FIRMWARE AND SOFTWARE FOR AN INTEGRATED ELECTROPHYSIOLOGY DATA ACQUISITION AND STIMULATION SYSTEM

Kyle D. Batzer, M.S.E

Western Michigan University, 2013

This thesis describes the firmware and software design for a prototype electrophysiology experimentation system. The overall system, built on the work of previous students and developed with fellow graduate student Mr. Donovan Squires, provides eight channels of acquisition and four channel of arbitrary waveform generation for stimulation of biological systems. In order to show the performance of the system, a common electrophysiology experiment was performed on the giant axon of an earthworm and the results were compared to previously validated systems. The developed system is intended to support future work at the Neurobiology Engineering Laboratory at Western Michigan University.

The user of the system is provided high-level control, experimentation scripting, and data visualization through use of a custom PC application. Real-time operations, such as data capture using an analog-to-digital converter and stimulation waveform output to a digital-to-analog converter, are implemented with a field-programmable gate array (FPGA). Domain specific support for -10mV to 10mV acquisition levels and differential waveform generation between -15V to 15V is provided via a custom printed circuit board when utilizing previously developed amplification and filtering circuitry.

FIRMWARE AND SOFTWARE FOR AN INTEGRATED ELECTROPHYSIOLOGY DATA ACQUISITION AND STIMULATION SYSTEM

by

Kyle D. Batzer

A thesis submitted to the Graduate College
in partial fulfillment of the requirements
for the Degree of Master of Science in Engineering (Computer)
Electrical and Computer Engineering
Western Michigan University
December 2013

Thesis Committee:

Damon A. Miller, Ph.D., Chair
Bradley J. Bazuin, Ph.D.
Frank L. Severance, Ph.D.

ACKNOWLEDGMENTS

I would like to thank Dr. Miller, Dr. Severance, and Dr. Bazuin for their guidance and support throughout the project, even well in advance of my related senior design team project. I am also grateful for the support and contributions of doctoral students Michael Ellinger and John Stahl.

I thank Donovan Squires for his work on the Data Acquisition and Stimulation System hardware. It was a pleasure working alongside him for much of the project and I am very happy with what we were able to accomplish.

I thank my wife Kayla for supporting me through the effort, being patient with the long hours, and giving me a solid foundation on which to stand.

Kyle D. Batzer

FOREWORD

This thesis and the companion Master of Science in Electrical Engineering thesis by Mr. Donovan Squires describe a complete hardware and software instrumentation system for conducting electrophysiology experiments. The developed system features multiple channels that can be used for measurement and stimulation of biological electrical activity. These theses represent the culmination of many previous projects in the Western Michigan University Neurobiology Engineering Laboratory. This foreword provides a brief history of those previous projects to provide context for these two theses.

The original motivation for this line of research was the availability of microelectrode arrays (MEAs) from companies such as Multi Channel Systems (www.multichannelsystems.com). MEAs are essentially culture dishes with an array of implanted electrodes that enable monitoring and stimulation of neuron and neuronal network electrical activity. Reference [1] provides an excellent overview of studying learning in neuronal networks using MEAs. I was first introduced to MEAs via an article provided by Dr. John Gesink, Emeritus Chair of the WMU Department of Electrical Engineering. My first steps into this area included submission of an unfunded grant *From Artificial to Naturally Intelligent Systems: Computing with Neuron Cell Cultures* and auditing Dr. John Jellies' outstanding Advanced Neurobiology course in the Spring 2006 semester.

After discussions with Dr. Frank Severance, we decided to start a new laboratory to conduct research using MEAs, and I subsequently visited with Mr. Alex Cadotte at Dr. Thomas DeMarse's lab at the University of Florida to learn more about these devices, including efforts to use neuronal networks as an intelligent closed loop controller [2, 3].

A WMU Faculty Research and Creative Activities Support Fund grant, myself, Dr. Gesink, Dr. Severance, and the WMU Department of Electrical and Computer Engineering provided funds for cell culture equipment and supplies and instrumentation components. The WMU College of Engineering and Applied Sciences configured a laboratory for cell culture work and the Neurobiology Engineering Laboratory was born. Graduate student Mr. Michael Ellinger led the challenging effort to successfully culture the first cells in the lab in Summer 2008. Establishing a cell culturing capability relied on help from many people, most notably Biological Sciences graduate student Sr. John-Mary Vianney and her advisor Dr. John Spitsbergen.

In 2007, two senior design groups [4, 5] developed an initial design for the instrumentation system based on the research literature (including the key references [6, 7]) and a commercial system from Multi Channel Systems. In particular, [6] identified challenges and solutions with recording and stimulating using the same electrode. A third senior design group [8] continued this effort, breadboarding a single analog input channel and an associated digital control circuit. Mr. John Stahl conducted research on the noise characteristics of the analog input channel, building a complete analog/digital two channel prototype [9]. Mr. Stahl then designed a printed circuit board implementation for two analog input channels. A fourth senior design group [10] made significant progress on both the instrumentation system hardware and software, including development of a printed circuit board implementation of two analog input channels with the associated digital control circuit. A fifth senior design group [11] worked on this system with a focus of using an FPGA based solution for data acquisition and control via a computer. Mr. Squires, working with Mr. Stahl, built on these previous accomplishments to produce

the instrumentation design presented in this thesis. His design features a modular system consisting of a “mother board" into which up to eight analog input channel cards (designed by Mr. Stahl) can be inserted. Mr. Batzer, leveraging his previous work as a member of the fourth senior design group, completed the design of an FPGA based solution for data acquisition and control of the instrumentation system via a computer, including a scripting language, as his thesis. A key laboratory accomplishment was using the developed instrumentation system to conduct a standard experiment that measures earthworm giant axon nerve impulses [9, 12, 13, 14] as described in this thesis.

I thank the many students that have worked in the lab over the years. I also thank my colleagues for their unselfish help in these projects, and in particular Dr. Frank Severance, who co-directs the lab, and Drs. Bazuin and Gesink. Finally, I thank the WMU College of Engineering and Applied Sciences, the Department of Electrical and Computer Engineering, the Lee Honors College, the Office of the Vice President for Research, and the NASA Michigan Space Grant Consortium for supporting the work of the Neurobiology Engineering Laboratory.

Dr. Damon A. Miller
Associate Professor of Electrical and Computer Engineering
Western Michigan University
Co-Director, WMU Neurobiology Engineering Laboratory
Kalamazoo, MI
May 2013

TABLE OF CONTENTS

Table of Contents - Continued

CHAPTER

Table of Contents – Continued

CHAPTER

Table of Contents - Continued

LIST OF FIGURES

List of Figures - Continued

LIST OF TABLES

List of Tables - Continued

1. Introduction[1]

An Integrated Electrophysiology Data Acquisition and Stimulation System to support electrophysiology research has been developed by building on previous work at WMU [4, 8, 9, 10, 14], studying commercial systems [16], and reviewing the research literature [3, 6, 17]. The developed prototype provides a real-time platform for measurement and stimulation of biological electrical activity and a PC application for controlling the real-time platform and visualizing cellular activity. The system can accommodate up to eight measurement channels and four stimulation channels, and the design can be expanded for up to 60 channels to support future research at the Western Michigan University (WMU) Neurobiology Engineering Laboratory.

The developed prototype is viable for a wide array of electrophysiology experiments, completely fulfilling the instrumentation needs of [12, 13, 18] and partially fulfilling the instrumentation needs of [1, 2, 3]. In particular, a standard electrophysiology experiment was performed on earthworm giant axon potentials to validate system functionality. The prototype is also intended for studying software and hardware principles required for performing research using cells cultured on a Microelectrode Array (MEA), e.g. [16]. A cell culture protocol has been developed [14] and previous work on such a system [4, 8, 10], including low noise amplification [9], has been completed. Initial analytical algorithms have also been developed [14].

[1] This section was co-authored with Donovan Squires, [15].

2. Specifications[2]

The Data Acquisition and Stimulation System (DASS) is expected to provide the following functionality:

1. Provide a platform for performing electrophysiology experiments with earthworms as described in [12, 13]
    a. Produce a voltage-controlled square wave stimulation pulse with widths from 0.01ms to 100ms and amplitudes from 0.1V to 10V
    b. Produce single stimulation pulses or multiple pulses at rates from 1Hz to 10Hz
    c. Provide at least one differential recording channel
    d. Record an action potential voltage from the time of a stimulation pulse for a minimum duration of 20ms
    e. Plot the recorded voltage
    f. Store the recorded voltage to a non-proprietary, standard file format
2. Provide a platform for stimulation and recording of neuron cell culture electrical activity via MEA electrodes
    a. Provide at least four recording channels
    b. Store data from recording channels continuously
    c. Provide at least four voltage-controlled arbitrary stimulation channels
    d. Output single-ended stimulation signals on recording electrodes and add culture voltage offset to the stimulation signal

[2] This section was co-authored with Donovan Squires, [15].

e. Provide an interface that can specify stimulation waveforms, locations, and intervals that can be updated based on data from the recording electrodes

3. Utilize Low-Noise Amplifier described in [9]
    a. Connect to each Low-Noise Amplifier channel with a PCI-Express card edge connector
    b. Provide ±7V to ±15V analog voltage supplies and ground via the card edge connector
    c. Provide ability to independently switch four digital inputs for each channel, $0_{IH} = 0.8V$ and $1_{IL} = 2.4V$
    d. Route differential analog input to the card edge connector for each channel
    e. Convert the 20Hz to 14.6kHz analog output signal [9] to digital samples
    f. Route a single-ended stimulation signal to each channel

3. Terminology

Data Acquisition and Stimulation System (DASS)[3]: System intended for electrophysiology experiments described in this thesis and corresponding thesis [15]. The DASS includes all hardware, software, and firmware.

Real Time System Controller (RTSC)[3]: Subsystem that implements the real-time functions of the system. It consists of a Digilent® Nexys2™ development board [19] with custom firmware.

Electrophysiology Interface[3]: Subsystem that provides the RTSC with an interface to biological systems, as described in [15].

Preamp[3]: Low noise instrumentation amplifier with stimulation dc bias addition for MEA experiments, described in [9,15].

Data Acquisition and Stimulation Control Center (DASCC)[3]: PC application, described in this thesis, for controlling and transferring data to and from the RTSC.

VHDL: Very High Speed Integrated Circuits (VHSIC) Hardware Description Language

Module: Used in the RTSC FPGA Configuration section of this document to describe a functional component within the VHDL hierarchy.

[3] Definition co-authored with Donovan Squires [15].

4. System Design: Software and Firmware

The developed Data Acquisition and Stimulation System (DASS) provides multi-channel data acquisition and arbitrary waveform generation for performing electrophysiology experiments. Figure 1 provides a top-level overview of the major components.

The first major component of the system is a standard Windows PC. A custom PC application was developed that provides a user interface for controlling the Real Time System Controller (RTSC) over an RS232 interface. Acquired data is captured and logged over a USB interface. Once captured to file, acquired data can be graphed and exported to .csv format for analysis in other tools (e.g. Matlab, Excel).

The second major component is the Real Time System Controller (RTSC). It is a Digilent® Nexys 2™ FPGA development board [19] with custom FPGA firmware that provides real-time control of four digital-to-analog converter (DAC) channels and eight analog-to-digital converter (ADC) channels for waveform generation and acquisition of biological signals. Unique arbitrary waveforms can be loaded from the PC application into the SDRAM for each of the four channels and any combination of the channels output simultaneously.

The third major component is the Electrophysiology Interface board developed in [15] to provide the DAC, ADC, eight PreAmp interfaces, and differential output amplification. The PreAmp and differential output provide signal levels required for work with biological systems.

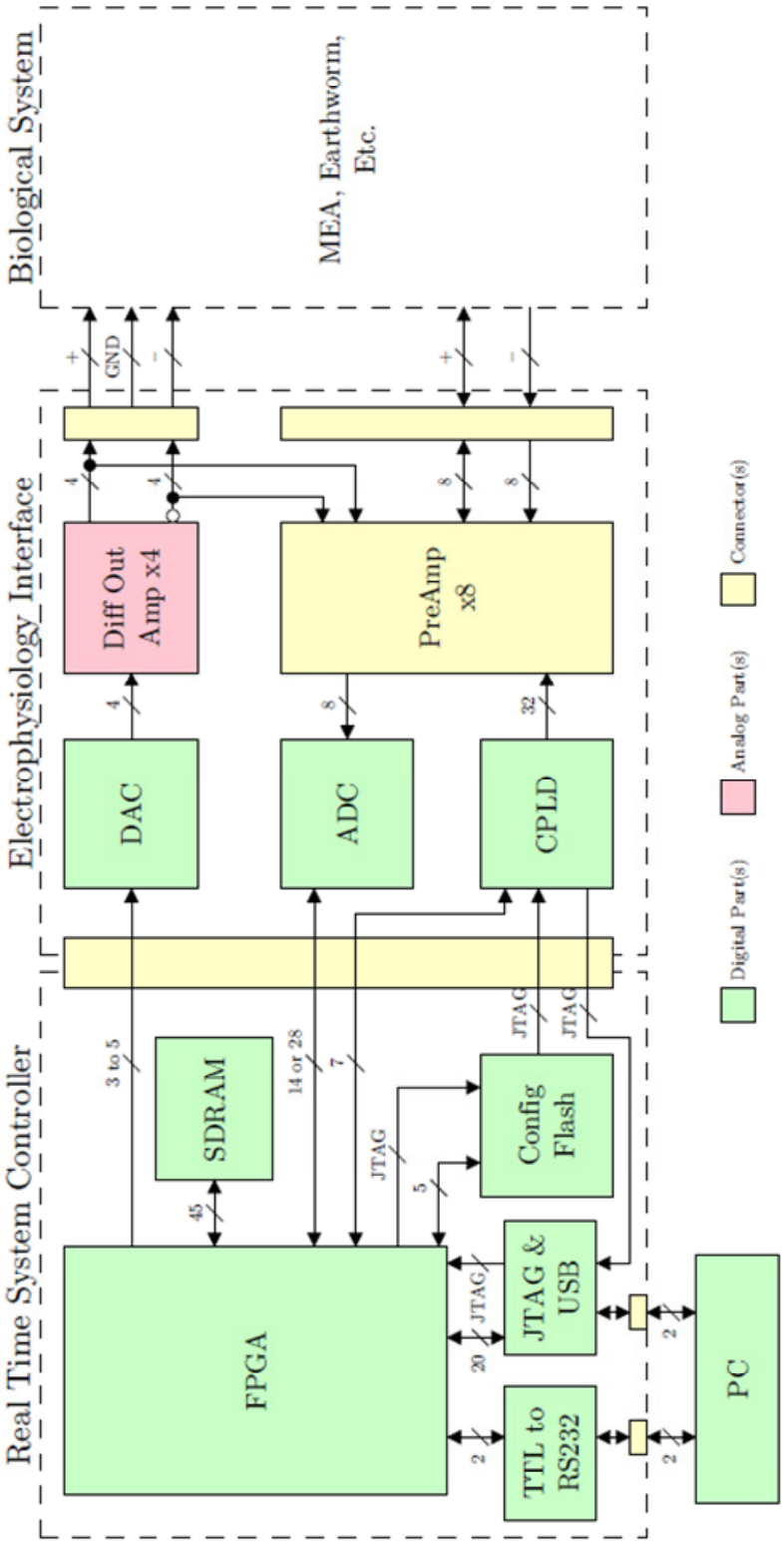

Figure 1: The Data Acquisition and Stimulation System. Figure by D. Squires [15].

4.1. Data Flow Analysis

Acquisition

One of the primary features of the Data Acquisition and Stimulation System is to sample multiple analog channels simultaneously and route that data to file on a PC for analysis. Figure 2 provides an overview of the acquisition data flow. Note that due to the non-real time characteristics of a Windows PC, it is necessary to buffer data where possible to limit susceptibility to times when the operating system is servicing another process.

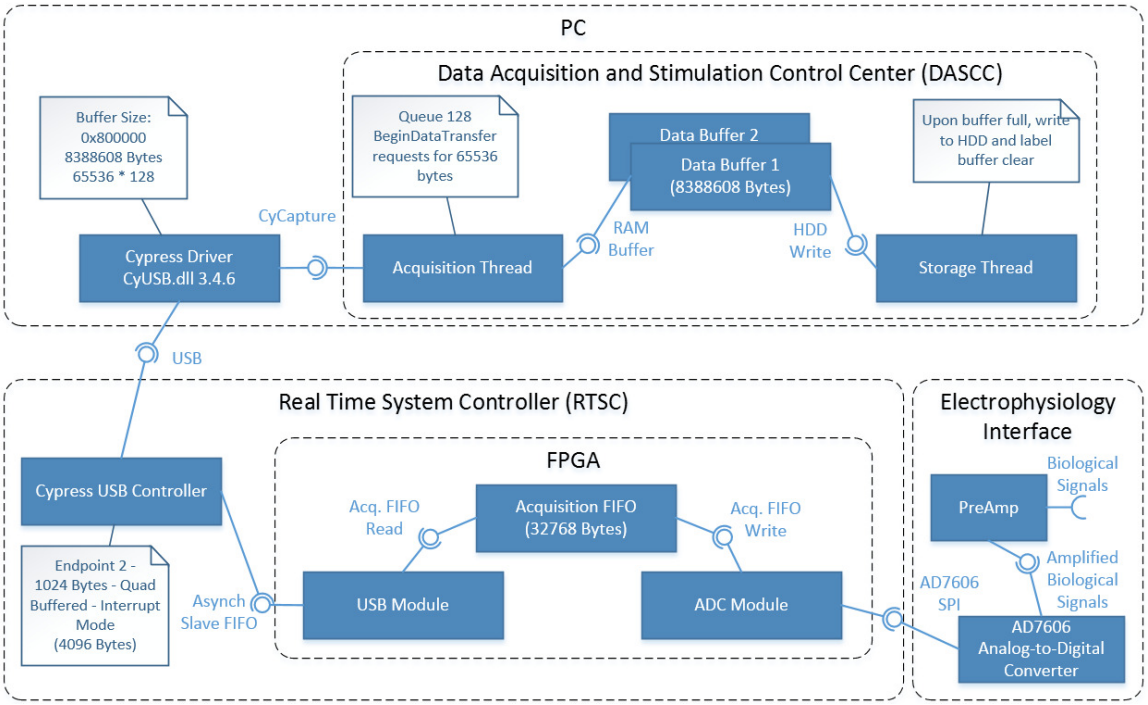

Figure 2: Acquisition Data Flow

The biological signal of interest is connected to the PreAmp. The PreAmp provides biological signal amplification (1000V/V gain) and filtering (passband between 20Hz and 14.6 kHz). The PreAmp accepts inputs in the range of -10mV to 10mV, which will result in -10V to 10V after applying the 1000V/V gain. The amplified biological

signal is sampled by the AD7606 at a 44.1 kHz rate with 16-bit resolution. The raw ADC data bandwidth over the AD7606 SPI interface is 0.7056 Mbytes/s, calculated as follows:

$$Raw\ ADC\ Data = \frac{8\ channels * 2\ bytes/channel * 44100Hz}{1000000} = 0.7056\ \text{Mbytes}/s.$$

The ADC Module takes this raw ADC data and packetizes it, adding meta-data to assist in detecting and validating the acquired data one it reaches the DASCC. This additional overhead results in 32 bytes for the eight channels of acquired data and a packetized ADC data bandwidth of 1.4112 Mbytes/s, calculated as follows:

$$Packetized\ ADC\ Data = \frac{32\ bytes * 44100Hz}{1000000} = 1.4112\ \text{Mbytes}/s.$$

The ADC module then passes the data into the Acquisition FIFO using the Acquisition FIFO Write interface, one byte at a time. The Acquisition FIFO provides a total of 32768 bytes of buffering for acquired data. From here the USB Module will read data from the Acquisition FIFO using the Acquisition FIFO Read interface and write it to the Cypress USB Controller using the Asynch Slave FIFO interface.

The Cypress USB Controller provides four FIFOs, each able to hold 1024 bytes. Upon filling one of these FIFOs, a USB interrupt mode transaction is started and the FIFOs contents are transmitted to the PC while another FIFO is used to capture incoming data over the Asynch Slave FIFO interface. Interrupt mode USB transfers have a guaranteed chance to be serviced every USB micro-frame (125 micro-seconds). The theoretical throughput for high-speed USB 2.0 interrupt transactions of 8.192 Mbytes/s is sufficient for handling the packetized ADC Data rate of 1.4112 Mbytes/s and is calculated as follows:

$$Interrupt\ Mode\ Bandwidth = 1024\ bytes * \frac{1}{125\ microseconds} = 8.192\ Mbytes/s.$$

On the PC, received USB data is initially buffered by the Cypress driver. The maximum buffer size has been increased to 8388608 bytes (65536 bytes * 128), to support the maximum requested bytes by the DASCC (128 asynchronous transfers of 65536 bytes).

The DASCC has an acquisition thread, dual 8388608 byte memory buffers, and a storage thread. The acquisition thread queues up 128 reads from the Cypress driver of 65536 bytes and then waits for the Cypress driver to signal that the data is available. When available, the 65536 bytes are read into a memory buffer and another read is queued, keeping the number of queued reads at 128. The memory buffer is filled until it reaches 8388608 bytes, and then the storage thread writes the buffer to hard drive and the acquisition thread begins to fill the second memory buffer. This dual buffer concept is generally referred to as a ping-pong buffer.

Stimulation

Another major feature of the Data Acquisition and Stimulation System is multi-channel arbitrary waveform generation. Figure 3 provides an overview of the stimulation data flow. Note that this includes both loading the waveform from a file on the PC to RAM on the RTSC and outputting a differential arbitrary waveform.

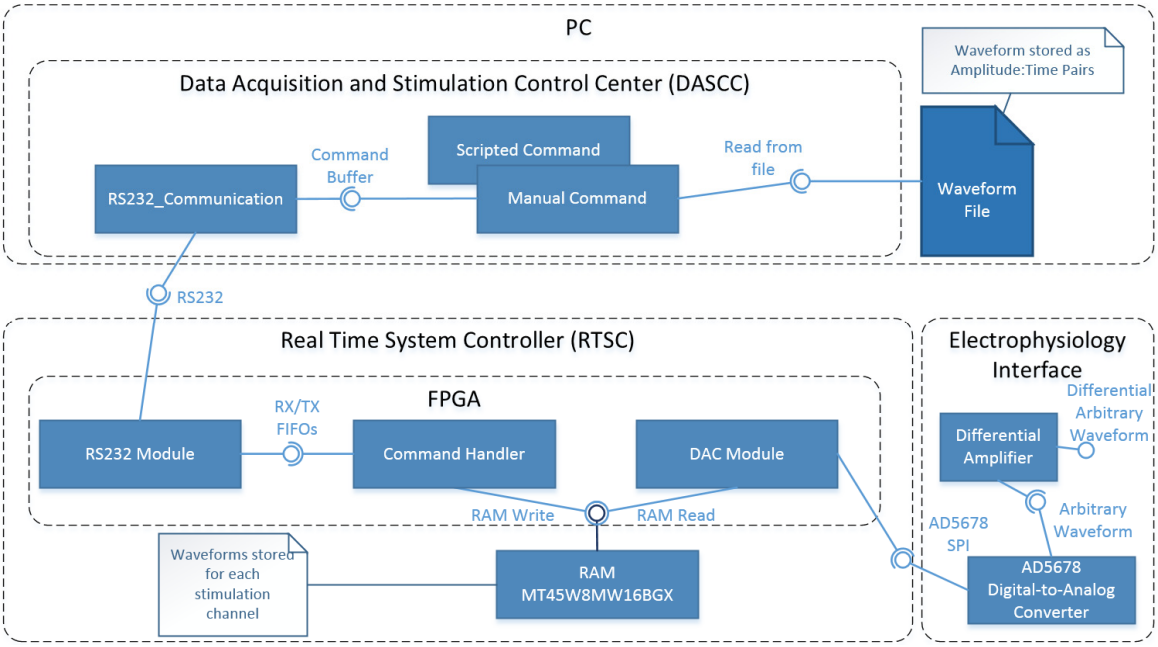

Figure 3: Stimulation Data Flow

Waveform descriptions must first be loaded into RAM on the RTSC before the stimulation channels can output the described arbitrary waveform. This process begins with reading in the waveform file with the DASCC and sending it out RS232, either from a scripted or manual command. The waveform is stored as amplitude to set and amount of time to maintain the amplitude. For more details on the amplitude:time pairs see the DAC Channel Module Implementation section of this document.

On the FPGA, the RS232 Module receives the waveform description via the Set Waveform reply RTSC API command. The Command Handler reads the command from

the RX FIFO and writes the waveform description to the respective channel RAM location (see Table 7: Memory Locations for Stimulation Waveforms). The Command Handler then places the Set Waveform reply RTSC API command into the TX FIFO, from where the RS232 Module transmits the reply back to the DASCC.

Once a waveform has been loaded for each desired channel, the RTSC can be commanded to begin outputting arbitrary waveforms on any combination of the four supported DAC channels simultaneously. The DAC Module contains separate logic for each DAC channel, each of which can be idle, outputting its waveform once (single stimulation), or outputting its waveform repeatedly (multi-stimulation). Upon the defined time for a sample elapsing, the amplitude is set for the next sample via command over the AD5678 SPI interface to the AD5678. From there, the DAC output passes through a differential amplifier, resulting in a single ended output of -7.5V to 7.5V or a differential output of -15V to 15V, depending on physical connections (see [15] for more details).

4.2. RTSC FPGA Configuration

The FPGA on the RTSC is used to implement the required real time system operation. Figure 4 a top-level view of the FPGA configuration.

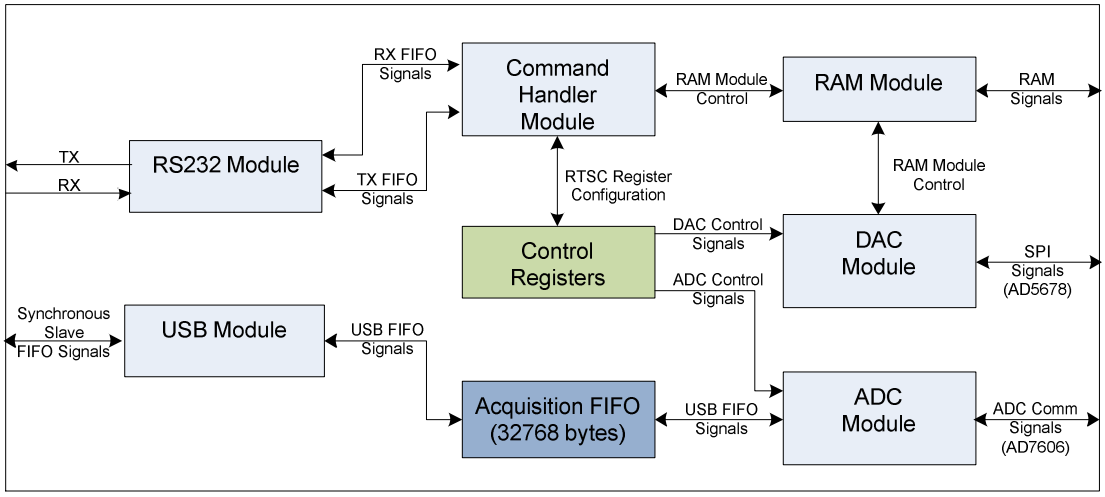

Figure 4: FPGA Top-Level Configuration

4.2.1. Control Registers

The state of the FPGA configuration is contained within the Control Registers. Access to the Control Registers is provided via the RS232 Module using the set of commands described in the RTSC Application Programming Interface (API) section of this document.

4.2.1.1. Channel Configuration Register

The Channel Configuration Register, shown in Table 1, provides the settings for a single Preamp board [9]. The Electrophysiology Interface board [15] can accept up to eight pre-amplifiers, each of which can be connected to a recording/stimulation electrode. Bits 0-3 control the switches for a Preamp board. Bit 4 is an identifier for whether the channel is set for stimulation or acquisition. Bits 5-7 are reserved.

At this time, the FPGA does not set the Preamps according to the Channel Configuration Registers. When developed, the FPGA will command each of the Preamps through an interface provided by the CPLD on the Electrophysiology Interface board.

Channel Configuration Registers (one per channel)							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
reserved			Stim-1/Acq-0	sw4	sw3	sw2	sw1

Table 1: Channel Configuration Registers

4.2.1.2. Stimulation Register

The Stimulation Register contains a channel mask that corresponds to each of the DAC channels provided by the Electrophysiology Interface board [15]. When the "channel active" bit is set for a given channel the Digital-to-Analog Converter module commands the DAC to update its voltage output based on Amplitude:Time pairs stored in DRAM on the RTSC. Further details on the Amplitude:Time pairs can be found in the DAC Channel Module Implementation section of this document.

Stimulation Register (single 8 bit register, expand to support more channels)							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
reserved				ch4_active	ch3_active	ch2_active	ch1_active

Table 2: Stimulation Register

4.2.1.3. Acquisition Register

The Acquisition Register contains a channel mask commanding the channels to acquire data from the ADC. The current use of this register and the ADC module is to acquire on all channels or none at all. By setting bit 0 all channels will acquire data and by clearing bit 0 all channels will stop acquiring data.

Acquisition Register (single 8 bit register, expand to support more channels)							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
reserved							acq_active

Table 3: Acquisition Register

4.2.2. Analog-to-Digital Converter Module

The Analog-to-Digital Converter Module is used to drive the AD7606 on the Electrophysiology Interface board. When bit 0 of the Acquisition Register is set the module acquires digitized readings from the AD7606 and routes them to the USB FIFO.

4.2.2.1. AD7606

The Analog-to-Digital converter (ADC) selected for this project the Analog Device AD7606 [20]. It provides eight channels with 16-bit resolution and can be read with a SPI-like interface as shown in Figure 5. A read operation is initiated by pulsing low convStA and convStB. When Busy signals the conversion is complete CS is dropped low and data is clocked in on doutA and doutB.

Note that convStA, convStB, and Busy are not shown in Figure 5. For further information see [20].

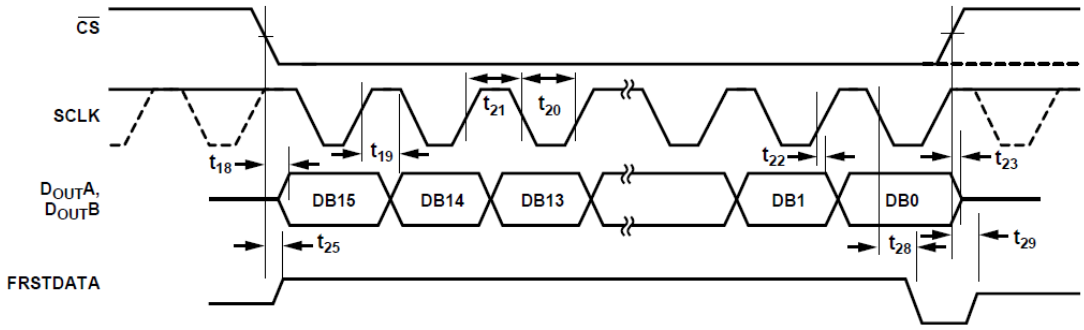

Figure 5: AD7606 Serial Read Operation (Figure 6 from [20])

4.2.2.2. ADC Module Implementation

The ADC Module triggers ADC reads, packetizing read data, and writing packets to the USB FIFO. Figure 6 provides a block diagram of the ADC Module. Table 4 provides an overview of all IO for the module. Figure 7 provides an overview of the

sequence of events performed by the ADC Module. Note that Table 4 groups signals into multiple headings (USB FIFO, ADC Comm, and ADC Control) and that Figure 6 uses these groupings for defining connections.

Signal Name	Description
USB FIFO Signals	
FIFO_DIN	8-bit parallel data bus for writing to the USB FIFO
FIFO_WR_EN	Pulsing signal writes data on FIFO_DIN to USB FIFO (Active High)
FIFO_WR_CLK	Clock driving the operation of the FIFO write side
FIFO_PROG_FULL	Flag indicating the FIFO has reached a programmable threshold (currently 32256). This is used to ensure that there is enough room for a full packet to be written to the FIFO before starting.
ADC Communication Signals (AD7606)	
CS	Chip Select. Frames data transfer (Active Low)
sCLK	Serial Clock for data transfers. Data clocked in on rising edge.
doutA	Serial data out A. Channels 1, 2, 3, and 4 are clocked in on doutA.
doutB	Serial data out B. Channels 5, 6, 7, and 8 are clocked in on doutB.
ADC Control Signals (AD7606)	
adcRANGE	Analog input range select. 0: +- 5 V 1: +- 10 V (RTSC Setting)
adcRESET	Reset signal (Active High). Upon initialization reset is held high for 30 ms.
adcSTDBY	Standby Mode (Active Low). For RTSC, adcSTDY = 1 (unused)
convStA	Conversion Start A (Active Low). Pulse to initiate conversions on analog input channels. ConvStA and ConvStB are tied together in the RTSC to allow synchronized sampling.
convStB	Conversion Start B (Active Low). Pulse to initiate conversions on analog input channels. ConvStA and ConvStB are tied together in the RTSC to allow synchronized sampling.
ovrSAMPLE	Used to select the oversampling ratio. Set to 000 to disable oversampling.
refSEL	0: Internal reference disabled 1: Internal reference used (RTSC Setting)
serSEL	0: Parallel Interface selected 1: Serial Interface selected (RTSC Setting)
Busy	Indicates to the RTSC when the conversion has started (set high by the AD7606) and when it completes (set low by the AD7606).

Table 4: Analog-to-Digital Converter Module Signals

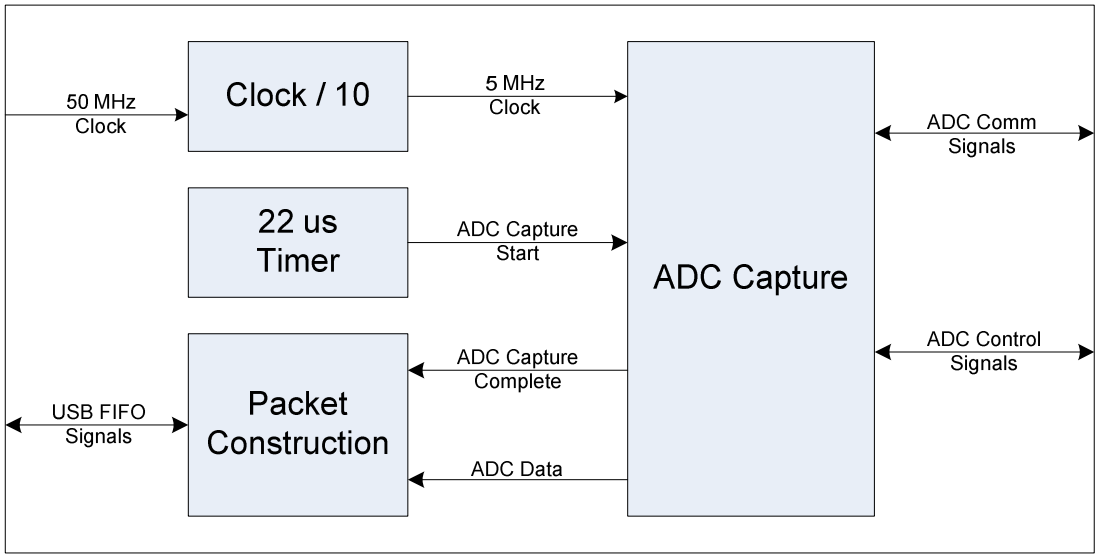

Figure 6: ADC Module Block Diagram

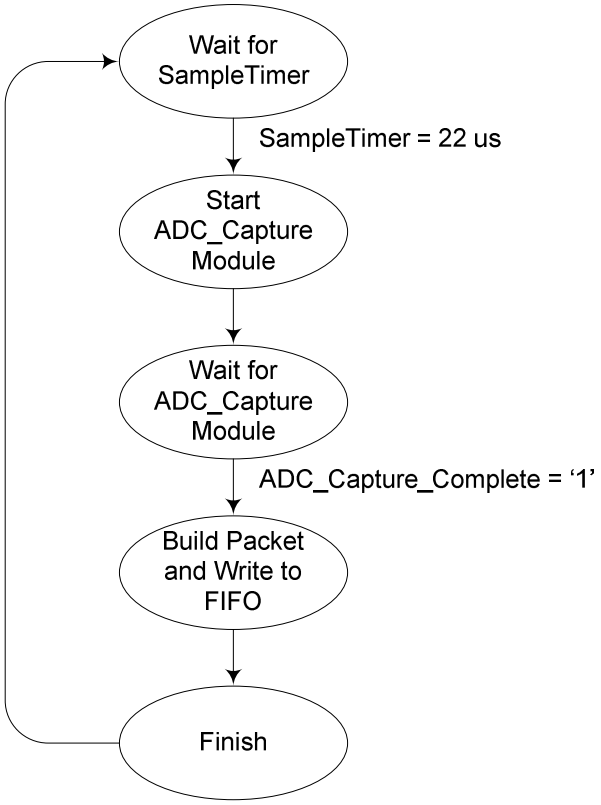

Figure 7: ADC Module Flow Chart

The ADC Module uses a 22 us timer to send a start signal to the ADC Capture Module, which initiates a conversion and reads the resulting sampled data. The ADC Module idles until the ADC Capture Module has completed its task. The captured data is then packetized for transmission and written to the USB FIFO.

Table 5 shows the contents of the 32 byte packet. The Start Flag (0xA55A) provides an easy to identify indicator for the start of the packet. It is followed by a 32-bit time counter, which indicates a time between 0 and 85.9 seconds assuming it is incremented with a 50 MHz (20 ns) clock. This provides a relative offset for data analysis. Each channel has one byte for its channel number and 2 bytes for its 16-bit ADC reading. The last two bytes of the packet are a 16-bit checksum for message validation.

Acquisition Packet Structure		
Byte #	**Field**	**Value**
1-2	Start Flag	0xA55A
3-6	Time Offset	32-bit time counter
7	Channel 1 Indicator	0x01
8-9	Channel 1 Data	16-bit Channel 1 ADC Data
10	Channel 2 Indicator	0x02
11-12	Channel 2 Data	16-bit Channel 2 ADC Data
13	Channel 3 Indicator	0x03
14-15	Channel 3 Data	16-bit Channel 3 ADC Data
16	Channel 4 Indicator	0x04
17-18	Channel 4 Data	16-bit Channel 4 ADC Data
19	Channel 5 Indicator	0x05
20-21	Channel 5 Data	16-bit Channel 5 ADC Data
22	Channel 6 Indicator	0x06
23-24	Channel 6 Data	16-bit Channel 6 ADC Data
25	Channel 7 Indicator	0x07
26-27	Channel 7 Data	16-bit Channel 7 ADC Data
28	Channel 8 Indicator	0x08
29-30	Channel 8 Data	16-bit Channel 8 ADC Data
31-32	Checksum	

Table 5: Acquisition Packet Structure

4.2.2.3. ADC Capture Module Implementation

The ADC Capture module drives the AD7606 signals.

Figure 8 provides a flow chart showing the functional operation of the ADC Capture module. Upon module reset and startup the ADC Capture Module holds the ADC7606 reset signal high. After 30 ms, reset is cleared and the module sits idle until it receives the ADC_Capture_Start signal from the ADC Module 22 us timer. It then pulses low convStA and convStB to command the AD7606 to start a conversion for all channels. The busy signal is then monitored to determine when the AD7606 conversion has completed. Upon completion, CS is dropped low and the data for each channel is clocked in on doutA and doutB. Channel data is placed into registers accessible to the ADC Module and the transaction complete signal is pulsed high to notify the ADC Module.

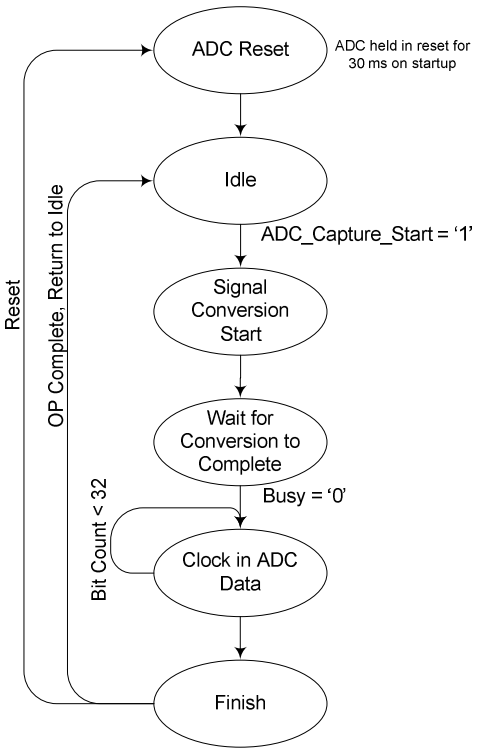

Figure 8: ADC Capture Module Flow Chart

4.2.3. Digital-to-Analog Converter Module

The Digital-to-Analog Converter (DAC) Module is used to drive the AD5678 on the Electrophysiology Interface board. Amplitude-Time pair waveforms are sent from the PC to the RS232 module and are saved to DRAM on the Real Time System Controller (RTSC). Bus arbitration on both the memory interface and the 3-wire communication interface to the AD5678 allows four unique waveforms to be output by the AD5678 simultaneously.

4.2.3.1. AD5678

The AD5678 is a digital-to-analog converter that provides four 16-bit channels and four 12-bit channels. For the Data Acquisition and Stimulation System only the four 16-bit channels are used. Each of these channels can output between 0V to 5V. The AD5678 supports a 3-wire synchronous serial communication interface shown in Figure 9. A write operation consists of the SYNC signal being driven low, 32-bits of data being clocked out, and the SYNC signal returning high. LDAC is pulled low in hardware, resulting in any update to the AD5678 output channels occurring immediately after the serial write is complete.

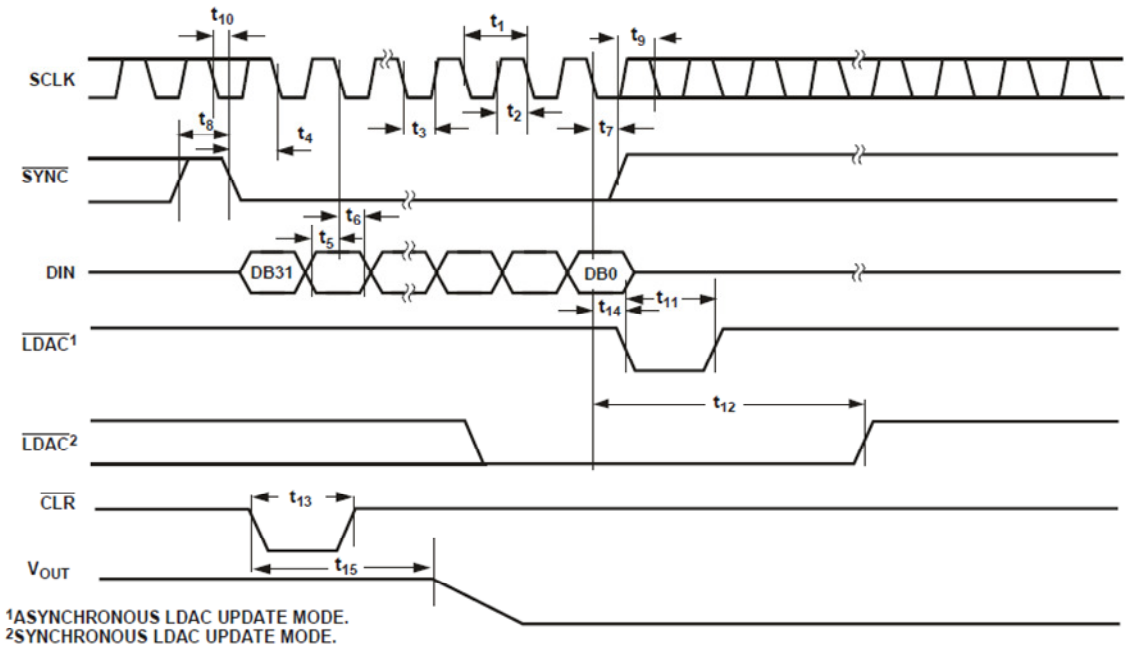

Figure 9: AD5678 Serial Write (Figure 2 from [21])

4.2.3.2. DAC Module Implementation

Figure 10 shows the structure of the DAC Module implementation. Table 6 provides a brief description of each of the DAC Module IO signals. Note that Table 6 groups signals into multiple headings (Control, SPI, RAM, and RAM Arbiter) and that Figure 10 uses these groupings for defining connections.

Signal Name	Description
clk	Main clock for module (50 MHz)
reset	Module reset (active low). All registers return to startup values.
Control Signals	
Stimulation	Stimulation register contents (active high). See Table 2. Channel bit pulse active results in single stimulation waveform. Channel bit held active results in multi-stimulation.
Stim_Active	Flag indicating channel is currently outputting a stimulation waveform.
Init_Complete	Flag indicating DAC_Init module has completed AD5678 configuration and the AD5678 is ready for stimulation output.
SPI Signals (AD5678)	
SPI_CLK	Clock for SPI bus (5 MHz)
CS	Frame indicator for AD5678 SPI bus (Active Low).
MOSI	Output data for SPI bus to AD5678. (Master Out Slave In)
RAM Signals	
RAM_Start_Op	Trigger for the RAM module to start a memory operation.
RAM_Op_Done	Flag indicating the memory operation has completed.
RAM_WE	Write enable indicator for the RAM module. 0: Write Operation 1: Read Operation
RAM_ADDR	Memory address for the commanded operation.
RAM_DOUT	16-bit output data bus from RAM used for read operations.
RAM_DIN	16-bit input data bus to RAM used for write operations.
RAM Arbiter Signals	
RAM_Bus_Request	Request signal for accessing memory interface. Bit 0: DAC_Channel 1 module Bus Request Bit 1: DAC_Channel 2 module Bus Request Bit 2: DAC_Channel 3 module Bus Request Bit 3: DAC_Channel 4 module Bus Request
RAM_Bus_Busy	Flag indicating memory interface is currently in use.
RAM_Bus_Grant	Signal granting access to memory interface received in response to a bus request. Bit 0: DAC_Channel 1 module Bus Grant Bit 1: DAC_Channel 2 module Bus Grant Bit 2: DAC_Channel 3 module Bus Grant Bit 3: DAC_Channel 4 module Bus Grant

Table 6: DAC Module Signals

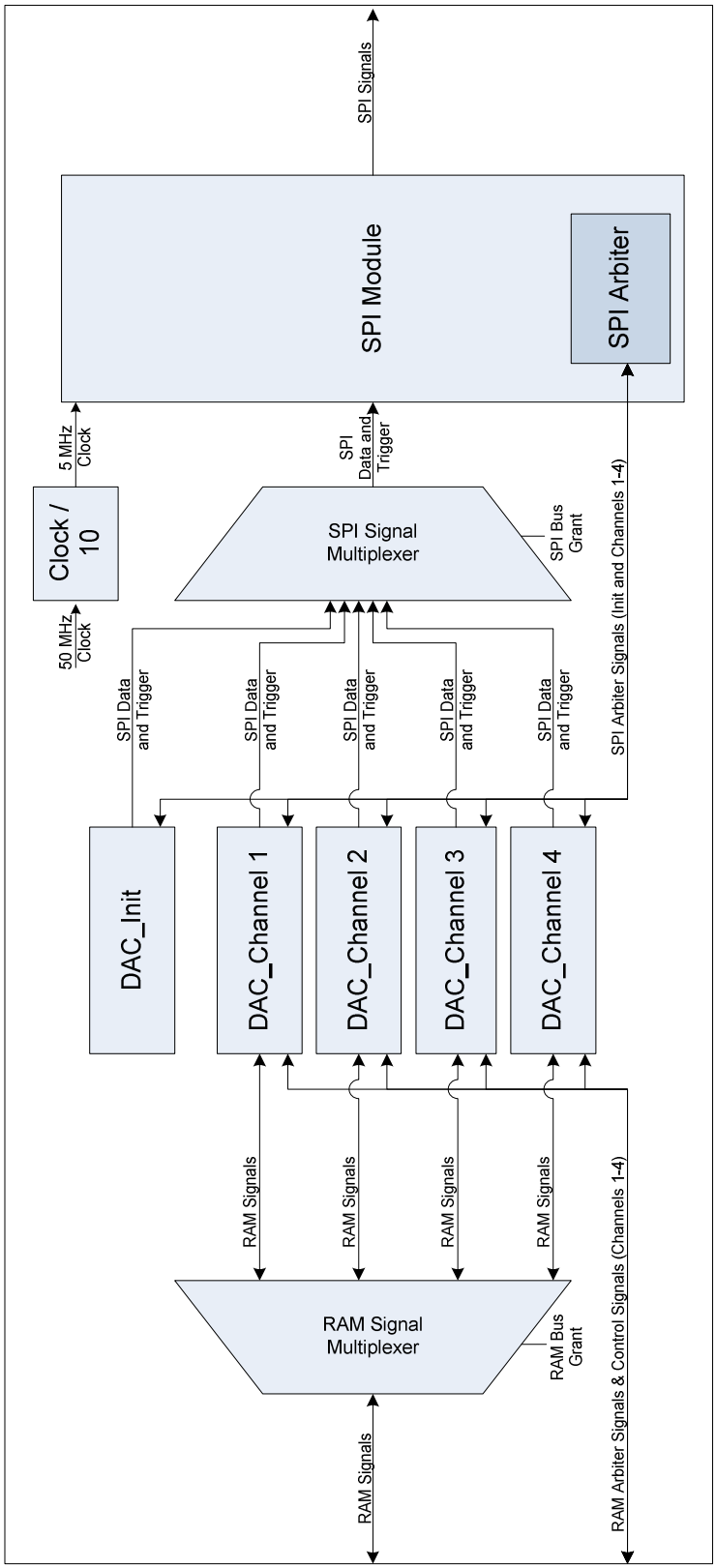

Figure 10: Block diagram of the DAC module IO and internal modules

4.2.3.3. DAC Init Module Implementation

On reset the DAC_Init state machine performs initialization of the AD5678 registers. The Power-On Reset command sets all DAC channel outputs to 0V. The Internal Reference Register is then configured to turn on and use the internal voltage reference of 2.5V. The LDAC register is configured to have all channels update immediately after receiving a new output command. The Clear Code Register is configured to set the output register of each DAC to zero if CLR is driven low; however, in the prototype this signal is simply pulled down and ignored. The output register for each of the DAC channels is then set to midscale, yielding 0V on the stimulation electrode connected to the Electrophysiology Interface board [15], and the DAC_Init module waits in its IDLE state until a reset command.

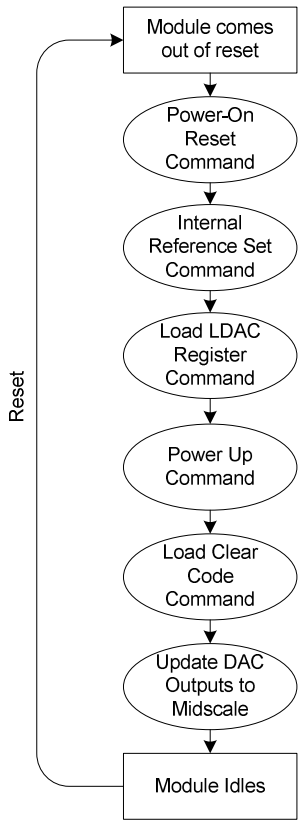

Figure 11: DAC_Init Flow Chart

4.2.3.4. DAC Channel Module Implementation

Each of the four supported DAC channels has its own separate module. Each channel has its own section of memory for storing waveforms as time-amplitude pairs. Time values are 16-bits, with each step accounting for 1 us of delay. Amplitude values are 16-bits, providing the value to update to the DAC output register for the given channel. Time values provide the amount of time to hold the corresponding amplitude value. Each of the four channels can output waveforms simultaneously due to bus arbitration logic for memory and SPI communication.

The DAC_Channel module supports single stimulation (sending the stored waveform once) and multi-stimulation (sending the stored waveform repeatedly). For single stimulation the "Stimulation" signal is set for a single clock cycle and for multi-stimulation the "Stimulation" signal is set and held high for as long as the repeating output is desired.

Upon setting the corresponding Stimulation Register bit (see Stimulation Register section of this document) of a DAC_Channel module, the number of samples stored in memory for the channel is read. From here, the first time-amplitude pair is read, SPI bus access requested, and the DAC output is updated. After waiting for the number of 1 us counts specified by the time value, the process is repeated for each time-amplitude pair. Upon reaching the last time-amplitude pair for a single stimulation, the DAC_Channel module returns to IDLE and waits for the next time "Stimulation" is set. During a multi-stimulation, the module returns to the first sample and loops through the entire set until "Stimulation" is no longer set.

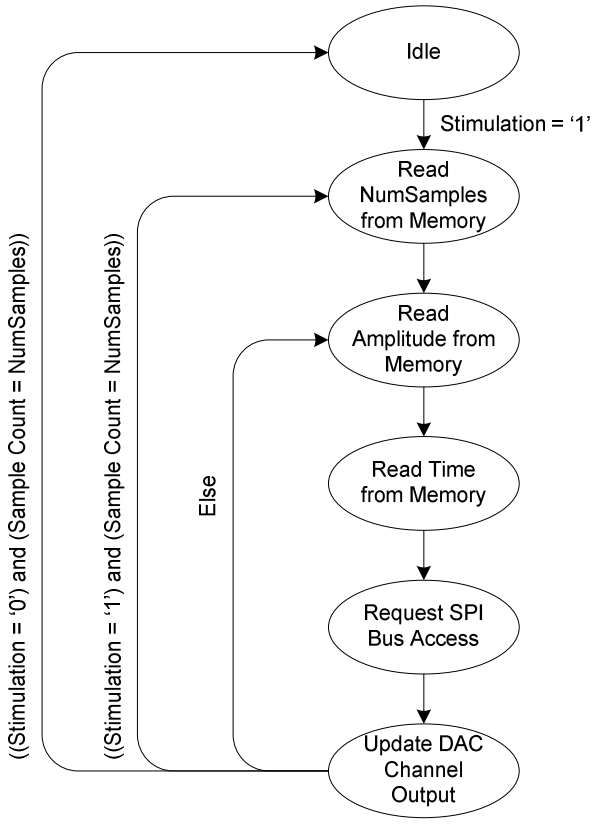

Figure 12: DAC_Channel Flow Chart

Table 7 provides the memory locations where waveforms are stored for each channel. Each channel is provided 0x1000 (4096) 8-bit memory locations. NumSamples is stored in the first location for each channel and from there samples are stored, each one taking 32-bits. This means each channel has a maximum number of samples per waveform of 1023 (4096 - 8-bit locations / 32-bit samples – 8-bit NumSamples value).

Memory Location (23-bit addresses, top 7 bits ignored)	Channel #	Content
0x0000	Channel 1	NumSamples
0x0001-0x0002	Channel 1	Sample 1 – Amplitude
0x0003-0x0004	Channel 1	Sample 1 - Time
…	…	…
0x1000	Channel 2	NumSamples
0x1001-0x1002	Channel 2	Sample 1 – Amplitude
0x1003-0x1004	Channel 2	Sample 1 - Time
…	…	…
0x2000	Channel 3	NumSamples
0x2001-0x2002	Channel 3	Sample 1 – Amplitude
0x2003-0x2004	Channel 3	Sample 1 - Time
…	…	…
0x3000	Channel 4	NumSamples
0x3001-0x3002	Channel 4	Sample 1 – Amplitude
0x3003-0x3004	Channel 4	Sample 1 - Time
…	…	…

Table 7: Memory Locations for Stimulation Waveforms

4.2.3.5. DAC SPI Module Implementation

The Real Time System Controller (RTSC) acts as the SPI master device and the AD5678 acts as the slave, meaning the RTSC drives “sclk”, “sync”, and “din”. The SPI Module is configured for 5MHz and only supports transmit (the AD5678 does not send information back to RTSC). The data to send is provided to the module and SPI_Start is pulsed high, beginning the transmission. Sync is driven low indicating the beginning of the transmission to the AD5678. The 32 data bits are then clocked out and the module returns to IDLE.

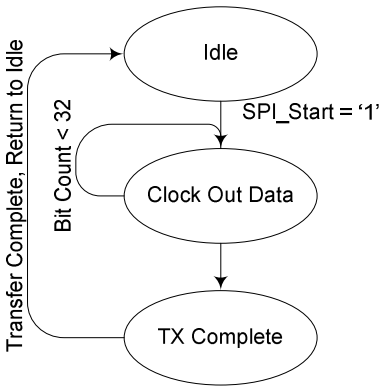

Figure 13: SPI Transmit Flow Chart

4.2.4. RS232 Module

The RS232 Module provides the communication interface to the PC application for command and control of the RTSC. Supported commands are described in the RTSC Application Programming Interface (API) section of this document. Received commands are placed in the RX FIFO, where they are processed by the Command Handler Module and an appropriate reply is placed in the TX FIFO and transmitted.

RS232 is an asynchronous, full-duplex protocol. The rate of transmission is known as the baud rate. For RTSC, 115200 baud is used, meaning each bit is 8.68 us wide. RTSC also uses standard 8 data bits, 1 stop bit, and no parity bit (N81). Figure 14 shows a standard N81 RS232 transaction (TTL logic levels). The data is high when idle, dropping low to indicate start of transmission. The 8 data bits are then transmitted least significant bit first. The transmission ends with the stop bit pulling the data line high.

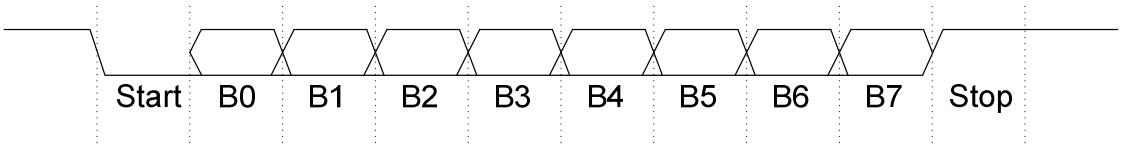

Figure 14: RS232 8-bit Transaction

4.2.4.1. RS232 Module Implementation

Figure 15 shows the structure of the RS232 Module implementation. Table 8 provides a brief description of each of the RS232 Module IO signals. Note that Table 8 groups signals into multiple headings (RS232, TX FIFO Write, RX FIFO Read, and Debug) and that Figure 15 uses these groupings for defining connections.

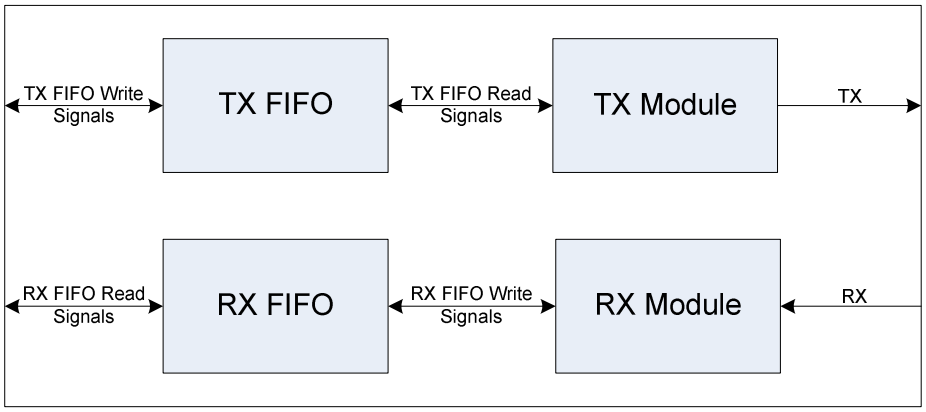

Figure 15: RS232 Block Diagram

Signal Name	Description
clk	Main clock for module (50 MHz)
reset	Module reset (active low). All registers return to startup values.
RS232 Signals	
TX	Asynchronous Serial Transmit (115200 baud)
RX	Asynchronous Serial Receive (115200 baud)
TX FIFO Write Signals	
TX_FIFO_WR_CLK	TX FIFO write clock.
TX_FIFO_DIN	8-bit data input to the TX FIFO.
TX_FIFO_WR_EN	TX FIFO Write Enable (Active High).
RX FIFO Read Signals	
RX_FIFO_RD_CLK	RX FIFO read clock.
RX_FIFO_DOUT	8-bit data output from the RX FIFO.
RX_FIFO_RD_EN	RX FIFO Read Enable (Active High).
RX_FIFO_EMPTY	Flag indicating the RX FIFO is empty.
Debug Signals	
TX_led	Flag indicating the TX module is active. Routed to LED on the RTSC for visual feedback.
RX_led	Flag indicating the RX module is active. Routed to LED on the RTSC for visual feedback.

Table 8: RS232 Module Signals

4.2.4.2. TX Module

The TX Module reads data from the TX FIFO and transmits it over the serial asynchronous TX RS232 line. Figure 16 provides a flow chart of the module.

The module waits in an idle state until the TX FIFO is not empty, indicating there is data ready for transmission. The TX Module then transmits data as shown in Figure 14, dropping the TX line low to indicate the start bit, transmitting the 8 data bits, and then

pulling the TX line high again to indicate end of transmission. The module then performs an inter-byte wait period of 200 ns, meaning it will not attempt to transmit another byte until the inter-byte period has elapsed. The TX FIFO read enable signal is then pulsed high to remove the transmitted data from the FIFO and the module returns to idle, awaiting additional data to be placed into the TX FIFO.

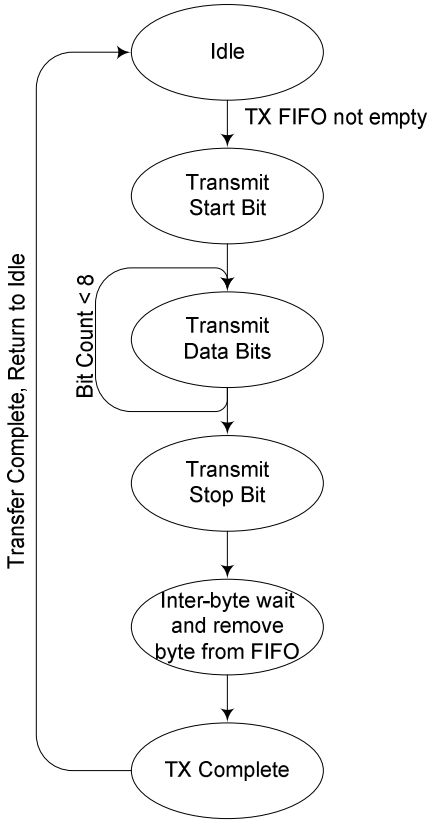

Figure 16: TX Module Flow Chart

4.2.4.3. RX Module

Figure 17 provides a flow chart of the RX Module. The module is idle until the falling edge of the RX line, signaling the start bit. The module then waits for 1.5 times the baud delay and reads the middle of the first data bit. Subsequent data bits are read in, with only the baud delay between to remain in the middle of the bit. Upon reading the 8th

bit, the module waits for the stop bit to indicate the end of the transmission and writes the byte to the RX FIFO.

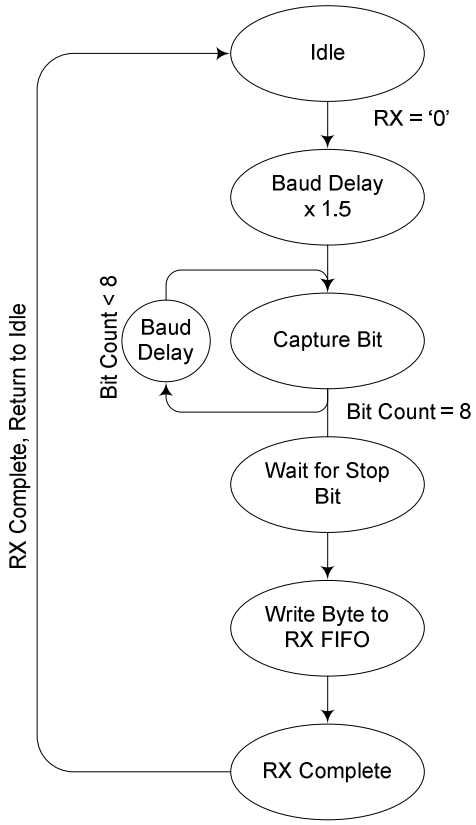

Figure 17: RX Module Flow Chart

4.2.5. RAM Module

The RAM module drives the interface to the Micron MT45W8MW16BGX RAM chip that is on the RTSC board. The RAM is used for storing stimulation waveforms. To allow different modules to make use of the RAM an arbitration module has also been implemented.

4.2.5.1. Micron MT45W8MW16BGX

The Micron MT45W8MW16BGX is a CMOS pseudo-static random access memory that provides 128 Mb of DRAM. For this application, single asynchronous reads and writes are used.

Figure 18 provides a timing diagram showing an asynchronous read operation. Note that write enable (WE#) is held high, indicating a read operation. The address must be valid when CE, OE, LB, and UB are pulled low and valid data is placed on the Data bus after 85 ns.

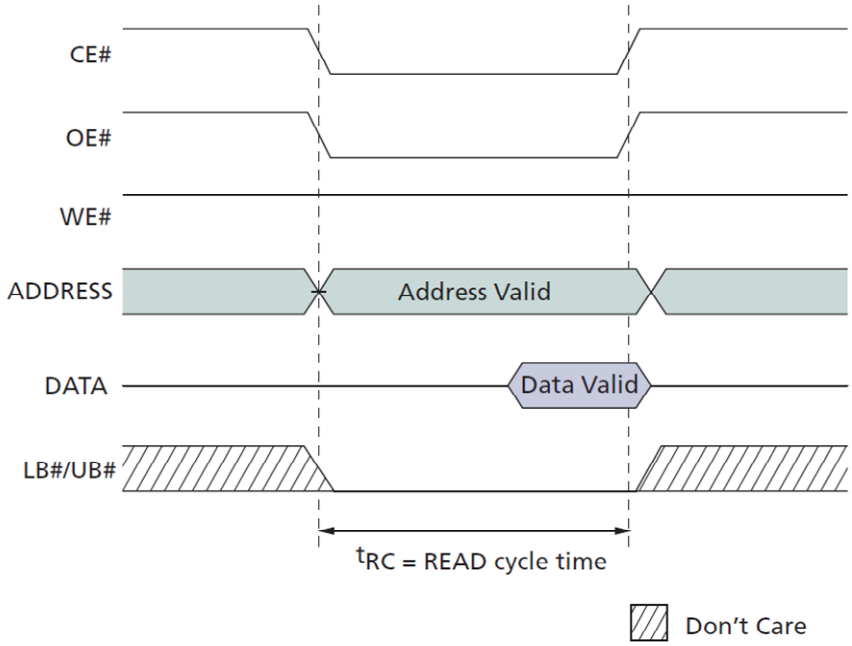

Figure 18: MT45W8MW16BGX Asynchronous Read Operation (Figure 5 from [22])

Figure 19 provides a timing diagram showing an asynchronous write operation. Note that write enable (WE#) is low, indicating a write operation. The address must be valid when CE, OE, LB, and UB are pulled low and valid data must be ready on the Data bus within 85 ns.

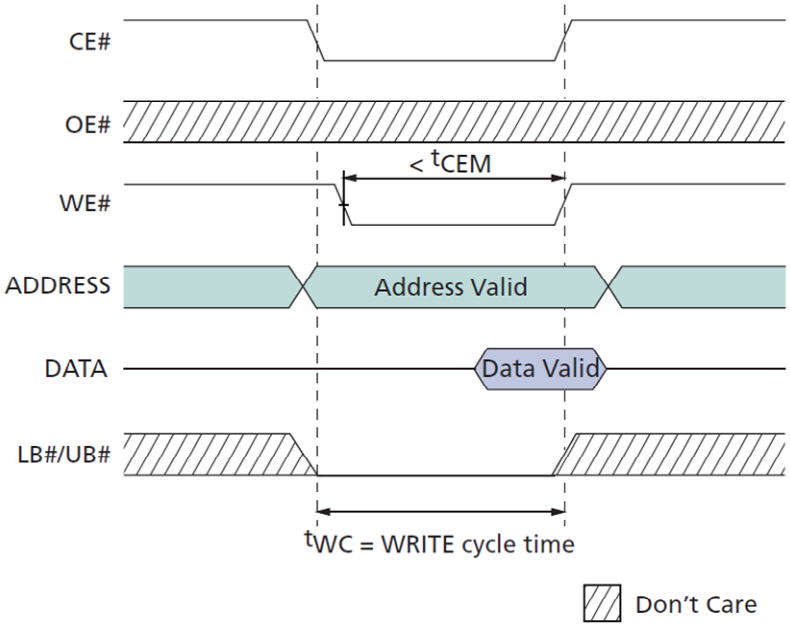

Figure 19: MT45W8MW16BGX Asynchronous Write Operation (Figure 6 from [22])

4.2.5.2. RAM Module Implementation

Figure 20 shows the structure of the RAM Module implementation. Table 9 provides a brief description of each of the RAM Module IO signals. Note that Table 9 groups signals into multiple headings (MT45W8MW16BGX, RAM Module Control, and RAM Arbiter) and that Figure 20 uses these groupings for defining connections.

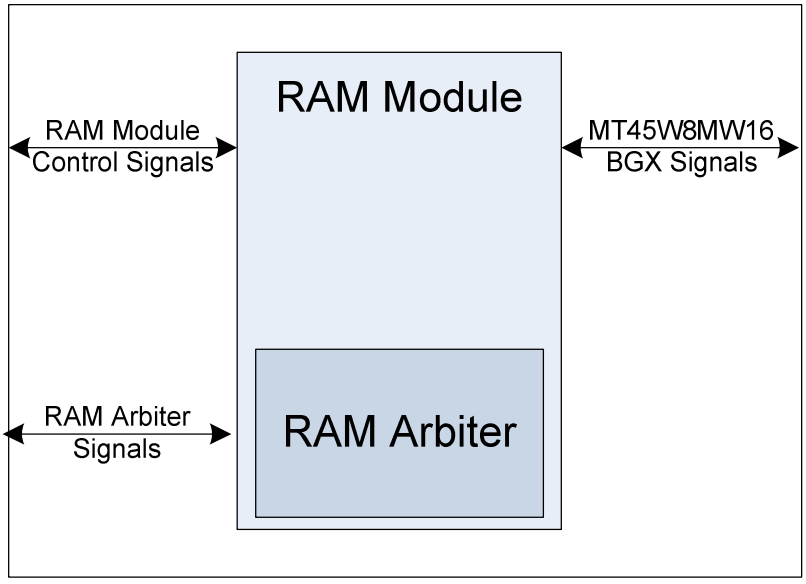

Figure 20: RAM Module Block Diagram

Signal Name	Description
clk	Main clock for module (50 MHz)
reset	Module reset (active low). All registers return to startup values.
MT45W8MW16BGX Signals	
MT_ADDR	RAM Address.
MT_DATA	16-bit bi-directional RAM data bus.
MT_OE	RAM Output enable (active low).
MT_WE	0: Write Operation 1: Read Operation
MT_ADV	Address valid (active low). Indicates the current value on MT_ADDR is valid.
MT_CLK	Clock. RTSC uses asynchronous operation and holds the clock low.
MT_UB	Upper Byte enable
MT_LB	Lower Byte enable
MT_CE	Chip Enable (active low).
MT_CRE	Control Register Enable. Held low for RTSC, feature unused.
RAM Module Control	
RAM_Start_Op	Trigger for the RAM module to start a memory operation.
RAM_Op_Done	Flag indicating the memory operation has completed.
RAM_WE	Write enable indicator for the RAM module. 0: Write Operation 1: Read Operation
RAM_ADDR	Memory address for the commanded operation.
RAM_DOUT	16-bit output data bus from RAM used for read operations.
RAM_DIN	16-bit input data bus to RAM used for write operations.
RAM Arbiter Signals	
RAM_Bus_Request	Request signal for accessing memory interface.
RAM_Bus_Busy	Flag indicating memory interface is currently in use.
RAM_Bus_Grant	Signal granting access to memory interface received in response to a bus request.

Table 9: RAM Module Signals

Figure 21 provides a flow chart of the RAM Module. Upon startup the module idles for 150 us allowing the MT45W8MW16BGX to initialize. The module then idles until a RAM start operation request is received. Based on RAM_WE, a single asynchronous read (RAM_WE = 1) or asynchronous write (RAM_WE = 0) is performed. The module then pulses RAM_Op_Done high to indicate the operation is complete and returns to the idle state.

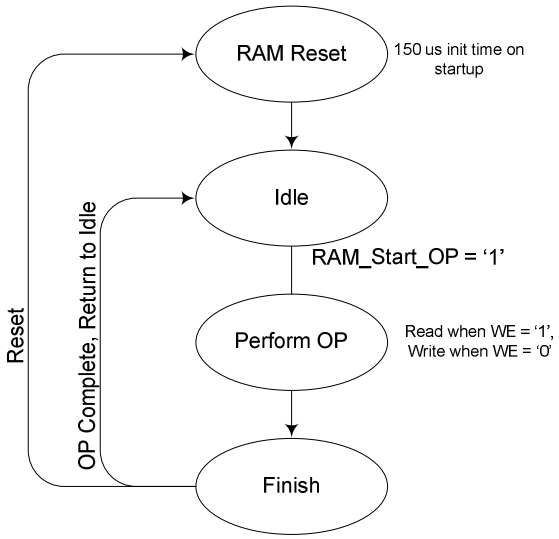

Figure 21: RAM Module Flow Chart

4.2.6. USB Module

The USB Module pulls packetized data from the USB FIFO and passes it on to the Cypress EZ-USB over the synchronous slave FIFO bus. Figure 22 provides a block diagram showing a synchronous slave FIFO transaction. For this interface the Cypress EZ-USB serves as the master and the FPGA serves as the slave. To start a transaction, the FPGA pulls the SLWR line low, indicating a slave write request. On each following rising edge of the slave FIFO clock the Cypress EZ-USB captures data into its internal FIFOs in preparation for USB transmission.

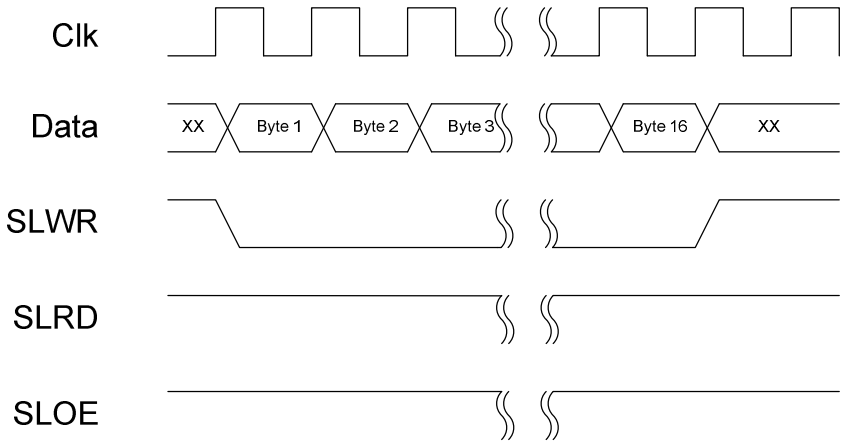

Figure 22: Synchronous Slave FIFO (as described in [23])

4.2.6.1. USB Module Implementation

Figure 23 shows the structure of the USB Module implementation. Table 10 provides a brief description of each of the USB Module IO signals. Note that Table 10 groups signals into multiple headings (Synchronous Slave FIFO, USB FIFO, and Debug Outputs) and that Figure 23 uses these groupings for defining connections.

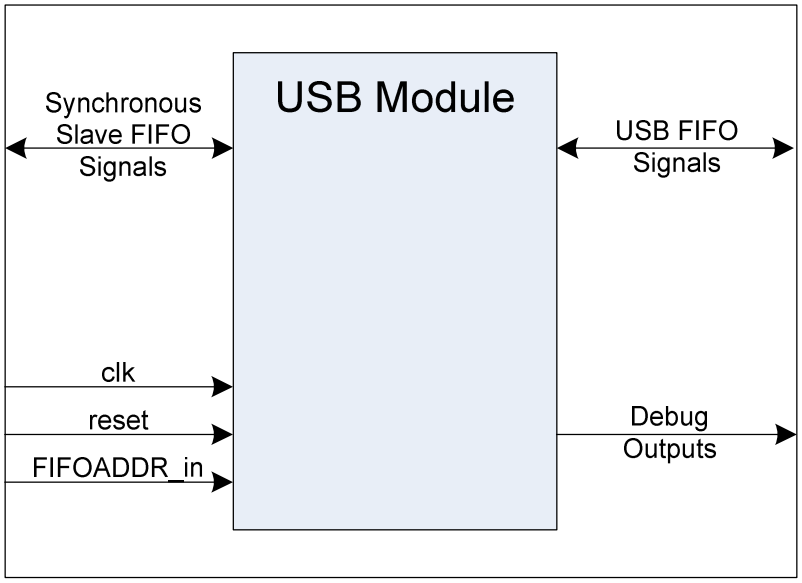

Figure 23: USB Module Block Diagram

Signal Name	Description
clk	Slave FIFO clock. Sourced from the Cypress EZ-USB.
reset	Module reset (active low). All registers return to startup values.
FIFOADDR_in	Input routed to switches on Nexys2 dev board. Address of Cypress EZ-USB endpoint FIFO to write to. Current configuration should keep this set to "00".
Synchronous Slave FIFO Signals	
Data	8-bit data bus
PktEnd	End of packet indicator (active low). Trigger Cypress EZ-USB to send a USB packet, even if the corresponding FIFO is not full. Unused for RTSC.
FlagB	Flag indicating the Cypress FIFOs are full (active low).
SLRD	Data is read from the Cypress EZ-USB FIFO upon assertion of SLRD (active read). Unused for RTSC.
SLWR	Data is written to the Cypress EZ-USB FIFO upon assertion of SLWR (active low).
SLOE	Enables the Cypress EZ-USB to drive data on the Data bus (active low). Used in conjunction with SLRD. Unused in RTSC.
FIFOADDR	Address of Cypress EZ-USB endpoint FIFO to write to.
USB_FIFO Signals	
FIFO_DOUT	8-bit parallel data bus
FIFO_RD_CLK	USB_FIFO read clock
FIFO_RD_EN	USB FIFO Read Enable (Active High).
FIFO_EMPTY	Flag indicating the USB FIFO is empty (Active High).
FIFO_ALMOST_EMPTY	Flag indicating one word from empty (Active High).
FIFO_PROG_EMPTY	Flag indicating less than or equal to 15 words in FIFO (Active High).
Debug Outputs	
FlagB_out	Display FlagB value to LED
Idle_out	Display module status to LED. When idle the USB_FIFO does not contain data to pass on to Cypress EZ-USB. On: Module is idle Off: Module is active

Table 10: USB Module Signals

Figure 24 provides a flow chart of the USB Module. The module is idle until the USB FIFO "FIFO_PROG_EMPTY" flag is de-asserted, indicating a full packet is available. The module then reads the first byte from the USB FIFO and writes it to the Cypress EZ-USB Synchronous Slave FIFO. This is repeated for all 16 bytes of the packet and then the module returns to idle.

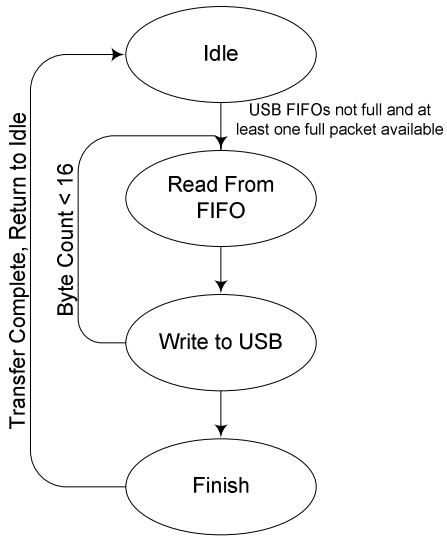

Figure 24: USB Module Flow Chart

4.2.7. Command Handler Module

RTSC API commands (see RTSC Application Programming Interface (API) section of this document) are received over RS232 and placed in the RX FIFO. The Command Handler process and acts on these commands and places the RS232 reply into the TX FIFO from where it will be transmitted back to the PC application.

4.2.7.1. Command Handler Implementation

Figure 25 shows the structure of the Command Handler implementation. Table 11 provides a brief description of the Command Handler IO signals. Note that Table 11 groups signals into multiple headings (RTSC Register Configuration, RX FIFO, TX FIFO, RAM Module, and RAM Arbiter) and that Figure 25 uses these groupings for defining connections.

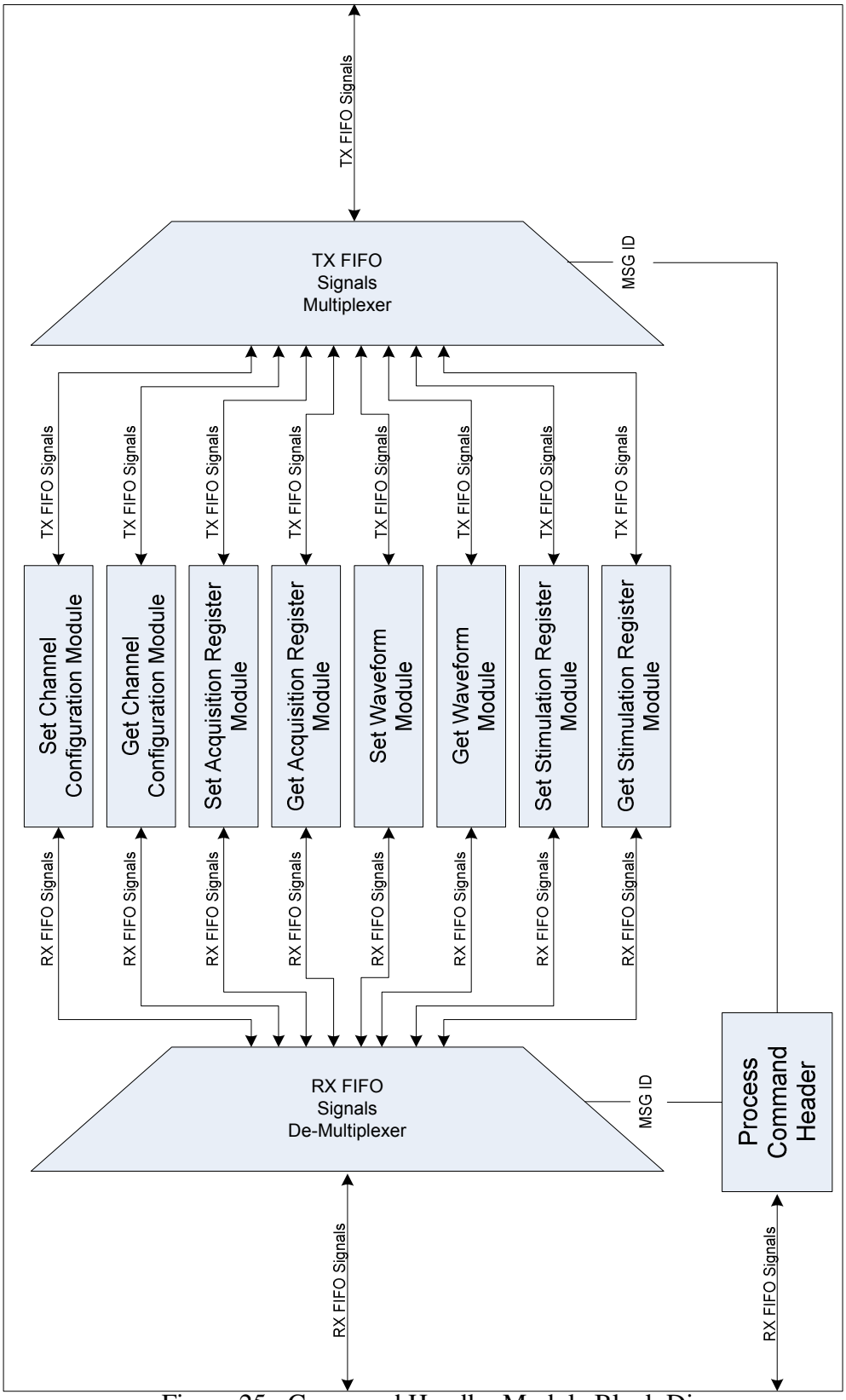

Figure 25: Command Handler Module Block Diagram

Signal Name	Description
clk	Slave FIFO clock. Sourced from the Cypress EZ-USB.
reset	Module reset (active low). All registers return to startup values.
RTSC Register Configuration	
Channel 1-8 Configuration	Channel Configuration Registers (see Table 1)
Stimulation	Stimulation Register (see Table 2)
Acquisition	Stimulation Register (see Table 3)
RX_FIFO Signals	
RX_FIFO_RD_CLK	RX FIFO read clock
RX_FIFO_DOUT	8-bit data bus for reading from RX FIFO
RX_FIFO_RD_EN	RX FIFO Read Enable (Active High).
RX_FIFO_EMPTY	Flag indicating the RX FIFO is empty (Active High).
TX_FIFO Signals	
TX_FIFO_WR_CLK	TX FIFO write clock
TX_FIFO_DIN	8-bit data bus for writing to TX FIFO
TX_FIFO_WR_EN	TX FIFO Write Enable (Active High).
RAM_Module Control	
RAM_Start_Op	Trigger for the RAM module to start a memory operation.
RAM_Op_Done	Flag indicating the memory operation has completed.
RAM_WE	Write enable indicator for the RAM module. 0: Write Operation 1: Read Operation
RAM_ADDR	Memory address for the commanded operation.
RAM_DOUT	16-bit output data bus from RAM used for read operations.
RAM_DIN	16-bit input data bus to RAM used for write operations.
RAM_Arbiter Signals	
RAM_Bus_Request	Request signal for accessing memory interface.
RAM_Bus_Busy	Flag indicating memory interface is currently in use.
RAM_Bus_Grant	Signal granting access to memory interface received in response to a bus request.

Table 11: Command Handler Module Signals

Figure 26 provides a flow chart of the Command Handler module. The module is idle until the RX_FIFO_EMPTY flag indicates that data is available in the RX FIFO. Upon reception of a start byte (0x5A), indicating the start of an API message, the module reads in and processes the message header. The header contains the Message ID, Message Length, and channel the message is intended for. From there, the appropriate message specific handler is started to perform the commanded action. See the RTSC Application Programming Interface (API) section of this document for more information regarding the supported API commands and actions performed upon reception of each command.

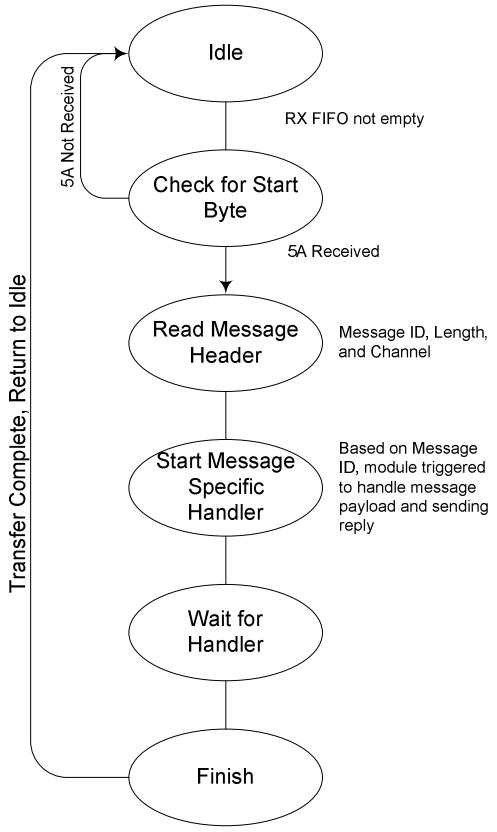

Figure 26: Command Handler Flow Chart

4.3. RTSC Cypress EZ-USB

The Cypress CY7C68013A [23] is provided on the Nexys2 development board. It is used for programming the FPGA via USB and supports user data transfers.

4.3.1. FPGA Programming

The Cypress CY7C68013A is connected to the Joint Test Action Group (JTAG) bus and can be used to load the configuration file for the FPGA. The Cypress CY7C68013A receives the configuration file and command to program the FPGA through USB and drives the JTAG bus to complete the command. The appendix provides a detailed overview of how to load the configuration file. The Xilinx Platform Flash configuration PROM and Xilinx XC9572XL CPLD are also on the JTAG scan chain and may be programmed by the Cypress CY7C68013A[15, 19].

4.3.2. USB Data Transfers

The original Digilent® firmware that is loaded from EEPROM on startup supports Bulk USB transfers of size 512 bytes and a 1024 byte FIFO. In order to provide minimal latency and guaranteed reception for acquired ADC data, custom firmware was required. The custom firmware configures the Cypress CY7C68013A for Interrupt transfers of size 1024 over a single endpoint with a 4096 byte FIFO. Digilent® does not support modification of the firmware and it is claimed that doing so will void the board. However, it is possible to write alternative firmware to the volatile internal memory of the chip, providing a solution where you are able reprogram the firmware, but the original Digilent® firmware remains in the EEPROM and is loaded whenever power to the board is cycled.

	VID:PID	USB Transfer Mode	Endpoints Supported	Packet Size	FIFO Size (per endpoint)
Digilent Firmware	1443:0005	Bulk	2,4,6,8	512	1024
Custom Firmware	04B4:8613	Interrupt	2	1024	4096

Table 12: Digilent Firmware Vs. Custom Firmware USB Configuration

Table 12 provides an overview of the USB configuration for Digilent's firmware and the custom firmware. As noted above, the custom firmware supports Interrupt USB transfers. Interrupt transfers are necessary because they have a guaranteed latency, meaning every 125 us microframe the OS will service the USB endpoint. Interrupt transfers also allow for error detection and retry to guarantee valid data reception.

4.4. Data Acquisition and Stimulation Control Center

The developed PC application issues commands (via RS232) to the Real Time System Controller (RTSC) board and receives acquired data (via USB) from the RTSC board. Commands can be sent manually from the GUI or automated through scripting. Acquired data can be graphed, including support for overlaying multiple channels on the same plot and the option to export to a comma separated values (.csv) file for analysis using other applications (ex. Matlab®, Excel®). Figure 27 provides a screenshot of the PC Application with a script loaded and two channels of acquired data overlaid on the same plot.

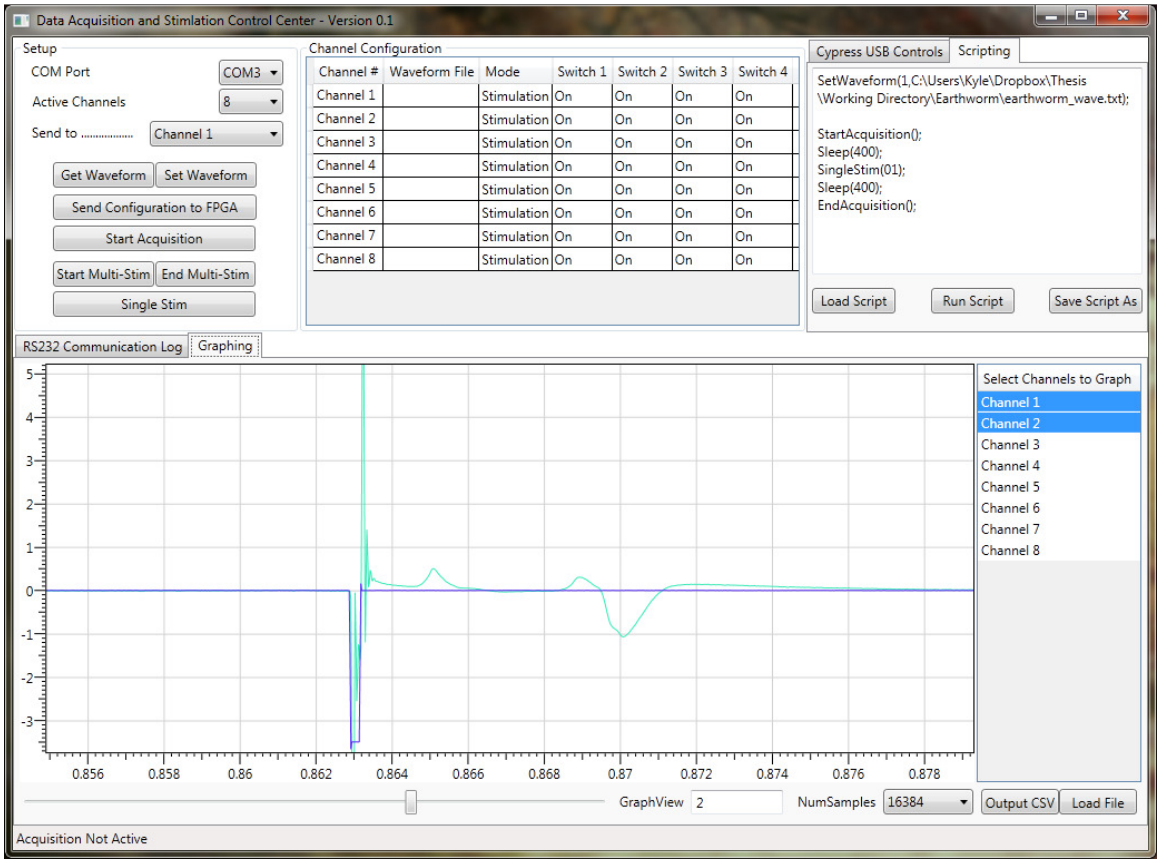

Figure 27: Data Acquisition and Stimulation Control Center

4.4.1. PC Application Design

Figure 28 provides the class diagram for the Data Acquisition and Stimulation Control Center. Each of the primary functional classes is displayed, ignoring the main window class and a few other supporting classes.

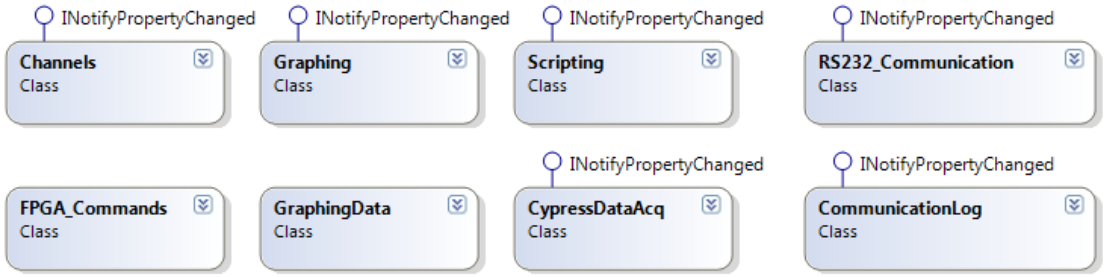

Figure 28: DASCC class diagram

4.4.1.1. Channels Class

The Channels class serves as a container for channel configuration data. The current version of DASCC supports from 1 to 8 active channels, each of which can be configured for Stimulation or Acquisition.

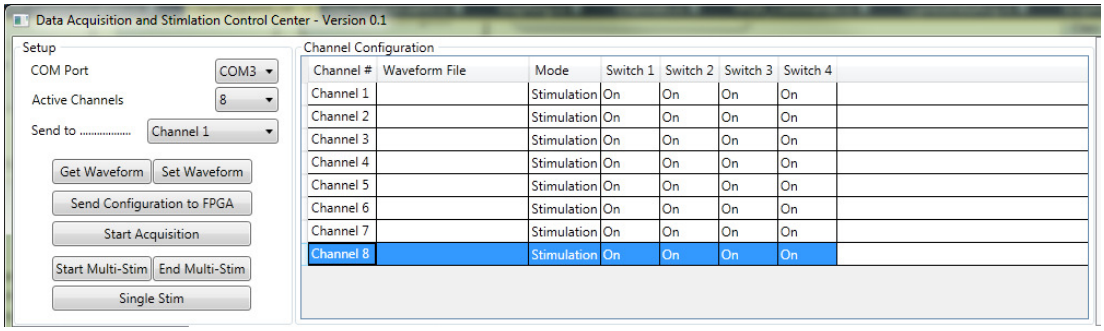

Figure 29: DASCC Channel Configuration

Figure 29 shows the portion of the GUI that is bound directly to each active Channels class object. By changing “Active Channels” the number of Channels class objects are displayed and each can be configured within the data grid view. Double Clicking the “Waveform File” column launches an “Open File” dialog box allowing the

user to point directly to the desired waveform file. The Switch 1-4 settings provided allow for future control of the pre-amp switches; RTSC does not yet support direct control of these switches.

4.4.1.2. Graphing Class

The Graphing Class provides graphing functionality and control over the set of data that is currently displayed. It leverages the Dynamic Data Display [27] graphing library. Support has ceased for the library but the base functionality it provides suited this application and it is available under Microsoft Reciprocal License (Ms-RL), a Microsoft open-source license. Figure 30 shows the Graphing view and related controls.

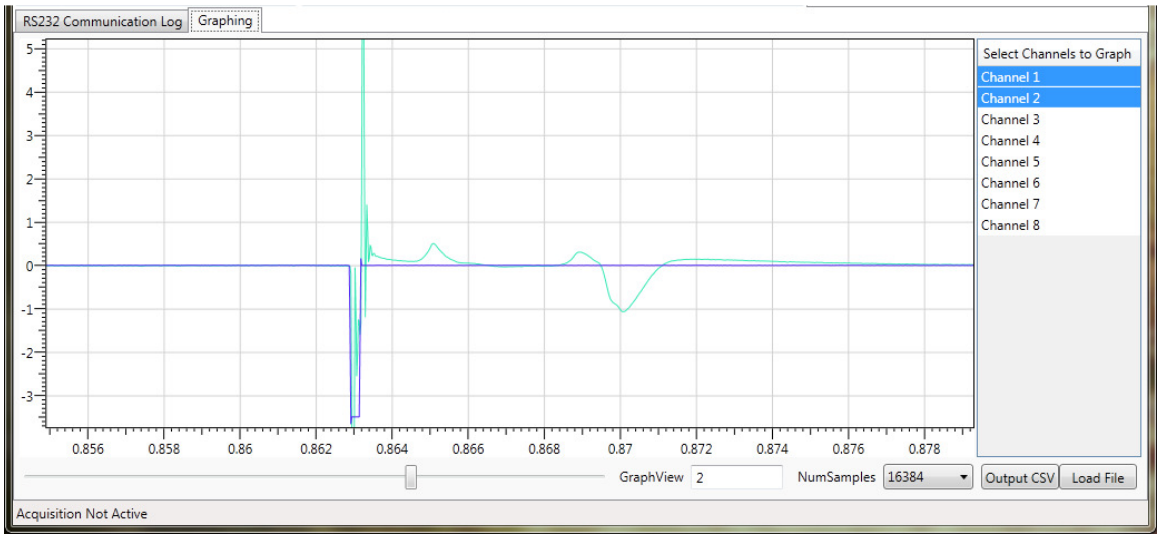

Figure 30: DASCC Graphing View

After data is acquired it can be loaded into the Graphing view by selecting the "Graphing" tab and hitting the "Load File" button. Based on the number of channels contained within the loaded data the "Select Channels to Graph" list box will be populated and the first channel will be selected. Data from multiple channels can be overlaid using Ctrl+LeftMouse to select and deselect channels. Due to the large number of samples each data file can hold, the view is divided into a configurable number of

samples selected using the "NumSamples" control. Upon dividing data, the slider bar is used to move through each NumSamples set of data. The current set index is shown in the "GraphView" textbox.

4.4.1.3. GraphingData Class

The GraphingData class serves as a container for the data loaded from file and the data that is displayed in the Graphing view. For more details on graphing functionality see the Graphing Class.

4.4.1.4. Scripting Class

The Scripting class interprets user scripts and triggers the appropriate actions sequentially. The appendix provides an overview of the supported scripting commands.

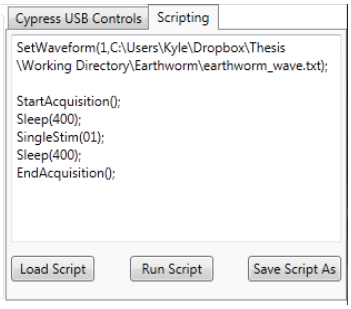

Figure 31: DASCC Scripting View

Figure 31 shows the scripting view with a loaded script. "Load Script" allows the user to select a script file to load. "Save Script As" with save the currently loaded script to file. "Run Script" sequentially performs the script commands currently loaded.

4.4.1.5. RS232 Communication Class

The RS232 Communication class provides the functionality of opening a port, closing a port, sending data, and receiving data. The FPGA_Commands class contains a RS322 Communication class object for performing serial communication. Manual and

scripted commands using the RTSC Application Programming Interface (API) are transmitted to the RTSC using the RS232 Communication Class.

4.4.1.6. FPGA_Commands Class

The FPGA Commands class builds RTSC Application Programming Interface (API) requests provided the command to send and the appropriate parameters. It contains a RS322 Communication class object for performing serial communication.

4.4.1.7. CypressDataAcq Class

The CypressDataAcq Class handles data acquisition from the Cypress CY7C68013A using the CyUSB C# Class Library. The CypressDataAcq code is a modified version of "Streamer", which is a demo application installed with Cypress Suite USB. Additions to the existing code include clearing of the data buffers and writing captured data to file. Modification of the internal data buffer size of the CyUSB C# Class Library is also performed, upping the buffer to 0x800000 (8388608) Bytes, or 128 65536 byte packets.

4.4.1.8. CommunicationLog Class

The CommunicationLog class serves as a container for all sent and received communication over the RS232 interfaces, including the related timestamp. A list is maintained of CommunicationLog objects and bound to the, "RS232 Communication Log" tab of the DASCC.

4.4.2. RTSC Application Programming Interface (API)

The Real Time System Controller (RTSC) board accepts commands over RS232 for controlling its operation. Collectively, these commands make up the RTSC API. The PC Application implements the RTSC API.

Each message has a common header that contains the StartByte (0x5A), the MessageID, and the 16-bit length of the message.

4.4.2.1. Set Channel Configuration

The Set Channel Configuration request allows the 8-bit Channel Configuration register (see Table 1) for each channel to be modified. The Channel field specifies the channel register to modify, with channel 1 being 0x01, on up to channel 8 being 0x08. The Configuration byte is the desired value for the Channel Configuration register.

Set Channel Configuration Request (0x01)		
Byte #	**Field**	**Value**
1	StartByte	0x5A
2	Message ID	0x01
3-4	Length	0x0007
5	Channel	
6	Configuration	
7	Checksum	

Table 13: Set Channel Configuration Request

The Set Channel Configuration reply returns the channel of the register that was modified and its updated value. This should be checked against the requested channel and configuration to verify the register has been updated successfully.

Set Channel Configuration Reply (0x81)		
Byte #	**Field**	**Value**
1	StartByte	0x5A
2	Message ID	0x81
3-4	Length	0x0007
5	Channel	same as request
6	Configuration	same as request
7	Checksum	

Table 14: Set Channel Configuration Reply

4.4.2.2. Get Channel Configuration

The Get Channel Configuration request queries the RTSC for the current value of the specified channel's Configuration Register.

Get Channel Configuration Request (0x02)		
Byte #	**Field**	**Value**
1	StartByte	0x5A
2	Message ID	0x02
3-4	Length	0x0006
5	Channel	
6	Checksum	

Table 15: Get Channel Configuration Request

The Get Channel Configuration reply returns the current value of the requested channel's Configuration Register.

Get Channel Configuration Reply (0x82)		
Byte #	**Field**	**Value**
1	StartByte	0x5A
2	Message ID	0x82
3-4	Length	0x0007
5	Channel	same as request
6	Configuration	
7	Checksum	

Table 16: Get Channel Configuration Reply

4.4.2.3. Set Acquisition Register

The Set Acquisition Register request provides the RTSC with a new value for the Acquisition Register. See Table 3 for guidance in the appropriate value for the register.

Set Acquisition Register Request (0x03)		
Byte #	**Field**	**Value**
1	StartByte	0x5A
2	Message ID	0x03
3-4	Length	0x0006
5	Acquisition Register	
6	Checksum	

Table 17: Set Acquisition Register Request

The Set Acquisition Register reply returns the update value of the Acquisition Register. This should be checked against the requested value to verify the register has been updated successfully.

Set Acquisition Register Reply (0x83)		
Byte #	**Field**	**Value**
1	StartByte	0x5A
2	Message ID	0x83
3-4	Length	0x0006
5	Acquisition Register	same as request
6	Checksum	

Table 18: Set Acquisition Register Reply

4.4.2.4. Get Acquisition Register

The Get Acquisition Register request queries the RTSC for the current value of the Acquisition Register.

Get Acquisition Register Request (0x04)		
Byte #	**Field**	**Value**
1	StartByte	0x5A
2	Message ID	0x04
3-4	Length	0x0005
5	Checksum	

Table 19: Get Acquisition Register Request

The Get Acquisition Register reply returns the current value of the Acquisition Register.

Get Acquisition Register Reply (0x84)		
Byte #	**Field**	**Value**
1	StartByte	0x5A
2	Message ID	0x84
3-4	Length	0x0006
5	Acquisition Register	same as request
6	Checksum	

Table 20: Get Acquisition Register Reply

4.4.2.5. Set Waveform

The Set Waveform request updates the stored stimulation waveform for the requested channel. Number of Samples provides the number of Amplitude:Time pairs for the waveform. Amplitude:Time pairs are sent sequentially, from sample 1 to sample n, followed by the message checksum. The waveform is stored to the requested channel's memory offset as described in Table 7.

Set Waveform Request (0x05)		
Byte #	**Field**	**Value**
1	StartByte	0x5A
2	Message ID	0x05
3-4	Length	6 + samples * 4
5	Channel	
6	Number of Samples	
7-8	Amplitude 1	
9-10	Time 1	
	Amplitude n	
	Time n	
6 + samples * 4	Checksum	

Table 21: Set Waveform Request

The Set Waveform reply returns a status for the Set Waveform request.

Set Waveform Reply (0x85)		
Byte #	**Field**	**Value**
1	StartByte	0x5A
2	Message ID	0x85
3-4	Length	0x0007
5	Channel	
6	Status	0x00: Success 0x01: Failure
7	Checksum	

Table 22: Set Waveform Reply

4.4.2.6. Get Waveform

The Get Waveform request queries the RTSC for the currently stored stimulation waveform of the specified channel.

Get Waveform Request (0x06)		
Byte #	**Field**	**Value**
1	StartByte	0x5A
2	Message ID	0x06
3-4	Length	0x0006
5	Channel	
6	Checksum	

Table 23: Get Waveform Request

The Get Waveform reply returns the requested channel's stored stimulation waveform. The Number of Samples is followed by each of the Amplitude:Time pairs of the waveform.

Get Waveform Reply (0x86)		
Byte #	**Field**	**Value**
1	StartByte	0x5A
2	Message ID	0x86
3-4	Length	7 + samples * 4
5	Channel	
6	Status	0x00: Success 0x01: Failure
7	Number of Samples	
8-9	Amplitude 1	
10-11	Time 1	
	Amplitude n	
	Time n	
7 + samples * 4	Checksum	

Table 24: Get Waveform Reply

4.4.2.7. Set Stimulation Register

The Set Stimulation Register request provides the RTSC with a new value for the Stimulation Register. See Table 2 for guidance in the appropriate value for the register. The Continuous Register allows for the support of single or multi-stimulation. One clock cycle after setting the Stimulation Register the values of the Continuous Register overwrite the Stimulation Register. This allows the Stimulation register to be pulsed high for single stimulation or held high for multi-stimulation.

As an example, assume the Stimulation Register is set to 0x03 and Continuous Register is set to 0x01. This means channels 1 and 2 are initially set, due to the 0x03 setting bits 0 and 1. After one clock cycle the contents of the Continuous Register are copied to the Stimulation Register. This means only channel 1 remains set, due to the 0x01 leaving bit 0 set and clearing bit 1. The overall result is channel 1 has been configured for multi-stimulation and channel 2 has been configured for single stimulation.

Set Stimulation Register Request (0x07)		
Byte #	**Field**	**Value**
1	StartByte	0x5A
2	Message ID	0x07
3-4	Length	0x0006
5	Stimulation Register	
6	Continuous Register	
7	Checksum	

Table 25: Set Stimulation Register Request

The Set Stimulation Register reply returns the current value of the Stimulation Register. Only channels configured for multi-stimulation should be set, meaning this should match the Continuous Register value from the request.

Set Stimulation Register Reply (0x87)		
Byte #	**Field**	**Value**
1	StartByte	0x5A
2	Message ID	0x87
3-4	Length	0x0006
5	Stimulation Register	
6	Checksum	

Table 26: Set Stimulation Register Reply

4.4.2.8. Get Stimulation Register

The Get Stimulation Register request queries the RTSC for the current value of the Stimulation Register.

Get Stimulation Register Request (0x08)		
Byte #	**Field**	**Value**
1	StartByte	0x5A
2	Message ID	0x08
3-4	Length	0x0005
5	Checksum	

Table 27: Get Stimulation Register Request

The Get Stimulation Register reply returns the current value of the Stimulation Register.

Set Stimulation Register Reply (0x88)		
Byte #	**Field**	**Value**
1	StartByte	0x5A
2	Message ID	0x88
3-4	Length	0x0006
5	Stimulation Register	
6	Checksum	

Table 28: Get Stimulation Register Reply

5. Electrophysiology Application[5]

To validate the stimulation and data acquisition system design, a standard electrophysiology experiment, described in [12, 13], was performed on earthworm giant axon action potentials, henceforth referred to as the Earthworm Experiment. This serves the purpose of showing that the Data Acquisition and Stimulation System is capable of being used in an electrophysiology experiment, and the procedure has been performed with previous designs, allowing the new design to be compared with previous results [9].

In the earthworm's nerve cord, there is a median giant axon and two smaller lateral giant axons on either side of the median axon. The lateral giants are connected to each other at multiple locations along the length of the earthworm which has the effect of lateral axons acting as a single giant axon [12]. A physical or electrical stimulation near the anterior or posterior ends of the earthworm will elicit a response from the earthworm in the form of an action potential that propagates along one or both of the giant axons. The propagation speed is related to the cross sectional area of the axon [13]. Varying the intensity of the electrical stimulation at the anterior end of the earthworm will show no response at low intensity, an action potential along the median axon soon after the stimulation at medium intensity, and at higher intensity, action potentials along both the median and lateral giant axons will be visible with the median response occurring sooner than the lateral response due to the differing propagation speeds. Figure 32 shows a cross section of an earthworm based on a figure in [18] with a more detailed representation of the nerve cord based on [13].

[5] This section was co-authored with Donovan Squires, [15].

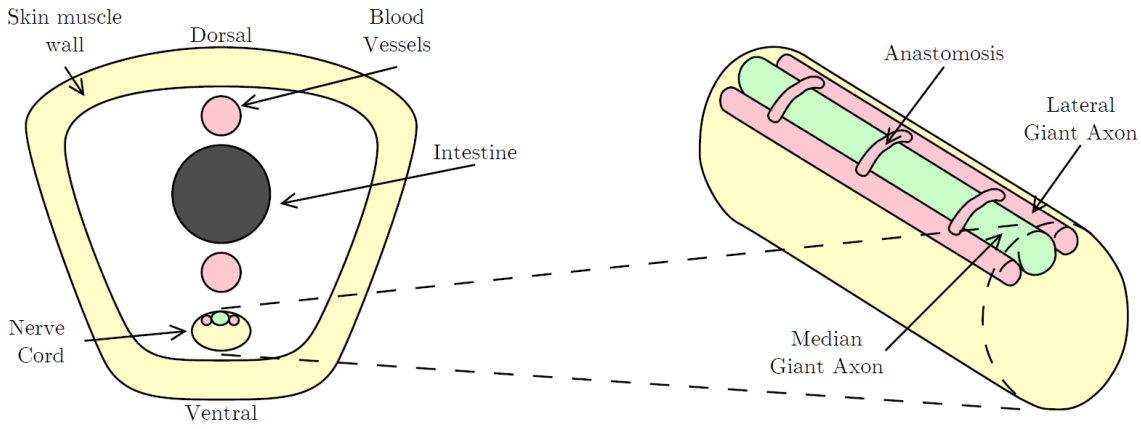

Figure 32: Cross section of an earthworm adapted from [13, 18]. Illustration by D. Squires [15].

Electrical stimulation is accomplished by placing two pins near the anterior end of the earthworm. Measurement of the action potentials along the axon is accomplished by dissecting the earthworm and placing two chlorided silver extracellular electrodes on the earthworm nerve cord. The extracellular electrodes measure the difference in electric potential at two points on the nerve cord. When an action potential propagates along the nerve cord, the electrodes measure a difference in voltage in time that will appear as a biphasic spike. The exact shape of the spike will depend on the placement and size of the electrodes and the condition of the nerve cord itself [12, 13, 24].

The Data Acquisition and Stimulation System was used to perform an experiment studying earthworm giant axon potentials with a previously developed and validated amplifier [9] and oscilloscope connected in parallel with the electrophysiology interface board, allowing the data to be compared.

5.1. Earthworm Setup

The dissection of the earthworm and setup of the stimulation and recording hardware is based on [9, 12, 13, 14]. Two pins are placed near the anterior end of the earthworm. Connected to these pins is the output of the stimulation circuitry. Near the middle of the earthworm, the skin is cut and folded to expose the interior of the worm. The intestine is moved out of the way, and the nerve cord is pulled away from the fluids with two chlorided silver electrodes that are connected to the amplification and recording circuitry. Between the stimulation pins and recording electrodes, a chlorided silver wire is placed under the body of the earthworm and connected to circuit ground, which may be connected to earth ground depending on the circuit setup. Figure 33 is a diagram of the DASS connected to the earthworm.

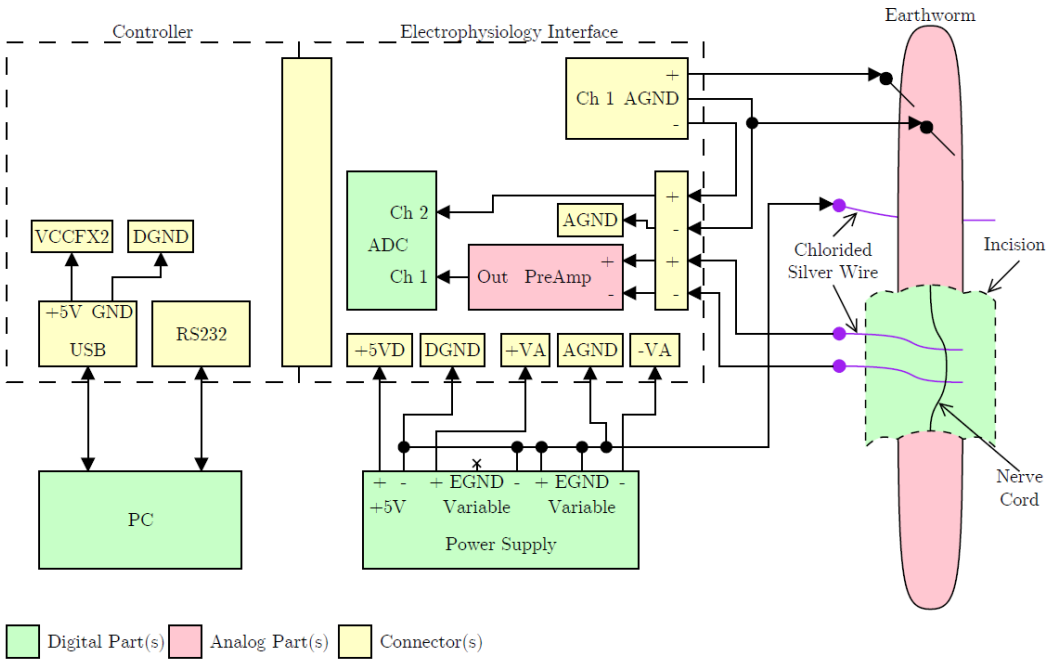

Figure 33: Connections between the DASS and an earthworm. Illustration by D. Squires [15].

To compare the Data Acquisition and Stimulation System with a previously validated setup, the Preamp from [9, 10], with its output connected to an oscilloscope, has its input connected in parallel with the recording electrodes as shown in Figure 34.

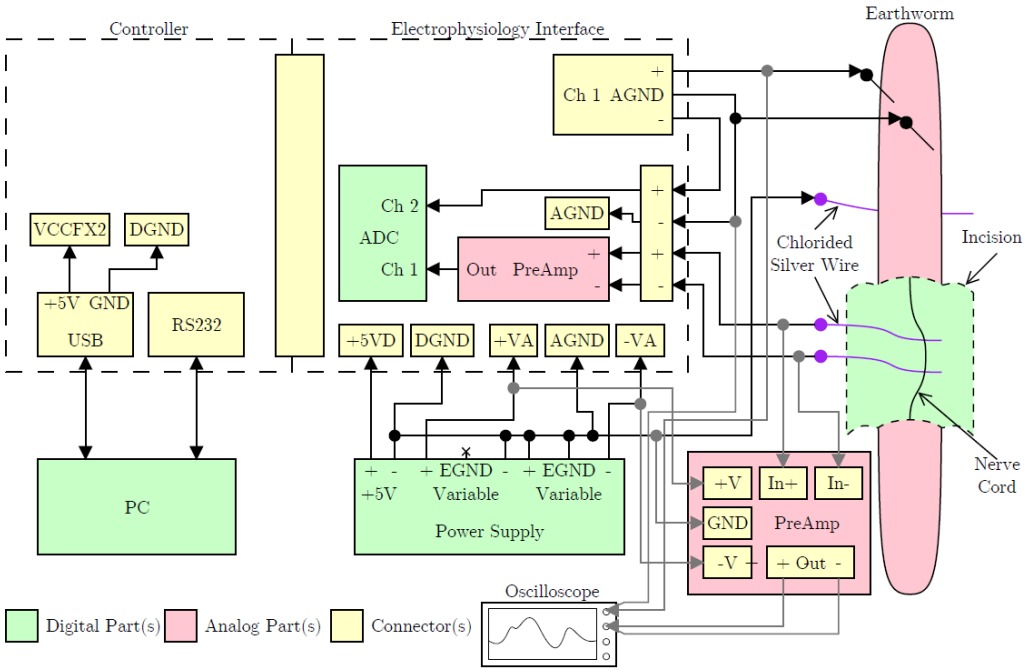

Figure 34: DASS connected in parallel with a previously validated recording system. Illustration by D. Squires [15].

5.2. Chloriding Silver Wire

The silver wire used for the extracellular electrodes must be coated in a layer of silver chloride. To create the coating, electroplating may be used as described in [13, 25]. A simpler method is to place the portion of the wire that will be in contact with living tissue in full-strength Clorox bleach and leave the wire in the bleach until the wire acquires a purplish-gray tint, which should take between ten and thirty minutes [25].

5.3. Earthworm Experiment Procedure

The earthworm experiment procedure was adapted from [12, 13].

5.3.1. Earthworm Dissection

- Place the earthworm in a 10% ethanol solution to anesthetize it
- Leave the worm in the ethanol solution for the minimum amount of time needed to make the worm easy to work with while dissecting it; 5-10 minutes may be all that is necessary
- Rinse the earthworm in tap water and pin it, dorsal (dark) side up, on the dissecting dish
- Create a capital-letter-I shaped incision, using surgical scissors, one or two inches long, along the length of the worm, about halfway between the anterior and posterior ends of the earthworm
- Pin the skin to the dissecting dish to expose the worm's internal anatomy
- Flush the cavity periodically, as needed, with worm Ringer's solution (in mM units: 102.7 NaCl, 1.6 KCl, 1.8 $CaCl_2$, and 4 $NaHCO_3$ [13]) to make the anatomy easier to view
- Move the intestine aside, using forceps and scissors, exposing the nerve cord
- Free a few centimeters of the nerve cord from its lateral and ventral connections, using the forceps and scissors, to allow the cord to be lifted above and away from the saline and other anatomy
- Position the chloride silver wires under the nerve cord
- Raise the fixture holding the silver wires until the wires and nerve cord are away from the saline and earthworm anatomy (a fraction of an inch is all that is necessary, blowing in the area can break up the surface tension if the saline solution is bridging the wires and cord with the rest of the earthworm)
- Moisten the nerve cord with Ringer's solution, often, throughout the experiment while making sure that the nerve cord and electrodes remain isolated from the rest of the saline and anatomy (it may be necessary to remove excess Ringer's solution)

5.3.2. Electrical Setup

- Place two pins near each other in the anterior end of the earthworm
- Connect the non-inverting output of the stimulation circuit to one pin and connect circuit ground or the inverting output to the second pin
- Connect, optionally, if the inverting output is not connected to a pin in the earthworm, the inverting output of the stimulation circuit to an unused electrode input channel on the Electrophysiology Interface board; ensure that there is no Preamp board in the PCI-Express socket and bypass the socket with a 0Ω resistor. See Figure 33 and Figure 34.
- Place a chloride silver wire under the body of the earthworm, between the stimulation pins and the exposed portion of the earthworm's body and connect the wire to circuit (which may be earth) ground
- Connect the chloride silver recording electrodes to the Preamp inputs with the non-inverting input connected toward the anterior end of the earthworm and the inverting input connector toward the posterior end (reversing the electrodes will simply invert the signal) (connecting the electrodes may be performed before the silver wires are placed under the nerve cord, to avoid disturbing the electrodes in the process of making the connections)
- Power on the Electrophysiology Interface, first, then power on the RTSC

5.3.3. Software Setup

- Update FPGA program as described in the appendix.
- Update Cypress EZ-USB firmware as described in the appendix.
- Create the script and corresponding waveform file as shown the appendix.
- Launch Data Acquisition and Stimulation Control Center (DASCC)
- Select the appropriate COM port from the dropdown list
- Select the appropriate Endpoint from the dropdown list, set Packets Per Xfer to 64, and set Xfers to Queue to 64
- Select the Scripting tab and load the Earthworm Script

- When prepared for single stimulus and capture hit the “Run Script” button. Changes to the voltage being output can be modified by changing the 2nd line of the Waveform file, using the appendix as a reference for selecting the appropriate value.
- Graph data by selecting the Graphing tab and hitting the “Load File” button. Once loaded, multiple channels can be displayed at the same time using ctrl + left mouse button to select/deselect channels.
- Click “Output CSV” to export data to a comma-separated value (CSV) text file.

5.3.4. Stimulation and Recording

- Stimulate the earthworm with a single, 0.2ms wide pulse with low amplitude (less than 1.0V)
- Repeat the stimulation while slowly increasing the pulse amplitude (in steps between 0.1V and 0.5V) until a response is seen between 2ms and 8ms after the stimulation artifact, this is the median giant response
- Save the recorded waveform
- Slowly increase the pulse amplitude, further, until a second response is seen between 6ms and 15ms after the stimulation artifact, this is the lateral giant response
- Save the recorded waveform

5.4. Results

The earthworm experiment was performed with the Data Acquisition and Stimulation System connected in parallel with a preamp and oscilloscope that was validated in [9]. A diagram of the experimental setup is shown in Figure 34. A picture of the experimental setup is shown in Figure 35.

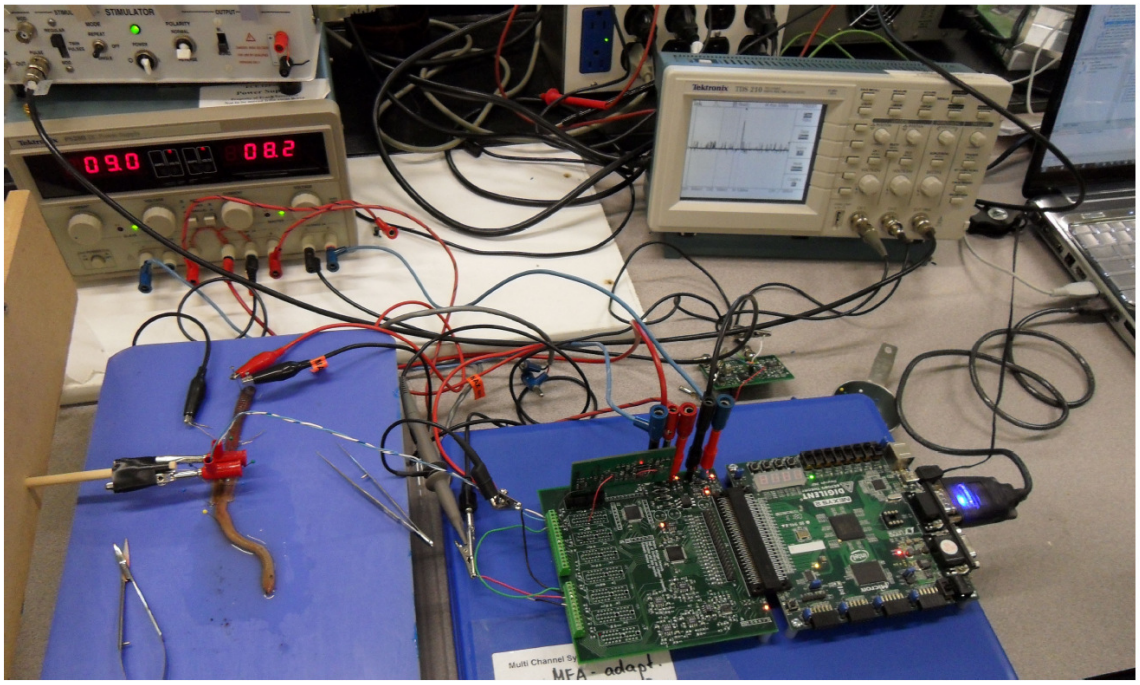

Figure 35: Picture of the Data Acquisition and Stimulation System connected in parallel with a previously validated recording system

Data recorded from a 2.0V stimulation pulse produced a median giant response similar to those seen in [9] and [14]. The data from the previously validated Preamp and the oscilloscope was plotted on top of the data from the Data Acquisition and Stimulation System. The data from the oscilloscope and the Data Acquisition and Stimulation System can be seen in Figure 36 and the data appears to be in very close agreement.

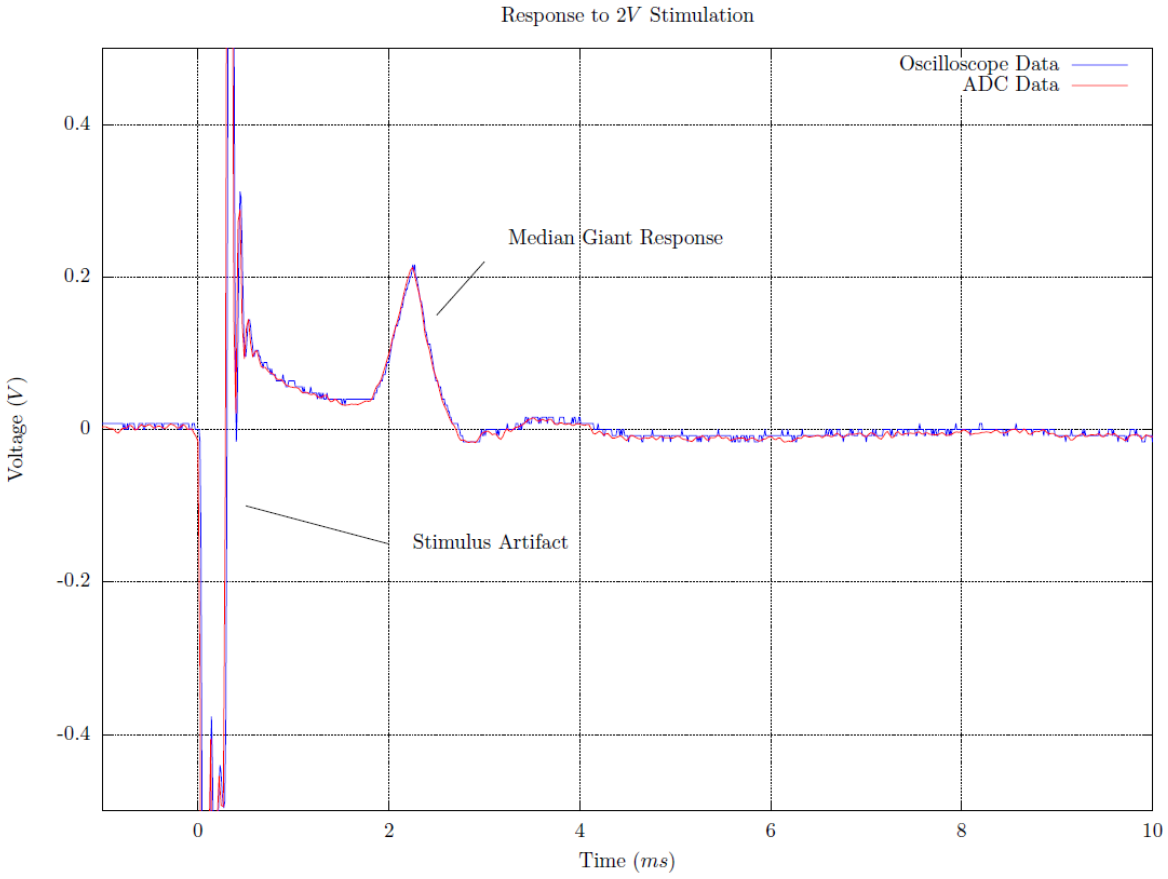

Figure 36: Earthworm response to 2.0V stimulation pulse with data recorded by an Oscilloscope and the Data Acquisition and Stimulation System

Data recorded from a 3.5V stimulation pulse produced median and lateral giant responses similar to those seen in [9] and [14]. The data shown in Figure 37 exhibits close agreement between the Data Acquisition and Stimulation System and the previously validate Preamp.

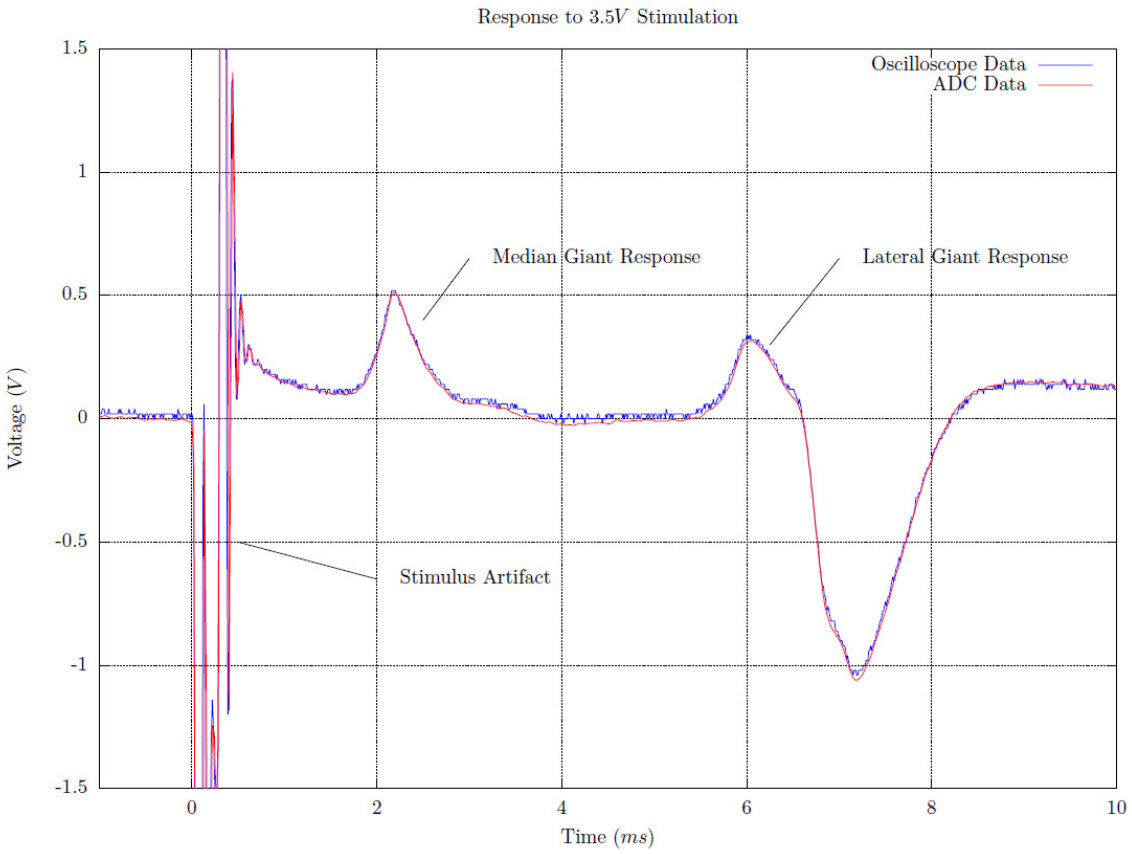

Figure 37: Earthworm response to 3.5V stimulation pulse with data recorded by an Oscilloscope and the Data Acquisition and Stimulation System

6. Specifications Review[6]

This section reviews the specifications set forth in the Specifications section and shows how well the Data Acquisition and Stimulation System described in this thesis meets those specifications.

1. Provide a platform for performing electrophysiology experiments with earthworms as described in [12, 13]

   Successful accomplishment of the earthworm experiment is shown in section 5.4 of this document.

   a. Produce a voltage-controlled square wave stimulation pulse with widths from $0.01ms$ to $100ms$ and amplitudes from $0.1V$ to $10V$

      The FPGA and Data Acquisition and Stimulation Control Center (DASCC) are capable of defining a stimulation waveform that updates the DAC output with a period as short as 1us and storing up to 4096 waveform samples with variable sample periods from 1us to 65,536ms. The DAC has a slew rate of 1.5V/us [21] and a differential output voltage range of ±14.8V as shown in [15].

   b. Produce single stimulation pulses or multiple pulses at rates from $1Hz$ to $10Hz$

      The FPGA and DASCC are capable of defining a stimulation waveform over 1s in length and repeating the waveform indefinitely.

   c. Provide at least one differential recording channel

      There is the potential for up to eight recording channels, each with a Preamp that has a differential input as shown in [15].

[6] This section was co-authored with Donovan Squires, [15].

d. Record an action potential voltage from the time of a stimulation pulse for a minimum duration of 20ms

   Data collected for results in the Earthworm Experiment Procedure Results section is more than 1s in duration.

e. Plot the recorded voltage

   The DASCC is capable of plotting recorded data.

f. Store the recorded voltage to a non-proprietary, standard file format

   The DASCC is capable of exporting recorded data to a comma separated value (CSV) file.

2. Provide a platform for stimulation and recording of neuron cell culture electrical activity via MEA electrodes

   Interfacing the DASS with the MEA is not yet tested.

   a. Provide at least four recording channels

      Eight recording electrodes can be connected to the DASS and stored as digital samples.

   b. Store data from recording channels continuously

      The data recording time limit has not been fully tested.

   c. Provide at least four voltage-controlled arbitrary stimulation channels

      The DASCC is capable of loading an arbitrary waveform, as defined by a text file, into the DASS which outputs the waveform using the AD5678 with four DAC outputs.

   d. Output single-ended stimulation signals on recording electrodes and add culture voltage offset to the stimulation signal

The stimulation signals are routed to the Preamp boards which output the stimulation signals on the recording electrodes and add the culture voltage offset as described in [15].

e. Provide an interface that can specify stimulation waveforms, locations, and intervals that can be updated based on data from the recording electrodes

The DASCC provides a scripting language for defining recording intervals, stimulation waveforms, and stimulation intervals, but does not have provisions for analyzing recorded data and adjusting stimulation strategy.

3. Utilize Low-Noise Amplifier described in [9]

The Low-Noise Amplifier, also known as the Preamp, is successfully utilized with the DASS; though, not all features are tested.

a. Connect to each Low-Noise Amplifier channel with a PCI-Express card edge connector

Eight PCI-Express card edge connector sockets are available on the Electrophysiology Interface board as described in [15].

b. Provide ±7V to ±15V analog voltage supplies and ground via the card edge connector

The Electrophysiology Interface board connects its analog voltage supply inputs, which tolerate ±7V to ±15V, to the Preamp connectors as described in [15].

c. Provide ability to independently switch four digital inputs for each channel, $0_{IH} = 0.8V$ and $1_{IL} = 2.4V$

The CPLD on the Electrophysiology Interface board enables the FPGA to control each digital input on every Preamp channel and voltage compatibility is shown in [15], but a logic configuration for the CPLD is not written.

d. Route differential analog input to the card edge connector for each channel

   The differential inputs of the Preamps are routed to a terminal block for simple connection to recording electrodes as described in [15].

e. Convert the 20Hz to 14.6kHz analog output signal [9] to digital samples

   An AD7606 ADC converts the Preamp output voltage to digital samples, has an analog low-pass input filter with a corner frequency of 23kHz, and can sample at up to 200kS/s, satisfying the sampling theorem, as described in [15].

f. Route a single-ended stimulation signal to each channel

   Four unique stimulation channels with differential outputs are connected single ended to the eight Preamp connectors as described in [15].

7. Conclusions

The Data Acquisition and Stimulation System (DASS) provides eight channels of acquisition and four channels of arbitrary waveform generation to support electrophysiology experiments, including basic (earthworm) and advanced (microelectrode array or MEAs) applications. The system is certainly not limited to the considered use cases; its features are likely to align with many other electrophysiological applications not yet investigated. The DASS expands the capabilities of the Neurobiology Engineering Laboratory at Western Michigan University.

Future Expansion

One of the major goals of the current design was leaving the architecture open for future expansion. A future target is full support of the 60 channels of a MEA. An initial discussion was provided in [15] of a possible hardware architecture. From a software/firmware perspective the main area of concern is the required acquisition data rates. A 60 channel system with a 188 byte packet size would have 8.2908 Mbytes/s, calculated as:

$$Packetized\ ADC\ Data = \frac{188\ bytes * 44100Hz}{1000000} = 8.2908\ \text{Mbytes}/s.$$

This data rate exceeds the previously calculated interrupt mode max throughput of 8.192 Mbytes/s, discussed in the Data Flow Analysis section of this document and calculated as:

$$Interrupt\ Mode\ Bandwidth = 1024\ bytes * \frac{1}{125\ microseconds} = 8.192\ Mbytes/s.$$

Using a different FPGA could provide larger FIFOs and make it possible to not require the 125 microsecond latency that interrupt mode USB transfers provide. With

this relinquished, bulk transfers become possible with a maximum theoretical throughput of 60 Mbytes/s. Despite the expectation that the full 60 Mbytes/s would not be achieved, it is reasonable to assume 8.2908 Mbytes/s of sustained bandwidth is achievable.

The FIFO size concern is better understood by reviewing the current amount of unresponsiveness from a Windows PC required to fill the current FIFO. This is calculated as follows:

$$Acq.FIFO\ Fill\ Time = \frac{\left(\frac{32768\ byte\ FIFO}{32\ byte\ packet} * \frac{1}{44.1kHz}\right)}{1000} = 23.22\ ms.$$

Additionally, the Cypress FIFOs are filled within the following time:

$$Cypress\ FIFO\ Fill\ Time = \frac{\left(\frac{4096\ byte\ FIFO}{32\ byte\ packet} * \frac{1}{44.1kHz}\right)}{1000} = 2.9025\ ms.$$

On the PC, the Cypress driver provides further buffering of data, which can be filled within the following time:

$$Cypress\ Driver\ Fill\ Time = \frac{\left(\frac{8388608\ byte\ FIFO}{32\ byte\ packet} * \frac{1}{44.1kHz}\right)}{1000} = 5944.308\ ms.$$

Thus there is approximately 6 seconds between being completely empty to completely full across the entire chain. If at any point the USB transactions stop being serviced the RTSC only has approximately 26 milliseconds of buffering capability. As is, it is possible to drop data on an overtaxed Windows PC. Further investigation of the Cypress driver and the way Windows interacts with it for maintaining data transfers would help gain confidence in the DASS data integrity.

Expanding on these buffer fill times for a 60 channel system, the same calculations, but with 188 byte packets taken into account, result in 3.952 milliseconds for Acquisition FIFO fill time, 0.494 ms for Cypress FIFO fill time, and 1011.797 ms

Cypress driver fill time. Table 29 summarizes the fill times for the 8 and 60 channel implementations.

	8 Channel Fill Time (ms)	60 Channel Fill Time (ms)
Acquisition FIFO Fill Time	23.22	3.952
Cypress FIFO Fill Time	2.9025	0.494
Cypress Driver Fill Time	5944.308	1011.797

Table 29: Buffer Fill Time

Given that interrupt mode does not provide sufficient bandwidth, bulk mode does not provide guaranteed latency, and the buffer fill time is less than ~1012 milliseconds for a 60 channel system, the ability to use a single USB 2.0 connection with the current FPGA size is unlikely. Further investigation is required to find a viable solution.

One possible consideration for maintaining the current architecture is a much larger FPGA that would have more BRAM resources, providing a larger FIFO. Another possible improvement would be to move to a USB 3.1 supported chip in the place of the current Cypress EZ-USB chip, effectively increasing the maximum bandwidth to 10Gbps.

Other considerations for future system include use of a Real-Time Operating System (RTOS) alongside the FPGA, providing more deterministic performance for the HDD data logging portion of the project. The Xilinx Zynq chip is one such example where a FPGA fabric and dual-core ARM A9 are provided on the same chip. Logging directly to SSD from the ARM A9, or even directly from an FPGA, would increase possible performance. A network connection could be created providing a Windows PC control over the RTOS/FPGA operation and non real-time feedback / decimated data visualization.

Personal Development

This has been a very rewarding project from a personal standpoint. I am happy to have had the opportunity to significantly extend my senior design project [10], from what was a solid baseline (albeit with many issues that made it overall unusable for real experimentation) to a working system for electrophysiological experimentation. This project has provided insight into the challenges of acquiring real-time data and commanding real-time operations from the convenient interface a Windows PC provides. It greatly expanded my knowledge of low-level FPGA development and introduced me to the world of high-level GUI based C# development.

Closing

Even though I have many ideas for work that could be done to improve the system as outlined in part here, I am proud to pass this project and architecture on to those to follow. It is a bittersweet concept; one that I am confident my family will find more sweet than bitter provided the many tireless hours spent in pursuit of a functional prototype.

Appendix A

GitHub Repository

A GitHub repository [26] has been created that contains all source and documentation related to this project. This includes the following:

- RTSC FPGA source code
- DASCC PC source code
- Cypress EZ-USB firmware source code
- Description of required software and drivers
- "Getting Started" guide, describing the steps required to get the system operational

Appendix B

Programming FPGA

The FPGA is programmed using Adept from Digilent [19]. It requires the original Digilent firmware to be running on the Cypress CY7C68013A, which is loaded from the I2C CMOS Serial EEPROM (24AA128) whenever power is cycled. This means the USB cable must be unplugged and plugged in to have the original firmware reloaded (toggling the power switch does not cycle power to the Cypress CY7C68013A).

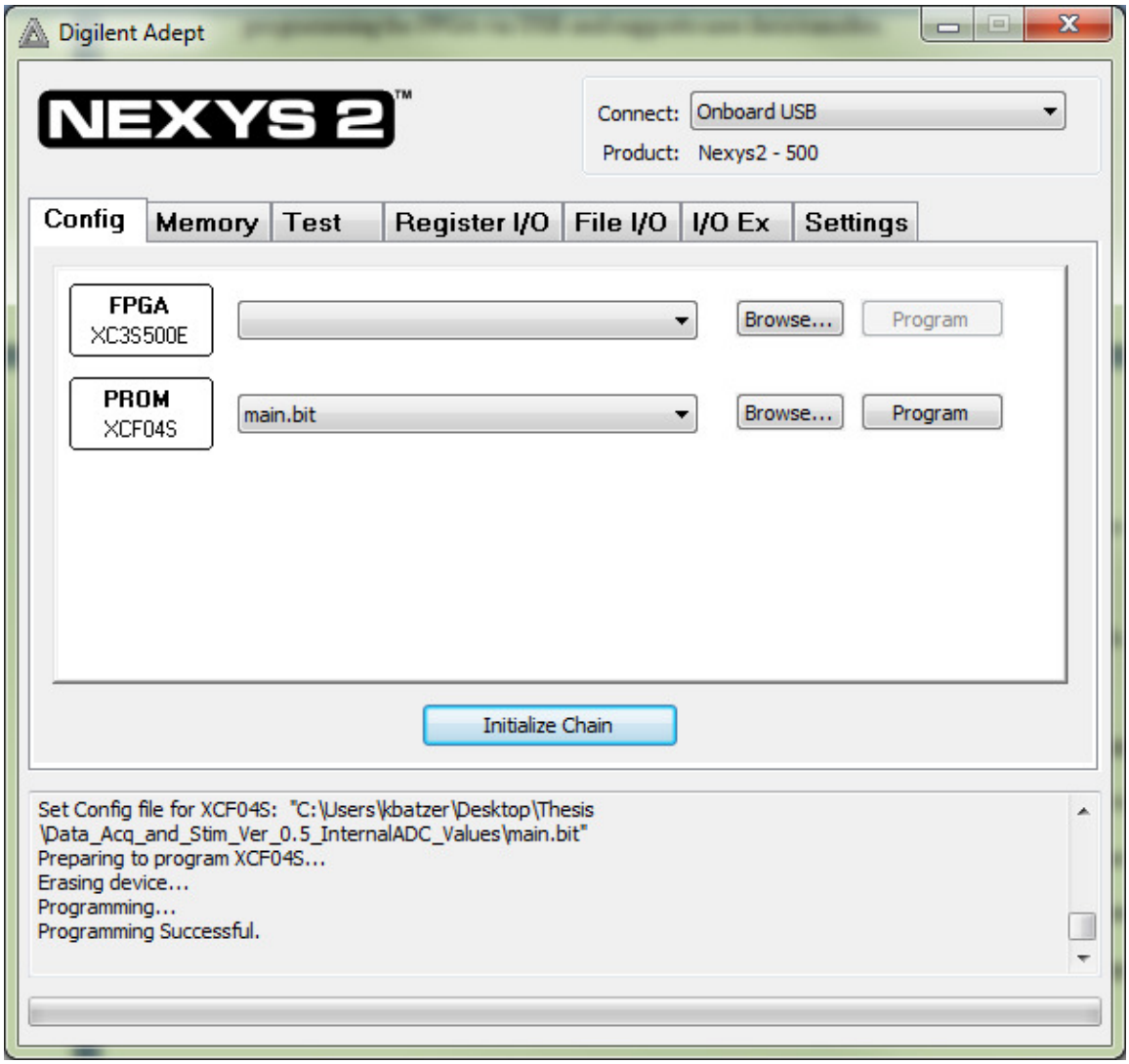

Figure 38: Digilent Adept software used for programming the FPGA

Adept provides a view into the JTAG chain of the Nexys2, which the CY7C68013A commands when programming the FPGA. shows two JTAG targets for programming: the FPGA and PROM. Targeting the FPGA will write to the FPGA RAM and thus programs will only be retained until power is cycled. Targeting the PROM will write to the ROM available on the Nexys2. It is necessary to have the MODE jumper, JP9, set to ROM (pins 2 and 3 connected) to have the FPGA load the program from ROM on power cycle.

For this project, it was desirable to be able to load the program into ROM, allowing the program to be loaded by the FPGA every time power was cycled. To accomplish this, the following steps are required:

1. Launch Adept
2. Unplug the USB Cable and plug it in.
3. In the "Connect" drop down on Adept select "Onboard USB".
4. On the "Config" tab of Adept push "Browse…" next to the PROM and select the appropriate .bit file.
5. Push Program.
6. Ensure JP9 (Mode Jumper) on the Nexys2 is set to load from ROM and unplug and plug in the USB cable to cycle power to the board.
7. The program should load from ROM when the board finishing power up from the previous step.

Appendix C

Programming Cypress EZ-USB (CY7C68013A)

1. Cycle power to the Nexys2 development board by unplugging and plugging in the USB cable (desired FPGA program should load from EEPROM)
2. Using fx2loader, update to an intermediate firmware that updates the VID:PID to 04B4:8613.
    a. Open cmd prompt and navigate to location of fx2loader.
    b. Type “fx2loader.exe -v 1443:0005 firmware.hex” and press enter.
3. Using CypressProgramUtility update to the desired firmware.
    a. Open cmd prompt and navigate to location of CypressProgramUtility.
    b. Type “ProgramUtility.exe ep2_int_in.hex” and press enter.
4. The firmware is ready to use.

Appendix D

DASCC Scripting Amplitude

DASCC scripting takes hex values from 0x0000 to 0xFFFF. In hardware, the DAC will output 0V to 5V, and the corresponding circuitry will take this single ended output and give -7.5V to 7.5V differential output. The following table provides a set of common amplitude values for quick reference throughout the full range available. For specific voltages not in the table use the function and convert the decimal value to HEX:

$$Decimal\ Value = (Desired\ Voltage + 7.5) * \frac{2^{16}}{15}$$

In Excel:

= DEC2HEX((DesiredVoltage + 7.5)*POWER(2,16)/(15))

Desired Voltage	HEX Value	Desired Voltage	HEX Value
-7.5	0	0.5	8888
-7	888	1	9111
-6.5	1111	1.5	9999
-6	1999	2	A222
-5.5	2222	2.5	AAAA
-5	2AAA	3	B333
-4.5	3333	3.5	BBBB
-4	3BBB	4	C444
-3.5	4444	4.5	CCCC
-3	4CCC	5	D555
-2.5	5555	5.5	DDDD
-2	5DDD	6	E666
-1.5	6666	6.5	EEEE
-1	6EEE	7	F777
-0.5	7777	7.49	FFFF
0	8000		

Table 30: DASCC Scripting Amplitude Hex Value Table

Appendix E

Earthworm Script and Waveform

This section provides the script and waveform file used to perform the earthworm experiment.

The script sends EarthwormWaveform.txt to the RTSC where it is stored in memory for channel one. The acquisition is then started, there is a 400 ms delay, and then the stimulation waveform is output on channel one. There is another 400 ms delay to make sure all response to the stimulus is captured and then the acquisition is stopped.

Script.txt

```
SetWaveform(1,EarthwormWaveform.txt);
StartAcquisition();
Sleep(400);
SingleStim(01);
Sleep(400);
EndAcquisition();
```

The waveform file contains the amplitude:time pairs to create the desired waveform. The first line places the output at midscale for 100 us. The second line sets the output to 1V for 100 us. The last line returns the output to midscale.

EarthwormWaveform.txt

```
7FFF,0100
9111,0100
7FFF,0100
```

Appendix F

DASCC Scripting Commands

The DASCC scripting allows the user an easy way to sequence the commands provided by the RTSC Application Programming Interface (API). DASCC scripts are interpreted at runtime.

SetConfig(Channel, config);

This command sends Set Channel Configuration with provided channel and configuration.

GetConfig(channel);

This command sends Get Channel Configuration for the requested channel.

StartAcquisition();

This command sends Set Acquisition Register, enabling acquisition on all channels by setting bit 0 of the Acquisition Register.

EndAcquisition();

This command sends Set Acquisition Register, disabling acquisition on all channels by clearing bit 0 of the Acquisition Register.

SingleStim(ChannelMask);

This command sends Set Stimulation Register, setting the Stimulation Register for a single cycle as specified by the ChannelMask parameter. The result is each channel specified will output its waveform once.

StartMultiStim(ChannelMask);

This command sends Set Stimulation Register, setting the Stimulation Register as specified by the ChannelMask parameter. The result is each channel specified will output its waveform repeatedly.

EndMultiStim(ChannelMask);

This command sends Set Stimulation Register, clearing the Stimulation Register as specified by the ChannelMask parameter. The result is each channel specified will stop outputting its waveform.

SetWaveform(channel,filename);

This command sends Set Waveform for the given channel. The provided filename is expected to have amplitude:time pairs in ascii format.

GetWaveform(channel);

This command sends Get Waveform for the given channel.

Sleep(time in milliseconds);

This command causes the script to wait for the provided number of milliseconds.

Appendix G

RTSC FPGA Configuration – Code

ADC_Capture.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 --
00019 -- Additional Comments:
00020 --
00021 ----------------------------------------------------------------------------------
00022 library IEEE;
00023 use IEEE.STD_LOGIC_1164.ALL;
00024 use IEEE.STD_LOGIC_ARITH.ALL;
00025 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00026
00027 ---- Uncomment the following library declaration if instantiating
00028 ---- any Xilinx primitives in this code.
00029 --library UNISIM;
00030 --use UNISIM.VComponents.all;
00031
00032 use work.ADC_Capture_pkg.all;
00033
00034 entity ADC_Capture is
00035     Port ( clk                  : in  STD_LOGIC;
00036            reset                  : in  STD_LOGIC;
00037
00038               -- ADC Serial Communication
00039               CS                      : out STD_LOGIC;
00040               sCLK                   : out STD_LOGIC;
00041               doutA                     : in  STD_LOGIC;
00042               doutB                     : in  STD_LOGIC;
00043
00044               -- ADC Control
00045               adcRANGE               : out STD_LOGIC;
00046               adcRESET               : out STD_LOGIC;
00047               adcSTDBY               : out STD_LOGIC;
00048               convStA                 : out STD_LOGIC;
00049               convStB                 : out STD_LOGIC;
00050               ovrSAMPLE                 : out STD_LOGIC_VECTOR(2 downto 0);
00051               refSEL                  : out STD_LOGIC;
00052               serSEL                  : out STD_LOGIC;
00053               Busy                   : in  STD_LOGIC;
00054
00055               -- Control Signals
00056               ADC_Capture_Start     : in  STD_LOGIC;
00057               ADC_Capture_Done   : out STD_LOGIC;
00058
00059               --
00060               Channel1_Data        : out STD_LOGIC_VECTOR(15 downto 0);
00061               Channel2_Data        : out STD_LOGIC_VECTOR(15 downto 0);
00062               Channel3_Data        : out STD_LOGIC_VECTOR(15 downto 0);
00063               Channel4_Data        : out STD_LOGIC_VECTOR(15 downto 0);
00064               Channel5_Data        : out STD_LOGIC_VECTOR(15 downto 0);
00065               Channel6_Data        : out STD_LOGIC_VECTOR(15 downto 0);
00066               Channel7_Data        : out STD_LOGIC_VECTOR(15 downto 0);
00067               Channel8_Data        : out STD_LOGIC_VECTOR(15 downto 0)
00068               );
00069 end ADC_Capture;
00070
00071 architecture Behavioral of ADC_Capture is
00072
00073 signal async_flags                   : STD_LOGIC_VECTOR(6 downto 0);
00074 signal count                           : STD_LOGIC_VECTOR(7 downto 0);
00075
00076 signal reset_counter               : STD_LOGIC_VECTOR(23 downto 0) := x"000000";
```

```
00077
00078 signal Channel1_data_reg           : STD_LOGIC_VECTOR(15 downto 0);
00079 signal Channel2_data_reg           : STD_LOGIC_VECTOR(15 downto 0);
00080 signal Channel3_data_reg           : STD_LOGIC_VECTOR(15 downto 0);
00081 signal Channel4_data_reg           : STD_LOGIC_VECTOR(15 downto 0);
00082 signal Channel5_data_reg           : STD_LOGIC_VECTOR(15 downto 0);
00083 signal Channel6_data_reg           : STD_LOGIC_VECTOR(15 downto 0);
00084 signal Channel7_data_reg           : STD_LOGIC_VECTOR(15 downto 0);
00085 signal Channel8_data_reg           : STD_LOGIC_VECTOR(15 downto 0);
00086
00087 component ADC_Capture_states
00088 Port (  clk                          : in  STD_LOGIC;
00089             rst_n                      : in  STD_LOGIC;
00090             ADC_Capture_start          : in  STD_LOGIC;            --start S.M. into
      motion
00091             Busy                            : in  STD_LOGIC;
00092             count                          : in  STD_LOGIC_VECTOR(7 downto 0);
00093             reset_counter              : in  STD_LOGIC_VECTOR(23 downto 0);
00094             async_flags                  : out STD_LOGIC_VECTOR(6 downto 0)  --flags to enable
      functions
00095             );
00096 end component;
00097
00098 begin
00099
00100 ADC_Capture_Done        <= async_flags(DONE_FLAG);
00101
00102 adcRANGE            <= '1';             -- 0: -5V to +5V, 1: -10V to +10V
00103 adcSTDBY            <= '1';             -- Standby mode not used
00104 ovrSAMPLE      <= "000";        -- allows rate of 50 kHz
00105 refSEL           <= '1';             -- internal reference
00106 serSEL           <= '1';          -- Serial Selected
00107 sCLK               <= clk;          -- route internal clock to sCLK
00108
00109 Channel1_Data <= Channel1_data_reg;
00110 Channel2_Data <= Channel2_data_reg;
00111 Channel3_Data <= Channel3_data_reg;
00112 Channel4_Data <= Channel4_data_reg;
00113 Channel5_Data <= Channel5_data_reg;
00114 Channel6_Data <= Channel6_data_reg;
00115 Channel7_Data <= Channel7_data_reg;
00116 Channel8_Data <= Channel8_data_reg;
00117
00118
00119 -- Channelx_Data
00120 process(clk, reset)
00121 begin
00122    if reset = '0' then
00123         Channel1_data_reg <= (others => '0');
00124         Channel2_data_reg <= (others => '0');
00125         Channel3_data_reg <= (others => '0');
00126         Channel4_data_reg <= (others => '0');
00127         Channel5_data_reg <= (others => '0');
00128         Channel6_data_reg <= (others => '0');
00129         Channel7_data_reg <= (others => '0');
00130         Channel8_data_reg <= (others => '0');
00131    elsif rising_edge(clk) and async_flags(CAPTURE_DATA_FLAG) = '1' then
00132         case count is
00133             when x"00" =>               Channel1_data_reg(15) <=
      doutA;
00134                                             Channel2_data_reg(15) <=
      doutB;          -- need to check what channel data comes from where and when
00135             when x"01" =>               Channel1_data_reg(14) <=
      doutA;
00136                                             Channel2_data_reg(14) <=
      doutB;
00137             when x"02" =>               Channel1_data_reg(13) <=
      doutA;
00138                                             Channel2_data_reg(13) <=
      doutB;
00139             when x"03" =>               Channel1_data_reg(12) <=
      doutA;
00140                                             Channel2_data_reg(12) <=
      doutB;
00141             when x"04" =>               Channel1_data_reg(11) <=
      doutA;
00142                                             Channel2_data_reg(11) <=
      doutB;
00143             when x"05" =>               Channel1_data_reg(10) <=
```

```
      doutA;
00144                                         Channel2_data_reg(10) <=
      doutB;
00145             when x"06" =>               Channel1_data_reg(9)  <=
      doutA;
00146                                         Channel2_data_reg(9)  <=
      doutB;
00147             when x"07" =>               Channel1_data_reg(8)  <=
      doutA;
00148                                         Channel2_data_reg(8)  <=
      doutB;
00149             when x"08" =>               Channel1_data_reg(7)  <=
      doutA;
00150                                         Channel2_data_reg(7)  <=
      doutB;
00151             when x"09" =>               Channel1_data_reg(6)  <=
      doutA;
00152                                         Channel2_data_reg(6)  <=
      doutB;
00153             when x"0A" =>               Channel1_data_reg(5)  <=
      doutA;
00154                                         Channel2_data_reg(5)  <=
      doutB;
00155             when x"0B" =>               Channel1_data_reg(4)  <=
      doutA;
00156                                         Channel2_data_reg(4)  <=
      doutB;
00157             when x"0C" =>               Channel1_data_reg(3)  <=
      doutA;
00158                                         Channel2_data_reg(3)  <=
      doutB;
00159             when x"0D" =>               Channel1_data_reg(2)  <=
      doutA;
00160                                         Channel2_data_reg(2)  <=
      doutB;
00161             when x"0E" =>               Channel1_data_reg(1)  <=
      doutA;
00162                                         Channel2_data_reg(1)  <=
      doutB;
00163             when x"0F" =>               Channel1_data_reg(0)  <=
      doutA;
00164                                         Channel2_data_reg(0)  <=
      doutB;
00165             when x"10" =>               Channel3_data_reg(15) <=
      doutA;
00166                                         Channel4_data_reg(15) <=
      doutB;       -- need to check what channel data comes from where and when
00167             when x"11" =>               Channel3_data_reg(14) <=
      doutA;
00168                                         Channel4_data_reg(14) <=
      doutB;
00169             when x"12" =>               Channel3_data_reg(13) <=
      doutA;
00170                                         Channel4_data_reg(13) <=
      doutB;
00171             when x"13" =>               Channel3_data_reg(12) <=
      doutA;
00172                                         Channel4_data_reg(12) <=
      doutB;
00173             when x"14" =>               Channel3_data_reg(11) <=
      doutA;
00174                                         Channel4_data_reg(11) <=
      doutB;
00175             when x"15" =>               Channel3_data_reg(10) <=
      doutA;
00176                                         Channel4_data_reg(10) <=
      doutB;
00177             when x"16" =>               Channel3_data_reg(9)  <=
      doutA;
00178                                         Channel4_data_reg(9)  <=
      doutB;
00179             when x"17" =>               Channel3_data_reg(8)  <=
      doutA;
00180                                         Channel4_data_reg(8)  <=
      doutB;
00181             when x"18" =>               Channel3_data_reg(7)  <=
      doutA;
00182                                         Channel4_data_reg(7)  <=
      doutB;
```

```
00183             when x"19" =>               Channel3_data_reg(6)  <=
      doutA;
00184                                             Channel4_data_reg(6)  <=
      doutB;
00185             when x"1A" =>               Channel3_data_reg(5)  <=
      doutA;
00186                                             Channel4_data_reg(5)  <=
      doutB;
00187             when x"1B" =>               Channel3_data_reg(4)  <=
      doutA;
00188                                             Channel4_data_reg(4)  <=
      doutB;
00189             when x"1C" =>               Channel3_data_reg(3)  <=
      doutA;
00190                                             Channel4_data_reg(3)  <=
      doutB;
00191             when x"1D" =>               Channel3_data_reg(2)  <=
      doutA;
00192                                             Channel4_data_reg(2)  <=
      doutB;
00193             when x"1E" =>               Channel3_data_reg(1)  <=
      doutA;
00194                                             Channel4_data_reg(1)  <=
      doutB;
00195             when x"1F" =>               Channel3_data_reg(0)  <=
      doutA;
00196                                             Channel4_data_reg(0)  <=
      doutB;
00197             when x"20" =>               Channel5_data_reg(15) <=
      doutA;
00198                                             Channel6_data_reg(15) <=
      doutB;          -- need to check what channel data comes from where and when
00199             when x"21" =>               Channel5_data_reg(14) <=
      doutA;
00200                                             Channel6_data_reg(14) <=
      doutB;
00201             when x"22" =>               Channel5_data_reg(13) <=
      doutA;
00202                                             Channel6_data_reg(13) <=
      doutB;
00203             when x"23" =>               Channel5_data_reg(12) <=
      doutA;
00204                                             Channel6_data_reg(12) <=
      doutB;
00205             when x"24" =>               Channel5_data_reg(11) <=
      doutA;
00206                                             Channel6_data_reg(11) <=
      doutB;
00207             when x"25" =>               Channel5_data_reg(10) <=
      doutA;
00208                                             Channel6_data_reg(10) <=
      doutB;
00209             when x"26" =>               Channel5_data_reg(9)  <=
      doutA;
00210                                             Channel6_data_reg(9)  <=
      doutB;
00211             when x"27" =>               Channel5_data_reg(8)  <=
      doutA;
00212                                             Channel6_data_reg(8)  <=
      doutB;
00213             when x"28" =>               Channel5_data_reg(7)  <=
      doutA;
00214                                             Channel6_data_reg(7)  <=
      doutB;
00215             when x"29" =>               Channel5_data_reg(6)  <=
      doutA;
00216                                             Channel6_data_reg(6)  <=
      doutB;
00217             when x"2A" =>               Channel5_data_reg(5)  <=
      doutA;
00218                                             Channel6_data_reg(5)  <=
      doutB;
00219             when x"2B" =>               Channel5_data_reg(4)  <=
      doutA;
00220                                             Channel6_data_reg(4)  <=
      doutB;
00221             when x"2C" =>               Channel5_data_reg(3)  <=
      doutA;
00222                                             Channel6_data_reg(3)  <=
```

```
      doutB;
00223             when x"2D" =>               Channel5_data_reg(2)  <=
      doutA;
00224                                             Channel6_data_reg(2)  <=
      doutB;
00225             when x"2E" =>               Channel5_data_reg(1)  <=
      doutA;
00226                                             Channel6_data_reg(1)  <=
      doutB;
00227             when x"2F" =>               Channel5_data_reg(0)  <=
      doutA;
00228                                             Channel6_data_reg(0)  <=
      doutB;
00229             when x"30" =>               Channel7_data_reg(15) <=
      doutA;
00230                                             Channel8_data_reg(15) <=
      doutB;          -- need to check what channel data comes from where and when
00231             when x"31" =>               Channel7_data_reg(14) <=
      doutA;
00232                                             Channel8_data_reg(14) <=
      doutB;
00233             when x"32" =>               Channel7_data_reg(13) <=
      doutA;
00234                                             Channel8_data_reg(13) <=
      doutB;
00235             when x"33" =>               Channel7_data_reg(12) <=
      doutA;
00236                                             Channel8_data_reg(12) <=
      doutB;
00237             when x"34" =>               Channel7_data_reg(11) <=
      doutA;
00238                                             Channel8_data_reg(11) <=
      doutB;
00239             when x"35" =>               Channel7_data_reg(10) <=
      doutA;
00240                                             Channel8_data_reg(10) <=
      doutB;
00241             when x"36" =>               Channel7_data_reg(9)  <=
      doutA;
00242                                             Channel8_data_reg(9)  <=
      doutB;
00243             when x"37" =>               Channel7_data_reg(8)  <=
      doutA;
00244                                             Channel8_data_reg(8)  <=
      doutB;
00245             when x"38" =>               Channel7_data_reg(7)  <=
      doutA;
00246                                             Channel8_data_reg(7)  <=
      doutB;
00247             when x"39" =>               Channel7_data_reg(6)  <=
      doutA;
00248                                             Channel8_data_reg(6)  <=
      doutB;
00249             when x"3A" =>               Channel7_data_reg(5)  <=
      doutA;
00250                                             Channel8_data_reg(5)  <=
      doutB;
00251             when x"3B" =>               Channel7_data_reg(4)  <=
      doutA;
00252                                             Channel8_data_reg(4)  <=
      doutB;
00253             when x"3C" =>               Channel7_data_reg(3)  <=
      doutA;
00254                                             Channel8_data_reg(3)  <=
      doutB;
00255             when x"3D" =>               Channel7_data_reg(2)  <=
      doutA;
00256                                             Channel8_data_reg(2)  <=
      doutB;
00257             when x"3E" =>               Channel7_data_reg(1)  <=
      doutA;
00258                                             Channel8_data_reg(1)  <=
      doutB;
00259             when x"3F" =>               Channel7_data_reg(0)  <=
      doutA;
00260                                             Channel8_data_reg(0)  <=
      doutB;
00261             when OTHERS =>
00262         end case;
```

```
00263     end if;
00264 end process;
00265 
00266 
00267 -- convST
00268 process(clk, reset)
00269 begin
00270    if reset = '0' then
00271         convStA <= '1';
00272         convStB <= '1';
00273    elsif rising_edge(clk) then
00274         if async_flags(CONVST_FLAG) = '1' then
00275             convStA <='0';
00276             convStB <='0';
00277         else
00278             convStA <= '1';
00279             convStB <= '1';
00280         end if;
00281     end if;
00282 end process;
00283 
00284 -- CS
00285 process(clk, reset)
00286 begin
00287    if reset = '0' then
00288         CS <= '1';
00289    elsif rising_edge(clk) then
00290         if async_flags(SET_CS_FLAG) = '1' then
00291             CS <='0';
00292         else
00293             CS <= '1';
00294         end if;
00295     end if;
00296 end process;
00297 
00298 -- adcRESET
00299 process(clk, reset)
00300 begin
00301    if reset = '0' then
00302         adcRESET <= '1';
00303    elsif rising_edge(clk) then
00304         if async_flags(ADC_RESET_FLAG) = '1' then
00305             adcRESET <='1';
00306         else
00307             adcRESET <= '0';
00308         end if;
00309     end if;
00310 end process;
00311 
00312 ------------------------------------------------------------------------------
00313 ---------------------- ADC_Capture_states ------------------------------------
00314 ------------------------------------------------------------------------------
00315 
00316 states : ADC_Capture_states
00317 port map(
00318     clk                             => clk,
00319     rst_n                           => reset,
00320     ADC_Capture_start               => ADC_Capture_Start,
00321     Busy                                 => Busy,
00322     count                               => count,
00323     reset_counter                   => reset_counter,
00324     async_flags                       => async_flags
00325     );
00326 
00327 ------------------------------------------------------------------------------
00328 ---------------------- Counter -----------------------------------------------
00329 ------------------------------------------------------------------------------
00330 
00331 -- count
00332 process(clk, reset)
00333 begin
00334    if reset = '0' then
00335         count <= (others => '0');
00336    elsif rising_edge(clk) then
00337         if async_flags(INC_COUNT_FLAG) = '1' then
00338             count <= count + 1;
00339         elsif async_flags(IDLE_FLAG) = '1' then
00340             count <= x"00";
00341         end if;
```

```
00342     end if;
00343 end process;
00344 
00345 
00346 -- reset_counter
00347 process(clk, reset)
00348 begin
00349    if reset = '0' then
00350         reset_counter <= (others => '0');
00351    elsif rising_edge(clk) then
00352         if async_flags(ADC_RESET_FLAG) = '1' then
00353             reset_counter <= reset_counter + 1;
00354         else
00355             reset_counter <= (others => '0');
00356         end if;
00357     end if;
00358 end process;
00359 
00360 
00361 end Behavioral;
00362 
```

ADC_Capture_main_states.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 --
00019 -- Additional Comments:
00020 --
00021 ----------------------------------------------------------------------------------
00022 library IEEE;
00023 use IEEE.STD_LOGIC_1164.ALL;
00024 use IEEE.numeric_std.all;
00025 --use IEEE.STD_LOGIC_ARITH.ALL;
00026 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00027 
00028 use work.ADC_Capture_pkg.all;
00029 
00030 ---- Uncomment the following library declaration if instantiating
00031 ---- any Xilinx primitives in this code.
00032 library UNISIM;
00033 use UNISIM.VComponents.all;
00034 
00035 entity ADC_Capture_states is
00036 Port (  clk                         : in  STD_LOGIC;
00037             rst_n                     : in  STD_LOGIC;
00038             ADC_Capture_start         : in  STD_LOGIC;            --start S.M. into
      motion
00039             Busy                            : in  STD_LOGIC;
00040             count                           : in  STD_LOGIC_VECTOR(7 downto 0);
00041             reset_counter             : in  STD_LOGIC_VECTOR(23 downto 0);
00042             async_flags                 : out STD_LOGIC_VECTOR(6 downto 0)  --flags to enable
      functions
00043             );
00044 end ADC_Capture_states;
00045 
00046 architecture Behavioral of ADC_Capture_states is
00047 
00048         --Control signals
00049 
```

```
00050 signal curr_state     : std_logic_vector(7 downto 0) := ADC_RESET;  -- FSM current state
00051 signal next_state     : std_logic_vector(7 downto 0) := ADC_RESET;  -- FSM next state
00052
00053 begin
00054 -------------------------------------------------------------------------------
00055         -- synchronous part of state machine here
00056 data_in_latch: process(clk, rst_n)
00057 begin
00058    if rst_n = '0' then
00059         curr_state <= (others => '0');
00060    elsif rising_edge(clk) then
00061         curr_state <= next_state;
00062    end if;
00063 end process;
00064
00065         -- async part of state machine to set function flags
00066 ADC_Capture_state: process(rst_n, curr_state)
00067 begin
00068    if rst_n = '0' then
00069         async_flags <= (others => '0');
00070    else
00071         async_flags <= (others => '0');
00072         case curr_state is
00073
00074             when ADC_RESET =>
00075                 async_flags(ADC_RESET_FLAG) <= '1';
00076
00077             when IDLE =>
00078                 async_flags(IDLE_FLAG) <= '1';           -- init
00079
00080             when CONVST =>
00081                 async_flags(CONVST_FLAG) <= '1';
00082
00083             when CAPTURE_1 =>
00084                 async_flags(SET_CS_FLAG) <= '1';
00085
00086             when CAPTURE_2 =>
00087                 async_flags(SET_CS_FLAG) <= '1';
00088                 async_flags(CAPTURE_DATA_FLAG) <= '1';
00089                 async_flags(INC_COUNT_FLAG) <= '1';
00090
00091             when FINISH =>
00092                 async_flags(DONE_FLAG) <= '1';           -- done flag
00093             when others =>
00094                 async_flags <= (others => '0');
00095         end case;
00096     end if;
00097 end process;
00098 -----------------------------------------------
00099         -- ADC_Capture state machine
00100 ADC_Capture_asynch_state: process(rst_n, curr_state,
      ADC_Capture_start, count, reset_counter)
00101 begin
00102    if rst_n = '0' then
00103         next_state <= ADC_RESET;
00104    else
00105         case curr_state is
00106             when ADC_RESET =>
00107                 if reset_counter = x"0249F0" then  -- 150000 200ns clock cycles = 30 ms
00108                     next_state <= IDLE;
00109                 else
00110                     next_state <= ADC_RESET;
00111                 end if;
00112
00113         when IDLE =>
00114                 if ADC_Capture_start = '1' then
00115                     next_state <= CONVST;
00116                 else
00117                     next_state <= IDLE;
00118                 end if;
00119
00120             when CONVST =>
00121                 next_state <= WAIT_BUSY_HIGH;
00122
00123             when WAIT_BUSY_HIGH =>
00124                 if Busy = '1' then --commment for internal data
00125                     next_state <= WAIT_BUSY_LOW;
00126                 else
00127                     next_state <= WAIT_BUSY_HIGH;
```

```
00128                 end if;
00129
00130             when WAIT_BUSY_LOW =>
00131                 if Busy = '0' then --commment for internal data
00132                     next_state <= CAPTURE_1;
00133                 else
00134                     next_state <= WAIT_BUSY_LOW;
00135                 end if;
00136
00137             when CAPTURE_1 =>
00138                 next_state <= CAPTURE_2;
00139
00140             when CAPTURE_2 =>
00141                 if count = 63 then -- 31 for 4 channel model
00142                     next_state <= FINISH;
00143                 else
00144                     next_state <= CAPTURE_2;
00145                 end if;
00146
00147
00148
00149             when FINISH =>
00150                     next_state <= IDLE;
00151
00152             when OTHERS =>
00153                 next_state <= IDLE;
00154         end case;
00155    end if;
00156 end process;
00157
00158
00159 end Behavioral;
00160
00161
```

ADC_Capture_pkg.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:     WMU - Thesis
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 --
00016 -- Dependencies:
00017 --
00018 -- Revision:
00019 -- Revision 0.01 - File Created
00020 -- Additional Comments:
00021 --
00022 ----------------------------------------------------------------------------------
00023 library IEEE;
00024 use IEEE.STD_LOGIC_1164.ALL;
00025 use IEEE.numeric_std.all;
00026 --use IEEE.STD_LOGIC_ARITH.ALL;
00027 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00028
00029 package ADC_Capture_pkg is
00030
00031
00032 constant ADC_RESET                              : STD_LOGIC_VECTOR(7 downto 0) := x"00";
00033
00034 constant IDLE                                       : STD_LOGIC_VECTOR(7 downto 0) := x"10";
00035
00036 constant CONVST                                   : STD_LOGIC_VECTOR(7 downto 0) := x"01";
00037 constant WAIT_BUSY_HIGH                         : STD_LOGIC_VECTOR(7 downto 0) := x"02";
00038 constant WAIT_BUSY_LOW                        : STD_LOGIC_VECTOR(7 downto 0) := x"03";
```

```
00039 constant CAPTURE_1                              : STD_LOGIC_VECTOR(7 downto 0) := x"04";
00040 constant CAPTURE_2                              : STD_LOGIC_VECTOR(7 downto 0) := x"05";
00041
00042 constant FINISH                                    : STD_LOGIC_VECTOR(7 downto 0) := x"FF";
00043
00044
00045
00046 constant IDLE_FLAG                              : integer := 0;
00047
00048 constant CONVST_FLAG                              : integer := 1;
00049 constant SET_CS_FLAG                              : integer := 2;
00050 constant CAPTURE_DATA_FLAG                    : integer := 3;
00051 constant INC_COUNT_FLAG                          : integer := 4;
00052 constant ADC_RESET_FLAG                          : integer := 5;
00053
00054 constant DONE_FLAG                              : integer := 6;
00055
00056
00057
00058 end ADC_Capture_pkg;
00059
00060 package body ADC_Capture_pkg is
00061
00062 end ADC_Capture_pkg;
00063
00064
```

ADC_Capture_tb.vhd

```
00001 --------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
00005 -- Create Date:   12:54:59 02/10/2012
00006 -- Design Name:
00007 -- Module Name:   C:/Users/Kyle/Desktop/Thesis/2-10-12/Cypress Compatible FPGA
      Code/Data_Acq_8Channel_ADC_Test/ADC_Capture_tb.vhd
00008 -- Project Name:  Data_Acq_8Channel_ADC_Test
00009 -- Target Device:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- VHDL Test Bench Created by ISE for module: ADC_Capture
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 -- Revision 0.01 - File Created
00019 -- Additional Comments:
00020 --
00021 -- Notes:
00022 -- This testbench has been automatically generated using types std_logic and
00023 -- std_logic_vector for the ports of the unit under test.  Xilinx recommends
00024 -- that these types always be used for the top-level I/O of a design in order
00025 -- to guarantee that the testbench will bind correctly to the post-implementation
00026 -- simulation model.
00027 --------------------------------------------------------------------------------
00028 LIBRARY ieee;
00029 USE ieee.std_logic_1164.ALL;
00030 USE ieee.std_logic_unsigned.all;
00031 USE ieee.numeric_std.ALL;
00032
00033 ENTITY ADC_Capture_tb IS
00034 END ADC_Capture_tb;
00035
00036 ARCHITECTURE behavior OF ADC_Capture_tb IS
00037
00038     -- Component Declaration for the Unit Under Test (UUT)
00039
00040     COMPONENT ADC_Capture
00041     PORT(
00042          clk : IN  std_logic;
00043          reset : IN  std_logic;
00044          CS : OUT  std_logic;
00045          adcRANGE : OUT  std_logic;
```

```
00046          adcRESET : OUT  std_logic;
00047          adcSTDBY : OUT  std_logic;
00048          convST : OUT  std_logic;
00049          ovrSAMPLE : OUT  std_logic_vector(2 downto 0);
00050          refSEL : OUT  std_logic;
00051          sCLK : OUT  std_logic;
00052          serSEL : OUT  std_logic;
00053          doutA : IN  std_logic;
00054          doutB : IN  std_logic;
00055          Busy : IN  std_logic;
00056          ADC_Capture_Start : IN  std_logic;
00057          ADC_Capture_Done : OUT  std_logic;
00058          Channel1_Data : OUT  std_logic_vector(15 downto 0);
00059          Channel2_Data : OUT  std_logic_vector(15 downto 0);
00060          Channel3_Data : OUT  std_logic_vector(15 downto 0);
00061          Channel4_Data : OUT  std_logic_vector(15 downto 0)
00062        );
00063     END COMPONENT;
00064
00065
00066    --Inputs
00067    signal clk : std_logic := '0';
00068    signal reset : std_logic := '0';
00069    signal doutA : std_logic := '0';
00070    signal doutB : std_logic := '0';
00071    signal Busy : std_logic := '0';
00072    signal ADC_Capture_Start : std_logic := '0';
00073
00074     --Outputs
00075    signal CS : std_logic;
00076    signal adcRANGE : std_logic;
00077    signal adcRESET : std_logic;
00078    signal adcSTDBY : std_logic;
00079    signal convST : std_logic;
00080    signal ovrSAMPLE : std_logic_vector(2 downto 0);
00081    signal refSEL : std_logic;
00082    signal sCLK : std_logic;
00083    signal serSEL : std_logic;
00084    signal ADC_Capture_Done : std_logic;
00085    signal Channel1_Data : std_logic_vector(15 downto 0);
00086    signal Channel2_Data : std_logic_vector(15 downto 0);
00087    signal Channel3_Data : std_logic_vector(15 downto 0);
00088    signal Channel4_Data : std_logic_vector(15 downto 0);
00089
00090    -- Clock period definitions
00091    constant clk_period : time := 200ns;  -- 5 MHz Clock
00092
00093 BEGIN
00094
00095     -- Instantiate the Unit Under Test (UUT)
00096    uut: ADC_Capture PORT MAP (
00097           clk => clk,
00098           reset => reset,
00099           CS  => CS,
00100           adcRANGE => adcRANGE,
00101           adcRESET => adcRESET,
00102           adcSTDBY => adcSTDBY,
00103           convST  => convST,
00104           ovrSAMPLE => ovrSAMPLE,
00105           refSEL => refSEL,
00106           sCLK => sCLK,
00107           serSEL => serSEL,
00108           doutA  => doutA,
00109           doutB  => doutB,
00110           Busy => Busy,
00111           ADC_Capture_Start => ADC_Capture_Start,
00112           ADC_Capture_Done => ADC_Capture_Done,
00113           Channel1_Data => Channel1_Data,
00114           Channel2_Data => Channel2_Data,
00115           Channel3_Data => Channel3_Data,
00116           Channel4_Data => Channel4_Data
00117         );
00118
00119    -- Clock process definitions
00120    clk_process :process
00121    begin
00122         clk <= '0';
00123         wait for clk_period/2;
00124         clk <= '1';
```

```
00125         wait for clk_period/2;
00126     end process;
00127 
00128 
00129     -- Stimulus process
00130     stim_proc: process
00131     begin
00132        -- hold reset state for 100ms.
00133          reset <= '0';
00134        wait for 1ms;
00135          reset <= '1';
00136          wait for 35ms;          -- wait for adcReset
00137 
00138          ADC_Capture_Start <= '1';
00139          wait for clk_period;
00140          ADC_Capture_Start <= '0';
00141          Busy <= '1';
00142          wait for clk_period*2;
00143          Busy <= '0';
00144          doutA <= '1';
00145 
00146 
00147          wait for 50us;
00148          ADC_Capture_Start <= '1';
00149          wait for clk_period;
00150          ADC_Capture_Start <= '0';
00151          Busy <= '1';
00152          wait for clk_period*2;
00153          Busy <= '0';
00154          doutA <= '0';
00155 
00156 
00157          wait for 50us;
00158          ADC_Capture_Start <= '1';
00159          wait for clk_period;
00160          ADC_Capture_Start <= '0';
00161          Busy <= '1';
00162          wait for clk_period*2;
00163          Busy <= '0';
00164          doutA <= '1';
00165 
00166 
00167 
00168 
00169 
00170 
00171 
00172       wait for clk_period*1000;
00173 
00174       -- insert stimulus here
00175 
00176       wait;
00177    end process;
00178 
00179 END;
```

ADC_Module.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
00005 -- Create Date:    19:44:35 01/27/2012
00006 -- Design Name:
00007 -- Module Name:    ADC_Module - Behavioral
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- Dependencies:
00014 --
00015 -- Revision:
00016 -- Revision 0.01 - File Created
00017 -- Additional Comments:
```

```
00018 --
00019 ----------------------------------------------------------------------------------
00020 library IEEE;
00021 use IEEE.STD_LOGIC_1164.ALL;
00022 use IEEE.STD_LOGIC_ARITH.ALL;
00023 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00024
00025 ---- Uncomment the following library declaration if instantiating
00026 ---- any Xilinx primitives in this code.
00027 --library UNISIM;
00028 --use UNISIM.VComponents.all;
00029
00030 use work.ADC_Module_pkg.all;
00031
00032 entity ADC_Module is
00033     Port ( clk                  : in  STD_LOGIC;
00034            reset                  : in  STD_LOGIC;
00035
00036               -- USB FIFO Signals
00037               FIFO_DIN              : out STD_LOGIC_VECTOR (7 downto 0);        --Data going into
      FIFO
00038               FIFO_WR_EN          : out STD_LOGIC;
00039               FIFO_WR_CLK            : out STD_LOGIC;
00040               FIFO_FULL                : in  STD_LOGIC;
00041               FIFO_ALMOST_FULL  : in  STD_LOGIC;
00042               FIFO_PROG_FULL      : in  STD_LOGIC;
00043
00044                -- ADC Serial Communication Signals
00045               CS                      : out STD_LOGIC;
00046               sCLK                   : out STD_LOGIC;
00047               doutA                     : in  STD_LOGIC;
00048               doutB                     : in  STD_LOGIC;
00049
00050               -- ADC Control Signals
00051               adcRANGE              : out STD_LOGIC;
00052               adcRESET              : out STD_LOGIC;
00053               adcSTDBY              : out STD_LOGIC;
00054               convStA                : out STD_LOGIC;
00055               convStB                : out STD_LOGIC;
00056               ovrSAMPLE                : out STD_LOGIC_VECTOR(2 downto 0);
00057               refSEL                  : out STD_LOGIC;
00058               serSEL                  : out STD_LOGIC;
00059               Busy                   : in  STD_LOGIC
00060               );
00061 end ADC_Module;
00062
00063 architecture Behavioral of ADC_Module is
00064
00065 signal test_data_counter          : std_logic_vector(15 downto 0) := x"0000";
00066 signal timestamp_counter          : std_logic_vector(31 downto 0) := x"00000000";
00067
00068 signal async_flags                   : STD_LOGIC_VECTOR(5 downto 0);
00069 signal count                           : STD_LOGIC_VECTOR(9 downto 0);
00070
00071 signal start_sig                       : STD_LOGIC := '0';
00072 signal data_sig                      : STD_LOGIC_VECTOR(7 downto 0);
00073
00074 ----------------------------------------------------------------------------------
00075 ---------------------- ADC_Module_states ---------------------------------------
00076 ----------------------------------------------------------------------------------
00077
00078 component ADC_Module_states
00079 Port (  clk                          : in  STD_LOGIC;
00080            rst_n                       : in  STD_LOGIC;
00081            ADC_Module_start            : in  STD_LOGIC;            --start S.M. into
      motion
00082            FIFO_Full_Flag                : in  STD_LOGIC;
00083            count                          : in  STD_LOGIC_VECTOR(9 downto 0);
00084            async_flags                  : out STD_LOGIC_VECTOR(5 downto 0)  --flags to enable
      functions
00085            );
00086 end component;
00087
00088 ----------------------------------------------------------------------------------
00089 ---------------------- ADC_Capture  --------------------------------------------
00090 ----------------------------------------------------------------------------------
00091 signal ADC_Capture_Done            : STD_LOGIC;
00092 signal ADC_Channel1_Data          : STD_LOGIC_VECTOR(15 downto 0);
00093 signal ADC_Channel2_Data          : STD_LOGIC_VECTOR(15 downto 0);
```

```
00094 signal ADC_Channel3_Data           : STD_LOGIC_VECTOR(15 downto 0);
00095 signal ADC_Channel4_Data           : STD_LOGIC_VECTOR(15 downto 0);
00096 signal ADC_Channel5_Data           : STD_LOGIC_VECTOR(15 downto 0);
00097 signal ADC_Channel6_Data           : STD_LOGIC_VECTOR(15 downto 0);
00098 signal ADC_Channel7_Data           : STD_LOGIC_VECTOR(15 downto 0);
00099 signal ADC_Channel8_Data           : STD_LOGIC_VECTOR(15 downto 0);
00100
00101 component ADC_Capture
00102     Port ( clk                   : in  STD_LOGIC;
00103            reset                 : in  STD_LOGIC;
00104               CS                    : out STD_LOGIC;
00105               adcRANGE              : out STD_LOGIC;
00106               adcRESET              : out STD_LOGIC;
00107               adcSTDBY              : out STD_LOGIC;
00108               convStA               : out STD_LOGIC;
00109               convStB               : out STD_LOGIC;
00110               ovrSAMPLE             : out STD_LOGIC_VECTOR(2 downto 0);
00111               refSEL                : out STD_LOGIC;
00112               sCLK                  : out STD_LOGIC;
00113               serSEL                : out STD_LOGIC;
00114               doutA                 : in  STD_LOGIC;
00115               doutB                 : in  STD_LOGIC;
00116               Busy                  : in  STD_LOGIC;
00117
00118               ADC_Capture_Start     : in  STD_LOGIC;
00119               ADC_Capture_Done  : out STD_LOGIC;
00120               Channel1_Data         : out STD_LOGIC_VECTOR(15 downto 0);
00121               Channel2_Data         : out STD_LOGIC_VECTOR(15 downto 0);
00122               Channel3_Data         : out STD_LOGIC_VECTOR(15 downto 0);
00123               Channel4_Data         : out STD_LOGIC_VECTOR(15 downto 0);
00124               Channel5_Data         : out STD_LOGIC_VECTOR(15 downto 0);
00125               Channel6_Data         : out STD_LOGIC_VECTOR(15 downto 0);
00126               Channel7_Data         : out STD_LOGIC_VECTOR(15 downto 0);
00127               Channel8_Data         : out STD_LOGIC_VECTOR(15 downto 0)
00128               );
00129 end component;
00130
00131 ----------------------------------------------------------------------------
00132 ---------------------- Clock_Divider ---------------------------------------
00133 ----------------------------------------------------------------------------
00134
00135 signal Clock_Divider_200ns       : STD_LOGIC;
00136 signal ADC_divide_count          : STD_LOGIC_VECTOR(7 downto 0);
00137
00138 component Clock_Divider
00139      Port ( clk_in : in  STD_LOGIC;
00140             reset : in  STD_LOGIC;
00141                divide_count : in STD_LOGIC_VECTOR(7 downto 0);
00142             clk_out : out  STD_LOGIC);
00143 end component;
00144
00145
00146
00147 begin
00148
00149 FIFO_WR_CLK <= clk;
00150
00151 -- FIFO_WR_EN
00152 process(clk, reset)
00153 begin
00154    if reset = '0' then
00155         FIFO_WR_EN <= '0';
00156    elsif rising_edge(clk) then
00157         if async_flags(SET_WR_EN_FLAG) = '1' then
00158             FIFO_WR_EN <= '1';
00159         else
00160             FIFO_WR_EN <= '0';
00161         end if;
00162     end if;
00163 end process;
00164
00165 -- FIFO_DIN
00166 process(clk, reset)
00167 begin
00168    if reset = '0' then
00169         FIFO_DIN <= x"FF";
00170    elsif rising_edge(clk) and async_flags(SET_DATA_FLAG) = '1' then
00171         case count is
00172             when "00" & x"00" =>        FIFO_DIN <= x"A5";
```

```
00173             when "00" & x"01" =>        FIFO_DIN <= x"5A";
00174             when "00" & x"02" =>        FIFO_DIN <= timestamp_counter(7 downto 0);
00175             when "00" & x"03" =>        FIFO_DIN <= timestamp_counter(15 downto 8)
    ;
00176             when "00" & x"04" =>        FIFO_DIN <= timestamp_counter(23 downto 16
    );
00177             when "00" & x"05" =>        FIFO_DIN <= timestamp_counter(31 downto 24
    );
00178
00179             --channel 1 data
00180             when "00" & x"06" =>        FIFO_DIN <= x"01";
00181             when "00" & x"07" =>        FIFO_DIN <= ADC_Channel1_Data(7 downto 0);
00182             when "00" & x"08" =>        FIFO_DIN <= ADC_Channel1_Data(15 downto 8)
    ;
00183
00184             --channel 2 data
00185             when "00" & x"09" =>        FIFO_DIN <= x"02";
00186             when "00" & x"0A" =>        FIFO_DIN <= ADC_Channel2_Data(7 downto 0);
00187             when "00" & x"0B" =>        FIFO_DIN <= ADC_Channel2_Data(15 downto 8)
    ;
00188
00189             --channel 3 data
00190             when "00" & x"0C" =>        FIFO_DIN <= x"03";
00191             when "00" & x"0D" =>        FIFO_DIN <= ADC_Channel3_Data(7 downto 0);
00192             when "00" & x"0E" =>        FIFO_DIN <= ADC_Channel3_Data(15 downto 8)
    ;
00193
00194             --channel 4 data
00195             when "00" & x"0F" =>        FIFO_DIN <= x"04";
00196             when "00" & x"10" =>        FIFO_DIN <= ADC_Channel4_Data(7 downto 0);
00197             when "00" & x"11" =>        FIFO_DIN <= ADC_Channel4_Data(15 downto 8)
    ;
00198
00199             --channel 5 data
00200             when "00" & x"12" =>        FIFO_DIN <= x"05";
00201             when "00" & x"13" =>        FIFO_DIN <= ADC_Channel5_Data(7 downto 0);
00202             when "00" & x"14" =>        FIFO_DIN <= ADC_Channel5_Data(15 downto 8)
    ;
00203
00204             --channel 6 data
00205             when "00" & x"15" =>        FIFO_DIN <= x"06";
00206             when "00" & x"16" =>        FIFO_DIN <= ADC_Channel6_Data(7 downto 0);
00207             when "00" & x"17" =>        FIFO_DIN <= ADC_Channel6_Data(15 downto 8)
    ;
00208
00209             --channel 7 data
00210             when "00" & x"18" =>        FIFO_DIN <= x"07";
00211             when "00" & x"19" =>        FIFO_DIN <= ADC_Channel7_Data(7 downto 0);
00212             when "00" & x"1A" =>        FIFO_DIN <= ADC_Channel7_Data(15 downto 8)
    ;
00213
00214             --channel 8 data
00215             when "00" & x"1B" =>        FIFO_DIN <= x"08";
00216             when "00" & x"1C" =>        FIFO_DIN <= ADC_Channel8_Data(7 downto 0);
00217             when "00" & x"1D" =>        FIFO_DIN <= ADC_Channel8_Data(15 downto 8)
    ;
00218
00219             --when "00" & x"0F" =>      FIFO_DIN <= x"FF";
00220             when "00" & x"1F" =>        FIFO_DIN <= x"FF";
00221             when OTHERS =>              FIFO_DIN <= data_sig;
00222         end case;
00223     end if;
00224 end process;
00225
00226 -- test_data_counter
00227 process(clk, reset)
00228 begin
00229    if reset = '0' then
00230         test_data_counter <= (others => '0');
00231         start_sig <= '0';
00232         data_sig <= x"00";
00233    elsif rising_edge(clk) then
00234         if test_data_counter = x"044C" and FIFO_PROG_FULL = '0' then
00235             test_data_counter <= (others => '0');
00236             start_sig <= '1';
00237             data_sig <= data_sig + 1;
00238         elsif test_data_counter = x"044C" then
00239             test_data_counter <= (others => '0');
00240             data_sig <= data_sig + 1;
```

```
00241         elsif test_data_counter = x"000A" then
00242             test_data_counter <= test_data_counter + 1;
00243             start_sig <= '0';
00244         else
00245             test_data_counter <= test_data_counter + 1;
00246         end if;
00247
00248     end if;
00249 end process;
00250
00251 -- timestamp_counter
00252 process(clk, reset)
00253 begin
00254    if reset = '0' then
00255         timestamp_counter <= (others => '0');
00256    elsif rising_edge(clk) then
00257         timestamp_counter <= timestamp_counter + 1;
00258     end if;
00259 end process;
00260
00261
00262 ------------------------------------------------------------------------------
00263 ---------------------- Clock_Divider --------------------------------------
00264 ------------------------------------------------------------------------------
00265 ADC_divide_count <= x"04";
00266
00267 CLK_200ns : Clock_Divider
00268 port map(
00269     clk_in                          => clk,
00270     reset                            => reset,
00271     divide_count                    => ADC_divide_count,
00272     clk_out                             => Clock_Divider_200ns
00273     );
00274
00275 ------------------------------------------------------------------------------
00276 ---------------------- ADC_Capture  --------------------------------------
00277 ------------------------------------------------------------------------------
00278
00279 AD7606_4 : ADC_Capture
00280 port map(
00281     clk                             => Clock_Divider_200ns,
00282     reset                           => reset,
00283     CS                                  => CS,
00284     adcRANGE                            => adcRANGE,
00285     adcRESET                            => adcRESET,
00286     adcSTDBY                            => adcSTDBY,
00287     convStA                           => convStA,
00288     convStB                           => convStB,
00289     ovrSAMPLE                       => ovrSAMPLE,
00290     refSEL                             => refSEL,
00291     sCLK                                => sCLK,
00292     serSEL                            => serSEL,
00293     doutA                              => doutA,
00294     doutB                              => doutB,
00295     Busy                                => Busy,
00296
00297     ADC_Capture_Start               => start_sig,
00298     ADC_Capture_Done                => ADC_Capture_Done,
00299     Channel1_Data                      => ADC_Channel1_Data,
00300     Channel2_Data                      => ADC_Channel2_Data,
00301     Channel3_Data                      => ADC_Channel3_Data,
00302     Channel4_Data                      => ADC_Channel4_Data,
00303     Channel5_Data                      => ADC_Channel5_Data,
00304     Channel6_Data                      => ADC_Channel6_Data,
00305     Channel7_Data                      => ADC_Channel7_Data,
00306     Channel8_Data                      => ADC_Channel8_Data
00307     );
00308
00309 ------------------------------------------------------------------------------
00310 ---------------------- ADC_Module_states ---------------------------------
00311 ------------------------------------------------------------------------------
00312
00313 states : ADC_Module_states
00314 port map(
00315     clk                             => clk,
00316     rst_n                           => reset,
00317     ADC_Module_start                => ADC_Capture_Done,
00318     FIFO_Full_Flag                     => FIFO_PROG_FULL,
00319     count                               => count,
```

```
00320     async_flags                     => async_flags
00321     );
00322
00323 ----------------------------------------------------------------------------
00324 ---------------------- Counter -------------------------------------------
00325 ----------------------------------------------------------------------------
00326
00327 -- count
00328 process(clk, reset)
00329 begin
00330    if reset = '0' then
00331         count <= (others => '0');
00332    elsif rising_edge(clk) then
00333         if async_flags(INC_COUNT_FLAG) = '1' then
00334             count <= count + 1;
00335         elsif async_flags(IDLE_FLAG) = '1' then
00336             count <= "00" & x"00";
00337         end if;
00338     end if;
00339 end process;
00340
00341
00342 end Behavioral;
00343
```

ADC_Module_main_states.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 --
00019 -- Additional Comments:
00020 --
00021 ----------------------------------------------------------------------------------
00022 library IEEE;
00023 use IEEE.STD_LOGIC_1164.ALL;
00024 use IEEE.numeric_std.all;
00025 --use IEEE.STD_LOGIC_ARITH.ALL;
00026 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00027
00028 use work.ADC_Module_pkg.all;
00029
00030 ---- Uncomment the following library declaration if instantiating
00031 ---- any Xilinx primitives in this code.
00032 library UNISIM;
00033 use UNISIM.VComponents.all;
00034
00035 entity ADC_Module_states is
00036 Port (  clk                         : in  STD_LOGIC;
00037             rst_n                     : in  STD_LOGIC;
00038             ADC_Module_start          : in  STD_LOGIC;            --start S.M. into
      motion
00039             FIFO_Full_Flag              : in  STD_LOGIC;
00040             count                       : in  STD_LOGIC_VECTOR(9 downto 0);
00041             async_flags                 : out STD_LOGIC_VECTOR(5 downto 0)  --flags to enable
      functions
00042             );
00043 end ADC_Module_states;
00044
00045 architecture Behavioral of ADC_Module_states is
00046
```

```
00047         --Control signals
00048
00049 signal curr_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM current state
00050 signal next_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM next state
00051
00052 begin
00053 -------------------------------------------------------------------------------
00054         -- synchronous part of state machine here
00055 data_in_latch: process(clk, rst_n)
00056 begin
00057    if rst_n = '0' then
00058         curr_state <= (others => '0');
00059    elsif rising_edge(clk) then
00060         curr_state <= next_state;
00061    end if;
00062 end process;
00063
00064         -- async part of state machine to set function flags
00065 ADC_Module_state: process(rst_n, curr_state)
00066 begin
00067    if rst_n = '0' then
00068         async_flags <= (others => '0');
00069    else
00070         async_flags <= (others => '0');
00071         case curr_state is
00072             when IDLE =>
00073                 async_flags(IDLE_FLAG) <= '1';           -- init
00074
00075
00076             when FIFO_WR_2 =>
00077                 async_flags(SET_WR_EN_FLAG) <= '1';
00078                 async_flags(SET_DATA_FLAG) <= '1';
00079                 async_flags(INC_COUNT_FLAG) <= '1';
00080
00081
00082             when FINISH =>
00083                 async_flags(DONE_FLAG) <= '1';           -- done flag
00084             when others =>
00085                 async_flags <= (others => '0');
00086         end case;
00087     end if;
00088 end process;
00089 -----------------------------------------------
00090         -- ADC_Module state machine
00091 ADC_Module_asynch_state: process(rst_n, curr_state,
      ADC_Module_start, count, FIFO_Full_Flag)
00092 begin
00093    if rst_n = '0' then
00094         next_state <= (others => '0');
00095    else
00096         case curr_state is
00097         when IDLE =>
00098                 --if Synch_Slave_FIFO_start = '1' and FlagA = '1' then
00099                 if ADC_Module_start = '1' then
00100                     next_state <= FIFO_WR_1;
00101                 else
00102                     next_state <= IDLE;
00103                 end if;
00104
00105             when FIFO_WR_1 =>
00106                 if FIFO_Full_Flag = '0' then  -- mapped to prog_full - 32256
00107                     next_state <= FIFO_WR_2;
00108                 else
00109                     next_state <= FIFO_WR_1;
00110                 end if;
00111             when FIFO_WR_2 =>
00112                 if count = 31 then
00113                     next_state <= FINISH;
00114                 else
00115                     next_state <= FIFO_WR_2;
00116                 end if;
00117
00118             when FINISH =>
00119                     next_state <= IDLE;
00120
00121             when OTHERS =>
00122                 next_state <= IDLE;
00123         end case;
00124    end if;
```

```
00125 end process;
00126
00127
00128 end Behavioral;
00129
```

ADC_Module_pkg.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 --
00019 -- Additional Comments:
00020 --
00021 ----------------------------------------------------------------------------------
00022 library IEEE;
00023 use IEEE.STD_LOGIC_1164.ALL;
00024 use IEEE.numeric_std.all;
00025 --use IEEE.STD_LOGIC_ARITH.ALL;
00026 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00027
00028 package ADC_Module_pkg is
00029
00030
00031 constant IDLE                                    : STD_LOGIC_VECTOR(7 downto 0) := x"00";
00032
00033 constant FIFO_WR_1                             : STD_LOGIC_VECTOR(7 downto 0) := x"10";
00034 constant FIFO_WR_2                             : STD_LOGIC_VECTOR(7 downto 0) := x"11";
00035
00036
00037
00038
00039
00040 constant FINISH                                  : STD_LOGIC_VECTOR(7 downto 0) := x"FF";
00041
00042
00043
00044 constant IDLE_FLAG                             : integer := 0;
00045
00046 constant SET_WR_EN_FLAG                          : integer := 1;
00047 constant SET_DATA_FLAG                         : integer := 2;
00048 --constant SET_PKTEND_FLAG                      : integer := 3;
00049 constant INC_COUNT_FLAG                          : integer := 4;
00050
00051 constant DONE_FLAG                             : integer := 5;
00052
00053
00054
00055 end ADC_Module_pkg;
00056
00057 package body ADC_Module_pkg is
00058
00059 end ADC_Module_pkg;
00060
00061
```

Arbiter.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
00005 -- Create Date:    19:44:35 01/27/2012
00006 -- Design Name:
00007 -- Module Name:    Arbiter - Behavioral
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- Dependencies:
00014 --
00015 -- Revision:
00016 -- Revision 0.01 - File Created
00017 -- Additional Comments:
00018 --
00019 ----------------------------------------------------------------------------------
00020 library IEEE;
00021 use IEEE.STD_LOGIC_1164.ALL;
00022 use IEEE.STD_LOGIC_ARITH.ALL;
00023 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00024
00025 ---- Uncomment the following library declaration if instantiating
00026 ---- any Xilinx primitives in this code.
00027 --library UNISIM;
00028 --use UNISIM.VComponents.all;
00029
00030 use work.Arbiter_pkg.all;
00031
00032 entity Arbiter is
00033     Port ( clk                      : in  STD_LOGIC;
00034            reset                    : in  STD_LOGIC;
00035               Bus_Request             : in  STD_LOGIC_VECTOR(7 downto 0);
00036               Bus_Busy                : in  STD_LOGIC;
00037
00038               Bus_Grant                   : out  STD_LOGIC_VECTOR(7 downto 0)
00039               );
00040 end Arbiter;
00041
00042 architecture Behavioral of Arbiter is
00043
00044 signal async_flags                  : STD_LOGIC_VECTOR(3 downto 0);
00045 signal count                        : STD_LOGIC_VECTOR(7 downto 0);
00046
00047 ----------------------------------------------------------------------------------
00048 ---------------------- Arbiter_states -------------------------------------
00049 ----------------------------------------------------------------------------------
00050
00051 component Arbiter_states
00052 Port (  clk                         : in  STD_LOGIC;
00053            rst_n                    : in  STD_LOGIC;
00054            count                       : in  STD_LOGIC_VECTOR(7 downto 0);
00055            Bus_Request              : in  STD_LOGIC_VECTOR(7 downto 0);
00056            Bus_Busy                    : in  STD_LOGIC;
00057            async_flags              : out STD_LOGIC_VECTOR(3 downto 0)  --flags to enable
      functions
00058            );
00059 end component;
00060
00061
00062 begin
00063
00064 -- count
00065 process(clk, reset)
00066 begin
00067    if reset = '0' then
00068         Bus_Grant <= (others => '0');
00069    elsif rising_edge(clk) then
00070         if async_flags(SET_BG_FLAG) = '1' then
00071            if Bus_Request(0) = '1' then
00072                Bus_Grant(0) <= '1';
00073            elsif Bus_Request(1) = '1' then
00074                Bus_Grant(1) <= '1';
00075            elsif Bus_Request(2) = '1' then
```

```
00076                 Bus_Grant(2) <= '1';
00077             elsif Bus_Request(3) = '1' then
00078                 Bus_Grant(3) <= '1';
00079             elsif Bus_Request(4) = '1' then
00080                 Bus_Grant(4) <= '1';
00081             elsif Bus_Request(5) = '1' then
00082                 Bus_Grant(5) <= '1';
00083             elsif Bus_Request(6) = '1' then
00084                 Bus_Grant(6) <= '1';
00085             elsif Bus_Request(7) = '1' then
00086                 Bus_Grant(7) <= '1';
00087             end if;
00088
00089         elsif async_flags(CLEAR_BG_FLAG) = '1' then
00090             Bus_Grant <= (others => '0');
00091         end if;
00092     end if;
00093 end process;
00094
00095 ---------------------------------------------------------------------------
00096 ---------------------- Arbiter_states -------------------------------------
00097 ---------------------------------------------------------------------------
00098
00099 states : Arbiter_states
00100 port map(
00101     clk                             => clk,
00102     rst_n                          => reset,
00103     count                              => count,
00104     Bus_Request                     => Bus_Request,
00105     Bus_Busy                              => Bus_Busy,
00106     async_flags                     => async_flags
00107     );
00108
00109 ---------------------------------------------------------------------------
00110 ---------------------- Counter --------------------------------------------
00111 ---------------------------------------------------------------------------
00112
00113 -- count
00114 process(clk, reset)
00115 begin
00116    if reset = '0' then
00117         count <= (others => '0');
00118    elsif rising_edge(clk) then
00119         if async_flags(INC_COUNT_FLAG) = '1' then
00120             count <= count + 1;
00121         elsif async_flags(IDLE_FLAG) = '1' then
00122             count <= x"00";
00123         end if;
00124     end if;
00125 end process;
00126
00127
00128 end Behavioral;
00129
```

Arbiter_main_states.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 --
```

```
00019 -- Additional Comments:
00020 --
00021 ----------------------------------------------------------------------------------
00022 library IEEE;
00023 use IEEE.STD_LOGIC_1164.ALL;
00024 use IEEE.numeric_std.all;
00025 --use IEEE.STD_LOGIC_ARITH.ALL;
00026 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00027
00028 use work.Arbiter_pkg.all;
00029
00030 ---- Uncomment the following library declaration if instantiating
00031 ---- any Xilinx primitives in this code.
00032 library UNISIM;
00033 use UNISIM.VComponents.all;
00034
00035 entity Arbiter_states is
00036 Port (  clk                         : in  STD_LOGIC;
00037             rst_n                     : in  STD_LOGIC;
00038             count                         : in  STD_LOGIC_VECTOR(7 downto 0);
00039             Bus_Request                   : in  STD_LOGIC_VECTOR(7 downto 0);
00040             Bus_Busy                        : in  STD_LOGIC;
00041             async_flags                   : out STD_LOGIC_VECTOR(3 downto 0)  --flags to enable
      functions
00042             );
00043 end Arbiter_states;
00044
00045 architecture Behavioral of Arbiter_states is
00046
00047         --Control signals
00048
00049 signal curr_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM current state
00050 signal next_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM next state
00051
00052 begin
00053 ----------------------------------------------------------------------------------
00054         -- synchronous part of state machine here
00055 data_in_latch: process(clk, rst_n)
00056 begin
00057    if rst_n = '0' then
00058         curr_state <= IDLE;
00059    elsif rising_edge(clk) then
00060         curr_state <= next_state;
00061    end if;
00062 end process;
00063
00064         -- async part of state machine to set function flags
00065 Arbiter_state: process(rst_n, curr_state)
00066 begin
00067    if rst_n = '0' then
00068         async_flags <= (others => '0');
00069    else
00070         async_flags <= (others => '0');
00071         case curr_state is
00072
00073             when IDLE =>
00074                 async_flags(IDLE_FLAG) <= '1';
00075
00076             when SET_BUS_GRANT =>
00077                 async_flags(SET_BG_FLAG) <= '1';
00078
00079             when WAIT1 =>
00080                 async_flags(INC_COUNT_FLAG) <= '1';
00081
00082             when CLEAR_BUS_GRANT =>
00083                 async_flags(CLEAR_BG_FLAG) <= '1';
00084
00085             when others =>
00086                 async_flags <= (others => '0');
00087         end case;
00088     end if;
00089 end process;
00090 ------------------------------------------------
00091         -- Arbiter state machine
00092 Arbiter_asynch_state: process(rst_n, curr_state,
      Bus_Request, Bus_Busy, count)
00093 begin
00094    if rst_n = '0' then
00095         next_state <= IDLE;
```

```
00096    else
00097         case curr_state is
00098
00099             when IDLE =>
00100                 if Bus_Request /= 0 then
00101                     next_state <= SET_BUS_GRANT;
00102                 else
00103                     next_state <= IDLE;
00104                 end if;
00105
00106             when SET_BUS_GRANT =>
00107                 next_state <= WAIT1;
00108
00109             when WAIT1 =>
00110                 if count = 3 then
00111                     next_state <= CLEAR_BUS_GRANT;
00112                 else
00113                     next_state <= WAIT1;
00114                 end if;
00115
00116             when CLEAR_BUS_GRANT =>
00117                 next_state <= WAIT_FOR_BUSY;
00118
00119             when WAIT_FOR_BUSY =>
00120                 if Bus_Busy = '0' then
00121                     next_state <= IDLE;
00122                 else
00123                     next_state <= WAIT_FOR_BUSY;
00124                 end if;
00125
00126
00127             when OTHERS =>
00128                 next_state <= IDLE;
00129         end case;
00130    end if;
00131 end process;
00132
00133
00134 end Behavioral;
00135
```

Arbiter_pkg.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 --
00019 -- Additional Comments:
00020 --
00021 ----------------------------------------------------------------------------------
00022 library IEEE;
00023 use IEEE.STD_LOGIC_1164.ALL;
00024 use IEEE.numeric_std.all;
00025 --use IEEE.STD_LOGIC_ARITH.ALL;
00026 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00027
00028 package Arbiter_pkg is
00029
00030
00031 constant IDLE                               : STD_LOGIC_VECTOR(7 downto 0) := x"00";
00032
```

```
00033 constant SET_BUS_GRANT                : STD_LOGIC_VECTOR(7 downto 0) := x"01";
00034 constant WAIT1                           : STD_LOGIC_VECTOR(7 downto 0) := x"02";
00035 constant CLEAR_BUS_GRANT            : STD_LOGIC_VECTOR(7 downto 0) := x"03";
00036 constant WAIT_FOR_BUSY               : STD_LOGIC_VECTOR(7 downto 0) := x"04";
00037
00038
00039 constant INC_COUNT_FLAG                  : integer := 0;
00040 constant SET_BG_FLAG                    : integer := 1;
00041 constant CLEAR_BG_FLAG                : integer := 2;
00042 constant IDLE_FLAG                      : integer := 3;
00043
00044
00045
00046
00047
00048
00049 end Arbiter_pkg;
00050
00051 package body Arbiter_pkg is
00052
00053 end Arbiter_pkg;
00054
00055
```

Arbiter_tb.vhd

```
00001 --------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
00005 -- Create Date:   17:07:59 05/11/2012
00006 -- Design Name:
00007 -- Module Name:   C:/Users/Kyle/Desktop/SVN_Thesis/FPGA Code/Data_Acq_and_Stim_Ver_0.5/Arbiter_tb.vhd
00008 -- Project Name:  Data_Acq_and_Stim_Ver_0.5
00009 -- Target Device:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- VHDL Test Bench Created by ISE for module: Arbiter
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 -- Revision 0.01 - File Created
00019 -- Additional Comments:
00020 --
00021 -- Notes:
00022 -- This testbench has been automatically generated using types std_logic and
00023 -- std_logic_vector for the ports of the unit under test.  Xilinx recommends
00024 -- that these types always be used for the top-level I/O of a design in order
00025 -- to guarantee that the testbench will bind correctly to the post-implementation
00026 -- simulation model.
00027 --------------------------------------------------------------------------------
00028 LIBRARY ieee;
00029 USE ieee.std_logic_1164.ALL;
00030 USE ieee.std_logic_unsigned.all;
00031 USE ieee.numeric_std.ALL;
00032
00033 ENTITY Arbiter_tb IS
00034 END Arbiter_tb;
00035
00036 ARCHITECTURE behavior OF Arbiter_tb IS
00037
00038     -- Component Declaration for the Unit Under Test (UUT)
00039
00040     COMPONENT Arbiter
00041     PORT(
00042          clk : IN  std_logic;
00043          reset : IN  std_logic;
00044          Bus_Request : IN  std_logic_vector(7 downto 0);
00045          Bus_Busy : IN  std_logic;
00046          Bus_Grant : OUT  std_logic_vector(7 downto 0)
00047         );
00048     END COMPONENT;
00049
```

```
00050
00051    --Inputs
00052    signal clk : std_logic := '0';
00053    signal reset : std_logic := '0';
00054    signal Bus_Request : std_logic_vector(7 downto 0) := (others => '0');
00055    signal Bus_Busy : std_logic := '0';
00056
00057     --Outputs
00058    signal Bus_Grant : std_logic_vector(7 downto 0);
00059
00060    -- Clock period definitions
00061    constant clk_period : time := 20us;
00062
00063 BEGIN
00064
00065     -- Instantiate the Unit Under Test (UUT)
00066    uut: Arbiter PORT MAP (
00067           clk => clk,
00068           reset => reset,
00069           Bus_Request => Bus_Request,
00070           Bus_Busy => Bus_Busy,
00071           Bus_Grant  => Bus_Grant
00072         );
00073
00074    -- Clock process definitions
00075    clk_process :process
00076    begin
00077         clk <= '0';
00078         wait for clk_period/2;
00079         clk <= '1';
00080         wait for clk_period/2;
00081    end process;
00082
00083
00084    -- Stimulus process
00085    stim_proc: process
00086    begin
00087       -- hold reset state for 100ms.
00088         Bus_Request <= x"00";
00089         Bus_Busy <= '0';
00090         reset <= '0';
00091       wait for 100ns;
00092         reset <= '1';
00093       wait for clk_period*5;
00094
00095         Bus_Request(0) <= '1';
00096         Bus_Request(5) <= '1';
00097         wait for clk_period*3;
00098         Bus_Request(0) <= '0';
00099         Bus_Busy <= '1';
00100         wait for clk_period*10;
00101         Bus_Busy <= '0';
00102
00103         wait for clk_period*10;
00104
00105         --Bus_Request(5) <= '1';
00106         wait for clk_period*3;
00107         Bus_Request(5) <= '0';
00108         Bus_Busy <= '1';
00109         wait for clk_period*10;
00110         Bus_Busy <= '0';
00111
00112       -- insert stimulus here
00113
00114       wait;
00115    end process;
00116
00117 END;
```

Clock_Divider.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
```

```
00005 -- Create Date:    13:29:16 02/10/2012
00006 -- Design Name:
00007 -- Module Name:    Clock_Divider - Behavioral
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- Dependencies:
00014 --
00015 -- Revision:
00016 -- Revision 0.01 - File Created
00017 -- Additional Comments:
00018 --
00019 ----------------------------------------------------------------------------------
00020 library IEEE;
00021 use IEEE.STD_LOGIC_1164.ALL;
00022 use IEEE.STD_LOGIC_ARITH.ALL;
00023 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00024
00025 ---- Uncomment the following library declaration if instantiating
00026 ---- any Xilinx primitives in this code.
00027 --library UNISIM;
00028 --use UNISIM.VComponents.all;
00029
00030 entity Clock_Divider is
00031     Port ( clk_in : in  STD_LOGIC;
00032            reset : in  STD_LOGIC;
00033               divide_count : in STD_LOGIC_VECTOR(7 downto 0);
00034            clk_out : out  STD_LOGIC);
00035 end Clock_Divider;
00036
00037 architecture Behavioral of Clock_Divider is
00038
00039 signal clk_div_count : STD_LOGIC_VECTOR(7 downto 0) := x"00";
00040 signal clk_out_sig   : STD_LOGIC := '0';
00041
00042 begin
00043
00044 clk_out <= clk_out_sig;
00045
00046 -- convST
00047 process(clk_in, reset)
00048 begin
00049    if reset = '0' then
00050         clk_out_sig <= '0';
00051         clk_div_count <= (others => '0');
00052    elsif rising_edge(clk_in) then
00053         if clk_div_count = divide_count then
00054             clk_div_count <= (others => '0');
00055             clk_out_sig <= not clk_out_sig;
00056         else
00057             clk_div_count <= clk_div_count + 1;
00058         end if;
00059     end if;
00060 end process;
00061
00062
00063 end Behavioral;
00064
```

Clock_Divider_tb.vhd

```
00001 --------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
00005 -- Create Date:   13:36:11 02/10/2012
00006 -- Design Name:
00007 -- Module Name:   C:/Users/Kyle/Desktop/Thesis/2-10-12/Cypress Compatible FPGA
      Code/Data_Acq_8Channel_ADC_Test/Clock_Divider_tb.vhd
00008 -- Project Name:  Data_Acq_8Channel_ADC_Test
00009 -- Target Device:
00010 -- Tool versions:
00011 -- Description:
```

```
00012 --
00013 -- VHDL Test Bench Created by ISE for module: Clock_Divider
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 -- Revision 0.01 - File Created
00019 -- Additional Comments:
00020 --
00021 -- Notes:
00022 -- This testbench has been automatically generated using types std_logic and
00023 -- std_logic_vector for the ports of the unit under test.  Xilinx recommends
00024 -- that these types always be used for the top-level I/O of a design in order
00025 -- to guarantee that the testbench will bind correctly to the post-implementation
00026 -- simulation model.
00027 --------------------------------------------------------------------------------
00028 LIBRARY ieee;
00029 USE ieee.std_logic_1164.ALL;
00030 USE ieee.std_logic_unsigned.all;
00031 USE ieee.numeric_std.ALL;
00032
00033 ENTITY Clock_Divider_tb IS
00034 END Clock_Divider_tb;
00035
00036 ARCHITECTURE behavior OF Clock_Divider_tb IS
00037
00038     -- Component Declaration for the Unit Under Test (UUT)
00039
00040     COMPONENT Clock_Divider
00041     PORT(
00042          clk_in : IN  std_logic;
00043          reset : IN  std_logic;
00044          clk_out : OUT  std_logic
00045         );
00046     END COMPONENT;
00047
00048
00049    --Inputs
00050    signal clk_in : std_logic := '0';
00051    signal reset : std_logic := '0';
00052
00053     --Outputs
00054    signal clk_out : std_logic;
00055
00056    -- Clock period definitions
00057    constant clk_in_period : time := 20ns;
00058   -- constant clk_out_period : time := 1us;
00059
00060 BEGIN
00061
00062     -- Instantiate the Unit Under Test (UUT)
00063    uut: Clock_Divider PORT MAP (
00064           clk_in => clk_in,
00065           reset  => reset,
00066           clk_out => clk_out
00067         );
00068
00069    -- Clock process definitions
00070    clk_in_process :process
00071    begin
00072         clk_in <= '0';
00073         wait for clk_in_period/2;
00074         clk_in <= '1';
00075         wait for clk_in_period/2;
00076    end process;
00077
00078
00079
00080
00081    -- Stimulus process
00082    stim_proc: process
00083    begin
00084       -- hold reset state for 100ms.
00085         reset <= '0';
00086       wait for 1ms;
00087         reset <= '1';
00088
00089       wait for clk_in_period*100;
00090
```

```
00091       -- insert stimulus here
00092
00093       wait;
00094    end process;
00095
00096 END;
```

Command_Handler.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
00005 -- Create Date:    19:44:35 01/27/2012
00006 -- Design Name:
00007 -- Module Name:    ADC_Module - Behavioral
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- Dependencies:
00014 --
00015 -- Revision:
00016 -- Revision 0.01 - File Created
00017 -- Additional Comments:
00018 --
00019 ----------------------------------------------------------------------------------
00020 library IEEE;
00021 use IEEE.STD_LOGIC_1164.ALL;
00022 use IEEE.STD_LOGIC_ARITH.ALL;
00023 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00024
00025 ---- Uncomment the following library declaration if instantiating
00026 ---- any Xilinx primitives in this code.
00027 --library UNISIM;
00028 --use UNISIM.VComponents.all;
00029
00030 use work.Command_Handler_pkg.all;
00031
00032 entity Command_Handler is
00033     Port ( clk                      : in  STD_LOGIC;
00034            reset                    : in  STD_LOGIC;
00035
00036               Channel1_Config       : out STD_LOGIC_VECTOR(7 downto 0);
00037               Channel2_Config       : out STD_LOGIC_VECTOR(7 downto 0);
00038               Channel3_Config       : out STD_LOGIC_VECTOR(7 downto 0);
00039               Channel4_Config       : out STD_LOGIC_VECTOR(7 downto 0);
00040               Channel5_Config       : out STD_LOGIC_VECTOR(7 downto 0);
00041               Channel6_Config       : out STD_LOGIC_VECTOR(7 downto 0);
00042               Channel7_Config       : out STD_LOGIC_VECTOR(7 downto 0);
00043               Channel8_Config       : out STD_LOGIC_VECTOR(7 downto 0);
00044
00045               Stimulation           : out STD_LOGIC_VECTOR(7 downto 0); -- current stim
      setting for each channel (currently only supports 1 at a time)
00046               Acquisition           : out STD_LOGIC_VECTOR(7 downto 0); -- current acq
      setting for each channel (currently only supports all or nothing)
00047
00048               -- RX_FIFO Signals
00049               RX_FIFO_RD_CLK        : out STD_LOGIC;
00050               RX_FIFO_DOUT          : in  STD_LOGIC_VECTOR (7 downto 0);
00051               RX_FIFO_RD_EN         : out STD_LOGIC;
00052               RX_FIFO_EMPTY         : in  STD_LOGIC;
00053
00054               -- TX FIFO Signals
00055               TX_FIFO_WR_CLK        : out STD_LOGIC;
00056               TX_FIFO_DIN           : out STD_LOGIC_VECTOR(7 downto 0);
00057               TX_FIFO_WR_EN         : out STD_LOGIC;
00058
00059               -- RAM_Module Control
00060               RAM_Start_Op          : out STD_LOGIC;
00061               RAM_Op_Done           : in  STD_LOGIC;
00062               RAM_WE                : out STD_LOGIC;
00063               RAM_ADDR              : out STD_LOGIC_VECTOR(22 downto 0);
00064               RAM_DOUT              : in  STD_LOGIC_VECTOR(15 downto 0);
```

```
00065               RAM_DIN                     : out   STD_LOGIC_VECTOR(15 downto 0);
00066
00067               -- RAM_Arbiter
00068               RAM_Bus_Request             : out STD_LOGIC;
00069               RAM_Bus_Busy               : out STD_LOGIC;
00070               RAM_Bus_Grant             : in STD_LOGIC
00071               );
00072 end Command_Handler;
00073
00074 architecture Behavioral of Command_Handler is
00075
00076 signal Channel1_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00077 signal Channel2_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00078 signal Channel3_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00079 signal Channel4_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00080 signal Channel5_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00081 signal Channel6_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00082 signal Channel7_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00083 signal Channel8_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00084
00085 signal Stimulation_reg               : STD_LOGIC_VECTOR(7 downto 0); -- current stim setting
       for each channel (currently only supports 1 at a time)
00086 signal Acquisition_reg               : STD_LOGIC_VECTOR(7 downto 0); -- current acq setting
      for each channel (currently only supports all or nothing)
00087
00088 signal async_flags                   : STD_LOGIC_VECTOR(15 downto 0);
00089 signal count                           : STD_LOGIC_VECTOR(7 downto 0);
00090
00091 signal MSG_Length                     : STD_LOGIC_VECTOR(15 downto 0);
00092 signal MSG_ID                         : STD_LOGIC_VECTOR(7 downto 0);
00093 signal MSG_Channel                   : STD_LOGIC_VECTOR(7 downto 0);
00094
00095 signal MSG_Complete                 : STD_LOGIC;
00096
00097 signal status                         : STD_LOGIC_VECTOR(7 downto 0) := x"45";
00098
00099
00100 -------------- MSG_ID 0x01 - CONFIG_CHAN signals -----------------------------
00101 signal MSG_01_Start                 : STD_LOGIC;
00102 signal MSG_01_Complete              : STD_LOGIC;
00103 signal MSG_01_RX_FIFO_RD_EN     : STD_LOGIC;
00104 signal MSG_01_TX_FIFO_DIN         : STD_LOGIC_VECTOR(7 downto 0);
00105 signal MSG_01_TX_FIFO_WR_EN     : STD_LOGIC;
00106
00107 component MSG_01_Config_Chan
00108     Port ( clk                      : in  STD_LOGIC;
00109            reset                    : in  STD_LOGIC;
00110               MSG_Start                 : in  STD_LOGIC;
00111               MSG_Complete             : out STD_LOGIC;
00112
00113               -- Header Information
00114               MSG_Channel               : in STD_LOGIC_VECTOR(7 downto 0);
00115
00116               -- Channel Configuration
00117               Channel1_Config        : out STD_LOGIC_VECTOR(7 downto 0);
00118               Channel2_Config        : out STD_LOGIC_VECTOR(7 downto 0);
00119               Channel3_Config        : out STD_LOGIC_VECTOR(7 downto 0);
00120               Channel4_Config        : out STD_LOGIC_VECTOR(7 downto 0);
00121               Channel5_Config        : out STD_LOGIC_VECTOR(7 downto 0);
00122               Channel6_Config        : out STD_LOGIC_VECTOR(7 downto 0);
00123               Channel7_Config        : out STD_LOGIC_VECTOR(7 downto 0);
00124               Channel8_Config        : out STD_LOGIC_VECTOR(7 downto 0);
00125
00126               -- RX_FIFO Signals
00127               RX_FIFO_DOUT             : in  STD_LOGIC_VECTOR (7 downto 0);
00128               RX_FIFO_RD_EN           : out STD_LOGIC;
00129               RX_FIFO_EMPTY           : in  STD_LOGIC;
00130
00131               -- TX FIFO Signals
00132               TX_FIFO_DIN               : out STD_LOGIC_VECTOR(7 downto 0);
00133               TX_FIFO_WR_EN           : out STD_LOGIC
00134               );
00135 end component;
00136
00137 -------------- MSG_ID 0x05 - SET_WAVEFORM signals -----------------------------
00138 signal MSG_05_Start                 : STD_LOGIC;
00139 signal MSG_05_Complete              : STD_LOGIC;
00140 signal MSG_05_RX_FIFO_RD_EN     : STD_LOGIC;
00141 signal MSG_05_TX_FIFO_DIN         : STD_LOGIC_VECTOR(7 downto 0);
```

```
00142 signal MSG_05_TX_FIFO_WR_EN     : STD_LOGIC;
00143 signal MSG_05_RAM_Start_Op           : STD_LOGIC;
00144 signal MSG_05_RAM_WE                  : STD_LOGIC;
00145 signal MSG_05_RAM_ADDR               : STD_LOGIC_VECTOR(22 downto 0);
00146 signal MSG_05_RAM_DIN             : STD_LOGIC_VECTOR(15 downto 0);
00147 signal MSG_05_RAM_Bus_Request     : STD_LOGIC;
00148 signal MSG_05_RAM_Bus_Busy          : STD_LOGIC;
00149
00150 component MSG_SET_WAVEFORM
00151     Port ( clk                      : in  STD_LOGIC;
00152            reset                      : in  STD_LOGIC;
00153               MSG_Start                   : in  STD_LOGIC;
00154               MSG_Complete              : out STD_LOGIC;
00155
00156               -- Header Information
00157               MSG_Channel                : in STD_LOGIC_VECTOR(7 downto 0);
00158
00159               -- RX_FIFO Signals
00160               RX_FIFO_DOUT              : in  STD_LOGIC_VECTOR (7 downto 0);
00161               RX_FIFO_RD_EN            : out STD_LOGIC;
00162               RX_FIFO_EMPTY            : in  STD_LOGIC;
00163
00164               -- TX FIFO Signals
00165               TX_FIFO_DIN                : out STD_LOGIC_VECTOR(7 downto 0);
00166               TX_FIFO_WR_EN            : out STD_LOGIC;
00167
00168               -- RAM_Module Control
00169               RAM_Start_Op              : out STD_LOGIC;
00170               RAM_Op_Done               : in  STD_LOGIC;
00171               RAM_WE                     : out STD_LOGIC;
00172               RAM_ADDR                  : out STD_LOGIC_VECTOR(22 downto 0);
00173               RAM_DOUT                  : in  STD_LOGIC_VECTOR(15 downto 0);
00174               RAM_DIN                    : out   STD_LOGIC_VECTOR(15 downto 0);
00175
00176               -- RAM_Arbiter
00177               RAM_Bus_Request           : out STD_LOGIC;
00178               RAM_Bus_Busy              : out STD_LOGIC;
00179               RAM_Bus_Grant             : in STD_LOGIC
00180               );
00181 end component;
00182
00183 -------------- MSG_ID 0x06 - GET_WAVEFORM signals ----------------------------
00184 signal MSG_06_Start                 : STD_LOGIC;
00185 signal MSG_06_Complete              : STD_LOGIC;
00186 signal MSG_06_RX_FIFO_RD_EN     : STD_LOGIC;
00187 signal MSG_06_TX_FIFO_DIN         : STD_LOGIC_VECTOR(7 downto 0);
00188 signal MSG_06_TX_FIFO_WR_EN     : STD_LOGIC;
00189 signal MSG_06_RAM_Start_Op           : STD_LOGIC;
00190 signal MSG_06_RAM_WE                  : STD_LOGIC;
00191 signal MSG_06_RAM_ADDR               : STD_LOGIC_VECTOR(22 downto 0);
00192 signal MSG_06_RAM_DIN             : STD_LOGIC_VECTOR(15 downto 0);
00193 signal MSG_06_RAM_Bus_Request     : STD_LOGIC;
00194 signal MSG_06_RAM_Bus_Busy          : STD_LOGIC;
00195
00196 component MSG_GET_WAVEFORM
00197     Port ( clk                      : in  STD_LOGIC;
00198            reset                      : in  STD_LOGIC;
00199               MSG_Start                   : in  STD_LOGIC;
00200               MSG_Complete              : out STD_LOGIC;
00201
00202               -- Header Information
00203               MSG_Channel                : in  STD_LOGIC_VECTOR(7 downto 0);
00204
00205               -- RX_FIFO Signals
00206               RX_FIFO_DOUT              : in  STD_LOGIC_VECTOR (7 downto 0);
00207               RX_FIFO_RD_EN            : out STD_LOGIC;
00208               RX_FIFO_EMPTY            : in  STD_LOGIC;
00209
00210               -- TX FIFO Signals
00211               TX_FIFO_DIN                : out STD_LOGIC_VECTOR(7 downto 0);
00212               TX_FIFO_WR_EN            : out STD_LOGIC;
00213
00214               -- RAM_Module Control
00215               RAM_Start_Op              : out STD_LOGIC;
00216               RAM_Op_Done               : in  STD_LOGIC;
00217               RAM_WE                     : out STD_LOGIC;
00218               RAM_ADDR                  : out STD_LOGIC_VECTOR(22 downto 0);
00219               RAM_DOUT                  : in  STD_LOGIC_VECTOR(15 downto 0);
00220               RAM_DIN                    : out   STD_LOGIC_VECTOR(15 downto 0);
```

```
00221
00222               -- RAM_Arbiter
00223               RAM_Bus_Request            : out STD_LOGIC;
00224               RAM_Bus_Busy              : out STD_LOGIC;
00225               RAM_Bus_Grant             : in  STD_LOGIC
00226               );
00227 end component;
00228
00229
00230 -------------- MSG_ID 0x07 - SET_STIM signals -----------------------------
00231 signal MSG_07_Start                 : STD_LOGIC;
00232 signal MSG_07_Complete               : STD_LOGIC;
00233 signal MSG_07_RX_FIFO_RD_EN     : STD_LOGIC;
00234 signal MSG_07_TX_FIFO_DIN         : STD_LOGIC_VECTOR(7 downto 0);
00235 signal MSG_07_TX_FIFO_WR_EN     : STD_LOGIC;
00236
00237 component MSG_SET_STIM
00238     Port ( clk                       : in  STD_LOGIC;
00239            reset                       : in  STD_LOGIC;
00240               MSG_Start                    : in  STD_LOGIC;
00241               MSG_Complete              : out STD_LOGIC;
00242
00243               -- Header Information
00244               MSG_Channel                : in STD_LOGIC_VECTOR(7 downto 0);
00245
00246               -- Channel Configuration
00247               Stimulation                : out STD_LOGIC_VECTOR(7 downto 0);
00248
00249               -- RX_FIFO Signals
00250               RX_FIFO_DOUT              : in  STD_LOGIC_VECTOR (7 downto 0);
00251               RX_FIFO_RD_EN             : out STD_LOGIC;
00252               RX_FIFO_EMPTY             : in  STD_LOGIC;
00253
00254               -- TX FIFO Signals
00255               TX_FIFO_DIN                : out STD_LOGIC_VECTOR(7 downto 0);
00256               TX_FIFO_WR_EN             : out STD_LOGIC
00257               );
00258 end component;
00259
00260
00261 component Command_Handler_states
00262 Port (  clk                           : in  STD_LOGIC;
00263            rst_n                        : in  STD_LOGIC;
00264            FIFO_EMPTY                      : in  STD_LOGIC;
00265            MSG_Complete                 : in  STD_LOGIC;
00266            RX_FIFO_DOUT                 : in  STD_LOGIC_VECTOR(7 downto 0);
00267            count                           : in  STD_LOGIC_VECTOR(7 downto 0);
00268            async_flags                   : out STD_LOGIC_VECTOR(15 downto 0) --flags to enable
     functions
00269            );
00270 end component;
00271
00272 begin
00273
00274 MSG_Complete <= MSG_01_Complete or MSG_05_Complete or
     MSG_06_Complete or MSG_07_Complete;
00275
00276 Channel1_Config <= Channel1_Config_reg;
00277 Channel2_Config <= Channel2_Config_reg;
00278 Channel3_Config <= Channel3_Config_reg;
00279 Channel4_Config <= Channel4_Config_reg;
00280 Channel5_Config <= Channel5_Config_reg;
00281 Channel6_Config <= Channel6_Config_reg;
00282 Channel7_Config <= Channel7_Config_reg;
00283 Channel8_Config <= Channel8_Config_reg;
00284
00285 Stimulation <= Stimulation_reg;
00286
00287 ------------------------------------------------------------------------------
00288 ---------------------- MSG_Header Information --------------------------------
00289 ------------------------------------------------------------------------------
00290 -- MSG_Length
00291 process(clk, reset)
00292 begin
00293    if reset = '0' then
00294         MSG_Length <= (others => '0');
00295    elsif rising_edge(clk) then
00296         if async_flags(READ_MESSAGE_FLAG) = '1' and count = x"02" then
00297            MSG_Length(15 downto 8) <= RX_FIFO_DOUT;
```

```
00298         elsif async_flags(READ_MESSAGE_FLAG) = '1' and count = x"03" then
00299             MSG_Length(7 downto 0) <= RX_FIFO_DOUT;
00300         elsif async_flags(IDLE_FLAG) = '1' then
00301             MSG_Length <= x"0008";
00302         end if;
00303     end if;
00304 end process;
00305 
00306 -- MSG_ID
00307 process(clk, reset)
00308 begin
00309     if reset = '0' then
00310         MSG_ID <= (others => '0');
00311     elsif rising_edge(clk) and async_flags(READ_MESSAGE_FLAG) = '1' and
      count = x"01" then
00312         MSG_ID <= RX_FIFO_DOUT;
00313     end if;
00314 end process;
00315 
00316 -- MSG_Channel
00317 process(clk, reset)
00318 begin
00319     if reset = '0' then
00320         MSG_Channel <= (others => '0');
00321     elsif rising_edge(clk) and async_flags(READ_MESSAGE_FLAG) = '1' and
      count = x"04" then
00322         MSG_Channel <= RX_FIFO_DOUT;
00323     end if;
00324 end process;
00325 
00326 --------------------------------------------------------------------------------
00327 ---------------------- RAM -----------------------------------------------------
00328 --------------------------------------------------------------------------------
00329 -- RAM_DIN
00330 process(clk, reset)
00331 begin
00332     if reset = '0' then
00333         RAM_DIN <= (others => '0');
00334     elsif rising_edge(clk) then
00335         if async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"05" then
00336             RAM_DIN <= MSG_05_RAM_DIN;
00337         elsif async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"06" then
00338             RAM_DIN <= MSG_06_RAM_DIN;
00339         end if;
00340     end if;
00341 end process;
00342 
00343 -- RAM_ADDR
00344 process(clk, reset)
00345 begin
00346     if reset = '0' then
00347         RAM_ADDR <= (others => '0');
00348     elsif rising_edge(clk) then
00349         if async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"05" then
00350             RAM_ADDR <= MSG_05_RAM_ADDR;
00351         elsif async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"06" then
00352             RAM_ADDR <= MSG_06_RAM_ADDR;
00353         end if;
00354     end if;
00355 end process;
00356 
00357 -- RAM_Start_Op
00358 process(clk, reset)
00359 begin
00360     if reset = '0' then
00361         RAM_Start_Op <= '0';
00362     elsif rising_edge(clk) then
00363         if async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"05" then
00364             RAM_Start_Op <= MSG_05_RAM_Start_Op;
00365         elsif async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"06" then
00366             RAM_Start_Op <= MSG_06_RAM_Start_Op;
00367         else
00368             RAM_Start_Op <= '0';
00369         end if;
00370     end if;
00371 end process;
00372 
00373 -- RAM_WE
00374 process(clk, reset)
```

```
00375 begin
00376    if reset = '0' then
00377         RAM_WE <= '0';
00378    elsif rising_edge(clk) then
00379         if async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"05" then
00380             RAM_WE <= MSG_05_RAM_WE;
00381         elsif async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"06" then
00382             RAM_WE <= MSG_06_RAM_WE;
00383         else
00384             RAM_WE <= '0';
00385         end if;
00386     end if;
00387 end process;
00388
00389 -- RAM_Bus_Request
00390 process(clk, reset)
00391 begin
00392    if reset = '0' then
00393         RAM_Bus_Request <= '0';
00394    elsif rising_edge(clk) then
00395         if async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"05" then
00396             RAM_Bus_Request <= MSG_05_RAM_Bus_Request;
00397         elsif async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"06" then
00398             RAM_Bus_Request <= MSG_06_RAM_Bus_Request;
00399         else
00400             RAM_Bus_Request <= '0';
00401         end if;
00402     end if;
00403 end process;
00404
00405 -- RAM_Bus_Busy
00406 process(clk, reset)
00407 begin
00408    if reset = '0' then
00409         RAM_Bus_Busy <= '0';
00410    elsif rising_edge(clk) then
00411         if async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"05" then
00412             RAM_Bus_Busy <= MSG_05_RAM_Bus_Busy;
00413         elsif async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"06" then
00414             RAM_Bus_Busy <= MSG_06_RAM_Bus_Busy;
00415         else
00416             RAM_Bus_Busy <= '0';
00417         end if;
00418     end if;
00419 end process;
00420
00421
00422 ---------------------------------------------------------------------------
00423 ---------------------- RX_FIFO --------------------------------------------
00424 ---------------------------------------------------------------------------
00425 RX_FIFO_RD_CLK <= clk;
00426
00427 -- RX_FIFO_RD_EN
00428 process(clk, reset)
00429 begin
00430    if reset = '0' then
00431         RX_FIFO_RD_EN <= '0';
00432    elsif rising_edge(clk) then
00433         if async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"01" then
00434             RX_FIFO_RD_EN <= MSG_01_RX_FIFO_RD_EN;
00435         elsif async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"05" then
00436             RX_FIFO_RD_EN <= MSG_05_RX_FIFO_RD_EN;
00437         elsif async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"06" then
00438             RX_FIFO_RD_EN <= MSG_06_RX_FIFO_RD_EN;
00439         elsif async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"07" then
00440             RX_FIFO_RD_EN <= MSG_07_RX_FIFO_RD_EN;
00441         elsif async_flags(RX_RD_EN_FLAG) = '1' then
00442             RX_FIFO_RD_EN <= '1';
00443         else
00444             RX_FIFO_RD_EN <= '0';
00445         end if;
00446     end if;
00447 end process;
00448
00449 ---------------------------------------------------------------------------
00450 ---------------------- TX_FIFO --------------------------------------------
00451 ---------------------------------------------------------------------------
00452 TX_FIFO_WR_CLK <= clk;
00453
```

```
00454 -- TX_FIFO_WR_EN
00455 process(clk, reset)
00456 begin
00457    if reset = '0' then
00458         TX_FIFO_WR_EN <= '0';
00459    elsif rising_edge(clk) then
00460         if async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"01" then
00461             TX_FIFO_WR_EN <= MSG_01_TX_FIFO_WR_EN;
00462         elsif async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"05" then
00463             TX_FIFO_WR_EN <= MSG_05_TX_FIFO_WR_EN;
00464         elsif async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"06" then
00465             TX_FIFO_WR_EN <= MSG_06_TX_FIFO_WR_EN;
00466         elsif async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"07" then
00467             TX_FIFO_WR_EN <= MSG_07_TX_FIFO_WR_EN;
00468         elsif async_flags(TX_WR_EN_FLAG) = '1' then
00469             TX_FIFO_WR_EN <= '1';
00470         else
00471             TX_FIFO_WR_EN <= '0';
00472         end if;
00473     end if;
00474 end process;
00475
00476 -- TX_FIFO_DIN
00477 process(clk, reset)
00478 begin
00479    if reset = '0' then
00480         TX_FIFO_DIN <= (others => '0');
00481    elsif rising_edge(clk) then
00482         if async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"01" then
00483             TX_FIFO_DIN <= MSG_01_TX_FIFO_DIN;
00484         elsif async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"05" then
00485             TX_FIFO_DIN <= MSG_05_TX_FIFO_DIN;
00486         elsif async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"06" then
00487             TX_FIFO_DIN <= MSG_06_TX_FIFO_DIN;
00488         elsif async_flags(PAYLOAD_FLAG) = '1' and MSG_ID = x"07" then
00489             TX_FIFO_DIN <= MSG_07_TX_FIFO_DIN;
00490         elsif async_flags(READ_MESSAGE_FLAG) = '1' then
00491             TX_FIFO_DIN <= RX_FIFO_DOUT;
00492         end if;
00493     end if;
00494 end process;
00495
00496 ------------------------------------------------------------------------------
00497 ---------------------- Command_Handler_states --------------------------------
00498 ------------------------------------------------------------------------------
00499 states : Command_Handler_states
00500 port map(
00501     clk                                 => clk,
00502     rst_n                              => reset,
00503     FIFO_EMPTY                          => RX_FIFO_EMPTY,
00504     MSG_Complete                        => MSG_Complete,
00505     RX_FIFO_DOUT                        => RX_FIFO_DOUT,
00506     count                                  => count,
00507     async_flags                         => async_flags
00508     );
00509
00510 ------------------------------------------------------------------------------
00511 ---------------------- MSG_01_Config_Chan ------------------------------------
00512 ------------------------------------------------------------------------------
00513 MSG_01_Start <= async_flags(PAYLOAD_START_FLAG) when
      MSG_ID = x"01" else
00514                     '0';
00515
00516 MSG_ID_01 : MSG_01_Config_Chan
00517 port map(
00518     clk                                 => clk,
00519    reset                               => reset,
00520     MSG_Start                           => MSG_01_Start,
00521     MSG_Complete                        => MSG_01_Complete,
00522
00523     -- Header Information
00524     MSG_Channel                         => MSG_Channel,
00525
00526     -- Channel Configuration
00527     Channel1_Config                 => Channel1_Config_reg,
00528     Channel2_Config                 => Channel2_Config_reg,
00529     Channel3_Config                 => Channel3_Config_reg,
00530     Channel4_Config                 => Channel4_Config_reg,
00531     Channel5_Config                 => Channel5_Config_reg,
```

```
00532     Channel6_Config                 => Channel6_Config_reg,
00533     Channel7_Config                 => Channel7_Config_reg,
00534     Channel8_Config                 => Channel8_Config_reg,
00535
00536     -- RX_FIFO Signals
00537     RX_FIFO_DOUT                          => RX_FIFO_DOUT,
00538     RX_FIFO_RD_EN                        => MSG_01_RX_FIFO_RD_EN,
00539     RX_FIFO_EMPTY                        => RX_FIFO_EMPTY,
00540
00541     -- TX FIFO Signals
00542     TX_FIFO_DIN                       => MSG_01_TX_FIFO_DIN,
00543     TX_FIFO_WR_EN                       => MSG_01_TX_FIFO_WR_EN
00544     );
00545
00546
00547 ------------------------------------------------------------------------------
00548 ---------------------- MSG_SET_WAVEFORM ------------------------------------
00549 ------------------------------------------------------------------------------
00550 MSG_05_Start <= async_flags(PAYLOAD_START_FLAG) when
      MSG_ID = x"05" else
00551                     '0';
00552
00553 MSG_ID_05 : MSG_SET_WAVEFORM
00554 port map(
00555     clk                                 => clk,
00556    reset                               => reset,
00557     MSG_Start                           => MSG_05_Start,
00558     MSG_Complete                         => MSG_05_Complete,
00559
00560     -- Header Information
00561     MSG_Channel                           => MSG_Channel,
00562
00563     -- RX_FIFO Signals
00564     RX_FIFO_DOUT                         => RX_FIFO_DOUT,
00565     RX_FIFO_RD_EN                       => MSG_05_RX_FIFO_RD_EN,
00566     RX_FIFO_EMPTY                       => RX_FIFO_EMPTY,
00567
00568     -- TX FIFO Signals
00569     TX_FIFO_DIN                           => MSG_05_TX_FIFO_DIN,
00570     TX_FIFO_WR_EN                       => MSG_05_TX_FIFO_WR_EN,
00571
00572     -- RAM_Module Control
00573     RAM_Start_Op                         => MSG_05_RAM_Start_Op,
00574     RAM_Op_Done                           => RAM_Op_Done,
00575     RAM_WE                                     => MSG_05_RAM_WE,
00576     RAM_ADDR                                 => MSG_05_RAM_ADDR,
00577     RAM_DOUT                                 => RAM_DOUT,
00578     RAM_DIN                                 => MSG_05_RAM_DIN,
00579
00580     -- RAM_Arbiter
00581    RAM_Bus_Request                    => MSG_05_RAM_Bus_Request,
00582     RAM_Bus_Busy                             => MSG_05_RAM_Bus_Busy,
00583     RAM_Bus_Grant                       => RAM_Bus_Grant
00584     );
00585
00586
00587 ------------------------------------------------------------------------------
00588 ---------------------- MSG_GET_WAVEFORM ------------------------------------
00589 ------------------------------------------------------------------------------
00590 MSG_06_Start <= async_flags(PAYLOAD_START_FLAG) when
      MSG_ID = x"06" else
00591                     '0';
00592
00593 MSG_ID_06 : MSG_GET_WAVEFORM
00594 port map(
00595     clk                                 => clk,
00596    reset                               => reset,
00597     MSG_Start                           => MSG_06_Start,
00598     MSG_Complete                         => MSG_06_Complete,
00599
00600     -- Header Information
00601     MSG_Channel                           => MSG_Channel,
00602
00603     -- RX_FIFO Signals
00604     RX_FIFO_DOUT                         => RX_FIFO_DOUT,
00605     RX_FIFO_RD_EN                       => MSG_06_RX_FIFO_RD_EN,
00606     RX_FIFO_EMPTY                       => RX_FIFO_EMPTY,
00607
00608     -- TX FIFO Signals
```

```
00609     TX_FIFO_DIN                          => MSG_06_TX_FIFO_DIN,
00610     TX_FIFO_WR_EN                       => MSG_06_TX_FIFO_WR_EN,
00611
00612     -- RAM_Module Control
00613     RAM_Start_Op                         => MSG_06_RAM_Start_Op,
00614     RAM_Op_Done                          => RAM_Op_Done,
00615     RAM_WE                                    => MSG_06_RAM_WE,
00616     RAM_ADDR                                => MSG_06_RAM_ADDR,
00617     RAM_DOUT                                => RAM_DOUT,
00618     RAM_DIN                                 => MSG_06_RAM_DIN,
00619
00620     -- RAM_Arbiter
00621    RAM_Bus_Request                    => MSG_06_RAM_Bus_Request,
00622     RAM_Bus_Busy                            => MSG_06_RAM_Bus_Busy,
00623     RAM_Bus_Grant                       => RAM_Bus_Grant
00624     );
00625
00626
00627 ------------------------------------------------------------------------------
00628 ---------------------- MSG_SET_STIM ------------------------------------------
00629 ------------------------------------------------------------------------------
00630 MSG_07_Start <= async_flags(PAYLOAD_START_FLAG) when
      MSG_ID = x"07" else
00631                     '0';
00632
00633 MSG_ID_07 : MSG_SET_STIM
00634 port map(
00635     clk                                  => clk,
00636    reset                                => reset,
00637     MSG_Start                           => MSG_07_Start,
00638     MSG_Complete                         => MSG_07_Complete,
00639
00640     -- Header Information
00641     MSG_Channel                          => MSG_Channel,
00642
00643     -- Stimulation Configuration
00644     Stimulation                        => Stimulation_reg,
00645
00646     -- RX_FIFO Signals
00647     RX_FIFO_DOUT                         => RX_FIFO_DOUT,
00648     RX_FIFO_RD_EN                       => MSG_07_RX_FIFO_RD_EN,
00649     RX_FIFO_EMPTY                       => RX_FIFO_EMPTY,
00650
00651     -- TX FIFO Signals
00652     TX_FIFO_DIN                          => MSG_07_TX_FIFO_DIN,
00653     TX_FIFO_WR_EN                       => MSG_07_TX_FIFO_WR_EN
00654     );
00655
00656 ------------------------------------------------------------------------------
00657 ----------------------- Counter ----------------------------------------------
00658 ------------------------------------------------------------------------------
00659
00660 -- count
00661 process(clk, reset)
00662 begin
00663    if reset = '0' then
00664         count <= (others => '0');
00665    elsif rising_edge(clk) then
00666         if async_flags(INC_COUNT_FLAG) = '1' then
00667             count <= count + 1;
00668         elsif async_flags(IDLE_FLAG) = '1' then
00669             count <= x"00";
00670         elsif async_flags(CLEAR_COUNT_FLAG) = '1' then
00671             count <= x"00";
00672         end if;
00673     end if;
00674 end process;
00675
00676
00677 end Behavioral;
00678
```

Command_Handler_main_states.vhd

```
00001 ------------------------------------------------------------------------------
```

```
00002 -- Company:
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 --
00019 -- Additional Comments:
00020 --
00021 ----------------------------------------------------------------------------------
00022 library IEEE;
00023 use IEEE.STD_LOGIC_1164.ALL;
00024 use IEEE.numeric_std.all;
00025 --use IEEE.STD_LOGIC_ARITH.ALL;
00026 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00027
00028 use work.Command_Handler_pkg.all;
00029
00030 ---- Uncomment the following library declaration if instantiating
00031 ---- any Xilinx primitives in this code.
00032 library UNISIM;
00033 use UNISIM.VComponents.all;
00034
00035 entity Command_Handler_states is
00036 Port (  clk                         : in  STD_LOGIC;
00037             rst_n                     : in  STD_LOGIC;
00038             FIFO_EMPTY                  : in  STD_LOGIC;
00039             MSG_Complete                : in  STD_LOGIC;
00040             RX_FIFO_DOUT                : in  STD_LOGIC_VECTOR(7 downto 0);
00041             count                       : in  STD_LOGIC_VECTOR(7 downto 0);
00042             async_flags                 : out STD_LOGIC_VECTOR(15 downto 0) --flags to enable
      functions
00043             );
00044 end Command_Handler_states;
00045
00046 architecture Behavioral of Command_Handler_states is
00047
00048         --Control signals
00049
00050 signal curr_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM current state
00051 signal next_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM next state
00052
00053 begin
00054 ----------------------------------------------------------------------------------
00055         -- synchronous part of state machine here
00056 data_in_latch: process(clk, rst_n)
00057 begin
00058    if rst_n = '0' then
00059         curr_state <= (others => '0');
00060    elsif rising_edge(clk) then
00061         curr_state <= next_state;
00062    end if;
00063 end process;
00064
00065         -- async part of state machine to set function flags
00066 Command_Handler_state: process(rst_n, curr_state)
00067 begin
00068    if rst_n = '0' then
00069         async_flags <= (others => '0');
00070    else
00071         async_flags <= (others => '0');
00072         case curr_state is
00073
00074             when IDLE =>
00075                 async_flags(IDLE_FLAG) <= '1';           -- init
00076
00077             when CHECK_START =>
00078                 async_flags(INC_COUNT_FLAG) <= '1';
00079                 async_flags(RX_RD_EN_FLAG) <= '1';
```

```
00080
00081
00082 -- Message Handler
00083             when READ_MESSAGE =>
00084                 async_flags(READ_MESSAGE_FLAG) <= '1';
00085
00086             when INC_RX_FIFO =>
00087                 async_flags(INC_COUNT_FLAG) <= '1';
00088                 async_flags(RX_RD_EN_FLAG) <= '1';
00089                 --async_flags(TX_WR_EN_FLAG) <= '1';  -- uncomment to enable TX loopback of incoming
      messages (except for start byte)
00090
00091             when PAYLOAD_START =>
00092                 async_flags(PAYLOAD_START_FLAG) <= '1';
00093
00094             when PAYLOAD =>
00095                 async_flags(PAYLOAD_FLAG) <= '1';
00096
00097
00098
00099             when FINISH =>
00100                 async_flags(DONE_FLAG) <= '1';           -- done flag
00101             when others =>
00102                 async_flags <= (others => '0');
00103         end case;
00104     end if;
00105 end process;
00106
00107
00108
      ------------------------------------------------------------------------------------------------------------------------
00109
      ------------------------------------------------------------------------------------------------------------------------
00110 -- Command_Handler state machine
00111
      ------------------------------------------------------------------------------------------------------------------------
00112
      ------------------------------------------------------------------------------------------------------------------------
00113 Command_Handler_asynch_state: process(rst_n,
      curr_state, count, RX_FIFO_DOUT, FIFO_EMPTY,
      MSG_Complete)
00114 begin
00115    if rst_n = '0' then
00116         next_state <= IDLE;
00117    else
00118         case curr_state is
00119         when IDLE =>
00120                 if FIFO_EMPTY = '0' then
00121                     next_state <= CHECK_START;
00122                 else
00123                     next_state <= IDLE;
00124                 end if;
00125
00126             when CHECK_START =>
00127                 if RX_FIFO_DOUT = x"5A" then
00128                     next_state <= DELAY_STATE;
00129                 else
00130                     next_state <= FINISH;
00131                 end if;
00132
00133
      ------------------------------------------------------------------------------------------------------------------------
00134 -- Message Handler States
00135
00136     -- DELAY_STATE:                 provides enough time for FIFO_EMPTY to go high after RX_RD_EN is set
00137
00138     -- WAIT_FOR_NEXT_BYTE:      wait for FIFO to have next byte
00139
00140     -- READ_MESSAGE:                assigns RX_FIFO_DOUT to appropriate registers based on count
00141
00142     -- INC_RX_FIFO:             increment FIFO
00143
00144     -- VALIDATE_MSG:                currently only used to clear count.  will be used for checksum.
00145
      ------------------------------------------------------------------------------------------------------------------------
00146             when DELAY_STATE =>
00147                 if count = 5 then
00148                     next_state <= PAYLOAD_START;
00149                 else
```

```
00150                     next_state <= WAIT_FOR_NEXT_BYTE;
00151                 end if;
00152
00153             when WAIT_FOR_NEXT_BYTE =>
00154                 if FIFO_EMPTY = '0' then
00155                     next_state <= READ_MESSAGE;
00156                 else
00157                     next_state <= WAIT_FOR_NEXT_BYTE;
00158                 end if;
00159
00160             when READ_MESSAGE =>
00161                 next_state <= INC_RX_FIFO;
00162
00163             when INC_RX_FIFO =>
00164                 next_state <= DELAY_STATE;
00165
00166
00167
    ----------------------------------------------------------------------------------------------------------------
00168 -- process payload and send reply
00169
    ----------------------------------------------------------------------------------------------------------------
00170             when PAYLOAD_START =>
00171                 next_state <= PAYLOAD;
00172
00173             when PAYLOAD =>
00174                 if MSG_Complete = '1' then
00175                     next_state <= FINISH;
00176                 else
00177                     next_state <= PAYLOAD;
00178                 end if;
00179
00180
00181
00182
00183             when FINISH =>
00184                     next_state <= IDLE;
00185
00186             when OTHERS =>
00187                 next_state <= IDLE;
00188         end case;
00189    end if;
00190 end process;
00191
00192
00193 end Behavioral;
00194
00195
```

Command_Handler_pkg.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:     WMU - Thesis
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 --
00016 -- Dependencies:
00017 --
00018 -- Revision:
00019 -- Revision 0.01 - File Created
00020 -- Additional Comments:
00021 --
00022 ----------------------------------------------------------------------------------
00023 library IEEE;
00024 use IEEE.STD_LOGIC_1164.ALL;
```

```
00025 use IEEE.numeric_std.all;
00026 --use IEEE.STD_LOGIC_ARITH.ALL;
00027 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00028
00029 package Command_Handler_pkg is
00030
00031
00032 constant IDLE                                   : STD_LOGIC_VECTOR(7 downto 0) := x"00";
00033
00034 constant CHECK_START                           : STD_LOGIC_VECTOR(7 downto 0) := x"01";
00035 constant READ_MESSAGE                         : STD_LOGIC_VECTOR(7 downto 0) := x"02";
00036 constant INC_RX_FIFO                           : STD_LOGIC_VECTOR(7 downto 0) := x"03";
00037 constant VALIDATE_MSG                         : STD_LOGIC_VECTOR(7 downto 0) := x"04";
00038 constant WAIT_FOR_NEXT_BYTE                  : STD_LOGIC_VECTOR(7 downto 0) := x"05";
00039 constant DELAY_STATE                          : STD_LOGIC_VECTOR(7 downto 0) := x"06";
00040
00041 constant PAYLOAD_START                          : STD_LOGIC_VECTOR(7 downto 0) := x"10";
00042 constant PAYLOAD                                  : STD_LOGIC_VECTOR(7 downto 0) := x"11";
00043
00044
00045 constant FINISH                                  : STD_LOGIC_VECTOR(7 downto 0) := x"FF";
00046
00047
00048
00049 constant IDLE_FLAG                            : integer := 0;
00050
00051 constant INC_COUNT_FLAG                         : integer := 1;
00052 constant CLEAR_COUNT_FLAG                     : integer := 2;
00053
00054 constant RX_RD_EN_FLAG                        : integer := 3;
00055 constant READ_MESSAGE_FLAG                    : integer := 4;
00056
00057 --constant SET_REPLY_BYTE_FLAG                 : integer := 5;
00058 constant TX_WR_EN_FLAG                        : integer := 6;
00059
00060 constant PAYLOAD_START_FLAG                   : integer := 7;
00061 constant PAYLOAD_FLAG                           : integer := 8;
00062
00063
00064 constant DONE_FLAG                            : integer := 15;
00065
00066
00067
00068 end Command_Handler_pkg;
00069
00070 package body Command_Handler_pkg is
00071
00072 end Command_Handler_pkg;
00073
00074
```

Command_Handler_tb.vhd

```
00001 --------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
00005 -- Create Date:   22:24:32 02/29/2012
00006 -- Design Name:
00007 -- Module Name:   C:/Users/Kyle/Desktop/SVN_Thesis/FPGA Code/RS232_Module_Test_Rev_0.3/Command_Handler_tb.
      vhd
00008 -- Project Name:  RS232_Module_Test_Rev_0.3
00009 -- Target Device:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- VHDL Test Bench Created by ISE for module: Command_Handler
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 -- Revision 0.01 - File Created
00019 -- Additional Comments:
00020 --
00021 -- Notes:
```

```
00022 -- This testbench has been automatically generated using types std_logic and
00023 -- std_logic_vector for the ports of the unit under test.  Xilinx recommends
00024 -- that these types always be used for the top-level I/O of a design in order
00025 -- to guarantee that the testbench will bind correctly to the post-implementation
00026 -- simulation model.
00027 --------------------------------------------------------------------------------
00028 LIBRARY ieee;
00029 USE ieee.std_logic_1164.ALL;
00030 USE ieee.std_logic_unsigned.all;
00031 USE ieee.numeric_std.ALL;
00032
00033 ENTITY Command_Handler_tb IS
00034 END Command_Handler_tb;
00035
00036 ARCHITECTURE behavior OF Command_Handler_tb IS
00037
00038     -- Component Declaration for the Unit Under Test (UUT)
00039
00040     COMPONENT Command_Handler
00041     Port ( clk                      : in  STD_LOGIC;
00042            reset                      : in  STD_LOGIC;
00043
00044                Channel1_Config      : out STD_LOGIC_VECTOR(7 downto 0);
00045                Channel2_Config      : out STD_LOGIC_VECTOR(7 downto 0);
00046                Channel3_Config      : out STD_LOGIC_VECTOR(7 downto 0);
00047                Channel4_Config      : out STD_LOGIC_VECTOR(7 downto 0);
00048                Channel5_Config      : out STD_LOGIC_VECTOR(7 downto 0);
00049                Channel6_Config      : out STD_LOGIC_VECTOR(7 downto 0);
00050                Channel7_Config      : out STD_LOGIC_VECTOR(7 downto 0);
00051                Channel8_Config      : out STD_LOGIC_VECTOR(7 downto 0);
00052
00053                -- RX_FIFO Signals
00054                RX_FIFO_RD_CLK        : out STD_LOGIC;
00055                RX_FIFO_DOUT            : in  STD_LOGIC_VECTOR (7 downto 0);
00056                RX_FIFO_RD_EN          : out STD_LOGIC;
00057                RX_FIFO_EMPTY          : in  STD_LOGIC;
00058
00059                -- TX FIFO Signals
00060                TX_FIFO_WR_CLK        : out STD_LOGIC;
00061                TX_FIFO_DIN              : out STD_LOGIC_VECTOR(7 downto 0);
00062                TX_FIFO_WR_EN          : out STD_LOGIC;
00063
00064                -- RAM_Module Control
00065                RAM_Start_Op            : out STD_LOGIC;
00066                RAM_Op_Done              : in  STD_LOGIC;
00067                RAM_WE                     : out STD_LOGIC;
00068                RAM_ADDR                 : out STD_LOGIC_VECTOR(22 downto 0);
00069                RAM_DOUT                 : in  STD_LOGIC_VECTOR(15 downto 0);
00070                RAM_DIN                   : out   STD_LOGIC_VECTOR(15 downto 0)
00071                );
00072     END COMPONENT;
00073
00074
00075    --Inputs
00076    signal clk : std_logic := '0';
00077    signal reset : std_logic := '0';
00078    signal RX_FIFO_DOUT : std_logic_vector(7 downto 0) := (others => '0');
00079    signal RX_FIFO_EMPTY : std_logic := '0';
00080     signal RAM_Op_Done : std_logic := '0';
00081     signal RAM_DOUT : std_logic_vector(15 downto 0) := (others => '0');
00082
00083     --Outputs
00084    signal Channel1_Config : std_logic_vector(7 downto 0);
00085    signal Channel2_Config : std_logic_vector(7 downto 0);
00086    signal Channel3_Config : std_logic_vector(7 downto 0);
00087    signal Channel4_Config : std_logic_vector(7 downto 0);
00088    signal Channel5_Config : std_logic_vector(7 downto 0);
00089    signal Channel6_Config : std_logic_vector(7 downto 0);
00090    signal Channel7_Config : std_logic_vector(7 downto 0);
00091    signal Channel8_Config : std_logic_vector(7 downto 0);
00092    signal RX_FIFO_RD_CLK : std_logic;
00093    signal RX_FIFO_RD_EN : std_logic;
00094    signal TX_FIFO_WR_CLK : std_logic;
00095    signal TX_FIFO_DIN : std_logic_vector(7 downto 0);
00096    signal TX_FIFO_WR_EN : std_logic;
00097     signal RAM_Start_Op : std_logic;
00098     signal RAM_WE : std_logic;
00099     signal RAM_ADDR : std_logic_vector(22 downto 0);
00100     signal RAM_DIN : std_logic_vector(15 downto 0);
```

```
00101
00102    -- Clock period definitions
00103    constant clk_period : time := 20ns;
00104
00105 BEGIN
00106
00107     -- RAM_Op_Done
00108 process(clk, reset)
00109 begin
00110    if reset = '0' then
00111         RAM_Op_Done <= '0';
00112    elsif rising_edge(clk) then
00113         if RAM_Start_Op = '1' then
00114             RAM_Op_Done <= '1';
00115         else
00116             RAM_Op_Done <= '0';
00117         end if;
00118     end if;
00119 end process;
00120
00121     -- Instantiate the Unit Under Test (UUT)
00122    uut: Command_Handler PORT MAP (
00123           clk => clk,
00124           reset => reset,
00125           Channel1_Config => Channel1_Config,
00126           Channel2_Config => Channel2_Config,
00127           Channel3_Config => Channel3_Config,
00128           Channel4_Config => Channel4_Config,
00129           Channel5_Config => Channel5_Config,
00130           Channel6_Config => Channel6_Config,
00131           Channel7_Config => Channel7_Config,
00132           Channel8_Config => Channel8_Config,
00133           RX_FIFO_RD_CLK => RX_FIFO_RD_CLK,
00134           RX_FIFO_DOUT => RX_FIFO_DOUT,
00135           RX_FIFO_RD_EN => RX_FIFO_RD_EN,
00136           RX_FIFO_EMPTY => RX_FIFO_EMPTY,
00137           TX_FIFO_WR_CLK => TX_FIFO_WR_CLK,
00138           TX_FIFO_DIN => TX_FIFO_DIN,
00139           TX_FIFO_WR_EN => TX_FIFO_WR_EN,
00140              RAM_Start_Op              => RAM_Start_Op,
00141              RAM_Op_Done            => RAM_Op_Done,
00142              RAM_WE                       => RAM_WE,
00143              RAM_ADDR                  => RAM_ADDR,
00144              RAM_DOUT                 => RAM_DOUT,
00145              RAM_DIN                    => RAM_DIN
00146         );
00147
00148    -- Clock process definitions
00149    clk_process :process
00150    begin
00151         clk <= '0';
00152         wait for clk_period/2;
00153         clk <= '1';
00154         wait for clk_period/2;
00155    end process;
00156
00157
00158    -- Stimulus process
00159    stim_proc: process
00160    begin
00161       -- hold reset state for 100ms.
00162         reset <= '0';
00163         RX_FIFO_EMPTY <= '1';
00164       wait for 100ns;
00165         reset <= '1';
00166
00167         RX_FIFO_EMPTY <= '0';
00168         RX_FIFO_DOUT <= x"5A";
00169         wait for 20ns;
00170
00171 -- config chan
00172 --       RX_FIFO_EMPTY <= '1';
00173 --       wait for 500ns;
00174 --       RX_FIFO_DOUT <= x"01";
00175 --       RX_FIFO_EMPTY <= '0';
00176 --       wait for 20ns;
00177 --
00178 --       RX_FIFO_EMPTY <= '1';
00179 --       wait for 500ns;
```

```
00180 --      RX_FIFO_DOUT <= x"00";
00181 --      RX_FIFO_EMPTY <= '0';
00182 --      wait for 20ns;
00183 --
00184 --      RX_FIFO_EMPTY <= '1';
00185 --      wait for 500ns;
00186 --      RX_FIFO_DOUT <= x"07";
00187 --      RX_FIFO_EMPTY <= '0';
00188 --      wait for 20ns;
00189 --
00190 --      RX_FIFO_EMPTY <= '1';
00191 --      wait for 500ns;
00192 --      RX_FIFO_DOUT <= x"01";
00193 --      RX_FIFO_EMPTY <= '0';
00194 --      wait for 20ns;
00195 --
00196 --      RX_FIFO_EMPTY <= '1';
00197 --      wait for 500ns;
00198 --      RX_FIFO_DOUT <= x"1F";
00199 --      RX_FIFO_EMPTY <= '0';
00200 --      wait for 20ns;
00201 --
00202 --      RX_FIFO_EMPTY <= '1';
00203 --      wait for 500ns;
00204 --      RX_FIFO_DOUT <= x"82";
00205 --      RX_FIFO_EMPTY <= '0';
00206 --      wait for 20ns;
00207
00208 -- SetWaveform
00209         RX_FIFO_EMPTY <= '1';
00210         wait for 500ns;
00211         RX_FIFO_DOUT <= x"05";
00212         RX_FIFO_EMPTY <= '0';
00213         wait for 20ns;
00214
00215         RX_FIFO_EMPTY <= '1';
00216         wait for 500ns;
00217         RX_FIFO_DOUT <= x"00";
00218         RX_FIFO_EMPTY <= '0';
00219         wait for 20ns;
00220
00221         RX_FIFO_EMPTY <= '1';
00222         wait for 500ns;
00223         RX_FIFO_DOUT <= x"0B";
00224         RX_FIFO_EMPTY <= '0';
00225         wait for 20ns;
00226
00227         RX_FIFO_EMPTY <= '1';
00228         wait for 500ns;
00229         RX_FIFO_DOUT <= x"01";
00230         RX_FIFO_EMPTY <= '0';
00231         wait for 20ns;
00232
00233         RX_FIFO_EMPTY <= '1';
00234         wait for 500ns;
00235         RX_FIFO_DOUT <= x"01";
00236         RX_FIFO_EMPTY <= '0';
00237         wait for 20ns;
00238
00239         RX_FIFO_EMPTY <= '1';
00240         wait for 500ns;
00241         RX_FIFO_DOUT <= x"12";
00242         RX_FIFO_EMPTY <= '0';
00243         wait for 20ns;
00244
00245         RX_FIFO_EMPTY <= '1';
00246         wait for 500ns;
00247         RX_FIFO_DOUT <= x"34";
00248         RX_FIFO_EMPTY <= '0';
00249         wait for 20ns;
00250
00251         RX_FIFO_EMPTY <= '1';
00252         wait for 500ns;
00253         RX_FIFO_DOUT <= x"56";
00254         RX_FIFO_EMPTY <= '0';
00255         wait for 20ns;
00256
00257         RX_FIFO_EMPTY <= '1';
00258         wait for 500ns;
```

```
00259         RX_FIFO_DOUT <= x"78";
00260         RX_FIFO_EMPTY <= '0';
00261         wait for 20ns;
00262
00263         RX_FIFO_EMPTY <= '1';
00264         wait for 500ns;
00265         RX_FIFO_DOUT <= x"FF";
00266         RX_FIFO_EMPTY <= '0';
00267         wait for 20ns;
00268
00269
00270
00271
00272       wait for clk_period*10;
00273
00274       -- insert stimulus here
00275
00276       wait;
00277    end process;
00278
00279 END;
```

DAC_Module.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
00005 -- Create Date:    19:44:35 01/27/2012
00006 -- Design Name:
00007 -- Module Name:    ADC_Module - Behavioral
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- Dependencies:
00014 --
00015 -- Revision:
00016 -- Revision 0.01 - File Created
00017 -- Additional Comments:
00018 --
00019 ----------------------------------------------------------------------------------
00020 library IEEE;
00021 use IEEE.STD_LOGIC_1164.ALL;
00022 use IEEE.STD_LOGIC_ARITH.ALL;
00023 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00024
00025 ---- Uncomment the following library declaration if instantiating
00026 ---- any Xilinx primitives in this code.
00027 --library UNISIM;
00028 --use UNISIM.VComponents.all;
00029
00030 use work.DAC_Module_pkg.all;
00031
00032 entity DAC_Module is
00033     Port ( clk                      : in  STD_LOGIC;
00034            reset                    : in  STD_LOGIC;
00035               Stimulation           : in  STD_LOGIC_VECTOR(7 downto 0); -- Pulse for single
       stim, Hold for multi stim
00036               Stim_Active           : out STD_LOGIC_VECTOR(3 downto 0); -- shows
      stimulation active for channels 1-4
00037               Init_Complete         : out STD_LOGIC;
00038
00039               -- SPI Signals
00040               SPI_CLK               : out STD_LOGIC;
00041            CS                       : out STD_LOGIC;
00042            MOSI                     : out STD_LOGIC;
00043
00044               -- RAM_Module Control
00045               RAM_Start_Op          : out STD_LOGIC;
00046               RAM_Op_Done           : in  STD_LOGIC;
00047               RAM_WE                : out STD_LOGIC;
00048               RAM_ADDR              : out STD_LOGIC_VECTOR(22 downto 0);
00049               RAM_DOUT              : in  STD_LOGIC_VECTOR(15 downto 0);
```

```
00050               RAM_DIN                     : out   STD_LOGIC_VECTOR(15 downto 0);
00051
00052               -- RAM_Arbiter
00053               RAM_Bus_Request             : out STD_LOGIC_VECTOR(3 downto 0);
00054               RAM_Bus_Busy               : out STD_LOGIC;
00055               RAM_Bus_Grant             : in  STD_LOGIC_VECTOR(3 downto 0)
00056               );
00057 end DAC_Module;
00058
00059 architecture Behavioral of DAC_Module is
00060
00061 ----------------------------------------------------------------------------------
00062 ---------------------- DAC_Channel -----------------------------------------------
00063 ----------------------------------------------------------------------------------
00064 signal Channel1_SPI_start         : STD_LOGIC;
00065 signal Channel1_SPI_data_reg       : STD_LOGIC_VECTOR(31 downto 0);
00066 signal Channel1_SPI_Bus_Busy       : STD_LOGIC;
00067 signal Channel1_RAM_Offset           : STD_LOGIC_VECTOR(15 downto 0);
00068 signal Channel1_DAC_Address     : STD_LOGIC_VECTOR(3 downto 0);
00069 signal Channel1_RAM_Bus_Busy       : STD_LOGIC;
00070 signal Channel1_RAM_Start_Op       : STD_LOGIC;
00071 signal Channel1_RAM_WE               : STD_LOGIC;
00072 signal Channel1_RAM_ADDR           : STD_LOGIC_VECTOR(22 downto 0);
00073 signal Channel1_RAM_DIN             : STD_LOGIC_VECTOR(15 downto 0);
00074
00075 signal Channel2_SPI_start         : STD_LOGIC;
00076 signal Channel2_SPI_data_reg       : STD_LOGIC_VECTOR(31 downto 0);
00077 signal Channel2_SPI_Bus_Busy       : STD_LOGIC;
00078 signal Channel2_RAM_Offset           : STD_LOGIC_VECTOR(15 downto 0);
00079 signal Channel2_DAC_Address     : STD_LOGIC_VECTOR(3 downto 0);
00080 signal Channel2_RAM_Bus_Busy       : STD_LOGIC;
00081 signal Channel2_RAM_Start_Op       : STD_LOGIC;
00082 signal Channel2_RAM_WE               : STD_LOGIC;
00083 signal Channel2_RAM_ADDR           : STD_LOGIC_VECTOR(22 downto 0);
00084 signal Channel2_RAM_DIN             : STD_LOGIC_VECTOR(15 downto 0);
00085
00086 signal Channel3_SPI_start         : STD_LOGIC;
00087 signal Channel3_SPI_data_reg       : STD_LOGIC_VECTOR(31 downto 0);
00088 signal Channel3_SPI_Bus_Busy       : STD_LOGIC;
00089 signal Channel3_RAM_Offset           : STD_LOGIC_VECTOR(15 downto 0);
00090 signal Channel3_DAC_Address     : STD_LOGIC_VECTOR(3 downto 0);
00091 signal Channel3_RAM_Bus_Busy       : STD_LOGIC;
00092 signal Channel3_RAM_Start_Op       : STD_LOGIC;
00093 signal Channel3_RAM_WE               : STD_LOGIC;
00094 signal Channel3_RAM_ADDR           : STD_LOGIC_VECTOR(22 downto 0);
00095 signal Channel3_RAM_DIN             : STD_LOGIC_VECTOR(15 downto 0);
00096
00097 signal Channel4_SPI_start         : STD_LOGIC;
00098 signal Channel4_SPI_data_reg       : STD_LOGIC_VECTOR(31 downto 0);
00099 signal Channel4_SPI_Bus_Busy       : STD_LOGIC;
00100 signal Channel4_RAM_Offset           : STD_LOGIC_VECTOR(15 downto 0);
00101 signal Channel4_DAC_Address     : STD_LOGIC_VECTOR(3 downto 0);
00102 signal Channel4_RAM_Bus_Busy       : STD_LOGIC;
00103 signal Channel4_RAM_Start_Op       : STD_LOGIC;
00104 signal Channel4_RAM_WE               : STD_LOGIC;
00105 signal Channel4_RAM_ADDR           : STD_LOGIC_VECTOR(22 downto 0);
00106 signal Channel4_RAM_DIN             : STD_LOGIC_VECTOR(15 downto 0);
00107
00108 component DAC_Channel
00109     Port ( clk                      : in  STD_LOGIC;
00110            reset                      : in  STD_LOGIC;
00111               Stimulation               : in  STD_LOGIC; -- Pulse to single stim, Hold to multi
       stim
00112               Stim_Active               : out STD_LOGIC;
00113               RAM_Offset              : in  STD_LOGIC_VECTOR(15 downto 0); -- Provides the
      starting location for RAM access for the channel
00114               DAC_Address               : in  STD_LOGIC_VECTOR(3 downto 0);
00115
00116               -- SPI Signals
00117               SPI_start                   : out STD_LOGIC;
00118            SPI_data                 : out STD_LOGIC_VECTOR(31 downto 0);
00119               SPI_Done                  : in  STD_LOGIC;
00120
00121               -- SPI_Arbiter
00122               SPI_Bus_Request            : out STD_LOGIC;
00123               SPI_Bus_Busy              : out STD_LOGIC;
00124               SPI_Bus_Grant             : in  STD_LOGIC;
00125
00126               -- RAM_Module Control
```

```
00127               RAM_Start_Op              : out STD_LOGIC;
00128               RAM_Op_Done                : in  STD_LOGIC;
00129               RAM_WE                      : out STD_LOGIC;
00130               RAM_ADDR                  : out STD_LOGIC_VECTOR(22 downto 0);
00131               RAM_DOUT                  : in  STD_LOGIC_VECTOR(15 downto 0);
00132               RAM_DIN                    : out   STD_LOGIC_VECTOR(15 downto 0);
00133
00134               -- RAM_Arbiter
00135               RAM_Bus_Request            : out STD_LOGIC;
00136               RAM_Bus_Busy              : out STD_LOGIC;
00137               RAM_Bus_Grant             : in  STD_LOGIC
00138               );
00139 end component;
00140
00141
00142 ----------------------------------------------------------------------------------
00143 ---------------------- DAC_Init ----------------------------------------------
00144 ----------------------------------------------------------------------------------
00145 signal Init_SPI_start         : STD_LOGIC;
00146 signal Init_SPI_data_reg       : STD_LOGIC_VECTOR(31 downto 0);
00147 signal Init_SPI_Bus_Busy       : STD_LOGIC;
00148
00149 component DAC_Init
00150     Port ( clk                      : in  STD_LOGIC;
00151            reset                     : in  STD_LOGIC;
00152               Init_Complete          : out STD_LOGIC;
00153
00154               -- SPI Signals
00155               SPI_start                   : out STD_LOGIC;
00156            SPI_data                : out STD_LOGIC_VECTOR(31 downto 0);
00157               SPI_Done                 : in  STD_LOGIC;
00158
00159               -- SPI_Arbiter
00160               SPI_Bus_Request          : out STD_LOGIC;
00161               SPI_Bus_Busy            : out STD_LOGIC;
00162               SPI_Bus_Grant           : in  STD_LOGIC
00163               );
00164 end component;
00165
00166
00167 ----------------------------------------------------------------------------------
00168 ---------------------- SPI ----------------------------------------------
00169 ----------------------------------------------------------------------------------
00170 signal SPI_Start                     : STD_LOGIC;
00171 signal SPI_Done                      : STD_LOGIC;
00172 signal SPI_data_reg                  : STD_LOGIC_VECTOR(31 downto 0);
00173
00174 signal SPI_Bus_Request              : STD_LOGIC_VECTOR(7 downto 0);
00175 signal SPI_Bus_Busy                    : STD_LOGIC;
00176 signal SPI_Bus_Grant              : STD_LOGIC_VECTOR(7 downto 0);
00177
00178 component SPI  -- Only supports sending 32 bits
00179 Port (  data_reg                       : in  STD_LOGIC_VECTOR(31 downto 0);
00180           SPI_start                   : in  STD_LOGIC;
00181           CLK                          : in  STD_LOGIC;
00182           rst                          : in  STD_LOGIC;
00183           CS                           : out STD_LOGIC;
00184           MOSI                         : out STD_LOGIC;
00185           SPI_Done                     : out STD_LOGIC;
00186
00187           -- SPI_Arbiter
00188           SPI_Bus_Request              : in  STD_LOGIC_VECTOR(7 downto 0);
00189           SPI_Bus_Busy                 : in  STD_LOGIC;
00190           SPI_Bus_Grant                : out STD_LOGIC_VECTOR(7 downto 0)
00191           );
00192 end component;
00193
00194 ----------------------------------------------------------------------------------
00195 ---------------------- Clock_Divider ----------------------------------------------
00196 ----------------------------------------------------------------------------------
00197
00198 --signal Clock_Divider_1us          : STD_LOGIC; -- used for incrementing time_count
00199 signal Clock_Divider_200ns          : STD_LOGIC;
00200 --signal ADC_divide_count_1us       : STD_LOGIC_VECTOR(7 downto 0);
00201 signal ADC_divide_count_200ns     : STD_LOGIC_VECTOR(7 downto 0);
00202
00203 component Clock_Divider
00204      Port ( clk_in : in  STD_LOGIC;
00205             reset : in  STD_LOGIC;
```

```
00206               divide_count : in STD_LOGIC_VECTOR(7 downto 0);
00207             clk_out : out  STD_LOGIC);
00208 end component;
00209
00210
00211
00212 begin
00213
00214 ------------------------------------------------------------------------
00215 ---------------------- RAM_Module ------------------------------------
00216 ------------------------------------------------------------------------
00217 RAM_Start_Op    <= Channel1_RAM_Start_Op when
      Channel1_RAM_Bus_Busy = '1' else
00218                      Channel2_RAM_Start_Op when
      Channel2_RAM_Bus_Busy = '1' else
00219                      Channel3_RAM_Start_Op when
      Channel3_RAM_Bus_Busy = '1' else
00220                      Channel4_RAM_Start_Op;
00221
00222 RAM_WE            <= Channel1_RAM_WE when
      Channel1_RAM_Bus_Busy = '1' else
00223                      Channel2_RAM_WE when
      Channel2_RAM_Bus_Busy = '1' else
00224                      Channel3_RAM_WE when
      Channel3_RAM_Bus_Busy = '1' else
00225                      Channel4_RAM_WE;
00226
00227 RAM_ADDR        <= Channel1_RAM_ADDR when
      Channel1_RAM_Bus_Busy = '1' else
00228                      Channel2_RAM_ADDR when
      Channel2_RAM_Bus_Busy = '1' else
00229                      Channel3_RAM_ADDR when
      Channel3_RAM_Bus_Busy = '1' else
00230                      Channel4_RAM_ADDR;
00231
00232 RAM_DIN          <= Channel1_RAM_DIN when
      Channel1_RAM_Bus_Busy = '1' else
00233                      Channel2_RAM_DIN when
      Channel2_RAM_Bus_Busy = '1' else
00234                      Channel3_RAM_DIN when
      Channel3_RAM_Bus_Busy = '1' else
00235                      Channel4_RAM_DIN;
00236
00237 RAM_Bus_Busy    <=  Channel1_RAM_Bus_Busy or
00238                      Channel2_RAM_Bus_Busy or
00239                      Channel3_RAM_Bus_Busy or
00240                      Channel4_RAM_Bus_Busy;
00241
00242
00243 ------------------------------------------------------------------------
00244 ---------------------- SPI -------------------------------------------
00245 ------------------------------------------------------------------------
00246 SPI_Start          <= Init_SPI_start when
      Init_SPI_Bus_Busy = '1' else
00247                         Channel1_SPI_start when
      Channel1_SPI_Bus_Busy = '1' else
00248                         Channel2_SPI_start when
      Channel2_SPI_Bus_Busy = '1' else
00249                         Channel3_SPI_start when
      Channel3_SPI_Bus_Busy = '1' else
00250                         Channel4_SPI_start;
00251
00252 SPI_data_reg        <= Init_SPI_data_reg when
      Init_SPI_Bus_Busy = '1' else
00253                         Channel1_SPI_data_reg when
      Channel1_SPI_Bus_Busy = '1' else
00254                         Channel2_SPI_data_reg when
      Channel2_SPI_Bus_Busy = '1' else
00255                         Channel3_SPI_data_reg when
      Channel3_SPI_Bus_Busy = '1' else
00256                         Channel4_SPI_data_reg;
00257
00258 SPI_Bus_Busy        <= Init_SPI_Bus_Busy or
00259                         Channel1_SPI_Bus_Busy or
00260                         Channel2_SPI_Bus_Busy or
00261                         Channel3_SPI_Bus_Busy or
00262                         Channel4_SPI_Bus_Busy;
00263
00264 SPI_CLK             <= Clock_Divider_200ns;
```

```
00265
00266 SPI_Module : SPI
00267 port map(
00268     data_reg                        => SPI_data_reg,
00269     SPI_start                      => SPI_start,
00270     CLK                                 => Clock_Divider_200ns,
00271     rst                               => reset,
00272     CS                                 => CS,
00273     MOSI                                => MOSI,
00274     SPI_Done                       => SPI_Done,
00275
00276     SPI_Bus_Request             => SPI_Bus_Request,
00277     SPI_Bus_Busy                       => SPI_Bus_Busy,
00278     SPI_Bus_Grant                     => SPI_Bus_Grant
00279     );
00280
00281 ------------------------------------------------------------------------------
00282 ---------------------- DAC_Init ----------------------------------------------
00283 ------------------------------------------------------------------------------
00284
00285 Init_DAC : DAC_Init
00286 port map(
00287     clk                              => clk,
00288    reset                            => reset,
00289     Init_Complete                 => Init_Complete,
00290
00291     -- SPI Signals
00292     SPI_start                          => Init_SPI_start,
00293     SPI_data                                => Init_SPI_data_reg,
00294     SPI_Done                          => SPI_Done,
00295
00296     -- SPI_Arbiter
00297     SPI_Bus_Request             => SPI_Bus_Request(4),
00298     SPI_Bus_Busy                   => Init_SPI_Bus_Busy,
00299     SPI_Bus_Grant                 => SPI_Bus_Grant(4)
00300     );
00301
00302
00303 ------------------------------------------------------------------------------
00304 ---------------------- DAC_Channel1 ------------------------------------------
00305 ------------------------------------------------------------------------------
00306 Channel1_RAM_Offset <= x"0000";
00307 Channel1_DAC_Address <= "0000";
00308
00309 Channel_1 : DAC_Channel
00310 port map(
00311     clk                              => clk,
00312    reset                            => reset,
00313     Stimulation                    => Stimulation(0),
00314     Stim_Active                    => Stim_Active(0),
00315     RAM_Offset                      => Channel1_RAM_Offset,
00316     DAC_Address                    => Channel1_DAC_Address,
00317
00318     -- SPI Signals
00319     SPI_start                         => Channel1_SPI_start,
00320     SPI_data                          => Channel1_SPI_data_reg,
00321     SPI_Done                          => SPI_Done,
00322
00323     -- SPI_Arbiter
00324     SPI_Bus_Request             => SPI_Bus_Request(0),
00325     SPI_Bus_Busy                   => Channel1_SPI_Bus_Busy,
00326     SPI_Bus_Grant                 => SPI_Bus_Grant(0),
00327
00328     -- RAM_Module Control
00329     RAM_Start_Op                  => Channel1_RAM_Start_Op,
00330     RAM_Op_Done                    => RAM_Op_Done,
00331     RAM_WE                              => Channel1_RAM_WE,
00332     RAM_ADDR                         => Channel1_RAM_ADDR,
00333     RAM_DOUT                         => RAM_DOUT,
00334     RAM_DIN                           => Channel1_RAM_DIN,
00335
00336     -- RAM_Arbiter
00337     RAM_Bus_Request             => RAM_Bus_Request(0),
00338     RAM_Bus_Busy                   => Channel1_RAM_Bus_Busy,
00339     RAM_Bus_Grant                 => RAM_Bus_Grant(0)
00340     );
00341
00342 ------------------------------------------------------------------------------
00343 ---------------------- DAC_Channel2 ------------------------------------------
```

```
00344 ------------------------------------------------------------------------------
00345 Channel2_RAM_Offset          <= x"1000";
00346 Channel2_DAC_Address        <= "0001";
00347
00348 Channel_2 : DAC_Channel
00349 port map(
00350     clk                             => clk,
00351    reset                           => reset,
00352     Stimulation                     => Stimulation(1),
00353     Stim_Active                     => Stim_Active(1),
00354     RAM_Offset                       => Channel2_RAM_Offset,
00355     DAC_Address                     => Channel2_DAC_Address,
00356
00357     -- SPI Signals
00358     SPI_start                         => Channel2_SPI_start,
00359     SPI_data                           => Channel2_SPI_data_reg,
00360     SPI_Done                           => SPI_Done,
00361
00362     -- SPI_Arbiter
00363     SPI_Bus_Request              => SPI_Bus_Request(1),
00364     SPI_Bus_Busy                    => Channel2_SPI_Bus_Busy,
00365     SPI_Bus_Grant                  => SPI_Bus_Grant(1),
00366
00367     -- RAM_Module Control
00368     RAM_Start_Op                    => Channel2_RAM_Start_Op,
00369     RAM_Op_Done                     => RAM_Op_Done,
00370     RAM_WE                               => Channel2_RAM_WE,
00371     RAM_ADDR                          => Channel2_RAM_ADDR,
00372     RAM_DOUT                          => RAM_DOUT,
00373     RAM_DIN                           => Channel2_RAM_DIN,
00374
00375     -- RAM_Arbiter
00376     RAM_Bus_Request              => RAM_Bus_Request(1),
00377     RAM_Bus_Busy                    => Channel2_RAM_Bus_Busy,
00378     RAM_Bus_Grant                  => RAM_Bus_Grant(1)
00379     );
00380
00381
00382 ------------------------------------------------------------------------------
00383 ---------------------- DAC_Channel3 ------------------------------------------
00384 ------------------------------------------------------------------------------
00385 Channel3_RAM_Offset          <= x"2000";
00386 Channel3_DAC_Address        <= "0110";
00387
00388 Channel_3 : DAC_Channel
00389 port map(
00390     clk                             => clk,
00391    reset                           => reset,
00392     Stimulation                     => Stimulation(2),
00393     Stim_Active                     => Stim_Active(2),
00394     RAM_Offset                       => Channel3_RAM_Offset,
00395     DAC_Address                     => Channel3_DAC_Address,
00396
00397     -- SPI Signals
00398     SPI_start                         => Channel3_SPI_start,
00399     SPI_data                           => Channel3_SPI_data_reg,
00400     SPI_Done                           => SPI_Done,
00401
00402     -- SPI_Arbiter
00403     SPI_Bus_Request              => SPI_Bus_Request(2),
00404     SPI_Bus_Busy                    => Channel3_SPI_Bus_Busy,
00405     SPI_Bus_Grant                  => SPI_Bus_Grant(2),
00406
00407     -- RAM_Module Control
00408     RAM_Start_Op                    => Channel3_RAM_Start_Op,
00409     RAM_Op_Done                     => RAM_Op_Done,
00410     RAM_WE                               => Channel3_RAM_WE,
00411     RAM_ADDR                          => Channel3_RAM_ADDR,
00412     RAM_DOUT                          => RAM_DOUT,
00413     RAM_DIN                           => Channel3_RAM_DIN,
00414
00415     -- RAM_Arbiter
00416     RAM_Bus_Request              => RAM_Bus_Request(2),
00417     RAM_Bus_Busy                    => Channel3_RAM_Bus_Busy,
00418     RAM_Bus_Grant                  => RAM_Bus_Grant(2)
00419     );
00420
00421
00422 ------------------------------------------------------------------------------
```

```
00423 ---------------------- DAC_Channel4 -----------------------------------------
00424 -----------------------------------------------------------------------------
00425 Channel4_RAM_Offset          <= x"3000";
00426 Channel4_DAC_Address        <= "0111";
00427
00428 Channel_4 : DAC_Channel
00429 port map(
00430     clk                             => clk,
00431    reset                           => reset,
00432     Stimulation                      => Stimulation(3),
00433     Stim_Active                      => Stim_Active(3),
00434     RAM_Offset                        => Channel4_RAM_Offset,
00435     DAC_Address                      => Channel4_DAC_Address,
00436
00437     -- SPI Signals
00438     SPI_start                          => Channel4_SPI_start,
00439     SPI_data                           => Channel4_SPI_data_reg,
00440     SPI_Done                           => SPI_Done,
00441
00442     -- SPI_Arbiter
00443     SPI_Bus_Request              => SPI_Bus_Request(3),
00444     SPI_Bus_Busy                    => Channel4_SPI_Bus_Busy,
00445     SPI_Bus_Grant                  => SPI_Bus_Grant(3),
00446
00447     -- RAM_Module Control
00448     RAM_Start_Op                     => Channel4_RAM_Start_Op,
00449     RAM_Op_Done                      => RAM_Op_Done,
00450     RAM_WE                                => Channel4_RAM_WE,
00451     RAM_ADDR                           => Channel4_RAM_ADDR,
00452     RAM_DOUT                           => RAM_DOUT,
00453     RAM_DIN                            => Channel4_RAM_DIN,
00454
00455     -- RAM_Arbiter
00456     RAM_Bus_Request              => RAM_Bus_Request(3),
00457     RAM_Bus_Busy                    => Channel4_RAM_Bus_Busy,
00458     RAM_Bus_Grant                  => RAM_Bus_Grant(3)
00459     );
00460
00461
00462
00463 -----------------------------------------------------------------------------
00464 ---------------------- Clock_Divider - 200ns --------------------------------
00465 -----------------------------------------------------------------------------
00466 ADC_divide_count_200ns <= x"04";
00467
00468 CLK_200ns : Clock_Divider
00469 port map(
00470     clk_in                             => clk,
00471     reset                               => reset,
00472     divide_count                     => ADC_divide_count_200ns,
00473     clk_out                                => Clock_Divider_200ns
00474     );
00475
00476 end Behavioral;
00477
```

DAC_Module_main_states.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
```

```
00018 --
00019 -- Additional Comments:
00020 --
00021 ----------------------------------------------------------------------------------
00022 library IEEE;
00023 use IEEE.STD_LOGIC_1164.ALL;
00024 use IEEE.numeric_std.all;
00025 --use IEEE.STD_LOGIC_ARITH.ALL;
00026 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00027
00028 use work.DAC_Module_pkg.all;
00029
00030 ---- Uncomment the following library declaration if instantiating
00031 ---- any Xilinx primitives in this code.
00032 library UNISIM;
00033 use UNISIM.VComponents.all;
00034
00035 entity DAC_Module_states is
00036 Port (  clk                         : in  STD_LOGIC;
00037             rst_n                     : in  STD_LOGIC;
00038             DAC_Module_start          : in  STD_LOGIC;            --start S.M. into
     motion
00039             SPI_Done                      : in  STD_LOGIC;
00040             RAM_Op_Done                 : in  STD_LOGIC;
00041             RAM_Bus_Grant               : in  STD_LOGIC;
00042             --continuous                   : in  STD_LOGIC;
00043             count                        : in  STD_LOGIC_VECTOR(7 downto 0);
00044             time_count                   : in  STD_LOGIC_VECTOR(15 downto 0);
00045             time_val                     : in  STD_LOGIC_VECTOR(15 downto 0);
00046             sample_count                : in  STD_LOGIC_VECTOR(7 downto 0);
00047             num_samples                 : in  STD_LOGIC_VECTOR(7 downto 0);
00048             async_flags                 : out STD_LOGIC_VECTOR(25 downto 0) --flags to enable
     functions
00049             );
00050 end DAC_Module_states;
00051
00052 architecture Behavioral of DAC_Module_states is
00053
00054         --Control signals
00055
00056 signal curr_state     : std_logic_vector(7 downto 0) := INIT;  -- FSM current state
00057 signal next_state     : std_logic_vector(7 downto 0) := INIT;  -- FSM next state
00058
00059 begin
00060 ----------------------------------------------------------------------------------
00061         -- synchronous part of state machine here
00062 data_in_latch: process(clk, rst_n)
00063 begin
00064    if rst_n = '0' then
00065         curr_state <= INIT;
00066    elsif rising_edge(clk) then
00067         curr_state <= next_state;
00068    end if;
00069 end process;
00070
00071         -- async part of state machine to set function flags
00072 DAC_Module_state: process(rst_n, curr_state,
     sample_count, num_samples, DAC_Module_start)
00073 begin
00074    if rst_n = '0' then
00075         async_flags <= (others => '0');
00076    else
00077         async_flags <= (others => '0');
00078         case curr_state is
00079             when INIT =>
00080                 async_flags(RESET_OP_SET_FLAG) <= '1';
00081                 async_flags(RESET_COUNT_FLAG) <= '1';
00082
00083             when RESET_OP_START =>
00084                 async_flags(START_FLAG) <= '1';
00085                 async_flags(INC_COUNT_FLAG) <= '1';
00086             when RESET_OP_WAIT =>
00087                 async_flags(RESET_COUNT_FLAG) <= '1';
00088             when RESET_OP_COMPLETE =>   -- due to different clock rate the SPI_Start signal is held high
     throughout the op. SPI waits for SPI_Start to go low
00089                 async_flags(INTERNAL_REF_REG_SET_FLAG) <= '1';
00090
00091             when INTERNAL_REF_REG_START =>
00092                 async_flags(START_FLAG) <= '1';
```

```
00093                async_flags(INC_COUNT_FLAG) <= '1';
00094            when INTERNAL_REF_REG_WAIT =>
00095                async_flags(RESET_COUNT_FLAG) <= '1';
00096            when INTERNAL_REF_REG_COMPLETE =>
00097                async_flags(LDAC_REG_SET_FLAG) <= '1';
00098
00099            when LDAC_REG_START =>
00100                async_flags(START_FLAG) <= '1';
00101                async_flags(INC_COUNT_FLAG) <= '1';
00102            when LDAC_REG_WAIT =>
00103                async_flags(RESET_COUNT_FLAG) <= '1';
00104            when LDAC_REG_COMPLETE =>
00105                async_flags(POWER_DAC_SET_FLAG) <= '1';
00106
00107            when POWER_DAC_START =>
00108                async_flags(START_FLAG) <= '1';
00109                async_flags(INC_COUNT_FLAG) <= '1';
00110            when POWER_DAC_WAIT =>
00111                async_flags(RESET_COUNT_FLAG) <= '1';
00112            when POWER_DAC_COMPLETE =>
00113                async_flags(CLEAR_CODE_SET_FLAG) <= '1';
00114
00115            when CLEAR_CODE_START =>
00116                async_flags(START_FLAG) <= '1';
00117                async_flags(INC_COUNT_FLAG) <= '1';
00118            when CLEAR_CODE_WAIT =>
00119                async_flags(RESET_COUNT_FLAG) <= '1';
00120
00121            when IDLE =>
00122                async_flags(IDLE_FLAG) <= '1';
00123
00124 ---------------------------------- Adding read from memory states ------------------------------
00125
00126 -- Reply - num_samples
00127            when NUM_SAMPLES_BR_WAIT =>      -- wait for RAM_Bus_Grant
00128                async_flags(RAM_BUS_REQUEST_FLAG) <= '1';
00129
00130            when NUM_SAMPLES_1 =>                          -- Start read from RAM for num_samples
00131                async_flags(RAM_BUS_BUSY_FLAG) <= '1';
00132                async_flags(START_MEM_OP_FLAG) <= '1';
00133                async_flags(NUM_SAMPLES_RD_FLAG) <= '1';
00134                --async_flags(CLEAR_COUNT_FLAG) <= '1';
00135
00136            when NUM_SAMPLES_2 =>                          -- wait for RAM Op to complete
00137                async_flags(RAM_BUS_BUSY_FLAG) <= '1';
00138                async_flags(NUM_SAMPLES_CAPTURE_FLAG) <= '1';
00139
00140    -- Read Amplitude from RAM and write to TX_FIFO
00141            when AMPLITUDE_BR_WAIT =>
00142                async_flags(RAM_BUS_REQUEST_FLAG) <= '1';
00143
00144            when AMPLITUDE_1 =>                 -- Start read from RAM for Amplitude
00145                async_flags(RAM_BUS_BUSY_FLAG) <= '1';
00146                async_flags(START_MEM_OP_FLAG) <= '1';
00147                async_flags(AMPLITUDE_RD_FLAG) <= '1';
00148
00149            when AMPLITUDE_2 =>                 -- wait for RAM Op to complete
00150                async_flags(RAM_BUS_BUSY_FLAG) <= '1';
00151                async_flags(AMPLITUDE_CAPTURE_FLAG) <= '1';
00152
00153    -- Read Time from RAM and write to TX_FIFO
00154            when TIME_BR_WAIT =>
00155                async_flags(RAM_BUS_REQUEST_FLAG) <= '1';
00156
00157            when TIME_1 =>                          -- Start read from RAM for Amplitude
00158                async_flags(RAM_BUS_BUSY_FLAG) <= '1';
00159                async_flags(START_MEM_OP_FLAG) <= '1';
00160                async_flags(TIME_RD_FLAG) <= '1';
00161
00162            when TIME_2 =>                          -- wait for RAM Op to complete
00163                async_flags(RAM_BUS_BUSY_FLAG) <= '1';
00164                async_flags(TIME_CAPTURE_FLAG) <= '1';
00165
00166 ---------------------------------- End read from memory states ---------------------------------
00167
00168            when SET_SAMPLE_DATA =>
00169                if sample_count = num_samples and
     DAC_Module_start = '0' then
00170                    async_flags(VOLTAGE_DEFAULT_FLAG) <= '1';
```

```
00171                 else
00172                     async_flags(VOLTAGE_SET_FLAG) <= '1';
00173                 end if;
00174 
00175             when TX_START =>
00176                 async_flags(START_FLAG) <= '1';
00177                 async_flags(INC_COUNT_FLAG) <= '1';
00178 
00179             when TX_WAIT  =>
00180                 async_flags(WAIT_FLAG) <= '1';
00181                 --async_flags(START_FLAG) <= '1';
00182                 async_flags(RESET_COUNT_FLAG) <= '1';
00183 
00184             when WAIT_FOR_TIME =>                   -- adds delay to implement the time part of the
    amplitude:time pair
00185                 async_flags(TIME_COUNT_INC_FLAG) <= '1';
00186 
00187             when TX_COMPLETE =>
00188                 async_flags(TIME_COUNT_RESET_FLAG) <= '1';
00189                 if sample_count = num_samples then
00190                     async_flags(SAMPLE_COUNT_RESET_FLAG) <= '1';
00191                 else
00192                     async_flags(SAMPLE_COUNT_INC_FLAG) <= '1';
00193                 end if;
00194 
00195             when others =>
00196                 async_flags <= (others => '0');
00197         end case;
00198     end if;
00199 end process;
00200 -------------------------------------------------
00201         -- DAC_Module state machine
00202 DAC_Module_asynch_state: process(rst_n, curr_state,
    DAC_Module_start, count, RAM_Op_Done, SPI_Done,
    time_count, time_val, sample_count, num_samples,
    RAM_Bus_Grant)
00203 begin
00204    if rst_n = '0' then
00205         next_state <= INIT;
00206    else
00207         case curr_state is
00208 
00209             when INIT =>
00210                 next_state <= RESET_OP_START;
00211 
00212             when RESET_OP_START =>
00213                 if count = 9 then
00214                     next_state <= RESET_OP_WAIT;
00215                 else
00216                     next_state <= RESET_OP_START;
00217                 end if;
00218             when RESET_OP_WAIT =>
00219                 if SPI_Done = '1' then
00220                     next_state <= RESET_OP_COMPLETE;
00221                 else
00222                     next_state <= RESET_OP_WAIT;
00223                 end if;
00224             when RESET_OP_COMPLETE =>
00225                 if SPI_Done = '0' then
00226                     next_state <= INTERNAL_REF_REG_START;
00227                 else
00228                     next_state <= RESET_OP_COMPLETE;
00229                 end if;
00230 
00231             when INTERNAL_REF_REG_START =>
00232                 if count = 9 then
00233                     next_state <= INTERNAL_REF_REG_WAIT;
00234                 else
00235                     next_state <= INTERNAL_REF_REG_START;
00236                 end if;
00237 
00238             when INTERNAL_REF_REG_WAIT =>
00239                 if SPI_Done = '1' then
00240                     next_state <= INTERNAL_REF_REG_COMPLETE;
00241                 else
00242                     next_state <= INTERNAL_REF_REG_WAIT;
00243                 end if;
00244             when INTERNAL_REF_REG_COMPLETE =>
00245                 if SPI_Done = '0' then
```

```
00246                   next_state <= LDAC_REG_START;
00247               else
00248                   next_state <= INTERNAL_REF_REG_COMPLETE;
00249               end if;
00250
00251           when LDAC_REG_START =>
00252               if count = 9 then
00253                   next_state <= LDAC_REG_WAIT;
00254               else
00255                   next_state <= LDAC_REG_START;
00256               end if;
00257
00258           when LDAC_REG_WAIT =>
00259               if SPI_Done = '1' then
00260                   next_state <= LDAC_REG_COMPLETE;
00261               else
00262                   next_state <= LDAC_REG_WAIT;
00263               end if;
00264           when LDAC_REG_COMPLETE =>
00265               if SPI_Done = '0' then
00266                   next_state <= POWER_DAC_START;
00267               else
00268                   next_state <= LDAC_REG_COMPLETE;
00269               end if;
00270
00271           when POWER_DAC_START =>
00272               if count = 9 then
00273                   next_state <= POWER_DAC_WAIT;
00274               else
00275                   next_state <= POWER_DAC_START;
00276               end if;
00277
00278           when POWER_DAC_WAIT =>
00279               if SPI_Done = '1' then
00280                   next_state <= POWER_DAC_COMPLETE;
00281               else
00282                   next_state <= POWER_DAC_WAIT;
00283               end if;
00284
00285           when POWER_DAC_COMPLETE =>
00286               if SPI_Done = '0' then
00287                   next_state <= CLEAR_CODE_START;
00288               else
00289                   next_state <= POWER_DAC_COMPLETE;
00290               end if;
00291
00292           when CLEAR_CODE_START =>
00293               if count = 9 then
00294                   next_state <= CLEAR_CODE_WAIT;
00295               else
00296                   next_state <= CLEAR_CODE_START;
00297               end if;
00298
00299           when CLEAR_CODE_WAIT =>
00300               if SPI_Done = '1' then
00301                   next_state <= CLEAR_CODE_COMPLETE;
00302               else
00303                   next_state <= CLEAR_CODE_WAIT;
00304               end if;
00305           when CLEAR_CODE_COMPLETE =>
00306               if DAC_Module_start = '0' and SPI_Done = '0' then
00307                   next_state <= IDLE;
00308               else
00309                   next_state <= CLEAR_CODE_COMPLETE;
00310               end if;
00311
00312
00313           when IDLE =>
00314               if DAC_Module_start = '1' then
00315                   next_state <= NUM_SAMPLES_BR_WAIT;
00316               else
00317                   next_state <= IDLE;
00318               end if;
00319
00320 ---------------------------------- Adding read from memory states -------------------------------
00321
00322 -- Reply - num_samples
00323           when NUM_SAMPLES_BR_WAIT =>     -- wait for RAM_Bus_Grant
00324               if RAM_Bus_Grant = '1' then
```

```
00325                   next_state <= NUM_SAMPLES_1;
00326               else
00327                   next_state <= NUM_SAMPLES_BR_WAIT;
00328               end if;
00329
00330           when NUM_SAMPLES_1 =>                       -- Start read from RAM for num_samples
00331               next_state <= NUM_SAMPLES_2;
00332
00333           when NUM_SAMPLES_2 =>                       -- wait for RAM Op to complete
00334               if RAM_Op_Done = '1' then
00335                   next_state <= AMPLITUDE_BR_WAIT;
00336               else
00337                   next_state <= NUM_SAMPLES_2;
00338               end if;
00339
00340   -- Read Amplitude from RAM and write to TX_FIFO
00341           when AMPLITUDE_BR_WAIT =>           -- wait for RAM_Bus_Grant
00342               if RAM_Bus_Grant = '1' then
00343                   next_state <= AMPLITUDE_1;
00344               else
00345                   next_state <= AMPLITUDE_BR_WAIT;
00346               end if;
00347
00348           when AMPLITUDE_1 =>                 -- Start read from RAM for Amplitude
00349               next_state <= AMPLITUDE_2;
00350
00351           when AMPLITUDE_2 =>                 -- wait for RAM Op to complete
00352               if RAM_Op_Done = '1' then
00353                   next_state <= TIME_BR_WAIT;
00354               else
00355                   next_state <= AMPLITUDE_2;
00356               end if;
00357
00358   -- Read Time from RAM and write to TX_FIFO
00359           when TIME_BR_WAIT =>                    -- wait for RAM_Bus_Grant
00360               if RAM_Bus_Grant = '1' then
00361                   next_state <= TIME_1;
00362               else
00363                   next_state <= TIME_BR_WAIT;
00364               end if;
00365
00366           when TIME_1 =>                          -- Start read from RAM for Amplitude
00367               next_state <= TIME_2;
00368
00369           when TIME_2 =>                          -- wait for RAM Op to complete
00370               if RAM_Op_Done = '1' then
00371                   next_state <= SET_SAMPLE_DATA;
00372               else
00373                   next_state <= TIME_2;
00374               end if;
00375
00376 --------------------------------- End read from memory states ---------------------------------
00377
00378           when SET_SAMPLE_DATA =>
00379               next_state <= TX_START;
00380
00381           when TX_START =>
00382               if count = 9 then
00383                   next_state <= TX_WAIT;
00384               else
00385                   next_state <= TX_START;
00386               end if;
00387
00388           when TX_WAIT =>
00389               if SPI_Done = '1' then
00390                   next_state <= WAIT_FOR_TIME;
00391               else
00392                   next_state <= TX_WAIT;
00393               end if;
00394
00395           when WAIT_FOR_TIME =>                   -- adds delay to implement the time part of the
    amplitude:time pair
00396               if time_count = time_val then
00397                   next_state <= TX_COMPLETE;
00398               else
00399                   next_state <= WAIT_FOR_TIME;
00400               end if;
00401
00402           when TX_COMPLETE =>
```

```
00403                 if sample_count = num_samples and
      DAC_Module_start = '0' then
00404                     next_state <= IDLE;
00405                 elsif sample_count = num_samples and
      DAC_Module_start = '1' then
00406                     next_state <= NUM_SAMPLES_1;
00407                 else
00408                     next_state <= AMPLITUDE_1;
00409                 end if;
00410
00411             when OTHERS =>
00412                 next_state <= IDLE;
00413         end case;
00414    end if;
00415 end process;
00416
00417
00418 end Behavioral;
00419
```

DAC_Module_pkg.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 --
00019 -- Additional Comments:
00020 --
00021 ----------------------------------------------------------------------------------
00022 library IEEE;
00023 use IEEE.STD_LOGIC_1164.ALL;
00024 use IEEE.numeric_std.all;
00025 --use IEEE.STD_LOGIC_ARITH.ALL;
00026 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00027
00028 package DAC_Module_pkg is
00029
00030
00031 constant INIT                               : STD_LOGIC_VECTOR(7 downto 0) := x"00";
00032
00033 constant TX_START                           : STD_LOGIC_VECTOR(7 downto 0) := x"01";
00034 constant TX_WAIT                            : STD_LOGIC_VECTOR(7 downto 0) := x"02";
00035
00036 constant RESET_OP_START                   : STD_LOGIC_VECTOR(7 downto 0) := x"04";
00037 constant RESET_OP_WAIT                  : STD_LOGIC_VECTOR(7 downto 0) := x"05";
00038 constant RESET_OP_COMPLETE              : STD_LOGIC_VECTOR(7 downto 0) := x"06";
00039 constant INTERNAL_REF_REG_START        : STD_LOGIC_VECTOR(7 downto 0) := x"07";
00040 constant INTERNAL_REF_REG_WAIT     : STD_LOGIC_VECTOR(7 downto 0) := x"08";
00041 constant INTERNAL_REF_REG_COMPLETE : STD_LOGIC_VECTOR(7 downto 0) := x"09";
00042 constant LDAC_REG_START                    : STD_LOGIC_VECTOR(7 downto 0) := x"10";
00043 constant LDAC_REG_WAIT                   : STD_LOGIC_VECTOR(7 downto 0) := x"11";
00044 constant LDAC_REG_COMPLETE              : STD_LOGIC_VECTOR(7 downto 0) := x"12";
00045 constant POWER_DAC_START              : STD_LOGIC_VECTOR(7 downto 0) := x"13";
00046 constant POWER_DAC_WAIT                   : STD_LOGIC_VECTOR(7 downto 0) := x"14";
00047 constant POWER_DAC_COMPLETE           : STD_LOGIC_VECTOR(7 downto 0) := x"15";
00048 constant CLEAR_CODE_START               : STD_LOGIC_VECTOR(7 downto 0) := x"16";
00049 constant CLEAR_CODE_WAIT              : STD_LOGIC_VECTOR(7 downto 0) := x"17";
00050 constant CLEAR_CODE_COMPLETE          : STD_LOGIC_VECTOR(7 downto 0) := x"18";
00051 constant SET_SAMPLE_DATA              : STD_LOGIC_VECTOR(7 downto 0) := x"19";
00052
00053 constant IDLE                               : STD_LOGIC_VECTOR(7 downto 0) := x"20";
```

```
00054
00055 constant NUM_SAMPLES_BR_WAIT         : STD_LOGIC_VECTOR(7 downto 0) := x"50";
00056 constant NUM_SAMPLES_1                : STD_LOGIC_VECTOR(7 downto 0) := x"51";
00057 constant NUM_SAMPLES_2                : STD_LOGIC_VECTOR(7 downto 0) := x"52";
00058 constant AMPLITUDE_BR_WAIT            : STD_LOGIC_VECTOR(7 downto 0) := x"60";
00059 constant AMPLITUDE_1                  : STD_LOGIC_VECTOR(7 downto 0) := x"61";
00060 constant AMPLITUDE_2                  : STD_LOGIC_VECTOR(7 downto 0) := x"62";
00061 constant TIME_BR_WAIT                 : STD_LOGIC_VECTOR(7 downto 0) := x"70";
00062 constant TIME_1                       : STD_LOGIC_VECTOR(7 downto 0) := x"71";
00063 constant TIME_2                       : STD_LOGIC_VECTOR(7 downto 0) := x"72";
00064
00065 constant WAIT_FOR_TIME                : STD_LOGIC_VECTOR(7 downto 0) := x"80";
00066
00067 constant TX_COMPLETE                  : STD_LOGIC_VECTOR(7 downto 0) := x"FF";
00068
00069 constant START_FLAG                   : integer := 0;
00070 constant WAIT_FLAG                    : integer := 1;
00071 constant DONE_FLAG                    : integer := 2;
00072 constant RESET_OP_SET_FLAG            : integer := 3;
00073 constant INTERNAL_REF_REG_SET_FLAG : integer := 4;
00074 constant LDAC_REG_SET_FLAG            : integer := 5;
00075 constant POWER_DAC_SET_FLAG          : integer := 6;
00076 constant CLEAR_CODE_SET_FLAG         : integer := 7;
00077 constant IDLE_FLAG                    : integer := 8;
00078 constant VOLTAGE_SET_FLAG             : integer := 9;
00079 constant VOLTAGE_DEFAULT_FLAG         : integer := 10;
00080
00081 constant START_MEM_OP_FLAG            : integer := 11;
00082
00083 constant NUM_SAMPLES_RD_FLAG         : integer := 12;
00084 constant NUM_SAMPLES_CAPTURE_FLAG   : integer := 13;
00085
00086 constant AMPLITUDE_RD_FLAG            : integer := 14;
00087 constant AMPLITUDE_CAPTURE_FLAG      : integer := 15;
00088
00089 constant TIME_RD_FLAG                 : integer := 16;
00090 constant TIME_CAPTURE_FLAG            : integer := 17;
00091
00092 constant TIME_COUNT_INC_FLAG         : integer := 18;
00093 constant TIME_COUNT_RESET_FLAG     : integer := 19;
00094
00095 constant SAMPLE_COUNT_INC_FLAG     : integer := 20;
00096 constant SAMPLE_COUNT_RESET_FLAG     : integer := 21;
00097
00098 constant INC_COUNT_FLAG                  : integer := 22;
00099 constant RESET_COUNT_FLAG              : integer := 23;
00100
00101 constant RAM_BUS_REQUEST_FLAG          : integer := 24;
00102 constant RAM_BUS_BUSY_FLAG            : integer := 25;
00103
00104
00105
00106 end DAC_Module_pkg;
00107
00108 package body DAC_Module_pkg is
00109
00110 end DAC_Module_pkg;
00111
00112
```

Main.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
00005 -- Create Date:    19:43:56 01/27/2012
00006 -- Design Name:
00007 -- Module Name:    Main - Behavioral
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- Dependencies:
```

```
00014 --
00015 -- Revision:
00016 -- Revision 0.01 - File Created
00017 -- Additional Comments:
00018 --
00019 ----------------------------------------------------------------------------------
00020 library IEEE;
00021 use IEEE.STD_LOGIC_1164.ALL;
00022 use IEEE.STD_LOGIC_ARITH.ALL;
00023 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00024
00025 ---- Uncomment the following library declaration if instantiating
00026 ---- any Xilinx primitives in this code.
00027 --library UNISIM;
00028 --use UNISIM.VComponents.all;
00029
00030 entity Main is
00031     Port ( main_clk         : in  STD_LOGIC;
00032            reset          : in  STD_LOGIC;
00033
00034               -- USB Module IO
00035               USB_clk        : in  STD_LOGIC;
00036               Data           : out  STD_LOGIC_VECTOR (7 downto 0);
00037               FlagB_out      : out  STD_LOGIC;
00038               idle_out       : out  STD_LOGIC;
00039               --done_out         : out  STD_LOGIC;
00040           PktEnd          : out  STD_LOGIC;
00041           --FlagA           : in   STD_LOGIC;
00042           FlagB            : in   STD_LOGIC;
00043           --FlagC           : in   STD_LOGIC;
00044           SLRD             : out  STD_LOGIC;
00045           SLWR             : out  STD_LOGIC;
00046           SLOE             : out  STD_LOGIC;
00047              FIFOADDR_in     : in   STD_LOGIC_VECTOR (1 downto 0);
00048           FIFOADDR         : out  STD_LOGIC_VECTOR (1 downto 0);
00049
00050              --ADC_Capture IO
00051              CS                      : out STD_LOGIC;
00052              adcRANGE               : out STD_LOGIC;
00053              adcRESET               : out STD_LOGIC;
00054              adcSTDBY               : out STD_LOGIC;
00055              convStA                 : out STD_LOGIC;
00056              convStB                 : out STD_LOGIC;
00057              ovrSAMPLE                : out STD_LOGIC_VECTOR(2 downto 0);
00058              refSEL                  : out STD_LOGIC;
00059              sCLK                   : out STD_LOGIC;
00060              serSEL                   : out STD_LOGIC;
00061              doutA                    : in  STD_LOGIC;
00062              doutB                    : in  STD_LOGIC;
00063              Busy                   : in  STD_LOGIC;
00064
00065              --RS232_Module
00066              RX                       : in  STD_LOGIC;
00067              TX                       : out STD_LOGIC;
00068              RX_led                   : out STD_LOGIC;
00069              TX_led                   : out STD_LOGIC;
00070
00071              -- MT45W8MW16BGX Signals
00072              MT_ADDR                : out STD_LOGIC_VECTOR(22 downto 0);
00073              MT_DATA                : inout STD_LOGIC_VECTOR(15 downto 0);
00074              MT_OE                    : out STD_LOGIC; -- active low
00075              MT_WE                    : out STD_LOGIC; -- active low
00076              MT_ADV                   : out STD_LOGIC; -- active low
00077              MT_CLK                   : out STD_LOGIC; -- during asynch operation, hold the clock low
00078              MT_UB                    : out STD_LOGIC; -- active low
00079              MT_LB                    : out STD_LOGIC; -- active low
00080              MT_CE                    : out STD_LOGIC; -- active low
00081              MT_CRE                   : out STD_LOGIC; -- held low, active high
00082              --MT_WAIT               : in  STD_LOGIC; -- ignored
00083
00084              -- SPI Signals
00085              Stim_Active_led            : out STD_LOGIC_VECTOR(3 downto 0);
00086              DAC_Init_Done_led        : out STD_LOGIC;
00087              SPI_CLK                    : out STD_LOGIC;
00088           DAC_CS                    : out STD_LOGIC;
00089           MOSI                       : out STD_LOGIC
00090              );
00091 end Main;
00092
```

```
00093 architecture Behavioral of Main is
00094
00095 signal reset_inv                       : STD_LOGIC;
00096
00097
00098
00099 signal Channel1_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00100 signal Channel2_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00101 signal Channel3_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00102 signal Channel4_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00103 signal Channel5_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00104 signal Channel6_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00105 signal Channel7_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00106 signal Channel8_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00107
00108 signal Stimulation_reg              : STD_LOGIC_VECTOR(7 downto 0); -- current stim setting
       for each channel (currently only supports 1 at a time)
00109 signal Acquisition_reg              : STD_LOGIC_VECTOR(7 downto 0); -- current acq setting
      for each channel (currently only supports all or nothing)
00110
00111 ------------------------------------------------------------------------------
00112 ---------------------- DAC_Module --------------------------------------------
00113 ------------------------------------------------------------------------------
00114 signal DAC_RAM_Start_Op            : STD_LOGIC;
00115 signal DAC_RAM_WE                    : STD_LOGIC;
00116 signal DAC_RAM_ADDR                : STD_LOGIC_VECTOR(22 downto 0);
00117 signal DAC_RAM_DIN                  : STD_LOGIC_VECTOR(15 downto 0);
00118 signal DAC_RAM_Bus_Busy            : STD_LOGIC;
00119
00120 signal Stim_Active                   : STD_LOGIC_VECTOR(3 downto 0); -- shows stimulation active
       for channels 1-4;
00121
00122 component DAC_Module
00123         Port ( clk                   : in  STD_LOGIC;
00124            reset                       : in  STD_LOGIC;
00125               Stimulation                : in  STD_LOGIC_VECTOR(7 downto 0); -- Pulse to single
      stim, Hold to multi stim
00126               Stim_Active                : out STD_LOGIC_VECTOR(3 downto 0); -- shows
      stimulation active for channels 1-4
00127               Init_Complete           : out STD_LOGIC;
00128
00129               -- SPI Signals
00130               SPI_CLK                    : out STD_LOGIC;
00131            CS                             : out STD_LOGIC;
00132            MOSI                          : out STD_LOGIC;
00133
00134               -- RAM_Module Control
00135               RAM_Start_Op            : out STD_LOGIC;
00136               RAM_Op_Done              : in  STD_LOGIC;
00137               RAM_WE                     : out STD_LOGIC;
00138               RAM_ADDR                 : out STD_LOGIC_VECTOR(22 downto 0);
00139               RAM_DOUT                 : in  STD_LOGIC_VECTOR(15 downto 0);
00140               RAM_DIN                    : out   STD_LOGIC_VECTOR(15 downto 0);
00141
00142               -- RAM_Arbiter
00143               RAM_Bus_Request          : out STD_LOGIC_VECTOR(3 downto 0);
00144               RAM_Bus_Busy             : out STD_LOGIC;
00145               RAM_Bus_Grant           : in  STD_LOGIC_VECTOR(3 downto 0)
00146               );
00147 end component;
00148
00149
00150 ------------------------------------------------------------------------------
00151 ---------------------- RAM_Module --------------------------------------------
00152 ------------------------------------------------------------------------------
00153 signal RAM_Start_Op                : STD_LOGIC;
00154 signal RAM_Op_Done              : STD_LOGIC;
00155 signal RAM_WE                        : STD_LOGIC;
00156 signal RAM_ADDR                    : STD_LOGIC_VECTOR(22 downto 0);
00157 signal RAM_DOUT                    : STD_LOGIC_VECTOR(15 downto 0);
00158 signal RAM_DIN                   : STD_LOGIC_VECTOR(15 downto 0);
00159
00160 signal RAM_Bus_Request          : STD_LOGIC_VECTOR(7 downto 0);
00161 signal RAM_Bus_Busy            : STD_LOGIC;
00162 signal RAM_Bus_Grant             : STD_LOGIC_VECTOR(7 downto 0);
00163
00164 component RAM_Module
00165     Port ( clk                   : in  STD_LOGIC;
00166            reset                   : in  STD_LOGIC;
```

```
00167
00168               -- MT45W8MW16BGX Signals
00169               MT_ADDR                : out STD_LOGIC_VECTOR(22 downto 0);
00170               MT_DATA                : inout STD_LOGIC_VECTOR(15 downto 0);
00171               MT_OE                    : out STD_LOGIC; -- active low
00172               MT_WE                    : out STD_LOGIC; -- active low
00173               MT_ADV                  : out STD_LOGIC; -- active low
00174               MT_CLK                  : out STD_LOGIC; -- during asynch operation, hold the clock low
00175               MT_UB                    : out STD_LOGIC; -- active low
00176               MT_LB                    : out STD_LOGIC; -- active low
00177               MT_CE                    : out STD_LOGIC; -- active low
00178               MT_CRE                  : out STD_LOGIC; -- held low, active high
00179               --MT_WAIT              : in  STD_LOGIC; -- ignored
00180
00181               -- RAM_Module Control
00182               RAM_Start_Op           : in  STD_LOGIC;
00183               RAM_Op_Done             : out STD_LOGIC;
00184               RAM_WE                   : in  STD_LOGIC;
00185               RAM_ADDR               : in  STD_LOGIC_VECTOR(22 downto 0);
00186               RAM_DOUT               : out STD_LOGIC_VECTOR(15 downto 0);
00187               RAM_DIN                  : in    STD_LOGIC_VECTOR(15 downto 0);
00188
00189               -- RAM_Arbiter
00190               RAM_Bus_Request        : in  STD_LOGIC_VECTOR(7 downto 0);
00191               RAM_Bus_Busy           : in  STD_LOGIC;
00192               RAM_Bus_Grant         : out  STD_LOGIC_VECTOR(7 downto 0)
00193               );
00194 end component;
00195
00196 ---------------------------------------------------------------------------------
00197 ----------------------- FIFO Componenet -----------------------------------------
00198 ---------------------------------------------------------------------------------
00199 signal FIFO_DOUT                    : STD_LOGIC_VECTOR (7 downto 0);
00200 signal FIFO_RD_CLK                : STD_LOGIC;
00201 signal FIFO_RD_EN                  : STD_LOGIC;
00202 signal FIFO_EMPTY                  : STD_LOGIC;
00203 signal FIFO_ALMOST_EMPTY        : STD_LOGIC;
00204 signal FIFO_DIN                    : STD_LOGIC_VECTOR (7 downto 0);
00205 signal FIFO_WR_EN                   : STD_LOGIC;
00206 signal FIFO_WR_CLK                 : STD_LOGIC;
00207 signal FIFO_FULL                     : STD_LOGIC;
00208 signal FIFO_ALMOST_FULL          : STD_LOGIC;
00209 signal FIFO_PROG_EMPTY            : STD_LOGIC; -- set to 16
00210 signal FIFO_PROG_FULL           : STD_LOGIC; -- set to 32256
00211
00212 component FIFO
00213     port (
00214     din: IN std_logic_VECTOR(7 downto 0);
00215     rd_clk: IN std_logic;
00216     rd_en: IN std_logic;
00217     rst: IN std_logic;
00218     wr_clk: IN std_logic;
00219     wr_en: IN std_logic;
00220     almost_empty: OUT std_logic;
00221     almost_full: OUT std_logic;
00222     dout: OUT std_logic_VECTOR(7 downto 0);
00223     empty: OUT std_logic;
00224     full: OUT std_logic;
00225     prog_empty: OUT std_logic;
00226     prog_full: OUT std_logic);
00227 END component;
00228
00229
00230 ---------------------------------------------------------------------------------
00231 ---------------------- Synch_Slave_FIFO Componenet  --------------------------
00232 ---------------------------------------------------------------------------------
00233 component Synch_Slave_FIFO
00234     Port ( Clk                    : in    STD_LOGIC;
00235                reset                    : in    STD_LOGIC;
00236            Data                      : out   STD_LOGIC_VECTOR (7 downto 0);
00237               FlagB_out                : out   STD_LOGIC;
00238               idle_out                 : out   STD_LOGIC;
00239               --done_out                  : out   STD_LOGIC;
00240            PktEnd                   : out   STD_LOGIC;
00241            --FlagA                     : in    STD_LOGIC;
00242            FlagB                     : in    STD_LOGIC;
00243            --FlagC                     : in    STD_LOGIC;
00244            SLRD                       : out   STD_LOGIC;
00245            SLWR                       : out   STD_LOGIC;
```

```
00246            SLOE                    : out  STD_LOGIC;
00247               FIFOADDR_in           : in   STD_LOGIC_VECTOR (1 downto 0);
00248            FIFOADDR                 : out  STD_LOGIC_VECTOR (1 downto 0);
00249
00250               -- USB_FIFO signals
00251               FIFO_DOUT                : in   STD_LOGIC_VECTOR (7 downto 0);
00252               FIFO_RD_CLK              : out  STD_LOGIC;
00253               FIFO_RD_EN           : out  STD_LOGIC;
00254               FIFO_EMPTY           : in   STD_LOGIC;
00255               FIFO_ALMOST_EMPTY    : in   STD_LOGIC;
00256               FIFO_PROG_EMPTY        : in   STD_LOGIC
00257               );
00258 end component;
00259
00260 ----------------------------------------------------------------------------
00261 ---------------------- ADC_Module Componenet  ------------------------------
00262 ----------------------------------------------------------------------------
00263 component ADC_Module
00264 Port (    clk                    : in  STD_LOGIC;
00265            reset                    : in  STD_LOGIC;
00266               FIFO_DIN              : out STD_LOGIC_VECTOR (7 downto 0);        --Data going into
    FIFO
00267               FIFO_WR_EN           : out STD_LOGIC;
00268               FIFO_WR_CLK            : out STD_LOGIC;
00269               FIFO_FULL                : in  STD_LOGIC;
00270               FIFO_ALMOST_FULL  : in  STD_LOGIC;
00271               FIFO_PROG_FULL      : in  STD_LOGIC;
00272
00273               --ADC_Capture IO
00274               CS                       : out STD_LOGIC;
00275               adcRANGE               : out STD_LOGIC;
00276               adcRESET               : out STD_LOGIC;
00277               adcSTDBY               : out STD_LOGIC;
00278               convStA                 : out STD_LOGIC;
00279               convStB                 : out STD_LOGIC;
00280               ovrSAMPLE                : out STD_LOGIC_VECTOR(2 downto 0);
00281               refSEL                  : out STD_LOGIC;
00282               sCLK                    : out STD_LOGIC;
00283               serSEL                   : out STD_LOGIC;
00284               doutA                    : in  STD_LOGIC;
00285               doutB                    : in  STD_LOGIC;
00286               Busy                    : in  STD_LOGIC
00287               );
00288 end component;
00289
00290
00291 ----------------------------------------------------------------------------
00292 ---------------------- Command_Handler -------------------------------------
00293 ----------------------------------------------------------------------------
00294 signal CMD_RAM_Start_Op             : STD_LOGIC;
00295 signal CMD_RAM_WE                     : STD_LOGIC;
00296 signal CMD_RAM_ADDR                 : STD_LOGIC_VECTOR(22 downto 0);
00297 signal CMD_RAM_DIN                   : STD_LOGIC_VECTOR(15 downto 0);
00298 signal CMD_RAM_Bus_Busy             : STD_LOGIC;
00299
00300 component Command_Handler
00301     Port ( clk                        : in  STD_LOGIC;
00302            reset                        : in  STD_LOGIC;
00303
00304               Channel1_Config         : out STD_LOGIC_VECTOR(7 downto 0);
00305               Channel2_Config         : out STD_LOGIC_VECTOR(7 downto 0);
00306               Channel3_Config         : out STD_LOGIC_VECTOR(7 downto 0);
00307               Channel4_Config         : out STD_LOGIC_VECTOR(7 downto 0);
00308               Channel5_Config         : out STD_LOGIC_VECTOR(7 downto 0);
00309               Channel6_Config         : out STD_LOGIC_VECTOR(7 downto 0);
00310               Channel7_Config         : out STD_LOGIC_VECTOR(7 downto 0);
00311               Channel8_Config         : out STD_LOGIC_VECTOR(7 downto 0);
00312
00313               Stimulation                : out STD_LOGIC_VECTOR(7 downto 0); -- current stim
    setting for each channel (currently only supports 1 at a time)
00314               Acquisition                : out STD_LOGIC_VECTOR(7 downto 0); -- current acq
    setting for each channel (currently only supports all or nothing)
00315
00316               -- RX_FIFO Signals
00317               RX_FIFO_RD_CLK          : out STD_LOGIC;
00318               RX_FIFO_DOUT              : in  STD_LOGIC_VECTOR (7 downto 0);
00319               RX_FIFO_RD_EN            : out STD_LOGIC;
00320               RX_FIFO_EMPTY             : in  STD_LOGIC;
00321
```

```
00322               -- TX FIFO Signals
00323               TX_FIFO_WR_CLK          : out STD_LOGIC;
00324               TX_FIFO_DIN               : out STD_LOGIC_VECTOR(7 downto 0);
00325               TX_FIFO_WR_EN            : out STD_LOGIC;
00326
00327               -- RAM_Module Control
00328               RAM_Start_Op              : out STD_LOGIC;
00329               RAM_Op_Done               : in  STD_LOGIC;
00330               RAM_WE                     : out STD_LOGIC;
00331               RAM_ADDR                  : out STD_LOGIC_VECTOR(22 downto 0);
00332               RAM_DOUT                  : in  STD_LOGIC_VECTOR(15 downto 0);
00333               RAM_DIN                    : out   STD_LOGIC_VECTOR(15 downto 0);
00334
00335               -- RAM_Arbiter
00336               RAM_Bus_Request           : out STD_LOGIC;
00337               RAM_Bus_Busy              : out STD_LOGIC;
00338               RAM_Bus_Grant             : in STD_LOGIC
00339               );
00340 end component;
00341
00342
00343 ------------------------------------------------------------------------
00344 ---------------------- RS232_Module ------------------------------------
00345 ------------------------------------------------------------------------
00346 signal RX_FIFO_RD_CLK             : STD_LOGIC;
00347 signal RX_FIFO_DOUT                : STD_LOGIC_VECTOR (7 downto 0);
00348 signal RX_FIFO_RD_EN               : STD_LOGIC;
00349 signal RX_FIFO_EMPTY               : STD_LOGIC;
00350
00351 signal TX_FIFO_WR_CLK             : STD_LOGIC;
00352 signal TX_FIFO_DIN                 : STD_LOGIC_VECTOR(7 downto 0);
00353 signal TX_FIFO_WR_EN               : STD_LOGIC;
00354
00355 component RS232_Module
00356     Port ( clk                      : in  STD_LOGIC;
00357            reset                    : in  STD_LOGIC;
00358
00359                RX                       : in  STD_LOGIC;
00360                TX                       : out STD_LOGIC;
00361                RX_led                   : out STD_LOGIC;
00362                TX_led                   : out STD_LOGIC;
00363
00364                -- RX_FIFO Signals
00365                RX_FIFO_RD_CLK          : in  STD_LOGIC;
00366                RX_FIFO_DOUT             : out STD_LOGIC_VECTOR (7 downto 0);
00367                RX_FIFO_RD_EN           : in  STD_LOGIC;
00368                RX_FIFO_EMPTY           : out STD_LOGIC;
00369
00370                -- TX_FIFO Signals
00371                TX_FIFO_WR_CLK          : in  STD_LOGIC;
00372                TX_FIFO_DIN               : in  STD_LOGIC_VECTOR (7 downto 0);
00373                TX_FIFO_WR_EN           : in  STD_LOGIC
00374                );
00375 end component;
00376
00377 begin
00378 reset_inv <= not Channel1_Config_reg(7);
00379
00380 --reset_inv <= not reset;
00381
00382 Stim_Active_led <= Stim_Active;
00383
00384 ------------------------------------------------------------------------
00385 ---------------------- DAC_Module --------------------------------------
00386 ------------------------------------------------------------------------
00387
00388 DAC : DAC_Module
00389 port map(
00390     clk                                 => main_clk,
00391    reset                               => reset,
00392     Stimulation                         => Stimulation_reg,
00393     Stim_Active                         => Stim_Active,
00394     Init_Complete                       => DAC_Init_Done_led,
00395
00396     -- SPI Signals
00397     SPI_CLK                             => SPI_CLK,
00398    CS                                   => DAC_CS,
00399    MOSI                                 => MOSI,
00400
```

```
00401     -- RAM_Module Control
00402     RAM_Start_Op                          => DAC_RAM_Start_Op,
00403     RAM_Op_Done                            => RAM_Op_Done,
00404     RAM_WE                                      => DAC_RAM_WE,
00405     RAM_ADDR                                      => DAC_RAM_ADDR,
00406     RAM_DOUT                                  => RAM_DOUT,
00407     RAM_DIN                                 => DAC_RAM_DIN,
00408
00409     -- RAM_Arbiter
00410     RAM_Bus_Request                  => RAM_Bus_Request(4 downto 1),
00411     RAM_Bus_Busy                        => DAC_RAM_Bus_Busy,
00412     RAM_Bus_Grant                     => RAM_Bus_Grant(4 downto 1)
00413     );
00414
00415
00416 -------------------------------------------------------------------------------
00417 ---------------------- RAM_Module ---------------------------------------------
00418 -------------------------------------------------------------------------------
00419 RAM_Start_Op    <= DAC_RAM_Start_Op when
      DAC_RAM_Bus_Busy = '1' else
00420                      CMD_RAM_Start_Op;
00421
00422 RAM_WE            <= DAC_RAM_WE when DAC_RAM_Bus_Busy = '1' else
00423                      CMD_RAM_WE;
00424
00425 RAM_ADDR        <= DAC_RAM_ADDR when DAC_RAM_Bus_Busy = '1' else
00426                      CMD_RAM_ADDR;
00427
00428 RAM_DIN          <= DAC_RAM_DIN when DAC_RAM_Bus_Busy = '1' else
00429                      CMD_RAM_DIN;
00430
00431 RAM_Bus_Busy    <=  CMD_RAM_Bus_Busy or
      DAC_RAM_Bus_Busy;
00432
00433 Onboard_RAM : RAM_Module
00434 port map(
00435     clk                                  => main_clk,
00436    reset                                => reset,
00437
00438    -- MT45W8MW16BGX Signals
00439    MT_ADDR                                => MT_ADDR,
00440    MT_DATA                                => MT_DATA,
00441    MT_OE                                    => MT_OE,
00442    MT_WE                                    => MT_WE,
00443    MT_ADV                                 => MT_ADV,
00444    MT_CLK                                 => MT_CLK,
00445    MT_UB                                      => MT_UB,
00446    MT_LB                                      => MT_LB,
00447    MT_CE                                      => MT_CE,
00448    MT_CRE                                 => MT_CRE,
00449    --MT_WAIT                                    => MT_WAIT,
00450
00451    -- RAM_Module Control
00452    RAM_Start_Op                       => RAM_Start_Op,
00453    RAM_Op_Done                          => RAM_Op_Done,
00454    RAM_WE                                  => RAM_WE,
00455    RAM_ADDR                              => RAM_ADDR,
00456    RAM_DOUT                                => RAM_DOUT,
00457    RAM_DIN                                 => RAM_DIN,
00458
00459     -- RAM_Arbiter
00460     RAM_Bus_Request                 => RAM_Bus_Request,
00461     RAM_Bus_Busy                        => RAM_Bus_Busy,
00462     RAM_Bus_Grant                     => RAM_Bus_Grant
00463     );
00464
00465 -------------------------------------------------------------------------------
00466 ---------------------- RS232_Module -------------------------------------------
00467 -------------------------------------------------------------------------------
00468 RS232 : RS232_Module
00469 port map(
00470     clk                                  => main_clk,
00471    reset                                => reset,
00472
00473     RX                                       => RX,
00474     TX                                       => TX,
00475     RX_led                                 => RX_led,
00476     TX_led                                 => TX_led,
00477
```

```
00478     RX_FIFO_RD_CLK                        => RX_FIFO_RD_CLK,
00479     RX_FIFO_DOUT                            => RX_FIFO_DOUT,
00480     RX_FIFO_RD_EN                       => RX_FIFO_RD_EN,
00481     RX_FIFO_EMPTY                       => RX_FIFO_EMPTY,
00482
00483     TX_FIFO_WR_CLK                        => TX_FIFO_WR_CLK,
00484     TX_FIFO_DIN                          => TX_FIFO_DIN,
00485     TX_FIFO_WR_EN                       => TX_FIFO_WR_EN
00486     );
00487
00488 ------------------------------------------------------------------------------
00489 ---------------------- Command_Handler ---------------------------------------
00490 ------------------------------------------------------------------------------
00491 CMD_Handler : Command_Handler
00492 port map(
00493     clk                                   => main_clk,
00494    reset                                 => reset,
00495
00496     Channel1_Config                    => Channel1_Config_reg,
00497     Channel2_Config                    => Channel2_Config_reg,
00498     Channel3_Config                    => Channel3_Config_reg,
00499     Channel4_Config                    => Channel4_Config_reg,
00500     Channel5_Config                    => Channel5_Config_reg,
00501     Channel6_Config                    => Channel6_Config_reg,
00502     Channel7_Config                    => Channel7_Config_reg,
00503     Channel8_Config                    => Channel8_Config_reg,
00504
00505     Stimulation                                 => Stimulation_reg,
00506     Acquisition                                 => Acquisition_reg,
00507
00508     -- RX_FIFO Signals
00509     RX_FIFO_RD_CLK                        => RX_FIFO_RD_CLK,
00510     RX_FIFO_DOUT                            => RX_FIFO_DOUT,
00511     RX_FIFO_RD_EN                       => RX_FIFO_RD_EN,
00512     RX_FIFO_EMPTY                       => RX_FIFO_EMPTY,
00513
00514     -- TX FIFO Signals
00515     TX_FIFO_WR_CLK                        => TX_FIFO_WR_CLK,
00516     TX_FIFO_DIN                          => TX_FIFO_DIN,
00517     TX_FIFO_WR_EN                       => TX_FIFO_WR_EN,
00518
00519     -- RAM_Module Control
00520    RAM_Start_Op                      => CMD_RAM_Start_Op,
00521    RAM_Op_Done                            => RAM_Op_Done,
00522    RAM_WE                                  => CMD_RAM_WE,
00523    RAM_ADDR                                  => CMD_RAM_ADDR,
00524    RAM_DOUT                                 => RAM_DOUT,
00525    RAM_DIN                                   => CMD_RAM_DIN,
00526
00527     -- RAM_Arbiter
00528     RAM_Bus_Request                   => RAM_Bus_Request(0),
00529     RAM_Bus_Busy                         => CMD_RAM_Bus_Busy,
00530     RAM_Bus_Grant                       => RAM_Bus_Grant(0)
00531     );
00532
00533 ------------------------------------------------------------------------------
00534 ---------------------- ADC_Module --------------------------------------------
00535 ------------------------------------------------------------------------------
00536 ADC : ADC_Module
00537 port map(
00538     clk                        => main_clk,
00539    --reset                    => reset,
00540     reset                     => Channel1_Config_reg(7),
00541     FIFO_DIN                       => FIFO_DIN,
00542     FIFO_WR_EN                   => FIFO_WR_EN,
00543     FIFO_WR_CLK                 => FIFO_WR_CLK,
00544     FIFO_FULL                     => FIFO_FULL,
00545     FIFO_ALMOST_FULL          => FIFO_ALMOST_FULL,
00546     FIFO_PROG_FULL              => FIFO_PROG_FULL,
00547     CS                              => CS,
00548     adcRANGE                      => adcRANGE,
00549     adcRESET                      => adcRESET,
00550     adcSTDBY                      => adcSTDBY,
00551     convStA                     => convStA,
00552     convStB                     => convStB,
00553     ovrSAMPLE                  => ovrSAMPLE,
00554     refSEL                        => refSEL,
00555     sCLK                           => sCLK,
00556     serSEL                       => serSEL,
```

```
00557     doutA                      => doutA,
00558     doutB                      => doutB,
00559     Busy                        => Busy
00560     );
00561
00562
00563 ----------------------------------------------------------------------------------
00564 ---------------------- Synch_Slave_FIFO ------------------------------------------
00565 ----------------------------------------------------------------------------------
00566 USB_Module : Synch_Slave_FIFO
00567 port map(
00568     Clk                      => USB_clk,
00569     reset                   => reset,
00570    Data                         => Data,
00571     FlagB_out              => FlagB_out,
00572     idle_out                    => idle_out,
00573     --done_out                 => done_out,
00574    PktEnd                     => PktEnd,
00575    --FlagA                   => FlagA,
00576    FlagB                    => FlagB,
00577    --FlagC                    => FlagC,
00578    SLRD                         => SLRD,
00579    SLWR                         => SLWR,
00580    SLOE                         => SLOE,
00581     FIFOADDR_in              => FIFOADDR_in,
00582    FIFOADDR                  => FIFOADDR,
00583
00584     -- USB_FIFO signals
00585     FIFO_DOUT                  => FIFO_DOUT,
00586     FIFO_RD_CLK              => FIFO_RD_CLK,
00587     FIFO_RD_EN                => FIFO_RD_EN,
00588     FIFO_EMPTY                => FIFO_EMPTY,
00589     FIFO_ALMOST_EMPTY      => FIFO_ALMOST_EMPTY,
00590     FIFO_PROG_EMPTY          => FIFO_PROG_EMPTY
00591     );
00592
00593 ----------------------------------------------------------------------------------
00594 ---------------------- FIFO Port Map ---------------------------------------------
00595 ----------------------------------------------------------------------------------
00596 USB_FIFO : FIFO
00597 port map(
00598     din                 => FIFO_DIN,
00599     rd_clk              => FIFO_RD_CLK,
00600     rd_en                  => FIFO_RD_EN ,
00601     rst                 => reset_inv,
00602     wr_clk              => FIFO_WR_CLK,
00603     wr_en                  => FIFO_WR_EN ,
00604     almost_empty        => FIFO_ALMOST_EMPTY,
00605     almost_full         => FIFO_ALMOST_FULL,
00606     dout                    => FIFO_DOUT,
00607     empty                   => FIFO_EMPTY,
00608     full                    => FIFO_FULL,
00609     prog_empty          => FIFO_PROG_EMPTY,
00610     prog_full           => FIFO_PROG_FULL
00611     );
00612
00613 end Behavioral;
00614
```

Main_tb.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
00005 -- Create Date:   22:12:09 01/27/2012
00006 -- Design Name:
00007 -- Module Name:   C:/Users/Kyle/Desktop/Dump to Desktop/Thesis/Cypress Compatible FPGA
      Code/Data_Acquisition_Test/Main_tb.vhd
00008 -- Project Name:  Data_Acquisition_Test
00009 -- Target Device:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- VHDL Test Bench Created by ISE for module: Main
```

```
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 -- Revision 0.01 - File Created
00019 -- Additional Comments:
00020 --
00021 -- Notes:
00022 -- This testbench has been automatically generated using types std_logic and
00023 -- std_logic_vector for the ports of the unit under test.  Xilinx recommends
00024 -- that these types always be used for the top-level I/O of a design in order
00025 -- to guarantee that the testbench will bind correctly to the post-implementation
00026 -- simulation model.
00027 --------------------------------------------------------------------------------
00028 LIBRARY ieee;
00029 USE ieee.std_logic_1164.ALL;
00030 USE ieee.std_logic_unsigned.all;
00031 USE ieee.numeric_std.ALL;
00032
00033 -- Import our UART tester package
00034 use work.UART_behavioural_model.all;
00035
00036 ENTITY Main_tb IS
00037 END Main_tb;
00038
00039 ARCHITECTURE behavior OF Main_tb IS
00040
00041     -- Component Declaration for the Unit Under Test (UUT)
00042
00043 COMPONENT Main
00044     Port ( main_clk         : in  STD_LOGIC;
00045            reset          : in  STD_LOGIC;
00046
00047              -- USB Module IO
00048              USB_clk         : in  STD_LOGIC;
00049              Data          : out  STD_LOGIC_VECTOR (7 downto 0);
00050              FlagB_out        : out  STD_LOGIC;
00051              idle_out      : out  STD_LOGIC;
00052              --done_out        : out  STD_LOGIC;
00053           PktEnd         : out  STD_LOGIC;
00054           --FlagA          : in   STD_LOGIC;
00055           FlagB           : in   STD_LOGIC;
00056           --FlagC          : in   STD_LOGIC;
00057           SLRD            : out  STD_LOGIC;
00058           SLWR            : out  STD_LOGIC;
00059           SLOE            : out  STD_LOGIC;
00060              FIFOADDR_in    : in   STD_LOGIC_VECTOR (1 downto 0);
00061           FIFOADDR        : out  STD_LOGIC_VECTOR (1 downto 0);
00062
00063              --ADC_Capture IO
00064              CS                    : out STD_LOGIC;
00065              adcRANGE             : out STD_LOGIC;
00066              adcRESET             : out STD_LOGIC;
00067              adcSTDBY             : out STD_LOGIC;
00068              convStA               : out STD_LOGIC;
00069              convStB               : out STD_LOGIC;
00070              ovrSAMPLE               : out STD_LOGIC_VECTOR(2 downto 0);
00071              refSEL                : out STD_LOGIC;
00072              sCLK                 : out STD_LOGIC;
00073              serSEL                : out STD_LOGIC;
00074              doutA                 : in  STD_LOGIC;
00075              doutB                 : in  STD_LOGIC;
00076              --Busy                  : in  STD_LOGIC;
00077               --RS232_Module
00078              RX                    : in  STD_LOGIC;
00079              TX                    : out STD_LOGIC;
00080              RX_led                : out STD_LOGIC;
00081              TX_led                : out STD_LOGIC;
00082
00083              -- MT45W8MW16BGX Signals
00084              MT_ADDR              : out STD_LOGIC_VECTOR(22 downto 0);
00085              MT_DATA              : inout STD_LOGIC_VECTOR(15 downto 0);
00086              MT_OE                 : out STD_LOGIC; -- active low
00087              MT_WE                 : out STD_LOGIC; -- active low
00088              MT_ADV               : out STD_LOGIC; -- active low
00089              MT_CLK               : out STD_LOGIC; -- during asynch operation, hold the clock low
00090              MT_UB                 : out STD_LOGIC; -- active low
00091              MT_LB                 : out STD_LOGIC; -- active low
00092              MT_CE                 : out STD_LOGIC; -- active low
```

```
00093               MT_CRE                  : out STD_LOGIC; -- held low, active high
00094               --MT_WAIT             : in  STD_LOGIC; -- ignored
00095
00096               -- SPI Signals
00097               Stim_Active_led        : out STD_LOGIC_VECTOR(3 downto 0);
00098               DAC_Init_Done_led    : out STD_LOGIC;
00099               SPI_CLK                 : out STD_LOGIC;
00100            DAC_CS                  : out STD_LOGIC;
00101            MOSI                    : out STD_LOGIC
00102               );
00103 END COMPONENT;
00104
00105
00106    --Inputs
00107    signal main_clk      : std_logic := '0';
00108    signal reset           : std_logic := '1';
00109    signal USB_clk        : std_logic := '0';
00110    --signal FlagA          : std_logic := '0';
00111    signal FlagB           : std_logic := '1';
00112    --signal FlagC          : std_logic := '0';
00113    signal FIFOADDR_in    : std_logic_vector(1 downto 0) := (others => '0');
00114     --signal Busy               : std_logic:= '0';
00115     signal doutA           : std_logic:= '0';
00116     signal doutB           : std_logic:= '0';
00117     signal RX             : std_logic:= '1';
00118     --signal MT_WAIT            : std_logic:= '0';
00119
00120     --BiDirs
00121    signal MT_DATA        : std_logic_vector(15 downto 0);
00122
00123     --Outputs
00124    signal Data          : std_logic_vector(7 downto 0);
00125    signal FlagB_out       : std_logic;
00126    signal idle_out      : std_logic;
00127    --signal done_out        : std_logic;
00128    signal PktEnd          : std_logic;
00129    signal SLRD          : std_logic;
00130    signal SLWR          : std_logic;
00131    signal SLOE          : std_logic;
00132    signal FIFOADDR      : std_logic_vector(1 downto 0);
00133     signal CS            : std_logic;
00134     signal adcRANGE     : std_logic;
00135     signal adcRESET     : std_logic;
00136     signal adcSTDBY     : std_logic;
00137     signal convStA           : std_logic;
00138     signal convStB           : std_logic;
00139     signal ovrSAMPLE       : std_logic_vector(2 downto 0);
00140     signal refSEL         : std_logic;
00141     signal sCLK            : std_logic;
00142     signal serSEL         : std_logic;
00143     signal MT_ADDR       : std_logic_vector(22 downto 0);
00144    signal MT_OE           : std_logic;
00145    signal MT_WE           : std_logic;
00146    signal MT_ADV          : std_logic;
00147    signal MT_CLK          : std_logic;
00148    signal MT_UB           : std_logic;
00149    signal MT_LB           : std_logic;
00150    signal MT_CE           : std_logic;
00151    signal MT_CRE          : std_logic;
00152     signal TX             : std_logic;
00153    signal RX_LED          : std_logic;
00154    signal TX_LED          : std_logic;
00155     signal SPI_CLK           : std_logic;
00156    signal DAC_CS          : std_logic;
00157    signal MOSI          : std_logic;
00158     signal Stim_Active_led :std_logic_VECTOR(3 downto 0);
00159     signal DAC_Init_Done_led       : STD_LOGIC;
00160
00161    -- Clock period definitions
00162    constant main_clk_period : time := 20ns;
00163    constant USB_clk_period : time := 30ns;
00164
00165     signal MT_DATA_reg : STD_LOGIC_VECTOR(15 downto 0);
00166
00167 BEGIN
00168 MT_DATA              <= MT_DATA_reg when MT_WE = '1' else -- read op
00169                         (others => 'Z');
00170
00171     -- Instantiate the Unit Under Test (UUT)
```

```
00172    uut: Main PORT MAP (
00173           main_clk => main_clk,
00174           reset => reset,
00175           USB_clk => USB_clk ,
00176           Data => Data,
00177           FlagB_out => FlagB_out,
00178           idle_out => idle_out,
00179           --done_out => done_out,
00180           PktEnd  => PktEnd,
00181           --FlagA => FlagA,
00182           FlagB => FlagB,
00183           --FlagC => FlagC,
00184           SLRD => SLRD,
00185           SLWR => SLWR,
00186           SLOE => SLOE,
00187           FIFOADDR_in => FIFOADDR_in,
00188           FIFOADDR => FIFOADDR,
00189              CS                          => CS,
00190              adcRANGE                 => adcRANGE,
00191              adcRESET                 => adcRESET,
00192              adcSTDBY                 => adcSTDBY,
00193              convStA                   => convStA,
00194              convStB                   => convStB,
00195              ovrSAMPLE               => ovrSAMPLE,
00196              refSEL                        => refSEL,
00197              sCLK                         => sCLK,
00198              serSEL                     => serSEL,
00199              doutA                    => doutA,
00200              doutB                    => doutB,
00201             -- Busy                        => Busy,
00202              MT_ADDR => MT_ADDR,
00203           MT_DATA => MT_DATA ,
00204           MT_OE => MT_OE,
00205           MT_WE => MT_WE,
00206           MT_ADV => MT_ADV,
00207           MT_CLK => MT_CLK,
00208           MT_UB => MT_UB,
00209           MT_LB => MT_LB,
00210           MT_CE => MT_CE,
00211           MT_CRE => MT_CRE,
00212         --   MT_WAIT => MT_WAIT,
00213              RX                                  => RX,
00214              TX                                  => TX,
00215              RX_led                              => RX_led,
00216              TX_led                              => TX_led,
00217              Stim_Active_led                  => Stim_Active_led,
00218              DAC_Init_Done_led              => DAC_Init_Done_led,
00219              SPI_CLK                           => SPI_CLK,
00220              DAC_CS                             => DAC_CS,
00221              MOSI                              => MOSI
00222         );
00223
00224    -- Clock process definitions
00225    main_clk_process :process
00226    begin
00227         main_clk <= '0';
00228         wait for main_clk_period/2;
00229         main_clk <= '1';
00230         wait for main_clk_period/2;
00231    end process;
00232
00233    USB_clk_process :process
00234    begin
00235         USB_clk <= '0';
00236         wait for USB_clk_period/2;
00237         USB_clk <= '1';
00238         wait for USB_clk_period/2;
00239    end process;
00240
00241
00242    -- Stimulus process
00243    stim_proc: process
00244    begin
00245       -- hold reset state for 100ms.
00246         reset <= '0';
00247       wait for 60ns;
00248         reset <= '1';
00249         wait for 1ms;
00250         FlagB <= '0';
```

```
00251         wait for 1ms;
00252         FlagB <= '1';
00253
00254       --wait for main_clk_period*100;
00255
00256       -- insert stimulus here
00257
00258       wait;
00259     end process;
00260
00261     Testing: process
00262
00263     constant My_Baud_Rate: integer := 115200;
00264     -- Make a jacket procedure around UART_tx, for convenience
00265     procedure send (data: in std_logic_vector) is
00266     begin
00267     UART_tx(
00268     tx_line => RX,
00269     data => data,
00270     baud_rate => My_Baud_Rate
00271     );
00272     end;
00273
00274     variable D: std_logic_vector(7 downto 0);
00275
00276     begin
00277
00278     MT_DATA_reg <= x"0003";
00279
00280     -- Idle awhile
00281     wait for 150 us;
00282
00283     -- Send Config_Chan
00284 --  send(x"5A"); wait for 50 us;
00285 --  send(x"01"); wait for 50 us;
00286 --  send(x"00"); wait for 50 us;
00287 --  send(x"07"); wait for 50 us;
00288 --  send(x"01"); wait for 50 us;
00289 --  send(x"1F"); wait for 50 us;
00290 --  send(x"FF"); wait for 50 us;
00291
00292     -- Send Set_Waveform
00293     send(x"5A"); wait for 50 us;
00294     send(x"05"); wait for 50 us;
00295     send(x"00"); wait for 50 us;
00296     send(x"0B"); wait for 50 us;
00297     send(x"01"); wait for 50 us;
00298     send(x"01"); wait for 50 us;
00299     send(x"12"); wait for 50 us;
00300     send(x"34"); wait for 50 us;
00301     send(x"56"); wait for 50 us;
00302     send(x"78"); wait for 50 us;
00303     send(x"FF"); wait for 50 us;
00304
00305     wait for 100 us;
00306
00307
00308     -- Send Set_Stim to start multi stim
00309     send(x"5A"); wait for 50 us;
00310     send(x"07"); wait for 50 us;
00311     send(x"00"); wait for 50 us;
00312     send(x"07"); wait for 50 us;
00313     send(x"01"); wait for 50 us;
00314     send(x"01"); wait for 50 us;
00315     send(x"FF"); wait for 50 us;
00316
00317     wait for 200 us;
00318
00319     -- Send Set_Stim to stop multi stim
00320     send(x"5A"); wait for 50 us;
00321     send(x"07"); wait for 50 us;
00322     send(x"00"); wait for 50 us;
00323     send(x"07"); wait for 50 us;
00324     send(x"00"); wait for 50 us;
00325     send(x"00"); wait for 50 us;
00326     send(x"FF"); wait for 50 us;
00327
00328
00329
```

```
00330 --  -- Some more characters - use a walking-ones pattern:
00331 --  for i in D'range loop
00332 --  D := (others => '0');
00333 --  D(i) := '1';
00334 --  send(D);
00335 --  wait for 50 us;
00336 --  end loop;
00337 --
00338 --  -- Idle some more
00339 --  wait for 50 us;
00340 --
00341 --  -- And finally, just for fun, send a 10-bit character:
00342 --  send("1111100000");
00343
00344     wait; -- That's All Folks
00345     end process;
00346
00347     -- Busy
00348 --process(main_clk, reset)
00349 --begin
00350 --   if reset = '0' then
00351 --      Busy <= '0';
00352 --   elsif rising_edge(main_clk) and convStA = '0' then
00353 --      Busy <= '1';
00354 --  elsif rising_edge(main_clk) then
00355 --      Busy <= '0';
00356 --  end if;
00357 --end process;
00358
00359 END;
```

MSG_01_Config_Chan.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
00005 -- Create Date:    19:44:35 01/27/2012
00006 -- Design Name:
00007 -- Module Name:    ADC_Module - Behavioral
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- Dependencies:
00014 --
00015 -- Revision:
00016 -- Revision 0.01 - File Created
00017 -- Additional Comments:
00018 --
00019 ----------------------------------------------------------------------------------
00020 library IEEE;
00021 use IEEE.STD_LOGIC_1164.ALL;
00022 use IEEE.STD_LOGIC_ARITH.ALL;
00023 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00024
00025 ---- Uncomment the following library declaration if instantiating
00026 ---- any Xilinx primitives in this code.
00027 --library UNISIM;
00028 --use UNISIM.VComponents.all;
00029
00030 use work.MSG_01_Config_Chan_pkg.all;
00031
00032 entity MSG_01_Config_Chan is
00033     Port ( clk                      : in  STD_LOGIC;
00034            reset                    : in  STD_LOGIC;
00035               MSG_Start                 : in  STD_LOGIC;
00036               MSG_Complete             : out STD_LOGIC;
00037
00038               -- Header Information
00039               MSG_Channel               : in STD_LOGIC_VECTOR(7 downto 0);
00040
00041               -- Channel Configuration
00042               Channel1_Config        : out STD_LOGIC_VECTOR(7 downto 0);
```

```
00043               Channel2_Config        : out STD_LOGIC_VECTOR(7 downto 0);
00044               Channel3_Config        : out STD_LOGIC_VECTOR(7 downto 0);
00045               Channel4_Config        : out STD_LOGIC_VECTOR(7 downto 0);
00046               Channel5_Config        : out STD_LOGIC_VECTOR(7 downto 0);
00047               Channel6_Config        : out STD_LOGIC_VECTOR(7 downto 0);
00048               Channel7_Config        : out STD_LOGIC_VECTOR(7 downto 0);
00049               Channel8_Config        : out STD_LOGIC_VECTOR(7 downto 0);
00050
00051               -- RX_FIFO Signals
00052               RX_FIFO_DOUT             : in  STD_LOGIC_VECTOR (7 downto 0);
00053               RX_FIFO_RD_EN           : out STD_LOGIC;
00054               RX_FIFO_EMPTY            : in  STD_LOGIC;
00055
00056               -- TX FIFO Signals
00057               TX_FIFO_DIN               : out STD_LOGIC_VECTOR(7 downto 0);
00058               TX_FIFO_WR_EN           : out STD_LOGIC
00059               );
00060 end MSG_01_Config_Chan;
00061
00062 architecture Behavioral of MSG_01_Config_Chan is
00063
00064 signal Channel1_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00065 signal Channel2_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00066 signal Channel3_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00067 signal Channel4_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00068 signal Channel5_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00069 signal Channel6_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00070 signal Channel7_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00071 signal Channel8_Config_reg       : STD_LOGIC_VECTOR(7 downto 0);
00072
00073 signal async_flags                   : STD_LOGIC_VECTOR(15 downto 0);
00074 signal count                           : STD_LOGIC_VECTOR(7 downto 0);
00075
00076 signal reply_length                 : STD_LOGIC_VECTOR(7 downto 0);
00077 signal status                          : STD_LOGIC_VECTOR(7 downto 0) := x"45";
00078
00079 -------------- MSG_ID 0x01 - CONFIG_CHAN signals ------------------------------
00080 signal CONFIG_CHAN_return_reg     : STD_LOGIC_VECTOR(7 downto 0);
00081
00082
00083 component MSG_01_Config_Chan_states
00084 Port (  clk                          : in  STD_LOGIC;
00085             rst_n                      : in  STD_LOGIC;
00086             MSG_Start                 : in  STD_LOGIC;
00087             FIFO_EMPTY                   : in  STD_LOGIC;
00088             reply_length                : in  STD_LOGIC_VECTOR(7 downto 0);
00089             count                          : in  STD_LOGIC_VECTOR(7 downto 0);
00090             async_flags                 : out STD_LOGIC_VECTOR(15 downto 0) --flags to enable
    functions
00091             );
00092 end component;
00093
00094 begin
00095
00096 MSG_Complete <= async_flags(DONE_FLAG);
00097
00098 ------------------------------------------------------------------------------
00099 ---------------------- MSG_Reply Information ---------------------------------
00100 ------------------------------------------------------------------------------
00101 --CONFIG_CHAN_return_reg
00102 process(clk, reset)
00103 begin
00104    if reset = '0' then
00105         CONFIG_CHAN_return_reg <= (others => '0');
00106    elsif rising_edge(clk) then
00107         case MSG_Channel is
00108             when x"01" =>   CONFIG_CHAN_return_reg <=
    Channel1_Config_reg;
00109             when x"02" =>   CONFIG_CHAN_return_reg <=
    Channel2_Config_reg;
00110             when x"03" =>   CONFIG_CHAN_return_reg <=
    Channel3_Config_reg;
00111             when x"04" =>   CONFIG_CHAN_return_reg <=
    Channel4_Config_reg;
00112             when x"05" =>   CONFIG_CHAN_return_reg <=
    Channel5_Config_reg;
00113             when x"06" =>   CONFIG_CHAN_return_reg <=
    Channel6_Config_reg;
00114             when x"07" =>   CONFIG_CHAN_return_reg <=
```

```
      Channel7_Config_reg;
00115             when x"08" =>   CONFIG_CHAN_return_reg <=
      Channel8_Config_reg;
00116             when others => CONFIG_CHAN_return_reg <=
      Channel1_Config_reg;
00117         end case;
00118     end if;
00119 end process;
00120
00121 -- TX_FIFO_DIN
00122 process(clk, reset)
00123 begin
00124    if reset = '0' then
00125         TX_FIFO_DIN <= (others => '0');
00126    elsif rising_edge(clk) then
00127         if async_flags(SET_REPLY_BYTE_FLAG) = '1' then -- MSG_ID x"01" - CONFIG_CHAN
00128             reply_length <= x"07";
00129             case count is
00130                 when x"00" => TX_FIFO_DIN <= x"5A";                                -- Start Byte
00131                 when x"01" => TX_FIFO_DIN <= x"81";       -- Reply MSG_ID
00132                 when x"02" => TX_FIFO_DIN <= x"00";                                    -- Reply
      Length_H
00133                 when x"03" => TX_FIFO_DIN <= x"07";                                    -- Reply
      Length_L
00134                 when x"04" => TX_FIFO_DIN <= MSG_Channel;
      -- MSG_Channel
00135                 when x"05" => TX_FIFO_DIN <= CONFIG_CHAN_return_reg;
00136                 when x"06" => TX_FIFO_DIN <= x"11";
00137                 when others => TX_FIFO_DIN <= x"25";
00138             end case;
00139         elsif async_flags(READ_MESSAGE_FLAG) = '1' then
00140             TX_FIFO_DIN <= RX_FIFO_DOUT;
00141         end if;
00142     end if;
00143 end process;
00144
00145
00146 ---------------------------------------------------------------------------
00147 ---------------------- Channel Configuration Registers --------------------
00148 ---------------------------------------------------------------------------
00149 -- ChannelX_Config_reg
00150 process(clk, reset)
00151 begin
00152    if reset = '0' then
00153         Channel1_Config_reg <= (others => '0');
00154         Channel2_Config_reg <= (others => '0');
00155         Channel3_Config_reg <= (others => '0');
00156         Channel4_Config_reg <= (others => '0');
00157         Channel5_Config_reg <= (others => '0');
00158         Channel6_Config_reg <= (others => '0');
00159         Channel7_Config_reg <= (others => '0');
00160         Channel8_Config_reg <= (others => '0');
00161    elsif rising_edge(clk) then
00162         if async_flags(READ_MESSAGE_FLAG) = '1' and count = x"00" then          -- Config
      Channel
00163             case MSG_Channel is
00164                 when x"01" =>   Channel1_Config_reg <=
      RX_FIFO_DOUT;
00165                 when x"02" =>   Channel2_Config_reg <=
      RX_FIFO_DOUT;
00166                 when x"03" =>   Channel3_Config_reg <=
      RX_FIFO_DOUT;
00167                 when x"04" =>   Channel4_Config_reg <=
      RX_FIFO_DOUT;
00168                 when x"05" =>   Channel5_Config_reg <=
      RX_FIFO_DOUT;
00169                 when x"06" =>   Channel6_Config_reg <=
      RX_FIFO_DOUT;
00170                 when x"07" =>   Channel7_Config_reg <=
      RX_FIFO_DOUT;
00171                 when x"08" =>   Channel8_Config_reg <=
      RX_FIFO_DOUT;
00172                 when others =>
00173             end case;
00174         end if;
00175     end if;
00176 end process;
00177
00178 Channel1_Config <= Channel1_Config_reg;
```

```
00179 Channel2_Config <= Channel2_Config_reg;
00180 Channel3_Config <= Channel3_Config_reg;
00181 Channel4_Config <= Channel4_Config_reg;
00182 Channel5_Config <= Channel5_Config_reg;
00183 Channel6_Config <= Channel6_Config_reg;
00184 Channel7_Config <= Channel7_Config_reg;
00185 Channel8_Config <= Channel8_Config_reg;
00186
00187 ------------------------------------------------------------------------------
00188 ---------------------- RX_FIFO -----------------------------------------------
00189 ------------------------------------------------------------------------------
00190 -- RX_FIFO_RD_EN
00191 process(clk, reset)
00192 begin
00193    if reset = '0' then
00194         RX_FIFO_RD_EN <= '0';
00195    elsif rising_edge(clk) then
00196         if async_flags(RX_RD_EN_FLAG) = '1' then
00197             RX_FIFO_RD_EN <= '1';
00198         else
00199             RX_FIFO_RD_EN <= '0';
00200         end if;
00201     end if;
00202 end process;
00203
00204 ------------------------------------------------------------------------------
00205 ---------------------- TX_FIFO -----------------------------------------------
00206 ------------------------------------------------------------------------------
00207 -- TX_FIFO_WR_EN
00208 process(clk, reset)
00209 begin
00210    if reset = '0' then
00211         TX_FIFO_WR_EN <= '0';
00212    elsif rising_edge(clk) then
00213         if async_flags(TX_WR_EN_FLAG) = '1' then
00214             TX_FIFO_WR_EN <= '1';
00215         else
00216             TX_FIFO_WR_EN <= '0';
00217         end if;
00218     end if;
00219 end process;
00220
00221 ------------------------------------------------------------------------------
00222 ---------------------- MSG_01_Config_Chan_states -----------------------------
00223 ------------------------------------------------------------------------------
00224 states : MSG_01_Config_Chan_states
00225 port map(
00226     clk                             => clk,
00227     rst_n                          => reset,
00228     MSG_Start                      => MSG_Start,
00229     FIFO_EMPTY                        => RX_FIFO_EMPTY,
00230     reply_length                    => reply_length,
00231     count                              => count,
00232     async_flags                      => async_flags
00233     );
00234
00235 ------------------------------------------------------------------------------
00236 ---------------------- Counter -----------------------------------------------
00237 ------------------------------------------------------------------------------
00238
00239 -- count
00240 process(clk, reset)
00241 begin
00242    if reset = '0' then
00243         count <= (others => '0');
00244    elsif rising_edge(clk) then
00245         if async_flags(INC_COUNT_FLAG) = '1' then
00246             count <= count + 1;
00247         elsif async_flags(IDLE_FLAG) = '1' then
00248             count <= x"00";
00249         elsif async_flags(CLEAR_COUNT_FLAG) = '1' then
00250             count <= x"00";
00251         end if;
00252     end if;
00253 end process;
00254
00255
00256 end Behavioral;
00257
```

MSG_01_Config_Chan_main_states.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 --
00019 -- Additional Comments:
00020 --
00021 ----------------------------------------------------------------------------------
00022 library IEEE;
00023 use IEEE.STD_LOGIC_1164.ALL;
00024 use IEEE.numeric_std.all;
00025 --use IEEE.STD_LOGIC_ARITH.ALL;
00026 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00027
00028 use work.MSG_01_Config_Chan_pkg.all;
00029
00030 ---- Uncomment the following library declaration if instantiating
00031 ---- any Xilinx primitives in this code.
00032 library UNISIM;
00033 use UNISIM.VComponents.all;
00034
00035 entity MSG_01_Config_Chan_states is
00036 Port (  clk                         : in  STD_LOGIC;
00037             rst_n                     : in  STD_LOGIC;
00038             MSG_Start                 : in  STD_LOGIC;
00039             FIFO_EMPTY                   : in  STD_LOGIC;
00040             reply_length               : in  STD_LOGIC_VECTOR(7 downto 0);
00041             count                         : in  STD_LOGIC_VECTOR(7 downto 0);
00042             async_flags                : out STD_LOGIC_VECTOR(15 downto 0) --flags to enable
      functions
00043             );
00044 end MSG_01_Config_Chan_states;
00045
00046 architecture Behavioral of MSG_01_Config_Chan_states is
00047
00048         --Control signals
00049
00050 signal curr_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM current state
00051 signal next_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM next state
00052
00053 begin
00054 ----------------------------------------------------------------------------------
00055         -- synchronous part of state machine here
00056 data_in_latch: process(clk, rst_n)
00057 begin
00058    if rst_n = '0' then
00059         curr_state <= (others => '0');
00060    elsif rising_edge(clk) then
00061         curr_state <= next_state;
00062    end if;
00063 end process;
00064
00065         -- async part of state machine to set function flags
00066 MSG_01_Config_Chan_state: process(rst_n, curr_state,
      reply_length, count)
00067 begin
00068    if rst_n = '0' then
00069         async_flags <= (others => '0');
00070    else
00071         async_flags <= (others => '0');
00072         case curr_state is
00073
00074             when IDLE =>
```

```
00075                 async_flags(IDLE_FLAG) <= '1';           -- init
00076
00077 -- Message Request Payload and Checksum
00078             when READ_MESSAGE =>
00079                 async_flags(READ_MESSAGE_FLAG) <= '1';
00080
00081             when INC_RX_FIFO =>
00082                     async_flags(INC_COUNT_FLAG) <= '1';
00083                     async_flags(RX_RD_EN_FLAG) <= '1';
00084                     --async_flags(TX_WR_EN_FLAG) <= '1';  -- uncomment to enable TX loopback of incoming
      messages (except for start byte)
00085
00086             when VALIDATE_MSG =>
00087                 async_flags(CLEAR_COUNT_FLAG) <= '1';
00088
00089
00090 -- Message Reply
00091             when SET_REPLY_BYTE =>
00092                 async_flags(SET_REPLY_BYTE_FLAG) <= '1';
00093
00094
00095             when SEND_REPLY_BYTE =>
00096                 if(count = reply_length) then
00097                 else
00098                     async_flags(TX_WR_EN_FLAG) <= '1';
00099                     async_flags(INC_COUNT_FLAG) <= '1';
00100                 end if;
00101
00102
00103             when FINISH =>
00104                 async_flags(DONE_FLAG) <= '1';           -- done flag
00105             when others =>
00106                 async_flags <= (others => '0');
00107         end case;
00108     end if;
00109 end process;
00110
00111
00112 ------------------------------------------------------------------------------------------------------------------------
00113
------------------------------------------------------------------------------------------------------------------------
00114 -- MSG_01_ConfigChan state machine
00115
------------------------------------------------------------------------------------------------------------------------
00116
------------------------------------------------------------------------------------------------------------------------
00117 MSG_01_Config_Chan_asynch_state: process(rst_n,
      curr_state, count, MSG_Start, FIFO_EMPTY,
      reply_length)
00118 begin
00119    if rst_n = '0' then
00120         next_state <= IDLE;
00121    else
00122         case curr_state is
00123         when IDLE =>
00124                 if MSG_Start = '1' then
00125                     next_state <= DELAY_STATE;
00126                 else
00127                     next_state <= IDLE;
00128                 end if;
00129
00130             when DELAY_STATE =>
00131                 if count = 2 then
00132                     next_state <= VALIDATE_MSG;
00133                 else
00134                     next_state <= WAIT_FOR_NEXT_BYTE;
00135                 end if;
00136
00137             when WAIT_FOR_NEXT_BYTE =>
00138                 if FIFO_EMPTY = '0' then
00139                     next_state <= READ_MESSAGE;
00140                 else
00141                     next_state <= WAIT_FOR_NEXT_BYTE;
00142                 end if;
00143
00144             when READ_MESSAGE =>
00145                 next_state <= INC_RX_FIFO;
00146
```

```
00147             when INC_RX_FIFO =>
00148                 next_state <= DELAY_STATE;
00149
00150             when VALIDATE_MSG =>
00151                 next_state <= SET_REPLY_BYTE;
00152
00153
    ----------------------------------------------------------------------------------------------------------------
00154 -- Message Reply
00155
    ----------------------------------------------------------------------------------------------------------------
00156             when SET_REPLY_BYTE =>
00157                 next_state <= SEND_REPLY_BYTE;
00158
00159             when SEND_REPLY_BYTE =>
00160                 if count = reply_length then
00161                     next_state <= FINISH;
00162                 else
00163                     next_state <= SET_REPLY_BYTE;
00164                 end if;
00165
00166             when FINISH =>
00167                     next_state <= IDLE;
00168
00169             when OTHERS =>
00170                 next_state <= IDLE;
00171         end case;
00172    end if;
00173 end process;
00174
00175
00176 end Behavioral;
00177
00178
```

MSG_01_Config_Chan_pkg.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:     WMU - Thesis
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 --
00016 -- Dependencies:
00017 --
00018 -- Revision:
00019 -- Revision 0.01 - File Created
00020 -- Additional Comments:
00021 --
00022 ----------------------------------------------------------------------------------
00023 library IEEE;
00024 use IEEE.STD_LOGIC_1164.ALL;
00025 use IEEE.numeric_std.all;
00026 --use IEEE.STD_LOGIC_ARITH.ALL;
00027 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00028
00029 package MSG_01_Config_Chan_pkg is
00030
00031
00032 constant IDLE                                   : STD_LOGIC_VECTOR(7 downto 0) := x"00";
00033
00034 constant DELAY_STATE                          : STD_LOGIC_VECTOR(7 downto 0) := x"01";
00035 constant WAIT_FOR_NEXT_BYTE                 : STD_LOGIC_VECTOR(7 downto 0) := x"02";
00036 constant READ_MESSAGE                        : STD_LOGIC_VECTOR(7 downto 0) := x"03";
00037 constant INC_RX_FIFO                          : STD_LOGIC_VECTOR(7 downto 0) := x"04";
00038 constant VALIDATE_MSG                        : STD_LOGIC_VECTOR(7 downto 0) := x"05";
```

```
00039
00040 constant SET_REPLY_BYTE                        : STD_LOGIC_VECTOR(7 downto 0) := x"10";
00041 constant SEND_REPLY_BYTE                          : STD_LOGIC_VECTOR(7 downto 0) := x"11";
00042
00043
00044 constant FINISH                                     : STD_LOGIC_VECTOR(7 downto 0) := x"FF";
00045
00046
00047
00048 constant IDLE_FLAG                                : integer := 0;
00049
00050 constant INC_COUNT_FLAG                             : integer := 1;
00051 constant CLEAR_COUNT_FLAG                          : integer := 2;
00052
00053 constant RX_RD_EN_FLAG                             : integer := 3;
00054 constant READ_MESSAGE_FLAG                         : integer := 4;
00055
00056 constant SET_REPLY_BYTE_FLAG                      : integer := 5;
00057 constant TX_WR_EN_FLAG                             : integer := 6;
00058
00059
00060
00061 constant DONE_FLAG                                 : integer := 15;
00062
00063
00064
00065 end MSG_01_Config_Chan_pkg;
00066
00067 package body MSG_01_Config_Chan_pkg is
00068
00069 end MSG_01_Config_Chan_pkg;
00070
00071
```

MSG_GET_WAVEFORM.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
00005 -- Create Date:    19:44:35 01/27/2012
00006 -- Design Name:
00007 -- Module Name:    ADC_Module - Behavioral
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- Dependencies:
00014 --
00015 -- Revision:
00016 -- Revision 0.01 - File Created
00017 -- Additional Comments:
00018 --
00019 ----------------------------------------------------------------------------------
00020 library IEEE;
00021 use IEEE.STD_LOGIC_1164.ALL;
00022 use IEEE.STD_LOGIC_ARITH.ALL;
00023 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00024
00025 ---- Uncomment the following library declaration if instantiating
00026 ---- any Xilinx primitives in this code.
00027 --library UNISIM;
00028 --use UNISIM.VComponents.all;
00029
00030 use work.MSG_GET_WAVEFORM_pkg.all;
00031
00032 entity MSG_GET_WAVEFORM is
00033     Port ( clk                          : in  STD_LOGIC;
00034            reset                          : in  STD_LOGIC;
00035               MSG_Start                       : in  STD_LOGIC;
00036               MSG_Complete                 : out STD_LOGIC;
00037
00038               -- Header Information
00039               MSG_Channel                   : in STD_LOGIC_VECTOR(7 downto 0);
```

```
00040
00041               -- RX_FIFO Signals
00042               RX_FIFO_DOUT              : in  STD_LOGIC_VECTOR (7 downto 0);
00043               RX_FIFO_RD_EN             : out STD_LOGIC;
00044               RX_FIFO_EMPTY             : in  STD_LOGIC;
00045
00046               -- TX FIFO Signals
00047               TX_FIFO_DIN                 : out STD_LOGIC_VECTOR(7 downto 0);
00048               TX_FIFO_WR_EN             : out STD_LOGIC;
00049
00050               -- RAM_Module Control
00051               RAM_Start_Op              : out STD_LOGIC;
00052               RAM_Op_Done                : in  STD_LOGIC;
00053               RAM_WE                      : out STD_LOGIC;
00054               RAM_ADDR                  : out STD_LOGIC_VECTOR(22 downto 0);
00055               RAM_DOUT                  : in  STD_LOGIC_VECTOR(15 downto 0);
00056               RAM_DIN                    : out   STD_LOGIC_VECTOR(15 downto 0);
00057
00058               -- RAM_Arbiter
00059               RAM_Bus_Request            : out STD_LOGIC;
00060               RAM_Bus_Busy              : out STD_LOGIC;
00061               RAM_Bus_Grant             : in STD_LOGIC
00062               );
00063 end MSG_GET_WAVEFORM;
00064
00065 architecture Behavioral of MSG_GET_WAVEFORM is
00066
00067 signal async_flags                   : STD_LOGIC_VECTOR(21 downto 0);
00068 signal count                          : STD_LOGIC_VECTOR(7 downto 0);
00069
00070 signal reply_header_length           : STD_LOGIC_VECTOR(7 downto 0);
00071 signal status                         : STD_LOGIC_VECTOR(7 downto 0) := x"55";
00072 signal checksum                     : STD_LOGIC_VECTOR(7 downto 0) := x"FF";
00073
00074 -------------- MSG_ID 0x06 - GET_WAVEFORM signals ------------------------------
00075 signal num_samples                   : STD_LOGIC_VECTOR(7 downto 0);
00076 signal amplitude_reg                 : STD_LOGIC_VECTOR(15 downto 0);
00077 signal time_reg                      : STD_LOGIC_VECTOR(15 downto 0);
00078 signal RAM_ADDR_reg                  : STD_LOGIC_VECTOR(22 downto 0);
00079
00080
00081 component MSG_GET_WAVEFORM_states
00082 Port (  clk                          : in  STD_LOGIC;
00083             rst_n                      : in  STD_LOGIC;
00084             MSG_Start                  : in  STD_LOGIC;
00085             FIFO_EMPTY                    : in  STD_LOGIC;
00086             RAM_Op_Done                : in  STD_LOGIC;
00087             RAM_Bus_Grant              : in  STD_LOGIC;
00088             reply_header_length      : in  STD_LOGIC_VECTOR(7 downto 0);
00089             num_samples                  : in  STD_LOGIC_VECTOR(7 downto 0);
00090             count                          : in  STD_LOGIC_VECTOR(7 downto 0);
00091             async_flags                  : out STD_LOGIC_VECTOR(21 downto 0) --flags to enable
    functions
00092             );
00093 end component;
00094
00095 begin
00096
00097 MSG_Complete <= async_flags(DONE_FLAG);
00098
00099 ---------------------------------------------------------------------------------
00100 ---------------------- Waveform Information -------------------------------------
00101 ---------------------------------------------------------------------------------
00102 -- num_samples
00103 process(clk, reset)
00104 begin
00105    if reset = '0' then
00106         num_samples <= (others => '0');
00107    elsif rising_edge(clk) then
00108         if async_flags(NUM_SAMPLES_CAPTURE_FLAG) = '1' then
00109             num_samples <= RAM_DOUT(7 downto 0);
00110         end if;
00111     end if;
00112 end process;
00113
00114 -- amplitude_reg
00115 process(clk, reset)
00116 begin
00117    if reset = '0' then
```

```
00118         amplitude_reg <= (others => '0');
00119     elsif rising_edge(clk) then
00120         if async_flags(AMPLITUDE_CAPTURE_FLAG) = '1' then
00121             amplitude_reg <= RAM_DOUT;
00122         end if;
00123     end if;
00124 end process;
00125 
00126 -- time_reg
00127 process(clk, reset)
00128 begin
00129     if reset = '0' then
00130         time_reg <= (others => '0');
00131     elsif rising_edge(clk) then
00132         if async_flags(TIME_CAPTURE_FLAG) = '1' then
00133             time_reg <= RAM_DOUT;
00134         end if;
00135     end if;
00136 end process;
00137 
00138 ------------------------------------------------------------------------------
00139 ---------------------- MSG_Reply Information ---------------------------------
00140 ------------------------------------------------------------------------------
00141 
00142 -- TX_FIFO_DIN
00143 process(clk, reset)
00144 begin
00145     if reset = '0' then
00146         TX_FIFO_DIN <= (others => '0');
00147     elsif rising_edge(clk) then
00148         if async_flags(SET_REPLY_BYTE_FLAG) = '1' then
00149             reply_header_length <= x"06";
00150             case count is
00151                 when x"00" => TX_FIFO_DIN <= x"5A";                                  -- Start Byte
00152                 when x"01" => TX_FIFO_DIN <= x"86";                                      -- Reply
    MSG_ID
00153                 when x"02" => TX_FIFO_DIN <= x"00";                                      -- Reply
    Length_H
00154                 when x"03" => TX_FIFO_DIN <= x"07";                                      -- Reply
    Length_L
00155                 when x"04" => TX_FIFO_DIN <= MSG_Channel;                         --
    channel
00156                 when x"05" => TX_FIFO_DIN <= status;                               --
    status
00157                 --when x"06" => TX_FIFO_DIN <= x"FF";
00158                 when others => TX_FIFO_DIN <= x"25";
00159             end case;
00160         elsif async_flags(NUM_SAMPLES_WR_FLAG) = '1' then
00161             TX_FIFO_DIN <= num_samples;
00162         elsif async_flags(AMPLITUDE_H_WR_FLAG) = '1' then
00163             TX_FIFO_DIN <= amplitude_reg(15 downto 8);
00164         elsif async_flags(AMPLITUDE_L_WR_FLAG) = '1' then
00165             TX_FIFO_DIN <= amplitude_reg(7 downto 0);
00166         elsif async_flags(TIME_H_WR_FLAG) = '1' then
00167             TX_FIFO_DIN <= time_reg(15 downto 8);
00168         elsif async_flags(TIME_L_WR_FLAG) = '1' then
00169             TX_FIFO_DIN <= time_reg(7 downto 0);
00170         elsif async_flags(CHECKSUM_WR_FLAG) = '1' then
00171             TX_FIFO_DIN <= checksum;
00172         end if;
00173     end if;
00174 end process;
00175 
00176 ------------------------------------------------------------------------------
00177 ---------------------- RAM ---------------------------------------------------
00178 ------------------------------------------------------------------------------
00179 RAM_WE <= '1';  -- always in set to read
00180 RAM_ADDR <= RAM_ADDR_reg;
00181 RAM_DIN <= (others => '1'); -- RAM_DIN not used
00182 
00183 -- RAM_Start_Op
00184 process(clk, reset)
00185 begin
00186     if reset = '0' then
00187         RAM_Start_Op <= '0';
00188     elsif rising_edge(clk) then
00189         if async_flags(START_MEM_OP_FLAG) = '1' then
00190             RAM_Start_Op <= '1';
00191         else
```

```
00192             RAM_Start_Op <= '0';
00193         end if;
00194     end if;
00195 end process;
00196 
00197 -- RAM_ADDR_reg
00198 process(clk, reset)
00199 begin
00200    if reset = '0' then
00201         RAM_ADDR_reg <= (others => '0');
00202    elsif rising_edge(clk) then
00203         if async_flags(NUM_SAMPLES_RD_FLAG) = '1' then
00204             RAM_ADDR_reg(22 downto 16) <= (others => '0');
00205             case MSG_Channel is
00206                 when x"01" =>           RAM_ADDR_reg(15 downto 0) <= x"0000";
00207                 when x"02" =>           RAM_ADDR_reg(15 downto 0) <= x"1000";
00208                 when x"03" =>           RAM_ADDR_reg(15 downto 0) <= x"2000";
00209                 when x"04" =>           RAM_ADDR_reg(15 downto 0) <= x"3000";
00210                 when x"05" =>           RAM_ADDR_reg(15 downto 0) <= x"4000";
00211                 when x"06" =>           RAM_ADDR_reg(15 downto 0) <= x"5000";
00212                 when x"07" =>           RAM_ADDR_reg(15 downto 0) <= x"6000";
00213                 when x"08" =>           RAM_ADDR_reg(15 downto 0) <= x"7000";
00214                 when others =>          RAM_ADDR_reg(15 downto 0) <= x"8000";
00215             end case;
00216         elsif async_flags(AMPLITUDE_RD_FLAG) = '1' then
00217             RAM_ADDR_reg <= RAM_ADDR_reg + 1;
00218         elsif async_flags(TIME_RD_FLAG) = '1' then
00219             RAM_ADDR_reg <= RAM_ADDR_reg + 1;
00220         end if;
00221     end if;
00222 end process;
00223 
00224 
00225 ---------------------------------------------------------------------------
00226 ---------------------- RX_FIFO --------------------------------------------
00227 ---------------------------------------------------------------------------
00228 -- RX_FIFO_RD_EN
00229 process(clk, reset)
00230 begin
00231    if reset = '0' then
00232         RX_FIFO_RD_EN <= '0';
00233    elsif rising_edge(clk) then
00234         if async_flags(RX_RD_EN_FLAG) = '1' then
00235             RX_FIFO_RD_EN <= '1';
00236         else
00237             RX_FIFO_RD_EN <= '0';
00238         end if;
00239     end if;
00240 end process;
00241 
00242 
00243 ---------------------------------------------------------------------------
00244 ---------------------- TX_FIFO --------------------------------------------
00245 ---------------------------------------------------------------------------
00246 -- TX_FIFO_WR_EN
00247 process(clk, reset)
00248 begin
00249    if reset = '0' then
00250         TX_FIFO_WR_EN <= '0';
00251    elsif rising_edge(clk) then
00252         if async_flags(TX_WR_EN_FLAG) = '1' then
00253             TX_FIFO_WR_EN <= '1';
00254         else
00255             TX_FIFO_WR_EN <= '0';
00256         end if;
00257     end if;
00258 end process;
00259 
00260 
00261 ---------------------------------------------------------------------------
00262 ---------------------- RAM_Arbiter ----------------------------------------
00263 ---------------------------------------------------------------------------
00264 
00265 -- RAM_Bus_Request
00266 process(clk, reset)
00267 begin
00268    if reset = '0' then
00269         RAM_Bus_Request <= '0';
00270    elsif rising_edge(clk) then
```

```
00271         if async_flags(RAM_BUS_REQUEST_FLAG) = '1' then
00272             RAM_Bus_Request <= '1';
00273         else
00274             RAM_Bus_Request <= '0';
00275         end if;
00276     end if;
00277 end process;
00278 
00279 -- RAM_Bus_Busy
00280 process(clk, reset)
00281 begin
00282    if reset = '0' then
00283         RAM_Bus_Busy <= '0';
00284    elsif rising_edge(clk) then
00285         if async_flags(RAM_BUS_BUSY_FLAG) = '1' then
00286             RAM_Bus_Busy <= '1';
00287         else
00288             RAM_Bus_Busy <= '0';
00289         end if;
00290     end if;
00291 end process;
00292 
00293 
00294 ---------------------------------------------------------------------------
00295 ---------------------- MSG_GET_WAVEFORM_states -----------------------------
00296 ---------------------------------------------------------------------------
00297 states : MSG_GET_WAVEFORM_states
00298 port map(
00299     clk                             => clk,
00300     rst_n                          => reset,
00301     MSG_Start                       => MSG_Start,
00302     FIFO_EMPTY                        => RX_FIFO_EMPTY,
00303     RAM_Op_Done                      => RAM_Op_Done,
00304     RAM_Bus_Grant                   => RAM_Bus_Grant,
00305     reply_header_length          => reply_header_length,
00306     num_samples                       => num_samples,
00307     count                               => count,
00308     async_flags                       => async_flags
00309     );
00310 
00311 ---------------------------------------------------------------------------
00312 ------------------------ Counter ------------------------------------------
00313 ---------------------------------------------------------------------------
00314 
00315 -- count
00316 process(clk, reset)
00317 begin
00318    if reset = '0' then
00319         count <= (others => '0');
00320    elsif rising_edge(clk) then
00321         if async_flags(INC_COUNT_FLAG) = '1' then
00322             count <= count + 1;
00323         elsif async_flags(IDLE_FLAG) = '1' then
00324             count <= x"00";
00325         elsif async_flags(CLEAR_COUNT_FLAG) = '1' then
00326             count <= x"00";
00327         end if;
00328     end if;
00329 end process;
00330 
00331 
00332 end Behavioral;
00333 
```

MSG_GET_WAVEFORM_main_states.vhd

```
00001 ---------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
```

```
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 --
00019 -- Additional Comments:
00020 --
00021 ----------------------------------------------------------------------------------
00022 library IEEE;
00023 use IEEE.STD_LOGIC_1164.ALL;
00024 use IEEE.numeric_std.all;
00025 --use IEEE.STD_LOGIC_ARITH.ALL;
00026 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00027
00028 use work.MSG_GET_WAVEFORM_pkg.all;
00029
00030 ---- Uncomment the following library declaration if instantiating
00031 ---- any Xilinx primitives in this code.
00032 library UNISIM;
00033 use UNISIM.VComponents.all;
00034
00035 entity MSG_GET_WAVEFORM_states is
00036 Port (  clk                         : in  STD_LOGIC;
00037             rst_n                     : in  STD_LOGIC;
00038             MSG_Start                 : in  STD_LOGIC;
00039             FIFO_EMPTY                   : in  STD_LOGIC;
00040             RAM_Op_Done                : in  STD_LOGIC;
00041             RAM_Bus_Grant              : in  STD_LOGIC;
00042             reply_header_length      : in  STD_LOGIC_VECTOR(7 downto 0);
00043             num_samples                : in  STD_LOGIC_VECTOR(7 downto 0);
00044             count                        : in  STD_LOGIC_VECTOR(7 downto 0);
00045             async_flags                : out STD_LOGIC_VECTOR(21 downto 0) --flags to enable
      functions
00046             );
00047 end MSG_GET_WAVEFORM_states;
00048
00049 architecture Behavioral of MSG_GET_WAVEFORM_states is
00050
00051         --Control signals
00052
00053 signal curr_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM current state
00054 signal next_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM next state
00055
00056 begin
00057 ----------------------------------------------------------------------------
00058         -- synchronous part of state machine here
00059 data_in_latch: process(clk, rst_n)
00060 begin
00061    if rst_n = '0' then
00062         curr_state <= (others => '0');
00063    elsif rising_edge(clk) then
00064         curr_state <= next_state;
00065    end if;
00066 end process;
00067
00068         -- async part of state machine to set function flags
00069 MSG_GET_WAVEFORM_state: process(rst_n, curr_state,
      reply_header_length, count)
00070 begin
00071    if rst_n = '0' then
00072         async_flags <= (others => '0');
00073    else
00074         async_flags <= (others => '0');
00075         case curr_state is
00076
00077             when IDLE =>
00078                 async_flags(IDLE_FLAG) <= '1';           -- init
00079
00080             when VALIDATE_MSG =>
00081
00082
    ------------------------------------------------------------------------------------------------------------------
00083 -- Message Reply
00084
    ------------------------------------------------------------------------------------------------------------------
```

```
00085     -- Reply Header
00086             when REPLY_HEADER_SET =>
00087                 async_flags(SET_REPLY_BYTE_FLAG) <= '1';
00088
00089             when REPLY_HEADER_SEND =>
00090                 if count = reply_header_length then
00091                 else
00092                     async_flags(TX_WR_EN_FLAG) <= '1';
00093                     async_flags(INC_COUNT_FLAG) <= '1';
00094                 end if;
00095
00096     -- Reply - num_samples
00097             when NUM_SAMPLES_BR_WAIT =>     -- wait for RAM_Bus_Grant
00098                 async_flags(RAM_BUS_REQUEST_FLAG) <= '1';
00099
00100            when NUM_SAMPLES_1 =>                        -- Start read from RAM for num_samples
00101                async_flags(RAM_BUS_BUSY_FLAG) <= '1';
00102                async_flags(START_MEM_OP_FLAG) <= '1';
00103                async_flags(NUM_SAMPLES_RD_FLAG) <= '1';
00104                async_flags(CLEAR_COUNT_FLAG) <= '1';
00105
00106            when NUM_SAMPLES_2 =>                        -- wait for RAM Op to complete
00107                async_flags(RAM_BUS_BUSY_FLAG) <= '1';
00108                async_flags(NUM_SAMPLES_CAPTURE_FLAG) <= '1';
00109
00110            when NUM_SAMPLES_3 =>                        -- Set TX_FIFO_DIN to num_samples
00111                async_flags(NUM_SAMPLES_WR_FLAG) <= '1';
00112
00113            when NUM_SAMPLES_4 =>                        -- TX_WR_EN
00114                async_flags(TX_WR_EN_FLAG) <= '1';
00115
00116    -- Reply - Loop1 - Amplitude:Time Pairs
00117            when LOOP1 =>
00118
00119            -- Read Amplitude from RAM and write to TX_FIFO
00120                    when AMPLITUDE_BR_WAIT =>
00121                        async_flags(RAM_BUS_REQUEST_FLAG) <= '1';
00122
00123                    when AMPLITUDE_1 =>                  -- Start read from RAM for Amplitude
00124                        async_flags(RAM_BUS_BUSY_FLAG) <= '1';
00125                        async_flags(START_MEM_OP_FLAG) <= '1';
00126                        async_flags(AMPLITUDE_RD_FLAG) <= '1';
00127
00128                    when AMPLITUDE_2 =>                  -- wait for RAM Op to complete
00129                        async_flags(RAM_BUS_BUSY_FLAG) <= '1';
00130                        async_flags(AMPLITUDE_CAPTURE_FLAG) <= '1';
00131
00132                    when AMPLITUDE_3 =>                  -- Set TX_FIFO_DIN to Amplitude_H
00133                        async_flags(AMPLITUDE_H_WR_FLAG) <= '1';
00134
00135                    when AMPLITUDE_4 =>                  -- TX_WR_EN
00136                        async_flags(TX_WR_EN_FLAG) <= '1';
00137
00138                    when AMPLITUDE_5 =>                  -- Set TX_FIFO_DIN to Amplitude_L
00139                        async_flags(AMPLITUDE_L_WR_FLAG) <= '1';
00140
00141                    when AMPLITUDE_6 =>                  -- TX_WR_EN
00142                        async_flags(TX_WR_EN_FLAG) <= '1';
00143
00144            -- Read Time from RAM and write to TX_FIFO
00145                    when TIME_BR_WAIT =>
00146                        async_flags(RAM_BUS_REQUEST_FLAG) <= '1';
00147
00148                    when TIME_1 =>                           -- Start read from RAM for Amplitude
00149                        async_flags(RAM_BUS_BUSY_FLAG) <= '1';
00150                        async_flags(START_MEM_OP_FLAG) <= '1';
00151                        async_flags(TIME_RD_FLAG) <= '1';
00152
00153                    when TIME_2 =>                           -- wait for RAM Op to complete
00154                        async_flags(RAM_BUS_BUSY_FLAG) <= '1';
00155                        async_flags(TIME_CAPTURE_FLAG) <= '1';
00156
00157                    when TIME_3 =>                           -- Set TX_FIFO_DIN to Amplitude_H
00158                        async_flags(TIME_H_WR_FLAG) <= '1';
00159
00160                    when TIME_4 =>                           -- TX_WR_EN
00161                        async_flags(TX_WR_EN_FLAG) <= '1';
00162
00163                    when TIME_5 =>                           -- Set TX_FIFO_DIN to Amplitude_L
```

```
00164                       async_flags(TIME_L_WR_FLAG) <= '1';
00165
00166                   when TIME_6 =>                          -- TX_WR_EN
00167                       async_flags(TX_WR_EN_FLAG) <= '1';
00168
00169             when LOOP1_COMPLETE =>
00170                 async_flags(INC_COUNT_FLAG) <= '1';
00171
00172
00173     -- Reply - Checksum
00174             when WRITE_CHECKSUM_1 =>
00175                 async_flags(CHECKSUM_WR_FLAG) <= '1';
00176
00177             when WRITE_CHECKSUM_2 =>
00178                 async_flags(TX_WR_EN_FLAG) <= '1';
00179
00180             when FINISH =>
00181                 async_flags(DONE_FLAG) <= '1';           -- done flag
00182             when others =>
00183                 async_flags <= (others => '0');
00184         end case;
00185     end if;
00186 end process;
00187
00188
00189
    ------------------------------------------------------------------------------------------------------------------------
00190
    ------------------------------------------------------------------------------------------------------------------------
00191 -- MSG_GET_WAVEFORM state machine
00192
    ------------------------------------------------------------------------------------------------------------------------
00193
    ------------------------------------------------------------------------------------------------------------------------
00194 MSG_GET_WAVEFORM_asynch_state: process(rst_n,
    curr_state, count, MSG_Start, FIFO_EMPTY,
    reply_header_length, num_samples, RAM_Op_Done,
    RAM_Bus_Grant)
00195 begin
00196    if rst_n = '0' then
00197         next_state <= IDLE;
00198    else
00199         case curr_state is
00200         when IDLE =>
00201                 if MSG_Start = '1' then
00202                     next_state <= VALIDATE_MSG;
00203                 else
00204                     next_state <= IDLE;
00205                 end if;
00206
00207             when VALIDATE_MSG =>
00208                 next_state <= REPLY_HEADER_SET;
00209
00210
    ------------------------------------------------------------------------------------------------------------------------
00211 -- Message Reply
00212
    ------------------------------------------------------------------------------------------------------------------------
00213     -- Reply Header
00214             when REPLY_HEADER_SET =>
00215                 next_state <= REPLY_HEADER_SEND;
00216
00217             when REPLY_HEADER_SEND =>
00218                 if count = reply_header_length then
00219                     next_state <= NUM_SAMPLES_BR_WAIT;
00220                 else
00221                     next_state <= REPLY_HEADER_SET;
00222                 end if;
00223
00224     -- Reply - num_samples
00225             when NUM_SAMPLES_BR_WAIT =>     -- wait for RAM_Bus_Grant
00226                 if RAM_Bus_Grant = '1' then
00227                     next_state <= NUM_SAMPLES_1;
00228                 else
00229                     next_state <= NUM_SAMPLES_BR_WAIT;
00230                 end if;
00231
00232             when NUM_SAMPLES_1 =>                       -- Start read from RAM for num_samples
00233                 next_state <= NUM_SAMPLES_2;
```

```
00234
00235             when NUM_SAMPLES_2 =>                         -- wait for RAM Op to complete
00236                 if RAM_Op_Done = '1' then
00237                     next_state <= NUM_SAMPLES_3;
00238                 else
00239                     next_state <= NUM_SAMPLES_2;
00240                 end if;
00241
00242             when NUM_SAMPLES_3 =>                         -- Set TX_FIFO_DIN to num_samples
00243                 next_state <= NUM_SAMPLES_4;
00244
00245             when NUM_SAMPLES_4 =>                         -- TX_WR_EN
00246                 next_state <= LOOP1;
00247
00248     -- Reply - Loop1 - Amplitude:Time Pairs
00249             when LOOP1 =>
00250                 if count = num_samples then
00251                     next_state <= WRITE_CHECKSUM_1;
00252                 else
00253                     next_state <= AMPLITUDE_BR_WAIT;
00254                 end if;
00255
00256             -- Read Amplitude from RAM and write to TX_FIFO
00257                     when AMPLITUDE_BR_WAIT =>             -- wait for RAM_Bus_Grant
00258                         if RAM_Bus_Grant = '1' then
00259                             next_state <= AMPLITUDE_1;
00260                         else
00261                             next_state <= AMPLITUDE_BR_WAIT;
00262                         end if;
00263
00264                     when AMPLITUDE_1 =>                   -- Start read from RAM for Amplitude
00265                         next_state <= AMPLITUDE_2;
00266
00267                     when AMPLITUDE_2 =>                   -- wait for RAM Op to complete
00268                         if RAM_Op_Done = '1' then
00269                             next_state <= AMPLITUDE_3;
00270                         else
00271                             next_state <= AMPLITUDE_2;
00272                         end if;
00273
00274                     when AMPLITUDE_3 =>                   -- Set TX_FIFO_DIN to Amplitude_H
00275                         next_state <= AMPLITUDE_4;
00276
00277                     when AMPLITUDE_4 =>                   -- TX_WR_EN
00278                         next_state <= AMPLITUDE_5;
00279
00280                     when AMPLITUDE_5 =>                   -- Set TX_FIFO_DIN to Amplitude_L
00281                         next_state <= AMPLITUDE_6;
00282
00283                     when AMPLITUDE_6 =>                   -- TX_WR_EN
00284                         next_state <= TIME_BR_WAIT;
00285
00286             -- Read Time from RAM and write to TX_FIFO
00287                     when TIME_BR_WAIT =>                      -- wait for RAM_Bus_Grant
00288                         if RAM_Bus_Grant = '1' then
00289                             next_state <= TIME_1;
00290                         else
00291                             next_state <= TIME_BR_WAIT;
00292                         end if;
00293
00294                     when TIME_1 =>                            -- Start read from RAM for Time
00295                         next_state <= TIME_2;
00296
00297                     when TIME_2 =>                            -- wait for RAM Op to complete
00298                         if RAM_Op_Done = '1' then
00299                             next_state <= TIME_3;
00300                         else
00301                             next_state <= TIME_2;
00302                         end if;
00303
00304                     when TIME_3 =>                            -- Set TX_FIFO_DIN to Amplitude_H
00305                         next_state <= TIME_4;
00306
00307                     when TIME_4 =>                            -- TX_WR_EN
00308                         next_state <= TIME_5;
00309
00310                     when TIME_5 =>                            -- Set TX_FIFO_DIN to Amplitude_L
00311                         next_state <= TIME_6;
00312
```

```
00313                     when TIME_6 =>                            -- TX_WR_EN
00314                         next_state <= LOOP1_COMPLETE;
00315
00316             when LOOP1_COMPLETE =>
00317                 next_state <= LOOP1;
00318
00319
00320     -- Reply - Checksum
00321             when WRITE_CHECKSUM_1 =>
00322                 next_state <= WRITE_CHECKSUM_2;
00323
00324             when WRITE_CHECKSUM_2 =>
00325                 next_state <= FINISH;
00326
00327
00328             when FINISH =>
00329                     next_state <= IDLE;
00330
00331             when OTHERS =>
00332                 next_state <= IDLE;
00333         end case;
00334    end if;
00335 end process;
00336
00337
00338 end Behavioral;
00339
00340
```

MSG_GET_WAVEFORM_pkg.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:     WMU - Thesis
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 --
00016 -- Dependencies:
00017 --
00018 -- Revision:
00019 -- Revision 0.01 - File Created
00020 -- Additional Comments:
00021 --
00022 ----------------------------------------------------------------------------------
00023 library IEEE;
00024 use IEEE.STD_LOGIC_1164.ALL;
00025 use IEEE.numeric_std.all;
00026 --use IEEE.STD_LOGIC_ARITH.ALL;
00027 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00028
00029 package MSG_GET_WAVEFORM_pkg is
00030
00031
00032 constant IDLE                                   : STD_LOGIC_VECTOR(7 downto 0) := x"00";
00033
00034 constant VALIDATE_MSG                          : STD_LOGIC_VECTOR(7 downto 0) := x"10";
00035
00036 constant REPLY_HEADER_SET                      : STD_LOGIC_VECTOR(7 downto 0) := x"20";
00037 constant REPLY_HEADER_SEND                     : STD_LOGIC_VECTOR(7 downto 0) := x"21";
00038
00039 constant NUM_SAMPLES_BR_WAIT                   : STD_LOGIC_VECTOR(7 downto 0) := x"34";
00040 constant NUM_SAMPLES_1                         : STD_LOGIC_VECTOR(7 downto 0) := x"30";
00041 constant NUM_SAMPLES_2                         : STD_LOGIC_VECTOR(7 downto 0) := x"31";
00042 constant NUM_SAMPLES_3                         : STD_LOGIC_VECTOR(7 downto 0) := x"32";
00043 constant NUM_SAMPLES_4                         : STD_LOGIC_VECTOR(7 downto 0) := x"33";
00044
```

```
00045 constant LOOP1                                    : STD_LOGIC_VECTOR(7 downto 0) := x"40";
00046
00047 constant AMPLITUDE_BR_WAIT                      : STD_LOGIC_VECTOR(7 downto 0) := x"56";
00048 constant AMPLITUDE_1                             : STD_LOGIC_VECTOR(7 downto 0) := x"50";
00049 constant AMPLITUDE_2                             : STD_LOGIC_VECTOR(7 downto 0) := x"51";
00050 constant AMPLITUDE_3                             : STD_LOGIC_VECTOR(7 downto 0) := x"52";
00051 constant AMPLITUDE_4                             : STD_LOGIC_VECTOR(7 downto 0) := x"53";
00052 constant AMPLITUDE_5                             : STD_LOGIC_VECTOR(7 downto 0) := x"54";
00053 constant AMPLITUDE_6                             : STD_LOGIC_VECTOR(7 downto 0) := x"55";
00054
00055 constant TIME_BR_WAIT                            : STD_LOGIC_VECTOR(7 downto 0) := x"66";
00056 constant TIME_1                                   : STD_LOGIC_VECTOR(7 downto 0) := x"60";
00057 constant TIME_2                                   : STD_LOGIC_VECTOR(7 downto 0) := x"61";
00058 constant TIME_3                                   : STD_LOGIC_VECTOR(7 downto 0) := x"62";
00059 constant TIME_4                                   : STD_LOGIC_VECTOR(7 downto 0) := x"63";
00060 constant TIME_5                                   : STD_LOGIC_VECTOR(7 downto 0) := x"64";
00061 constant TIME_6                                   : STD_LOGIC_VECTOR(7 downto 0) := x"65";
00062
00063 constant LOOP1_COMPLETE                           : STD_LOGIC_VECTOR(7 downto 0) := x"70";
00064
00065 constant WRITE_CHECKSUM_1                         : STD_LOGIC_VECTOR(7 downto 0) := x"80";
00066 constant WRITE_CHECKSUM_2                         : STD_LOGIC_VECTOR(7 downto 0) := x"81";
00067
00068 constant FINISH                                   : STD_LOGIC_VECTOR(7 downto 0) := x"FF";
00069
00070
00071
00072 constant IDLE_FLAG                                : integer := 0;
00073
00074 constant INC_COUNT_FLAG                           : integer := 1;
00075 constant CLEAR_COUNT_FLAG                         : integer := 2;
00076 constant RX_RD_EN_FLAG                            : integer := 3;
00077 constant TX_WR_EN_FLAG                            : integer := 4;
00078 constant START_MEM_OP_FLAG                        : integer := 5;
00079
00080 constant NUM_SAMPLES_RD_FLAG                      : integer := 6;
00081 constant NUM_SAMPLES_WR_FLAG                      : integer := 7;
00082 constant NUM_SAMPLES_CAPTURE_FLAG                 : integer := 8;
00083
00084 constant AMPLITUDE_H_WR_FLAG                      : integer := 9;
00085 constant AMPLITUDE_L_WR_FLAG                      : integer := 10;
00086 constant AMPLITUDE_RD_FLAG                        : integer := 11;
00087 constant AMPLITUDE_CAPTURE_FLAG                   : integer := 12;
00088
00089 constant TIME_H_WR_FLAG                           : integer := 13;
00090 constant TIME_L_WR_FLAG                           : integer := 14;
00091 constant TIME_RD_FLAG                             : integer := 15;
00092 constant TIME_CAPTURE_FLAG                        : integer := 16;
00093
00094 constant CHECKSUM_WR_FLAG                         : integer := 17;
00095
00096 constant SET_REPLY_BYTE_FLAG                      : integer := 18;
00097
00098 constant RAM_BUS_REQUEST_FLAG                     : integer := 19;
00099 constant RAM_BUS_BUSY_FLAG                        : integer := 20;
00100
00101 constant DONE_FLAG                                : integer := 21;
00102
00103
00104
00105 end MSG_GET_WAVEFORM_pkg;
00106
00107 package body MSG_GET_WAVEFORM_pkg is
00108
00109 end MSG_GET_WAVEFORM_pkg;
00110
00111
```

MSG_SET_STIM.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
00005 -- Create Date:    19:44:35 01/27/2012
```

```
00006 -- Design Name:
00007 -- Module Name:    ADC_Module - Behavioral
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- Dependencies:
00014 --
00015 -- Revision:
00016 -- Revision 0.01 - File Created
00017 -- Additional Comments:
00018 --
00019 ----------------------------------------------------------------------------------
00020 library IEEE;
00021 use IEEE.STD_LOGIC_1164.ALL;
00022 use IEEE.STD_LOGIC_ARITH.ALL;
00023 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00024
00025 ---- Uncomment the following library declaration if instantiating
00026 ---- any Xilinx primitives in this code.
00027 --library UNISIM;
00028 --use UNISIM.VComponents.all;
00029
00030 use work.MSG_SET_STIM_pkg.all;
00031
00032 entity MSG_SET_STIM is
00033     Port ( clk                      : in  STD_LOGIC;
00034            reset                    : in  STD_LOGIC;
00035            MSG_Start                : in  STD_LOGIC;
00036            MSG_Complete             : out STD_LOGIC;
00037
00038            -- Header Information
00039            MSG_Channel              : in STD_LOGIC_VECTOR(7 downto 0);
00040
00041            -- Channel Configuration
00042            Stimulation              : out STD_LOGIC_VECTOR(7 downto 0);
00043
00044            -- RX_FIFO Signals
00045            RX_FIFO_DOUT             : in  STD_LOGIC_VECTOR (7 downto 0);
00046            RX_FIFO_RD_EN            : out STD_LOGIC;
00047            RX_FIFO_EMPTY            : in  STD_LOGIC;
00048
00049            -- TX FIFO Signals
00050            TX_FIFO_DIN              : out STD_LOGIC_VECTOR(7 downto 0);
00051            TX_FIFO_WR_EN            : out STD_LOGIC
00052            );
00053 end MSG_SET_STIM;
00054
00055 architecture Behavioral of MSG_SET_STIM is
00056
00057 signal Stimulation_reg              : STD_LOGIC_VECTOR(7 downto 0);  -- Current stim
      setting for each channel (currently only supports 1 at a time)
00058 signal Continuous_reg             : STD_LOGIC_VECTOR(7 downto 0);     -- Overwrites Stim reg,
       causing single pulse for those channels that
00059                                                                                    -- are not set for
      continuous stim
00060
00061 signal async_flags                 : STD_LOGIC_VECTOR(15 downto 0);
00062 signal count                       : STD_LOGIC_VECTOR(7 downto 0);
00063
00064 signal reply_length                : STD_LOGIC_VECTOR(7 downto 0);
00065 signal status                      : STD_LOGIC_VECTOR(7 downto 0) := x"45";
00066
00067 component MSG_SET_STIM_states
00068 Port (  clk                        : in  STD_LOGIC;
00069            rst_n                    : in  STD_LOGIC;
00070            MSG_Start                : in  STD_LOGIC;
00071            FIFO_EMPTY                  : in  STD_LOGIC;
00072            reply_length             : in  STD_LOGIC_VECTOR(7 downto 0);
00073            count                       : in  STD_LOGIC_VECTOR(7 downto 0);
00074            async_flags              : out STD_LOGIC_VECTOR(15 downto 0) --flags to enable
      functions
00075            );
00076 end component;
00077
00078 begin
00079
00080 MSG_Complete <= async_flags(DONE_FLAG);
```

```
00081
00082 ------------------------------------------------------------------------------
00083 ---------------------- MSG_Reply Information ---------------------------------
00084 ------------------------------------------------------------------------------
00085
00086 -- TX_FIFO_DIN
00087 process(clk, reset)
00088 begin
00089    if reset = '0' then
00090         TX_FIFO_DIN <= (others => '0');
00091    elsif rising_edge(clk) then
00092         if async_flags(SET_REPLY_BYTE_FLAG) = '1' then
00093             reply_length <= x"07";
00094             case count is
00095                 when x"00" => TX_FIFO_DIN <= x"5A";                              -- Start Byte
00096                 when x"01" => TX_FIFO_DIN <= x"87";                                  -- Reply
      MSG_ID
00097                 when x"02" => TX_FIFO_DIN <= x"00";                                  -- Reply
      Length_H
00098                 when x"03" => TX_FIFO_DIN <= x"07";                                  -- Reply
      Length_L
00099                 when x"04" => TX_FIFO_DIN <= MSG_Channel;                         --
      MSG_Channel
00100                 when x"05" => TX_FIFO_DIN <= Stimulation_reg;                --
      Send back current stim reg setting (only those with continuous will remain set)
00101                 when x"06" => TX_FIFO_DIN <= x"FF";
00102                 when others => TX_FIFO_DIN <= x"25";
00103             end case;
00104         elsif async_flags(READ_MESSAGE_FLAG) = '1' then
00105             TX_FIFO_DIN <= RX_FIFO_DOUT;
00106         end if;
00107     end if;
00108 end process;
00109
00110
00111 ------------------------------------------------------------------------------
00112 ---------------------- Stimulation_reg ---------------------------------------
00113 ------------------------------------------------------------------------------
00114 -- Continuous_reg
00115 process(clk, reset)
00116 begin
00117    if reset = '0' then
00118         Continuous_reg <= (others => '0');
00119    elsif rising_edge(clk) then
00120         if async_flags(READ_MESSAGE_FLAG) = '1' and count = x"00" then          -- Config
      Channel
00121             Continuous_reg <= RX_FIFO_DOUT;
00122         end if;
00123     end if;
00124 end process;
00125
00126 -- Stimulation_reg
00127 process(clk, reset)
00128 begin
00129    if reset = '0' then
00130         Stimulation_reg <= (others => '0');
00131    elsif rising_edge(clk) then
00132         if async_flags(READ_MESSAGE_FLAG) = '1' and count = x"00" then          -- Config
      Channel
00133             Stimulation_reg <= MSG_Channel;
00134         elsif async_flags(SET_CONTINUOUS_FLAG) = '1'  then       -- Config Channel
00135             Stimulation_reg <= Continuous_reg;
00136         end if;
00137     end if;
00138 end process;
00139
00140 Stimulation <= Stimulation_reg;
00141
00142
00143 ------------------------------------------------------------------------------
00144 ---------------------- RX_FIFO -----------------------------------------------
00145 ------------------------------------------------------------------------------
00146 -- RX_FIFO_RD_EN
00147 process(clk, reset)
00148 begin
00149    if reset = '0' then
00150         RX_FIFO_RD_EN <= '0';
00151    elsif rising_edge(clk) then
00152         if async_flags(RX_RD_EN_FLAG) = '1' then
```

```
00153             RX_FIFO_RD_EN <= '1';
00154         else
00155             RX_FIFO_RD_EN <= '0';
00156         end if;
00157     end if;
00158 end process;
00159 
00160 ---------------------------------------------------------------------------
00161 ---------------------- TX_FIFO ------------------------------------------
00162 ---------------------------------------------------------------------------
00163 -- TX_FIFO_WR_EN
00164 process(clk, reset)
00165 begin
00166    if reset = '0' then
00167         TX_FIFO_WR_EN <= '0';
00168    elsif rising_edge(clk) then
00169         if async_flags(TX_WR_EN_FLAG) = '1' then
00170             TX_FIFO_WR_EN <= '1';
00171         else
00172             TX_FIFO_WR_EN <= '0';
00173         end if;
00174     end if;
00175 end process;
00176 
00177 ---------------------------------------------------------------------------
00178 ---------------------- MSG_SINGLE_STIM_states ---------------------------
00179 ---------------------------------------------------------------------------
00180 states : MSG_SET_STIM_states
00181 port map(
00182     clk                             => clk,
00183     rst_n                          => reset,
00184     MSG_Start                      => MSG_Start,
00185     FIFO_EMPTY                        => RX_FIFO_EMPTY,
00186     reply_length                    => reply_length,
00187     count                              => count,
00188     async_flags                      => async_flags
00189     );
00190 
00191 ---------------------------------------------------------------------------
00192 ---------------------- Counter ------------------------------------------
00193 ---------------------------------------------------------------------------
00194 
00195 -- count
00196 process(clk, reset)
00197 begin
00198    if reset = '0' then
00199         count <= (others => '0');
00200    elsif rising_edge(clk) then
00201         if async_flags(INC_COUNT_FLAG) = '1' then
00202             count <= count + 1;
00203         elsif async_flags(IDLE_FLAG) = '1' then
00204             count <= x"00";
00205         elsif async_flags(CLEAR_COUNT_FLAG) = '1' then
00206             count <= x"00";
00207         end if;
00208     end if;
00209 end process;
00210 
00211 
00212 end Behavioral;
00213 
```

MSG_SET_STIM_main_states.vhd

```
00001 ---------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
```

```
00012 --
00013 --
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 --
00019 -- Additional Comments:
00020 --
00021 ----------------------------------------------------------------------------------
00022 library IEEE;
00023 use IEEE.STD_LOGIC_1164.ALL;
00024 use IEEE.numeric_std.all;
00025 --use IEEE.STD_LOGIC_ARITH.ALL;
00026 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00027
00028 use work.MSG_SET_STIM_pkg.all;
00029
00030 ---- Uncomment the following library declaration if instantiating
00031 ---- any Xilinx primitives in this code.
00032 library UNISIM;
00033 use UNISIM.VComponents.all;
00034
00035 entity MSG_SET_STIM_states is
00036 Port (  clk                          : in  STD_LOGIC;
00037             rst_n                      : in  STD_LOGIC;
00038             MSG_Start                  : in  STD_LOGIC;
00039             FIFO_EMPTY                    : in  STD_LOGIC;
00040             reply_length                : in  STD_LOGIC_VECTOR(7 downto 0);
00041             count                          : in  STD_LOGIC_VECTOR(7 downto 0);
00042             async_flags                  : out STD_LOGIC_VECTOR(15 downto 0) --flags to enable
      functions
00043             );
00044 end MSG_SET_STIM_states;
00045
00046 architecture Behavioral of MSG_SET_STIM_states is
00047
00048         --Control signals
00049
00050 signal curr_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM current state
00051 signal next_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM next state
00052
00053 begin
00054 ----------------------------------------------------------------------------------
00055         -- synchronous part of state machine here
00056 data_in_latch: process(clk, rst_n)
00057 begin
00058    if rst_n = '0' then
00059         curr_state <= (others => '0');
00060    elsif rising_edge(clk) then
00061         curr_state <= next_state;
00062    end if;
00063 end process;
00064
00065         -- async part of state machine to set function flags
00066 MSG_SET_STIM_state: process(rst_n, curr_state,
      reply_length, count)
00067 begin
00068    if rst_n = '0' then
00069         async_flags <= (others => '0');
00070    else
00071         async_flags <= (others => '0');
00072         case curr_state is
00073
00074             when IDLE =>
00075                 async_flags(IDLE_FLAG) <= '1';           -- init
00076
00077 -- Message Request Payload and Checksum
00078             when READ_MESSAGE =>
00079                 async_flags(READ_MESSAGE_FLAG) <= '1';
00080
00081             when INC_RX_FIFO =>
00082                    async_flags(INC_COUNT_FLAG) <= '1';
00083                    async_flags(RX_RD_EN_FLAG) <= '1';
00084                    --async_flags(TX_WR_EN_FLAG) <= '1';  -- uncomment to enable TX loopback of incoming
      messages (except for start byte)
00085                    if count = 0 then
00086                        async_flags(SET_CONTINUOUS_FLAG) <= '1';
00087                    end if;
```

```
00088
00089             when VALIDATE_MSG =>
00090                 async_flags(CLEAR_COUNT_FLAG) <= '1';
00091
00092
00093 -- Message Reply
00094             when SET_REPLY_BYTE =>
00095                 async_flags(SET_REPLY_BYTE_FLAG) <= '1';
00096
00097
00098             when SEND_REPLY_BYTE =>
00099                 if(count = reply_length) then
00100                 else
00101                     async_flags(TX_WR_EN_FLAG) <= '1';
00102                     async_flags(INC_COUNT_FLAG) <= '1';
00103                 end if;
00104
00105
00106             when FINISH =>
00107                 async_flags(DONE_FLAG) <= '1';           -- done flag
00108             when others =>
00109                 async_flags <= (others => '0');
00110         end case;
00111     end if;
00112 end process;
00113
00114
00115
    ------------------------------------------------------------------------------------------------------------------------
00116
    ------------------------------------------------------------------------------------------------------------------------
00117 -- MSG_SET_STIM state machine
00118
    ------------------------------------------------------------------------------------------------------------------------
00119
    ------------------------------------------------------------------------------------------------------------------------
00120 MSG_SET_STIM_asynch_state: process(rst_n,
    curr_state, count, MSG_Start, FIFO_EMPTY,
    reply_length)
00121 begin
00122    if rst_n = '0' then
00123         next_state <= IDLE;
00124    else
00125         case curr_state is
00126         when IDLE =>
00127                 if MSG_Start = '1' then
00128                     next_state <= DELAY_STATE;
00129                 else
00130                     next_state <= IDLE;
00131                 end if;
00132
00133             when DELAY_STATE =>
00134                 if count = 2 then
00135                     next_state <= VALIDATE_MSG;
00136                 else
00137                     next_state <= WAIT_FOR_NEXT_BYTE;
00138                 end if;
00139
00140             when WAIT_FOR_NEXT_BYTE =>
00141                 if FIFO_EMPTY = '0' then
00142                     next_state <= READ_MESSAGE;
00143                 else
00144                     next_state <= WAIT_FOR_NEXT_BYTE;
00145                 end if;
00146
00147             when READ_MESSAGE =>
00148                 next_state <= INC_RX_FIFO;
00149
00150             when INC_RX_FIFO =>
00151                 next_state <= DELAY_STATE;
00152
00153             when VALIDATE_MSG =>
00154                 next_state <= SET_REPLY_BYTE;
00155
00156
    ------------------------------------------------------------------------------------------------------------------------
00157 -- Message Reply
00158
    ------------------------------------------------------------------------------------------------------------------------
```

```
00159             when SET_REPLY_BYTE =>
00160                 next_state <= SEND_REPLY_BYTE;
00161
00162             when SEND_REPLY_BYTE =>
00163                 if count = reply_length then
00164                     next_state <= FINISH;
00165                 else
00166                     next_state <= SET_REPLY_BYTE;
00167                 end if;
00168
00169             when FINISH =>
00170                     next_state <= IDLE;
00171
00172             when OTHERS =>
00173                 next_state <= IDLE;
00174         end case;
00175    end if;
00176 end process;
00177
00178
00179 end Behavioral;
00180
00181
```

MSG_SET_STIM_pkg.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:     WMU - Thesis
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 --
00016 -- Dependencies:
00017 --
00018 -- Revision:
00019 -- Revision 0.01 - File Created
00020 -- Additional Comments:
00021 --
00022 ----------------------------------------------------------------------------------
00023 library IEEE;
00024 use IEEE.STD_LOGIC_1164.ALL;
00025 use IEEE.numeric_std.all;
00026 --use IEEE.STD_LOGIC_ARITH.ALL;
00027 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00028
00029 package MSG_SET_STIM_pkg is
00030
00031
00032 constant IDLE                                   : STD_LOGIC_VECTOR(7 downto 0) := x"00";
00033
00034 constant DELAY_STATE                          : STD_LOGIC_VECTOR(7 downto 0) := x"01";
00035 constant WAIT_FOR_NEXT_BYTE                 : STD_LOGIC_VECTOR(7 downto 0) := x"02";
00036 constant READ_MESSAGE                         : STD_LOGIC_VECTOR(7 downto 0) := x"03";
00037 constant INC_RX_FIFO                          : STD_LOGIC_VECTOR(7 downto 0) := x"04";
00038 constant VALIDATE_MSG                        : STD_LOGIC_VECTOR(7 downto 0) := x"05";
00039
00040 constant SET_REPLY_BYTE                     : STD_LOGIC_VECTOR(7 downto 0) := x"10";
00041 constant SEND_REPLY_BYTE                      : STD_LOGIC_VECTOR(7 downto 0) := x"11";
00042
00043
00044 constant FINISH                                 : STD_LOGIC_VECTOR(7 downto 0) := x"FF";
00045
00046
00047
00048 constant IDLE_FLAG                           : integer := 0;
00049
```

```
00050 constant INC_COUNT_FLAG                          : integer := 1;
00051 constant CLEAR_COUNT_FLAG                       : integer := 2;
00052
00053 constant RX_RD_EN_FLAG                          : integer := 3;
00054 constant READ_MESSAGE_FLAG                     : integer := 4;
00055
00056 constant SET_REPLY_BYTE_FLAG                 : integer := 5;
00057 constant TX_WR_EN_FLAG                          : integer := 6;
00058
00059 constant SET_CONTINUOUS_FLAG                : integer := 7;
00060
00061
00062
00063 constant DONE_FLAG                                : integer := 15;
00064
00065
00066
00067 end MSG_SET_STIM_pkg;
00068
00069 package body MSG_SET_STIM_pkg is
00070
00071 end MSG_SET_STIM_pkg;
00072
00073
```

MSG_SET_WAVEFORM.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
00005 -- Create Date:    19:44:35 01/27/2012
00006 -- Design Name:
00007 -- Module Name:    ADC_Module - Behavioral
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- Dependencies:
00014 --
00015 -- Revision:
00016 -- Revision 0.01 - File Created
00017 -- Additional Comments:
00018 --
00019 ----------------------------------------------------------------------------------
00020 library IEEE;
00021 use IEEE.STD_LOGIC_1164.ALL;
00022 use IEEE.STD_LOGIC_ARITH.ALL;
00023 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00024
00025 ---- Uncomment the following library declaration if instantiating
00026 ---- any Xilinx primitives in this code.
00027 --library UNISIM;
00028 --use UNISIM.VComponents.all;
00029
00030 use work.MSG_SET_WAVEFORM_pkg.all;
00031
00032 entity MSG_SET_WAVEFORM is
00033     Port ( clk                        : in  STD_LOGIC;
00034            reset                      : in  STD_LOGIC;
00035               MSG_Start                    : in  STD_LOGIC;
00036               MSG_Complete              : out STD_LOGIC;
00037
00038               -- Header Information
00039               MSG_Channel                : in STD_LOGIC_VECTOR(7 downto 0);
00040
00041               -- RX_FIFO Signals
00042               RX_FIFO_DOUT             : in  STD_LOGIC_VECTOR (7 downto 0);
00043               RX_FIFO_RD_EN            : out STD_LOGIC;
00044               RX_FIFO_EMPTY            : in  STD_LOGIC;
00045
00046               -- TX FIFO Signals
00047               TX_FIFO_DIN                : out STD_LOGIC_VECTOR(7 downto 0);
00048               TX_FIFO_WR_EN            : out STD_LOGIC;
```

```
00049
00050               -- RAM_Module Control
00051               RAM_Start_Op              : out STD_LOGIC;
00052               RAM_Op_Done                : in  STD_LOGIC;
00053               RAM_WE                      : out STD_LOGIC;
00054               RAM_ADDR                  : out STD_LOGIC_VECTOR(22 downto 0);
00055               RAM_DOUT                  : in  STD_LOGIC_VECTOR(15 downto 0);
00056               RAM_DIN                    : out   STD_LOGIC_VECTOR(15 downto 0);
00057
00058               -- RAM_Arbiter
00059               RAM_Bus_Request            : out STD_LOGIC;
00060               RAM_Bus_Busy              : out STD_LOGIC;
00061               RAM_Bus_Grant            : in STD_LOGIC
00062               );
00063 end MSG_SET_WAVEFORM;
00064
00065 architecture Behavioral of MSG_SET_WAVEFORM is
00066
00067
00068
00069 signal async_flags                  : STD_LOGIC_VECTOR(17 downto 0);
00070 signal count                         : STD_LOGIC_VECTOR(7 downto 0);
00071
00072 signal reply_length                : STD_LOGIC_VECTOR(7 downto 0);
00073 signal status                        : STD_LOGIC_VECTOR(7 downto 0) := x"45";
00074
00075 -------------- MSG_ID 0x05 - SET_WAVEFORM signals -----------------------------
00076 signal num_samples                  : STD_LOGIC_VECTOR(7 downto 0);
00077 signal amplitude_reg                : STD_LOGIC_VECTOR(15 downto 0);
00078 signal time_reg                     : STD_LOGIC_VECTOR(15 downto 0);
00079 signal RAM_ADDR_reg                 : STD_LOGIC_VECTOR(22 downto 0);
00080
00081
00082 component MSG_SET_WAVEFORM_states
00083 Port (  clk                         : in  STD_LOGIC;
00084             rst_n                     : in  STD_LOGIC;
00085             MSG_Start                 : in  STD_LOGIC;
00086             FIFO_EMPTY                  : in  STD_LOGIC;
00087             RAM_Op_Done                : in  STD_LOGIC;
00088             RAM_Bus_Grant              : in  STD_LOGIC;
00089             reply_length                : in  STD_LOGIC_VECTOR(7 downto 0);
00090             num_samples                 : in  STD_LOGIC_VECTOR(7 downto 0);
00091             count                         : in  STD_LOGIC_VECTOR(7 downto 0);
00092             async_flags                 : out STD_LOGIC_VECTOR(17 downto 0) --flags to enable
    functions
00093             );
00094 end component;
00095
00096 begin
00097
00098 MSG_Complete <= async_flags(DONE_FLAG);
00099
00100 ---------------------------------------------------------------------------
00101 ---------------------- MSG Payload ----------------------------------------
00102 ---------------------------------------------------------------------------
00103 -- num_samples
00104 process(clk, reset)
00105 begin
00106    if reset = '0' then
00107         num_samples <= (others => '0');
00108    elsif rising_edge(clk) then
00109         if async_flags(NUM_SAMPLES_RD_FLAG) = '1' then
00110             num_samples <= RX_FIFO_DOUT;
00111         end if;
00112     end if;
00113 end process;
00114
00115 -- amplitude_reg
00116 process(clk, reset)
00117 begin
00118    if reset = '0' then
00119         amplitude_reg <= (others => '0');
00120    elsif rising_edge(clk) then
00121         if async_flags(AMPLITUDE_H_RD_FLAG) = '1' then
00122             amplitude_reg(15 downto 8) <= RX_FIFO_DOUT;
00123         elsif async_flags(AMPLITUDE_L_RD_FLAG) = '1' then
00124             amplitude_reg(7 downto 0) <= RX_FIFO_DOUT;
00125         end if;
00126     end if;
```

```
00127 end process;
00128
00129 -- time_reg
00130 process(clk, reset)
00131 begin
00132    if reset = '0' then
00133         time_reg <= (others => '0');
00134    elsif rising_edge(clk) then
00135         if async_flags(TIME_H_RD_FLAG) = '1' then
00136             time_reg(15 downto 8) <= RX_FIFO_DOUT;
00137         elsif async_flags(TIME_L_RD_FLAG) = '1' then
00138             time_reg(7 downto 0) <= RX_FIFO_DOUT;
00139         end if;
00140     end if;
00141 end process;
00142
00143 ------------------------------------------------------------------------------
00144 ---------------------- MSG_Reply Information ---------------------------------
00145 ------------------------------------------------------------------------------
00146
00147 -- TX_FIFO_DIN
00148 process(clk, reset)
00149 begin
00150    if reset = '0' then
00151         TX_FIFO_DIN <= (others => '0');
00152         reply_length <= x"00";
00153    elsif rising_edge(clk) then
00154         if async_flags(SET_REPLY_BYTE_FLAG) = '1' then -- MSG_ID x"01" - CONFIG_CHAN
00155             reply_length <= x"07";
00156             case count is
00157                 when x"00" => TX_FIFO_DIN <= x"5A";                              -- Start Byte
00158                 when x"01" => TX_FIFO_DIN <= x"85";                                  -- Reply
      MSG_ID
00159                 when x"02" => TX_FIFO_DIN <= x"00";                                  -- Reply
      Length_H
00160                 when x"03" => TX_FIFO_DIN <= x"07";                                  -- Reply
      Length_L
00161                 when x"04" => TX_FIFO_DIN <= status;                                  --
      status
00162                 when x"05" => TX_FIFO_DIN <= status;
00163                 when x"06" => TX_FIFO_DIN <= x"FF";
00164                 when others => TX_FIFO_DIN <= x"25";
00165             end case;
00166         elsif async_flags(NUM_SAMPLES_RD_FLAG) = '1' then
00167             TX_FIFO_DIN <= RX_FIFO_DOUT;
00168         elsif async_flags(AMPLITUDE_H_RD_FLAG) = '1' then
00169             TX_FIFO_DIN <= RX_FIFO_DOUT;
00170         elsif async_flags(AMPLITUDE_L_RD_FLAG) = '1' then
00171             TX_FIFO_DIN <= RX_FIFO_DOUT;
00172         elsif async_flags(TIME_H_RD_FLAG) = '1' then
00173             TX_FIFO_DIN <= RX_FIFO_DOUT;
00174         elsif async_flags(TIME_L_RD_FLAG) = '1' then
00175             TX_FIFO_DIN <= RX_FIFO_DOUT;
00176         end if;
00177     end if;
00178 end process;
00179
00180 ------------------------------------------------------------------------------
00181 ---------------------- RAM ---------------------------------------------------
00182 ------------------------------------------------------------------------------
00183 RAM_WE <= '0';  -- always in set to write
00184 RAM_ADDR <= RAM_ADDR_reg;
00185
00186 -- RAM_Start_Op
00187 process(clk, reset)
00188 begin
00189    if reset = '0' then
00190         RAM_Start_Op <= '0';
00191    elsif rising_edge(clk) then
00192         if async_flags(START_MEM_OP_FLAG) = '1' then
00193             RAM_Start_Op <= '1';
00194         else
00195             RAM_Start_Op <= '0';
00196         end if;
00197     end if;
00198 end process;
00199
00200 -- RAM_DIN
00201 process(clk, reset)
```

```
00202 begin
00203    if reset = '0' then
00204         RAM_DIN <= (others => '0');
00205    elsif rising_edge(clk) then
00206         if async_flags(NUM_SAMPLES_WR_FLAG) = '1' then
00207             RAM_DIN <= x"00" & num_samples;
00208         elsif async_flags(AMPLITUDE_WR_FLAG) = '1' then
00209             RAM_DIN <= amplitude_reg;
00210         elsif async_flags(TIME_WR_FLAG) = '1' then
00211             RAM_DIN <= time_reg;
00212         end if;
00213     end if;
00214 end process;
00215
00216 -- RAM_ADDR_reg
00217 process(clk, reset)
00218 begin
00219    if reset = '0' then
00220         RAM_ADDR_reg <= (others => '0');
00221    elsif rising_edge(clk) then
00222         if async_flags(NUM_SAMPLES_WR_FLAG) = '1' then
00223             RAM_ADDR_reg(22 downto 16) <= (others => '0');
00224             case MSG_Channel is
00225                 when x"01" =>           RAM_ADDR_reg(15 downto 0) <= x"0000";
00226                 when x"02" =>           RAM_ADDR_reg(15 downto 0) <= x"1000";
00227                 when x"03" =>           RAM_ADDR_reg(15 downto 0) <= x"2000";
00228                 when x"04" =>           RAM_ADDR_reg(15 downto 0) <= x"3000";
00229                 when x"05" =>           RAM_ADDR_reg(15 downto 0) <= x"4000";
00230                 when x"06" =>           RAM_ADDR_reg(15 downto 0) <= x"5000";
00231                 when x"07" =>           RAM_ADDR_reg(15 downto 0) <= x"6000";
00232                 when x"08" =>           RAM_ADDR_reg(15 downto 0) <= x"7000";
00233                 when others =>          RAM_ADDR_reg(15 downto 0) <= x"8000";
00234             end case;
00235         elsif async_flags(AMPLITUDE_WR_FLAG) = '1' then
00236             RAM_ADDR_reg <= RAM_ADDR_reg + 1;
00237         elsif async_flags(TIME_WR_FLAG) = '1' then
00238             RAM_ADDR_reg <= RAM_ADDR_reg + 1;
00239         end if;
00240     end if;
00241 end process;
00242
00243
00244 ------------------------------------------------------------------------------
00245 ---------------------- RX_FIFO -----------------------------------------------
00246 ------------------------------------------------------------------------------
00247 -- RX_FIFO_RD_EN
00248 process(clk, reset)
00249 begin
00250    if reset = '0' then
00251         RX_FIFO_RD_EN <= '0';
00252    elsif rising_edge(clk) then
00253         if async_flags(RX_RD_EN_FLAG) = '1' then
00254             RX_FIFO_RD_EN <= '1';
00255         else
00256             RX_FIFO_RD_EN <= '0';
00257         end if;
00258     end if;
00259 end process;
00260
00261
00262 ------------------------------------------------------------------------------
00263 ---------------------- TX_FIFO -----------------------------------------------
00264 ------------------------------------------------------------------------------
00265 -- TX_FIFO_WR_EN
00266 process(clk, reset)
00267 begin
00268    if reset = '0' then
00269         TX_FIFO_WR_EN <= '0';
00270    elsif rising_edge(clk) then
00271         if async_flags(TX_WR_EN_FLAG) = '1' then
00272             TX_FIFO_WR_EN <= '1';
00273         else
00274             TX_FIFO_WR_EN <= '0';
00275         end if;
00276     end if;
00277 end process;
00278
00279 ------------------------------------------------------------------------------
00280 ---------------------- RAM_Arbiter -------------------------------------------
```

```
00281 ----------------------------------------------------------------------------
00282
00283 -- RAM_Bus_Request
00284 process(clk, reset)
00285 begin
00286    if reset = '0' then
00287         RAM_Bus_Request <= '0';
00288    elsif rising_edge(clk) then
00289         if async_flags(RAM_BUS_REQUEST_FLAG) = '1' then
00290             RAM_Bus_Request <= '1';
00291         else
00292             RAM_Bus_Request <= '0';
00293         end if;
00294     end if;
00295 end process;
00296
00297 -- RAM_Bus_Busy
00298 process(clk, reset)
00299 begin
00300    if reset = '0' then
00301         RAM_Bus_Busy <= '0';
00302    elsif rising_edge(clk) then
00303         if async_flags(RAM_BUS_BUSY_FLAG) = '1' then
00304             RAM_Bus_Busy <= '1';
00305         else
00306             RAM_Bus_Busy <= '0';
00307         end if;
00308     end if;
00309 end process;
00310
00311 ----------------------------------------------------------------------------
00312 ---------------------- MSG_SET_WAVEFORM_states ------------------------------
00313 ----------------------------------------------------------------------------
00314 states : MSG_SET_WAVEFORM_states
00315 port map(
00316     clk                             => clk,
00317     rst_n                          => reset,
00318     MSG_Start                      => MSG_Start,
00319     FIFO_EMPTY                        => RX_FIFO_EMPTY,
00320     RAM_Op_Done                      => RAM_Op_Done,
00321     RAM_Bus_Grant                  => RAM_Bus_Grant,
00322     reply_length                    => reply_length,
00323     num_samples                      => num_samples,
00324     count                              => count,
00325     async_flags                      => async_flags
00326     );
00327
00328 ----------------------------------------------------------------------------
00329 ----------------------- Counter --------------------------------------------
00330 ----------------------------------------------------------------------------
00331
00332 -- count
00333 process(clk, reset)
00334 begin
00335    if reset = '0' then
00336         count <= (others => '0');
00337    elsif rising_edge(clk) then
00338         if async_flags(INC_COUNT_FLAG) = '1' then
00339             count <= count + 1;
00340         elsif async_flags(IDLE_FLAG) = '1' then
00341             count <= x"00";
00342         elsif async_flags(CLEAR_COUNT_FLAG) = '1' then
00343             count <= x"00";
00344         end if;
00345     end if;
00346 end process;
00347
00348
00349 end Behavioral;
00350
```

MSG_SET_WAVEFORM_main_states.vhd

```
00001 ----------------------------------------------------------------------------
00002 -- Company:
```

```
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 --
00019 -- Additional Comments:
00020 --
00021 ----------------------------------------------------------------------------------
00022 library IEEE;
00023 use IEEE.STD_LOGIC_1164.ALL;
00024 use IEEE.numeric_std.all;
00025 --use IEEE.STD_LOGIC_ARITH.ALL;
00026 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00027
00028 use work.MSG_SET_WAVEFORM_pkg.all;
00029
00030 ---- Uncomment the following library declaration if instantiating
00031 ---- any Xilinx primitives in this code.
00032 library UNISIM;
00033 use UNISIM.VComponents.all;
00034
00035 entity MSG_SET_WAVEFORM_states is
00036 Port (  clk                         : in  STD_LOGIC;
00037             rst_n                     : in  STD_LOGIC;
00038             MSG_Start                 : in  STD_LOGIC;
00039             FIFO_EMPTY                  : in  STD_LOGIC;
00040             RAM_Op_Done                : in  STD_LOGIC;
00041             RAM_Bus_Grant              : in  STD_LOGIC;
00042             reply_length               : in  STD_LOGIC_VECTOR(7 downto 0);
00043             num_samples                : in  STD_LOGIC_VECTOR(7 downto 0);
00044             count                        : in  STD_LOGIC_VECTOR(7 downto 0);
00045             async_flags                : out STD_LOGIC_VECTOR(17 downto 0) --flags to enable
      functions
00046             );
00047 end MSG_SET_WAVEFORM_states;
00048
00049 architecture Behavioral of MSG_SET_WAVEFORM_states is
00050
00051         --Control signals
00052
00053 signal curr_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM current state
00054 signal next_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM next state
00055
00056 begin
00057 ----------------------------------------------------------------------------------
00058         -- synchronous part of state machine here
00059 data_in_latch: process(clk, rst_n)
00060 begin
00061    if rst_n = '0' then
00062         curr_state <= (others => '0');
00063    elsif rising_edge(clk) then
00064         curr_state <= next_state;
00065    end if;
00066 end process;
00067
00068         -- async part of state machine to set function flags
00069 MSG_SET_WAVEFORM_state: process(rst_n, curr_state,
      reply_length, count)
00070 begin
00071    if rst_n = '0' then
00072         async_flags <= (others => '0');
00073    else
00074         async_flags <= (others => '0');
00075         case curr_state is
00076
00077             when IDLE =>
00078                 async_flags(IDLE_FLAG) <= '1';           -- init
00079
```

```
00080 -- Message Request Payload and Checksum
00081
00082             when NUM_SAMPLES_2 =>       -- read from RX_FIFO
00083                 async_flags(NUM_SAMPLES_RD_FLAG) <= '1';
00084
00085             when NUM_SAMPLES_BR_WAIT =>     -- wait for mem bus grant
00086                 async_flags(RAM_BUS_REQUEST_FLAG) <= '1';
00087
00088             when NUM_SAMPLES_3 =>       -- inc fifo and start mem op (set all inputs and trigger start op)
00089                 async_flags(RAM_BUS_BUSY_FLAG) <= '1';
00090                 async_flags(RX_RD_EN_FLAG) <= '1';
00091                 async_flags(START_MEM_OP_FLAG) <= '1';
00092                 async_flags(NUM_SAMPLES_WR_FLAG) <= '1';
00093                 --async_flags(TX_WR_EN_FLAG) <= '1';  --adding rs232 loopback for debug
00094
00095             when NUM_SAMPLES_4 =>       -- Hold RAM_Bus_Busy
00096                 async_flags(RAM_BUS_BUSY_FLAG) <= '1';
00097
00098 -- Loop Amplitude:Time read from RX_FIFO and Write to RAM
00099             when LOOP1 =>
00100
00101             -- Read Amplitude and Write to RAM (amplitude is 16 bits, high and low bytes are read from RX
      FIFO and written to RAM)
00102
00103                 when AMPLITUDE_H_2 =>               -- read Amplitude high byte
00104                     async_flags(AMPLITUDE_H_RD_FLAG) <= '1';
00105
00106                 when AMPLITUDE_H_3 =>               -- inc fifo
00107                     async_flags(RX_RD_EN_FLAG) <= '1';
00108                     --async_flags(TX_WR_EN_FLAG) <= '1';  --adding rs232 loopback for debug
00109
00110                 when AMPLITUDE_L_2 =>               -- read Amplitude low byte
00111                     async_flags(AMPLITUDE_L_RD_FLAG) <= '1';
00112
00113                 when AMPLITUDE_BR_WAIT =>           -- wait for RAM_Bus_Grant
00114                     async_flags(RAM_BUS_REQUEST_FLAG) <= '1';
00115
00116                 when AMPLITUDE_L_3 =>               -- inc fifo and start mem op (set all inputs and
      trigger start op)
00117                     async_flags(RAM_BUS_BUSY_FLAG) <= '1';
00118                     async_flags(RX_RD_EN_FLAG) <= '1';
00119                     async_flags(START_MEM_OP_FLAG) <= '1';
00120                     async_flags(AMPLITUDE_WR_FLAG) <= '1';
00121                     --async_flags(TX_WR_EN_FLAG) <= '1';  --adding rs232 loopback for debug
00122
00123                 when AMPLITUDE_L_4 =>               -- Hold RAM_Bus_Busy
00124                     async_flags(RAM_BUS_BUSY_FLAG) <= '1';
00125
00126             -- Read Time and Write to RAM (Time is 16 bits, high and low bytes are read from RX FIFO and
      written to RAM)
00127
00128                 when TIME_H_2 =>                        -- read Time high byte
00129                     async_flags(TIME_H_RD_FLAG) <= '1';
00130
00131                 when TIME_H_3 =>                        -- inc fifo
00132                     async_flags(RX_RD_EN_FLAG) <= '1';
00133                     --async_flags(TX_WR_EN_FLAG) <= '1';  --adding rs232 loopback for debug
00134
00135                 when TIME_L_2 =>                        -- read Time low byte
00136                     async_flags(TIME_L_RD_FLAG) <= '1';
00137
00138                 when TIME_BR_WAIT =>                    -- wait for RAM_Bus_Grant
00139                     async_flags(RAM_BUS_REQUEST_FLAG) <= '1';
00140
00141                 when TIME_L_3 =>                        -- inc fifo and start mem op (set all inputs
      and trigger start op)
00142                     async_flags(RAM_BUS_BUSY_FLAG) <= '1';
00143                     async_flags(RX_RD_EN_FLAG) <= '1';
00144                     async_flags(START_MEM_OP_FLAG) <= '1';
00145                     async_flags(TIME_WR_FLAG) <= '1';
00146                     --async_flags(TX_WR_EN_FLAG) <= '1';  --adding rs232 loopback for debug
00147
00148                 when TIME_L_4 =>                        -- Hold RAM_Bus_Busy
00149                     async_flags(RAM_BUS_BUSY_FLAG) <= '1';
00150
00151             when LOOP1_END =>
00152                 async_flags(INC_COUNT_FLAG) <= '1';
00153
00154             when VALIDATE_MSG =>
```

```
00155                 async_flags(CLEAR_COUNT_FLAG) <= '1';
00156
00157
00158 -- Message Reply
00159             when SET_REPLY_BYTE =>
00160                 async_flags(SET_REPLY_BYTE_FLAG) <= '1';
00161
00162
00163             when SEND_REPLY_BYTE =>
00164                 if(count = reply_length) then
00165                 else
00166                     async_flags(TX_WR_EN_FLAG) <= '1';
00167                     async_flags(INC_COUNT_FLAG) <= '1';
00168                 end if;
00169
00170             when FINISH =>
00171                 async_flags(DONE_FLAG) <= '1';           -- done flag
00172             when others =>
00173                 async_flags <= (others => '0');
00174         end case;
00175     end if;
00176 end process;
00177
00178
00179
    ------------------------------------------------------------------------------------------------------------------------
00180
    ------------------------------------------------------------------------------------------------------------------------
00181 -- MSG_SET_WAVEFORM state machine
00182
    ------------------------------------------------------------------------------------------------------------------------
00183
    ------------------------------------------------------------------------------------------------------------------------
00184 MSG_SET_WAVEFORM_asynch_state: process(rst_n,
    curr_state, count, MSG_Start, FIFO_EMPTY,
    reply_length, num_samples, RAM_Op_Done,
    RAM_Bus_Grant)
00185 begin
00186    if rst_n = '0' then
00187         next_state <= IDLE;
00188    else
00189         case curr_state is
00190         when IDLE =>
00191                 if MSG_Start = '1' then
00192                     next_state <= NUM_SAMPLES_1;
00193                 else
00194                     next_state <= IDLE;
00195                 end if;
00196
00197 -- Read Num Samples and Write to RAM
00198             when NUM_SAMPLES_1 =>       -- wait for byte in RX_FIFO
00199                 if FIFO_EMPTY = '0' then
00200                     next_state <= NUM_SAMPLES_2;
00201                 else
00202                     next_state <= NUM_SAMPLES_1;
00203                 end if;
00204
00205             when NUM_SAMPLES_2 =>           -- read
00206                 next_state <= NUM_SAMPLES_BR_WAIT;
00207
00208             when NUM_SAMPLES_BR_WAIT =>     -- wait for RAM_Bus_Grant
00209                 if RAM_Bus_Grant = '1' then
00210                     next_state <= NUM_SAMPLES_3;
00211                 else
00212                     next_state <= NUM_SAMPLES_BR_WAIT;
00213                 end if;
00214
00215             when NUM_SAMPLES_3 =>           -- inc fifo and request mem bus (set all inputs and trigger
    start op)
00216                 next_state <= NUM_SAMPLES_4;
00217
00218             when NUM_SAMPLES_4 =>           -- wait for mem op to complete
00219                 if RAM_Op_Done = '1' then
00220                     next_state <= LOOP1;
00221                 else
00222                     next_state <= NUM_SAMPLES_4;
00223                 end if;
00224
00225 -- Loop Amplitude:Time read from RX_FIFO and Write to RAM
```

```
00226             when LOOP1 =>
00227                 if count = num_samples then
00228                     next_state <= VALIDATE_MSG;
00229                 else
00230                     next_state <= AMPLITUDE_H_1;
00231                 end if;
00232
00233             -- Read Amplitude and Write to RAM (amplitude is 16 bits, high and low bytes are read from RX
      FIFO and written to RAM)
00234
00235                     when AMPLITUDE_H_1 =>                -- wait for amplitude high byte in RX_FIFO
00236                         if FIFO_EMPTY = '0' then
00237                             next_state <= AMPLITUDE_H_2;
00238                         else
00239                             next_state <= AMPLITUDE_H_1;
00240                         end if;
00241
00242                     when AMPLITUDE_H_2 =>                -- read Amplitude high byte
00243                         next_state <= AMPLITUDE_H_3;
00244
00245                     when AMPLITUDE_H_3 =>                -- inc fifo
00246                         next_state <= AMPLITUDE_H_4;
00247
00248                     when AMPLITUDE_H_4 =>                -- delay state to allow FIFO_EMPTY to be set after
      inc_fifo
00249                         next_state <= AMPLITUDE_H_5;
00250                     when AMPLITUDE_H_5 =>                -- delay state to allow FIFO_EMPTY to be set after
      inc_fifo
00251                         next_state <= AMPLITUDE_L_1;
00252
00253                     when AMPLITUDE_L_1 =>                -- wait for amplitude low byte in RX_FIFO
00254                         if FIFO_EMPTY = '0' then
00255                             next_state <= AMPLITUDE_L_2;
00256                         else
00257                             next_state <= AMPLITUDE_L_1;
00258                         end if;
00259
00260                     when AMPLITUDE_L_2 =>                -- read Amplitude low byte
00261                         next_state <= AMPLITUDE_BR_WAIT;
00262
00263                     when AMPLITUDE_BR_WAIT =>            -- wait for RAM_Bus_Grant
00264                         if RAM_Bus_Grant = '1' then
00265                             next_state <= AMPLITUDE_L_3;
00266                         else
00267                             next_state <= AMPLITUDE_BR_WAIT;
00268                         end if;
00269
00270                     when AMPLITUDE_L_3 =>                -- inc fifo and start mem op (set all inputs and
      trigger start op)
00271                         next_state <= AMPLITUDE_L_4;
00272
00273                     when AMPLITUDE_L_4 =>                -- wait for mem op to complete
00274                         if RAM_Op_Done = '1' then
00275                             next_state <= TIME_H_1;
00276                         else
00277                             next_state <= AMPLITUDE_L_4;
00278                         end if;
00279
00280
00281             -- Read Time and Write to RAM (Time is 16 bits, high and low bytes are read from RX FIFO and
      written to RAM)
00282
00283                     when TIME_H_1 =>                         -- wait for Time high byte in RX_FIFO
00284                         if FIFO_EMPTY = '0' then
00285                             next_state <= TIME_H_2;
00286                         else
00287                             next_state <= TIME_H_1;
00288                         end if;
00289
00290                     when TIME_H_2 =>                         -- read Time high byte
00291                         next_state <= TIME_H_3;
00292
00293                     when TIME_H_3 =>                         -- inc fifo
00294                         next_state <= TIME_H_4;
00295
00296                     when TIME_H_4 =>                         -- delay state to allow FIFO_EMPTY to be set
      after inc_fifo
00297                         next_state <= TIME_H_5;
00298
```

```
00299                     when TIME_H_5 =>                          -- delay state to allow FIFO_EMPTY to be set
      after inc_fifo
00300                         next_state <= TIME_L_1;
00301
00302                     when TIME_L_1 =>                          -- wait for Time low byte in RX_FIFO
00303                         if FIFO_EMPTY = '0' then
00304                             next_state <= TIME_L_2;
00305                         else
00306                             next_state <= TIME_L_1;
00307                         end if;
00308
00309                     when TIME_L_2 =>                          -- read Time low byte
00310                         next_state <= TIME_BR_WAIT;
00311
00312                     when TIME_BR_WAIT =>                      -- wait for RAM_Bus_Grant
00313                         if RAM_Bus_Grant = '1' then
00314                             next_state <= TIME_L_3;
00315                         else
00316                             next_state <= TIME_BR_WAIT;
00317                         end if;
00318
00319                     when TIME_L_3 =>                          -- inc fifo and start mem op (set all inputs
      and trigger start op)
00320                         next_state <= TIME_L_4;
00321
00322                     when TIME_L_4 =>                          -- wait for mem op to complete
00323                         if RAM_Op_Done = '1' then
00324                             next_state <= LOOP1_END;
00325                         else
00326                             next_state <= TIME_L_4;
00327                         end if;
00328
00329             when LOOP1_END =>
00330                 next_state <= LOOP1;
00331
00332
00333             when VALIDATE_MSG =>
00334                 next_state <= SET_REPLY_BYTE;
00335
00336
      ------------------------------------------------------------------------------------------------------------------
00337 -- Message Reply
00338
      ------------------------------------------------------------------------------------------------------------------
00339
00340             when SET_REPLY_BYTE =>
00341                 next_state <= SEND_REPLY_BYTE;
00342
00343             when SEND_REPLY_BYTE =>
00344                 if count = reply_length then
00345                     next_state <= FINISH;
00346                 else
00347                     next_state <= SET_REPLY_BYTE;
00348                 end if;
00349
00350             when FINISH =>
00351                     next_state <= IDLE;
00352
00353             when OTHERS =>
00354                 next_state <= IDLE;
00355         end case;
00356    end if;
00357 end process;
00358
00359
00360 end Behavioral;
00361
00362
```

MSG_SET_WAVEFORM_pkg.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:     WMU - Thesis
00003 -- Engineer:    KDB
00004 --
```

```
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 --
00016 -- Dependencies:
00017 --
00018 -- Revision:
00019 -- Revision 0.01 - File Created
00020 -- Additional Comments:
00021 --
00022 ----------------------------------------------------------------------------------
00023 library IEEE;
00024 use IEEE.STD_LOGIC_1164.ALL;
00025 use IEEE.numeric_std.all;
00026 --use IEEE.STD_LOGIC_ARITH.ALL;
00027 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00028
00029 package MSG_SET_WAVEFORM_pkg is
00030
00031
00032 constant IDLE                                   : STD_LOGIC_VECTOR(7 downto 0) := x"00";
00033
00034 constant NUM_SAMPLES_1                        : STD_LOGIC_VECTOR(7 downto 0) := x"01";
00035 constant NUM_SAMPLES_2                        : STD_LOGIC_VECTOR(7 downto 0) := x"02";
00036 constant NUM_SAMPLES_BR_WAIT                 : STD_LOGIC_VECTOR(7 downto 0) := x"03";
00037 constant NUM_SAMPLES_3                        : STD_LOGIC_VECTOR(7 downto 0) := x"04";
00038 constant NUM_SAMPLES_4                        : STD_LOGIC_VECTOR(7 downto 0) := x"05";
00039
00040 constant LOOP1                                   : STD_LOGIC_VECTOR(7 downto 0) := x"06";
00041
00042 constant AMPLITUDE_H_1                        : STD_LOGIC_VECTOR(7 downto 0) := x"10";
00043 constant AMPLITUDE_H_2                        : STD_LOGIC_VECTOR(7 downto 0) := x"11";
00044 constant AMPLITUDE_H_3                        : STD_LOGIC_VECTOR(7 downto 0) := x"12";
00045 constant AMPLITUDE_H_4                        : STD_LOGIC_VECTOR(7 downto 0) := x"13";
00046 constant AMPLITUDE_H_5                        : STD_LOGIC_VECTOR(7 downto 0) := x"14";
00047
00048 constant AMPLITUDE_L_1                        : STD_LOGIC_VECTOR(7 downto 0) := x"20";
00049 constant AMPLITUDE_L_2                        : STD_LOGIC_VECTOR(7 downto 0) := x"21";
00050 constant AMPLITUDE_BR_WAIT                    : STD_LOGIC_VECTOR(7 downto 0) := x"22";
00051 constant AMPLITUDE_L_3                        : STD_LOGIC_VECTOR(7 downto 0) := x"23";
00052 constant AMPLITUDE_L_4                        : STD_LOGIC_VECTOR(7 downto 0) := x"24";
00053
00054 constant TIME_H_1                                : STD_LOGIC_VECTOR(7 downto 0) := x"30";
00055 constant TIME_H_2                                : STD_LOGIC_VECTOR(7 downto 0) := x"31";
00056 constant TIME_H_3                                : STD_LOGIC_VECTOR(7 downto 0) := x"32";
00057 constant TIME_H_4                                : STD_LOGIC_VECTOR(7 downto 0) := x"33";
00058 constant TIME_H_5                                : STD_LOGIC_VECTOR(7 downto 0) := x"34";
00059
00060 constant TIME_L_1                                : STD_LOGIC_VECTOR(7 downto 0) := x"40";
00061 constant TIME_L_2                                : STD_LOGIC_VECTOR(7 downto 0) := x"41";
00062 constant TIME_BR_WAIT                          : STD_LOGIC_VECTOR(7 downto 0) := x"42";
00063 constant TIME_L_3                                : STD_LOGIC_VECTOR(7 downto 0) := x"43";
00064 constant TIME_L_4                                : STD_LOGIC_VECTOR(7 downto 0) := x"44";
00065
00066 constant LOOP1_END                             : STD_LOGIC_VECTOR(7 downto 0) := x"50";
00067
00068 constant VALIDATE_MSG                          : STD_LOGIC_VECTOR(7 downto 0) := x"60";
00069
00070 constant SET_REPLY_BYTE                       : STD_LOGIC_VECTOR(7 downto 0) := x"70";
00071 constant SEND_REPLY_BYTE                        : STD_LOGIC_VECTOR(7 downto 0) := x"71";
00072
00073
00074 constant FINISH                                   : STD_LOGIC_VECTOR(7 downto 0) := x"FF";
00075
00076
00077
00078 constant IDLE_FLAG                             : integer := 0;
00079
00080 constant INC_COUNT_FLAG                          : integer := 1;
00081 constant RX_RD_EN_FLAG                        : integer := 2;
00082 constant TX_WR_EN_FLAG                        : integer := 3;
00083 constant START_MEM_OP_FLAG                    : integer := 4;
```

```
00084
00085 constant NUM_SAMPLES_RD_FLAG                : integer := 5;
00086 constant NUM_SAMPLES_WR_FLAG                : integer := 6;
00087
00088 constant AMPLITUDE_H_RD_FLAG                : integer := 7;
00089 constant AMPLITUDE_L_RD_FLAG                : integer := 8;
00090 constant AMPLITUDE_WR_FLAG                    : integer := 9;
00091
00092 constant TIME_H_RD_FLAG                          : integer := 10;
00093 constant TIME_L_RD_FLAG                          : integer := 11;
00094 constant TIME_WR_FLAG                          : integer := 12;
00095
00096 constant SET_REPLY_BYTE_FLAG                : integer := 13;
00097
00098 constant CLEAR_COUNT_FLAG                      : integer := 14;
00099
00100 constant RAM_BUS_REQUEST_FLAG                : integer := 15;
00101 constant RAM_BUS_BUSY_FLAG                    : integer := 16;
00102
00103
00104 constant DONE_FLAG                              : integer := 17;
00105
00106
00107
00108 end MSG_SET_WAVEFORM_pkg;
00109
00110 package body MSG_SET_WAVEFORM_pkg is
00111
00112 end MSG_SET_WAVEFORM_pkg;
00113
00114
```

RAM_Module.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
00005 -- Create Date:    19:44:35 01/27/2012
00006 -- Design Name:
00007 -- Module Name:    ADC_Module - Behavioral
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- Dependencies:
00014 --
00015 -- Revision:
00016 -- Revision 0.01 - File Created
00017 -- Additional Comments:
00018 --
00019 ----------------------------------------------------------------------------------
00020 library IEEE;
00021 use IEEE.STD_LOGIC_1164.ALL;
00022 use IEEE.STD_LOGIC_ARITH.ALL;
00023 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00024
00025 ---- Uncomment the following library declaration if instantiating
00026 ---- any Xilinx primitives in this code.
00027 --library UNISIM;
00028 --use UNISIM.VComponents.all;
00029
00030 use work.RAM_Module_pkg.all;
00031
00032 entity RAM_Module is
00033     Port ( clk                  : in  STD_LOGIC;
00034            reset                  : in  STD_LOGIC;
00035
00036               -- MT45W8MW16BGX Signals
00037               MT_ADDR               : out STD_LOGIC_VECTOR(22 downto 0);
00038               MT_DATA               : inout STD_LOGIC_VECTOR(15 downto 0);
00039               MT_OE                   : out STD_LOGIC; -- active low
00040               MT_WE                   : out STD_LOGIC; -- active low
00041               MT_ADV                 : out STD_LOGIC; -- active low
```

```
00042               MT_CLK                  : out STD_LOGIC; -- during asynch operation, hold the clock low
00043               MT_UB                    : out STD_LOGIC; -- active low
00044               MT_LB                    : out STD_LOGIC; -- active low
00045               MT_CE                    : out STD_LOGIC; -- active low
00046               MT_CRE                  : out STD_LOGIC;
00047               --MT_WAIT              : in  STD_LOGIC; -- ignored
00048
00049               -- RAM_Module Control
00050               RAM_Start_Op           : in  STD_LOGIC;
00051               RAM_Op_Done             : out STD_LOGIC;
00052               RAM_WE                   : in  STD_LOGIC;
00053               RAM_ADDR               : in  STD_LOGIC_VECTOR(22 downto 0);
00054               RAM_DOUT               : out STD_LOGIC_VECTOR(15 downto 0);
00055               RAM_DIN                 : in    STD_LOGIC_VECTOR(15 downto 0);
00056
00057               -- RAM_Arbiter
00058               RAM_Bus_Request         : in  STD_LOGIC_VECTOR(7 downto 0);
00059               RAM_Bus_Busy           : in  STD_LOGIC;
00060               RAM_Bus_Grant         : out  STD_LOGIC_VECTOR(7 downto 0)
00061               );
00062 end RAM_Module;
00063
00064 architecture Behavioral of RAM_Module is
00065
00066 signal async_flags                   : STD_LOGIC_VECTOR(4 downto 0);
00067 signal count                           : STD_LOGIC_VECTOR(7 downto 0);
00068
00069 signal init_counter                    : STD_LOGIC_VECTOR(15 downto 0) := x"0000";
00070
00071 signal RAM_ADDR_reg                 : STD_LOGIC_VECTOR(22 downto 0);
00072 signal RAM_DATA_reg                 : STD_LOGIC_VECTOR(15 downto 0);
00073 signal RAM_WE_sig                    : STD_LOGIC;
00074
00075 component RAM_Module_states
00076 Port (  clk                          : in  STD_LOGIC;
00077             rst_n                      : in  STD_LOGIC;
00078             RAM_Start_Op               : in  STD_LOGIC;            --start S.M. into motion
00079             count                        : in  STD_LOGIC_VECTOR(7 downto 0);
00080             init_counter               : in  STD_LOGIC_VECTOR(15 downto 0);
00081             async_flags                : out STD_LOGIC_VECTOR(4 downto 0)  --flags to enable
      functions
00082             );
00083 end component;
00084
00085 component Arbiter
00086     Port ( clk                         : in  STD_LOGIC;
00087            reset                       : in  STD_LOGIC;
00088               Bus_Request              : in  STD_LOGIC_VECTOR(7 downto 0);
00089               Bus_Busy                 : in  STD_LOGIC;
00090
00091               Bus_Grant                   : out  STD_LOGIC_VECTOR(7 downto 0)
00092               );
00093 end component;
00094
00095 begin
00096
00097 ---------------------------------------------------------------------------
00098 ---------------------- RAM_Module_Control ---------------------------------
00099 ---------------------------------------------------------------------------
00100 RAM_Op_Done      <= async_flags(DONE_FLAG);
00101
00102 -- RAM_DOUT
00103 process(clk, reset)
00104 begin
00105    if reset = '0' then
00106         RAM_DOUT <= (others => '0');
00107    elsif rising_edge(clk) then
00108         if async_flags(PERFORM_OP_FLAG) = '1' and count = 5 and
      RAM_WE_sig = '1' then -- read op
00109              RAM_DOUT <= RAM_DATA_reg;
00110         end if;
00111     end if;
00112 end process;
00113
00114 -- RAM_DATA_reg
00115 process(clk, reset)
00116 begin
00117    if reset = '0' then
00118         RAM_DATA_reg <= (others => '0');
```

```
00119    elsif rising_edge(clk) then
00120         if async_flags(PERFORM_OP_FLAG) = '1' and count = 0 and
      RAM_WE_sig = '0' then -- write op
00121              RAM_DATA_reg <= RAM_DIN;
00122         elsif async_flags(PERFORM_OP_FLAG) = '1' and count = 4 and
      RAM_WE_sig = '1' then -- read op
00123              RAM_DATA_reg <= MT_DATA;
00124         end if;
00125     end if;
00126 end process;
00127
00128 -- RAM_ADDR_reg
00129 process(clk, reset)
00130 begin
00131    if reset = '0' then
00132         RAM_ADDR_reg <= (others => '0');
00133    elsif rising_edge(clk) then
00134         if async_flags(IDLE_FLAG) = '1' then
00135              RAM_ADDR_reg <= RAM_ADDR;
00136         end if;
00137     end if;
00138 end process;
00139
00140 -- RAM_WE_sig
00141 process(clk, reset)
00142 begin
00143    if reset = '0' then
00144         RAM_WE_sig <= '0';
00145    elsif rising_edge(clk) then
00146         if async_flags(IDLE_FLAG) = '1' then
00147              RAM_WE_sig <= RAM_WE;
00148         end if;
00149     end if;
00150 end process;
00151
00152 ------------------------------------------------------------------------------
00153 ---------------------- MT45W8MW16BGX Signals ---------------------------------
00154 ------------------------------------------------------------------------------
00155 MT_DATA              <= RAM_DATA_reg when RAM_WE_sig = '0' else -- write op
00156                           (others => 'Z');
00157
00158 MT_CLK <= '0';
00159
00160 MT_CRE <= '0';    -- hold low, active high, control register enable
00161
00162 MT_ADDR <= RAM_ADDR_reg;
00163
00164 -- MT_WE
00165 process(clk, reset)
00166 begin
00167    if reset = '0' then
00168         MT_WE <= '1';
00169    elsif rising_edge(clk) then
00170         if async_flags(PERFORM_OP_FLAG) = '1' then
00171              MT_WE <= RAM_WE_sig;
00172         else
00173              MT_WE <= '1';
00174         end if;
00175     end if;
00176 end process;
00177
00178 -- Control Signals
00179 process(clk, reset)
00180 begin
00181    if reset = '0' then
00182         MT_OE  <= '1';
00183         MT_ADV    <= '1';
00184         MT_UB  <= '1';
00185         MT_LB  <= '1';
00186         MT_CE  <= '1';
00187    elsif rising_edge(clk) then
00188         if async_flags(PERFORM_OP_FLAG) = '1' then
00189              MT_OE  <= '0';
00190              MT_ADV    <= '0';
00191              MT_UB  <= '0';
00192              MT_LB  <= '0';
00193              MT_CE  <= '0';
00194         else
00195              MT_OE  <= '1';
```

```
00196             MT_ADV    <= '1';
00197             MT_UB  <= '1';
00198             MT_LB  <= '1';
00199             MT_CE  <= '1';
00200         end if;
00201     end if;
00202 end process;
00203 
00204 ------------------------------------------------------------------------------
00205 ---------------------- RAM_Module_states -------------------------------------
00206 ------------------------------------------------------------------------------
00207 
00208 states : RAM_Module_states
00209 port map(
00210     clk                             => clk,
00211     rst_n                          => reset,
00212     RAM_Start_Op                    => RAM_Start_Op,
00213     count                              => count,
00214     init_counter                    => init_counter,
00215     async_flags                      => async_flags
00216     );
00217 
00218 ------------------------------------------------------------------------------
00219 ---------------------- RAM_Arbiter -------------------------------------------
00220 ------------------------------------------------------------------------------
00221 
00222 RAM_Arbiter : Arbiter
00223 port map(
00224     clk                             => clk,
00225     reset                          => reset,
00226     Bus_Request                       => RAM_Bus_Request,
00227     Bus_Busy                                 => RAM_Bus_Busy,
00228     Bus_Grant                          => RAM_Bus_Grant
00229     );
00230 
00231 ------------------------------------------------------------------------------
00232 ---------------------- Counter -----------------------------------------------
00233 ------------------------------------------------------------------------------
00234 
00235 -- count
00236 process(clk, reset)
00237 begin
00238    if reset = '0' then
00239         count <= (others => '0');
00240    elsif rising_edge(clk) then
00241         if async_flags(INC_COUNT_FLAG) = '1' then
00242             count <= count + 1;
00243         elsif async_flags(IDLE_FLAG) = '1' then
00244             count <= x"00";
00245         end if;
00246     end if;
00247 end process;
00248 
00249 
00250 -- init_counter
00251 process(clk, reset)
00252 begin
00253    if reset = '0' then
00254         init_counter <= (others => '0');
00255    elsif rising_edge(clk) then
00256         if async_flags(RAM_RESET_FLAG) = '1' then
00257             init_counter <= init_counter + 1;
00258         elsif async_flags(IDLE_FLAG) = '1' then
00259             init_counter <= (others => '0');
00260         end if;
00261     end if;
00262 end process;
00263 
00264 
00265 end Behavioral;
00266 
```

RAM_Module_main_states.vhd

```
00001 ------------------------------------------------------------------------------
```

```
00002 -- Company:
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 --
00019 -- Additional Comments:
00020 --
00021 ----------------------------------------------------------------------------------
00022 library IEEE;
00023 use IEEE.STD_LOGIC_1164.ALL;
00024 use IEEE.numeric_std.all;
00025 --use IEEE.STD_LOGIC_ARITH.ALL;
00026 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00027
00028 use work.RAM_Module_pkg.all;
00029
00030 ---- Uncomment the following library declaration if instantiating
00031 ---- any Xilinx primitives in this code.
00032 library UNISIM;
00033 use UNISIM.VComponents.all;
00034
00035 entity RAM_Module_states is
00036 Port (  clk                         : in  STD_LOGIC;
00037             rst_n                     : in  STD_LOGIC;
00038             RAM_Start_Op              : in  STD_LOGIC;            --start S.M. into motion
00039             count                        : in  STD_LOGIC_VECTOR(7 downto 0);
00040             init_counter              : in  STD_LOGIC_VECTOR(15 downto 0);
00041             async_flags               : out STD_LOGIC_VECTOR(4 downto 0)  --flags to enable
    functions
00042             );
00043 end RAM_Module_states;
00044
00045 architecture Behavioral of RAM_Module_states is
00046
00047         --Control signals
00048
00049 signal curr_state     : std_logic_vector(7 downto 0) := RAM_RESET;  -- FSM current state
00050 signal next_state     : std_logic_vector(7 downto 0) := RAM_RESET;  -- FSM next state
00051
00052 begin
00053 ----------------------------------------------------------------------------------
00054         -- synchronous part of state machine here
00055 data_in_latch: process(clk, rst_n)
00056 begin
00057    if rst_n = '0' then
00058         curr_state <= RAM_RESET;
00059    elsif rising_edge(clk) then
00060         curr_state <= next_state;
00061    end if;
00062 end process;
00063
00064         -- async part of state machine to set function flags
00065 RAM_Module_state: process(rst_n, curr_state)
00066 begin
00067    if rst_n = '0' then
00068         async_flags <= (others => '0');
00069    else
00070         async_flags <= (others => '0');
00071         case curr_state is
00072
00073             when RAM_RESET =>
00074                 async_flags(RAM_RESET_FLAG) <= '1';
00075
00076             when IDLE =>
00077                 async_flags(IDLE_FLAG) <= '1';           -- init
00078
00079             when PERFORM_OP =>
```

```
00080                 async_flags(PERFORM_OP_FLAG) <= '1';
00081                 async_flags(INC_COUNT_FLAG) <= '1';
00082
00083             when FINISH =>
00084                 async_flags(DONE_FLAG) <= '1';            -- done flag
00085                 async_flags(INC_COUNT_FLAG) <= '1';
00086             when others =>
00087                 async_flags <= (others => '0');
00088         end case;
00089     end if;
00090 end process;
00091 ------------------------------------------------
00092         -- RAM_Module state machine
00093 RAM_Module_asynch_state: process(rst_n, curr_state,
      RAM_Start_Op, count, init_counter)
00094 begin
00095    if rst_n = '0' then
00096         next_state <= RAM_RESET;
00097    else
00098         case curr_state is
00099             when RAM_RESET =>
00100                 if init_counter = x"1D4B" then
00101                     next_state <= IDLE;
00102                 else
00103                     next_state <= RAM_RESET;
00104                 end if;
00105
00106         when IDLE =>
00107                 if RAM_Start_Op = '1' then
00108                     next_state <= PERFORM_OP;
00109                 else
00110                     next_state <= IDLE;
00111                 end if;
00112
00113             when PERFORM_OP =>
00114                 if count = 5 then
00115                     next_state <= FINISH;
00116                 else
00117                     next_state <= PERFORM_OP;
00118                 end if;
00119
00120
00121
00122
00123             when FINISH =>
00124                 --if count = 8 then
00125                     next_state <= IDLE;
00126                 --else
00127                     --next_state <= FINISH;
00128                 --end if;
00129
00130             when OTHERS =>
00131                 next_state <= IDLE;
00132         end case;
00133    end if;
00134 end process;
00135
00136
00137 end Behavioral;
00138
00139
```

RAM_Module_pkg.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:     WMU - Thesis
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
```

```
00012 --
00013 --
00014 --
00015 --
00016 -- Dependencies:
00017 --
00018 -- Revision:
00019 -- Revision 0.01 - File Created
00020 -- Additional Comments:
00021 --
00022 ----------------------------------------------------------------------------------
00023 library IEEE;
00024 use IEEE.STD_LOGIC_1164.ALL;
00025 use IEEE.numeric_std.all;
00026 --use IEEE.STD_LOGIC_ARITH.ALL;
00027 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00028
00029 package RAM_Module_pkg is
00030
00031
00032 constant RAM_RESET                              : STD_LOGIC_VECTOR(7 downto 0) := x"10";
00033
00034 constant IDLE                                        : STD_LOGIC_VECTOR(7 downto 0) := x"00";
00035 constant PERFORM_OP                                : STD_LOGIC_VECTOR(7 downto 0) := x"01";
00036
00037 constant FINISH                                    : STD_LOGIC_VECTOR(7 downto 0) := x"FF";
00038
00039
00040
00041 constant IDLE_FLAG                              : integer := 0;
00042
00043 constant PERFORM_OP_FLAG                     : integer := 1;
00044 constant INC_COUNT_FLAG                          : integer := 2;
00045 constant RAM_RESET_FLAG                          : integer := 3;
00046
00047
00048
00049 constant DONE_FLAG                              : integer := 4;
00050
00051
00052
00053 end RAM_Module_pkg;
00054
00055 package body RAM_Module_pkg is
00056
00057 end RAM_Module_pkg;
00058
00059
```

RAM_Module_tb.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
00005 -- Create Date:   17:03:04 03/07/2012
00006 -- Design Name:
00007 -- Module Name:   C:/Users/Kyle/Desktop/SVN_Thesis/FPGA Code/Micron_RAM_Test/RAM_Module_tb.vhd
00008 -- Project Name:  Micron_RAM_Test
00009 -- Target Device:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- VHDL Test Bench Created by ISE for module: RAM_Module
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 -- Revision 0.01 - File Created
00019 -- Additional Comments:
00020 --
00021 -- Notes:
00022 -- This testbench has been automatically generated using types std_logic and
00023 -- std_logic_vector for the ports of the unit under test.  Xilinx recommends
00024 -- that these types always be used for the top-level I/O of a design in order
```

```
00025 -- to guarantee that the testbench will bind correctly to the post-implementation
00026 -- simulation model.
00027 --------------------------------------------------------------------------------
00028 LIBRARY ieee;
00029 USE ieee.std_logic_1164.ALL;
00030 USE ieee.std_logic_unsigned.all;
00031 USE ieee.numeric_std.ALL;
00032
00033 ENTITY RAM_Module_tb IS
00034 END RAM_Module_tb;
00035
00036 ARCHITECTURE behavior OF RAM_Module_tb IS
00037
00038     -- Component Declaration for the Unit Under Test (UUT)
00039
00040     COMPONENT RAM_Module
00041     PORT(
00042          clk : IN  std_logic;
00043          reset : IN  std_logic;
00044          MT_ADDR : OUT  std_logic_vector(22 downto 0);
00045          MT_DATA : INOUT  std_logic_vector(15 downto 0);
00046          MT_OE : OUT  std_logic;
00047          MT_WE : OUT  std_logic;
00048          MT_ADV : OUT  std_logic;
00049          MT_CLK : OUT  std_logic;
00050          MT_UB : OUT  std_logic;
00051          MT_LB : OUT  std_logic;
00052          MT_CE : OUT  std_logic;
00053          MT_CRE : OUT  std_logic;
00054          MT_WAIT : IN  std_logic;
00055          RAM_Start_Op : IN  std_logic;
00056          RAM_Op_Done : OUT  std_logic;
00057          RAM_WE : IN  std_logic;
00058          RAM_ADDR : IN  std_logic_vector(22 downto 0);
00059          RAM_DOUT : OUT  std_logic_vector(15 downto 0);
00060          RAM_DIN : IN  std_logic_vector(15 downto 0)
00061         );
00062     END COMPONENT;
00063
00064
00065    --Inputs
00066    signal clk : std_logic := '0';
00067    signal reset : std_logic := '0';
00068    signal MT_WAIT : std_logic := '0';
00069    signal RAM_Start_Op : std_logic := '0';
00070    signal RAM_WE : std_logic := '0';
00071    signal RAM_ADDR : std_logic_vector(22 downto 0) := (others => '0');
00072    signal RAM_DIN : std_logic_vector(15 downto 0) := (others => '0');
00073
00074     --BiDirs
00075    signal MT_DATA : std_logic_vector(15 downto 0);
00076
00077     --Outputs
00078    signal MT_ADDR : std_logic_vector(22 downto 0);
00079    signal MT_OE : std_logic;
00080    signal MT_WE : std_logic;
00081    signal MT_ADV : std_logic;
00082    signal MT_CLK : std_logic;
00083    signal MT_UB : std_logic;
00084    signal MT_LB : std_logic;
00085    signal MT_CE : std_logic;
00086    signal MT_CRE : std_logic;
00087    signal RAM_Op_Done : std_logic;
00088    signal RAM_DOUT : std_logic_vector(15 downto 0);
00089
00090    -- Clock period definitions
00091    constant clk_period : time := 20ns;
00092
00093     signal MT_DATA_reg : STD_LOGIC_VECTOR(15 downto 0);
00094
00095 BEGIN
00096 MT_DATA              <= MT_DATA_reg when MT_WE = '1' else -- read op
00097                         (others => 'Z');
00098
00099
00100     -- Instantiate the Unit Under Test (UUT)
00101    uut: RAM_Module PORT MAP (
00102           clk => clk,
00103           reset => reset,
```

```
00104           MT_ADDR => MT_ADDR ,
00105           MT_DATA => MT_DATA ,
00106           MT_OE => MT_OE,
00107           MT_WE => MT_WE,
00108           MT_ADV => MT_ADV,
00109           MT_CLK => MT_CLK,
00110           MT_UB => MT_UB,
00111           MT_LB => MT_LB,
00112           MT_CE => MT_CE,
00113           MT_CRE => MT_CRE,
00114           MT_WAIT => MT_WAIT ,
00115           RAM_Start_Op => RAM_Start_Op,
00116           RAM_Op_Done => RAM_Op_Done,
00117           RAM_WE => RAM_WE,
00118           RAM_ADDR => RAM_ADDR,
00119           RAM_DOUT => RAM_DOUT,
00120           RAM_DIN => RAM_DIN
00121         );
00122
00123    -- Clock process definitions
00124    clk_process :process
00125    begin
00126         clk <= '0';
00127         wait for clk_period/2;
00128         clk <= '1';
00129         wait for clk_period/2;
00130    end process;
00131
00132
00133    -- Stimulus process
00134    stim_proc: process
00135    begin
00136         reset <= '0';
00137
00138       wait for 100ns;
00139         reset <= '1';
00140
00141         wait for 150us;
00142
00143         RAM_WE <= '0';
00144         RAM_ADDR <= "0000000" & x"0010";
00145         RAM_DIN <= x"A55A";
00146         wait for 20ns;
00147         RAM_Start_Op <= '1';
00148         wait for 20ns;
00149         RAM_Start_Op <= '0';
00150
00151         wait for 500ns;
00152
00153         RAM_WE <= '1';
00154         RAM_ADDR <= "0000000" & x"0010";
00155         --RAM_DIN <= x"A55A";
00156         MT_DATA_reg <= x"F0F0";
00157         wait for 20ns;
00158         RAM_Start_Op <= '1';
00159         wait for 20ns;
00160         RAM_Start_Op <= '0';
00161
00162
00163
00164       wait for clk_period*10;
00165
00166       -- insert stimulus here
00167
00168       wait;
00169    end process;
00170
00171 END;
```

RS232_Module.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
```

```
00005 -- Create Date:    19:44:35 01/27/2012
00006 -- Design Name:
00007 -- Module Name:    ADC_Module - Behavioral
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- Dependencies:
00014 --
00015 -- Revision:
00016 -- Revision 0.01 - File Created
00017 -- Additional Comments:
00018 --
00019 ----------------------------------------------------------------------------------
00020 library IEEE;
00021 use IEEE.STD_LOGIC_1164.ALL;
00022 use IEEE.STD_LOGIC_ARITH.ALL;
00023 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00024
00025 ---- Uncomment the following library declaration if instantiating
00026 ---- any Xilinx primitives in this code.
00027 --library UNISIM;
00028 --use UNISIM.VComponents.all;
00029
00030 entity RS232_Module is
00031     Port ( clk                      : in  STD_LOGIC;
00032            reset                      : in  STD_LOGIC;
00033
00034               RX                         : in  STD_LOGIC;
00035               TX                         : out STD_LOGIC;
00036               RX_led                     : out STD_LOGIC;
00037               TX_led                     : out STD_LOGIC;
00038
00039               -- RX_FIFO Signals
00040               RX_FIFO_RD_CLK          : in STD_LOGIC;
00041               RX_FIFO_DOUT              : out STD_LOGIC_VECTOR (7 downto 0);
00042               RX_FIFO_RD_EN           : in STD_LOGIC;
00043               RX_FIFO_EMPTY           : out STD_LOGIC;
00044
00045               -- TX_FIFO Signals
00046               TX_FIFO_WR_CLK          : in STD_LOGIC;
00047               TX_FIFO_DIN               : in STD_LOGIC_VECTOR (7 downto 0);
00048               TX_FIFO_WR_EN           : in STD_LOGIC
00049               );
00050 end RS232_Module;
00051
00052 architecture Behavioral of RS232_Module is
00053
00054 ----------------------------------------------------------------------------------
00055 ---------------------- TX_Module ---------------------------------------------
00056 ----------------------------------------------------------------------------------
00057 component TX_Module
00058     Port ( clk                    : in  STD_LOGIC;
00059            reset                    : in  STD_LOGIC;
00060               TX                       : out STD_LOGIC;
00061               TX_led                   : out STD_LOGIC;
00062
00063               -- TX_FIFO Signals
00064               FIFO_DOUT                : in  STD_LOGIC_VECTOR (7 downto 0);
00065               FIFO_RD_EN          : out STD_LOGIC;
00066               FIFO_EMPTY          : in  STD_LOGIC
00067               );
00068 end component;
00069
00070 ----------------------------------------------------------------------------------
00071 ---------------------- RX_Module ---------------------------------------------
00072 ----------------------------------------------------------------------------------
00073 component RX_Module
00074     Port ( clk                    : in  STD_LOGIC;
00075            reset                    : in  STD_LOGIC;
00076               RX                       : in  STD_LOGIC;
00077               RX_led                   : out STD_LOGIC;
00078
00079               -- RX_FIFO Signals
00080               FIFO_WR_CLK            : out  STD_LOGIC;
00081               FIFO_DIN              : out  STD_LOGIC_VECTOR (7 downto 0);
00082               FIFO_WR_EN          : out  STD_LOGIC
00083               );
```

```
00084 end component;
00085
00086 ------------------------------------------------------------------------------
00087 ---------------------- Clock_Divider -----------------------------------------
00088 ------------------------------------------------------------------------------
00089 signal RS232_CLK                        : STD_LOGIC;
00090 signal RS232_divide_count          : STD_LOGIC_VECTOR(7 downto 0);
00091
00092 component Clock_Divider
00093     Port ( clk_in : in  STD_LOGIC;
00094            reset : in  STD_LOGIC;
00095               divide_count : in STD_LOGIC_VECTOR(7 downto 0);
00096            clk_out : out  STD_LOGIC);
00097 end component;
00098
00099 ------------------------------------------------------------------------------
00100 ---------------------- FIFO_8_512 --------------------------------------------
00101 ------------------------------------------------------------------------------
00102 signal reset_inv                         : STD_LOGIC;
00103
00104 signal RX_FIFO_DIN                    : STD_LOGIC_VECTOR(7 downto 0);
00105 --signal RX_FIFO_RD_CLK              : STD_LOGIC;
00106 --signal RX_FIFO_RD_EN                  : STD_LOGIC;
00107 signal RX_FIFO_DOUT_sig             : STD_LOGIC_VECTOR(7 downto 0);
00108 signal RX_FIFO_WR_CLK              : STD_LOGIC;
00109 signal RX_FIFO_WR_EN                   : STD_LOGIC;
00110 --signal RX_FIFO_EMPTY                  : STD_LOGIC;
00111 signal RX_FIFO_FULL                 : STD_LOGIC;
00112
00113 --signal TX_FIFO_DIN                      : STD_LOGIC_VECTOR(7 downto 0);
00114 signal TX_FIFO_RD_CLK              : STD_LOGIC;
00115 signal TX_FIFO_RD_EN                   : STD_LOGIC;
00116 signal TX_FIFO_DOUT                 : STD_LOGIC_VECTOR(7 downto 0);
00117 --signal TX_FIFO_WR_CLK              : STD_LOGIC;
00118 --signal TX_FIFO_WR_EN                   : STD_LOGIC;
00119 signal TX_FIFO_EMPTY                  : STD_LOGIC;
00120 signal TX_FIFO_FULL                 : STD_LOGIC;
00121
00122 component FIFO_8_512 IS
00123     port (
00124     din: IN std_logic_VECTOR(7 downto 0);
00125     rd_clk: IN std_logic;
00126     rd_en: IN std_logic;
00127     rst: IN std_logic;
00128     wr_clk: IN std_logic;
00129     wr_en: IN std_logic;
00130     dout: OUT std_logic_VECTOR(7 downto 0);
00131     empty: OUT std_logic;
00132     full: OUT std_logic);
00133 END component;
00134
00135 begin
00136
00137 reset_inv <= not reset;
00138
00139 RX_FIFO_DOUT <= RX_FIFO_DOUT_sig;
00140
00141 ------------------------------------------------------------------------------
00142 ---------------------- TX_Module ---------------------------------------------
00143 ------------------------------------------------------------------------------
00144 TX_Mod : TX_Module
00145 port map(
00146     clk                              => RS232_CLK,
00147    reset                            => reset,
00148     TX                                    => TX,
00149     TX_led                             => TX_led,
00150     FIFO_DOUT                       => TX_FIFO_DOUT,
00151     FIFO_RD_EN                        => TX_FIFO_RD_EN,
00152     FIFO_EMPTY                        => TX_FIFO_EMPTY
00153     );
00154
00155 ------------------------------------------------------------------------------
00156 ---------------------- RX_Module ---------------------------------------------
00157 ------------------------------------------------------------------------------
00158 RX_Mod : RX_Module
00159 port map(
00160     clk                              => clk,
00161    reset                            => reset,
00162     RX                                    => RX,
```

```
00163     RX_led                            => RX_led,
00164     FIFO_WR_CLK                      => RX_FIFO_WR_CLK,
00165     FIFO_DIN                            => RX_FIFO_DIN,
00166     FIFO_WR_EN                        => RX_FIFO_WR_EN
00167     );
00168
00169 ----------------------------------------------------------------------------------
00170 ---------------------- Clock Divider --------------------------------------------
00171 ----------------------------------------------------------------------------------
00172 RS232_divide_count <= x"D8";  -- 50 MHz / 216 =
00173
00174 RS232_Clock : Clock_Divider
00175 port map(
00176     clk_in                            => clk,
00177    reset                             => reset,
00178     divide_count                     => RS232_divide_count,
00179    clk_out                           => RS232_CLK
00180     );
00181
00182 ----------------------------------------------------------------------------------
00183 ---------------------- RX_FIFO_8_512 --------------------------------------------
00184 ----------------------------------------------------------------------------------
00185
00186 RX_FIFO : FIFO_8_512
00187 port map(
00188     din                               => RX_FIFO_DIN,
00189     rd_clk                            => clk,
00190     rd_en                                 => RX_FIFO_RD_EN,
00191     rst                               => reset_inv,
00192     wr_clk                            => RX_FIFO_WR_CLK,
00193     wr_en                                 => RX_FIFO_WR_EN,
00194     dout                                  => RX_FIFO_DOUT_sig,
00195     empty                                 => RX_FIFO_EMPTY,
00196     full                                  => RX_FIFO_FULL
00197     );
00198
00199
00200 ----------------------------------------------------------------------------------
00201 ---------------------- TX_FIFO_8_512 --------------------------------------------
00202 ----------------------------------------------------------------------------------
00203
00204 TX_FIFO : FIFO_8_512
00205 port map(
00206     din                               => TX_FIFO_DIN,
00207     rd_clk                            => RS232_CLK,
00208     rd_en                                 => TX_FIFO_RD_EN,
00209     rst                               => reset_inv,
00210     wr_clk                            => TX_FIFO_WR_CLK,
00211     wr_en                                 => TX_FIFO_WR_EN,
00212     dout                                  => TX_FIFO_DOUT,
00213     empty                                 => TX_FIFO_EMPTY,
00214     full                                  => TX_FIFO_FULL
00215     );
00216
00217 end Behavioral;
00218
```

RS232_Test.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
00005 -- Create Date:    19:44:35 01/27/2012
00006 -- Design Name:
00007 -- Module Name:    ADC_Module - Behavioral
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- Dependencies:
00014 --
00015 -- Revision:
00016 -- Revision 0.01 - File Created
```

```
00017 -- Additional Comments:
00018 --
00019 ----------------------------------------------------------------------------------
00020 library IEEE;
00021 use IEEE.STD_LOGIC_1164.ALL;
00022 use IEEE.STD_LOGIC_ARITH.ALL;
00023 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00024
00025 use work.RS232_Test_pkg.all;
00026
00027 ---- Uncomment the following library declaration if instantiating
00028 ---- any Xilinx primitives in this code.
00029 --library UNISIM;
00030 --use UNISIM.VComponents.all;
00031
00032 entity RS232_Test is
00033     Port ( clk                          : in  STD_LOGIC;
00034            reset                          : in  STD_LOGIC;
00035
00036               RX                             : in  STD_LOGIC;
00037               TX                             : out STD_LOGIC;
00038               RX_led                         : out STD_LOGIC;
00039               TX_led                         : out STD_LOGIC
00040               );
00041 end RS232_Test;
00042
00043 architecture Behavioral of RS232_Test is
00044
00045 signal async_flags                    : STD_LOGIC_VECTOR(5 downto 0);
00046
00047 ----------------------------------------------------------------------------------
00048 ---------------------- RS232_Module ----------------------------------------------
00049 ----------------------------------------------------------------------------------
00050 signal RX_FIFO_RD_CLK             : STD_LOGIC;
00051 signal RX_FIFO_DOUT                 : STD_LOGIC_VECTOR (7 downto 0);
00052 signal RX_FIFO_RD_EN                   : STD_LOGIC;
00053 signal RX_FIFO_EMPTY                   : STD_LOGIC;
00054
00055 signal TX_FIFO_WR_CLK             : STD_LOGIC;
00056 signal TX_FIFO_DIN                   : STD_LOGIC_VECTOR(7 downto 0);
00057 signal TX_FIFO_WR_EN                   : STD_LOGIC;
00058
00059 component RS232_Module is
00060     Port ( clk                          : in  STD_LOGIC;
00061            reset                          : in  STD_LOGIC;
00062
00063               RX                             : in  STD_LOGIC;
00064               TX                             : out STD_LOGIC;
00065               RX_led                         : out STD_LOGIC;
00066               TX_led                         : out STD_LOGIC;
00067
00068               -- RX_FIFO Signals
00069               RX_FIFO_RD_CLK          : in STD_LOGIC;
00070               RX_FIFO_DOUT              : out STD_LOGIC_VECTOR (7 downto 0);
00071               RX_FIFO_RD_EN            : in STD_LOGIC;
00072               RX_FIFO_EMPTY            : out STD_LOGIC;
00073
00074               -- TX_FIFO Signals
00075               TX_FIFO_WR_CLK          : in STD_LOGIC;
00076               TX_FIFO_DIN                : in STD_LOGIC_VECTOR (7 downto 0);
00077               TX_FIFO_WR_EN            : in STD_LOGIC
00078               );
00079 end component;
00080
00081 component RS232_Test_states
00082 Port (  clk                           : in  STD_LOGIC;
00083             rst_n                        : in  STD_LOGIC;
00084             RX_FIFO_EMPTY              : in  STD_LOGIC;
00085             async_flags                  : out STD_LOGIC_VECTOR(5 downto 0)  --flags to enable
      functions
00086             );
00087 end component;
00088
00089
00090 begin
00091
00092 TX_FIFO_DIN <= RX_FIFO_DOUT;
00093 RX_FIFO_RD_CLK <= clk;
00094 TX_FIFO_WR_CLK <= clk;
```

```
00095
00096 -- FIFO_WR_EN
00097 process(clk, reset)
00098 begin
00099    if reset = '0' then
00100         RX_FIFO_RD_EN <= '0';
00101    elsif rising_edge(clk) then
00102         if async_flags(RX_RD_EN_FLAG) = '1' then
00103             RX_FIFO_RD_EN <= '1';
00104         else
00105             RX_FIFO_RD_EN <= '0';
00106         end if;
00107     end if;
00108 end process;
00109
00110 -- TX_FIFO_WR_EN
00111 process(clk, reset)
00112 begin
00113    if reset = '0' then
00114         TX_FIFO_WR_EN <= '0';
00115    elsif rising_edge(clk) then
00116         if async_flags(TX_WR_EN_FLAG) = '1' then
00117             TX_FIFO_WR_EN <= '1';
00118         else
00119             TX_FIFO_WR_EN <= '0';
00120         end if;
00121     end if;
00122 end process;
00123
00124 ---------------------------------------------------------------------------------
00125 --------------------- RS232_Module ----------------------------------------------
00126 ---------------------------------------------------------------------------------
00127 RS232 : RS232_Module
00128 port map(
00129     clk                                 => clk,
00130    reset                               => reset,
00131
00132     RX                                      => RX,
00133     TX                                      => TX ,
00134     RX_led                                 => RX_led,
00135     TX_led                                 => TX_led,
00136
00137     RX_FIFO_RD_CLK                        => RX_FIFO_RD_CLK,
00138     RX_FIFO_DOUT                            => RX_FIFO_DOUT,
00139     RX_FIFO_RD_EN                       => RX_FIFO_RD_EN,
00140     RX_FIFO_EMPTY                       => RX_FIFO_EMPTY,
00141
00142     TX_FIFO_WR_CLK                        => TX_FIFO_WR_CLK,
00143     TX_FIFO_DIN                           => TX_FIFO_DIN,
00144     TX_FIFO_WR_EN                       => TX_FIFO_WR_EN
00145     );
00146
00147
00148 States : RS232_Test_states
00149 port map(
00150     clk                          => clk,
00151     rst_n                      => reset,
00152     RX_FIFO_EMPTY              => RX_FIFO_EMPTY,
00153     async_flags                  => async_flags
00154     );
00155
00156 end Behavioral;
00157
00158
00159
```

RS232_Test_main_states.vhd

```
00001 ---------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
```

```
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 --
00019 -- Additional Comments:
00020 --
00021 ----------------------------------------------------------------------------------
00022 library IEEE;
00023 use IEEE.STD_LOGIC_1164.ALL;
00024 use IEEE.numeric_std.all;
00025 --use IEEE.STD_LOGIC_ARITH.ALL;
00026 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00027
00028 use work.RS232_Test_pkg.all;
00029
00030 ---- Uncomment the following library declaration if instantiating
00031 ---- any Xilinx primitives in this code.
00032 library UNISIM;
00033 use UNISIM.VComponents.all;
00034
00035 entity RS232_Test_states is
00036 Port (  clk                         : in  STD_LOGIC;
00037             rst_n                     : in  STD_LOGIC;
00038             RX_FIFO_EMPTY             : in  STD_LOGIC;
00039             async_flags                 : out STD_LOGIC_VECTOR(5 downto 0)  --flags to enable
      functions
00040             );
00041 end RS232_Test_states;
00042
00043 architecture Behavioral of RS232_Test_states is
00044
00045         --Control signals
00046
00047 signal curr_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM current state
00048 signal next_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM next state
00049
00050 begin
00051 ----------------------------------------------------------------------------------
00052         -- synchronous part of state machine here
00053 data_in_latch: process(clk, rst_n)
00054 begin
00055    if rst_n = '0' then
00056         curr_state <= (others => '0');
00057    elsif rising_edge(clk) then
00058         curr_state <= next_state;
00059    end if;
00060 end process;
00061
00062         -- async part of state machine to set function flags
00063 RS232_Test_state: process(rst_n, curr_state)
00064 begin
00065    if rst_n = '0' then
00066         async_flags <= (others => '0');
00067    else
00068         async_flags <= (others => '0');
00069         case curr_state is
00070
00071             when IDLE =>
00072                 --async_flags(IDLE_FLAG) <= '1';            -- init
00073                 --async_flags(CAPTURE_BYTE_FLAG) <= '1';
00074
00075             when RX_TO_TX =>
00076                 async_flags(TX_WR_EN_FLAG) <= '1';
00077                 async_flags(RX_RD_EN_FLAG) <= '1';
00078
00079             when FINISH =>
00080                 --async_flags(DONE_FLAG) <= '1';            -- done flag
00081             when others =>
00082                 async_flags <= (others => '0');
00083         end case;
00084     end if;
00085 end process;
```

```
00086 ------------------------------------------------
00087         -- RS232_Test state machine
00088 RS232_Test_asynch_state: process(rst_n, curr_state,
      RX_FIFO_EMPTY)
00089 begin
00090    if rst_n = '0' then
00091         next_state <= IDLE;
00092    else
00093         case curr_state is
00094         when IDLE =>
00095                 if RX_FIFO_EMPTY = '0' then
00096                     next_state <= RX_TO_TX;
00097                 else
00098                     next_state <= IDLE;
00099                 end if;
00100
00101             when RX_TO_TX =>
00102                 next_state <= WAIT_1;
00103
00104             when WAIT_1 =>
00105                 next_state <= WAIT_2;
00106             when WAIT_2 =>
00107                 next_state <= WAIT_3;
00108             when WAIT_3 =>
00109                 next_state <= FINISH;
00110
00111             when FINISH =>
00112                 next_state <= IDLE;
00113
00114             when OTHERS =>
00115                 next_state <= IDLE;
00116         end case;
00117    end if;
00118 end process;
00119
00120
00121 end Behavioral;
00122
00123
```

RS232_Test_pkg.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:     WMU - Thesis
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 --
00016 -- Dependencies:
00017 --
00018 -- Revision:
00019 -- Revision 0.01 - File Created
00020 -- Additional Comments:
00021 --
00022 ----------------------------------------------------------------------------------
00023 library IEEE;
00024 use IEEE.STD_LOGIC_1164.ALL;
00025 use IEEE.numeric_std.all;
00026 --use IEEE.STD_LOGIC_ARITH.ALL;
00027 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00028
00029 package RS232_Test_pkg is
00030
00031
00032 constant IDLE                                    : STD_LOGIC_VECTOR(7 downto 0) := x"00";
00033
```

```
00034 constant RX_TO_TX                                  : STD_LOGIC_VECTOR(7 downto 0) := x"01";
00035 constant WAIT_1                                  : STD_LOGIC_VECTOR(7 downto 0) := x"02";
00036 constant WAIT_2                                  : STD_LOGIC_VECTOR(7 downto 0) := x"03";
00037 constant WAIT_3                                  : STD_LOGIC_VECTOR(7 downto 0) := x"04";
00038
00039
00040 constant FINISH                                  : STD_LOGIC_VECTOR(7 downto 0) := x"FF";
00041
00042
00043
00044 --constant IDLE_FLAG                                 : integer := 0;
00045
00046 constant TX_WR_EN_FLAG                          : integer := 1;
00047 constant RX_RD_EN_FLAG                          : integer := 2;
00048
00049
00050
00051 --constant DONE_FLAG                                 : integer := 6;
00052
00053
00054
00055 end RS232_Test_pkg;
00056
00057 package body RS232_Test_pkg is
00058
00059 end RS232_Test_pkg;
00060
00061
```

RX_Module.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
00005 -- Create Date:    19:44:35 01/27/2012
00006 -- Design Name:
00007 -- Module Name:    ADC_Module - Behavioral
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- Dependencies:
00014 --
00015 -- Revision:
00016 -- Revision 0.01 - File Created
00017 -- Additional Comments:
00018 --
00019 ----------------------------------------------------------------------------------
00020 library IEEE;
00021 use IEEE.STD_LOGIC_1164.ALL;
00022 use IEEE.STD_LOGIC_ARITH.ALL;
00023 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00024
00025 ---- Uncomment the following library declaration if instantiating
00026 ---- any Xilinx primitives in this code.
00027 --library UNISIM;
00028 --use UNISIM.VComponents.all;
00029
00030 use work.RX_Module_pkg.all;
00031
00032 entity RX_Module is
00033     Port ( clk                  : in  STD_LOGIC;
00034            reset                  : in  STD_LOGIC;
00035               RX                     : in  STD_LOGIC;
00036               RX_led                 : out STD_LOGIC;
00037
00038               -- RX_FIFO Signals
00039               FIFO_WR_CLK            : out  STD_LOGIC;
00040               FIFO_DIN               : out  STD_LOGIC_VECTOR (7 downto 0);
00041               FIFO_WR_EN          : out  STD_LOGIC
00042               );
00043 end RX_Module;
00044
```

```
00045 architecture Behavioral of RX_Module is
00046
00047 signal async_flags                   : STD_LOGIC_VECTOR(5 downto 0);
00048 signal count                         : STD_LOGIC_VECTOR(7 downto 0);
00049 signal baud_count                     : STD_LOGIC_VECTOR(15 downto 0);
00050
00051 signal received_byte                  : STD_LOGIC_VECTOR(7 downto 0);
00052
00053 component RX_Module_states
00054 Port (  clk                           : in  STD_LOGIC;
00055            rst_n                       : in  STD_LOGIC;
00056            RX                                : in  STD_LOGIC;
00057            baud_count                    : in  STD_LOGIC_VECTOR(15 downto 0);
00058            count                          : in  STD_LOGIC_VECTOR(7 downto 0);
00059            async_flags                  : out STD_LOGIC_VECTOR(5 downto 0)  --flags to enable
     functions
00060            );
00061 end component;
00062
00063 begin
00064
00065 FIFO_WR_CLK <= clk;
00066
00067 RX_led <= not async_flags(IDLE_FLAG);
00068
00069 FIFO_DIN <= received_byte;
00070
00071 -- received_byte
00072 process(clk, reset)
00073 begin
00074    if reset = '0' then
00075         received_byte <= (others => '0');
00076    elsif rising_edge(clk) and async_flags(CAPTURE_BYTE_FLAG) = '1' then
00077         --received_byte <= RX & received_byte(7 downto 1);
00078         case count is
00079             when x"00" =>               received_byte(0) <= RX;
00080             when x"01" =>               received_byte(1) <= RX;
00081             when x"02" =>               received_byte(2) <= RX;
00082             when x"03" =>               received_byte(3) <= RX;
00083             when x"04" =>               received_byte(4) <= RX;
00084             when x"05" =>               received_byte(5) <= RX;
00085             when x"06" =>               received_byte(6) <= RX;
00086             when x"07" =>               received_byte(7) <= RX;
00087             when OTHERS =>
00088         end case;
00089     end if;
00090 end process;
00091
00092 -- FIFO_WR_EN
00093 process(clk, reset)
00094 begin
00095    if reset = '0' then
00096         FIFO_WR_EN <= '0';
00097    elsif rising_edge(clk) then
00098         if async_flags(WR_EN_FLAG) = '1' then
00099             FIFO_WR_EN <= '1';
00100         else
00101             FIFO_WR_EN <= '0';
00102         end if;
00103     end if;
00104 end process;
00105
00106 ----------------------------------------------------------------------------
00107 ---------------------- RX_Module_states ------------------------------------
00108 ----------------------------------------------------------------------------
00109
00110 states : RX_Module_states
00111 port map(
00112     clk                             => clk,
00113     rst_n                          => reset,
00114     RX                                  => RX,
00115     baud_count                      => baud_count,
00116     count                            => count,
00117     async_flags                    => async_flags
00118     );
00119
00120 ----------------------------------------------------------------------------
00121 ---------------------- Counter ---------------------------------------------
00122 ----------------------------------------------------------------------------
```

```
00123
00124 -- count
00125 process(clk, reset)
00126 begin
00127    if reset = '0' then
00128         count <= (others => '0');
00129    elsif rising_edge(clk) then
00130         if async_flags(INC_COUNT_FLAG) = '1' then
00131             count <= count + 1;
00132         elsif async_flags(IDLE_FLAG) = '1' then
00133             count <= x"00";
00134         end if;
00135     end if;
00136 end process;
00137
00138 -- baud_count
00139 process(clk, reset)
00140 begin
00141    if reset = '0' then
00142         baud_count <= (others => '0');
00143    elsif rising_edge(clk) then
00144         if async_flags(INC_BAUD_COUNT_FLAG) = '1' then
00145             baud_count <= baud_count + 1;
00146         elsif async_flags(CLEAR_BAUD_COUNT_FLAG) = '1' then
00147             baud_count <= (others => '0');
00148         elsif async_flags(IDLE_FLAG) = '1' then
00149             baud_count <= (others => '0');
00150         end if;
00151     end if;
00152 end process;
00153
00154
00155 end Behavioral;
00156
```

RX_Module_main_states.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 --
00019 -- Additional Comments:
00020 --
00021 ----------------------------------------------------------------------------------
00022 library IEEE;
00023 use IEEE.STD_LOGIC_1164.ALL;
00024 use IEEE.numeric_std.all;
00025 --use IEEE.STD_LOGIC_ARITH.ALL;
00026 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00027
00028 use work.RX_Module_pkg.all;
00029
00030 ---- Uncomment the following library declaration if instantiating
00031 ---- any Xilinx primitives in this code.
00032 library UNISIM;
00033 use UNISIM.VComponents.all;
00034
00035 entity RX_Module_states is
00036 Port (  clk                         : in  STD_LOGIC;
00037             rst_n                     : in  STD_LOGIC;
00038             RX                              : in  STD_LOGIC;
```

```
00039             baud_count                    : in  STD_LOGIC_VECTOR(15 downto 0);
00040             count                         : in  STD_LOGIC_VECTOR(7 downto 0);
00041             async_flags                   : out STD_LOGIC_VECTOR(5 downto 0)  --flags to enable
      functions
00042             );
00043 end RX_Module_states;
00044
00045 architecture Behavioral of RX_Module_states is
00046
00047         --Control signals
00048
00049 signal curr_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM current state
00050 signal next_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM next state
00051
00052 begin
00053 -------------------------------------------------------------------------------
00054         -- synchronous part of state machine here
00055 data_in_latch: process(clk, rst_n)
00056 begin
00057    if rst_n = '0' then
00058         curr_state <= (others => '0');
00059    elsif rising_edge(clk) then
00060         curr_state <= next_state;
00061    end if;
00062 end process;
00063
00064         -- async part of state machine to set function flags
00065 RX_Module_state: process(rst_n, curr_state)
00066 begin
00067    if rst_n = '0' then
00068         async_flags <= (others => '0');
00069    else
00070         async_flags <= (others => '0');
00071         case curr_state is
00072
00073             when IDLE =>
00074                 async_flags(IDLE_FLAG) <= '1';            -- init
00075                 --async_flags(CAPTURE_BYTE_FLAG) <= '1';
00076
00077             when WAIT_BAUD =>
00078                 async_flags(INC_BAUD_COUNT_FLAG) <= '1';
00079
00080             when WAIT_BAUD2 =>
00081                 async_flags(INC_BAUD_COUNT_FLAG) <= '1';
00082
00083             when CAPTURE_BYTE =>
00084                 async_flags(CAPTURE_BYTE_FLAG) <= '1';
00085                 async_flags(INC_COUNT_FLAG) <= '1';
00086                 async_flags(CLEAR_BAUD_COUNT_FLAG) <= '1';
00087
00088             when WRITE_TO_FIFO =>
00089                 async_flags(WR_EN_FLAG) <= '1';
00090
00091             when WAIT_BAUD3 =>
00092                 async_flags(INC_BAUD_COUNT_FLAG) <= '1';
00093
00094
00095             when FINISH =>
00096                 --async_flags(DONE_FLAG) <= '1';            -- done flag
00097             when others =>
00098                 async_flags <= (others => '0');
00099         end case;
00100     end if;
00101 end process;
00102 -------------------------------------------------
00103         -- RX_Module state machine
00104 RX_Module_asynch_state: process(rst_n, curr_state,
      RX, count, baud_count)
00105 begin
00106    if rst_n = '0' then
00107         next_state <= IDLE;
00108    else
00109         case curr_state is
00110         when IDLE =>
00111                 if RX = '0' then
00112                     next_state <= WAIT_BAUD;
00113                 else
00114                     next_state <= IDLE;
00115                 end if;
```

```
00116
00117             when WAIT_BAUD =>
00118                 if baud_count = x"028B"   then
00119                     next_state <= CAPTURE_BYTE;
00120                 else
00121                     next_state <= WAIT_BAUD;
00122                 end if;
00123             when WAIT_BAUD2 =>
00124                 if baud_count = x"01B2" then
00125                     next_state <= CAPTURE_BYTE;
00126                 else
00127                     next_state <= WAIT_BAUD2;
00128                 end if;
00129
00130             when CAPTURE_BYTE =>
00131                 if count = x"07" then
00132                     next_state <= WAIT_BAUD3;
00133                 else
00134                     next_state <= WAIT_BAUD2;
00135                 end if;
00136
00137             when WAIT_BAUD3 =>
00138                 if baud_count = x"0100" then --FA
00139                     next_state <= WRITE_TO_FIFO;
00140                 else
00141                     next_state <= WAIT_BAUD3;
00142                 end if;
00143
00144             when WRITE_TO_FIFO =>
00145                 next_state <= FINISH;
00146
00147             when FINISH =>
00148                 --if RX = '1' then
00149                     next_state <= IDLE;
00150                 --else
00151                 --  next_state <= FINISH;
00152                 --end if;
00153
00154             when OTHERS =>
00155                 next_state <= IDLE;
00156         end case;
00157    end if;
00158 end process;
00159
00160
00161 end Behavioral;
00162
00163
```

RX_Module_pkg.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:     WMU - Thesis
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 --
00016 -- Dependencies:
00017 --
00018 -- Revision:
00019 -- Revision 0.01 - File Created
00020 -- Additional Comments:
00021 --
00022 ----------------------------------------------------------------------------------
00023 library IEEE;
00024 use IEEE.STD_LOGIC_1164.ALL;
```

```
00025 use IEEE.numeric_std.all;
00026 --use IEEE.STD_LOGIC_ARITH.ALL;
00027 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00028
00029 package RX_Module_pkg is
00030
00031
00032 constant IDLE                                   : STD_LOGIC_VECTOR(7 downto 0) := x"00";
00033
00034 constant WAIT_BAUD                            : STD_LOGIC_VECTOR(7 downto 0) := x"01";
00035 constant CAPTURE_BYTE                          : STD_LOGIC_VECTOR(7 downto 0) := x"02";
00036 constant WRITE_TO_FIFO                        : STD_LOGIC_VECTOR(7 downto 0) := x"03";
00037 constant WAIT_BAUD2                              : STD_LOGIC_VECTOR(7 downto 0) := x"04";
00038 constant WAIT_BAUD3                              : STD_LOGIC_VECTOR(7 downto 0) := x"05";
00039
00040
00041 constant FINISH                                  : STD_LOGIC_VECTOR(7 downto 0) := x"FF";
00042
00043
00044
00045 constant IDLE_FLAG                            : integer := 0;
00046
00047 constant CAPTURE_BYTE_FLAG                    : integer := 1;
00048 constant INC_COUNT_FLAG                          : integer := 2;
00049 constant WR_EN_FLAG                              : integer := 3;
00050 constant INC_BAUD_COUNT_FLAG                  : integer := 4;
00051 constant CLEAR_BAUD_COUNT_FLAG              : integer := 5;
00052
00053
00054 --constant DONE_FLAG                               : integer := 6;
00055
00056
00057
00058 end RX_Module_pkg;
00059
00060 package body RX_Module_pkg is
00061
00062 end RX_Module_pkg;
00063
00064
```

SPI.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
00005 -- Create Date:    16:52:43 09/20/2010
00006 -- Design Name:
00007 -- Module Name:    SPI - Behavioral
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- Dependencies:
00014 --
00015 -- Revision:
00016 -- Revision 0.01 - File Created
00017 -- Additional Comments:
00018 --
00019 ----------------------------------------------------------------------------------
00020 library IEEE;
00021 use IEEE.STD_LOGIC_1164.ALL;
00022 use IEEE.STD_LOGIC_ARITH.ALL;
00023 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00024
00025 ---- Uncomment the following library declaration if instantiating
00026 ---- any Xilinx primitives in this code.
00027 --library UNISIM;
00028 --use UNISIM.VComponents.all;
00029
00030 use work.SPI_pkg.all;
00031
00032 entity SPI is
```

```
00033     Port ( data_reg                : in  STD_LOGIC_VECTOR(31 downto 0);
00034               SPI_start            : in  STD_LOGIC;
00035               CLK                   : in  STD_LOGIC;
00036               rst                   : in  STD_LOGIC;
00037            CS                        : out STD_LOGIC;
00038            MOSI                     : out STD_LOGIC;
00039               SPI_Done              : out STD_LOGIC;
00040
00041               -- SPI_Arbiter
00042               SPI_Bus_Request       : in  STD_LOGIC_VECTOR(7 downto 0);
00043               SPI_Bus_Busy          : in  STD_LOGIC;
00044               SPI_Bus_Grant         : out STD_LOGIC_VECTOR(7 downto 0)
00045               );
00046 end SPI;
00047
00048 architecture Behavioral of SPI is
00049
00050 component Arbiter
00051     Port ( clk                      : in  STD_LOGIC;
00052            reset                    : in  STD_LOGIC;
00053               Bus_Request           : in  STD_LOGIC_VECTOR(7 downto 0);
00054               Bus_Busy              : in  STD_LOGIC;
00055
00056               Bus_Grant             : out  STD_LOGIC_VECTOR(7 downto 0)
00057               );
00058 end component;
00059
00060 component SPI_states
00061 Port (  clk                    : in  STD_LOGIC;
00062            rst                  : in  STD_LOGIC;
00063            SPI_start            : in  STD_LOGIC;            --start S.M. into motion
00064            --count                : in  STD_LOGIC_VECTOR(4 downto 0);
00065            async_flags          : out STD_LOGIC_VECTOR(33 downto 0) --flags to enable
      functions
00066               );
00067 end component;
00068
00069 signal count : integer range 0 to 7;
00070 --signal count : STD_LOGIC_VECTOR(4 downto 0);
00071 signal async_flags : STD_LOGIC_VECTOR(33 downto 0);  --flags to enable functions
00072
00073 begin
00074
00075
00076 ----------------------------------------------------------------------------
00077 ---------------------- SPI_states ------------------------------------------
00078 ----------------------------------------------------------------------------
00079
00080 states : SPI_states
00081 port map(
00082     clk                     => clk,
00083     rst                     => rst,
00084     SPI_start             => SPI_start,
00085     async_flags          => async_flags
00086     );
00087
00088
00089 ----------------------------------------------------------------------------
00090 ---------------------- SPI_Arbiter -----------------------------------------
00091 ----------------------------------------------------------------------------
00092
00093 SPI_Arbiter : Arbiter
00094 port map(
00095     clk                           => clk,
00096     reset                        => rst,
00097     Bus_Request                   => SPI_Bus_Request,
00098     Bus_Busy                             => SPI_Bus_Busy,
00099     Bus_Grant                       => SPI_Bus_Grant
00100     );
00101
00102
00103 --CS <= not async_flags(OUTPUT_FLAG);
00104
00105 --process(clk, rst)
00106 --begin
00107 --  if rst = '0' then
00108 --      CS <= '1';
00109 --  elsif rising_edge(clk) then
00110 --      if async_flags(OUTPUT_FLAG) = '1' then
```

```
00111 --          CS <= '0';
00112 --      end if;
00113 --      if async_flags(DONE_FLAG) = '1' then
00114 --          CS <= '1';
00115 --      end if;
00116 --      if async_flags(INIT_FLAG) = '1' then
00117 --          CS <= '1';
00118 --      end if;
00119 --  end if;
00120 --end process;
00121
00122 process(clk, rst)
00123 begin
00124     if rst = '0' then
00125         MOSI <= '0';
00126         CS <= '1';
00127     elsif rising_edge(clk) then
00128 --      if async_flags(OUTPUT_FLAG) = '1' then
00129 --          --MOSI <= data_reg(CONV_INTEGER(count));
00130 --          MOSI <= data_reg(count);
00131 --          --count <= count - 1;
00132 --          count <= count + 1;
00133 --      end if;
00134 --      if async_flags(DONE_FLAG) = '1' then
00135 --          count <= (others => '0');
00136 --      end if;
00137 --      if async_flags(INIT_FLAG) = '1' then
00138 --          --count <= "00111";
00139 --          count <= 0;
00140 --      end if;
00141         if async_flags(DONE_FLAG) = '1' then
00142             CS <= '1';
00143         end if;
00144         if async_flags(INIT_FLAG) = '1' then
00145             CS <= '1';
00146         end if;
00147
00148
00149         if async_flags(B31_FLAG) = '1' then
00150             MOSI <= data_reg(B31_FLAG);
00151             CS <= '0';
00152         elsif async_flags(B30_FLAG) = '1' then
00153             MOSI <= data_reg(B30_FLAG);
00154             CS <= '0';
00155         elsif async_flags(B29_FLAG) = '1' then
00156             MOSI <= data_reg(B29_FLAG);
00157             CS <= '0';
00158         elsif async_flags(B28_FLAG) = '1' then
00159             MOSI <= data_reg(B28_FLAG);
00160             CS <= '0';
00161         elsif async_flags(B27_FLAG) = '1' then
00162             MOSI <= data_reg(B27_FLAG);
00163             CS <= '0';
00164         elsif async_flags(B26_FLAG) = '1' then
00165             MOSI <= data_reg(B26_FLAG);
00166             CS <= '0';
00167         elsif async_flags(B25_FLAG) = '1' then
00168             MOSI <= data_reg(B25_FLAG);
00169             CS <= '0';
00170         elsif async_flags(B24_FLAG) = '1' then
00171             MOSI <= data_reg(B24_FLAG);
00172             CS <= '0';
00173         elsif async_flags(B23_FLAG) = '1' then
00174             MOSI <= data_reg(B23_FLAG);
00175             CS <= '0';
00176         elsif async_flags(B22_FLAG) = '1' then
00177             MOSI <= data_reg(B22_FLAG);
00178             CS <= '0';
00179         elsif async_flags(B21_FLAG) = '1' then
00180             MOSI <= data_reg(B21_FLAG);
00181             CS <= '0';
00182         elsif async_flags(B20_FLAG) = '1' then
00183             MOSI <= data_reg(B20_FLAG);
00184             CS <= '0';
00185         elsif async_flags(B19_FLAG) = '1' then
00186             MOSI <= data_reg(B19_FLAG);
00187             CS <= '0';
00188         elsif async_flags(B18_FLAG) = '1' then
00189             MOSI <= data_reg(B18_FLAG);
```

```
00190             CS <= '0';
00191         elsif async_flags(B17_FLAG) = '1' then
00192             MOSI <= data_reg(B17_FLAG);
00193             CS <= '0';
00194         elsif async_flags(B16_FLAG) = '1' then
00195             MOSI <= data_reg(B16_FLAG);
00196             CS <= '0';
00197         elsif async_flags(B15_FLAG) = '1' then
00198             MOSI <= data_reg(B15_FLAG);
00199             CS <= '0';
00200         elsif async_flags(B14_FLAG) = '1' then
00201             MOSI <= data_reg(B14_FLAG);
00202             CS <= '0';
00203         elsif async_flags(B13_FLAG) = '1' then
00204             MOSI <= data_reg(B13_FLAG);
00205             CS <= '0';
00206         elsif async_flags(B12_FLAG) = '1' then
00207             MOSI <= data_reg(B12_FLAG);
00208             CS <= '0';
00209         elsif async_flags(B11_FLAG) = '1' then
00210             MOSI <= data_reg(B11_FLAG);
00211             CS <= '0';
00212         elsif async_flags(B10_FLAG) = '1' then
00213             MOSI <= data_reg(B10_FLAG);
00214             CS <= '0';
00215         elsif async_flags(B9_FLAG) = '1' then
00216             MOSI <= data_reg(B9_FLAG);
00217             CS <= '0';
00218         elsif async_flags(B8_FLAG) = '1' then
00219             MOSI <= data_reg(B8_FLAG);
00220             CS <= '0';
00221         elsif async_flags(B7_FLAG) = '1' then
00222             MOSI <= data_reg(B7_FLAG);
00223             CS <= '0';
00224         elsif async_flags(B6_FLAG) = '1' then
00225             MOSI <= data_reg(B6_FLAG);
00226             CS <= '0';
00227         elsif async_flags(B5_FLAG) = '1' then
00228             MOSI <= data_reg(B5_FLAG);
00229             CS <= '0';
00230         elsif async_flags(B4_FLAG) = '1' then
00231             MOSI <= data_reg(B4_FLAG);
00232             CS <= '0';
00233         elsif async_flags(B3_FLAG) = '1' then
00234             MOSI <= data_reg(B3_FLAG);
00235             CS <= '0';
00236         elsif async_flags(B2_FLAG) = '1' then
00237             MOSI <= data_reg(B2_FLAG);
00238             CS <= '0';
00239         elsif async_flags(B1_FLAG) = '1' then
00240             MOSI <= data_reg(B1_FLAG);
00241             CS <= '0';
00242         elsif async_flags(B0_FLAG) = '1' then
00243             MOSI <= data_reg(B0_FLAG);
00244             CS <= '0';
00245         end if;
00246     end if;
00247 end process;
00248 
00249 SPI_Done <= async_flags(DONE_FLAG);
00250 
00251 end Behavioral;
00252 
```

SPI_main_states.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
```

```
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 --
00019 -- Additional Comments:
00020 --
00021 ----------------------------------------------------------------------------------
00022 library IEEE;
00023 use IEEE.STD_LOGIC_1164.ALL;
00024 use IEEE.numeric_std.all;
00025 use IEEE.STD_LOGIC_ARITH.ALL;
00026 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00027
00028 use work.SPI_pkg.all;
00029
00030 ---- Uncomment the following library declaration if instantiating
00031 ---- any Xilinx primitives in this code.
00032 library UNISIM;
00033 use UNISIM.VComponents.all;
00034
00035 entity SPI_states is
00036 Port (  clk                     : in  STD_LOGIC;
00037             rst                     : in  STD_LOGIC;
00038             SPI_start               : in  STD_LOGIC;            --start S.M. into motion
00039         --  count                   : in  STD_LOGIC_VECTOR(4 downto 0);
00040             async_flags             : out STD_LOGIC_VECTOR(33 downto 0) --flags to enable
      functions
00041              );
00042 end SPI_states;
00043
00044 architecture Behavioral of SPI_states is
00045
00046         --Control signals
00047
00048 signal curr_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM current state
00049 signal next_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM next state
00050
00051 begin
00052 ----------------------------------------------------------------------------------
00053         -- synchronous part of state machine here
00054 data_in_latch: process(clk, rst)
00055 begin
00056    if rst = '0' then
00057         curr_state <= (others => '0');
00058    elsif rising_edge(clk) then
00059         curr_state <= next_state;
00060    end if;
00061 end process;
00062
00063         -- async part of state machine to set function flags
00064 SPI_state: process(rst, curr_state)
00065 begin
00066    if rst = '0' then
00067         async_flags <= (others => '0');
00068    else
00069         async_flags <= (others => '0');
00070         case curr_state is
00071             when IDLE =>
00072                 async_flags(INIT_FLAG) <= '1';
00073             --when OUTPUT_DATA =>
00074                 --async_flags(OUTPUT_FLAG) <= '1';
00075
00076             when OUTPUT_B31 =>
00077                 async_flags(B31_FLAG) <= '1';
00078             when OUTPUT_B30 =>
00079                 async_flags(B30_FLAG) <= '1';
00080             when OUTPUT_B29 =>
00081                 async_flags(B29_FLAG) <= '1';
00082             when OUTPUT_B28 =>
00083                 async_flags(B28_FLAG) <= '1';
00084             when OUTPUT_B27 =>
00085                 async_flags(B27_FLAG) <= '1';
00086             when OUTPUT_B26 =>
00087                 async_flags(B26_FLAG) <= '1';
```

```
00088             when OUTPUT_B25 =>
00089                 async_flags(B25_FLAG) <= '1';
00090             when OUTPUT_B24 =>
00091                 async_flags(B24_FLAG) <= '1';
00092             when OUTPUT_B23 =>
00093                 async_flags(B23_FLAG) <= '1';
00094             when OUTPUT_B22 =>
00095                 async_flags(B22_FLAG) <= '1';
00096             when OUTPUT_B21 =>
00097                 async_flags(B21_FLAG) <= '1';
00098             when OUTPUT_B20 =>
00099                 async_flags(B20_FLAG) <= '1';
00100             when OUTPUT_B19 =>
00101                 async_flags(B19_FLAG) <= '1';
00102             when OUTPUT_B18 =>
00103                 async_flags(B18_FLAG) <= '1';
00104             when OUTPUT_B17 =>
00105                 async_flags(B17_FLAG) <= '1';
00106             when OUTPUT_B16 =>
00107                 async_flags(B16_FLAG) <= '1';
00108             when OUTPUT_B15 =>
00109                 async_flags(B15_FLAG) <= '1';
00110             when OUTPUT_B14 =>
00111                 async_flags(B14_FLAG) <= '1';
00112             when OUTPUT_B13 =>
00113                 async_flags(B13_FLAG) <= '1';
00114             when OUTPUT_B12 =>
00115                 async_flags(B12_FLAG) <= '1';
00116             when OUTPUT_B11 =>
00117                 async_flags(B11_FLAG) <= '1';
00118             when OUTPUT_B10 =>
00119                 async_flags(B10_FLAG) <= '1';
00120             when OUTPUT_B9 =>
00121                 async_flags(B9_FLAG) <= '1';
00122             when OUTPUT_B8 =>
00123                 async_flags(B8_FLAG) <= '1';
00124             when OUTPUT_B7 =>
00125                 async_flags(B7_FLAG) <= '1';
00126             when OUTPUT_B6 =>
00127                 async_flags(B6_FLAG) <= '1';
00128             when OUTPUT_B5 =>
00129                 async_flags(B5_FLAG) <= '1';
00130             when OUTPUT_B4 =>
00131                 async_flags(B4_FLAG) <= '1';
00132             when OUTPUT_B3 =>
00133                 async_flags(B3_FLAG) <= '1';
00134             when OUTPUT_B2 =>
00135                 async_flags(B2_FLAG) <= '1';
00136             when OUTPUT_B1 =>
00137                 async_flags(B1_FLAG) <= '1';
00138             when OUTPUT_B0 =>
00139                 async_flags(B0_FLAG) <= '1';
00140 
00141             when TX_COMPLETE =>
00142                 async_flags(DONE_FLAG) <= '1';
00143             when others =>
00144                 async_flags <= (others => '0');
00145         end case;
00146     end if;
00147 end process;
00148 ------------------------------------------------
00149         -- SPI state machine
00150 SPI_async_state: process(rst, curr_state, SPI_start)
00151 begin
00152    if rst = '0' then
00153         next_state <= (others => '0');
00154    else
00155         case curr_state is
00156 
00157             when IDLE =>
00158                 if SPI_start = '1' then
00159                     next_state <= OUTPUT_B31;
00160                 else
00161                     next_state <= IDLE;
00162                 end if;
00163 
00164 --          when OUTPUT_DATA =>
00165 --              if count < 7 then
00166 --                  --next_state <= TX_COMPLETE;
```

```
00167 --                 next_state <= OUTPUT_DATA;
00168 --             else
00169 --                 next_state <= TX_COMPLETE;
00170 --                 --next_state <= OUTPUT_DATA;
00171 --             end if;
00172
00173             when OUTPUT_B31 =>
00174                 next_state <= OUTPUT_B30;
00175             when OUTPUT_B30 =>
00176                 next_state <= OUTPUT_B29;
00177             when OUTPUT_B29 =>
00178                 next_state <= OUTPUT_B28;
00179             when OUTPUT_B28 =>
00180                 next_state <= OUTPUT_B27;
00181             when OUTPUT_B27 =>
00182                 next_state <= OUTPUT_B26;
00183             when OUTPUT_B26 =>
00184                 next_state <= OUTPUT_B25;
00185             when OUTPUT_B25 =>
00186                 next_state <= OUTPUT_B24;
00187             when OUTPUT_B24 =>
00188                 next_state <= OUTPUT_B23;
00189             when OUTPUT_B23 =>
00190                 next_state <= OUTPUT_B22;
00191             when OUTPUT_B22 =>
00192                 next_state <= OUTPUT_B21;
00193             when OUTPUT_B21 =>
00194                 next_state <= OUTPUT_B20;
00195             when OUTPUT_B20 =>
00196                 next_state <= OUTPUT_B19;
00197             when OUTPUT_B19 =>
00198                 next_state <= OUTPUT_B18;
00199             when OUTPUT_B18 =>
00200                 next_state <= OUTPUT_B17;
00201             when OUTPUT_B17 =>
00202                 next_state <= OUTPUT_B16;
00203             when OUTPUT_B16 =>
00204                 next_state <= OUTPUT_B15;
00205             when OUTPUT_B15 =>
00206                 next_state <= OUTPUT_B14;
00207             when OUTPUT_B14 =>
00208                 next_state <= OUTPUT_B13;
00209             when OUTPUT_B13 =>
00210                 next_state <= OUTPUT_B12;
00211             when OUTPUT_B12 =>
00212                 next_state <= OUTPUT_B11;
00213             when OUTPUT_B11 =>
00214                 next_state <= OUTPUT_B10;
00215             when OUTPUT_B10 =>
00216                 next_state <= OUTPUT_B9;
00217             when OUTPUT_B9 =>
00218                 next_state <= OUTPUT_B8;
00219             when OUTPUT_B8 =>
00220                 next_state <= OUTPUT_B7;
00221             when OUTPUT_B7 =>
00222                 next_state <= OUTPUT_B6;
00223             when OUTPUT_B6 =>
00224                 next_state <= OUTPUT_B5;
00225             when OUTPUT_B5 =>
00226                 next_state <= OUTPUT_B4;
00227             when OUTPUT_B4 =>
00228                 next_state <= OUTPUT_B3;
00229             when OUTPUT_B3 =>
00230                 next_state <= OUTPUT_B2;
00231             when OUTPUT_B2 =>
00232                 next_state <= OUTPUT_B1;
00233             when OUTPUT_B1 =>
00234                 next_state <= OUTPUT_B0;
00235             when OUTPUT_B0 =>
00236                 next_state <= TX_COMPLETE;
00237
00238             when TX_COMPLETE =>
00239                 --if SPI_start = '0' then
00240                     next_state <= IDLE;
00241                 --else
00242                 --  next_state <= TX_COMPLETE;
00243                 --end if;
00244
00245             when others =>
```

```
00246                 next_state <= IDLE;
00247
00248         end case;
00249    end if;
00250 end process;
00251
00252
00253 end Behavioral;
00254
```

SPI_pkg.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 --
00019 -- Additional Comments:
00020 --
00021 ----------------------------------------------------------------------------------
00022 library IEEE;
00023 use IEEE.STD_LOGIC_1164.ALL;
00024 use IEEE.numeric_std.all;
00025 --use IEEE.STD_LOGIC_ARITH.ALL;
00026 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00027
00028 package SPI_pkg is
00029
00030
00031 constant IDLE                          : STD_LOGIC_VECTOR(7 downto 0) := x"00";
00032 --constant OUTPUT_DATA              : STD_LOGIC_VECTOR(7 downto 0) := x"01";
00033 constant OUTPUT_B31                  : STD_LOGIC_VECTOR(7 downto 0) := x"01";
00034 constant OUTPUT_B30                  : STD_LOGIC_VECTOR(7 downto 0) := x"02";
00035 constant OUTPUT_B29                  : STD_LOGIC_VECTOR(7 downto 0) := x"03";
00036 constant OUTPUT_B28                  : STD_LOGIC_VECTOR(7 downto 0) := x"04";
00037 constant OUTPUT_B27                  : STD_LOGIC_VECTOR(7 downto 0) := x"05";
00038 constant OUTPUT_B26                  : STD_LOGIC_VECTOR(7 downto 0) := x"06";
00039 constant OUTPUT_B25                  : STD_LOGIC_VECTOR(7 downto 0) := x"07";
00040 constant OUTPUT_B24                  : STD_LOGIC_VECTOR(7 downto 0) := x"08";
00041 constant OUTPUT_B23                  : STD_LOGIC_VECTOR(7 downto 0) := x"09";
00042 constant OUTPUT_B22                  : STD_LOGIC_VECTOR(7 downto 0) := x"10";
00043 constant OUTPUT_B21                  : STD_LOGIC_VECTOR(7 downto 0) := x"11";
00044 constant OUTPUT_B20                  : STD_LOGIC_VECTOR(7 downto 0) := x"12";
00045 constant OUTPUT_B19                  : STD_LOGIC_VECTOR(7 downto 0) := x"13";
00046 constant OUTPUT_B18                  : STD_LOGIC_VECTOR(7 downto 0) := x"14";
00047 constant OUTPUT_B17                  : STD_LOGIC_VECTOR(7 downto 0) := x"15";
00048 constant OUTPUT_B16                  : STD_LOGIC_VECTOR(7 downto 0) := x"16";
00049 constant OUTPUT_B15                  : STD_LOGIC_VECTOR(7 downto 0) := x"17";
00050 constant OUTPUT_B14                  : STD_LOGIC_VECTOR(7 downto 0) := x"18";
00051 constant OUTPUT_B13                  : STD_LOGIC_VECTOR(7 downto 0) := x"19";
00052 constant OUTPUT_B12                  : STD_LOGIC_VECTOR(7 downto 0) := x"1A";
00053 constant OUTPUT_B11                  : STD_LOGIC_VECTOR(7 downto 0) := x"1B";
00054 constant OUTPUT_B10                  : STD_LOGIC_VECTOR(7 downto 0) := x"1C";
00055 constant OUTPUT_B9                 : STD_LOGIC_VECTOR(7 downto 0) := x"1D";
00056 constant OUTPUT_B8                     : STD_LOGIC_VECTOR(7 downto 0) := x"1E";
00057 constant OUTPUT_B7                 : STD_LOGIC_VECTOR(7 downto 0) := x"1F";
00058 constant OUTPUT_B6                     : STD_LOGIC_VECTOR(7 downto 0) := x"20";
00059 constant OUTPUT_B5                     : STD_LOGIC_VECTOR(7 downto 0) := x"21";
00060 constant OUTPUT_B4                     : STD_LOGIC_VECTOR(7 downto 0) := x"22";
00061 constant OUTPUT_B3                     : STD_LOGIC_VECTOR(7 downto 0) := x"23";
00062 constant OUTPUT_B2                     : STD_LOGIC_VECTOR(7 downto 0) := x"24";
00063 constant OUTPUT_B1                 : STD_LOGIC_VECTOR(7 downto 0) := x"25";
```

```
00064 constant OUTPUT_B0                    : STD_LOGIC_VECTOR(7 downto 0) := x"26";
00065 constant TX_COMPLETE                : STD_LOGIC_VECTOR(7 downto 0) := x"FF";
00066
00067 constant INIT_FLAG                 : integer := 32;
00068
00069 constant B31_FLAG                      : integer := 31;
00070 constant B30_FLAG                      : integer := 30;
00071 constant B29_FLAG                      : integer := 29;
00072 constant B28_FLAG                      : integer := 28;
00073 constant B27_FLAG                      : integer := 27;
00074 constant B26_FLAG                      : integer := 26;
00075 constant B25_FLAG                      : integer := 25;
00076 constant B24_FLAG                      : integer := 24;
00077 constant B23_FLAG                      : integer := 23;
00078 constant B22_FLAG                      : integer := 22;
00079 constant B21_FLAG                      : integer := 21;
00080 constant B20_FLAG                      : integer := 20;
00081 constant B19_FLAG                      : integer := 19;
00082 constant B18_FLAG                      : integer := 18;
00083 constant B17_FLAG                      : integer := 17;
00084 constant B16_FLAG                      : integer := 16;
00085 constant B15_FLAG                      : integer := 15;
00086 constant B14_FLAG                      : integer := 14;
00087 constant B13_FLAG                      : integer := 13;
00088 constant B12_FLAG                      : integer := 12;
00089 constant B11_FLAG                      : integer := 11;
00090 constant B10_FLAG                      : integer := 10;
00091 constant B9_FLAG                     : integer := 9;
00092 constant B8_FLAG                     : integer := 8;
00093 constant B7_FLAG                     : integer := 7;
00094 constant B6_FLAG                     : integer := 6;
00095 constant B5_FLAG                     : integer := 5;
00096 constant B4_FLAG                     : integer := 4;
00097 constant B3_FLAG                     : integer := 3;
00098 constant B2_FLAG                     : integer := 2;
00099 constant B1_FLAG                     : integer := 1;
00100 constant B0_FLAG                     : integer := 0;
00101
00102 --constant OUTPUT_FLAG                   : integer := 1;
00103 constant DONE_FLAG                 : integer := 33;
00104
00105 --type inter_3l is array (0 to 30) of STD_LOGIC_VECTOR(15 downto 0);
00106 --constant inter_3l_array : inter_3l := (x"7303", x"6277", x"396D", x"0C47", x"EECE",
00107 --                                                  x"E926", x"F521", x"04BB", x"0C3A", x"08D3",
00108 --                                                  x"0000", x"F98C", x"F97E", x"FE30", x"02F4",
00109 --                                                  x"044B", x"0226", x"FF0B", x"FD86", x"FE3D",
00110 --                                                  x"0000", x"0134", x"0128", x"004E", x"FF88",
00111 --                                                  x"FF5B", x"FFB1", x"0022", x"005B", x"0046",
00112 --                                                  x"0000");
00113
00114 end SPI_pkg;
00115
00116
00117 package body SPI_pkg is
00118
00119 end SPI_pkg;
00120
```

Synch_Slave_FIFO.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
00005 -- Create Date:    09:58:29 01/13/2012
00006 -- Design Name:
00007 -- Module Name:    Synch_Slave_FIFO - Behavioral
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- Dependencies:
00014 --
00015 -- Revision:
```

```
00016 -- Revision 0.01 - File Created
00017 -- Additional Comments:
00018 --
00019 ----------------------------------------------------------------------------------
00020 library IEEE;
00021 use IEEE.STD_LOGIC_1164.ALL;
00022 use IEEE.numeric_std.all;
00023 --use IEEE.STD_LOGIC_ARITH.ALL;
00024 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00025
00026
00027 use work.Synch_Slave_FIFO_pkg.all;
00028
00029 entity Synch_Slave_FIFO is
00030     Port ( Clk                  : in   STD_LOGIC;
00031            reset                : in   STD_LOGIC;                                -- Module
      reset (active low)
00032            FIFOADDR_in          : in   STD_LOGIC_VECTOR (1 downto 0);       -- Selects
      endpoint
00033
00034            -- Cypress Synchronous Slave FIFO IO
00035            Data                 : out  STD_LOGIC_VECTOR (7 downto 0);       -- Parallel data bus
00036            PktEnd               : out  STD_LOGIC;
00037            FlagB                : in   STD_LOGIC;                                -- Indicates the
      Cypress FIFOs are full (active low)
00038            SLRD                 : out  STD_LOGIC;
00039            SLWR                 : out  STD_LOGIC;
00040            SLOE                 : out  STD_LOGIC;
00041            FIFOADDR             : out  STD_LOGIC_VECTOR (1 downto 0);
00042
00043            -- USB_FIFO signals
00044            FIFO_DOUT              : in   STD_LOGIC_VECTOR (7 downto 0);
00045            FIFO_RD_CLK            : out  STD_LOGIC;
00046            FIFO_RD_EN           : out  STD_LOGIC;
00047            FIFO_EMPTY           : in   STD_LOGIC;
00048            FIFO_ALMOST_EMPTY    : in   STD_LOGIC;
00049            FIFO_PROG_EMPTY        : in   STD_LOGIC;
00050
00051            -- Debug Outputs
00052            FlagB_out            : out  STD_LOGIC;                                -- Debug output of
      Cypress FIFO full flag
00053            idle_out             : out  STD_LOGIC
00054            );
00055 end Synch_Slave_FIFO;
00056
00057 ----------------------------------------------------------------------------------
00058 -- Synch_Slave_FIFO Behavioral
00059 ----------------------------------------------------------------------------------
00060 architecture Behavioral of Synch_Slave_FIFO is
00061
00062     signal async_flags                  : STD_LOGIC_VECTOR(5 downto 0);
00063     signal count                          : STD_LOGIC_VECTOR(9 downto 0);
00064
00065     ----------------------------------------------------------------------------------
00066     -- Synch_Slave_FIFO_states Component
00067     ----------------------------------------------------------------------------------
00068     component Synch_Slave_FIFO_states
00069         Port (
00070             clk                         : in  STD_LOGIC;
00071             rst_n                     : in  STD_LOGIC;
00072             USB_Full_Flag             : in  STD_LOGIC;
00073             FIFO_EMPTY                  : in  STD_LOGIC;
00074             FIFO_ALMOST_EMPTY         : in  STD_LOGIC;
00075             FIFO_PROG_EMPTY         : in  STD_LOGIC;
00076             count                         : in  STD_LOGIC_VECTOR(9 downto 0);
00077             async_flags                 : out STD_LOGIC_VECTOR(5 downto 0)          --flags to
      enable functions
00078             );
00079     end component;
00080
00081 begin
00082
00083
00084 ----------------------------------------------------------------------------------
00085 -- Cypress Synch Slave FIFO IO
00086 ----------------------------------------------------------------------------------
00087     SLOE            <= '1';
00088     FIFOADDR    <= FIFOADDR_in;
00089     SLRD            <= '1';
```

```
00090     Data            <= FIFO_DOUT;
00091     PktEnd        <= '1';
00092
00093     -- SLWR Process
00094     process(Clk, reset)
00095     begin
00096         if reset = '0' then
00097             SLWR <= '1';
00098         elsif rising_edge(Clk) then
00099             if async_flags(SET_SLWR_FLAG) = '1' then
00100                 SLWR <= '0';
00101             else
00102                 SLWR <= '1';
00103             end if;
00104         end if;
00105     end process;
00106
00107 ---------------------------------------------------------------------------
00108 -- USB FIFO IO
00109 ---------------------------------------------------------------------------
00110     FIFO_RD_CLK <= Clk;
00111
00112     -- FIFO_RD_EN Process
00113     process(Clk, reset)
00114     begin
00115         if reset = '0' then
00116             FIFO_RD_EN <= '0';
00117         elsif rising_edge(Clk) then
00118             if async_flags(SET_RD_EN_FLAG) = '1' then
00119                 FIFO_RD_EN <= '1';
00120             else
00121                 FIFO_RD_EN <= '0';
00122             end if;
00123         end if;
00124     end process;
00125
00126
00127 ---------------------------------------------------------------------------
00128 -- Debug IO
00129 ---------------------------------------------------------------------------
00130     FlagB_out    <= FlagB;
00131     idle_out     <= async_flags(IDLE_FLAG);
00132
00133
00134 ---------------------------------------------------------------------------
00135 ---------------------- Synch_Slave_FIFO_states -----------------------------
00136 ---------------------------------------------------------------------------
00137
00138 states : Synch_Slave_FIFO_states
00139 port map(
00140     clk                             => Clk,
00141     rst_n                         => reset,
00142     USB_Full_Flag                 => FlagB,
00143     FIFO_EMPTY                       => FIFO_EMPTY,
00144     FIFO_ALMOST_EMPTY             => FIFO_ALMOST_EMPTY,
00145     FIFO_PROG_EMPTY                 => FIFO_PROG_EMPTY,
00146     count                             => count,
00147     async_flags                       => async_flags
00148     );
00149
00150
00151 ---------------------------------------------------------------------------
00152 ---------------------- Counter ------------------------------------------
00153 ---------------------------------------------------------------------------
00154
00155 -- count
00156 process(Clk, reset)
00157 begin
00158    if reset = '0' then
00159         count <= (others => '0');
00160    elsif rising_edge(Clk) then
00161         if async_flags(INC_COUNT_FLAG) = '1' then
00162             count <= count + 1;
00163         elsif async_flags(IDLE_FLAG) = '1' then
00164             count <= "00" & x"00";
00165         end if;
00166     end if;
00167 end process;
00168
```

```
00169 end Behavioral;
00170
```

Synch_Slave_FIFO_main_states.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 --
00019 -- Additional Comments:
00020 --
00021 ----------------------------------------------------------------------------------
00022 library IEEE;
00023 use IEEE.STD_LOGIC_1164.ALL;
00024 use IEEE.numeric_std.all;
00025 --use IEEE.STD_LOGIC_ARITH.ALL;
00026 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00027
00028 use work.Synch_Slave_FIFO_pkg.all;
00029
00030 ---- Uncomment the following library declaration if instantiating
00031 ---- any Xilinx primitives in this code.
00032 library UNISIM;
00033 use UNISIM.VComponents.all;
00034
00035 entity Synch_Slave_FIFO_states is
00036 Port (  clk                         : in  STD_LOGIC;
00037             rst_n                     : in  STD_LOGIC;
00038             --Synch_Slave_FIFO_start    : in  STD_LOGIC;            --start S.M. into motion
00039             USB_Full_Flag             : in  STD_LOGIC;
00040             FIFO_EMPTY                   : in  STD_LOGIC;
00041             FIFO_ALMOST_EMPTY         : in  STD_LOGIC;
00042             FIFO_PROG_EMPTY           : in  STD_LOGIC;
00043             count                        : in  STD_LOGIC_VECTOR(9 downto 0);
00044             async_flags                : out STD_LOGIC_VECTOR(5 downto 0)  --flags to enable
      functions
00045             );
00046 end Synch_Slave_FIFO_states;
00047
00048 architecture Behavioral of Synch_Slave_FIFO_states is
00049
00050         --Control signals
00051
00052 signal curr_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM current state
00053 signal next_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM next state
00054
00055 begin
00056 ----------------------------------------------------------------------------------
00057         -- synchronous part of state machine here
00058 data_in_latch: process(clk, rst_n)
00059 begin
00060    if rst_n = '0' then
00061         curr_state <= (others => '0');
00062    elsif rising_edge(clk) then
00063         curr_state <= next_state;
00064    end if;
00065 end process;
00066
00067         -- async part of state machine to set function flags
00068 Synch_Slave_FIFO_state: process(rst_n, curr_state)
00069 begin
```

```
00070    if rst_n = '0' then
00071         async_flags <= (others => '0');
00072    else
00073         async_flags <= (others => '0');
00074         case curr_state is
00075             when IDLE =>
00076                 async_flags(IDLE_FLAG) <= '1';           -- init
00077
00078             when USB_1 =>
00079                 async_flags(SET_RD_EN_FLAG) <= '1';
00080
00081
00082             when USB_2 =>
00083                 if count = 15 then
00084                     --async_flags(SET_RD_EN_FLAG) <= '1';
00085                     async_flags(SET_SLWR_FLAG) <= '1';
00086                     async_flags(SET_DATA_FLAG) <= '1';
00087                     async_flags(INC_COUNT_FLAG) <= '1';
00088 --              elsif count = 14 then
00089 --                  --async_flags(SET_RD_EN_FLAG) <= '1';
00090 --                  async_flags(SET_SLWR_FLAG) <= '1';
00091 --                  async_flags(SET_DATA_FLAG) <= '1';
00092 --                  async_flags(INC_COUNT_FLAG) <= '1';
00093                 else
00094                     async_flags(SET_RD_EN_FLAG) <= '1';
00095                     async_flags(SET_SLWR_FLAG) <= '1';
00096                     async_flags(SET_DATA_FLAG) <= '1';
00097                     async_flags(INC_COUNT_FLAG) <= '1';
00098                 end if;
00099
00100
00101
00102
00103             when FINISH =>
00104                 async_flags(DONE_FLAG) <= '1';           -- done flag
00105             when others =>
00106                 async_flags <= (others => '0');
00107         end case;
00108     end if;
00109 end process;
00110 -------------------------------------------------
00111         -- Synch_Slave_FIFO state machine
00112 Synch_Slave_FIFO_asynch_state: process(rst_n,
      curr_state, count, USB_Full_Flag, FIFO_EMPTY,
      FIFO_ALMOST_EMPTY, FIFO_PROG_EMPTY)
00113 begin
00114    if rst_n = '0' then
00115         next_state <= (others => '0');
00116    else
00117         case curr_state is
00118         when IDLE =>
00119                 if USB_Full_Flag = '1' and FIFO_PROG_EMPTY = '0' then
00120                     next_state <= USB_1;
00121                 else
00122                     next_state <= IDLE;
00123                 end if;
00124
00125             when USB_1 =>
00126                 next_state <= USB_2;
00127
00128             when USB_2 =>
00129                 if count = 15 then
00130                     next_state <= FINISH;
00131                 else
00132                     next_state <= USB_2;
00133                 end if;
00134
00135             when FINISH =>
00136                 next_state <= IDLE;
00137             when OTHERS =>
00138                 next_state <= IDLE;
00139         end case;
00140    end if;
00141 end process;
00142
00143
00144 end Behavioral;
00145
```

Synch_Slave_FIFO_pkg.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 --
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 --
00019 -- Additional Comments:
00020 --
00021 ----------------------------------------------------------------------------------
00022 library IEEE;
00023 use IEEE.STD_LOGIC_1164.ALL;
00024 use IEEE.numeric_std.all;
00025 --use IEEE.STD_LOGIC_ARITH.ALL;
00026 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00027
00028 package Synch_Slave_FIFO_pkg is
00029
00030
00031 constant IDLE                                    : STD_LOGIC_VECTOR(7 downto 0) := x"00";
00032
00033 constant USB_1                                   : STD_LOGIC_VECTOR(7 downto 0) := x"10";
00034 constant USB_2                                   : STD_LOGIC_VECTOR(7 downto 0) := x"11";
00035
00036
00037
00038
00039
00040 constant FINISH                                  : STD_LOGIC_VECTOR(7 downto 0) := x"FF";
00041
00042
00043
00044 constant IDLE_FLAG                           : integer := 0;
00045
00046 constant SET_SLWR_FLAG                       : integer := 1;
00047 constant SET_DATA_FLAG                       : integer := 2;
00048 constant SET_RD_EN_FLAG                         : integer := 3;
00049 constant INC_COUNT_FLAG                         : integer := 4;
00050
00051 constant DONE_FLAG                           : integer := 5;
00052
00053
00054
00055 end Synch_Slave_FIFO_pkg;
00056
00057 package body Synch_Slave_FIFO_pkg is
00058
00059 end Synch_Slave_FIFO_pkg;
00060
00061
```

TX_Module.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:
00004 --
00005 -- Create Date:    19:44:35 01/27/2012
00006 -- Design Name:
00007 -- Module Name:    ADC_Module - Behavioral
00008 -- Project Name:
```

```
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
00013 -- Dependencies:
00014 --
00015 -- Revision:
00016 -- Revision 0.01 - File Created
00017 -- Additional Comments:
00018 --
00019 ----------------------------------------------------------------------------------
00020 library IEEE;
00021 use IEEE.STD_LOGIC_1164.ALL;
00022 use IEEE.STD_LOGIC_ARITH.ALL;
00023 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00024
00025 ---- Uncomment the following library declaration if instantiating
00026 ---- any Xilinx primitives in this code.
00027 --library UNISIM;
00028 --use UNISIM.VComponents.all;
00029
00030 use work.TX_Module_pkg.all;
00031
00032 entity TX_Module is
00033     Port ( clk                    : in  STD_LOGIC;
00034            reset                  : in  STD_LOGIC;
00035            TX                     : out STD_LOGIC;
00036            TX_led                 : out STD_LOGIC;
00037
00038            -- TX_FIFO Signals
00039            FIFO_DOUT              : in  STD_LOGIC_VECTOR (7 downto 0);
00040            FIFO_RD_EN             : out STD_LOGIC;
00041            FIFO_EMPTY             : in  STD_LOGIC
00042            );
00043 end TX_Module;
00044
00045 architecture Behavioral of TX_Module is
00046
00047 signal async_flags                 : STD_LOGIC_VECTOR(5 downto 0);
00048 signal count                       : STD_LOGIC_VECTOR(7 downto 0);
00049
00050
00051
00052 component TX_Module_states
00053 Port (  clk                        : in  STD_LOGIC;
00054            rst_n                  : in  STD_LOGIC;
00055            FIFO_EMPTY             : in  STD_LOGIC;
00056            count                  : in  STD_LOGIC_VECTOR(7 downto 0);
00057            async_flags            : out STD_LOGIC_VECTOR(5 downto 0)  --flags to enable
     functions
00058            );
00059 end component;
00060
00061 begin
00062
00063 TX_led <= not async_flags(IDLE_FLAG);
00064
00065 -- TX
00066 process(clk, reset)
00067 begin
00068    if reset = '0' then
00069         TX <= '1';
00070    elsif rising_edge(clk) then
00071         if async_flags(START_BIT_FLAG) = '1' then
00072             TX <='0';
00073         elsif async_flags(SEND_BYTE_FLAG) = '1' then
00074             case count is
00075                 when x"00" =>               TX <= FIFO_DOUT(0);
00076                 when x"01" =>               TX <= FIFO_DOUT(1);
00077                 when x"02" =>               TX <= FIFO_DOUT(2);
00078                 when x"03" =>               TX <= FIFO_DOUT(3);
00079                 when x"04" =>               TX <= FIFO_DOUT(4);
00080                 when x"05" =>               TX <= FIFO_DOUT(5);
00081                 when x"06" =>               TX <= FIFO_DOUT(6);
00082                 when x"07" =>               TX <= FIFO_DOUT(7);
00083                 when OTHERS =>
00084             end case;
00085         elsif async_flags(END_BIT_FLAG) = '1' then
00086             TX <= '1';
```

```
00087         else
00088             TX <= '1';
00089         end if;
00090     end if;
00091 end process;
00092 
00093 -- FIFO_RD_EN
00094 process(clk, reset)
00095 begin
00096    if reset = '0' then
00097         FIFO_RD_EN <= '0';
00098    elsif rising_edge(clk) then
00099         if async_flags(RD_EN_FLAG) = '1' then
00100             FIFO_RD_EN <= '1';
00101         else
00102             FIFO_RD_EN <= '0';
00103         end if;
00104     end if;
00105 end process;
00106 
00107 
00108 
00109 
00110 
00111 
00112 ---------------------------------------------------------------------------------
00113 ---------------------- TX_Module_states -----------------------------------------
00114 ---------------------------------------------------------------------------------
00115 
00116 states : TX_Module_states
00117 port map(
00118     clk                             => clk,
00119     rst_n                          => reset,
00120     FIFO_EMPTY                        => FIFO_EMPTY,
00121     count                              => count,
00122     async_flags                       => async_flags
00123     );
00124 
00125 ---------------------------------------------------------------------------------
00126 ---------------------- Counter --------------------------------------------------
00127 ---------------------------------------------------------------------------------
00128 
00129 -- count
00130 process(clk, reset)
00131 begin
00132    if reset = '0' then
00133         count <= (others => '0');
00134    elsif rising_edge(clk) then
00135         if async_flags(INC_COUNT_FLAG) = '1' then
00136             count <= count + 1;
00137         elsif async_flags(IDLE_FLAG) = '1' then
00138             count <= x"00";
00139         elsif async_flags(END_BIT_FLAG) = '1' then
00140             count <= x"00";
00141         end if;
00142     end if;
00143 end process;
00144 
00145 
00146 end Behavioral;
00147 
```

TX_Module_main_states.vhd

```
00001 ---------------------------------------------------------------------------------
00002 -- Company:
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
```

```
00012 --
00013 --
00014 --
00015 -- Dependencies:
00016 --
00017 -- Revision:
00018 --
00019 -- Additional Comments:
00020 --
00021 ----------------------------------------------------------------------------------
00022 library IEEE;
00023 use IEEE.STD_LOGIC_1164.ALL;
00024 use IEEE.numeric_std.all;
00025 --use IEEE.STD_LOGIC_ARITH.ALL;
00026 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00027
00028 use work.TX_Module_pkg.all;
00029
00030 ---- Uncomment the following library declaration if instantiating
00031 ---- any Xilinx primitives in this code.
00032 library UNISIM;
00033 use UNISIM.VComponents.all;
00034
00035 entity TX_Module_states is
00036 Port (  clk                         : in  STD_LOGIC;
00037             rst_n                     : in  STD_LOGIC;
00038             FIFO_EMPTY                   : in  STD_LOGIC;
00039             count                         : in  STD_LOGIC_VECTOR(7 downto 0);
00040             async_flags                 : out STD_LOGIC_VECTOR(5 downto 0)  --flags to enable
      functions
00041             );
00042 end TX_Module_states;
00043
00044 architecture Behavioral of TX_Module_states is
00045
00046         --Control signals
00047
00048 signal curr_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM current state
00049 signal next_state     : std_logic_vector(7 downto 0) := IDLE;  -- FSM next state
00050
00051 begin
00052 ----------------------------------------------------------------------------
00053         -- synchronous part of state machine here
00054 data_in_latch: process(clk, rst_n)
00055 begin
00056    if rst_n = '0' then
00057         curr_state <= (others => '0');
00058    elsif rising_edge(clk) then
00059         curr_state <= next_state;
00060    end if;
00061 end process;
00062
00063         -- async part of state machine to set function flags
00064 TX_Module_state: process(rst_n, curr_state)
00065 begin
00066    if rst_n = '0' then
00067         async_flags <= (others => '0');
00068    else
00069         async_flags <= (others => '0');
00070         case curr_state is
00071
00072             when IDLE =>
00073                 async_flags(IDLE_FLAG) <= '1';           -- init
00074
00075             when START_BIT =>
00076                 async_flags(START_BIT_FLAG) <= '1';
00077
00078             when SEND_BYTE =>
00079                 async_flags(SEND_BYTE_FLAG) <= '1';
00080                 async_flags(INC_COUNT_FLAG) <= '1';
00081
00082             when END_BIT =>
00083                 async_flags(END_BIT_FLAG) <= '1';
00084
00085             when INC_FIFO =>
00086                 async_flags(RD_EN_FLAG) <= '1';
00087
00088             when INTER_BYTE_DELAY =>
00089                 async_flags(INC_COUNT_FLAG) <= '1';
```

```
00090
00091             when FINISH =>
00092                 --async_flags(DONE_FLAG) <= '1';            -- done flag
00093             when others =>
00094                 async_flags <= (others => '0');
00095         end case;
00096     end if;
00097 end process;
00098 ------------------------------------------------
00099         -- TX_Module state machine
00100 TX_Module_asynch_state: process(rst_n, curr_state,
      FIFO_EMPTY, count)
00101 begin
00102    if rst_n = '0' then
00103         next_state <= IDLE;
00104    else
00105         case curr_state is
00106         when IDLE =>
00107                 if FIFO_EMPTY = '0' then
00108                     next_state <= START_BIT;
00109                 else
00110                     next_state <= IDLE;
00111                 end if;
00112
00113             when START_BIT =>
00114                 next_state <= SEND_BYTE;
00115
00116             when SEND_BYTE =>
00117                 if count = 7 then
00118                     next_state <= END_BIT;
00119                 else
00120                     next_state <= SEND_BYTE;
00121                 end if;
00122
00123             when END_BIT =>
00124                 next_state <= INC_FIFO;
00125
00126             when INC_FIFO =>
00127                 next_state <= INTER_BYTE_DELAY;
00128
00129             when INTER_BYTE_DELAY =>
00130                 if count = 10 then
00131                     next_state <= FINISH;
00132                 else
00133                     next_state <= INTER_BYTE_DELAY;
00134                 end if;
00135
00136             when FINISH =>
00137                     next_state <= IDLE;
00138
00139             when OTHERS =>
00140                 next_state <= IDLE;
00141         end case;
00142    end if;
00143 end process;
00144
00145
00146 end Behavioral;
00147
00148
```

TX_Module_pkg.vhd

```
00001 ----------------------------------------------------------------------------------
00002 -- Company:     WMU - Thesis
00003 -- Engineer:    KDB
00004 --
00005 -- Create Date:
00006 -- Design Name:
00007 -- Module Name:
00008 -- Project Name:
00009 -- Target Devices:
00010 -- Tool versions:
00011 -- Description:
00012 --
```

```
00013 --
00014 --
00015 --
00016 -- Dependencies:
00017 --
00018 -- Revision:
00019 -- Revision 0.01 - File Created
00020 -- Additional Comments:
00021 --
00022 ----------------------------------------------------------------------------------
00023 library IEEE;
00024 use IEEE.STD_LOGIC_1164.ALL;
00025 use IEEE.numeric_std.all;
00026 --use IEEE.STD_LOGIC_ARITH.ALL;
00027 use IEEE.STD_LOGIC_UNSIGNED.ALL;
00028
00029 package TX_Module_pkg is
00030
00031
00032 constant IDLE                                   : STD_LOGIC_VECTOR(7 downto 0) := x"00";
00033
00034 constant START_BIT                             : STD_LOGIC_VECTOR(7 downto 0) := x"01";
00035 constant SEND_BYTE                                : STD_LOGIC_VECTOR(7 downto 0) := x"02";
00036 constant END_BIT                                  : STD_LOGIC_VECTOR(7 downto 0) := x"03";
00037 constant INC_FIFO                                 : STD_LOGIC_VECTOR(7 downto 0) := x"04";
00038 constant INTER_BYTE_DELAY                       : STD_LOGIC_VECTOR(7 downto 0) := x"05";
00039
00040 constant FINISH                                   : STD_LOGIC_VECTOR(7 downto 0) := x"FF";
00041
00042
00043
00044 constant IDLE_FLAG                              : integer := 0;
00045
00046 constant START_BIT_FLAG                           : integer := 1;
00047 constant SEND_BYTE_FLAG                           : integer := 2;
00048 constant INC_COUNT_FLAG                           : integer := 3;
00049 constant END_BIT_FLAG                            : integer := 4;
00050 constant RD_EN_FLAG                               : integer := 5;
00051
00052 --constant DONE_FLAG                                : integer := 6;
00053
00054
00055
00056 end TX_Module_pkg;
00057
00058 package body TX_Module_pkg is
00059
00060 end TX_Module_pkg;
00061
00062
```

UART_behavioural_model.vhd

```
00001 ---------------------------------------------------- package ---
00002
00003 library IEEE;
00004 use IEEE.std_logic_1164.all;
00005
00006 package UART_behavioural_model is
00007
00008
00009 -- The signal that is to be driven by this model...
00010 -- Inputs to control how to send one character:procedure UART_tx (
00011
00012 signal tx_line: out std_logic;
00013
00014 data: in std_logic_vector; -- usually 8 bits
00015 baud_rate:in integer -- e.g. 9600
00016 );
00017
00018 end package UART_behavioural_model;
00019
00020 ---------------------------------------------------- package body ---
00021
00022 package body UART_behavioural_model is
```

```
00023
00024
00025 -- The signal that is to be driven by this model...
00026 -- Inputs to control how to send one character:procedure UART_tx (
00027
00028 signal tx_line: out std_logic;
00029
00030 data: in std_logic_vector; -- usually 8 bits
00031 baud_rate:in integer -- e.g. 9600
00032 ) is
00033
00034 constant bit_time: time := 1 sec / baud_rate;
00035
00036 begin
00037
00038 -- Send the start bit
00039 tx_line <= '0';
00040 wait for bit_time;
00041
00042 -- Send the data bits, least significant first
00043 for i in data'reverse_range loop
00044 tx_line <= data(i);
00045 wait for bit_time;
00046 end loop;
00047
00048 -- Send the stop bit
00049 tx_line <= '1';
00050 wait for bit_time;
00051
00052 end; -- procedure UART_tx
00053
00054 end package body UART_behavioural_model;
```

Appendix H

Data Acquisition and Stimulation – Code

Channels.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ComponentModel;

namespace Data_Acq_and_Stim_Control_Center
{
 /*********************************************************************************************
 * Channels class
 *
 * contains configuration information for channels
/*********************************************************************************************/
    public class Channels : INotifyPropertyChanged
    {
        private string _channel;
        private string _waveform_file;
        private string _mode;
        private string _sw1;
        private string _sw2;
        private string _sw3;
        private string _sw4;

        public event PropertyChangedEventHandler PropertyChanged;

        public Channels(string channel, string waveform_file, string mode)
        {
            _channel = channel;
            _waveform_file = waveform_file;
            _mode = mode;
            if (mode == "Stimulation")
            {
                _sw1 = "On";
                _sw2 = "On";
                _sw3 = "On";
                _sw4 = "On";
            }
            else
            {
                _sw1 = "Off";
                _sw2 = "Off";
                _sw3 = "Off";
                _sw4 = "Off";
            }
        }

        public string channel
        {
            get { return _channel; }
            set
            {
                _channel = value;
                this.NotifyPropertyChanged("channel");
            }
        }

        public string waveform_file
        {
            get { return _waveform_file; }
            set
            {
                _waveform_file = value;
                this.NotifyPropertyChanged("waveform_file");
            }
        }

        public string mode
        {
            get { return _mode; }
            set
            {
                _mode = value;
                this.NotifyPropertyChanged("mode");
            }
        }
```

```
        public string sw1
        {
            get { return _sw1; }
            set
            {
                _sw1 = value;
                this.NotifyPropertyChanged("sw1");
            }
        }

        public string sw2
        {
            get { return _sw2; }
            set
            {
                _sw2 = value;
                this.NotifyPropertyChanged("sw2");
            }
        }

        public string sw3
        {
            get { return _sw3; }
            set
            {
                _sw3 = value;
                this.NotifyPropertyChanged("sw3");
            }
        }

        public string sw4
        {
            get { return _sw4; }
            set
            {
                _sw4 = value;
                this.NotifyPropertyChanged("sw4");
            }
        }

        private void NotifyPropertyChanged(string name)
        {
            if (PropertyChanged != null)
                PropertyChanged(this, new PropertyChangedEventArgs(name));
        }
    }
}
```

CypressDataAcq.cs

This code was derived from Cypress Streamer 3.4.7.0 available in CySuiteUSB.

FPGA_Commands.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.IO;

namespace Data_Acq_and_Stim_Control_Center
{
    public class FPGA_Commands
    {
        public RS232_Communication RS232_Com;


        public FPGA_Commands()
        {
            RS232_Com = new RS232_Communication();
        }

        public void FPGA_SetConfig(Byte Channel, Byte Config)
        {
            byte length_h = 0x00;
            byte length_l = 0x07;
            byte checksum = 0x00;

            // Build Message
            byte[] msg = new byte[] { 0x5A, 0x01, length_h, length_l, Channel, Config, checksum };

            // Calculate Checksum
            for (int j = 0; j < msg.Length - 1; j++)
            {
                checksum += msg[j];
            }
            msg[msg.Length - 1] = checksum;

            // Send Message
            RS232_Com.SendData(msg);
        }

        public void FPGA_GetConfig(Byte Channel)
        {
            Byte length_h = 0x00;
            Byte length_l = 0x06;
            Byte checksum = 0x00;

            // Build Message
            byte[] msg = new byte[] { 0x5A, 0x02, length_h, length_l, Channel, checksum };

            // Calculate Checksum
            for (int j = 0; j < msg.Length - 1; j++)
            {
                checksum += msg[j];
            }
            msg[msg.Length - 1] = checksum;

            // Send Message
            RS232_Com.SendData(msg);

        }

        public void FPGA_SetWaveform(Byte Channel, string Filename)
        {
            List<Byte> wave_data = new List<Byte>();

            try
            {
                using (StreamReader sr = new StreamReader(Filename))
                {
                    String line;
                    String[] split_str = new String[2];

                    while ((line = sr.ReadLine()) != null)
                    {
                        //this.Dispatcher.Invoke(DispatcherPriority.Normal, (Action)(() =>
                        //{
```

```
                //    textBox1.AppendText(line + "\r\n");
                //}));

                split_str = line.Split(',');
                string temp1 = split_str[0].Substring(0, 2);
                string temp2 = split_str[0].Substring(2, 2);
                string temp3 = split_str[1].Substring(0, 2);
                string temp4 = split_str[1].Substring(2, 2);
                wave_data.Add(Convert.ToByte((ToNibble(temp1[0]) << 4) + ToNibble(temp1[1])));
                wave_data.Add(Convert.ToByte((ToNibble(temp2[0]) << 4) + ToNibble(temp2[1])));
                wave_data.Add(Convert.ToByte((ToNibble(temp3[0]) << 4) + ToNibble(temp3[1])));
                wave_data.Add(Convert.ToByte((ToNibble(temp4[0]) << 4) + ToNibble(temp4[1])));

                //port.WriteLine();
            }
        }
        byte[] newarray = wave_data.ToArray();

        byte[] msg_buf = new byte[7 + wave_data.Count];
        msg_buf[0] = 0x5A;                                          // Start Byte
        msg_buf[1] = 0x05;                                          // MSG_ID
        msg_buf[2] = 0x00;                                          // Length_High
        msg_buf[3] = Convert.ToByte(7 + wave_data.Count);           // Length_Low
        msg_buf[4] = Channel;                                       // Channel
        msg_buf[5] = Convert.ToByte(wave_data.Count / 4);           // Samples
        newarray.CopyTo(msg_buf, 6);
        msg_buf[msg_buf[3] - 1] = 0xFF;

        // Send Message
        RS232_Com.SendData(msg_buf);
    }
    catch { }
}

public void FPGA_GetWaveform(Byte Channel)
{
    byte length_h = 0x00;
    byte length_l = 0x06;
    //byte channel = Convert.ToByte(setWaveChan_CB.SelectedIndex + 1);
    byte checksum = 0x00;

    // Build Message
    byte[] msg_buf = new byte[] { 0x5A, 0x06, length_h, length_l, Channel, checksum };

    // Calculate Checksum
    for (int j = 0; j < msg_buf.Length - 1; j++)
    {
        checksum += msg_buf[j];
    }
    msg_buf[msg_buf.Length - 1] = checksum;

    // Send Message
    RS232_Com.SendData(msg_buf);
}

public void FPGA_StartAcquisition()
{
    // Adding call to start cypress acquisition
    //MainWindow.CypressDA.Start_Cypress_Acq();
    //Start_Cypress_Acq();

    // Temporary Start Acq
    byte[] temp_start_msg = new byte[] { 0x5A, 0x01, 0x00, 0x07, 0x01, 0x9F, 0xFF };

    // Send command
    RS232_Com.SendData(temp_start_msg);

}

public void FPGA_EndAcquisition()
{
    // Adding call to stop cypress acquisition
    //MainWindow.CypressDA.Stop_Cypress_Acq();
    //Stop_Cypress_Acq();

    // Temporary Start Acq
    byte[] temp_end_msg = new byte[] { 0x5A, 0x01, 0x00, 0x07, 0x01, 0x1F, 0xFF };

    // Send command
```

```
            RS232_Com.SendData(temp_end_msg);
        }

        public void FPGA_StartMultiStim(Byte Channel)
        {
            byte length_h = 0x00;
            byte length_l = 0x07;
            byte continuous = Channel;
            byte checksum = 0x00;

            // Build Message
            byte[] msg = new byte[] { 0x5A, 0x07, length_h, length_l, Channel, continuous, checksum };

            // Calculate Checksum
            for (int j = 0; j < msg.Length - 1; j++)
            {
                checksum += msg[j];
            }
            msg[msg.Length - 1] = checksum;

            // Send Message
            RS232_Com.SendData(msg);
        }

        public void FPGA_EndMuliStim()
        {
            byte length_h = 0x00;
            byte length_l = 0x07;
            byte channel = 0x00;
            byte continuous = 0x00;
            byte checksum = 0x00;

            // Build Message
            byte[] msg = new byte[] { 0x5A, 0x07, length_h, length_l, channel, continuous, checksum };

            // Calculate Checksum
            for (int j = 0; j < msg.Length - 1; j++)
            {
                checksum += msg[j];
            }
            msg[msg.Length - 1] = checksum;

            // Send Message
            RS232_Com.SendData(msg);
        }

        public void FPGA_SingleStim(Byte Channel)
        {
            byte length_h = 0x00;
            byte length_l = 0x07;
            byte continuous = 0x00;
            byte checksum = 0x00;

            // Build Message
            byte[] msg = new byte[] { 0x5A, 0x07, length_h, length_l, Channel, continuous, checksum };

            // Calculate Checksum
            for (int j = 0; j < msg.Length - 1; j++)
            {
                checksum += msg[j];
            }
            msg[msg.Length - 1] = checksum;

            // Send Message
            RS232_Com.SendData(msg);
        }

        private byte ToNibble(char c)
        {
            if ('0' <= c && c <= '9') { return Convert.ToByte(c - '0'); }
            else if ('a' <= c && c <= 'f') { return Convert.ToByte(c - 'a' + 10); }
            else if ('A' <= c && c <= 'F') { return Convert.ToByte(c - 'A' + 10); }
            else
            {
                throw new ArgumentException(String.Format("Character '{0}' cannot be translated to a hexadecimal
value because it is not one of 0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f,A,B,C,D,E,F", c));
            }
        }
    }
```

```
}
```

Graphing.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.Collections;
using System.Windows;
using System.IO;
using System.ComponentModel;

namespace Data_Acq_and_Stim_Control_Center
{
    public class Graphing : INotifyPropertyChanged
    {
        #region graphing globals

        public List<int> SupportedNumSamples = new List<int>();


        //static int NUM_CHANNELS_GRAPH = 8;
        static int NUM_PACKETS = 65536;
        //static int NUM_SAMPLES_DISPLAYED = 32768 / 4;

        static int previous_view = 0;

        Double time_from_start = 0;
        Double prev_time_calc = 0;

        //GraphingData Channel1_Data = new GraphingData();

        public List<GraphingData> GraphData = new List<GraphingData>();

        #endregion

        private int _SamplesPerView;
        public int SamplesPerView
        {
            get { return _SamplesPerView; }
            set
            {
                _SamplesPerView = value;
                this.NotifyPropertyChanged("SamplesPerView");
            }
        }

        private int _TotalViews;
        public int TotalViews
        {
            get { return _TotalViews; }
            set
            {
                _TotalViews = value;
                this.NotifyPropertyChanged("TotalViews");
            }
        }

        private int _CurrentView;
        public int CurrentView
        {
            get { return _CurrentView; }
            set
            {
                _CurrentView = value;
                this.NotifyPropertyChanged("CurrentView");
            }
        }

        private int _ChannelsToGraph;
        public int ChannelsToGraph
        {
            get { return _ChannelsToGraph; }
            set
            {
                _ChannelsToGraph = value;
                this.NotifyPropertyChanged("ChannelsToGraph");
```

```
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;
    private void NotifyPropertyChanged(string name)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new PropertyChangedEventArgs(name));
    }


    #region graphing code

    public Graphing()
    {
        SupportedNumSamples.Add(512);
        SupportedNumSamples.Add(1024);
        SupportedNumSamples.Add(2048);
        SupportedNumSamples.Add(4096);
        SupportedNumSamples.Add(8192);
        SupportedNumSamples.Add(16384);

    }


    public void LoadFile()
    {
        Stream myStream = null;
        Int16 val;
        UInt32 time_offset;
        //Int16[] channel = new Int16[NUM_CHANNELS];
        Double time_calc;
        int pos = 0;
        string packet_display = "";

        Int32 buffer_loc = 0;

        Double SampleCount = 0;


        // Show the dialog and get result.
        Microsoft.Win32.OpenFileDialog openFileDialog1 = new Microsoft.Win32.OpenFileDialog();

        bool? result = openFileDialog1.ShowDialog();
        if (result == true) // Test result.
        {
            if ((myStream = openFileDialog1.OpenFile()) != null)
            {

                foreach (GraphingData data in GraphData)
                {
                    data.Channel_AllData.Collection.Clear();
                }

                //textBox1.AppendText(openFileDialog1.FileName + " Opened Successfully! File Size: " +
myStream.Length + "\r\n"); // <-- For debugging use only.
                //myStream.Length
                byte[] buffer = new byte[myStream.Length];
                // buffer_loc += 32 * NUM_PACKETS;
                while (buffer_loc < myStream.Length)
                {
                    //myStream.Read(buffer,
                    myStream.Read(buffer, 0, NUM_PACKETS);
                    byte[] packet = new byte[32];

                    while (pos < NUM_PACKETS)
                    {
                        // packet_display = "";
                        Array.Copy(buffer, pos, packet, 0, 32);
                        /*for (int i = 0; i < 32; i++)
                        {
                            packet[i] = buffer[i + pos];

                            //textBox1.AppendText(packet[i].ToString("X") + " ");
                            //packet_display += packet[i].ToString("X") + " ";

                        }*/
```

```
                            val = BitConverter.ToInt16(packet, 0);
                            time_offset = BitConverter.ToUInt32(packet, 2);
                            time_calc = time_offset * .00000002;

                            if (SampleCount == 0)
                            {
                                time_from_start = time_calc;
                                prev_time_calc = time_calc;
                            }
                            else if (time_calc < prev_time_calc)
                            {
                                time_from_start += (4294967296 * .00000002 - prev_time_calc) + time_calc;
                                prev_time_calc = time_calc;
                            }
                            else
                            {
                                time_from_start += time_calc - prev_time_calc;
                                prev_time_calc = time_calc;
                            }

                            //textBox1.AppendText("\r\n\r\nTimeOffset: " + time_calc.ToString("00.000000"));
                            // packet_display += "\r\nTimeOffset: " + time_calc.ToString("00.000000");

                            for (int i = 0; i < ChannelsToGraph; i++)
                            {
                                Int16 raw_voltage = BitConverter.ToInt16(packet, 7 + i * 3);
                                //textBox1.AppendText("\tChannel" + i.ToString() + ": " + (channel[i] / 32768.0
* 5.0).ToString("0.000") + "V\t");
                                // packet_display += "\tChannel" + i.ToString() + ": " + (channel[i] / 32768.0 *
5.0).ToString("0.000") + "V\t";
                                if (raw_voltage == 0) { raw_voltage = 1; }
                                Point p1 = new Point(time_from_start, raw_voltage / 32768.0 * 10.0);
                                GraphData[i].Channel_AllData.Collection.Add(p1);
                            }
                            //textBox1.AppendText(packet_display + "\r\n\r\n");

                            //Point p1 = new Point(time_from_start, channel[0] / 32768.0 * 5.0);
                            //Point p1 = new Point(SampleCount, channel[0] / 32768.0 * 5.0);
                            //Point p1 = new Point(SampleCount, SampleCount);
                            //Channel1_Data.AppendAsync(Dispatcher, p1);
                            //Channel1_Data.Channel_AllData.Collection.Add(p1);
                            //Channel1_Data.Collection.Add(p1);
                            // Thread.Sleep(5);


                            pos += 32;
                            SampleCount++;

                            //plotter.Viewport.FitToView();
                        }
                        buffer_loc += NUM_PACKETS;
                        pos = 0;
                        if (buffer_loc > myStream.Length)
                        {
                        }
                    }



                }
                myStream.Close();
                //System.Text.Encoding.
                /*foreach (Channels temp in Graph_DataGrid.SelectedItems)
                {
                    string channel_name = temp.channel;
                    string[] split = channel_name.Split(' ');
                    int index = Convert.ToInt16(split[1]) - 1;

                    Point[] AllPoints = GraphData[index].Channel_AllData.Collection.ToArray();
                    Point[] GraphPoints = new Point[NUM_SAMPLES_DISPLAYED];
                    Array.Copy(AllPoints, AllPoints.Length - NUM_SAMPLES_DISPLAYED, GraphPoints, 0,
NUM_SAMPLES_DISPLAYED);

                    GraphData[index].Channel_GraphData.Collection.AddMany(GraphPoints);
                }*/
                //plotter.Children.RemoveAll<LineGraph>();

                //foreach (GraphingData data in GraphData)
```

```
                //{
                //    plotter.AddLineGraph(data.Channel_GraphData);
                //data.SetAllPointArray();

                // }
                //Graph_DataGrid.SelectedIndex = 0;

                //chan1Points = Channel1_Data.Channel_AllData.Collection.ToArray();


                // Point[] points = new Point[NUM_SAMPLES_DISPLAYED];
                //Array.Copy(chan1Points, chan1Points.Length - NUM_SAMPLES_DISPLAYED, points, 0,
NUM_SAMPLES_DISPLAYED);


                // Channel1_Data.Channel_GraphData.Collection.AddMany(points);
                //plotter.AddLineGraph(Channel1_Data.Channel_GraphData, 2, "Data row 1");
                //plotter.AddLineGraph(Channel1_Data, 2, "Data row 1");
            }

        }

        //private void Graph_DataGrid_SelectionChanged(object sender, SelectionChangedEventArgs e)
        //{
        //    int selected_view = Convert.ToInt16(slider1.Value);
        //    plotter.Children.RemoveAll<LineGraph>();

        //    foreach (GraphingData data in GraphData)
        //    {
        //        data.Channel_GraphData.Collection.Clear();
        //    }

        //    foreach (Channels temp in Graph_DataGrid.SelectedItems)
        //    {
        //        string channel_name = temp.channel;
        //        string[] split = channel_name.Split(' ');
        //        int index = Convert.ToInt16(split[1]) - 1;

        //        Point[] AllPoints = GraphData[index].Channel_AllData.Collection.ToArray();
        //        Point[] GraphPoints = new Point[NUM_SAMPLES_DISPLAYED];
        //        Array.Copy(AllPoints, selected_view * NUM_SAMPLES_DISPLAYED, GraphPoints, 0,
NUM_SAMPLES_DISPLAYED);

        //        //GraphData[index].Channel_GraphData.Collection.Clear();
        //        GraphData[index].Channel_GraphData.AppendMany(GraphPoints);

        //        plotter.AddLineGraph(GraphData[index].Channel_GraphData, 1, temp.channel);
        //    }
        //    plotter.FitToView();
        //}

        //private void slider1_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
        //{

        //    int selected_view = Convert.ToInt16(slider1.Value);
        //    ViewSelect_TextBox.Text = Convert.ToInt16(slider1.Value).ToString();

        //    if (selected_view != previous_view)
        //    {
        //        // chan1Points = Channel1_Data.Collection.ToArray();
        //        //Point[] points = new Point[NUM_SAMPLES_DISPLAYED];
        //        //Array.Copy(chan1Points, selected_view * NUM_SAMPLES_DISPLAYED, points, 0,
NUM_SAMPLES_DISPLAYED);
        //        //Channel1_Data.Channel_GraphData.Collection.Clear();
        //        //Channel1_Data.Channel_GraphData.AppendMany(points);

        //        foreach (GraphingData data in GraphData)
        //        {
        //            data.Channel_GraphData.Collection.Clear();
        //        }

        //        foreach (Channels temp in Graph_DataGrid.SelectedItems)
        //        {
        //            string channel_name = temp.channel;
        //            string[] split = channel_name.Split(' ');
        //            int index = Convert.ToInt16(split[1]) - 1;

        //            Point[] AllPoints = GraphData[index].Channel_AllData.Collection.ToArray();
        //            Point[] GraphPoints = new Point[NUM_SAMPLES_DISPLAYED];
```

```
        //            Array.Copy(AllPoints, selected_view * NUM_SAMPLES_DISPLAYED, GraphPoints, 0,
NUM_SAMPLES_DISPLAYED);

        //            //GraphData[index].Channel_GraphData.Collection.Clear();
        //            GraphData[index].Channel_GraphData.AppendMany(GraphPoints);
        //        }
        //        plotter.FitToView();
        //    }
        //    previous_view = selected_view;
        //}


        #endregion


    }
}
```

GraphingData.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Research.DynamicDataDisplay.DataSources;
using System.Windows;

namespace Data_Acq_and_Stim_Control_Center
{
    public class GraphingData
    {
        public ObservableDataSource<Point> Channel_AllData = null;
        public ObservableDataSource<Point> Channel_GraphData = null;

        public GraphingData()
        {
            Channel_AllData = new ObservableDataSource<Point>();
            Channel_GraphData = new ObservableDataSource<Point>();
            Channel_GraphData.SetXYMapping(p => p);
        }
    }
}
```

MainWindow.xaml

```
<Window x:Class="Data_Acq_and_Stim_Control_Center.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:col="clr-namespace:System.Collections;assembly=mscorlib"
        xmlns:sys="clr-namespace:System;assembly=mscorlib"
        xmlns:d3="http://research.microsoft.com/DynamicDataDisplay/1.0"
        Title="MainWindow" Height="768" Width="1024" Closing="Window_Closing">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="265" />
            <RowDefinition Height="247*" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="257" />
            <ColumnDefinition Width="497*" />
            <ColumnDefinition Width="307" />
        </Grid.ColumnDefinitions>

        <GroupBox Header="Channel Configuration" HorizontalAlignment="Stretch" Name="groupBox1"
VerticalAlignment="Stretch" Grid.Row="0" Grid.Column="1">
            <DataGrid HorizontalAlignment="Stretch" Name="dataGrid1" VerticalAlignment="Stretch"
AutoGenerateColumns="False" CanUserSortColumns="False" BeginningEdit="dataGrid1_BeginningEdit">
                <DataGrid.Columns>
                    <DataGridTextColumn Header="Channel #" Binding="{Binding Path=channel}"
IsReadOnly="True"></DataGridTextColumn>
                    <DataGridTextColumn Header="Waveform File" Binding="{Binding
Path=waveform_file}"></DataGridTextColumn>
                    <DataGridComboBoxColumn Header="Mode" SelectedItemBinding="{Binding Path=mode}">
                        <DataGridComboBoxColumn.ItemsSource>
                            <col:ArrayList>
                                <sys:String>Acquisition</sys:String>
                                <sys:String>Stimulation</sys:String>
                            </col:ArrayList>
                        </DataGridComboBoxColumn.ItemsSource>
                    </DataGridComboBoxColumn>
                    <DataGridComboBoxColumn Header="Switch 1" SelectedItemBinding="{Binding Path=sw1}" >
                        <DataGridComboBoxColumn.ItemsSource>
                            <col:ArrayList>
                                <sys:String>On</sys:String>
                                <sys:String>Off</sys:String>
                            </col:ArrayList>
                        </DataGridComboBoxColumn.ItemsSource>
                    </DataGridComboBoxColumn>
                    <DataGridComboBoxColumn Header="Switch 2" SelectedItemBinding="{Binding Path=sw2}">
                        <DataGridComboBoxColumn.ItemsSource>
                            <col:ArrayList>
                                <sys:String>On</sys:String>
                                <sys:String>Off</sys:String>
                            </col:ArrayList>
                        </DataGridComboBoxColumn.ItemsSource>
                    </DataGridComboBoxColumn>
                    <DataGridComboBoxColumn Header="Switch 3" SelectedItemBinding="{Binding Path=sw3}">
                        <DataGridComboBoxColumn.ItemsSource>
                            <col:ArrayList>
                                <sys:String>On</sys:String>
                                <sys:String>Off</sys:String>
                            </col:ArrayList>
                        </DataGridComboBoxColumn.ItemsSource>
                    </DataGridComboBoxColumn>
                    <DataGridComboBoxColumn Header="Switch 4" SelectedItemBinding="{Binding Path=sw4}">
                        <DataGridComboBoxColumn.ItemsSource>
                            <col:ArrayList>
                                <sys:String>On</sys:String>
                                <sys:String>Off</sys:String>
                            </col:ArrayList>
                        </DataGridComboBoxColumn.ItemsSource>
                    </DataGridComboBoxColumn>

                </DataGrid.Columns>

            </DataGrid>
        </GroupBox>
```

```
        <GroupBox Header="Setup" HorizontalAlignment="Stretch" Name="groupBox2" VerticalAlignment="Stretch"
Grid.Row="0" Grid.Column="0">
            <Grid>
                <!-- <Grid.ColumnDefinitions>
                    <ColumnDefinition/>
                    <ColumnDefinition/>
                </Grid.ColumnDefinitions>
                <Grid.RowDefinitions>
                    <RowDefinition  />
                    <RowDefinition  />
                    <RowDefinition  />
                    <RowDefinition  />
                    <RowDefinition  />
                </Grid.RowDefinitions>-->

                <Label Content="Active Channels" Height="29" Name="label1" Width="96"
HorizontalAlignment="Center" VerticalAlignment="Top" Margin="6,27,142,0" />
                <ComboBox Height="23" HorizontalAlignment="Center" Name="channels_CB" VerticalAlignment="Top"
Width="58" SelectionChanged="channels_CB_SelectionChanged" Margin="179,29,8,0">
                    <ComboBoxItem>1</ComboBoxItem>
                    <ComboBoxItem>2</ComboBoxItem>
                    <ComboBoxItem>3</ComboBoxItem>
                    <ComboBoxItem>4</ComboBoxItem>
                    <ComboBoxItem>5</ComboBoxItem>
                    <ComboBoxItem>6</ComboBoxItem>
                    <ComboBoxItem>7</ComboBoxItem>
                    <ComboBoxItem>8</ComboBoxItem>
                </ComboBox>
                <Label Content="COM Port" Height="29" Name="label2" Width="96" HorizontalAlignment="Center"
VerticalAlignment="Top" Margin="6,0,144,0" />
                <Button Content="Send Configuration to FPGA" Height="23" HorizontalAlignment="Center"
Name="SendConfig_Button" VerticalAlignment="Top" Width="183" Click="SendConfig_Button_Click"
Margin="30,124,32,0" />
                <ComboBox ItemsSource="{Binding}" Height="23" HorizontalAlignment="Center" Name="com_CB"
VerticalAlignment="Top" Width="58" SelectionChanged="com_CB_SelectionChanged" Margin="179,3,8,0">
                    <!--<ComboBoxItem>COM3</ComboBoxItem>
                    <ComboBoxItem>COM4</ComboBoxItem>-->
                </ComboBox>
                <Button Content="Start Acquisition" Height="23" HorizontalAlignment="Center" Name="acq_button"
VerticalAlignment="Top" Width="182" Click="acq_button_Click" Margin="30,152,33,0" />
                <Button Content="Set Waveform" Height="23" HorizontalAlignment="Center"  Name="setWave_button"
VerticalAlignment="Top" Width="91" Click="setWave_button_Click" Margin="122,95,32,0" />
                <ComboBox Height="23" HorizontalAlignment="Center"  Name="setWaveChan_CB"
VerticalAlignment="Center" Width="120" Margin="117,58,8,161" />

                <Button Content="Get Waveform" Height="23" HorizontalAlignment="Center"  Name="getWave_button"
VerticalAlignment="Bottom" Width="91" Click="getWave_button_Click" Margin="30,0,124,124" />
                <Label Content="Send to ..................." Height="28" HorizontalAlignment="Left"
Margin="6,55,0,0" Name="label8" VerticalAlignment="Top" />
                <Button Content="Start Multi-Stim" Height="23" HorizontalAlignment="Left" Margin="30,184,0,0"
Name="StartMuliStim_Button" VerticalAlignment="Top" Width="91" Click="StartMuliStim_Button_Click" />
                <Button Content="End Multi-Stim" Height="23" HorizontalAlignment="Left" Margin="122,184,0,0"
Name="EndMuliStim_Button" VerticalAlignment="Top" Width="90" Click="EndMuliStim_Button_Click" />
                <Button Content="Single Stim" Height="23" HorizontalAlignment="Left" Margin="30,211,0,0"
Name="SingleStim_Button" VerticalAlignment="Top" Width="182" Click="SingleStim_Button_Click" />
            </Grid>
        </GroupBox>

        <TabControl Grid.Column="2" HorizontalAlignment="Stretch" Name="tabControl1" VerticalAlignment="Stretch"
>
            <TabItem Header="Cypress USB Controls" Name="tabItem1">
                <Grid>
                    <Label Content="Endpoint....." Height="28" HorizontalAlignment="Left" Margin="6,9,0,0"
Name="label3" VerticalAlignment="Top" Width="74"/>
                    <ComboBox Height="23" HorizontalAlignment="Left" Margin="86,11,0,0" Name="EndPointsComboBox"
VerticalAlignment="Top" Width="203" SelectionChanged="EndPointsComboBox_SelectionChanged" />
                    <Label Content="Packets Per Xfer" Height="28" HorizontalAlignment="Left" Margin="6,43,0,0"
Name="label4" VerticalAlignment="Top" />
                    <ComboBox Height="23" HorizontalAlignment="Left" Margin="169,43,0,0" Name="PpxBox"
VerticalAlignment="Top" Width="120">
                        <ComboBoxItem>1</ComboBoxItem>
                        <ComboBoxItem>2</ComboBoxItem>
                        <ComboBoxItem>4</ComboBoxItem>
                        <ComboBoxItem>8</ComboBoxItem>
                        <ComboBoxItem>16</ComboBoxItem>
                        <ComboBoxItem>32</ComboBoxItem>
                        <ComboBoxItem>64</ComboBoxItem>
                        <ComboBoxItem>128</ComboBoxItem>
                    </ComboBox>
```

```
                    <Label Content="Xfers to Queue" Height="28" HorizontalAlignment="Left" Margin="6,77,0,0"
Name="label5" VerticalAlignment="Top" />
                    <ComboBox Height="23" HorizontalAlignment="Left" Margin="169,77,0,0" Name="QueueBox"
VerticalAlignment="Top" Width="120">
                        <ComboBoxItem>1</ComboBoxItem>
                        <ComboBoxItem>2</ComboBoxItem>
                        <ComboBoxItem>4</ComboBoxItem>
                        <ComboBoxItem>8</ComboBoxItem>
                        <ComboBoxItem>16</ComboBoxItem>
                        <ComboBoxItem>32</ComboBoxItem>
                        <ComboBoxItem>64</ComboBoxItem>
                        <ComboBoxItem>128</ComboBoxItem>
                    </ComboBox>
                    <Label Content="Successes" Height="28" HorizontalAlignment="Left" Margin="6,117,0,0"
Name="label6" VerticalAlignment="Top" />
                    <Label Content="Failures" Height="28" HorizontalAlignment="Left" Margin="151,117,0,0"
Name="label7" VerticalAlignment="Top" />
                    <TextBox Height="23" HorizontalAlignment="Left" Margin="74,120,0,0" Name="SuccessBox"
VerticalAlignment="Top" Width="71" />
                    <TextBox Height="23" HorizontalAlignment="Left" Margin="207,120,0,0" Name="FailuresBox"
VerticalAlignment="Top" Width="71" />
                    <GroupBox Header="Throughput (KB/s)" Height="63" HorizontalAlignment="Left"
Margin="6,153,0,0" Name="groupBox5" VerticalAlignment="Top" Width="283">
                        <Grid>
                            <ProgressBar Height="10" HorizontalAlignment="Left" Margin="6,8,0,0"
Name="ProgressBar" VerticalAlignment="Top" Width="259" />
                            <Label Content="Label" Height="28" HorizontalAlignment="Left" Margin="74,15,0,0"
Name="ThroughputLabel" VerticalAlignment="Top" Width="128" HorizontalContentAlignment="Center" />
                        </Grid>
                    </GroupBox>
                </Grid>
            </TabItem>
            <TabItem Header="Scripting" Name="tabItem2">
                <Grid>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="186*" />
                        <RowDefinition Height="47*" />
                    </Grid.RowDefinitions>
                    <TextBox Text="{Binding Path=ScriptText}" HorizontalAlignment="Stretch"
Name="script_Textbox" VerticalAlignment="Stretch" AcceptsReturn="True" AcceptsTab="True"
VerticalScrollBarVisibility="Auto" TextWrapping="Wrap"/>
                    <Button Content="Run Script" Grid.Row="1" Height="23" HorizontalAlignment="Center"
Name="RunScript_Button" VerticalAlignment="Center" Width="75" Click="RunScript_Button_Click"
Margin="107,12,115,12" />
                    <Button Content="Load Script" Grid.Row="1" Height="23" HorizontalAlignment="Left"
Name="LoadScript_Button" VerticalAlignment="Center" Width="75" Click="LoadScript_Button_Click" />
                    <Button Content="Save Script As" Grid.Row="1" Height="23" HorizontalAlignment="Right"
Name="SaveAsScript_Button" VerticalAlignment="Center" Width="84" Margin="0,12" Click="SaveAsScript_Button_Click"
/>
                </Grid>
            </TabItem>
        </TabControl>

        <TabControl Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="3" HorizontalAlignment="Stretch"
Name="tabControl2" VerticalAlignment="Stretch" >
            <TabItem Header="RS232 Communication Log" Name="RS232_Tab">
                <Grid>
                    <DataGrid HorizontalAlignment="Stretch" Name="dataGrid_Comm" VerticalAlignment="Stretch"
AutoGenerateColumns="False" CanUserSortColumns="False" FrozenColumnCount="3" VerticalGridLinesBrush="Gray"
HorizontalGridLinesBrush="Gray" Focusable="False" RowHeaderWidth="0" FontSize="12" Background="#FFF2EFEF"
GridLinesVisibility="All" >
                        <DataGrid.ContextMenu>
                            <ContextMenu>
                                <MenuItem Header="Clear Communication Log" Name="clearContext"
Click="clearContext_Click" />
                            </ContextMenu>
                        </DataGrid.ContextMenu>
                        <DataGrid.Resources>
                            <Style TargetType="{x:Type DataGridCell}">
                                <Style.Triggers>
                                    <Trigger Property="DataGridCell.IsSelected" Value="True">
                                        <Setter Property="Background" Value="Black" />
                                        <Setter Property="BorderBrush" Value="Black" />
                                    </Trigger>
                                </Style.Triggers>
                                <Setter Property="Template">
                                    <Setter.Value>
                                        <ControlTemplate TargetType="{x:Type DataGridCell}">
                                            <Border Name="DataGridCellBorder">
```

```
                                            <ContentControl Content="{TemplateBinding Content}">
                                                <ContentControl.ContentTemplate>
                                                    <DataTemplate>
                                                        <TextBlock Background="#FFF2EFEF"
TextWrapping="WrapWithOverflow" TextTrimming="CharacterEllipsis"
                                Height="auto" Width="auto" Text="{Binding Text}" Foreground="Black" />
                                                    </DataTemplate>
                                                </ContentControl.ContentTemplate>
                                            </ContentControl>
                                        </Border>
                                    </ControlTemplate>

                                </Setter.Value>
                            </Setter>
                        </Style>
                    </DataGrid.Resources>

                    <DataGrid.Columns>
                        <DataGridTextColumn Header="Timestamp" Binding="{Binding Path=Timestamp}"
IsReadOnly="True" Width="200">
                        </DataGridTextColumn>
                        <DataGridTextColumn Header="Send" Binding="{Binding Path=Send}" IsReadOnly="True"
Width="*"></DataGridTextColumn>
                        <DataGridTextColumn Header="Receive" Binding="{Binding Path=Receive}"
IsReadOnly="True" Width="*"></DataGridTextColumn>
                    </DataGrid.Columns>
                </DataGrid>
            </Grid>
        </TabItem>
        <TabItem Header="Graphing" Name="Graphing_Tab">
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="700*" />
                    <ColumnDefinition Width="164" />
                    <ColumnDefinition Width="164" />
                    <ColumnDefinition Width="150" />
                </Grid.ColumnDefinitions>
                <Grid.RowDefinitions>
                    <RowDefinition Height="182*" />
                    <RowDefinition Height="33" />
                </Grid.RowDefinitions>

                <d3:ChartPlotter Grid.ColumnSpan="3" Name="plotter" Margin="1,1,1,1">

                </d3:ChartPlotter>
                <Button Content="Load File" Grid.Row="1" Grid.Column="3" Height="23"
HorizontalAlignment="Center" Name="LoadFile_Button" VerticalAlignment="Center" Width="70"
Click="LoadFile_Button_Click" Margin="70,0,0,0" />
                <Button Content="Output CSV" Grid.Row="1" Grid.Column="3" Height="23"
HorizontalAlignment="Center" Name="Output_CSV_Button" VerticalAlignment="Center" Width="70"
Click="Output_CSV_Button_Click" Margin="0,0,70,0" />


                <Slider Grid.Row="1" HorizontalAlignment="Stretch" Name="slider1" VerticalAlignment="Center"
ValueChanged="slider1_ValueChanged"  />
                <Label Content="GraphView" Grid.Column="1" Grid.Row="1" Height="28"
HorizontalAlignment="Left" Margin="3,3,0,0" Name="label9" VerticalAlignment="Top" />
                <TextBox HorizontalAlignment="Right" Name="ViewSelect_TextBox" VerticalAlignment="Top"
VerticalScrollBarVisibility="Auto" HorizontalScrollBarVisibility="Auto" IsReadOnly="True" Grid.Row="1"
Grid.Column="1" Margin="0,6,9,0" Width="83" />
                <Label Content="NumSamples" Grid.Column="1" Grid.Row="1" Height="28"
HorizontalAlignment="Left" Margin="163,3,0,0" Name="label10" VerticalAlignment="Top" Grid.ColumnSpan="2" />
                <DataGrid AutoGenerateColumns="False" Grid.Row="0" Grid.Column="3"
HorizontalAlignment="Stretch"  Name="Graph_DataGrid" VerticalAlignment="Stretch"
HorizontalGridLinesBrush="White" Foreground="Black" CanUserResizeColumns="False" CanUserReorderColumns="False"
CanUserResizeRows="False" CanUserSortColumns="False" IsReadOnly="True" VerticalGridLinesBrush="White"
HorizontalScrollBarVisibility="Disabled" VerticalScrollBarVisibility="Auto" RowHeaderWidth="0" Margin="1,1,1,1"
SelectionChanged="Graph_DataGrid_SelectionChanged">
                    <DataGrid.Columns>
                        <DataGridTextColumn Header="Select Channels to Graph" Binding="{Binding
Path=channel}" IsReadOnly="True" Width="150"></DataGridTextColumn>
                    </DataGrid.Columns>
                </DataGrid>
                <ComboBox Grid.Column="2" Grid.Row="1" Height="23" HorizontalAlignment="Left"
Margin="81,5,0,0" Name="NumSamples_CB" VerticalAlignment="Top" Width="81"
SelectionChanged="NumSamples_CB_SelectionChanged" />
            </Grid>
        </TabItem>
    </TabControl>
```

```
        <StatusBar Grid.Row="2" Grid.ColumnSpan="3">
            <StatusBarItem Name="AcqStatus" Content="{Binding Path=TransferStatusText}"/>
        </StatusBar>
    </Grid>
</Window>
```

MainWindows.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.IO;
using System.Windows.Forms;
using System.Threading;
using System.ComponentModel;
using System.Windows.Threading;
using CyUSB;
using Microsoft.Research.DynamicDataDisplay;
using Microsoft.Research.DynamicDataDisplay.DataSources;

namespace Data_Acq_and_Stim_Control_Center
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        int NUM_CHANNELS;
        Scripting script;
        public static FPGA_Commands fpga_control;
        string[] theSerialPortNames;
        BindingList<Channels> Channel_List = new BindingList<Channels>();

        CypressDataAcq CypressDA;
        Graphing graph;
        int previous_graph_view = 0;

        public MainWindow()
        {
            InitializeComponent();

            script = new Scripting();
            fpga_control = new FPGA_Commands();
            graph = new Graphing();
            CypressDA = new CypressDataAcq();

            dataGrid_Comm.ItemsSource = fpga_control.RS232_Com.ComLog;

            channels_CB.SelectedIndex = 7;

            theSerialPortNames = System.IO.Ports.SerialPort.GetPortNames();
            com_CB.DataContext = theSerialPortNames;

            this.Title = "Data Acquisition and Stimlation Control Center - Version 0.1";

            dataGrid1.ItemsSource = Channel_List;

            com_CB.SelectedIndex = 1;

            setWaveChan_CB.ItemsSource = Channel_List;
            setWaveChan_CB.DisplayMemberPath = "channel";
            setWaveChan_CB.SelectedIndex = 0;

            fpga_control.RS232_Com.ComLog.ListChanged += new
System.ComponentModel.ListChangedEventHandler(Comm_Log_Changed);

            script_Textbox.DataContext = script;

            EndPointsComboBox.ItemsSource = CypressDA.EndpointList;

            if (EndPointsComboBox.Items.Count > 0)
                EndPointsComboBox.SelectedIndex = 0;
```

```
            AcqStatus.DataContext = CypressDA;


            NumSamples_CB.ItemsSource = graph.SupportedNumSamples;
            NumSamples_CB.SelectedIndex = 0;

            Graph_DataGrid.ItemsSource = Channel_List;


            //script.StartAcquisitionEvent += script_StartAcqHandler;
            //script.EndAcquisitionEvent += script_EndAcqHandler;

            script.Register(t => ScriptHandler(t));
        }
        private void ScriptHandler(String str)
        {
            if (str == "StartAcquisition") { acq_triggered(); }
            else if (str == "EndAcquisition") { acq_triggered(); }
        }

        //private void script_StartAcqHandler(object sender, EventArgs e)
        //{
        //    acq_button_Click(this, new RoutedEventArgs());
        //    //CypressDA.Start_Cypress_Acq();
        //}
        //private void script_EndAcqHandler(object sender, EventArgs e)
        //{
        //    acq_button_Click(this, new RoutedEventArgs());
        //    //CypressDA.Stop_Cypress_Acq();
        //}

        #region Config Events
        /*****************************************************************************************************
 * dataGrid1 BeginningEdit Event Handler
 *
 * Launches OpenFileDialog when attempting to change waveform_file
/*****************************************************************************************************/
        private void dataGrid1_BeginningEdit(object sender, DataGridBeginningEditEventArgs e)
        {
            if (dataGrid1.CurrentCell.Column.DisplayIndex == 1)
            {
                System.Windows.Forms.OpenFileDialog fd = new System.Windows.Forms.OpenFileDialog();
                fd.ShowDialog();

                if (fd.FileName != "") { Channel_List[dataGrid1.SelectedIndex].waveform_file = fd.FileName; }
            }
        }

        /*****************************************************************************************************
         * Number of Channels Combo Box Event Handler
         *
         * Update NUM_CHANNELS from combo box and calls Update_Channel_List
        /*****************************************************************************************************/
        private void channels_CB_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            NUM_CHANNELS = Convert.ToInt16(channels_CB.SelectedIndex.ToString()) + 1;
            Update_Channel_List();
            graph.ChannelsToGraph = NUM_CHANNELS;
        }

        /*****************************************************************************************************
         * Number of Channels Combo Box Event Handler
         *
         * Update NUM_CHANNELS from combo box and calls Update_Channel_List
        /*****************************************************************************************************/
        private void com_CB_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            fpga_control.RS232_Com.PortName = theSerialPortNames[com_CB.SelectedIndex];
            fpga_control.RS232_Com.Init_Port();
        }

        /*****************************************************************************************************
         * Update_Channel_List
         *
         * Modifies the bindinglist used for storing channel configuration by adding or removing channels
         * until the Channel_List count equals NUM_CHANNELS
        /*****************************************************************************************************/
        private void Update_Channel_List()
        {
```

```
            while (Channel_List.Count != NUM_CHANNELS)
            {
                if (Channel_List.Count > NUM_CHANNELS)
                {
                    Channel_List.RemoveAt(Channel_List.Count - 1);
                    graph.GraphData.RemoveAt(graph.GraphData.Count - 1);
                    plotter.Children.RemoveAt(plotter.Children.Count - 1);
                }
                else
                {
                    Channel_List.Add(new Channels("Channel " + (Channel_List.Count + 1).ToString(), "",
"Stimulation"));
                    graph.GraphData.Add(new GraphingData());
                    //plotter.AddLineGraph(graph.GraphData[Channel_List.Count - 1].Channel_GraphData);
                }
            }
        }
        #endregion //Config Events

        #region Manual Command Events
        /**************************************************************************************************
         * SendConfig Button Click Event
         *
         * Sends SendConfig Message out serial port to FPGA
        **************************************************************************************************/
        private void SendConfig_Button_Click(object sender, RoutedEventArgs e)
        {
            Byte channel = Convert.ToByte(setWaveChan_CB.SelectedIndex + 1);
            Byte config = 0x00;

            if (Channel_List[setWaveChan_CB.SelectedIndex].mode == "Stimulation") { config += 0x10; }
            if (Channel_List[setWaveChan_CB.SelectedIndex].sw4 == "On") { config += 0x08; }
            if (Channel_List[setWaveChan_CB.SelectedIndex].sw3 == "On") { config += 0x04; }
            if (Channel_List[setWaveChan_CB.SelectedIndex].sw2 == "On") { config += 0x02; }
            if (Channel_List[setWaveChan_CB.SelectedIndex].sw1 == "On") { config += 0x01; }

            fpga_control.FPGA_SetConfig(channel, config);
        }


        private void acq_button_Click(object sender, RoutedEventArgs e)
        {
            acq_triggered();
        }

        private void acq_triggered()
        {
            if (acq_button.Content.ToString() == "Start Acquisition")
            {
                CypressDA.Flush_Cypress_Chip();

                // Adding call to start cypress acquisition
                CypressDA.Start_Cypress_Acq();

                fpga_control.FPGA_StartAcquisition();

                acq_button.Content = "End Acquisition";

            }
            else
            {
                // Adding call to stop cypress acquisition
                CypressDA.Stop_Cypress_Acq();

                fpga_control.FPGA_EndAcquisition();

                acq_button.Content = "Start Acquisition";
            }
        }

        private void setWave_button_Click(object sender, RoutedEventArgs e)
        {
            Byte Channel = Convert.ToByte(setWaveChan_CB.SelectedIndex + 1);
            String Filename = Channel_List[setWaveChan_CB.SelectedIndex].waveform_file;

            fpga_control.FPGA_SetWaveform(Channel, Filename);
        }

        private void getWave_button_Click(object sender, RoutedEventArgs e)
```

```
{
    Byte Channel = Convert.ToByte(setWaveChan_CB.SelectedIndex + 1);

    fpga_control.FPGA_GetWaveform(Channel);
}

private void StartMuliStim_Button_Click(object sender, RoutedEventArgs e)
{
    int chan_index = setWaveChan_CB.SelectedIndex;
    Byte channel = 0x00;

    // Set Channel
    if (chan_index == 0) { channel = 0x01; }
    else if (chan_index == 1) { channel = 0x02; }
    else if (chan_index == 2) { channel = 0x04; }
    else if (chan_index == 3) { channel = 0x08; }
    else if (chan_index == 4) { channel = 0x10; }
    else if (chan_index == 5) { channel = 0x20; }
    else if (chan_index == 6) { channel = 0x40; }
    else if (chan_index == 7) { channel = 0x80; }

    fpga_control.FPGA_StartMultiStim(channel);
}

private void EndMuliStim_Button_Click(object sender, RoutedEventArgs e)
{
    fpga_control.FPGA_EndMuliStim();
}

private void SingleStim_Button_Click(object sender, RoutedEventArgs e)
{
    int chan_index = setWaveChan_CB.SelectedIndex;
    byte channel = 0x00;

    // Set Channel and Continuous
    if (chan_index == 0) { channel = 0x01; }
    else if (chan_index == 1) { channel = 0x02; }
    else if (chan_index == 2) { channel = 0x04; }
    else if (chan_index == 3) { channel = 0x08; }
    else if (chan_index == 4) { channel = 0x10; }
    else if (chan_index == 5) { channel = 0x20; }
    else if (chan_index == 6) { channel = 0x40; }
    else if (chan_index == 7) { channel = 0x80; }

    fpga_control.FPGA_SingleStim(channel);
}
#endregion //Manual Command Controls

#region Scripting Events
private void SaveAsScript_Button_Click(object sender, RoutedEventArgs e)
{
    System.Windows.Forms.SaveFileDialog fd = new System.Windows.Forms.SaveFileDialog();
    fd.ShowDialog();

    try
    {
        using (StreamWriter sw = new StreamWriter(fd.FileName))
        {
            sw.Write(script.ScriptText);
        }

    }
    catch { }
}


private void LoadScript_Button_Click(object sender, RoutedEventArgs e)
{
    System.Windows.Forms.OpenFileDialog fd = new System.Windows.Forms.OpenFileDialog();
    fd.ShowDialog();

    script_Textbox.Text = "";

    try
    {
        using (StreamReader sr = new StreamReader(fd.FileName))
        {
            while (!sr.EndOfStream)
            {
```

```
                    //sr.ReadLine();
                    script.ScriptText += sr.ReadLine() + "\r\n";
                }

            }

        }
        catch { }
    }

    private void RunScript_Button_Click(object sender, RoutedEventArgs e)
    {
        script.StartScript();
    }


    #endregion // Scripting Controls


    private void Comm_Log_Changed(object sender, EventArgs e)
    {
        if (dataGrid_Comm.Items.Count > 2)
        {
            dataGrid_Comm.ScrollIntoView(dataGrid_Comm.Items[dataGrid_Comm.Items.Count - 1]);
        }
    }

    private void Window_Closing(object sender, CancelEventArgs e)
    {
        if (CypressDA.usbDevices != null)
            CypressDA.usbDevices.Dispose();
    }

    /*Summary
     This is a system event handler, when the selected index changes(end point selection).
    */
    private void EndPointsComboBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
    {
        CypressDA.SetEndpoint(EndPointsComboBox.SelectedIndex);
    }

    #region graphing events
    private void Graph_DataGrid_SelectionChanged(object sender, SelectionChangedEventArgs e)
    {
        graph.CurrentView = Convert.ToInt16(slider1.Value);
        plotter.Children.RemoveAll<LineGraph>();

        foreach (GraphingData data in graph.GraphData)
        {
            data.Channel_GraphData.Collection.Clear();
        }

        foreach (Channels temp in Graph_DataGrid.SelectedItems)
        {
            string channel_name = temp.channel;
            string[] split = channel_name.Split(' ');
            int index = Convert.ToInt16(split[1]) - 1;

            Point[] AllPoints = graph.GraphData[index].Channel_AllData.Collection.ToArray();

            int points_to_graph = 0; //The last view could have fewer points than SamplesPerView
            while (AllPoints.Length < graph.SamplesPerView)
            {
                NumSamples_CB.SelectedIndex--;
            }

            if (graph.CurrentView == slider1.Maximum)
            {
                points_to_graph = AllPoints.Length - graph.CurrentView * graph.SamplesPerView;
            }
            else
            {
                points_to_graph = graph.SamplesPerView;
            }

            Point[] GraphPoints = new Point[points_to_graph];
            Array.Copy(AllPoints, graph.CurrentView * graph.SamplesPerView, GraphPoints, 0,
points_to_graph);
```

```
                //graph.GraphData[index].Channel_GraphData.Collection.Clear();
                graph.GraphData[index].Channel_GraphData.AppendMany(GraphPoints);

                plotter.AddLineGraph(graph.GraphData[index].Channel_GraphData, 1, temp.channel);
            }
            plotter.FitToView();
        }

        private void slider1_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
        {
            graph.CurrentView = Convert.ToInt16(slider1.Value);
            ViewSelect_TextBox.Text = Convert.ToInt16(slider1.Value).ToString();

            if (graph.CurrentView != previous_graph_view)
            {
                // chan1Points = Channel1_Data.Collection.ToArray();
                //Point[] points = new Point[NUM_SAMPLES_DISPLAYED];
                //Array.Copy(chan1Points, selected_view * NUM_SAMPLES_DISPLAYED, points, 0,
NUM_SAMPLES_DISPLAYED);
                //Channel1_Data.Channel_GraphData.Collection.Clear();
                //Channel1_Data.Channel_GraphData.AppendMany(points);

                foreach (GraphingData data in graph.GraphData)
                {
                    data.Channel_GraphData.Collection.Clear();
                }

                foreach (Channels temp in Graph_DataGrid.SelectedItems)
                {
                    string channel_name = temp.channel;
                    string[] split = channel_name.Split(' ');
                    int index = Convert.ToInt16(split[1]) - 1;

                    Point[] AllPoints = graph.GraphData[index].Channel_AllData.Collection.ToArray();

                    int points_to_graph = 0; //The last view could have fewer points than SamplesPerView
                    while (AllPoints.Length < graph.SamplesPerView)
                    {
                        NumSamples_CB.SelectedIndex--;
                    }

                    if (graph.CurrentView == slider1.Maximum)
                    {
                        points_to_graph = AllPoints.Length - graph.CurrentView * graph.SamplesPerView;
                    }
                    else
                    {
                        points_to_graph = graph.SamplesPerView;
                    }


                    Point[] GraphPoints = new Point[points_to_graph];
                    Array.Copy(AllPoints, graph.CurrentView * graph.SamplesPerView, GraphPoints, 0,
points_to_graph);

                    //GraphData[index].Channel_GraphData.Collection.Clear();
                    graph.GraphData[index].Channel_GraphData.AppendMany(GraphPoints);
                }
                plotter.FitToView();
            }
            previous_graph_view = graph.CurrentView;
        }

        private void LoadFile_Button_Click(object sender, RoutedEventArgs e)
        {
            // Start computation process in second thread
            /*Thread simThread = new Thread(new ThreadStart(LoadFile));
            simThread.IsBackground = true;
            simThread.Start();*/
            try
            {
                graph.LoadFile();
                //NumSamples_CB_SelectionChanged();
                Graph_DataGrid.SelectedIndex = 0;

                NumSamples_CB_SelectionChanged();

                //graph.SamplesPerView = graph.SupportedNumSamples[NumSamples_CB.SelectedIndex];
```

```
                //if (graph.GraphData[Graph_DataGrid.SelectedIndex].Channel_AllData.Collection.Count %
graph.SamplesPerView == 0)
                //{
                //    slider1.Maximum = graph.TotalViews - 1;
                //}
                //else { slider1.Maximum = graph.TotalViews; }

                //slider1.Minimum = 0;
                ////slider1.Maximum = graph.TotalViews - 1;
                //slider1.Value = slider1.Maximum;
            }
            catch { }
        }

        private void NumSamples_CB_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            graph.SamplesPerView = graph.SupportedNumSamples[NumSamples_CB.SelectedIndex];



            if (graph.GraphData[0].Channel_AllData.Collection.Count != 0)
            {
                while (graph.GraphData[0].Channel_AllData.Collection.Count < graph.SamplesPerView)
                {
                    NumSamples_CB.SelectedIndex--;
                }

                graph.TotalViews =
graph.GraphData[Graph_DataGrid.SelectedIndex].Channel_AllData.Collection.Count / graph.SamplesPerView;
                slider1.Minimum = 0;

                if (graph.GraphData[Graph_DataGrid.SelectedIndex].Channel_AllData.Collection.Count %
graph.SamplesPerView == 0)
                {
                    slider1.Maximum = graph.TotalViews - 1;
                }
                else { slider1.Maximum = graph.TotalViews; }

                slider1.Value = slider1.Maximum;
            }
        }


        private void NumSamples_CB_SelectionChanged()
        {
            graph.SamplesPerView = graph.SupportedNumSamples[NumSamples_CB.SelectedIndex];



            if (graph.GraphData[0].Channel_AllData.Collection.Count != 0)
            {
                while (graph.GraphData[0].Channel_AllData.Collection.Count < graph.SamplesPerView)
                {
                    NumSamples_CB.SelectedIndex--;
                }

                graph.TotalViews =
graph.GraphData[Graph_DataGrid.SelectedIndex].Channel_AllData.Collection.Count / graph.SamplesPerView;
                slider1.Minimum = 0;

                if (graph.GraphData[Graph_DataGrid.SelectedIndex].Channel_AllData.Collection.Count %
graph.SamplesPerView == 0)
                {
                    slider1.Maximum = graph.TotalViews - 1;
                }
                else { slider1.Maximum = graph.TotalViews; }

                slider1.Value = slider1.Maximum;
            }
        }

        #endregion // graphing events



        private void clearContext_Click(object sender, RoutedEventArgs e)
        {
            fpga_control.RS232_Com.ComLog.Clear();
        }
```

```
        private void Output_CSV_Button_Click(object sender, RoutedEventArgs e)
        {
            System.Windows.Forms.SaveFileDialog fd = new System.Windows.Forms.SaveFileDialog();
            fd.ShowDialog();

            try
            {
                using (StreamWriter sw = new StreamWriter(fd.FileName))
                {


sw.WriteLine("Time,Channel1,Channel2,Channel3,Channel4,Channel5,Channel6,Channel7,Channel8");

                    int j = 0;

                    for(int i = 0; i < graph.GraphData[0].Channel_AllData.Collection.Count; i++)
                    {
                        sw.Write("{0},", graph.GraphData[0].Channel_AllData.Collection[i].X);
                        for(j = 0; j < graph.GraphData.Count; j++)
                        {
                            sw.Write("{0},", graph.GraphData[j].Channel_AllData.Collection[i].Y);
                        }
                        sw.WriteLine("");
                    }


                }

            }
            catch { }
        }


    }

}
```

RS232_Communication.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.IO.Ports;
using System.ComponentModel;
using System.Windows.Threading;
using System.Runtime.InteropServices;

namespace Data_Acq_and_Stim_Control_Center
{
    public class RS232_Communication : INotifyPropertyChanged
    {
        private readonly SynchronizationContext syncContext;
        private readonly List<Action<CommunicationLog>> actions;

        SerialPort port;
        public BindingList<CommunicationLog> ComLog;

        public RS232_Communication()
        {
            syncContext = AsyncOperationManager.SynchronizationContext;
            actions = new List<Action<CommunicationLog>>();
            actions.Add(t => AddToComLog(t));


            ComLog = new BindingList<CommunicationLog>();

            // Instantiate the communications port with some basic settings
            port = new SerialPort("COM4", 115200, Parity.None, 8, StopBits.One);

            PortStatus = "RS232 Port not Initialized - Select Comm Port!";

            // Attach a method to be called when there
            // is data waiting in the port's buffer
            port.DataReceived += new SerialDataReceivedEventHandler(port_DataReceived);
        }

        private void AddToComLog(CommunicationLog temp)
        {
            ComLog.Add(temp);
        }

        public bool Init_Port()
        {
            if (port.IsOpen)
            {
                port.Close();
            }
            port.PortName = PortName;

            try
            {
                port.Open();
                PortStatus = PortName + " Opened Successfully!";
                return true;
            }
            catch { PortStatus = PortName + " not opened!"; return false; }
        }

        public void SendData(byte[] msg)
        {
            string temp = "";

            for (int i = 0; i < msg.Length; i++)
            {
                temp += "0x" + msg[i].ToString("X2") + " ";
            }

            try
            {
                port.Write(msg, 0, msg.Length);
                syncContext.Post(t => RS232Data_Received((CommunicationLog)t), new
CommunicationLog(DateTime.Now.ToString("MM/dd/yyyy HH:mm:ss.fff"), temp, ""));
```

```
            }
            catch
            {
                syncContext.Post(t => RS232Data_Received((CommunicationLog)t), new
CommunicationLog(DateTime.Now.ToString("MM/dd/yyyy HH:mm:ss.fff"), "Error writing to port!", ""));
                //ComLog.Add(new CommunicationLog(DateTime.Now.ToString("MM/dd/yyyy HH:mm:ss.fff"), "Error
writing to port!", ""));
            }
        }


        /*****************************************************************************************************
         * RS232 Data Received Event
         *
         * Update RS232 Communication Log with received data
        *****************************************************************************************************/
        private void port_DataReceived(object sender, SerialDataReceivedEventArgs e)
        {
            // Show all the incoming data in the port's buffer
            Thread.Sleep(100);
            byte[] buf = new byte[255];
            int bytes_to_read = port.BytesToRead;
            port.Read(buf, 0, bytes_to_read);

            string temp = "";

            for (int i = 0; i < bytes_to_read; i++)
            {
                temp += "0x" + buf[i].ToString("X2") + " ";
            }
            //this.Dispatcher.Invoke(DispatcherPriority.Normal, (Action)(() =>
            //{
                //ComLog.Add(new CommunicationLog(DateTime.Now.ToString("MM/dd/yyyy HH:mm:ss.fff"), "", temp));
            //}));
            syncContext.Post(t => RS232Data_Received((CommunicationLog)t), new
CommunicationLog(DateTime.Now.ToString("MM/dd/yyyy HH:mm:ss.fff"), "", temp));
        }

        private void RS232Data_Received(CommunicationLog data)
        {
            foreach (var action in actions)
                action(data);
        }

        private string _portStatus;
        public string PortStatus
        {
            get { return _portStatus; }
            set
            {
                _portStatus = value;
                this.NotifyPropertyChanged("PortStatus");
            }
        }

        private string _portName;
        public string PortName
        {
            get { return _portName; }
            set
            {
                _portName = value;
                this.NotifyPropertyChanged("PortName");
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;
        private void NotifyPropertyChanged(string name)
        {
            if (PropertyChanged != null)
                PropertyChanged(this, new PropertyChangedEventArgs(name));
        }
    }

/*****************************************************************************************************
 * Communication Log class
 *
 * contains configuration information for channels
*****************************************************************************************************/
```

```
    public class CommunicationLog : INotifyPropertyChanged
    {
        private string _Timestamp;
        private string _Send;
        private string _Receive;

        public event PropertyChangedEventHandler PropertyChanged;

        public CommunicationLog(string Timestamp, string Send, string Receive)
        {
            _Timestamp = Timestamp;
            _Send = Send;
            _Receive = Receive;
        }

        public string Timestamp
        {
            get { return _Timestamp; }
            set
            {
                _Timestamp = value;
                this.NotifyPropertyChanged("Timestamp");
            }
        }

        public string Send
        {
            get { return _Send; }
            set
            {
                _Send = value;
                this.NotifyPropertyChanged("Send");
            }
        }

        public string Receive
        {
            get { return _Receive; }
            set
            {
                _Receive = value;
                this.NotifyPropertyChanged("Receive");
            }
        }

        private void NotifyPropertyChanged(string name)
        {
            if (PropertyChanged != null)
                PropertyChanged(this, new PropertyChangedEventArgs(name));
        }
    }
}
```

Scripting.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Windows.Threading;
using System.ComponentModel;

namespace Data_Acq_and_Stim_Control_Center
{
    public class Scripting : INotifyPropertyChanged
    {
        //public delegate void StartAcquisitionHandler(object sender, EventArgs e);
        //public event StartAcquisitionHandler StartAcquisitionEvent;

        //public delegate void EndAcquisitionHandler(object sender, EventArgs e);
        //public event EndAcquisitionHandler EndAcquisitionEvent;

        #region SynchronizationContext - Script_Command_To_Main

        private readonly SynchronizationContext syncContext;
        private readonly List<Action<String>> actions;

        private void Script_Command_To_Main(String data)
        {
            foreach (var action in actions)
                action(data);
        }

        public void Register(Action<string> action)
        {
            actions.Add(action);
        }
        #endregion //SynchronizationContext - Script_Command_To_Main


        public Scripting()
        {
            syncContext = AsyncOperationManager.SynchronizationContext;
            actions = new List<Action<string>>();
        }

        public void StartScript()
        {
            var thread = new Thread(RunScript);
            thread.IsBackground = true;
            thread.Start();
        }

        private void RunScript()
        {
            string[] commands = (ScriptText).Split(new String[] { Environment.NewLine },
StringSplitOptions.RemoveEmptyEntries);

            foreach (string str in commands)
            {
                if (str.StartsWith("SetConfig(")) { SetConfig(str); }
                else if (str.StartsWith("GetConfig(")) { GetConfig(str); }
                else if (str.StartsWith("SetWaveform(")) { SetWaveform(str); }
                else if (str.StartsWith("GetWaveform(")) { GetWaveform(str); }
                else if (str.StartsWith("StartAcquisition(")) {
                    syncContext.Post(e => Script_Command_To_Main((string)e), "StartAcquisition");

//MainWindow.fpga_control.FPGA_StartAcquisition();
                }
                else if (str.StartsWith("EndAcquisition(")){ syncContext.Post(e =>
Script_Command_To_Main((string)e), "EndAcquisition");
                                                            //MainWindow.fpga_control.FPGA_EndAcquisition();
                }
                else if (str.StartsWith("SingleStim(")) { SingleStim(str); }
                else if (str.StartsWith("StartMultiStim(")) { StartMultiStim(str); }
                else if (str.StartsWith("EndMultiStim(")) { MainWindow.fpga_control.FPGA_EndMuliStim(); }
                else if (str.StartsWith("Sleep(")) { SleepCmd(str); }
                else { }
            }
```

```
}

private void SetConfig(string str)
{
    int payloadStart = str.IndexOf('(') + 1;
    int payloadEnd = str.IndexOf(')') - 1;
    int payloadLength = payloadEnd - payloadStart + 1;
    string[] payload = (str.Substring(payloadStart, payloadLength)).Split(',');

    if (payload[0].Length == 1) { payload[0] = "0" + payload[0]; }
    if (payload[1].Length == 1) { payload[1] = "0" + payload[0]; }

    Byte Channel = Convert.ToByte((ToNibble(payload[0][0]) << 4) + ToNibble(payload[0][1]));
    Byte Config = Convert.ToByte((ToNibble(payload[1][0]) << 4) + ToNibble(payload[1][1]));

    MainWindow.fpga_control.FPGA_SetConfig(Channel, Config);
}

private void GetConfig(string str)
{
    int payloadStart = str.IndexOf('(') + 1;
    int payloadEnd = str.IndexOf(')') - 1;
    int payloadLength = payloadEnd - payloadStart + 1;
    string[] payload = (str.Substring(payloadStart, payloadLength)).Split(',');

    if (payload[0].Length == 1) { payload[0] = "0" + payload[0]; }

    Byte Channel = Convert.ToByte((ToNibble(payload[0][0]) << 4) + ToNibble(payload[0][1]));

    MainWindow.fpga_control.FPGA_GetConfig(Channel);
}

private void SetWaveform(string str)
{
    int payloadStart = str.IndexOf('(') + 1;
    int payloadEnd = str.IndexOf(')') - 1;
    int payloadLength = payloadEnd - payloadStart + 1;
    string[] payload = (str.Substring(payloadStart, payloadLength)).Split(',');

    if (payload[0].Length == 1) { payload[0] = "0" + payload[0]; }

    Byte Channel = Convert.ToByte((ToNibble(payload[0][0]) << 4) + ToNibble(payload[0][1]));

    string Filename = payload[1];

    MainWindow.fpga_control.FPGA_SetWaveform(Channel, Filename);
}

private void GetWaveform(string str)
{
    int payloadStart = str.IndexOf('(') + 1;
    int payloadEnd = str.IndexOf(')') - 1;
    int payloadLength = payloadEnd - payloadStart + 1;
    string[] payload = (str.Substring(payloadStart, payloadLength)).Split(',');

    if (payload[0].Length == 1) { payload[0] = "0" + payload[0]; }

    Byte Channel = Convert.ToByte((ToNibble(payload[0][0]) << 4) + ToNibble(payload[0][1]));

    MainWindow.fpga_control.FPGA_GetWaveform(Channel);
}

private void StartMultiStim(string str)
{
    int payloadStart = str.IndexOf('(') + 1;
    int payloadEnd = str.IndexOf(')') - 1;
    int payloadLength = payloadEnd - payloadStart + 1;
    string[] payload = (str.Substring(payloadStart, payloadLength)).Split(',');

    if (payload[0].Length == 1) { payload[0] = "0" + payload[0]; }

    Byte Channel = Convert.ToByte((ToNibble(payload[0][0]) << 4) + ToNibble(payload[0][1]));

    MainWindow.fpga_control.FPGA_StartMultiStim(Channel);
}

private void SingleStim(string str)
{
    int payloadStart = str.IndexOf('(') + 1;
```

```
            int payloadEnd = str.IndexOf(')') - 1;
            int payloadLength = payloadEnd - payloadStart + 1;
            string[] payload = (str.Substring(payloadStart, payloadLength)).Split(',');

            if (payload[0].Length == 1) { payload[0] = "0" + payload[0]; }

            Byte Channel = Convert.ToByte((ToNibble(payload[0][0]) << 4) + ToNibble(payload[0][1]));

            MainWindow.fpga_control.FPGA_SingleStim(Channel);
        }

        private void SleepCmd(string str)
        {
            int payloadStart = str.IndexOf('(') + 1;
            int payloadEnd = str.IndexOf(')') - 1;
            int payloadLength = payloadEnd - payloadStart + 1;
            Int16 sleep = Convert.ToInt16(str.Substring(payloadStart, payloadLength));
            //Wait(sleep);
            Thread.Sleep(sleep);
        }

        private byte ToNibble(char c)
        {
            if ('0' <= c && c <= '9') { return Convert.ToByte(c - '0'); }
            else if ('a' <= c && c <= 'f') { return Convert.ToByte(c - 'a' + 10); }
            else if ('A' <= c && c <= 'F') { return Convert.ToByte(c - 'A' + 10); }
            else
            {
                throw new ArgumentException(String.Format("Character '{0}' cannot be translated to a hexadecimal
value because it is not one of 0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f,A,B,C,D,E,F", c));
            }
        }

        private string _scriptText;
        public string ScriptText
        {
            get { return _scriptText; }
            set
            {
                _scriptText = value;
                this.NotifyPropertyChanged("ScriptText");
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;
        private void NotifyPropertyChanged(string name)
        {
            if (PropertyChanged != null)
                PropertyChanged(this, new PropertyChangedEventArgs(name));
        }

    }
}
```

References

[1] S. Marom and G. Shahaf,"Development, learning, and memory in large random networks of cortical neurons: lessons beyond anatomy," *Quarterly Review of Biophysics*, vol. 35, pp. 63-87, 2002.

[2] T. DeMarse, A. Cadotte, P. Douglas, P. He, and V. Trinh, "Computation within cultured neural networks," in *Proceedings of the 26th Annual Conference of the IEEE Engineering in Medicine and Biology Society*, San Francisco, CA, September 1-5, 2004.

[3] S. M. Potter, D. A. Wagenaar, and T. B. DeMarse, "Closing the loop: Stimulation feedback systems for embodied MEA cultures," in *Advances in Network Electrophysiology Using Multi-Electrode Arrays*, M. Taketani and M. Baudry, Eds. Springer, 2006, pp. 215-242.

[4] E. Armstrong, A. Ranganathan, and S. Westbrooks, *Real-Time Feedback Control of Neuron Cell Culture Electrical Activity*, 2007, Senior Design Project, Faculty advisors: B. Bazuin, J. Gesink, D. A. Miller, and F. L. Severance.

[5] N. McCaskey, J. P. John, and P. Vandeusen, *Real-Time Feedback Control of Neuron Cell Culture Electrical Activity*, 2007, Senior Design Project, Faculty advisors: B. Bazuin, J. Gesink, D. A. Miller, and F. L. Severance.

[6] Y. Jimbo, N. Kasai, K. Torimitsu, T. Tateno, and H. P. C. Robinsion, "A system for MEA-Based multisite stimulation," *IEEE Transactions on Biomedical Engineering*, vol. 50, no. 2, pp. 241-248, February 2003.

[7] D. A. Wagenaar and S. M. Potter, "A versatile all-channel stimulator for electrode arrays, with real-time control," *Journal of Neural Engineering*, vol. 1, pp. 39-45, 2004.

[8] T. Caruso, E. Daiek, and E. Jones, *Low Noise Amplification and Stimulation System for Multi-Electrode Arrays*, 2008, Senior Design Project, Faculty advisors: B. Bazuin, J. Gesink, D. A. Miller, and F. L. Severance.

[9] J. D. Stahl, "Dual channel low noise amplifier for experiments in neurophysiology," Master's thesis, Western Michigan University, 2009, Committee members: B. Bazuin, J. Gesink, D. A. Miller (Chair), and F. L. Severance.

[10] K. Batzer, R. Corsi, and E. Crampton, *Electrophysiology Measurement and Stimulation System*, Fall 2009-Spring 2010, Senior Design Project, Faculty advisors: B. Bazuin, J. Gesink, D. A. Miller, and F. L. Severance; Contributors: M. Ellinger and J. Stahl.

[11] B. Berger, S. Goveia, and L. Morgan, *Electrophysiology Data Acquisition System*, 2012, Senior Design Project, Faculty advisors: B. Bazuin, J. Gesink, D. A. Miller, and F. L. Severance; Contributor: K. Batzer.

[12] R. F. Olivo, *Lab 4: Action Potentials in Earthworm Giant Axons*, Smith College, Northampton, MA, available at http://www.science.smith.edu/departments/NeuroSci/courses/bio330/labs/L4giants.html.

[13] D. Kueh and C. Linn, "Neurophysiology in the earthworm," *Laboratory Manual for Human Physiology*, 2nd ed. Mason, Ohio: Cengage Learning, 2012, pp. 19-45.

[14] M. Ellinger, "Acquisition and analysis of biological neural network action potential sequences," Master's thesis, Western Michigan University, 2009, Committee members: B Bazuin, J. Gesink, D. A. Miller, and F. L. Severance (Chair).

[15] D. Squires, "Instrumentation Electronics for an Integrated Electrophysiology Data Acquisition and Stimulation System," Master's Thesis, Western Michigan University, 2013, Committee members: B. Bazuin, D. A. Miller (Chair), and F. L. Severance; Contributors: K. Batzer. and J. Stahl.

[16] "In vitro MEA-systems," Multi Channel Systems MCS GmbH, Reutlingen, Germany, Internet, [March 22, 2013]. [Online]. Available: http://www.multichannelsystems.com/products/vitro-mea-systems.

[17] R. Blum, J. Ross, E. Brown, and S. DeWeerth, "An integrated system for simultaneous, multichannel neuron stimulation and recording," *IEEE Transactions on Circuits and Systems*, vol. 54, no. 12, pp. 2608-2618, December 2007.

[18] N. Kladt, U. Hansklik, and H.-G. Heinzel, "Teaching basic neurophysiology using intact earthworms," *The Journal of Undergraduate Neuroscience Education*, vol. 9(1), pp. A20-A35, 2010.

[19] "Digilent nexys2 board reference manual," Digilent, Inc., Pullman, WA, Reference Manual 502-107, June 2008. [Online]. Available: http://www.digilentinc.com/Data/Products/NEXYS2/Nexys2 rm.pdf.

[20] "AD7606 data sheet rev c," Analog Devices, Inc., Norwood, MA, Data Sheet Rev. C, February 2012. [Online]. Available: http://www.analog.com/static/imported-files/data_sheets/AD7606_7606-6_7606-4.pdf

[21] "AD5678 data sheet rev c," Analog Devices, Inc., Norwood, MA, Data Sheet Rev. C, February 2011. [Online]. Available: http://www.analog.com/static/imported-files/data_sheets/AD5678.pdf

[22] "Async/Page/Burst CellularRAM 1.5," Micron Tech., Inc., Boise, ID, Data Sheet Rev H, August 2007. [Online]. Available:

www.micron.com/~/media/Documents/Products/Data%20Sheet/DRAM/Mobile%20DRAM/PSRAM/128mb_burst_cr1_5_p26z.pdf

[23] “EZ-USB FX2LP USB microcontroller high-speed USB peripheral controller," Cypress Semiconductor Corp., San Jose, CA, Data Sheet 38-08032 Rev. *W, July 2013. [Online]. Available: http://www.cypress.com/?docID=45142

[24] “The McGill physiology virtual laboratory, compound action potential, background, recording technique," McGill University, Montreal, Quebec, Internet, 2005, [August 20, 2013]. [Online]. Available: http://www.medicine.mcgill.ca/physio/vlab/CAP/recording.htm

[25] “Chloriding Ag/AgCl electrodes disk, pellet, or wire," Warner Instruments, Hamden, CT, Technical Reference, February 2004. [Online]. Available: http://www.warneronline.com/Documents/uploader/Chloriding%20Ag-AgCl%20electrodes%20%20%282004.02.02%29.pdf

[26] “GitHub Repository – KyleBatzer – Thesis” [Online]. Available: https://github.com/KyleBatzer/Thesis

[27] “Dynamic Data Display” [Online]. Available: http://dynamicdatadisplay.codeplex.com/