
49
50     -- Outputs
51     signal strobe : std_logic;
52
53     -- Clock Period
54     constant clock_period : time := 10ns;
55
56     begin
57     -- Instantitate the UUT.
58     UUT : strobe_gen
59         port map(clock => clock, reset => reset, enable => enable, rate => rate,
60             strobe_in => strobe_in, strobe => strobe);
61
62     -- Clock Process
63     clock_process : process
64     begin
65         clock <= '0';
66         wait for clock_period / 2;
67         clock <= '1';
68         wait for clock_period / 2;
69     end process clock_process;
70
71     -- Stimulus Process
72     stim_proc : process
73     begin
74         -- Do nothing for one clock period.
75         wait for clock_period;
76
77         -- Assert the reset.
78         reset <= '1';
79         wait for clock_period;
80
81         -- Set the rate to 16.
82         rate <= "00010000";
83
84         -- Deassert the reset, assert the enable.
85         reset <= '0';
86         enable <= '1';
87         wait for clock_period;
88
89         -- As long as strobe_in is not asserted, nothing will happen.

```

```

87      -- Test this by waiting 16 clocks.
88      wait for 16 * clock_period;
89
90      -- Now, assert strobe_in. We'll just leave it that way for easier
91      -- testing, but normal operation might include periods of time between
92      -- strobe_in assertions.
93      strobe_in <= '1';
94      wait for 16 * clock_period;
95
96      -- Do it again.
97      wait for 16 * clock_period;
98
99      -- Disable.
100     enable <= '0';
101     wait for clock_period;
102
103     -- Set a new rate, assert enable. Wait for eight strobes.
104     rate <= "00000010";
105     enable <= '1';
106     wait for 16 * clock_period;
107
108     -- Assert the reset without disabling.
109     reset <= '1';
110     wait for clock_period;
111
112     -- Deassert the reset to continue operation.
113     reset <= '0';
114     wait for 4 * clock_period;
115
116     -- End test.
117     wait;
118 end process stim_proc;
119 end behavioral;

```

Listing C.26: TB_tx_chain.vhd

```

1  -----
2  -- Test Bench for tx_chain
3  -- Last Modified: 22 February 2011
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2011 Worcester Polytechnic Institute
7  --
8  -- This program is free software; you can redistribute it and/or modify
9  -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA

```

```

21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_UNSIGNED.ALL;
26
27 entity TB_tx_chain is
28 end TB_tx_chain;
29
30 architecture behavioral of TB_tx_chain is
31     -- Component declaration for the Unit Under Test (UUT)
32     component tx_chain is
33         port(
34             clock                : in    std_logic;
35             reset                 : in    std_logic;
36             enable                : in    std_logic;
37             interp_rate           : in    std_logic_vector(7 downto 0);
38             sample_strobe        : in    std_logic;
39             interpolator_strobe  : in    std_logic;
40             freq                  : in    std_logic_vector(31 downto 0);
41             i_in                  : in    std_logic_vector(15 downto 0);
42             q_in                  : in    std_logic_vector(15 downto 0);
43             i_out                 : out   std_logic_vector(15 downto 0);
44             q_out                 : out   std_logic_vector(15 downto 0)
45         );
46     end component tx_chain;
47
48     -- Inputs
49     signal clock : std_logic := '0';
50     signal reset : std_logic := '0';
51     signal enable : std_logic := '0';
52     signal interp_rate : std_logic_vector(7 downto 0) := (others => '0');
53     signal sample_strobe : std_logic := '0';
54     signal interpolator_strobe : std_logic := '0';
55     signal freq : std_logic_vector(31 downto 0) := (others => '0');
56     signal i_in : std_logic_vector(15 downto 0) := (others => '0');
57     signal q_in : std_logic_vector(15 downto 0) := (others => '0');
58
59     -- Outputs
60     signal i_out : std_logic_vector(15 downto 0);
61     signal q_out : std_logic_vector(15 downto 0);
62
63     -- Clock Period
64     constant clock_period : time := 10ns;
65 begin
66     -- Instantiate the UUT.
67     UUT : tx_chain
68         port map(clock => clock, reset => reset, enable => enable,
69             interp_rate => interp_rate, sample_strobe => sample_strobe,
70             interpolator_strobe => interpolator_strobe, freq => freq,
71             i_in => i_in, q_in => q_in, i_out => i_out, q_out => q_out);
72
73     -- Clock Process
74     clock_process : process
75     begin
76         clock <= '0';
77         wait for clock_period / 2;

```

```

78     clock <= '1';
79     wait for clock_period / 2;
80 end process clock_process;
81
82 -- Strobe Process
83 strobe_proc : process
84 begin
85     wait for 7 * clock_period;
86     interpolator_strobe <= '1';
87     wait for clock_period;
88     interpolator_strobe <= '0';
89 end process strobe_proc;
90
91 -- We want a sample on every clock, so just keep sample_strobe enabled.
92 sample_strobe <= '1';
93
94 -- Stimulus Process
95 stim_proc : process
96 begin
97     -- Do nothing for one clock cycle.
98     wait for clock_period;
99
100    -- Assert the reset.
101    reset <= '1';
102    wait for clock_period;
103
104    -- Enable the filter.
105    reset <= '0';
106    enable <= '1';
107
108    -- Set the rate to 7. This is actually a rate of 32 because of the
109    -- codecs that we use. The formula is (rate+1)*4.
110    interp_rate <= conv_std_logic_vector(7, interp_rate'LENGTH);
111
112    -- Test values of 1, max, min, and -1. Q will go in reverse.
113    i_in <= "0000000000000001";
114    q_in <= "1111111111111111";
115    wait for 100 * clock_period;
116    i_in <= "0111111111111111";
117    q_in <= "1000000000000000";
118    wait for 100 * clock_period;
119    i_in <= "1000000000000000";
120    q_in <= "0111111111111111";
121    wait for 100 * clock_period;
122    i_in <= "1111111111111111";
123    q_in <= "0000000000000001";
124    wait for 100 * clock_period;
125
126    -- End test.
127    wait;
128 end process stim_proc;
129 end behavioral;

```

Listing C.27: TB_usrp_mux.vhd

```

2  -- Test Bench for usrp_mux
3  -- Last Modified: 14 February 2011
4  -- VHDL Author: Steve Olivieri
5  --
6  -- Copyright (C) 2010 Worcester Polytechnic Institute
7  --
8  -- This program is free software; you can redistribute it and/or modify
9  -- it under the terms of the GNU General Public License as published by
10 -- the Free Software Foundation; either version 2 of the License, or
11 -- (at your option) any later version.
12 --
13 -- This program is distributed in the hope that it will be useful,
14 -- but WITHOUT ANY WARRANTY; without even the implied warranty of
15 -- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 -- GNU General Public License for more details.
17 --
18 -- You should have received a copy of the GNU General Public License
19 -- along with this program; if not, write to the Free Software
20 -- Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301 USA
21 -----
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_UNSIGNED.ALL;
26
27 entity TB_usrp_mux is
28 end TB_usrp_mux;
29
30 architecture behavioral of TB_usrp_mux is
31     -- Component declaration for the Unit Under Test (UUT)
32     component usrp_mux is
33         port(
34             dac_mux      : in    std_logic_vector(3 downto 0);
35             i_out_0      : in    std_logic_vector(15 downto 0);
36             q_out_0      : in    std_logic_vector(15 downto 0);
37             i_out_1      : in    std_logic_vector(15 downto 0);
38             q_out_1      : in    std_logic_vector(15 downto 0);
39             d_out        : out   std_logic_vector(15 downto 0)
40         );
41     end component usrp_mux;
42
43     -- Inputs
44     signal dac_mux : std_logic_vector(3 downto 0) := (others => '0');
45     signal i_out_0 : std_logic_vector(15 downto 0) := (others => '0');
46     signal q_out_0 : std_logic_vector(15 downto 0) := (others => '0');
47     signal i_out_1 : std_logic_vector(15 downto 0) := (others => '0');
48     signal q_out_1 : std_logic_vector(15 downto 0) := (others => '0');
49
50     -- Outputs
51     signal d_out : std_logic_vector(15 downto 0);
52 begin
53     -- Instantiate the UUT.
54     UUT : usrp_mux
55         port map(dac_mux => dac_mux, i_out_0 => i_out_0, q_out_0 => q_out_0,
56             i_out_1 => i_out_1, q_out_1 => q_out_1, d_out => d_out);
57
58     -- Stimulus Process

```



```
59     stim_proc : process
60     begin
61         -- Do nothing for 10ns.
62         wait for 10ns;
63
64         -- Set data inputs.
65         i_out_0 <= "0000000000000001";
66         q_out_0 <= "0000000000000010";
67         i_out_1 <= "0000000000000011";
68         q_out_1 <= "0000000000000100";
69
70         -- Provide every possible combination for dac_mux.
71         for i in 0 to 15 loop
72             dac_mux <= conv_std_logic_vector(i, dac_mux'LENGTH);
73             wait for 10ns;
74         end loop;
75
76         -- End test.
77         wait;
78     end process stim_proc;
79 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. Available 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_manuels/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>.