DS-94 Oscilloscope User Manual

Version 1.0

Contents

Preface

License and Warranty

The entirety of this product (including the hardware and design) is released under the open-source MIT License, which is reproduced below:

General Safety

Use the Correct Power Cord. Use only the included 5x Power Supply cord to power the product. Make sure this cord works for your country.

Ground the Oscilloscope. The oscilloscope's ground should be connected to earth ground to avoid electric shock. Ensure that the probe's ground terminal is connected to the product's ground.

Avoid Circuitry. Do not touch exposed board when product is powered.

Leave Display Connected. Do not needlessly connect and disconnect the display. This will cause strain on the board and also increases the probability of connecting the display wrong which could destroy the system.

Do Not Operate in Wet/Damp Conditions. Keep product dry at all times.

Conventions

Throughout this manual, menu options are represented in Camel Case (for example: Trigger Level, Trigger Slope) and specific buttons are represented in all caps (for example: MENU, TOGGLE, RESET).

Getting Started and Installation

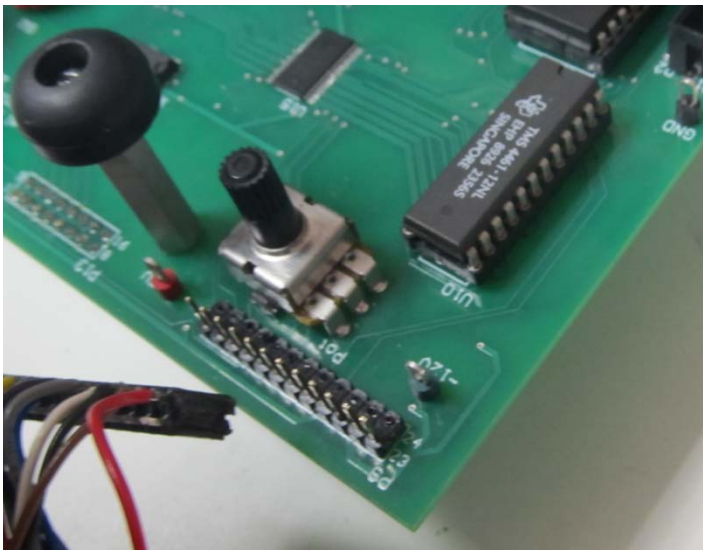
Installation

Installation of this product consists of two steps: connecting the display and connecting the power supply. These steps should be performed in that order.

Connecting the Display

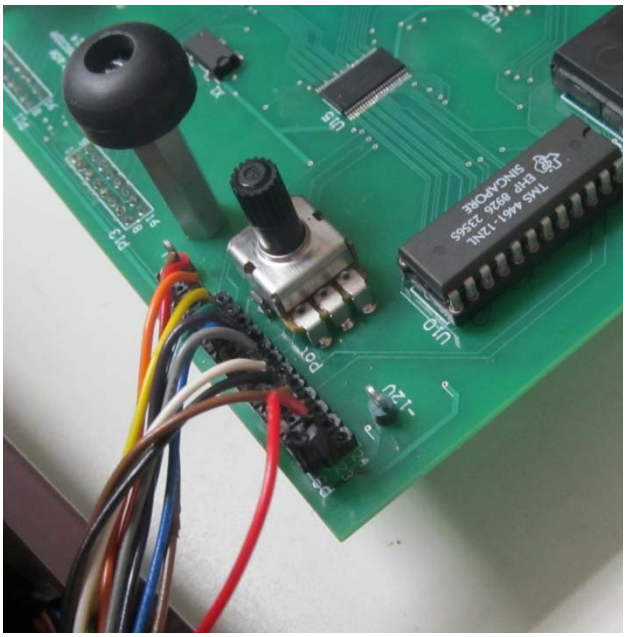
Important: Make sure power is turned OFF when connecting the display.

The board has a 12x1 header pin connector in the top-left corner. This connector is shown below.

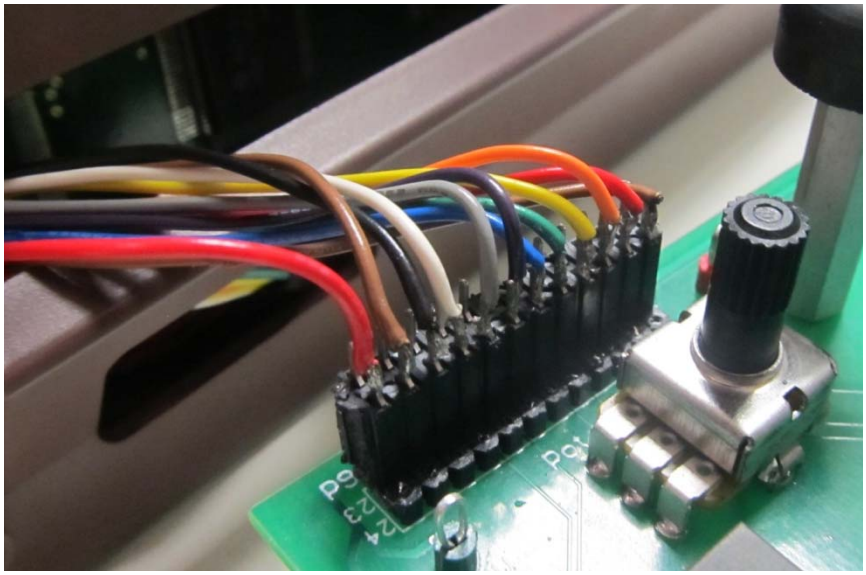


Display Connector on Board

The display has a 12x2 female header as a connector. This female connector can be connected to the male header on the board by firmly pressing it in. Hold on to the bottom of the board while doing this to avoid snapping the board. When plugged in, the black wire (ground) should be closer to both edges of the board. This is shown below in two views to clarify.



Proper Connection for Display Connector, View 1



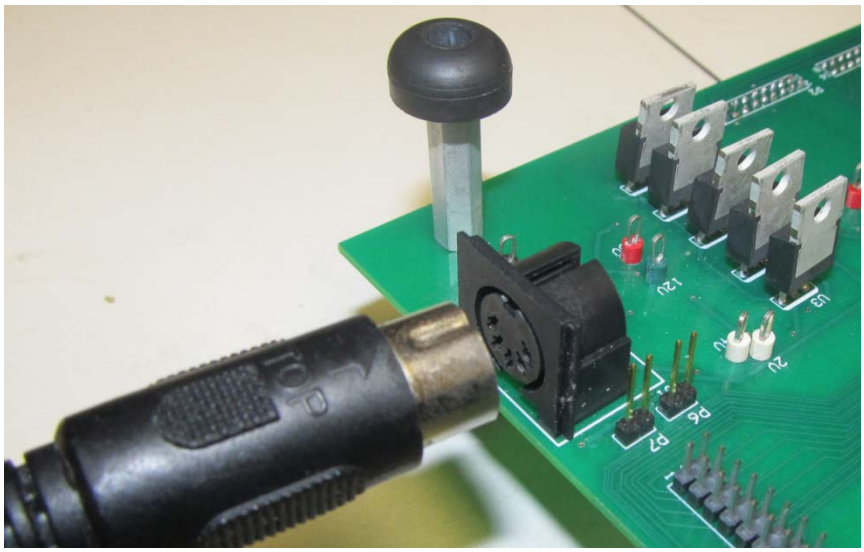
Proper Connection for Display Connector, View 2
In order from left to right: RED, BROWN, BLACK, WHITE, GRAY, PURPLE, BLUE, GREEN, YELLOW, ORANGE, RED, BROWN.

Important. Ensure that this is the orientation of the connector before powering the board. Permanent board or display damage could occur otherwise.

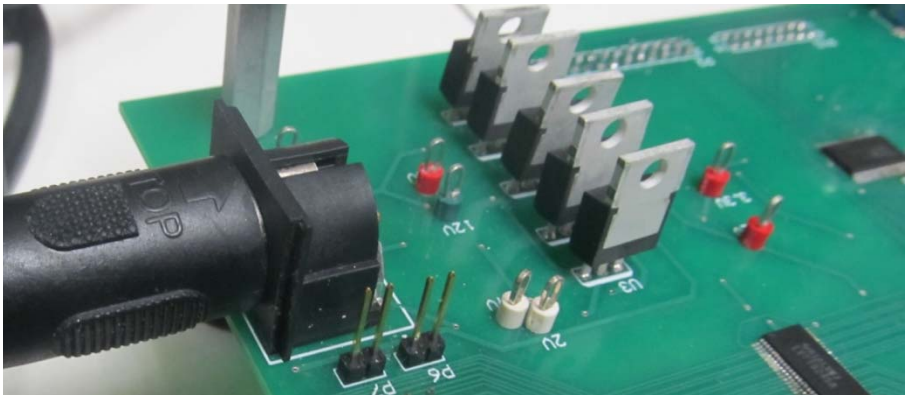
Connecting Power Supply

It is suggested to connect the power supply to the board before connecting it to power.

The power supply connector has only one orientation. It should slide into the connector as shown below.



Power Supply Disconnected

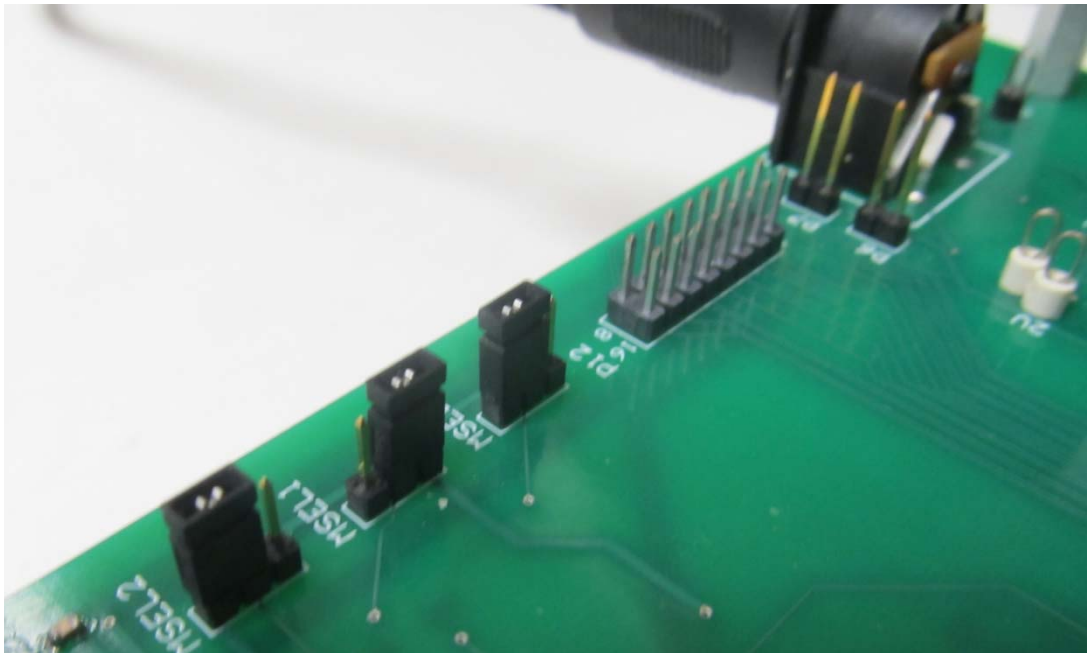


Power Supply Connected

Simple Installation Check

The oscilloscope will start functioning a second after being powered. A set of axes and a menu should show up on the display when this happens. If this does not occur within a few seconds, disconnect power immediately and ensure the previous steps were followed correctly.

If all previous steps were correctly performed, then connect power again and turn the potentiometer (knob near display connector shown in display connection instructions) until the axes become visible. If this does not occur at all, then check the configuration pins. These should be set as shown below. Restart the system if this was not the case.



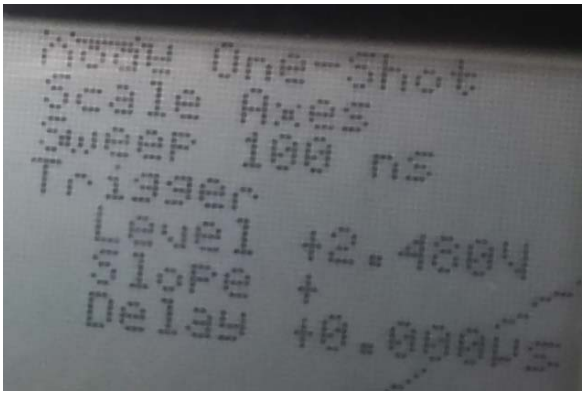
Correct Configuration Pin Settings

If this does not solve the problem, the board is likely defective; see a specialist to further assess the problem.

Basics of Operation

Menu Description

The DS-94 Oscilloscope uses a menu-based interface to control system parameters. The menu appears in the top right corner of the display and is shown below.



Menu

The possible settings are:

1. Mode [Normal, One-Shot, Automatic] – The mode of the scope. Normal mode will only sample when triggered, one-shot mode will sample only once on a trigger, and automatic will sample when triggered or when a 0.1s counter times out.
2. Scale [Axes, None, Grid] – What scale to show on the display. Axes gives a set of axes with markers, none shows only the waveform, and grid will show a grid to ease measurement.
3. Sweep [Range of Values, 100ns to 20ms] – Specifies how often to take a sample from the analog signal (see system specs for full range of values).
4. Trigger – Moving left or right here causes the one-shot mode to trigger again.
    a. Level – Specifies the trigger level from 0 V to 4 V in steps of 31 mV.
    b. Slope – Specifies the trigger slope, either positive (+) or negative (-).
    c. Delay – Specifies the horizontal trigger delay from 0s to 50ms.

Startup Configuration

The system starts up with the following settings:

- Normal Mode
- Axes
- Sweep of 100 ns
- Positive, mid-level trigger (2 V).
- Delay of 0 seconds

Using Keys to Navigate the Menu

The system has two keys (pushbuttons) and an encoder. Their functions are listed below:

1. MENU – Toggles the menu on and off. This is useful when the menu is obscuring the waveform.
2. TOGGLE – Toggles the directionality of the rotary encoder. This key does not need to be held down in order to do this. The system starts toggled.
3. Encoder [LEFT, RIGHT, UP, DOWN] – Moving the encoder left or right when the system is not toggled will change the value of whatever is currently selected on the menu – equivalent to a LEFT/RIGHT key press. Moving the encoder left or right when system is toggled changes the currently selected menu option – equivalent to an UP/DOWN key press. Because the system starts toggled, moving the encoder after startup will change the currently selected menu item until the toggle key is pressed.

The keys are shown below.

Keys & Encoder

The white key is the MENU key. The TOGGLE key is activated by pushing down on the encoder, which should result in a clicking noise. The encoder is also shown.

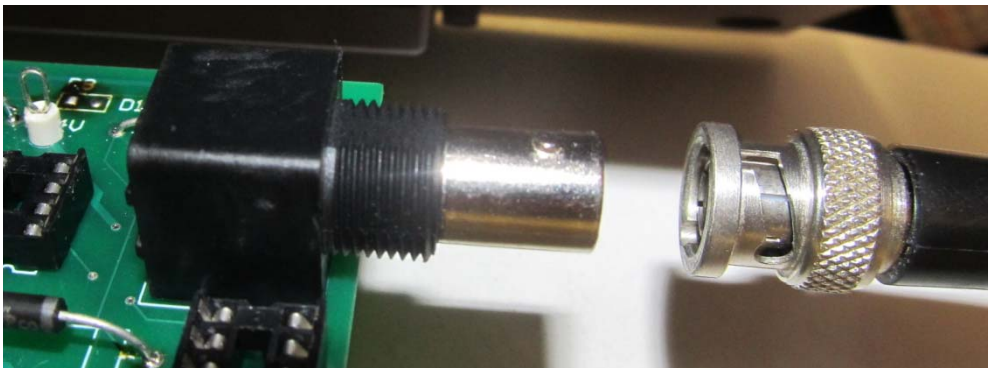
Resetting the System

If for any reason the scope seems to not be functioning correctly, press the red RESET key to reset the system. This is equivalent to the hardware as a power off, but it is safer and less damaging. The RESET key can be found near the middle of the board as shown below.

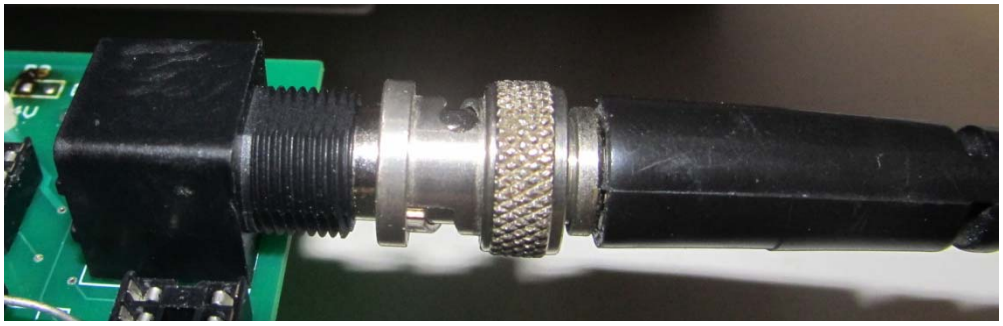
RESET key

Capturing a Trace

If the system has been set up as stated in the installation section of this manual, then it is ready to start sampling an analog source. In order to do this, connect a probe, function generator, or some other signal source to the BNC connector on the board. This is shown below.



BNC connector unconnected



BNC connector connected

Important. Do not connect a signal to the board if it is not powered. This could harm the hardware.

The system will start sampling as soon as it gets a trigger (or the automatic trigger times out if used in Automatic Mode). Thus, if no trace is visible and the Mode setting is in Normal Mode, try changing the trigger level or setting it to Automatic Mode to find the sign.

Appendix

System Specifications

- Input Range: 0 to 4 Volts
- Sampling Frequencies: 100 ns, 200 ns, 500ns; 1 us, 2 us, 5 us, 10 us, 20 us, 50 us, 100 us, 200 us, 500 us; 1 ms, 2 ms, 5 ms, 10 ms, 20ms
- Sampling Resolution: 8 bits
- Trigger Level Resolution: 7 bits
- Positive or Negative Trigger Slope
- Horizontal Delay: Zero to 50000 samples (16 bits of resolution software-limited to 50,000).
- Constant holdoff of 0.25 seconds.
- Graphics LCD with resolution of 480 x 240 pixels

Measurement Error Considerations

The DS-94 Oscilloscope is a measuring tool, but it can't measure voltages perfectly. There is both a resolution error from converting the analog signal to a digital signal and a fixed error from hardware or software sources and noise.

The resolution is limited to 8 bits, and the bottom bit is considered noise so it's thrown out. Thus, one cannot expect more than 7 bits of resolution. The input range is from 0 Volts to 4 Volts, so the highest resolution one could possibly expect is in the order of 31mV.

In reality, this resolution is unrealistic because of noise and other conversion errors. These errors were not measured, and they will be considered in future editions of this manual. However, please keep them in consideration and consider measuring them if accuracy is critical to the application of the DS-94 Oscilloscope.

DS-94 Oscilloscope Technical Manual

Version 1.0

Contents

Table of Figures

Preface

License and Warranty

The entirety of this product (including the hardware and design) is released under the open-source MIT License, which is reproduced below:

General Safety

Use the Correct Power Cord. Use only the included 5x Power Supply cord to power the product. Make sure this cord works for your country.

Ground the Oscilloscope. The oscilloscope's ground should be connected to earth ground to avoid electric shock. Ensure that the probe's ground terminal is connected to the product's ground.

Avoid Circuitry. Do not touch exposed board when product is powered.

Leave Display Connected. Do not needlessly connect and disconnect the display. This will cause strain on the board and also increases the probability of connecting the display wrong which could destroy the system.

Do Not Operate in Wet/Damp Conditions. Keep product dry at all times.

Conventions

A "module" is defined as a group of hardware components that perform a specific function.

Component designators on the board are shown in bold inside of square brackets. For example, the JTAG Connector is shown as [**P1**] and the VRAM Chip is shown as [**U10**].

Voltages are labeled in the schematics and other documents as shown below.

Voltage	Name
1.25 V	VCCInt
2.5 V	VCCA
3.3 V	VCCIO
5 V	VCC5, VCCAnalog
12 V	VCC12
-12 V	VCC-12

Table 1 - Names Given to Various Voltage Levels

Related Documentation

See the User's Manual for instructions on setting up and interfacing with the system and notes on system specifications.

Appendices

Portions of the Altium schematics and Quartus block diagram files are shown throughout the document. The appendices show the complete files, as well as other information. Here is a list of the appendices; these can be found in order at the end of this document.

Hardware Overview

Block Diagram

This section illustrates the high-level interconnectivity of the hardware components on the DS-94 Oscilloscope board through two diagrams, one for modules inside the FPGA and one for external hardware.

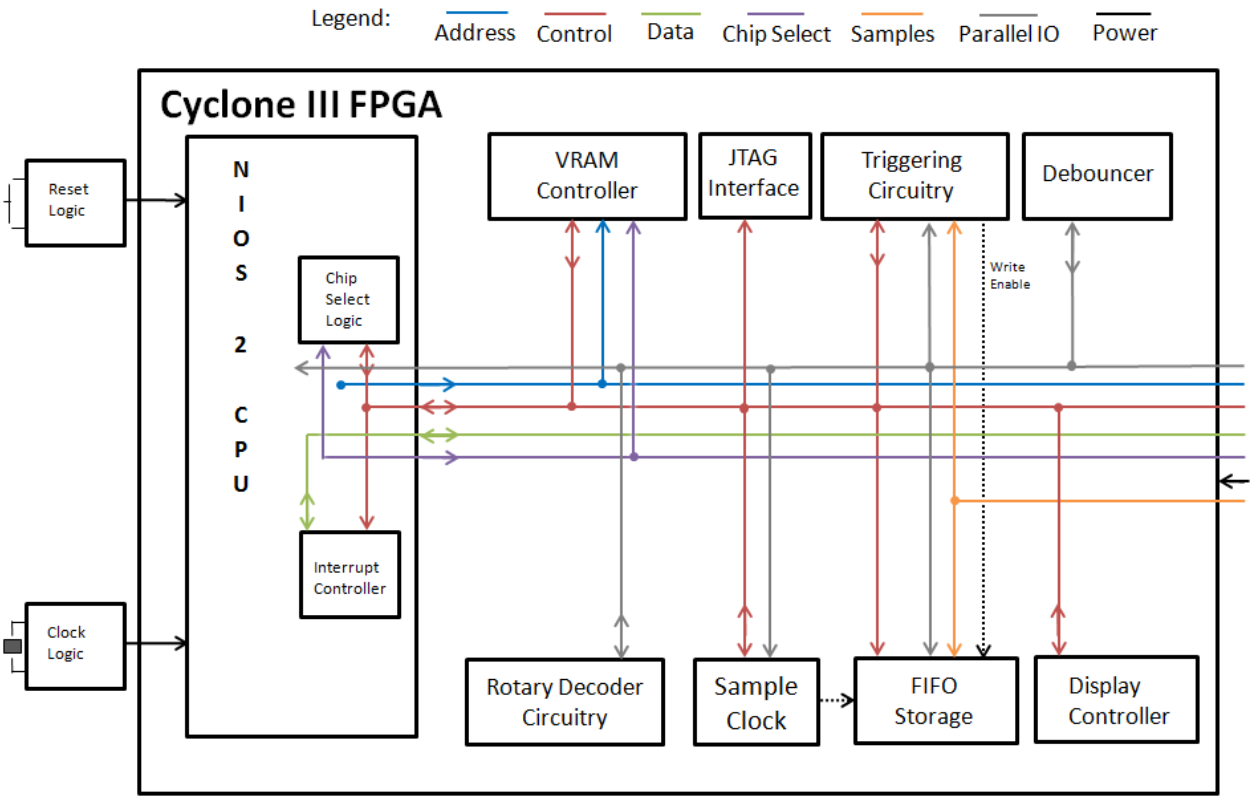


Figure 1 - High Level Hardware Block Diagram of FPGA Components

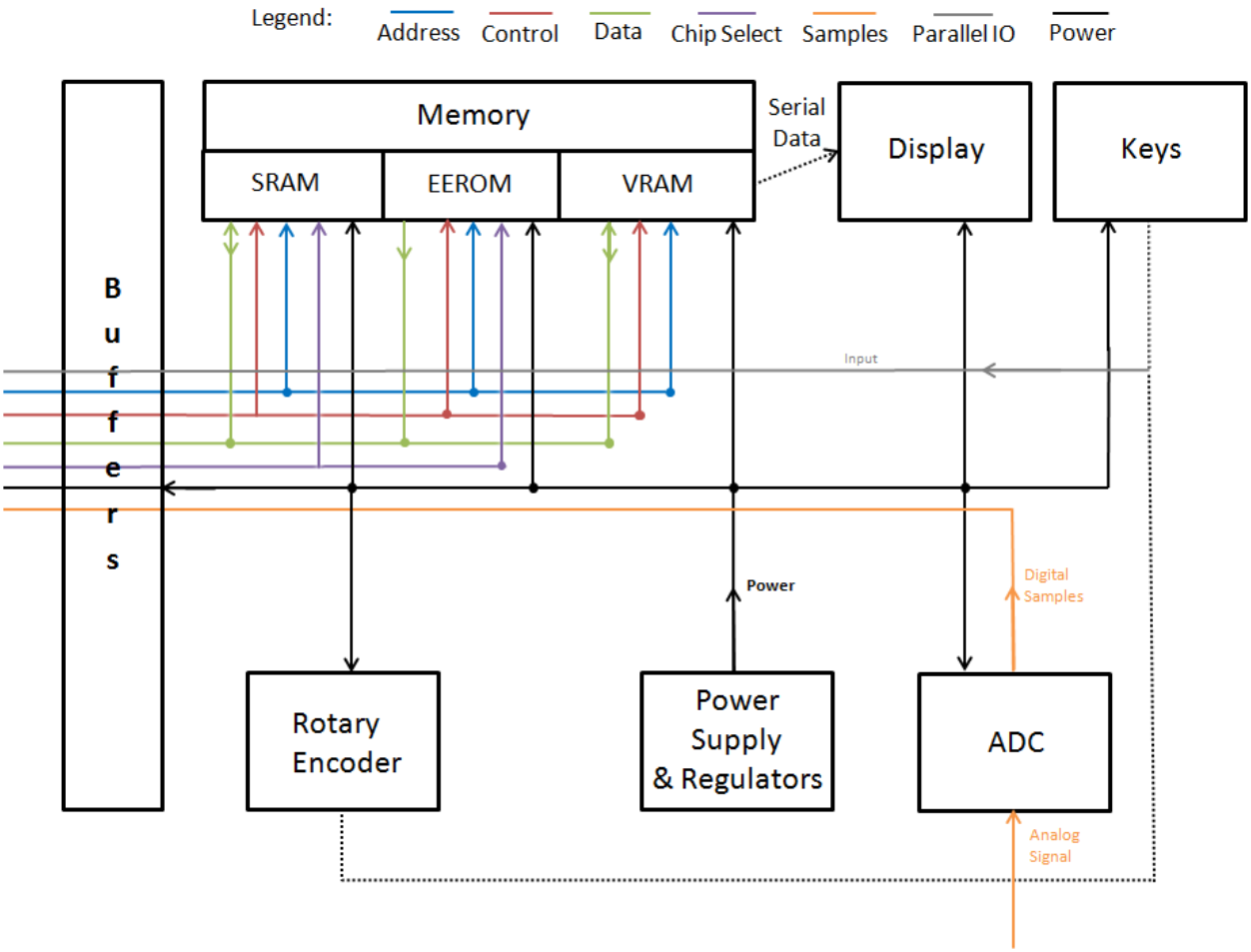


Figure 2 - High Level Hardware Block Diagram of External Components

System Overview

General Modules

Cyclone III FPGA.

The FPGA is used to house the CPU (Nios 2) as well as generate the logic to interface with the rest of the hardware.

Nios 2 CPU.

The CPU interacts with the hardware and runs the code. It is interrupted by the Debouncer (in the case of key presses), by the Rotary Encoder Decoder (in the case of the Rotary Encoder moving), and the FIFO Memory when it is full and the samples need to be processed.

JTAG Interface.

The JTAG interface is used to program the FPGA when it is in programmable mode. It sends the necessary control signals to an external connector which can be connected to a computer through a serial or USB interface with the necessary hardware.

Reset Circuitry.

The reset circuitry resets the FPGA configuration settings and reloads the FPGA configuration and code of the system.

Clock Logic.

The system uses a 24MHz clock box as the system clock, which clocks everything except the FIFO Memory, which is clocked at different rates specified by the user.

Buffers.

The buffers buffer every active (used in design or left as general IO) FPGA pin in case extra current is needed and to protect the FPGA.

Power Suppy & Regulators.

The Power Supply takes standard 120 Volts and converts it to 5 Volts, 12 Volts, and -12 Volts. Regulators then convert this to the necessary voltage to power the system, namely: 5 V, 3.3 V, 2 V, 4 V, and 1.25 V.

Serial Configuration Device.

The serial configuration device is a ROM device that holds the FPGA configuration information. This chip is used to program the FPGA when the system turns on.

Memory Modules

SRAM.

The SRAM is used to store software data and configuration information. It receives address and control signals from the CPU.

EEROM.

The EEROM is used to store the system's software. It receives address and control signals from the CPU.

VRAM Controller.

The VRAM controller is a state machine implemented in the FPGA that generates the necessary control signals to interface with the VRAM. It receives control and address information from the CPU and uses this to generate these signals.

VRAM.

The VRAM is used to store display information. It receives address and control information generated in the VRAM controller. It also receives a serial clock control signal from the display controller to determine when to send serial data to the display.

Display Modules

Display Controller.

The display controller generates the necessary clocks to display pixels on the display. It receives control information from the VRAM (an acknowledge signal) and sends control signals to the display.

Display.

The display receives information from the display controller and serial data from the VRAM and uses this to draw pixels.

Sampling Modules

Triggering Circuitry.

The triggering circuitry analyzes the incoming signal to determine when a trigger event should occur. It receives information via parallel IO to determine the current

trigger setting of the system and it controls when the signal should be stored into the FIFO Memory.

Sample Clock.

The sample clock receives sweep rate information from the CPU and uses this to generate an appropriate clock which is used to determine when to store samples in the FIFO Memory.

FIFO Storage/FIFO Memory.

The FIFO (First In, First Out) memory is used as a temporary storage for incoming samples. When the FIFO is enabled by the triggering circuitry, it stores 480 samples, and, when full, sends this information to the CPU for processing and displaying the trace. The rate at which the FIFO is filled is determined by the Sample Clock. The FIFO memory is not written to when the CPU is reading it. The FIFO Memory interrupts the CPU via parallel IO when it is full. The rate at which the FIFO is read is determined by the CPU (via Parallel IO) using a read clock control signal.

ADC.

The ADC (Analog to Digital Converter) is used to convert the analog waveform into digital samples. The ADC is always clocked at the maximum sample rate (12 MHz).

User Input Modules

Keys.

There are two keys: the menu key and the toggle key. These keys send their status (pressed or not pressed) to the Debouncer in the FPGA.

Rotary Encoder.

The encoder sends its status to the Debouncer in the FPGA.

Debouncer.

The Debouncer debounces the Keys and Rotary Encoder. The debounced key and encoder output is sent to the CPU (via a parallel IO interrupt) in the case of the keys and to the Rotary Encoder Decoder in the case of the rotary encoder.

Rotary Encoder Decoder.

The rotary encoder decoder determines the status of the encoder (moving left, moving right, or not moving) by looking at the debounced output of encoder and sends this information to the CPU via a parallel IO interrupt.

Board Signal Layout

This section describes how each module is mapped to the DS-94 Oscilloscope board. The signal flow is shown as in the Block Diagram section, with the exception of chip select signals being merged into control signals and having IO lines merged into data lines.

There are four different diagrams, each of which describes the flow of one of the below:

- Address
- Control
- Data and IO
- Power

Please refer to these diagrams or the PCB layout diagram in Appendix B to know where things are located on the board.

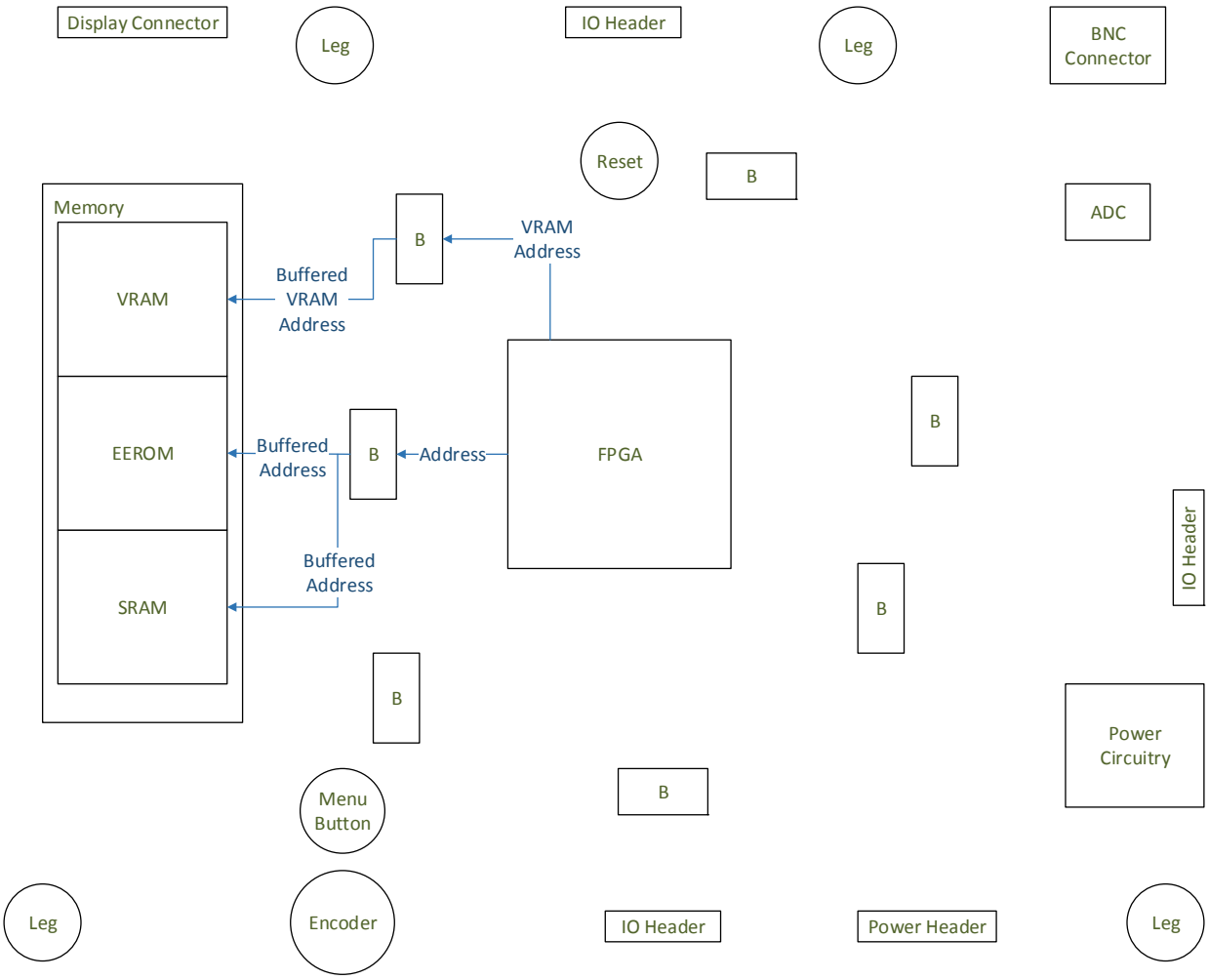


Figure 3 - Board Layout with Address Lines Labeled

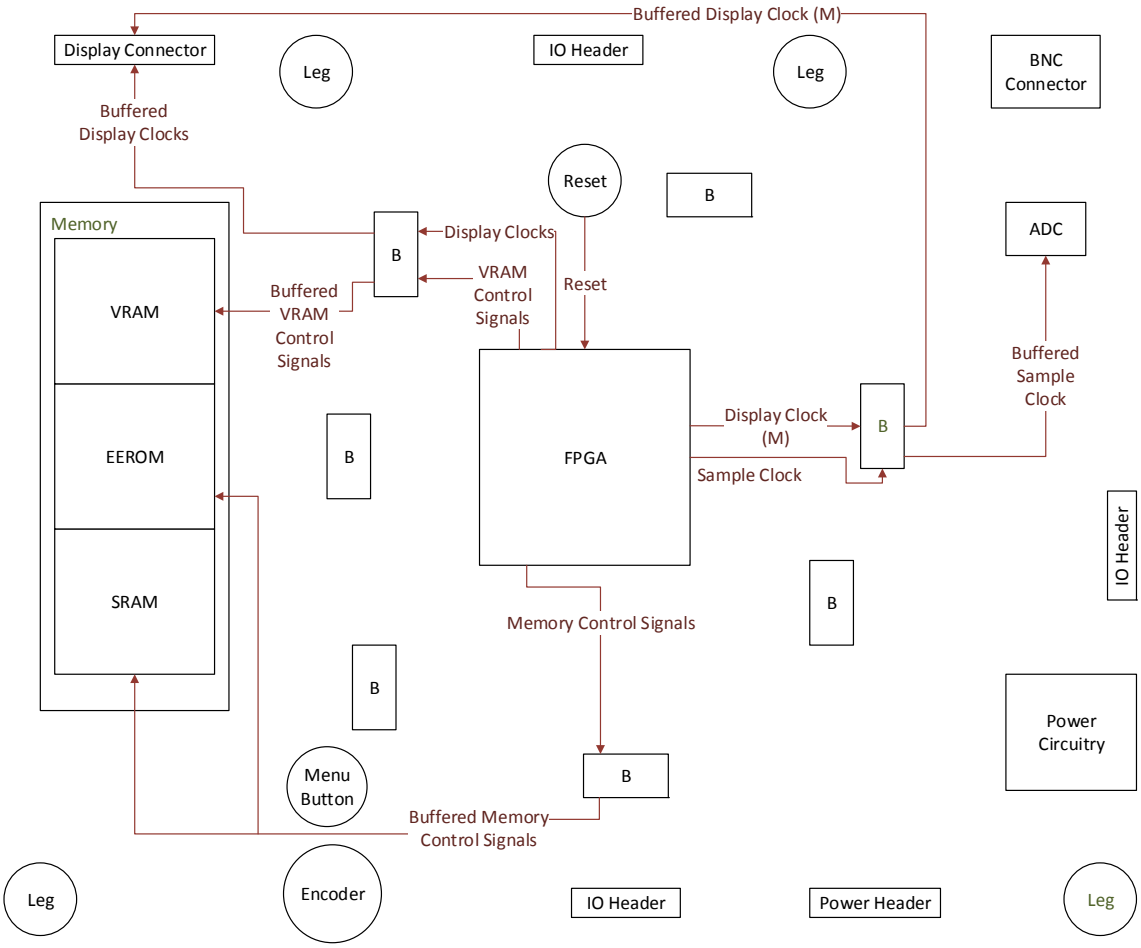


Figure 4 - Board Layout with Control Signals Labeled

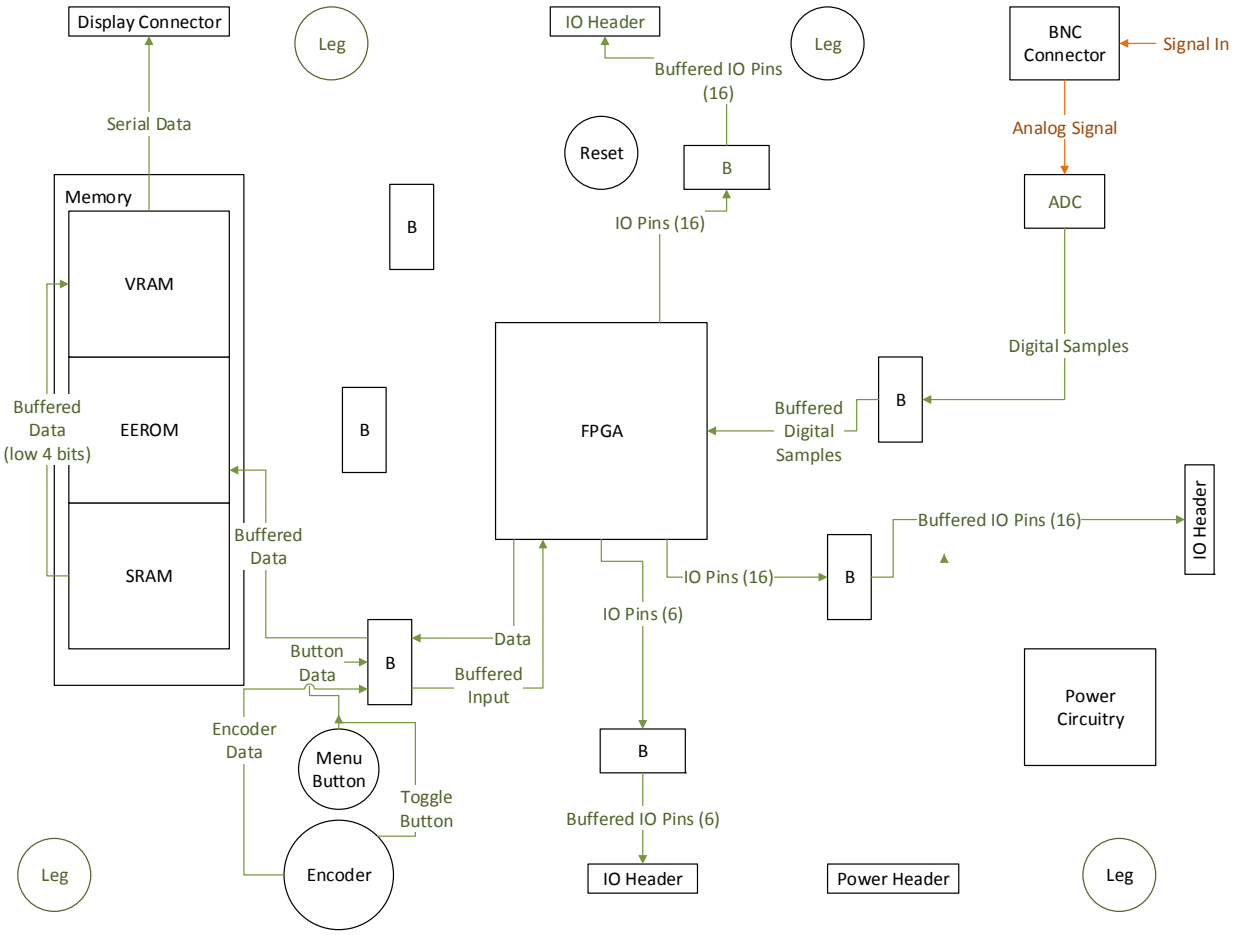


Figure 5 - Board Layout with Data and IO Lines Labeled

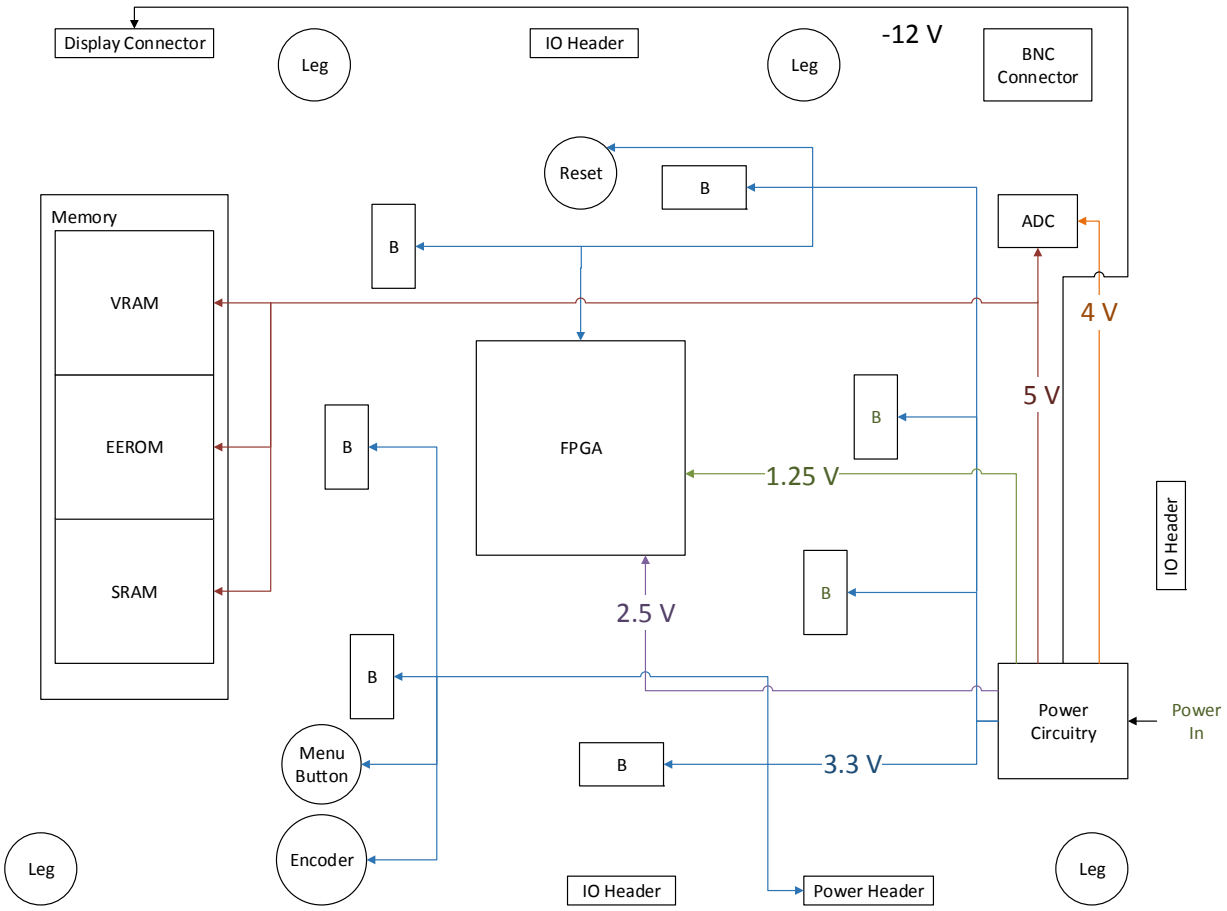


Figure 6 - Board Layout with Power Rails Labeled

Note that, unlike the other diagrams, the lines here just represent which voltages go to each device, and are not indicative of where the rails are actually located on the board.

Memory Map

The memory map describes where each device is mapped in the address space. There may be multiple mappings, but only the assigned mappings of each device are shown. This memory mapping information is also shown in the memory map section of the CPU Datasheet attached in Appendix D.

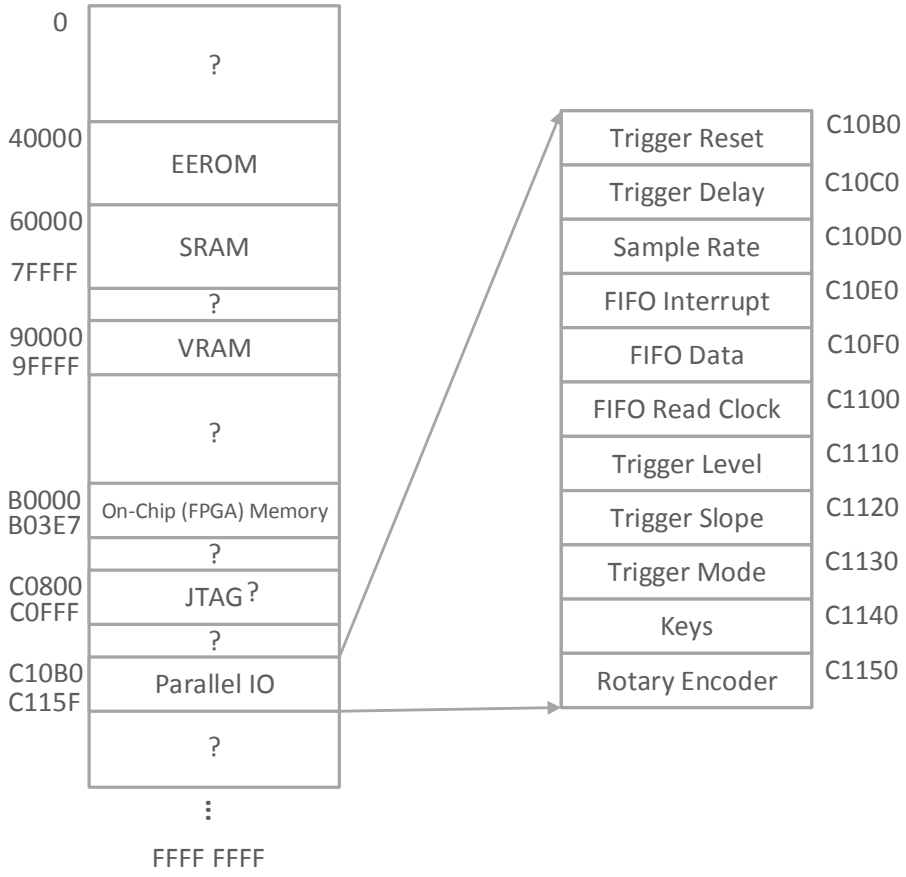


Figure 7 - System Memory Map for Assigned Base Addresses

Detailed Hardware Description

This section describes each module in detail. This includes both a description of the signals that involve that module and the timing on those signals, as well as any other relevant information on that module.

CPU

The system uses an Altera NIOS 2 processor configured in a standard way through the Altera Qsys interface provided in the Quartus software. The fast processor was chosen to improve speed but the data cache was disabled to avoid graphics problems that occurred when reading and writing to VRAM because of it.

The CPU also has three memory interfaces and also built-in, on-chip memory. The on-chip memory (**onchip_memory2_0**) was used to hold the reset and interrupt vectors when the system was built, and it is only 1000 bytes of memory. It is not used in the final project because everything on it was moved to the ROM or SRAM. The three memory interfaces are used to interface with the SRAM (**generic_tristate_controller_0**), the ROM (**generic_tristate_controller_1**) and the VRAM (**generic_tristate_controller_2**). The outputs of these controllers are processed in the pin sharer (**tristate_conduit_pin_sharer_0**) and the bridge (**Bridge**) as specified in the Altera manuals and are then sent out directly to the SRAM chip, the ROM chip, or the VRAM controller and chip, depending on what has happened in the system.

The CPU also has 11 parallel IO ports used to interface with external hardware in the FPGA. These PIO ports are described in more detail in the table below.

Qsys Name	Output Name	Bus/Line Name in Quartus	Interrupt Line	Input or Output	Description
pio_0	p0	p0_export[1..0]	0	Input	Key inputs
pio_1	p1	p1_export[1..0]	1	Input	Rotary encoder inputs
pio_2	trig_auto	auto	-	Output	Bit indicating trigger mode
pio_3	trig_slope	ts	-	Output	Bit indicating trigger slope
pio_4	trig_level	trig_level[7..0]	-	Output	Trigger level output
pio_5	rdclk	rdclk	-	Output	Read clock (to FIFO)
pio_6	fifo_out	fifo_out[6..0]	-	Input	Data input from FIFO
pio_7	fifo_intr	fifo_intr	2	Input	Service interrupt from FIFO
pio_8	smpl_rate	smpl_rate[17..0]	-	Output	Sample rate output
pio_9	delay	delay[15..0]	-	Output	Trigger delay output
pio_10	smpl_start	trig_reset	-	Output	Trigger reset output

Table 2 - Description of CPU Parallel IO Ports

This is an overview of the CPU. For more detailed information, please reference the CPU datasheet in Appendix E or the CPU connection diagram among the Quartus Block Diagrams in Appendix D.

FPGA Configuration

The FPGA configuration pins were set up so that the FPGA could be programmed through the JTAG interface. The pins were tied to power and ground via resistors in the schematic so that this configuration could be changed if necessary. In the end, all configuration pins except the MSEL pins were pulled high through 10KΩ resistors, including INIT_DONE, which had been left floating in the schematic.

None of these signals are buffered. The MSEL pins are all pulled low to GND to program the system, and MSEL1 is pulled high to VCCA when the system gets its configuration from the serial ROM device.

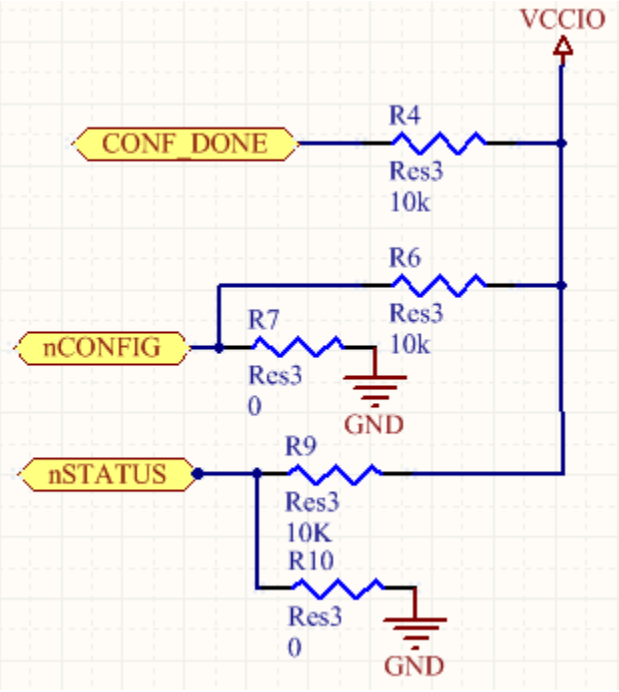


Figure 8 - FPGA Configuration Pins in the Schematic

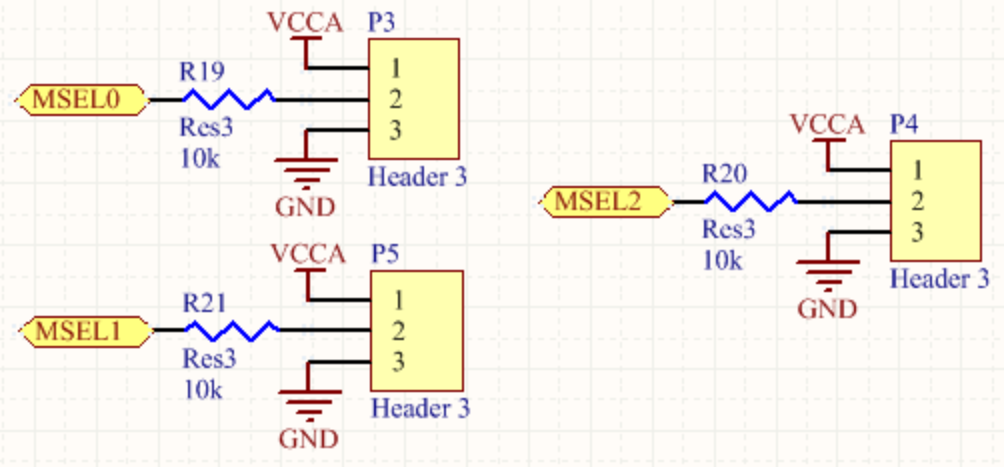


Figure 9 - MSEL Configuration Pins in Schematic

Clock

The schematic for the 24MHz clock box [**X1**] is shown below. The clock output goes straight into the FPGA without being buffered.

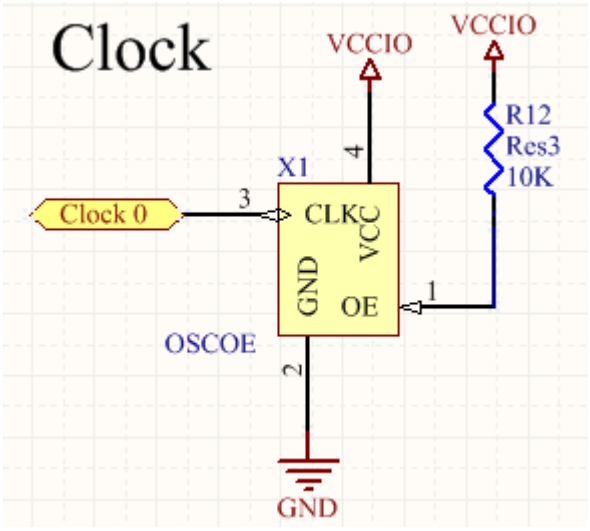


Figure 10 - Clock Schematic

Reset Circuitry

The system uses a MAX706AP (a 3 Volt version of the popular MAX705) chip as a reset chip. Despite this, the system was not configured to use any of the advanced inputs or outputs of the chip, such as the watchdog or the power-fail lines. However, the

RESET line was tied to CONF_DONE to reset the system whenever the button was pushed. The schematic for the reset circuitry is shown below.

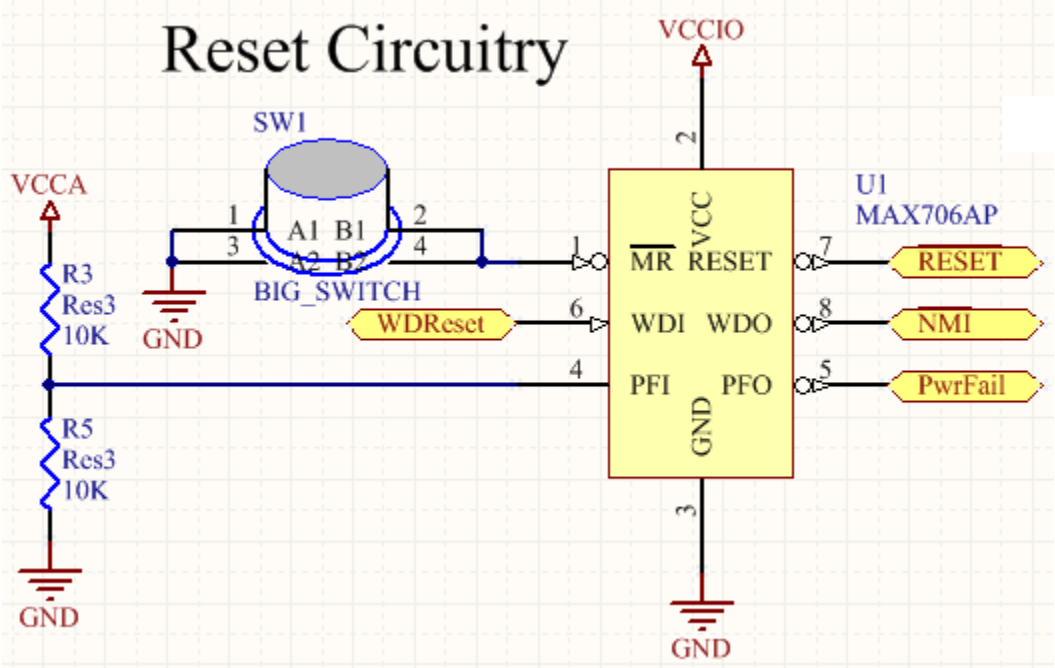


Figure 11 - Schematic for Reset Circuitry

JTAG Connector

The JTAG connector [**P1**] is used to program the FPGA from a computer by connecting an Altera ByteBlaster or USBBlaster board. It is a five by two header configured as shown below.

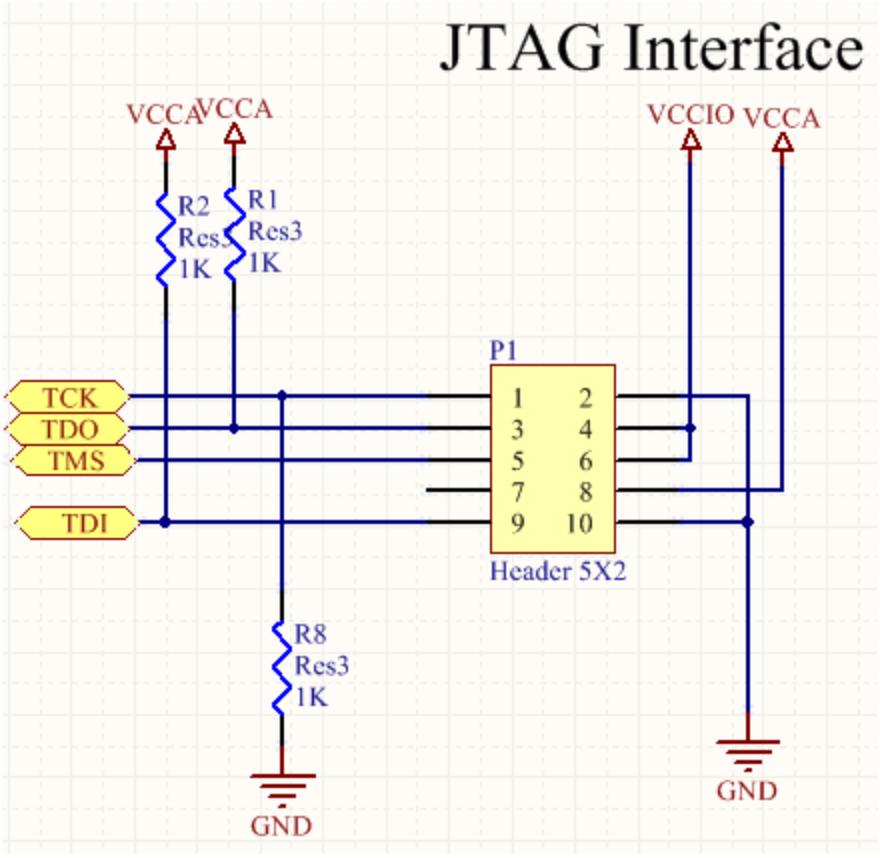


Figure 12 - JTAG Connector Header Configuration

The TCK, TDO, TMS, and TDI signals are control signals that come directly from the FPGA from the pins shown below. These signals are not buffered.

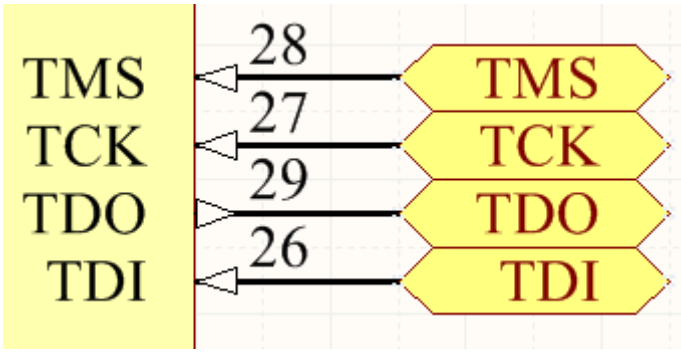


Figure 13 - JTAG Configuration Pins on FPGA

The header is laid out on the top of the board with pin one further away from the edge of the board. Pin seven is left floating.

Buffers

There are a total of seven buffers [**U12 – U18**]. These buffers are 74LVTH162245 chips. Two of these buffers are fully dedicated to general IO pins, whereas the rest are either dedicated completely to signals used in design or are a mix of used signals and general IO pins. The output enable signals on the buffers are set to always output for all buffers.

*Address Buffer [**U12**]:*

The address buffer buffers the address bus, with the exception of address lines sixteen through eighteen (a16 – a18), which are buffered by the static memory control signal buffer [**U13**]. The direction pins are set to constantly output, and the chip is always enabled. The schematic for this buffer is shown below.

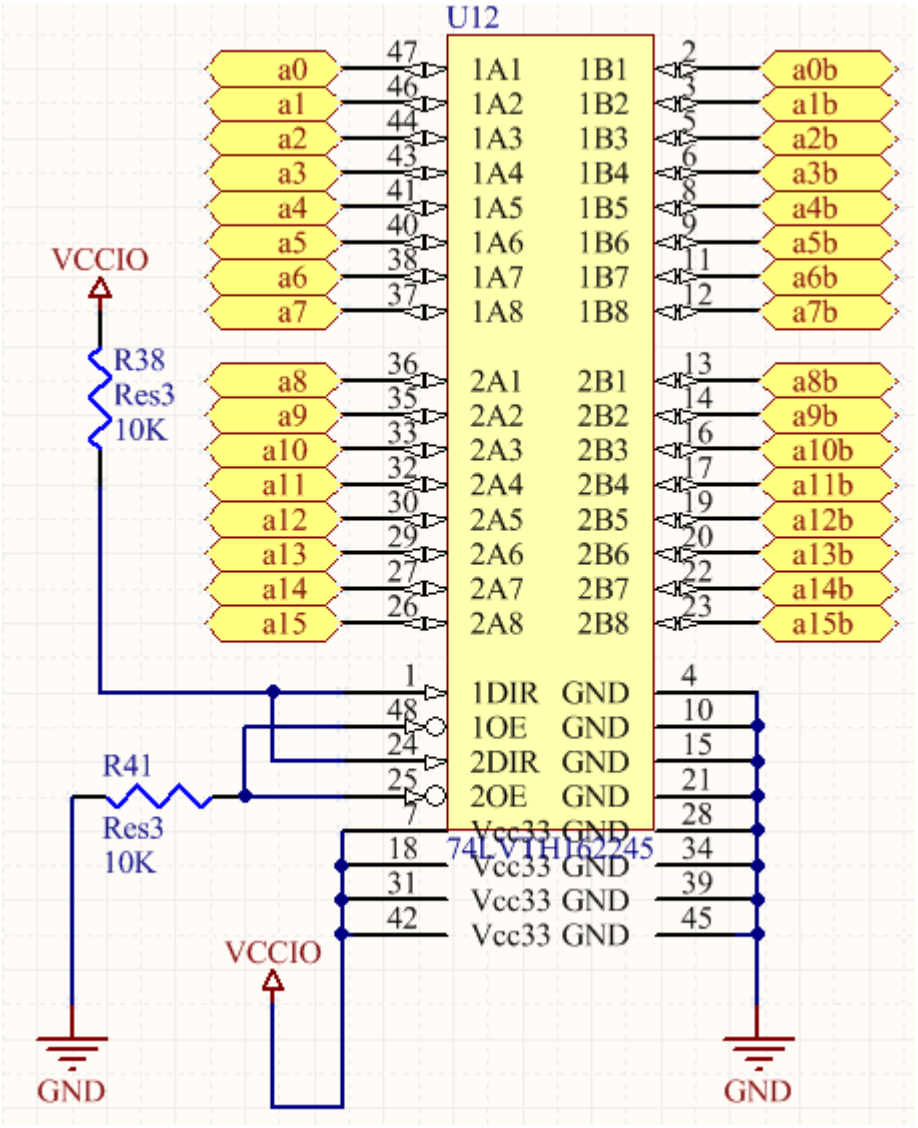


Figure 14 - Schematic for Address Buffer

*Static Memory Control Signal Buffer [**U13**]:*

The static memory controls signal buffer buffers control signals (chip select and read/write) for the SRAM and the EEROM. It also buffers six general IO pins. The direction pin for the control signals is set to constantly output, whereas the direction pin for the IO pins is controlled by another IO pin. The schematic for this buffer is shown below.

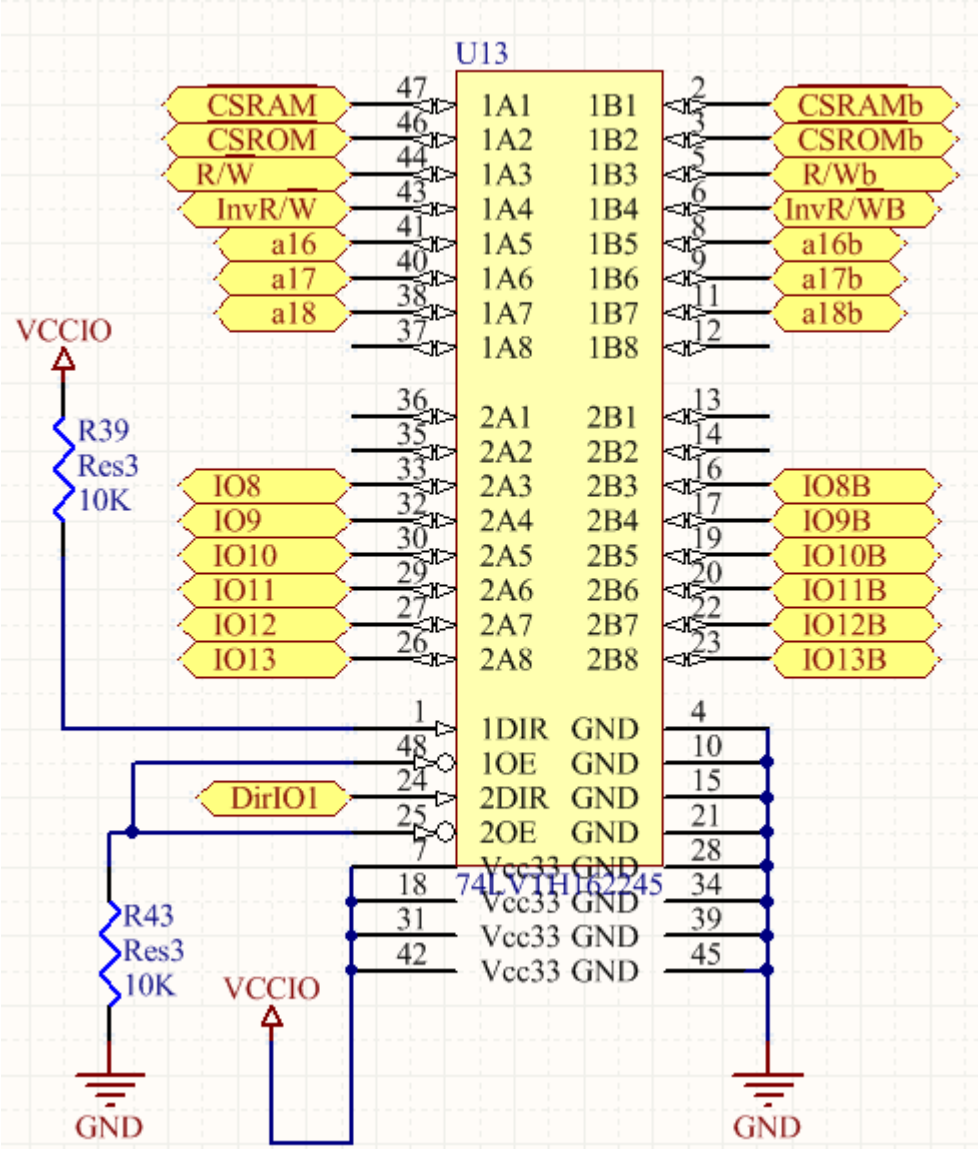


Figure 15 - Schematic for Static Memory Control Signal Buffer

*General IO Buffer One [**U14**]:*

This buffer buffers general IO signals from the FPGA. The direction of the buffer is determined by two IO signals from the FPGA. The schematic for this buffer is shown below.

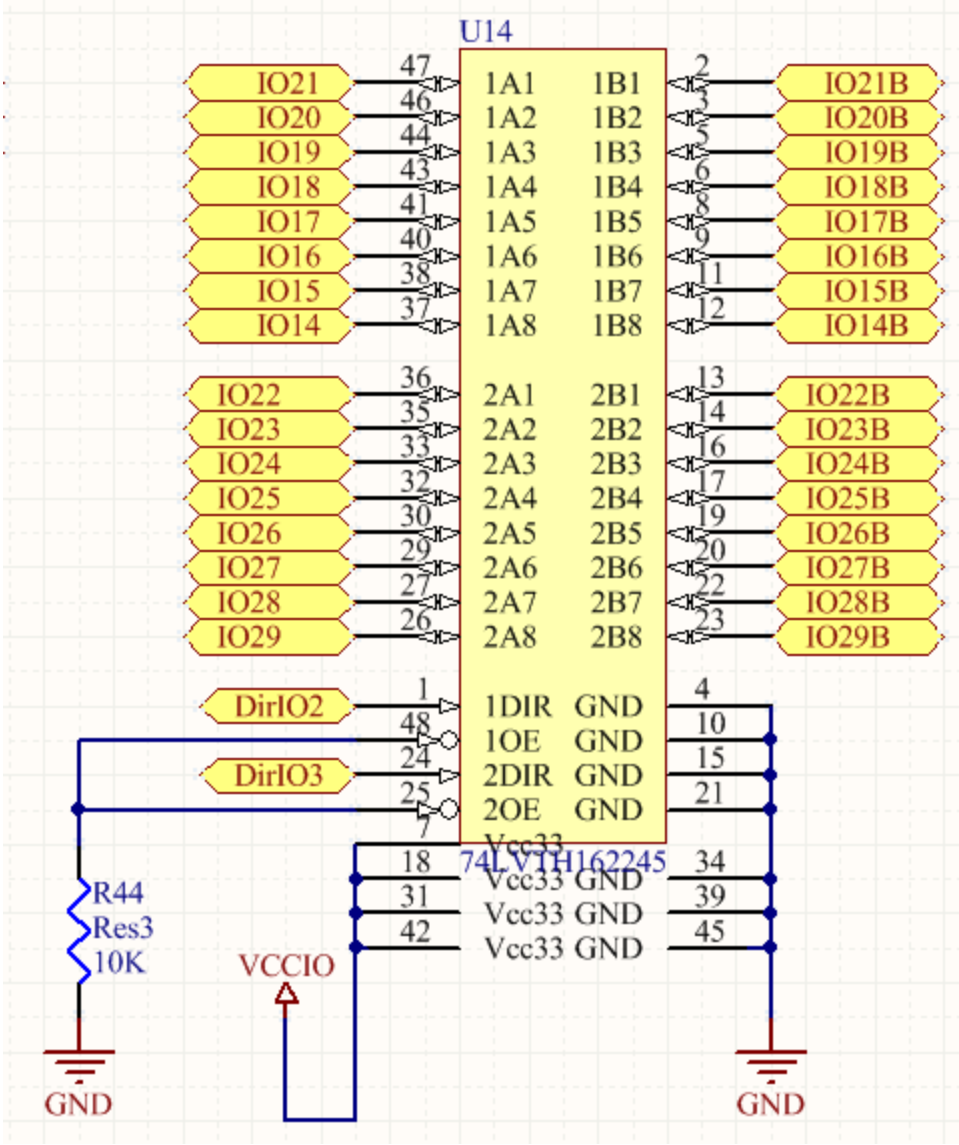


Figure 16 - Schematic for General IO Buffer One

*VRAM and Display Signal Buffer [**U15**]:*

The VRAM and display signal buffer buffers all of the VRAM signals (including address lines) and most of the display clocks. The only exception is the display M signal, which is buffered in the ADC Signal Buffer [**U16**]. Both direction lines are set to constantly output. The schematic for this buffer is shown below.

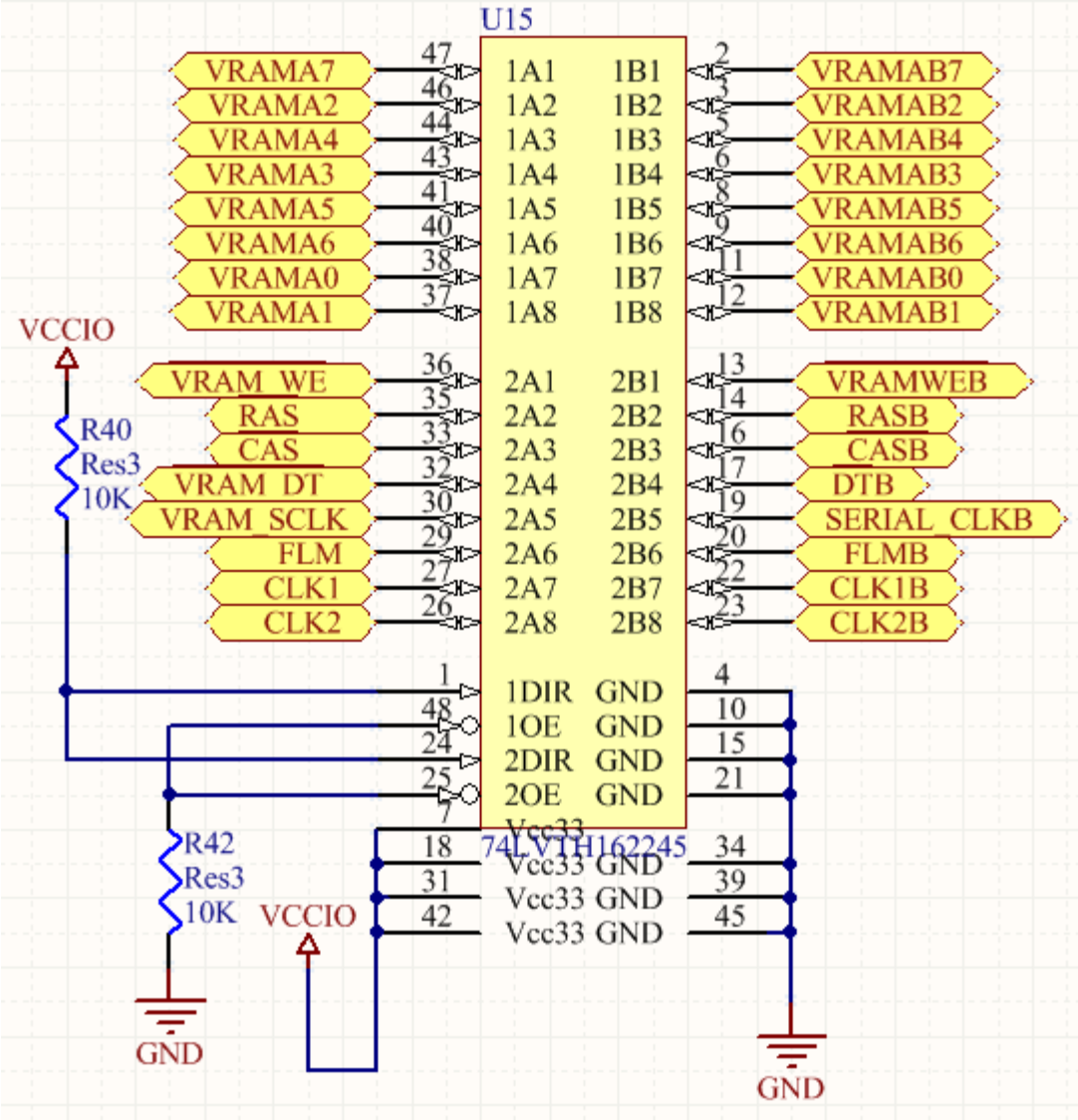


Figure 17 - Schematic for the VRAM and Display Signal Buffer

*ADC Signal Buffer [**U16**]:*

The ADC signal buffer buffers all the ADC signals (the digital samples and the sample clock) as well as a single display controls signal (M). The direction pin for the ADC samples is set to constantly input, whereas the direction pin for the other two signals is set to constantly output. The schematic for this buffer is shown below.

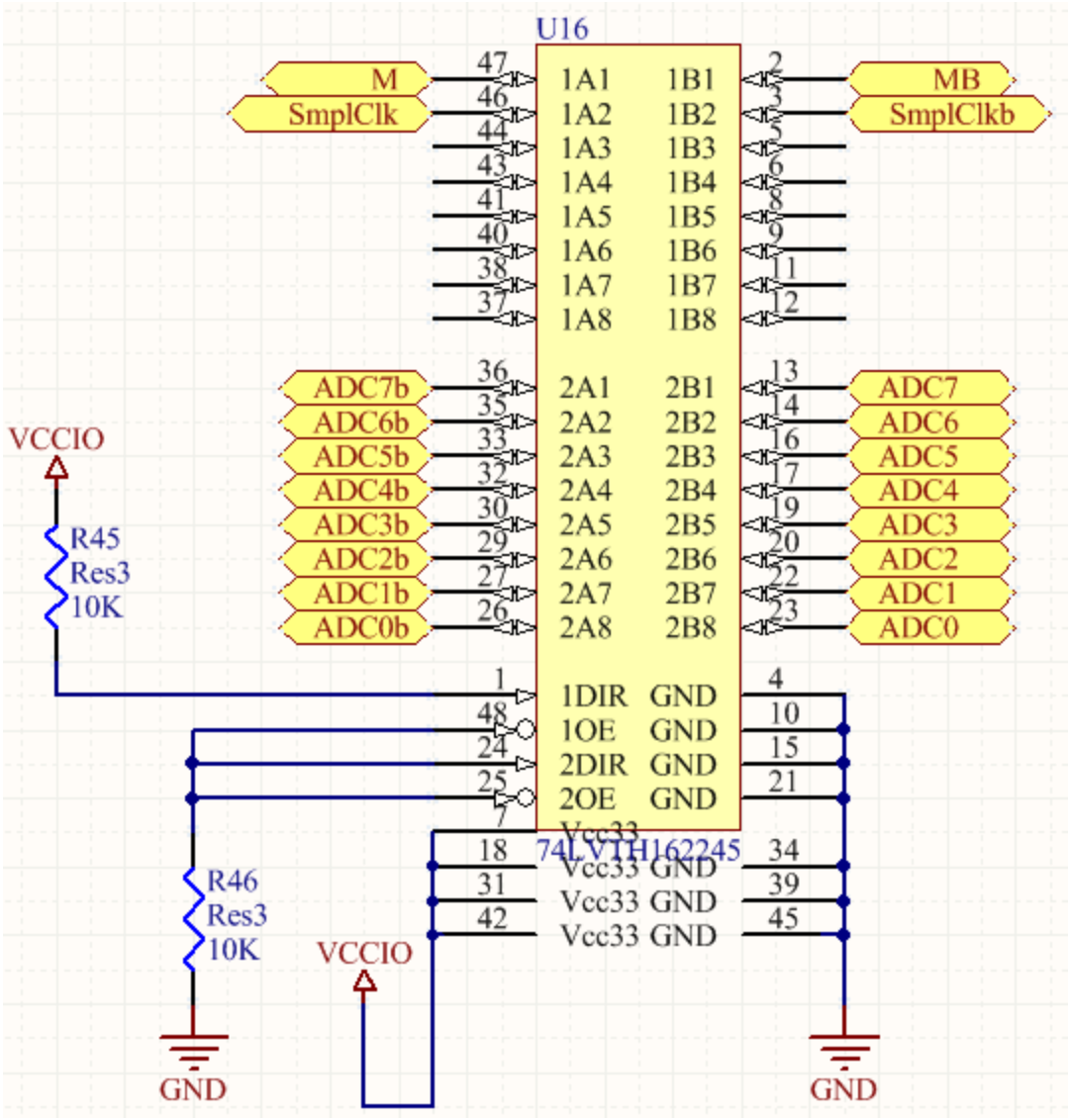


Figure 18 - Schematic for the ADC Signal Buffer

*Data and Input Buffer [**U17**]:*

The data and input buffer buffers the bidirectional data bus and the key and encoder signals. The direction pin for the data bus is connected to the inverse of the read/write signal (an IO pin set up in the FPGA) and the direction pin for the input signal is set to always input. The schematic for this buffer is shown below.

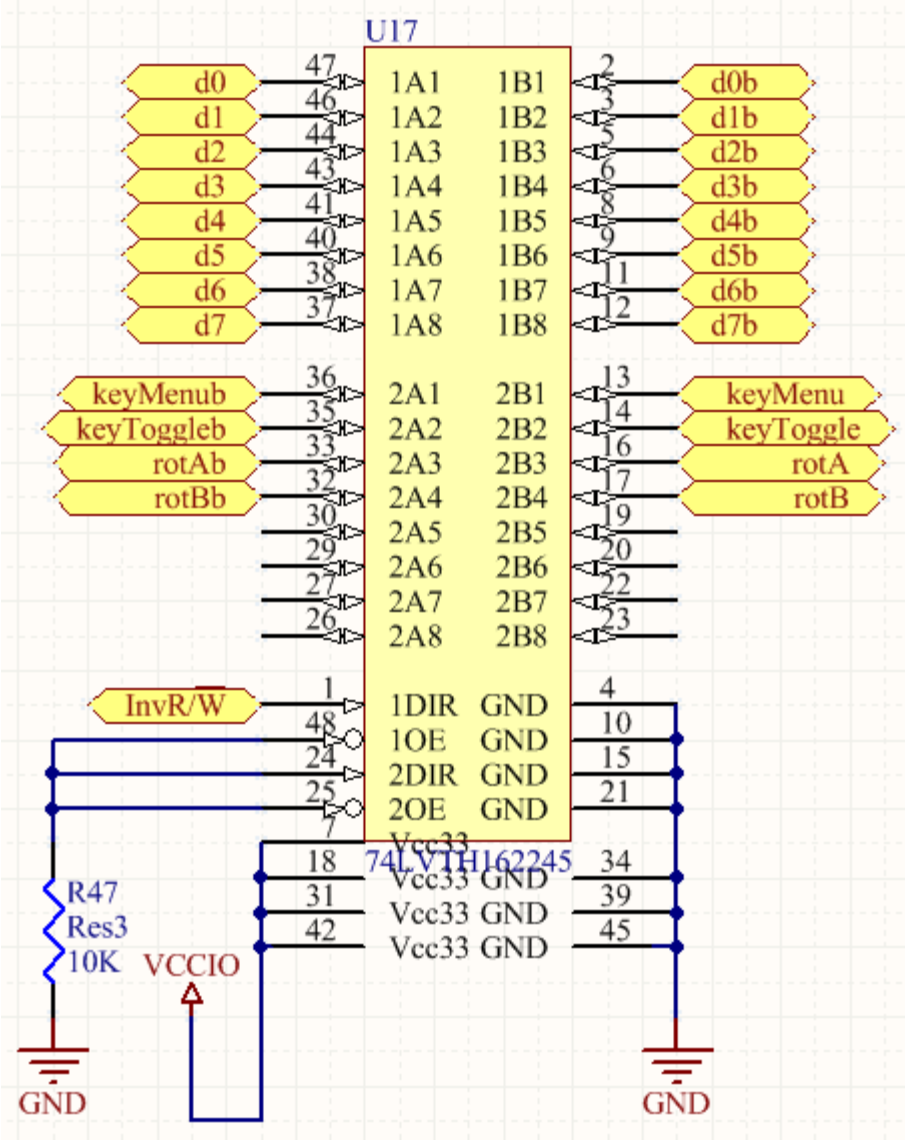


Figure 19 - Schematic for Data and Input Buffer

*General IO Buffer Two [**U18**]:*

This buffer buffers general IO signals from the FPGA. The direction of the buffer is determined by two IO signals from the FPGA. The schematic for this buffer is shown below.

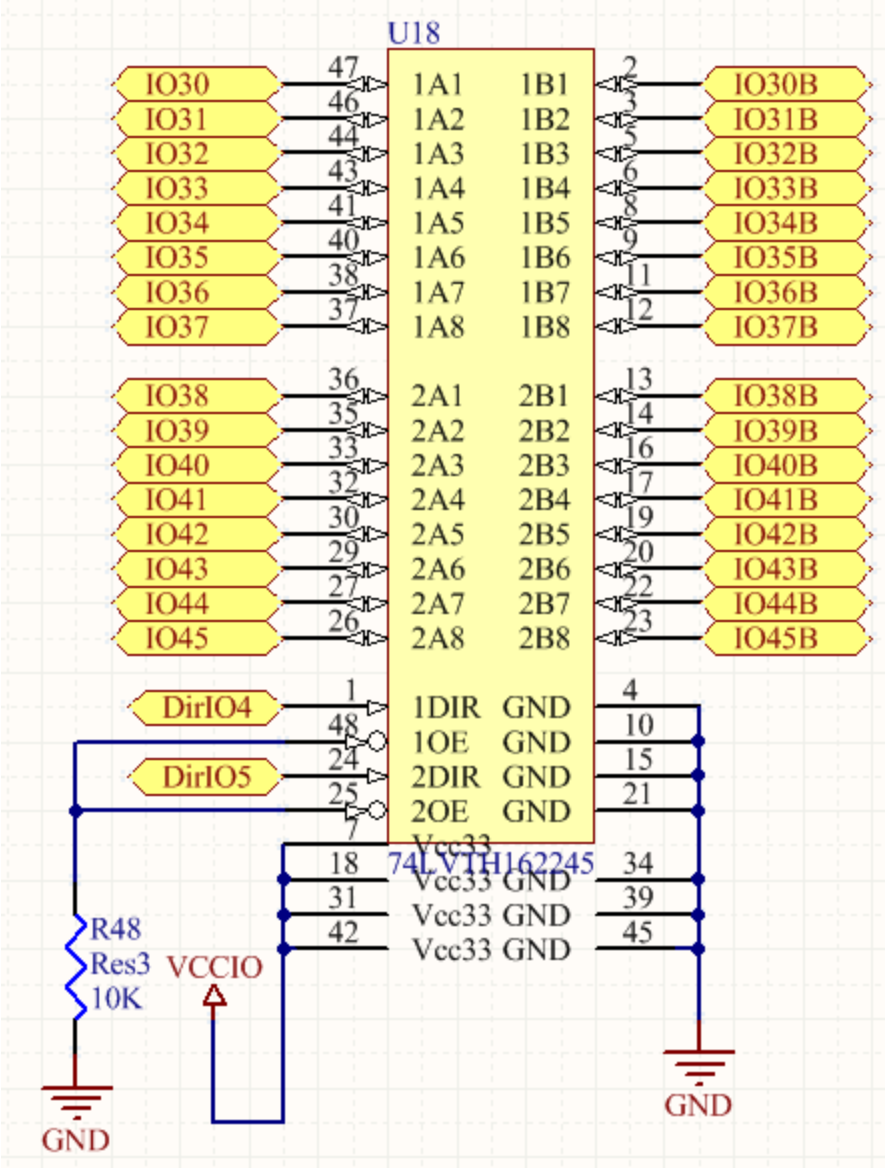


Figure 20 - Schematic for General IO Buffer Two

IO Pins & Headers

There are a total of 38 general IO pins in the system. All of these pins are buffered and sent to 8x2 headers located on the edges of the board as shown in the Board Layout section. The headers are shown in the schematic below.

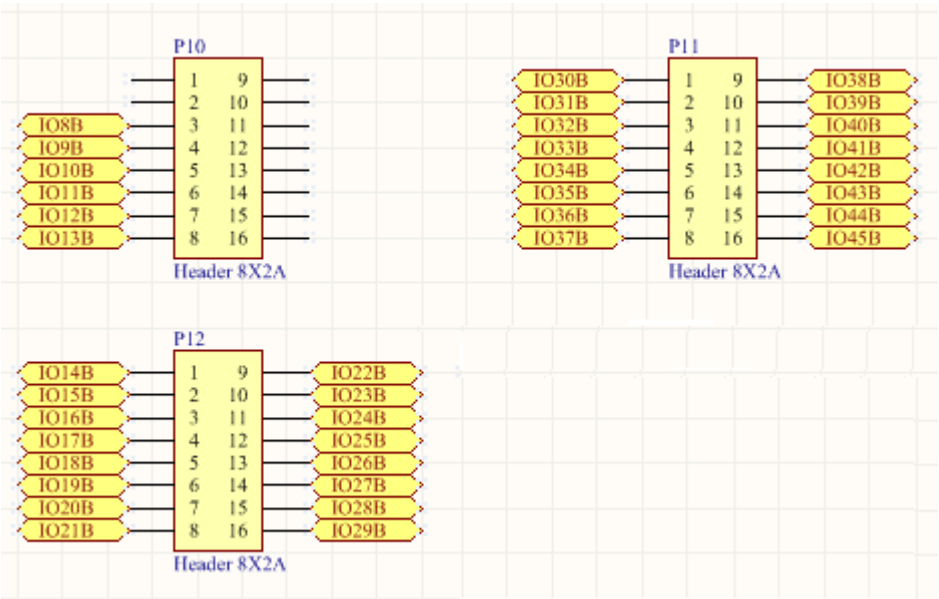


Figure 21 - Schematic for IO Pins and Headers

Power Supply & Regulators

The system uses a standard 5x power supply connected through a DIN5 connector, shown below.

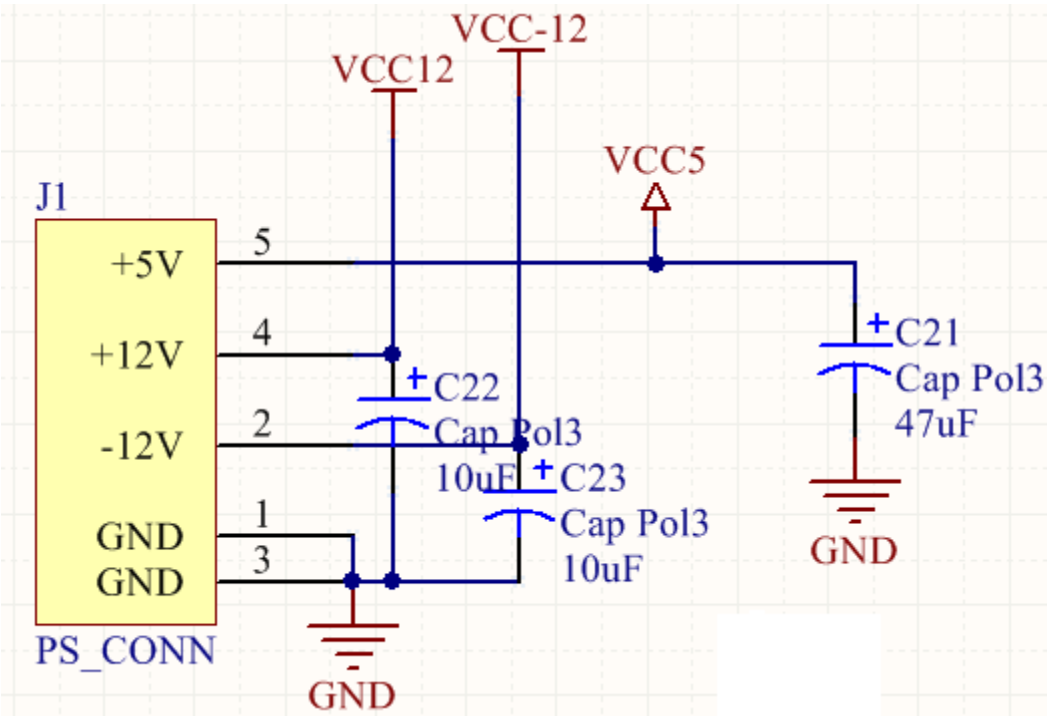


Figure 22 - Schematic for Power Supply Connections

The system uses adjustable LDO LM1086 regulators to generate the other voltages needed in the system. If R1 is the resistor from pin 1 to ground and R2 is the resistor from pin 2 to ground, the output voltage is calculated as follows, provided that R2 is kept as close as possible to 100Ω:

$$V_{out} = 1.25\,V\,(1 + \frac{R1}{R2})$$

Equation 1 - Output Voltage of Adjustable LM1086 Voltage Regulators

Five regulators were used, three for FPGA voltages (1.25 Volts, 2.5 Volts, and 3.3 Volts), and two for front-end and ADC purposes (2 Volts and 4 Volts). The schematic for these is shown below. The bypass capacitors for each power pin were drawn next to the regulators, but these are not always shown below for simplicity.

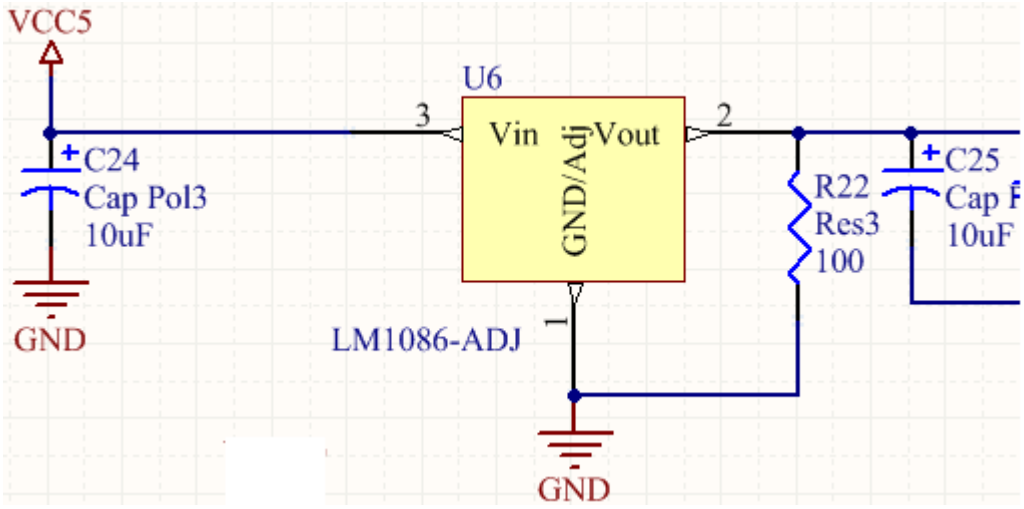


Figure 23 - Schematic for 1.25 V Voltage Regulator

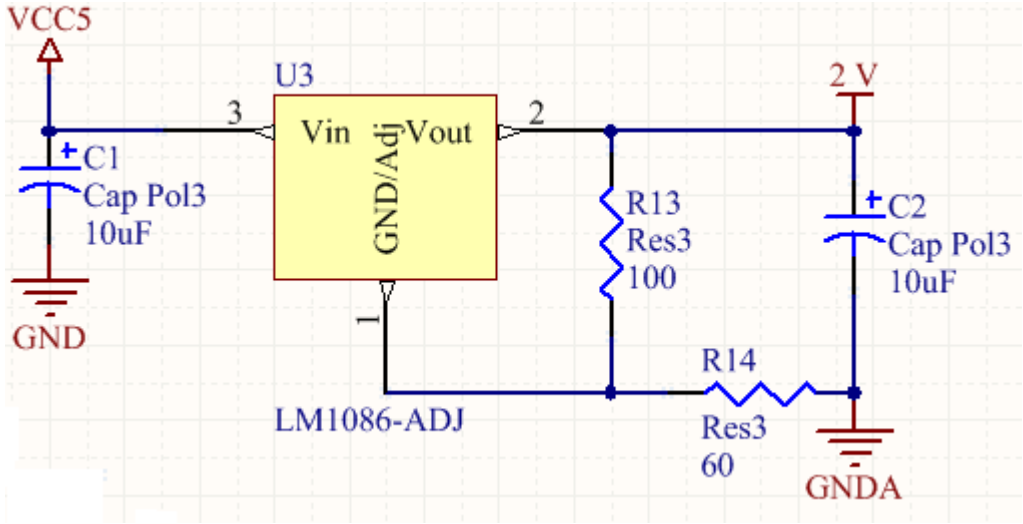


Figure 24 - Schematic for 2 V Voltage Regulator (connected to analog ground because this voltage is only used in the analog front-end design)

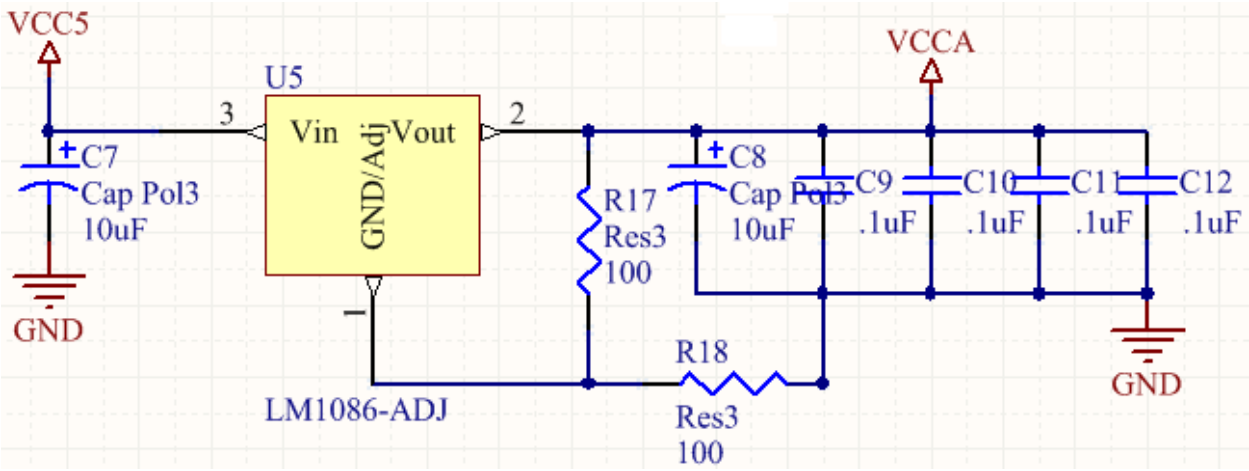


Figure 25 - Schematic for 2.5 V Voltage Regulator

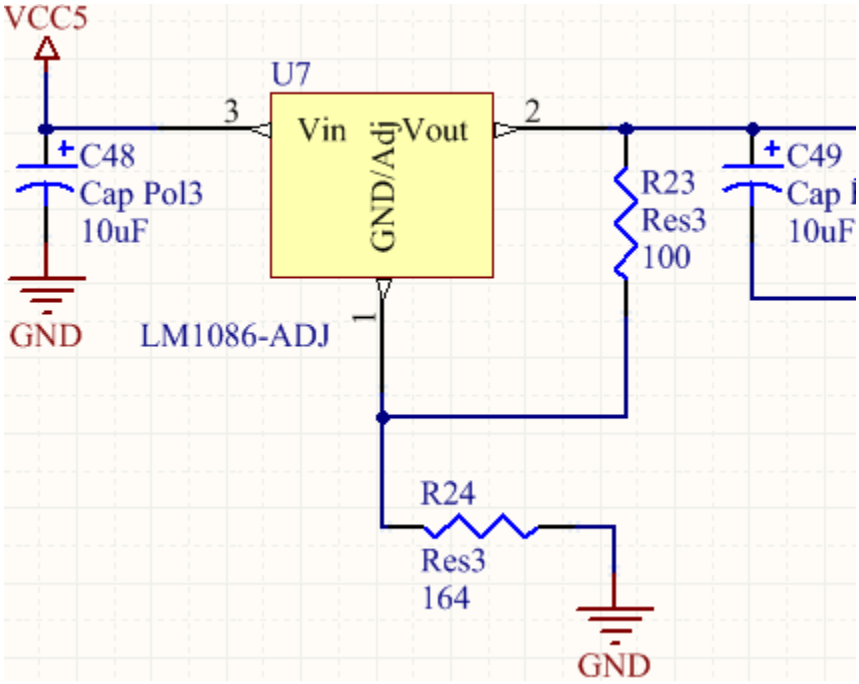


Figure 26 - Schematic for 3.3 V Voltage Regulator

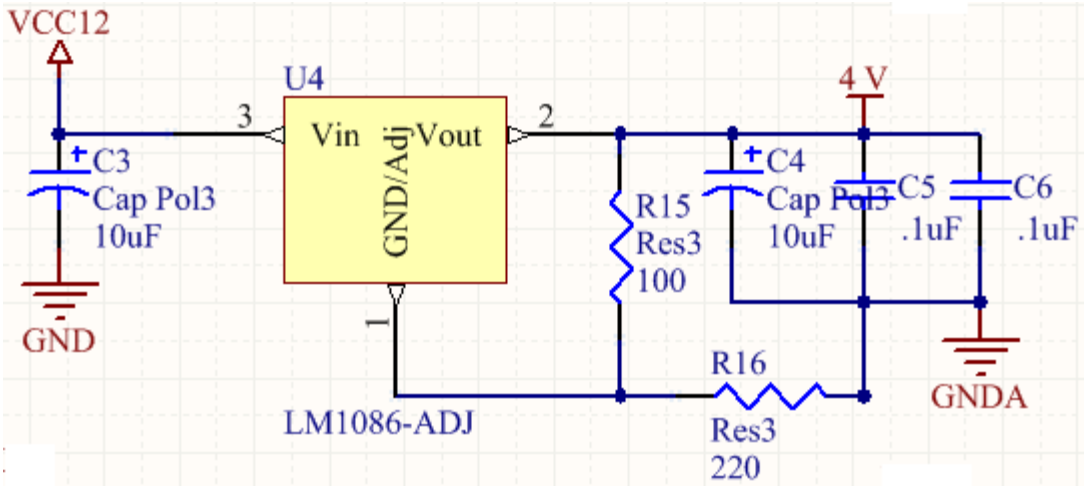


Figure 27 - Schematic for 4 V Voltage Regulator

A table of voltage levels, their schematic names, and their purpose is given below.

Voltage	Name	Purpose
1.25 V	VCCInt	FPGA Power
2 V	2 V	Front End Scaling (not implemented, but in design)
2.5 V	VCCA	FPGA Power
3.3 V	VCCIO	FPGA, Buffer, and User Input Device Power
4 V	4 V	ADC Reference Voltage
5 V	VCC5, VCCAnalog	Voltage Regulators, Memory Chip and ADC Power
12 V	VCC12	Front End Scaling (not implemented) Power & Voltage Regulators
-12 V	VCC-12	Front End Scaling (not implemented) & Display Power

Table 3- Voltage Levels and Uses in System

Serial Configuration Device

The serial configuration device [**U2**] is an Altera EPCS16 device used to program the FPGA on system startup. The schematic and Quartus block necessary to configure this chip are shown below.

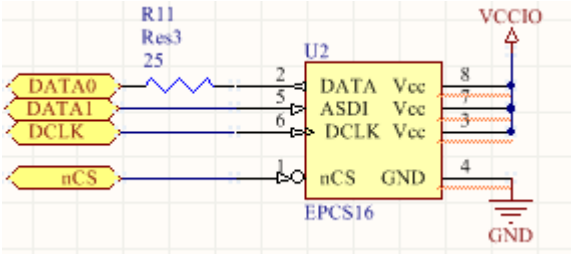


Figure 28 - Schematic for Serial Configuration Device

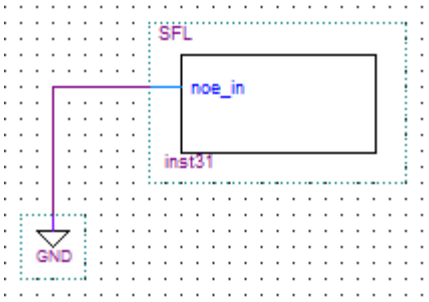


Figure 29 - Quartus-Designed block for Serial Configuration Device

Static RAM (SRAM)

The SRAM chip is a Hitachi 128K x 8 bit (1 M) HM628128B CMOS DIP chip. This chip is used to store software data and configuration settings.

The read/write signal is active high on read, so the signal should be high when reading from the chip and low when writing to it. The output enable signal is tied to the inverse of the read/write signal so that output is enabled during a write. The first chip select is the CPU control signal that determines when the chip is active. The second chip select was tied high (through a pull-up resistor) for simplicity. The timing diagram for these signals and the schematic for the SRAM chip [**U9**] is shown below.

SRAM Read Timing

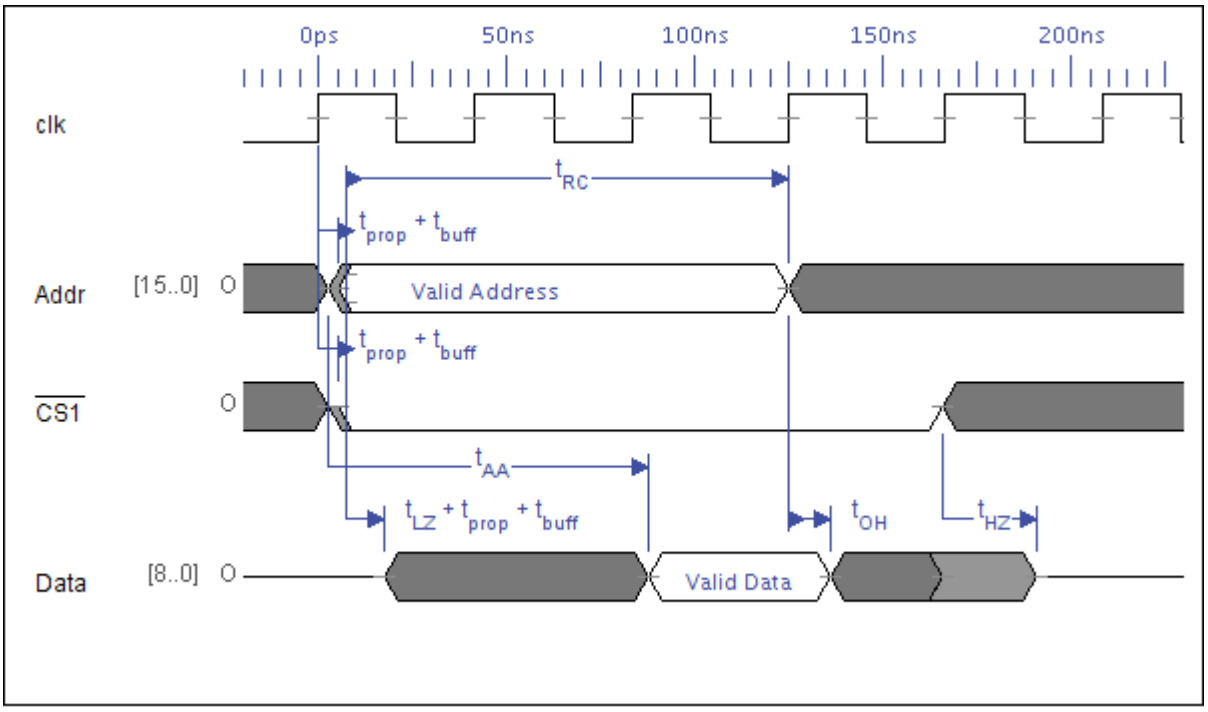


Figure 30 - Read Timing Diagram for SRAM Chip

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX
tprop + tbuff	Propagation and Buffer Delays	Delays from CPU to chp	2.5ns	5ns
tAA	Address delay	Delay between chip is addressed and data is valid	85ns	
tLZ + tprop + tbuff	Low-Z delay	High-impedance to unknown value delay	15ns	
tOH	Output hold from	Data should be valid this long after	10ns	

	address change	address becomes invalid.		
tHZ	High-Z delay	Delay in getting to high-impedance	0ns	25ns
tRC	Read cycle time	How long address must be valid for a whole read cycle	85ns	

Table 4 - Timing Parameters for Read Operation of SRAM

SRAM Write Timing

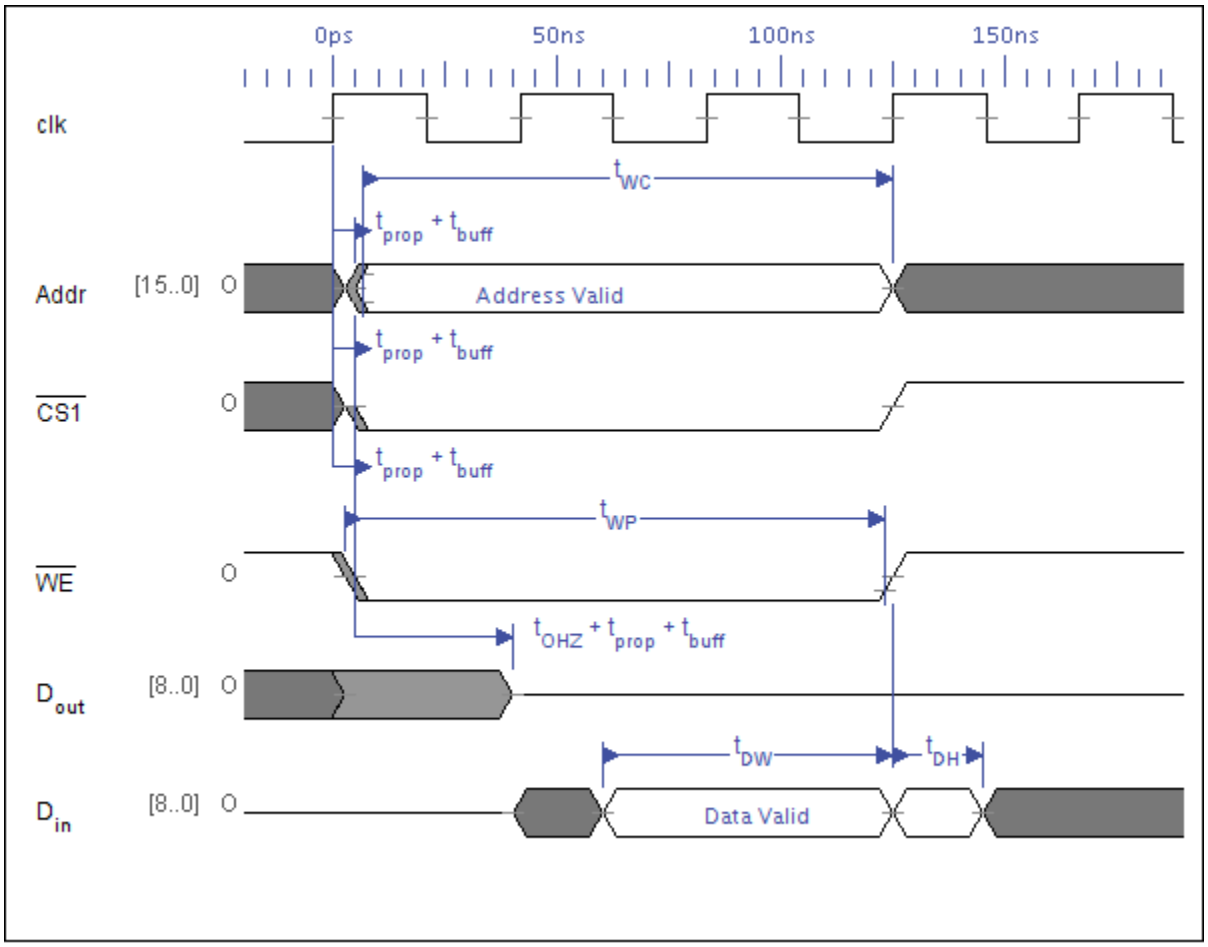


Figure 31 - Write Timing Diagram for SRAM Chip

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX
tWC	Write cycle time	Address must be valid this amount of time	85ns	
tprop + tbuff	Propagation and buffer times	Time to get from CPU to chip	2.5ns	5ns
tOHZ + tprop + tbuff	Output to High Impedance	Delay between OE going high and Dout in high impedance		35ns

tWP	Write pulse width	Minimum time between WE and valid data	55ns
tDW	Data to write time overlap	Overlap between when data validates and it is sampled	35ns
tDH	Data hold to write time	How much data should be held after sample	0ns

Table 5 – Timing Parameters for Write Operation of SRAM

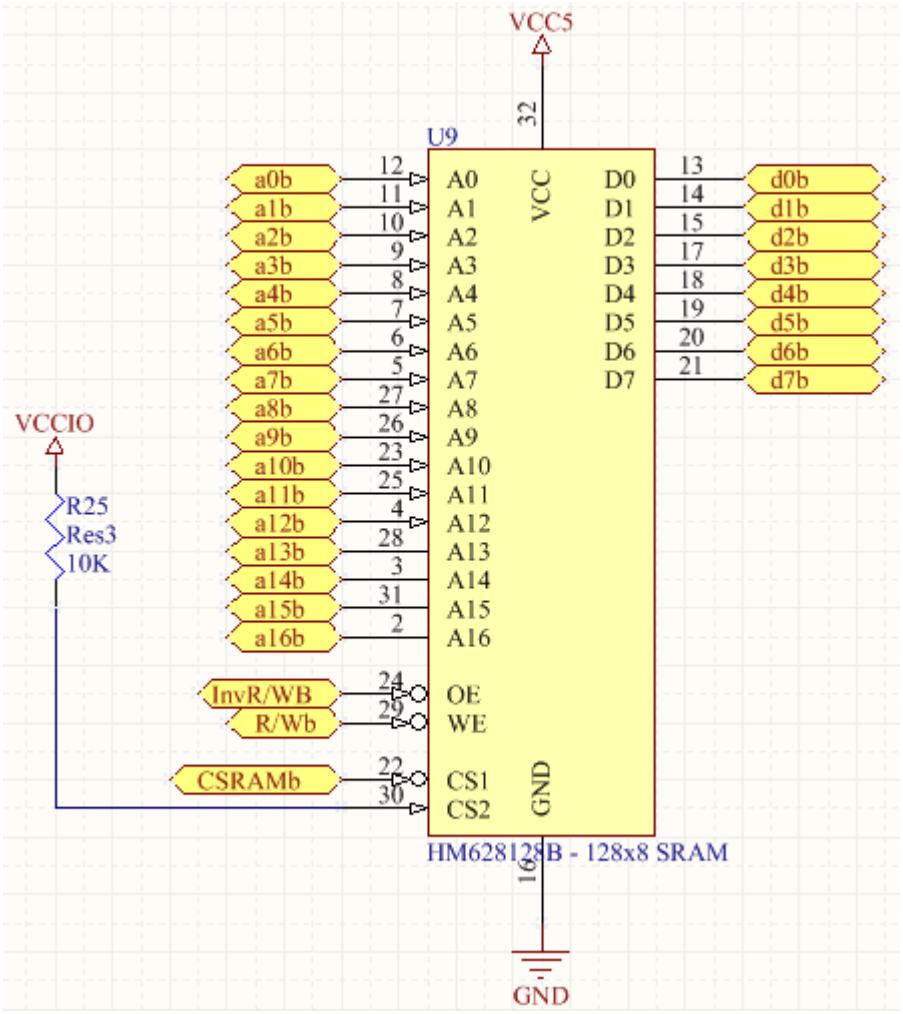


Figure 32 - Schematic for SRAM Chip

EEROM

The EEROM used is an AMD AM29F010A 128K x 8 bit chip, although the system was designed to use an AMD AM29F040 512kword x 8 bit chip, which has the same interface but with more memory and more address lines. Thus, although the system was designed to use nineteen address lines (a_0-a_{18}), only seventeen are actually used (a_0-a_{16}).

The output enable of the chip is tied low, so the chip is always enabled. The read/write signal is shared between the ROM and the RAM, so this signal should be high when reading from the chip and low when writing , although ROM writes should not happen. The chip select signal is the chip control signal coming from the CPU. The timing for these signals is shown below.

EEROM Read Timing

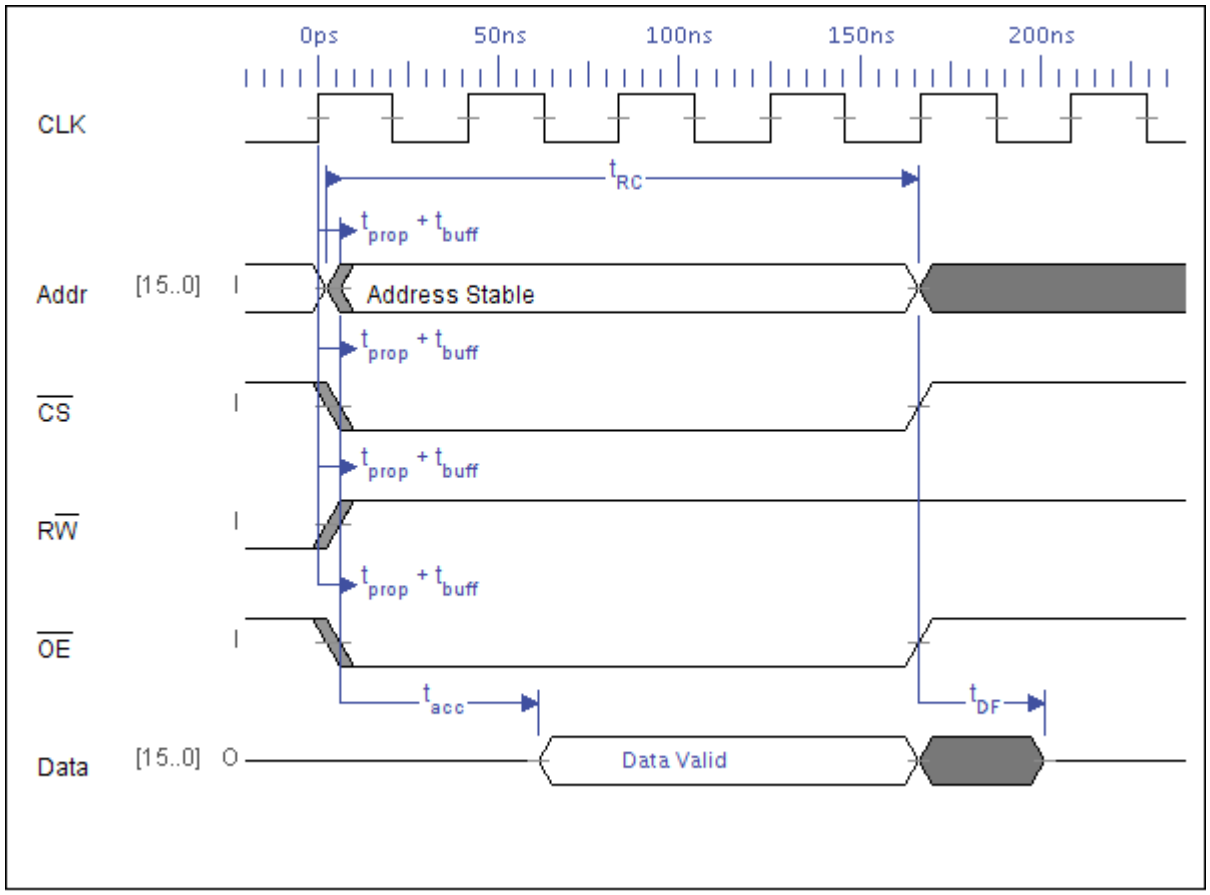


Figure 33 - EEROM Read Timing

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX
tprop + tbuff	Propagation Delay	Propagation time from CPU to FPGA Pin	2.5ns	6ns
tOEH	Output Enable Hold Time	Hold time for output enable signal	0ns	
tRC	Read Time Cycle	Minimum address valid time	150ns	
tacc	Address to output delay	Delay between address stable and data output		55ns
tDF	Chip Enable/Address Change to High Z	Chip Enable/Address Change to High Z		35ns

Table 6 - EEROM Read Timing Parameters

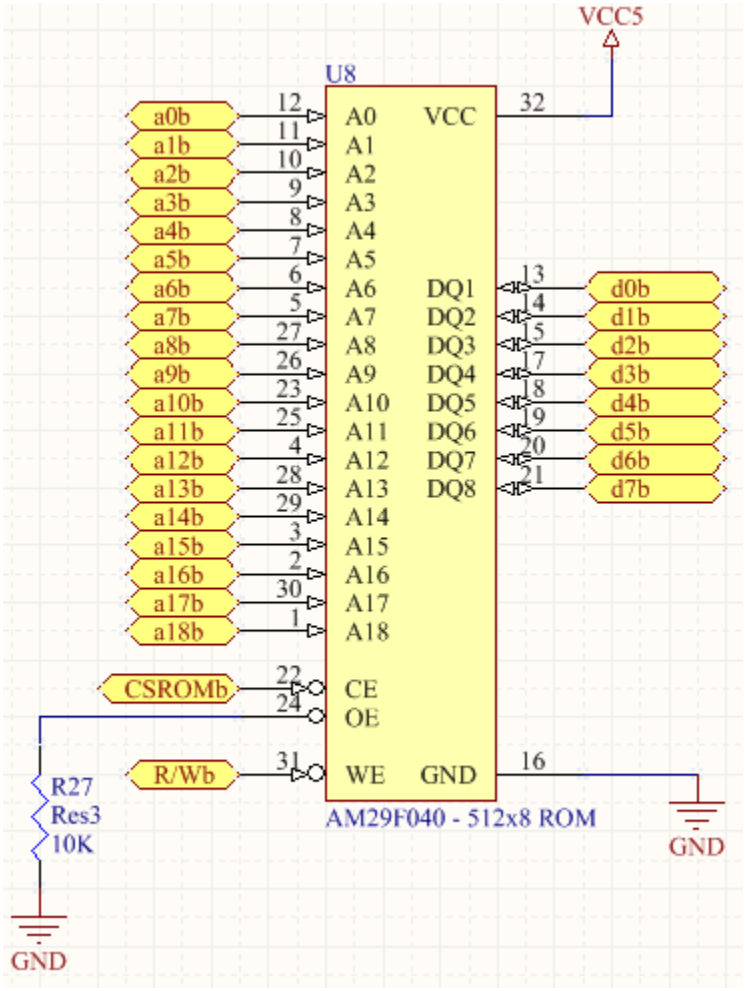


Figure 34 - Schematic for EEROM chip

VRAM & Controller

The system was designed to use a NEC uPD41264 64K x 4 bit VRAM chip. The VRAM chip uses a multiplexed address bus, which takes an 8-bit row address followed by an 8-bit column address to determine the full 16-bit address. It has two modes of operation, one in which it behaves as a normal memory chip with read, write, and refresh operations, and one ("the serial transfer cycle") in which it transfers a row of data serially to the display. The timing for these cycles is shown below.

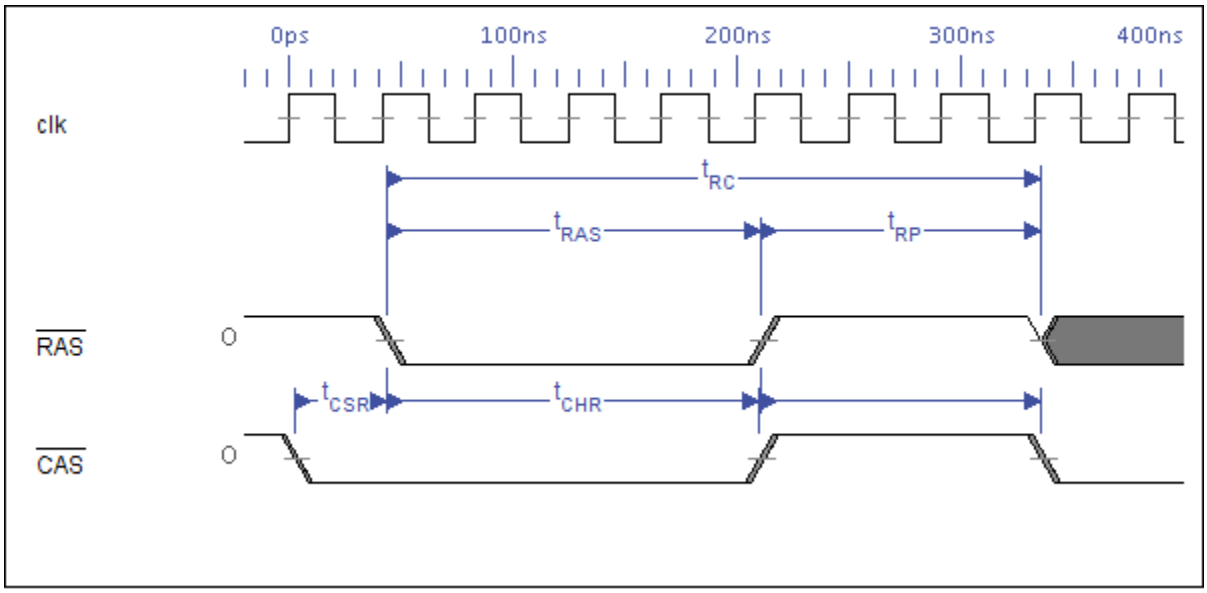


Figure 35 - VRAM Refresh Timing

SYMBOL	DEFINITION	MIN	MAX
tRC	Read cycle time	270ns	
tRAS	RAS pulse width	150ns	10us
tRP	RAS high time	100ns	
tCSR	CAS setup time	10ns	
tCHR	CAS hold time	30ns	
tprop + tbuff	Propagation delay	2.5ns	5ns

Table 7 - VRAM Refresh Parameters

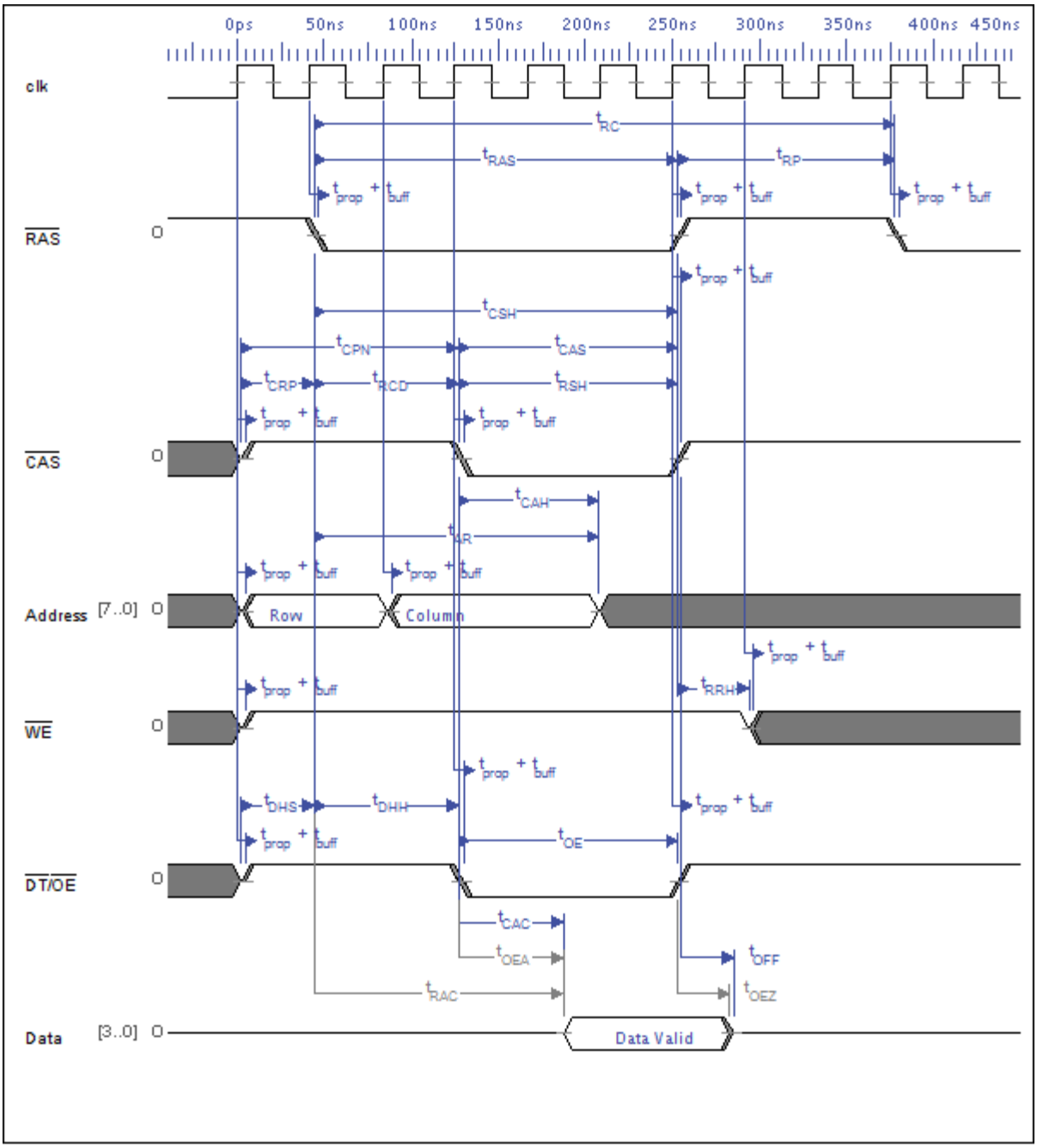


Figure 36 - VRAM Read Timing

SYMBOL	DEFINITION	MIN	MAX
tRAS	RAS pulse width	150ns	
tRC	Read cycle time	270ns	
tRP	Refresh interval	100ns	
tCRP	CAS high to to RAS low precharge time	10ns	
tRCD	RAS to CAS delay time	30ns	

tCPN	CAS precharge time	30ns	
tCAS	CAS pulse width	75ns	
tRSH	RAS hold time	75ns	
tCSH	CAS hold time	150ns	
tRAH	Row address hold time	20ns	
tCAH	Column address hold time	25ns	
tAR	Column address time after RAS low	100ns	
tRRH	Read command hold after RAS high	20ns	
tRAC	Access time from RAS	120ns	
tCAC	Access time from CAS	60ns	
tOFF	Output disable time from CAS high	30ns	30ns
tprop + tbuff	Propagation delay	2.5ns	5ns
tDHS	DT high setup time	0ns	
tDHH	DT high hold time	25ns	
tOEA	Access time from OE	30ns	
tOE	OE pulse width	40ns	
tOEZ	Output disable time from OE high	0ns	30ns

Table 8 - VRAM Read Timing Parameters

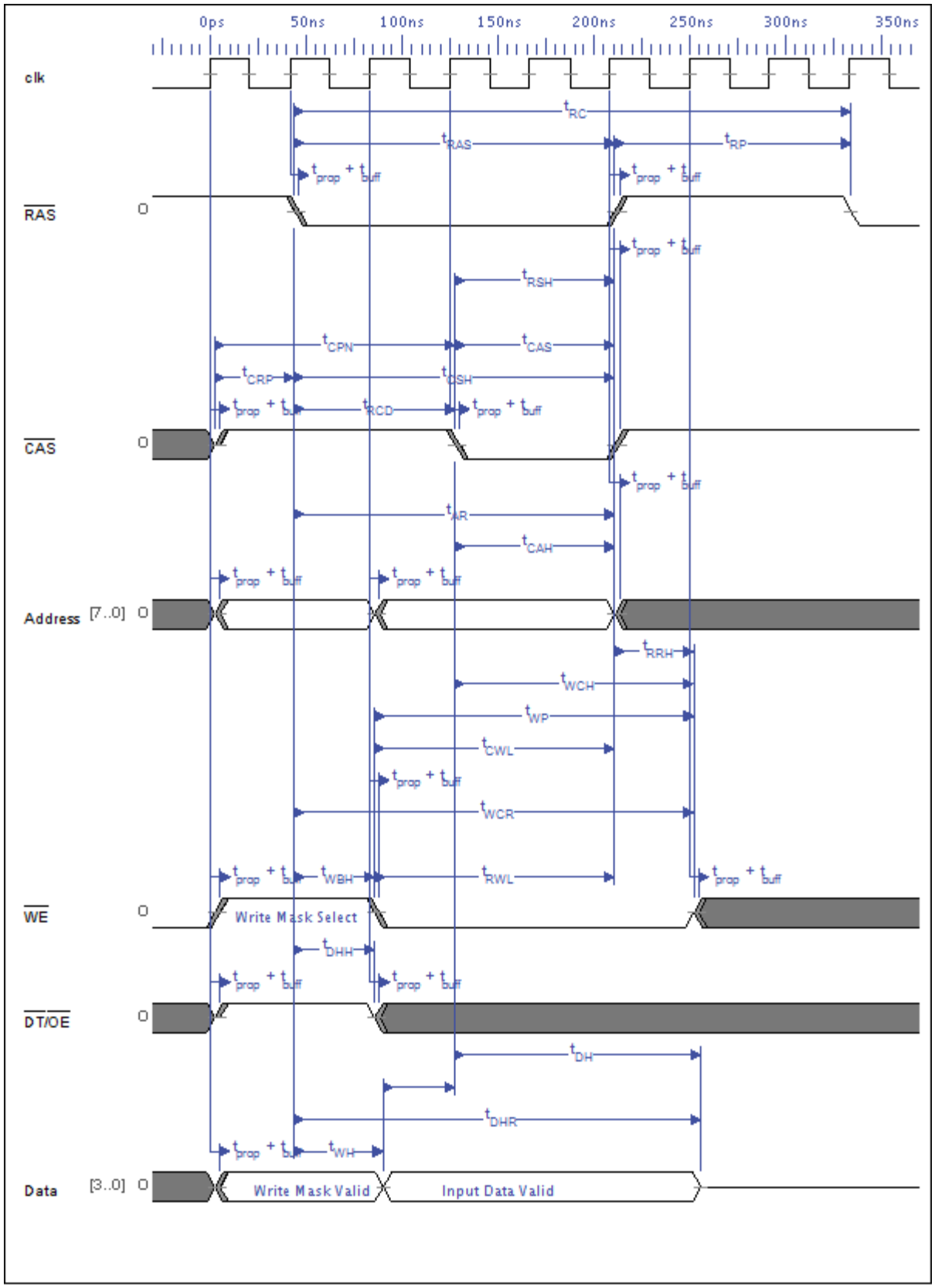


Figure 37 - VRAM Write Timing

SYMBOL	DEFINITION	MIN	MAX
tRAS	RAS pulse width	150ns	
tRC	Read cycle time	270ns	
tRP	Refresh interval	100ns	
tCRP	CAS high to to RAS low precharge time	10ns	
tRCD	RAS to CAS delay time	30ns	
tCPN	CAS precharge time	30ns	
tCAS	CAS pulse width	75ns	
tRSH	RAS hold time	75ns	
tCSH	CAS hold time	150ns	
tRAH	Row address hold time	20ns	
tCAH	Column address hold time	25ns	
tAR	Column address time after RAS low	100ns	
tRRH	Read command hold after RAS high	20ns	
tRAC	Access time from RAS		120ns
tCAC	Access time from CAS		60ns
tOFF	Output disable time from CAS high	0ns	30ns
tWBH	Write-per-bit hold time	25ns	
tCWL	Write command to CAS lead time	45ns	
tWP	WE pulse width	45ns	
tRWL	Write command to RAS lead time	45ns	
tWCR	Writ ecommand hold time after RAS low	120ns	
tWCH	Write command hold time	45ns	
tDHR	Data-in hold time after RAS low	120ns	
tDH	Data-in hold time	45ns	
tWH	Write bit selection hold time	25ns	
tprop + tbuff	Propagation delay	2.5ns	5ns
tDHH	DT hold time	25ns	

Table 9 - VRAM Write Timing Parameters

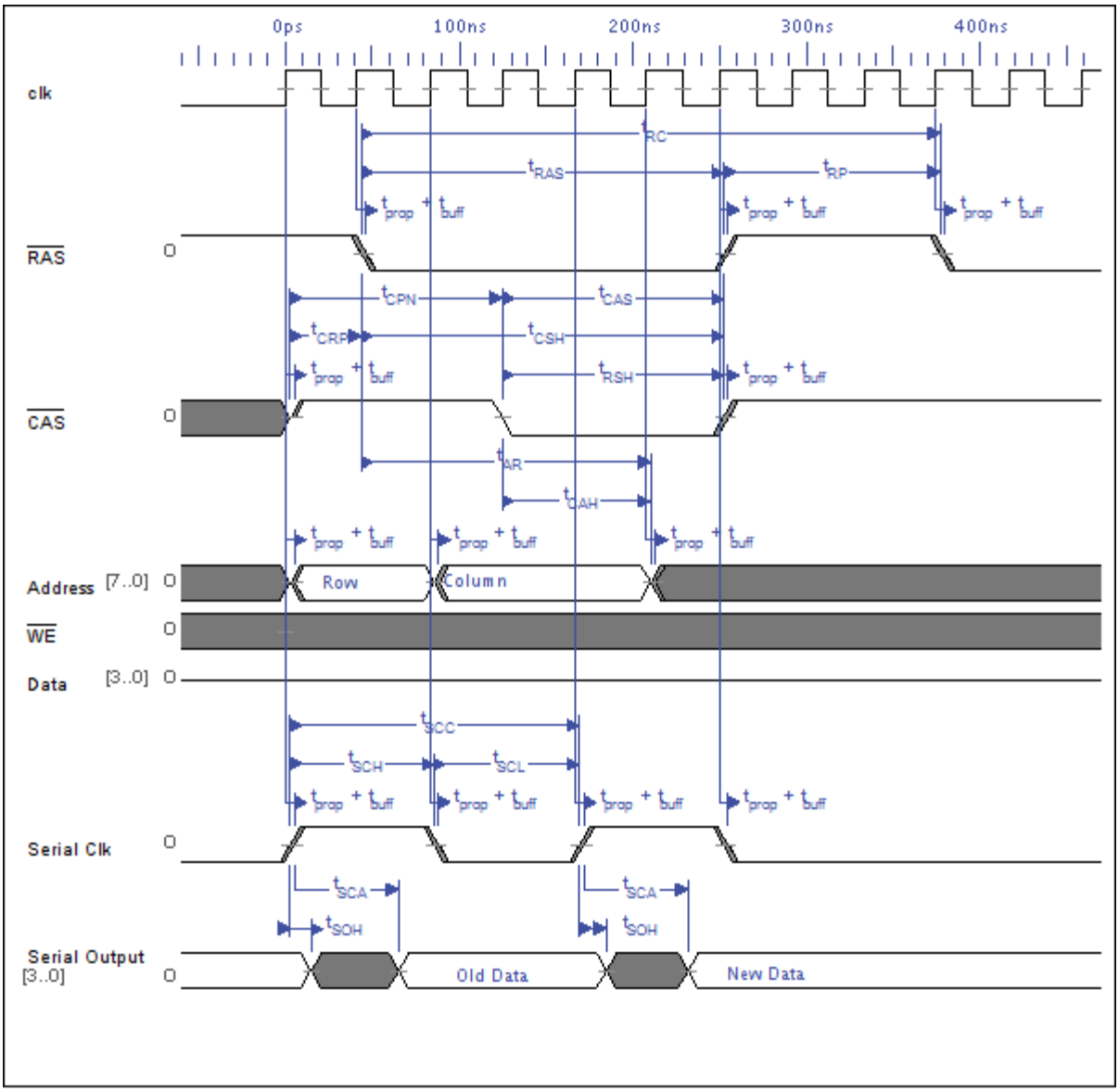


Figure 38 - VRAM Serial Row Transfer Cycle Timing

SYMBOL	DEFINITION	MIN	MAX
tRAS	RAS pulse width	150ns	
tRC	Read cycle time	270ns	
tRP	Refresh interval	100ns	
tCRP	CAS high to to RAS low precharge time	10ns	
tRCD	RAS to CAS delay time	30ns	75ns
tCPN	CAS precharge time	30ns	
tCAS	CAS pulse width	75ns	
tRSH	RAS hold time	75ns	
tCSH	CAS hold time	150ns	

tRAH	Row address hold time	20ns	
tCAH	Column address hold time	25ns	
tAR	Column address time after RAS low	100ns	
tRRH	Read command hold after RAS high	20ns	
tRAC	Access time from RAS		120ns
tCAC	Access time from CAS		60ns
tOFF	Output disable time from CAS high	0ns	30ns
tSCC	Serial clock cycle	60ns	50us
tSCH	SC pulse width (high)	20ns	
tSCL	SC precharge time	20ns	
tSCA	Serial output access time from SC		60ns
tSOH	Serial output hold time after SC high	10ns	
tprop + tbuff	Propagation delay	2.5ns	5ns

Table 10 - VRAM Serial Row Transfer Cycle Parameters

The schematic for the VRAM chip is shown below.

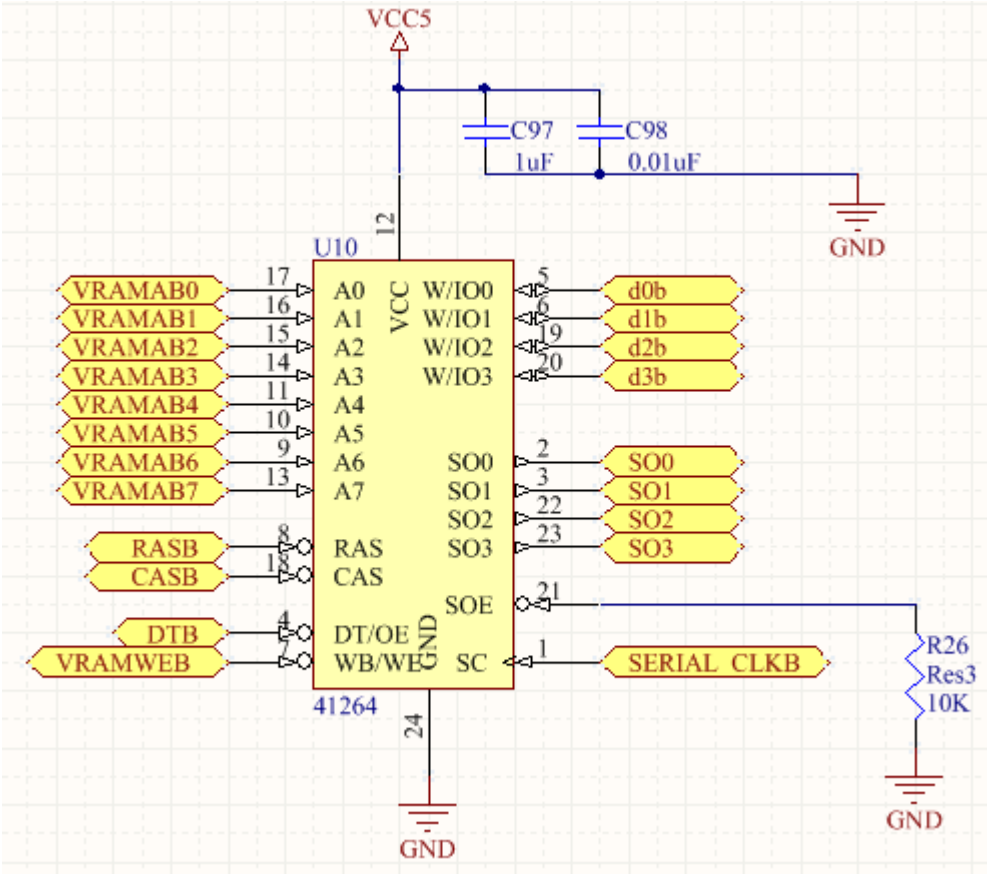


Figure 39 - Schematic for VRAM Chip

The VRAM is controlled through a VRAM controller programmed in VHDL in the FPGA. This controller takes the control signals from the CPU (read/write and chip select) and converts them into the signals VRAM uses (RAS, CAS, read/write, output enable, data transfer, and address signals). The row done signal from the Display Controller informs the VRAM Controller that the Display needs another row and that it should start a new serial row transfer cycle. The VRAM controller sends an acknowledge signal back to the Display Controller after the serial row transfer cycle is completed. This signal flow is shown in the diagram below.

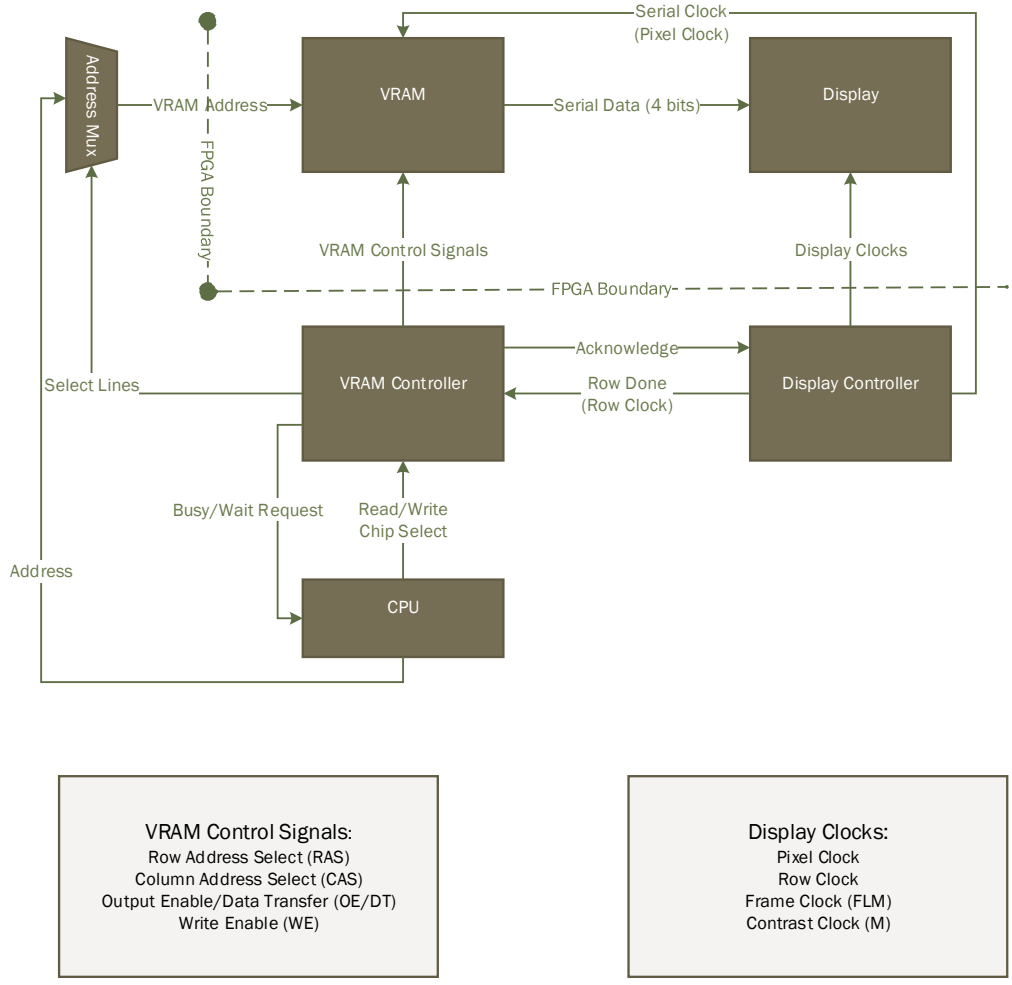


Figure 40 - VRAM and Display Flow Diagram

The VRAM refreshes whenever it is not doing a read, write, or serial row transfer operation as specified in the VRAM Controller. The code for the VRAM controller is shown below.

```
--------------------------------------------------------------------
--
--  VRAM Controller
--
--  A VRAM controller for the VRAM of the scope project. It's a simple
--  state machine that controls the inputs into the VRAM so that
--  it follows timing requirements for a 24MHz oscillator.
--
--
--
--  Revision History:
--     21 Feb 14   Daniel Seabra       Initial revision.
--     03 March 14 Daniel Seabra       Debugging and fixing with simulation
--
----------------------------------------------------------------------------


-- bring in the necessary packages
library  ieee;
use  ieee.std_logic_1164.all;


--
--  Oscilloscope Digital Trigger entity declaration
--

entity  VRAM_Controller  is
    port (
        RW         :  in  std_logic;      -- R/W from CPU
        CS         :  in  std_logic;      -- Chip select from CPU
        sys_clk    :  in  std_logic;      -- System clock from CPU
        row_done   :  in  std_logic;      -- Get new row from display controller
        reset      :  in  std_logic;      -- Reset Logic to go into Idle
        ack        :  out std_logic;      -- R.T acknowledge to LCD controller
        busy       :  out std_logic;      -- Busy signal to CPU
        RWO        :  out std_logic;      -- R/W to VRAM
        RAS        :  out std_logic;      -- RAS signal to VRAM (row)
        CAS        :  out std_logic;      -- CAS signal to VRAM (column)
        DT         :  out std_logic;      -- Data transfer to VRAM
        row_slct   :  out std_logic;      -- Row select (0 if row, 1 if column)
        addr_slct  :  out std_logic       -- Address select (0 if CPU addr, 1
otherwise)
    );
end  VRAM_Controller;

--
--  Oscilloscope Digital Trigger Moore State Machine
--     State Assignment Architecture
--
--  This architecture just shows the basic state machine syntax when the state
--  assignments are made manually.  This is useful for minimizing output
--  decoding logic and avoiding glitches in the output (due to the decoding
--  logic).
--
```

```
architecture  assign_statebits  of  VRAM_Controller  is

    subtype  states  is  std_logic_vector(14 downto 0);     -- state type

    -- define the actual states as constants
    -- first eight bits are outputs
    -- next three bits are type: idle, read, write, refresh, serial
    -- final four bits are states
    constant  IDLE   : states := "011111000000000";

    constant  READ0  : states := "011111000010000";
    constant  READ1  : states := "011011000010001";
    constant  READ2  : states := "011011100010010";
    constant  READ3  : states := "011000100010011";
    constant  READ4  : states := "001000100010100";
    constant  READ5  : states := "011000000010101";
    constant  READ6  : states := "011111000010110";
    constant  READ7  : states := "011111000010111";
    constant  READ8  : states := "011111000011000";
    constant  READ9  : states := "011111000011001";

    constant  WRITE0 : states := "011111000100000";
    constant  WRITE1 : states := "011011000100001";
    constant  WRITE2 : states := "010011100100010";
    constant  WRITE3 : states := "000001100100011";
    constant  WRITE4 : states := "010001100100100";
    constant  WRITE5 : states := "011111000100101";
    constant  WRITE6 : states := "011111000100110";
    constant  WRITE7 : states := "011111000100111";
    constant  WRITE8 : states := "011111000101000";

    constant  RFRSH0 : states := "011101000110000";
    constant  RFRSH1 : states := "011001000110001";
    constant  RFRSH2 : states := "011001000110010";
    constant  RFRSH3 : states := "011001000110011";
    constant  RFRSH4 : states := "011001000110100";
    constant  RFRSH5 : states := "011111000110101";
    constant  RFRSH6 : states := "011111000110110";
    constant  RFRSH7 : states := "011111000110111";
    constant  RFRSH8 : states := "011111000111000";

    constant  SRIAL0 : states := "011110011000000";
    constant  SRIAL1 : states := "011010011000001";
    constant  SRIAL2 : states := "011010111000010";
    constant  SRIAL3 : states := "011000111000011";
    constant  SRIAL4 : states := "011000111000100";
    constant  SRIAL5 : states := "011001011000101";
    constant  SRIAL6 : states := "011111011000110";
    constant  SRIAL7 : states := "011111011000111";
    constant  SRIAL8 : states := "111111011001000";


    signal  CurrentState  :  states;    -- current state
    signal  NextState     :  states;    -- next state

begin
```

```
-- the outputs are in the same order as the entity declaration
ack <= CurrentState(14);
busy <= CurrentState(13);
RWO <= CurrentState(12);
RAS <= CurrentState(11);
CAS <= CurrentState(10);
DT <= CurrentState(9);
row_slct <= CurrentState(8);
addr_slct <= CurrentState(7);


-- compute the next state (function of current state and inputs)

transition:  process (RW, CS, row_done, reset, CurrentState)
begin

    if (reset = '0') then
        NextState <= IDLE;
    elsif (reset = '1') then

        case  CurrentState  is           -- Do the state transition/output

            when IDLE =>
                if (row_done = '1') then
                    NextState <= SRIAL0;
                elsif (CS = '0' and RW = '1') then
                    NextState <= READ0;
                elsif (CS = '0' and RW = '0') then
                    NextState <= WRITE0;
                else
                    NextState <= RFRSH0;
                end if;

            when  READ0 =>
                NextState <= READ1;

            when  READ1 =>
                NextState <= READ2;

            when  READ2 =>
                NextState <= READ3;

            when  READ3 =>
                NextState <= READ4;

            when  READ4 =>
                NextState <= READ5;

            when  READ5 =>
                NextState <= READ6;

            when  READ6 =>
                NextState <= READ7;
```

```
when  READ7 =>
    NextState <= READ8;

when  READ8 =>
    NextState <= READ9;

when  READ9 =>
    NextState <= IDLE;


when  WRITE0 =>
    NextState <= WRITE1;

when  WRITE1 =>
    NextState <= WRITE2;

when  WRITE2 =>
    NextState <= WRITE3;

when  WRITE3 =>
    NextState <= WRITE4;

when  WRITE4 =>
    NextState <= WRITE5;

when  WRITE5 =>
    NextState <= WRITE6;

when  WRITE6 =>
    NextState <= WRITE7;

when  WRITE7 =>
    NextState <= WRITE8;

when WRITE8 =>
    NextState <= IDLE;



when  RFRSH0 =>
    NextState <= RFRSH1;

when  RFRSH1 =>
    NextState <= RFRSH2;

when  RFRSH2 =>
    NextState <= RFRSH3;

when  RFRSH3 =>
    NextState <= RFRSH4;

when  RFRSH4 =>
    NextState <= RFRSH5;

when  RFRSH5 =>
    NextState <= RFRSH6;
```

```
            when  RFRSH6 =>
                NextState <= RFRSH7;

            when  RFRSH7 =>
                NextState <= RFRSH8;

            when  RFRSH8 =>
                NextState <= IDLE;



            when  SRIAL0 =>
                NextState <= SRIAL1;

            when  SRIAL1 =>
                NextState <= SRIAL2;

            when  SRIAL2 =>
                NextState <= SRIAL3;

            when  SRIAL3 =>
                NextState <= SRIAL4;

            when  SRIAL4 =>
                NextState <= SRIAL5;

            when  SRIAL5 =>
                NextState <= SRIAL6;

            when  SRIAL6 =>
                NextState <= SRIAL7;

            when  SRIAL7 =>
                NextState <= SRIAL8;

            when  SRIAL8 =>
                NextState <= IDLE;

            when others =>
                 NextState <= IDLE;

        end case;

    end if;

end process transition;


-- storage of current state (loads the next state on the clock)

process (sys_clk)
begin

    if  sys_clk = '1'  then         -- only change on rising edge of clock
        CurrentState <= NextState;  -- save the new state information
```

```
        end if;

    end process;


end  assign_statebits;
```

Display & Controller

The system uses a Federal Express LM215XB 480x128 pixel black and white display. This display is controlled by four display clocks: a pixel clock that goes high every pixel of the display (the “pixel clock” or “CL2”), a row clock that goes high every row of the display (the “row clock” or “CL1”), a frame clock that goes high every time the screen is refreshed (“FLM”), and a contrast clock that alternates every FLM (“M”). The timing for these clocks is shown below.

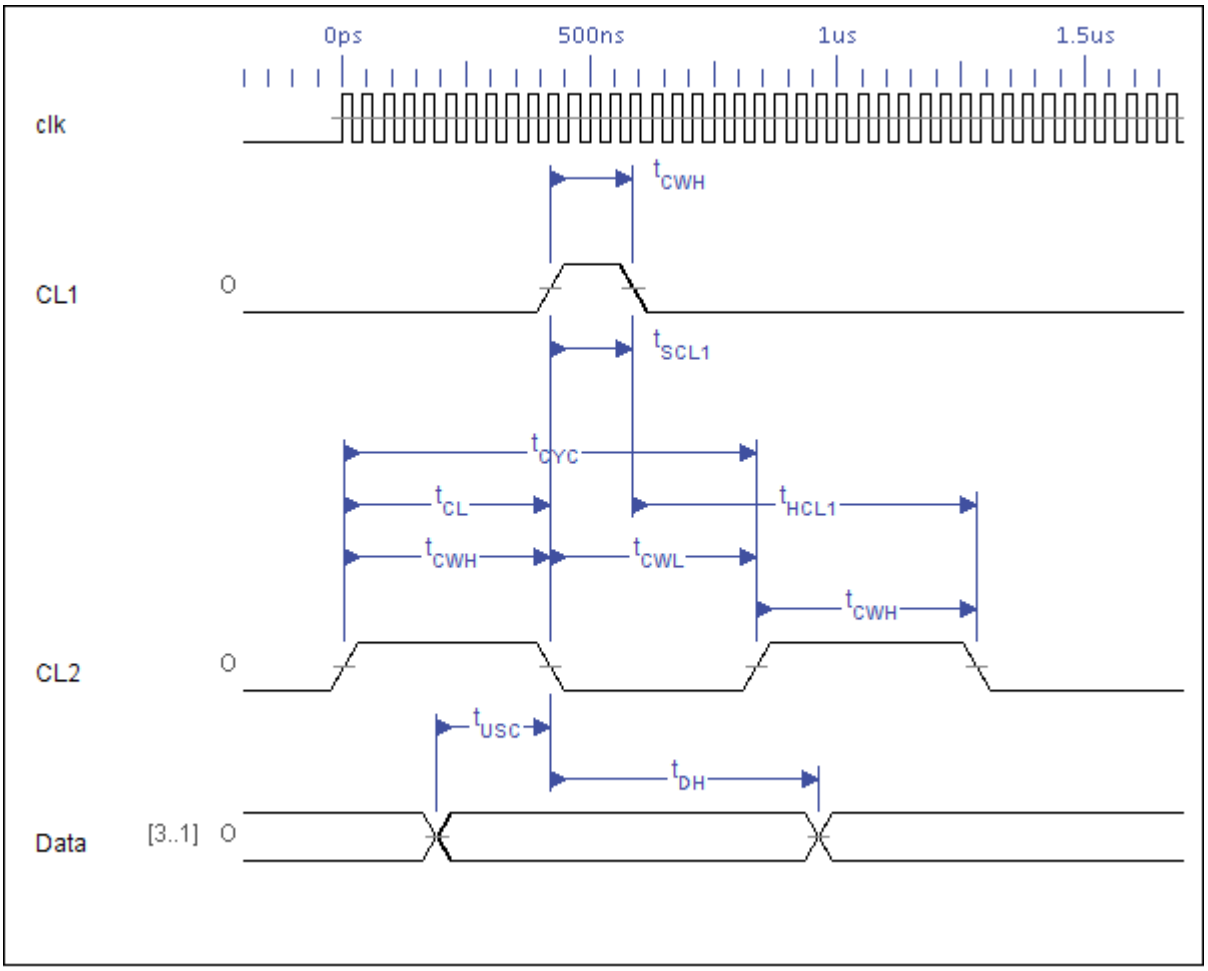


Figure 41 - Clocks for the LCD Display

SYMBOL	DEFINITION	MIN
tCWH	CL2 and CL1 Pulse Widths (H)	150ns
tCWL	CL2 and CL1 Pulse Widths (L)	150ns
tCL	CL1 delay time	150ns
tHCL1	CL1 hold time	150ns
tUSC	Data set up time	100ns

tDH	Data hold time	100ns
tSCL1	CL1 set up time	150ns
tCYC	CL2 cycle time	810ns

Table 11 - LCD Timing Parameters

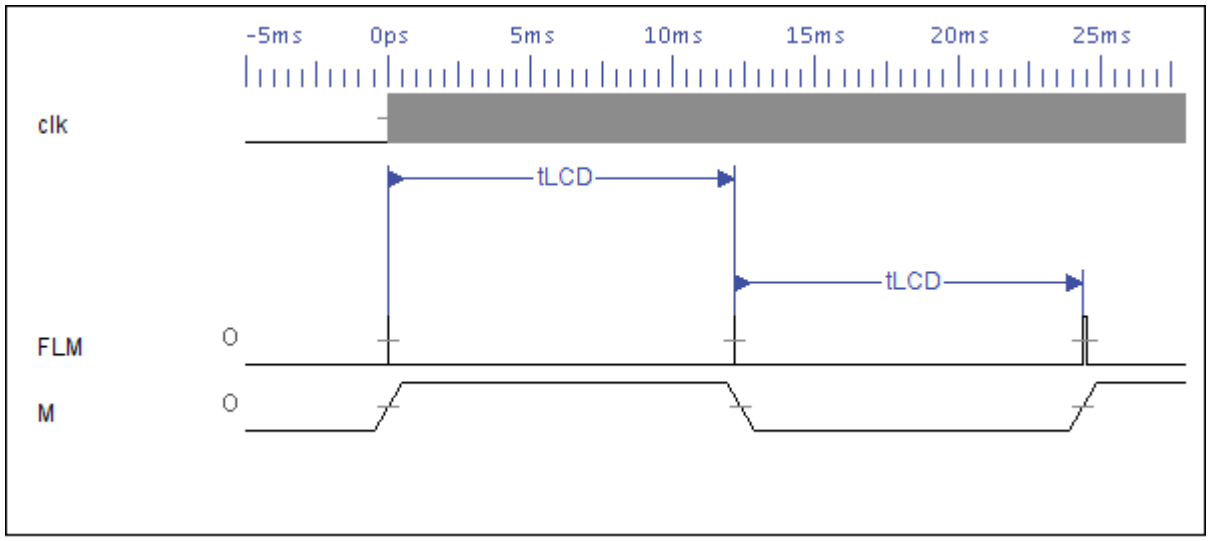


Figure 42 - Second Diagram for LCD Display Clocks

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX
tRC	Row cycle time	Time to fill an entire row of the LCD	0.19ms	0.23ms
tLCD	LCD cycle time	Time to fill the entire LCD	12.16ms	
tFS	FLM setup time		1us	
tFH	FLM hold time		100ns	
tCM	M delay time			300ns

Table 12 - Second Set of LCD Timing Parameters

These clocks are generated in the FPGA by using logic in a Quartus Block Diagram, as shown below. Note that the clocks are cleared every row clock until the acknowledge comes back from the VRAM controller. This is to ensure that all the data from the VRAM is ready to be transferred to the display every pixel clock.

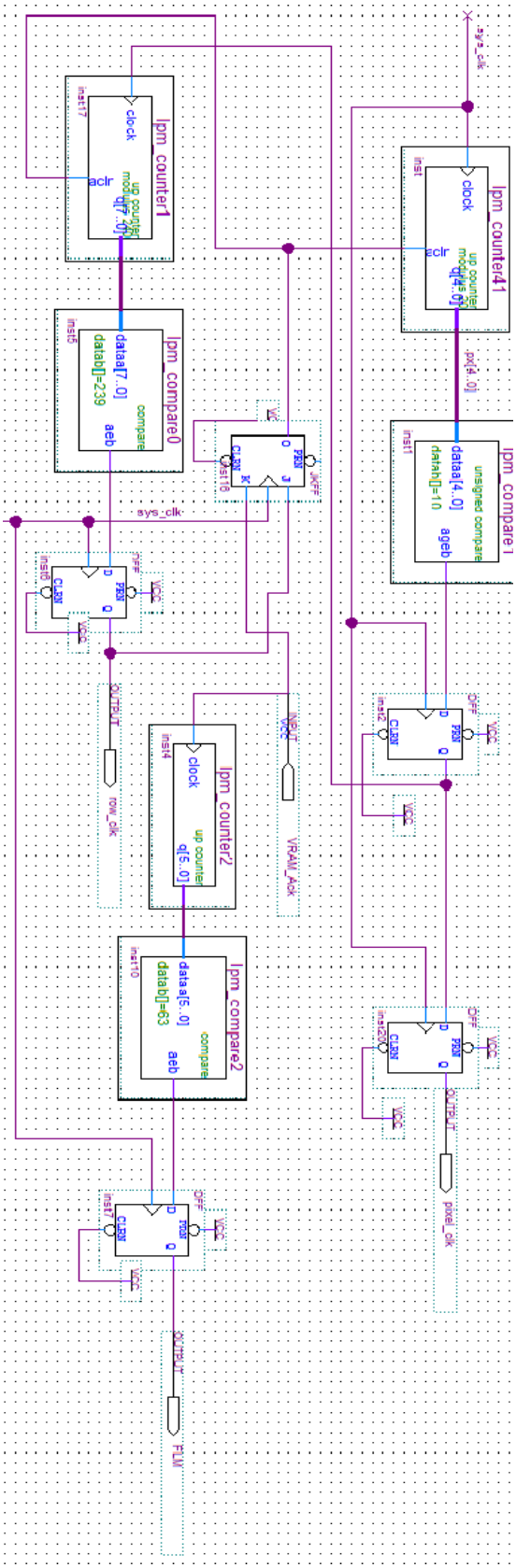


Figure 43 - Logic for Generating Pixel Clock, Row Clock, and FLM

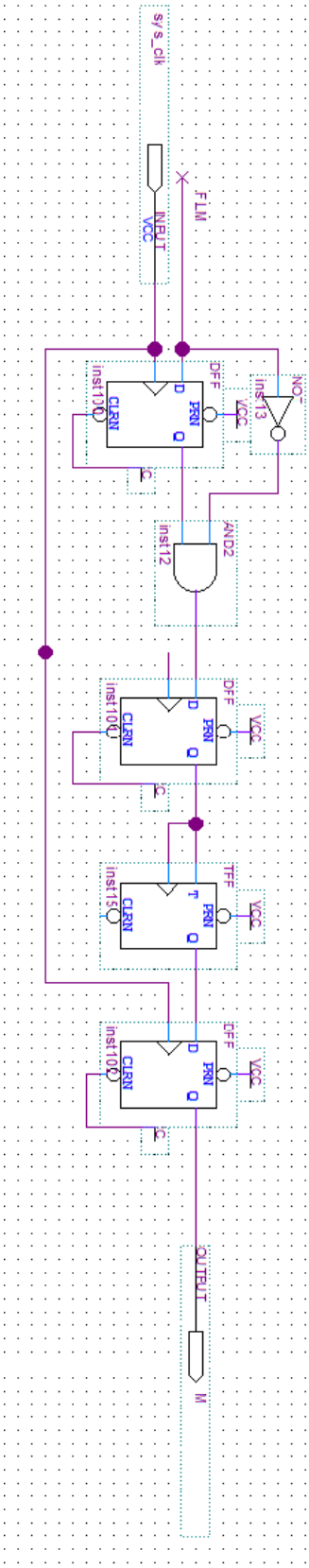


Figure 44 - Logic for Generating 'M' Display Clock Signal

In Figure 43, a counter and compare (lpm_counter41 and lpm_compare1) are used to generate the pixel clock. This pixel clock is high for ten system clocks and low for ten system clocks to comply with the timing parameters.

The row clock is generated by counting 240 pixel clocks (using lpm_counter1 and lpm_compare0), because there are 240 pixels per row of the display. When the row clock gets high, the logic clears the counters and continues to clear them until the display controller gets the acknowledge signal back from the VRAM controller saying that the serial row transfer cycle has completed. When this happens, the clocks continue to count. This is the function of the JK flip-flop.

The FLM signal is generated by counting the acknowledge signal from the VRAM controller (using lpm_counter2 and lpm_compare2). When 64 of these have occurred, FLM goes high (because there are 64 rows in the display)

In Figure 44, rising edge detection is used on FLM to toggle the M signal every FLM pulse using a toggle flip-flop.

The schematic for the display connector on the board (used to transfer the signals to the actual display) is shown below. The connector [**P9**] is a simple 12x2 header. There is a potentiometer to control the contrast on the display.

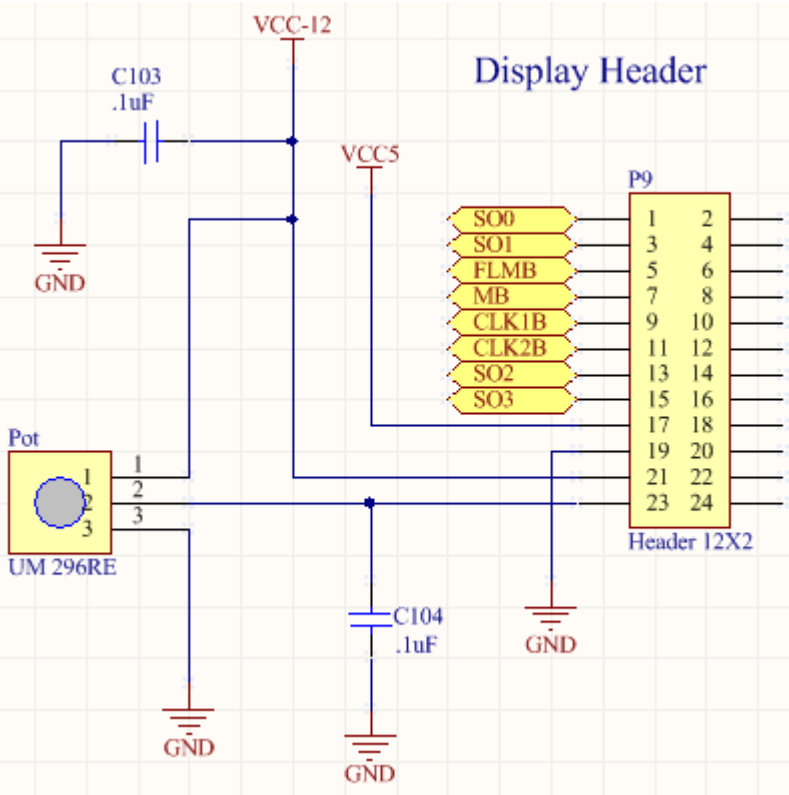


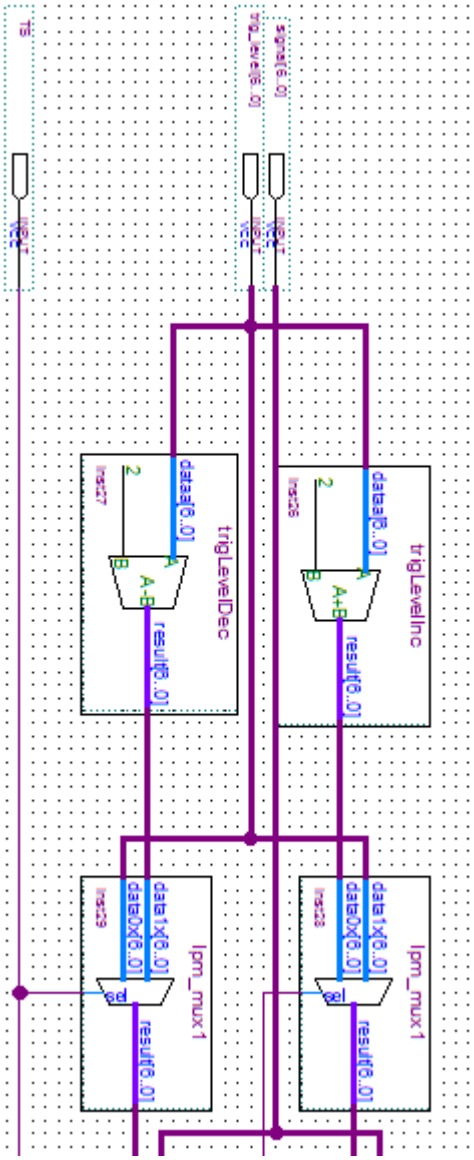
Figure 45 - Schematic for Display Connector and Potentiometer

Triggering Circuitry

The triggering circuitry is a set of FPGA logic that is applied to the incoming digital signal to determine when to start sampling. This can be divided into several different parts as follows.

Trigger Level & Slope Checking

The check to determine whether the signal has the appropriate trigger level and slope is done by a few comparators and a state machine written in VHDL. The system determines that there is a trigger when the signal falls within a certain window of the trigger level (the trigger level plus 2 bits if looking for a positive slope, or the trigger level minus 2 bits if looking for a negative slope). The logic for generating this new trigger level and determining whether the signal is greater or smaller than it is shown below.



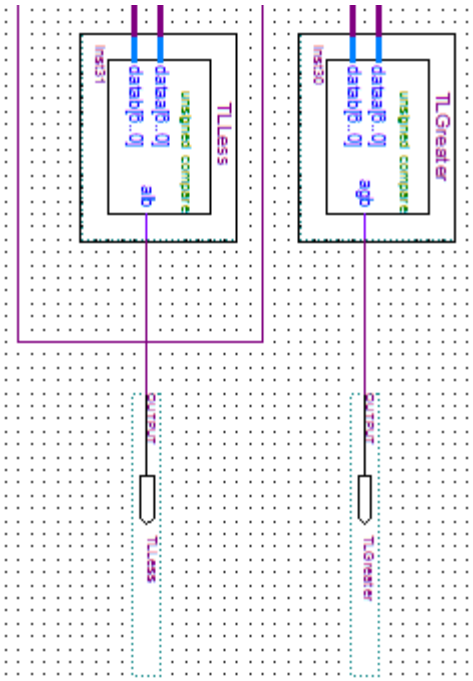


Figure 46 - Noise Reduction and Level Checking Logic

The outputs (TLLess and TLGreater) as well as the trigger slope are fed into the VHDL state machine to determine whether there is a trigger. The state machine code is shown below.

```
--------------------------------------------------------------------------------
--
--  Oscilloscope Digital Trigger
--
--  This is an implementation of a trigger for a digital oscilloscope in
--  VHDL.  There are three inputs to the system, one selects the trigger
--  slope and the other two determine the relationship between the trigger
--  level and the signal level.  The only output is a trigger signal which
--  indicates a trigger event has occurred.
--
--  The file contains multiple architectures for a Moore state machine
--  implementation to demonstrate the different ways of building a state
--  machine.
--
--
--  Revision History:
--     13 Apr 04  Glen George       Initial revision.
--      4 Nov 05  Glen George       Updated comments.
--     17 Nov 07  Glen George       Updated comments.
--     13 Feb 10  Glen George       Added more example architectures.
--     18 Jun 14  Daniel Andrade    Modified to reduce noise.
--------------------------------------------------------------------------------


-- bring in the necessary packages
library  ieee;
```

```
use  ieee.std_logic_1164.all;


--
--  Oscilloscope Digital Trigger entity declaration
--

entity  ScopeTrigger  is
    port (
        TS         :  in  std_logic;      -- trigger slope (1 -> negative, 0 ->
positive)
        TGT        :  in  std_logic;      -- signal level > trigger level
        TLT        :  in  std_logic;      -- signal level < trigger level
        clk        :  in  std_logic;      -- clock
        Reset      :  in  std_logic;      -- reset the system
        TrigEvent  :  out  std_logic      -- a trigger event has occurred
    );
end  ScopeTrigger;


--
--  Oscilloscope Digital Trigger Moore State Machine
--     State Assignment Architecture
--
--  This architecture just shows the basic state machine syntax when the state
--  assignments are made manually.  This is useful for minimizing output
--  decoding logic and avoiding glitches in the output (due to the decoding
--  logic).
--

architecture  assign_statebits  of  ScopeTrigger  is

    subtype  states  is  std_logic_vector(2 downto 0);     -- state type

    -- define the actual states as constants
    constant  IDLE      : states := "000";  -- waiting for start of trigger event
    constant  WAIT_POS  : states := "001";  -- waiting for positive slope trigger
    constant  WAIT_NEG  : states := "010";  -- waiting for negative slope trigger
    constant  TRIGGER   : states := "100";  -- got a trigger event


    signal  CurrentState  :  states;    -- current state
    signal  NextState     :  states;    -- next state

begin


    -- the output is always the high bit of the state encoding
    TrigEvent <= CurrentState(2);


    -- compute the next state (function of current state and inputs)

    transition:  process (Reset, TS, TGT, TLT, CurrentState)
    begin
```

```
        if  Reset = '1'  then           -- reset overrides everything
            NextState <= IDLE;          --   go to idle on reset
        end if;

        case  CurrentState  is          -- do the state transition/output

            when  IDLE =>               -- in idle state, do transition
                if  (TS = '0' and TLT = '1')  then
                    NextState <= WAIT_POS;      -- below trigger and + slope
                elsif  (TS = '1' and TGT = '1')  then
                    NextState <= WAIT_NEG;      -- above trigger and - slope
                else
                    NextState <= IDLE;          -- trigger not possible yet
                end if;

            when  WAIT_POS =>           -- waiting for positive slope trigger
                if  (TS = '0' and TLT = '1')  then
                    NextState <= WAIT_POS;      -- no trigger yet
                elsif  (TS = '0' and TLT = '0' and TGT = '0')  then
                    NextState <= TRIGGER;       -- got a trigger
                else
                    NextState <= IDLE;          -- trigger slope changed
                end if;

            when  WAIT_NEG =>           -- waiting for negative slope trigger
                if  (TS = '1' and TGT = '1')  then
                    NextState <= WAIT_NEG;      -- no trigger yet
                elsif  (TS = '1' and TGT = '0' and TLT = '0')  then
                    NextState <= TRIGGER;       -- got a trigger
                else
                    NextState <= IDLE;          -- trigger slope changed
                end if;

            when  TRIGGER =>            -- in the trigger state
                NextState <= IDLE;      -- always go back to idle

                        when others =>
                               NextState <= IDLE;

        end case;

    end process transition;


    -- storage of current state (loads the next state on the clock)

    process (clk)
    begin

        if  clk = '1'  then             -- only change on rising edge of clock
            CurrentState <= NextState;  -- save the new state information
        end if;

    end process;
end  assign_statebits;
```

Automatic Trigger Timeout

There is also logic added to trigger periodically if there has been no trigger and the system is in automatic mode. The timeout is set to a tenth of a second. This logic is shown below.

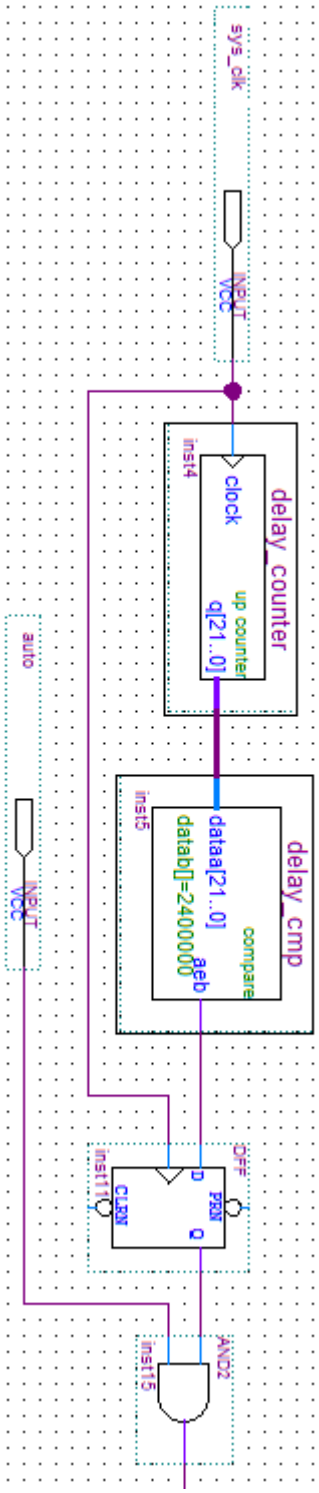
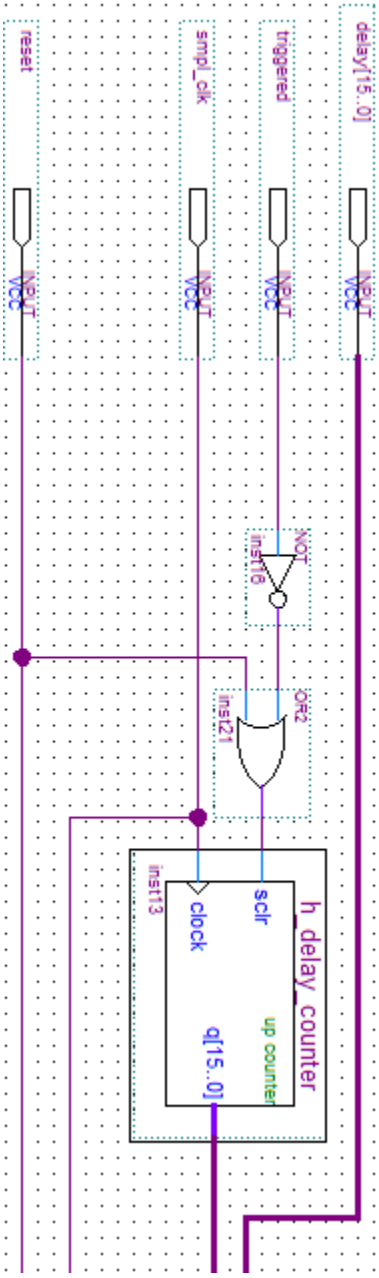


Figure 47 - Auto Trigger Timeout Logic

Trigger Delay Logic

The trigger delay logic enables the FIFO write line only after a certain amount of time (specified via parallel IO by the CPU) has passed since the actual trigger event. The trigger delay logic is shown below.



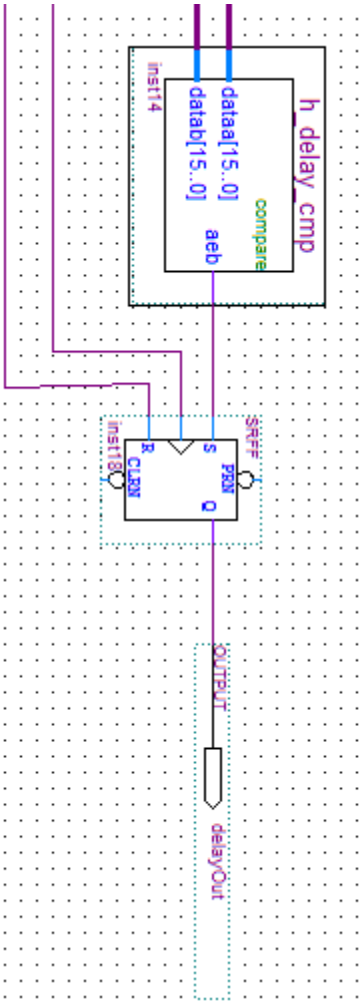


Figure 48 - Trigger Delay Logic

Trigger Holdoff Logic

The trigger holdoff logic resets the system after the FIFO fills (when a screen-full of samples has been taken) and keeps it reset for a fourth of a second. The holdoff_out line is ORed with the reset line so that either signal can reset the triggering logic. The logic for the trigger holdoff is shown below.

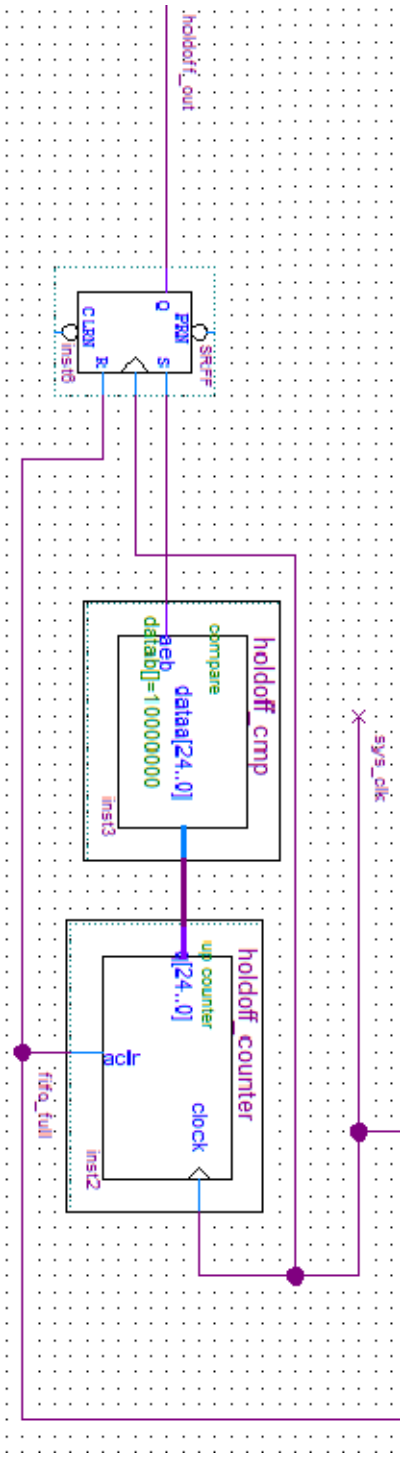


Figure 49 - Trigger Holdoff Logic

Sample Clock

The sample clock is generated using counter/compare logic. The number of system clocks per sample clock is computed in the CPU and output via parallel IO (smpl_rate). The sample clock is used to determine the rate at which to store the samples in the FIFO. The logic to generate the sample clock is shown below.

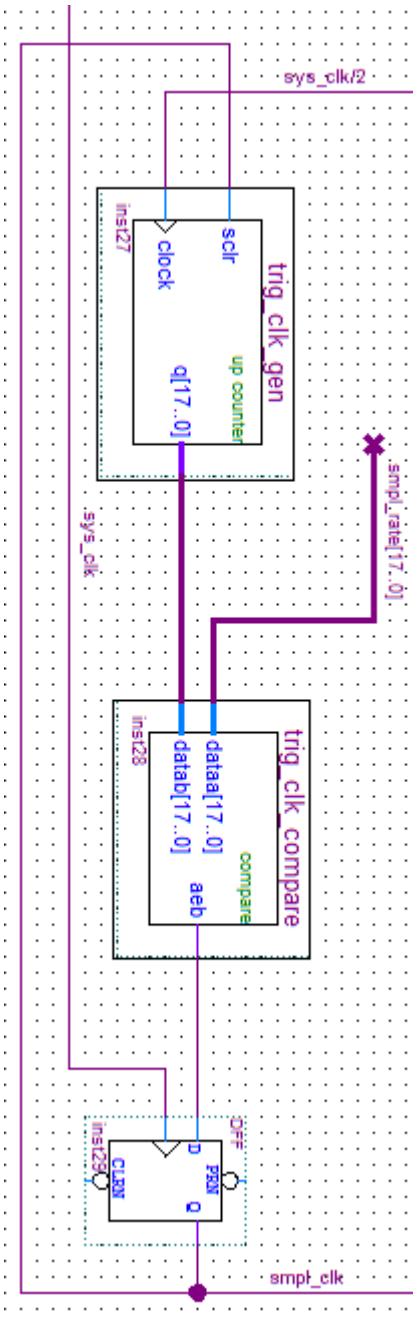


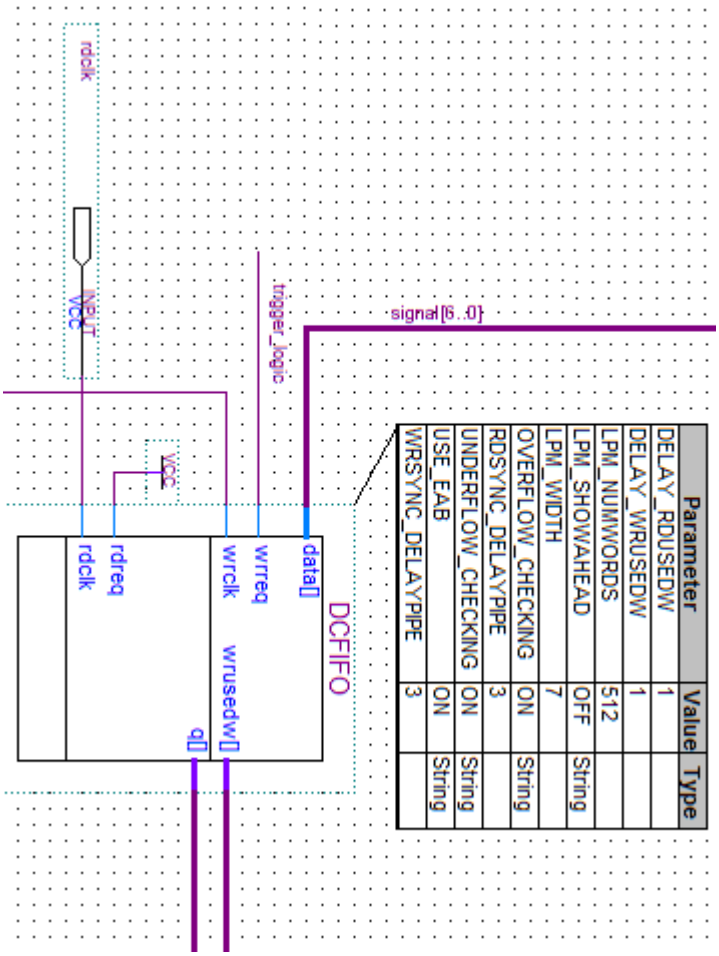
Figure 50 - Sample Clock Logic

FIFO Memory

The FIFO memory is part of the FPGA and is used as temporary storage for the digital samples. The write enable line of the FIFO is controlled by the trigger logic so that data is only written to the FIFO on a trigger. The read clock on the FIFO is controlled via parallel IO from the CPU.

The logic looks at the number of words used to determine when the FIFO is full (480 samples have been collected, because there are 480 pixels per row of the display) and uses this to interrupt the CPU so that it can start reading the samples. The FIFO cannot be written to again until it is completely empty.

The logic for the FIFO is shown below.



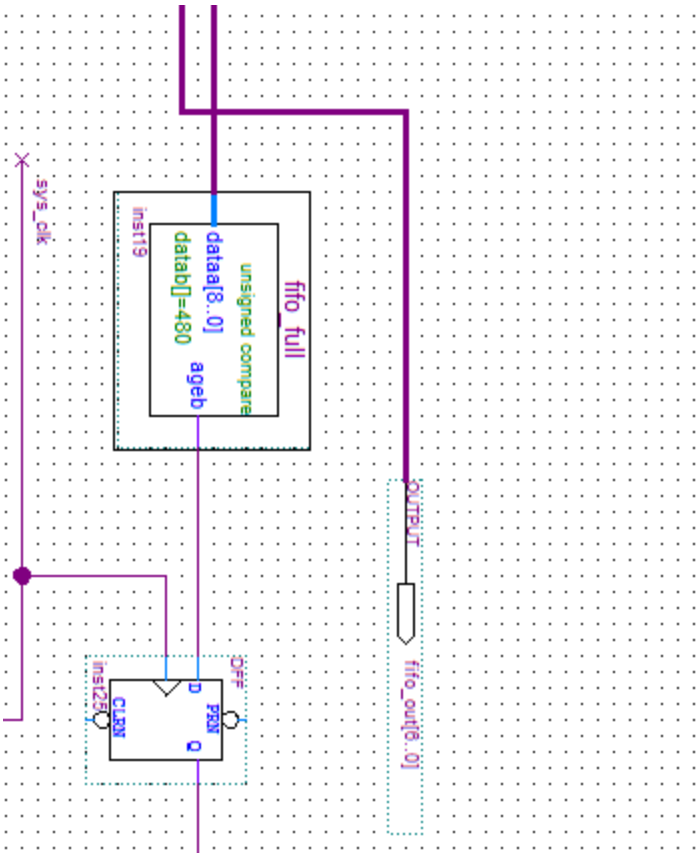


Figure 51 - Logic for FIFO Memory

ADC

The system uses a TLC5510A 8-bit 20MSPS ADC, which has an input range of 0 Volts to 4 Volts. The ADC is always sampled at the system's maximum sampling frequency, which is 12MHz. Two diodes ([**D2**] and [**D4**]) are used to protect the ADC. The schematic for the ADC is shown below.

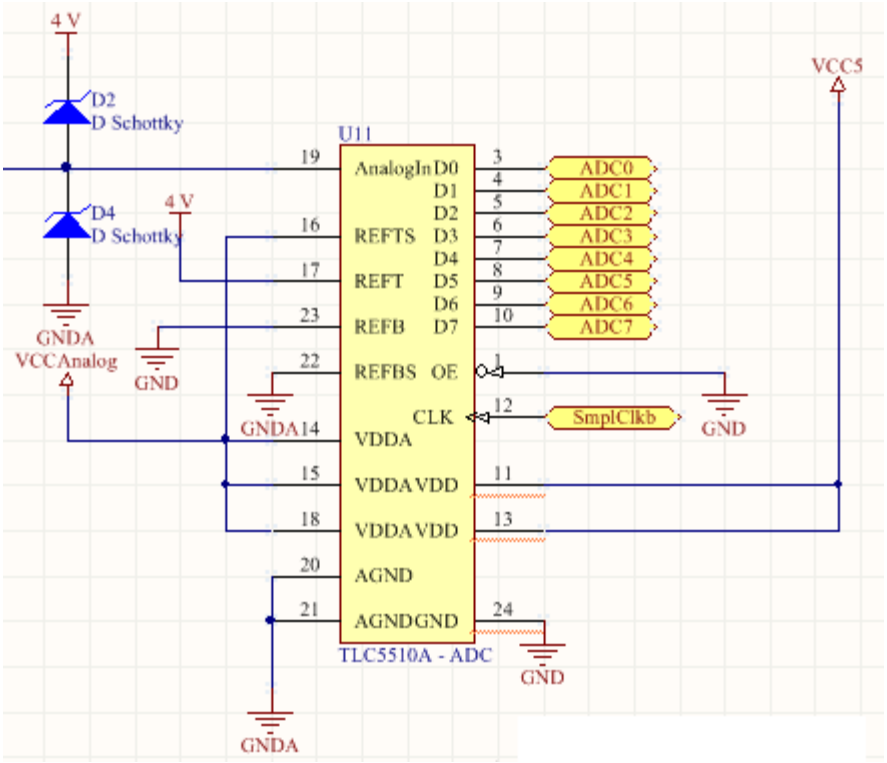


Figure 52 - ADC Schematic

Keys

The system has two mechanical push-button keys ([**SW1**] and [**SW3**]). [**SW1**] is a red pushbutton used as a reset button for the system. See the section on reset circuitry for more information.

[**SW3**] is used as the 'menu button' for the system to toggle the user interface menu on the screen. It is buffered and sent to the FPGA. The schematic for this switch is shown below.

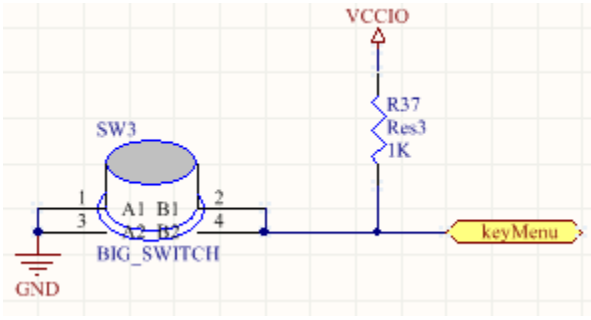


Figure 53 - Schematic for Menu Button

Rotary Encoder

The system also has a rotary encoder for user input ([**SW2**]). The rotary encoder is used to navigate the menu. Additionally, the rotary encoder has a built-in pushbutton used to toggle the way the software reads the direction of the encoder ("up and down" or "left and right"). The schematic for the rotary encoder is shown below.

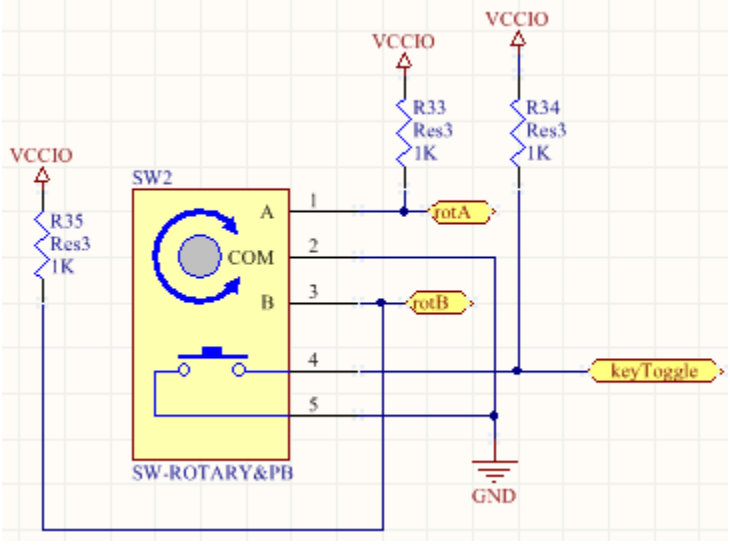


Figure 54 - Schematic for Rotary Encoder

Debouncers

Two debouncers (implemented in the FPGA) are used in the system. The debouncer for the toggle and menu keys is made with auto-repeat (approximately) every second, whereas the debouncer for the rotary encoder does not have an auto-repeat feature. The reset button is not debounced. The logic for the debouncer with auto-repeat is shown below.

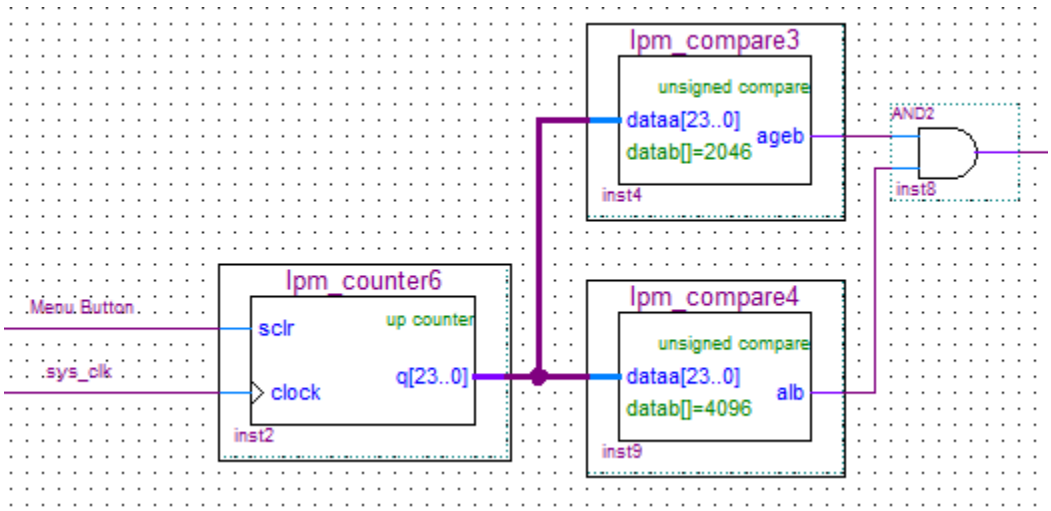


Figure 55 - Debouncer with Auto-Repeat

The button is used as the clear input to a counter. When the button is continuously low for 2046 system clocks (approx. 85 microseconds), the output signal is high. This signal stays high for another 2050 clocks, after which it goes low. This gives the signal a pulse-width of 4096 clocks or approximately 170 microseconds. If the button is held down, the signal will auto-repeat every 2^24 clocks or approximately every 0.7 seconds.

The logic for the debouncer without auto-repeat used for the encoder is shown below:

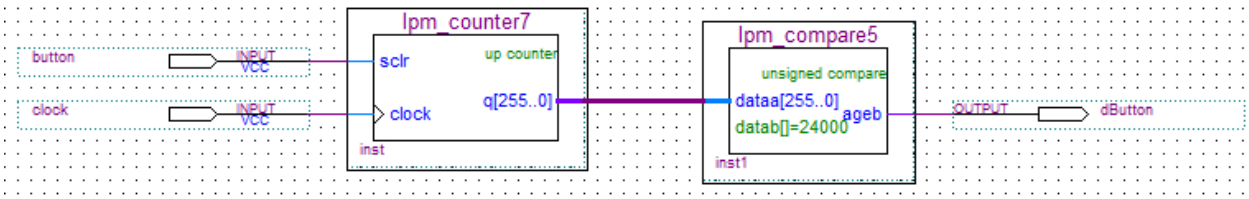


Figure 56 - Debouncer with no Auto-Repeat

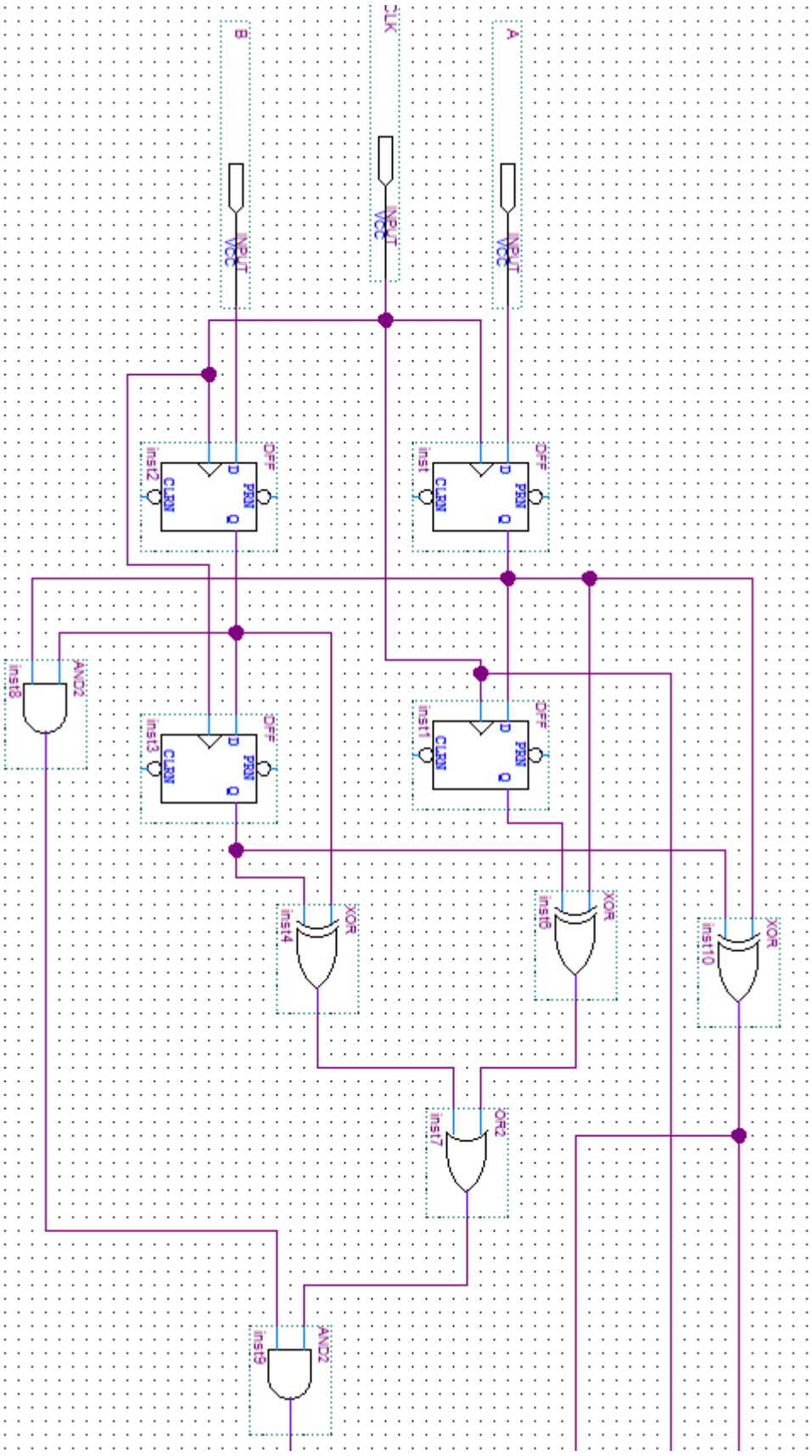
This debouncer works in a similar fashion to the other one, but it will not auto-repeat very frequently because the counter and compare used are 255 bits, so the auto-repeat rate is negligible. Additionally, the signal here will only stay high for one clock, it does not have an extended pulse-width like in the above debouncer.

Rotary Encoder Decoder

The rotary encoder decoder decodes the two debounced signals (A and B) coming from the encoder to determine whether the encoder is moving left, moving right, or not moving at all. The decoder allows the system to treat the rotary encoder output as two keys. The logic for the decoder is shown below (in two pages).

In the diagram, A and B are the two signals coming from the rotary encoder. The logic applied to these inputs checks whether the encoder was rotated (by doing any edge detection on the inputs) and also gets the direction of the rotation (by applying [A XOR previous B] OR [B XOR previous A]). It also applies a check to make sure that the current input is at a detent by seeing if the encoder's current A and B inputs are both high, which is the only detent the encoder has.

The outputs are LEFT and RIGHT. LEFT is high if the encoder moved and it moved to the left. RIGHT is high if the encoder moved and it moved to the right. Both of these signals remain high for only one clock.


CLK
B
A
INPUT
VCC
inst12
inst1
inst13
inst11
DFF
D
CLRN
PRN
Q
inst18
AND2
inst14
inst16
inst10
XOR
inst17
OR2
inst19

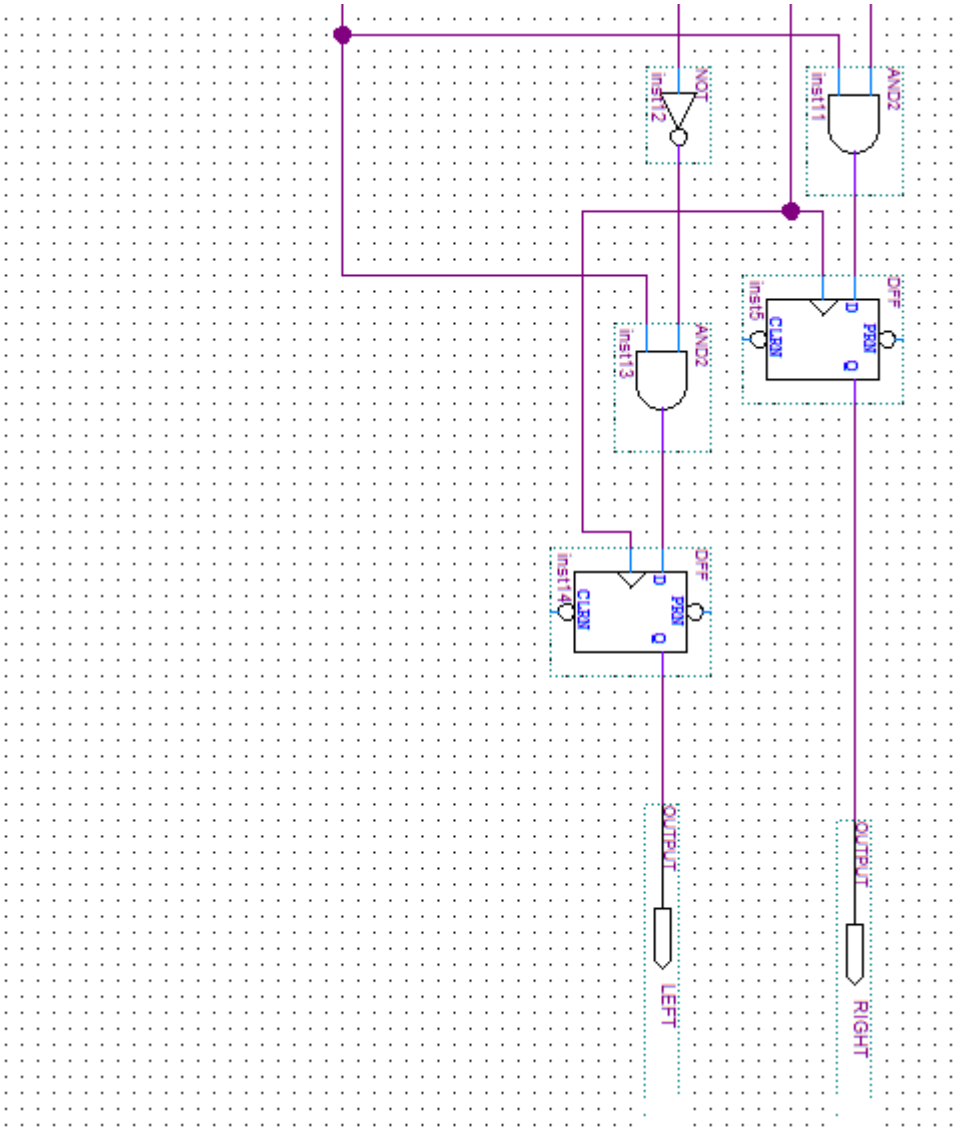


Figure 57 - Logic for the Rotary Encoder Decoder

Software Overview

The system's software is based on a continuous loop that gets samples, draws traces, and gets any key presses. This loop is in the file mainloop.c.

The first part of the loop checks whether a sample is currently being taken. If not, then it starts sampling and sets all the trigger parameters according to whatever the user has input into the user interface.

The second part of the loop checks whether the sample has just finished. If so, it plots the trace, clearing the entire display and redrawing everything, including the UI.

The third part of the loop checks to see if any key presses have occurred since the last check. If so, it updates the UI accordingly and redraws the screen if necessary.

In the flow diagram below, the curvy chevrons show associated files for each block. The functions shown in blue are conditional functions, meaning that the flow will not proceed to the next block unless that function returns a non-zero value.

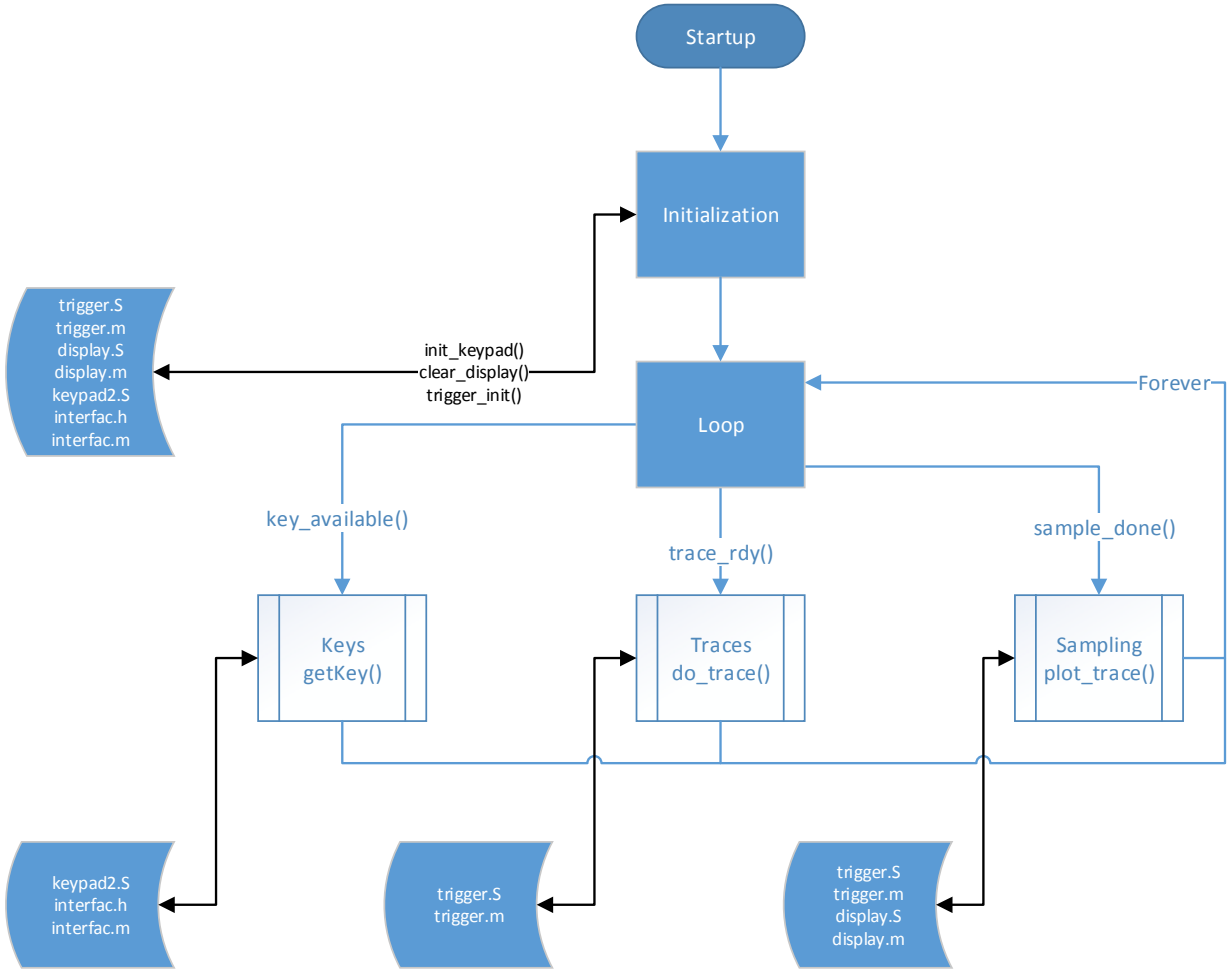


Figure 58 - Software Flow Diagram

Detailed Software Description

This section describes the system software used to interface with the hardware at a more detailed level. The software files are attached at the end of the document in Appendix G for reference and more detailed comments. The software used to interface with the hardware can be divided into three sections:

- User Input Functions (for keys and encoder)
- Display Functions
- Sampling/Triggering Functions

Each of these sections is described in greater detail below.

User Input Code

Symbols

- `periphBaseKey` – Base address of key PIO
- `periphBaseRot` – Base address of rotary encoder PIO
- `edgeCaptOffset` – Offset of the edge capture register in PIO
- `KEY_MENU` – Menu key press
- `KEY_UP` – Up key press
- `KEY_DOWN` – Down key press
- `KEY_LEFT` – Left key press
- `KEY_RIGHT` – Right key press

Shared Variables

- button_pressed – indicates whether a button has been pressed (TRUE) or not (FALSE). Set in interrupt handler, read in key_available.
- current_button – indicates which button (if any) was pressed last, using the symbols specified above.
- toggled – indicates whether the encoder is toggled (TRUE) or not (FALSE).

Initialization

The input functions are initialized by the function **keypad_init().** This function initializes the shared variables and install the interrupt handlers for the keys and the encoder. This function also unmasks the interrupts.

Interrupt Handlers

There are two interrupt handlers, one for key presses (**key_interrupt_handler()**) and another for the encoder (**rot_interrupt_handler()**).These interrupt handlers determine which key was pressed and set the shared variables appropriately.

Polling Functions

The first polling function (**unsigned char key_available()**) determines whether a key has been pressed since the last check by reading the button_pressed shared variable.

Other Functions

The function **int getKey()** is only called if key_available() returned TRUE, and it returns whichever key was pressed last by reading the current_button shared variable.

Display Code

Symbols

- `colsPerRow  - Columns per row of VRAM`
- `pixPerQH    - Pixels per horizontal quadrant of the display`
- `pixPerQV    - Pixels per vertical quadrant of the display`
- `PIXEL_BLACK - Constant for writing a black pixel to the display`
- `PIXEL_WHITE - Constant for writing a white pixel to the display`

Shared Variables

None.

Clearing the Display

The function **clear_display()** clears the display by written PIXEL_WHITE to every location in VRAM.

Plotting a Pixel

The function **plot_pixel(unsigned int x, unsigned int y, int p)** plots a pixel of color p to the display. First, this function determines the correct byte to write to in VRAM. Each byte in VRAM draws to four pixels, one in each quadrant of the display. Therefore, some bit logic is applied to that existing byte to avoid overwritten what is already on the display. This bit logic is different depending on whether the pixel written is black or white.

This function additionally corrects for some timing issues in the display. In particular, pixels were drawn two rows lower than they should have been, so this was taken into account when calculating the location to write to in VRAM.

Sampling/Triggering Functions

Symbols

- clkFreq - Frequency of sample clock sent to ADC
- sampleSize - Size of points per sample (equal to MAX_SAMPLE_SIZE)
- rateAddress - Address of PIO for storing sample rate information
- tSAddress - Address of PIO for storing trigger slope information
- tLAddress - Address of PIO for storing trigger level information
- startAddress - Address of PIO for trigger reset
- autoAddress - Address of PIO for storing whether auto trigger is on
- rdClkAddress - Address of PIO for the read clock generated by the CPU
- fifoDataAddress – Address of PIO for the FIFO output data
- delayAddress - Address of PIO for storing the trigger delay information
- fifoIntrAddress – Address of PIO for the FIFO interrupt line
- MIN_LEVEL - Minimum trigger level
- MAX_LEVEL - Maximum trigger level
- MIN_DELAY - Minimum trigger delay
- MAX_DELAY - Maximum trigger delay

Shared Variables

- sample_buffer – An array used to temporarily store samples before they are drawn to the screen.
- sample_flag – A flag indicating whether the current sample is done.

Initialization

The **trigger_init()** function installs and unmasks the interrupt handler for the FIFO interrupt.

Interrupt Handler

The **trigger_intr()** function serves as the interrupt handler. It resets the trigger so that the FIFO will not continue filling up and loops sampleSize times, reading values from the FIFO by pulsing the read clock and looking at the PIO with FIFO data in it, while storing each sample in the buffer. When it is done, it sets the sample_flag to let the system know that the sample is completed.

Trigger Variable Setting Functions

The **int set_sample_rate(long int rate)** function sets the sample rate (in samples per second) and returns the number of samples to be taken at this rate.

The **set_trigger(int level, int slope)** function sets the trigger level and slope, with 0 being positive slope 1 being negative slope, and level ranging from 0 to 127.

The **set_delay(long int delay)** function sets the trigger delay (in samples).

Polling Functions

The **unsigned char* sample_done()** function is a polling function that returns a pointer to the sample buffer if the sample is done and returns NULL otherwise. This is implemented by looking at the sample_flag variable to determine if the sample is done. If not, NULL is returned. Otherwise, the sample_flag is reset and a pointer to the buffer is returned.

Other Functions

The **sample_start(int auto)** function starts the sample. The reset line on the trigger is pulled low to enable the trigger logic to start looking for a sample, and the auto line is pulled high or low depending on whether the system should be auto-triggered.

Changes to Design

This section describes any changes that were made to the original design when building the system.

Permanent changes are changes that were made to the system in order to make it function. Without these changes, the system could not have possible worked as specified. These changes can be also be seen by comparing the revised schematics in Appendix A to the original schematics in Appendix C.

Nonpermanent changes were made due to some hardware problems that should not occur if the system is built again (for example, to avoid dealing with shorts). In other words, there was no design problem with the original design, just the implementation. Therefore these changes are not shown in the revised schematics, but are documented here for reference.

Permanent Changes

JTAG Configuration

In the original schematics, TMS was pulled high to VCCA via the resistor [**R1**] and TDO went straight to the header. In fact, these two should have been flipped, with TDO being pulled high through [**R1**] and TMS going straight to the header. This was fixed on the board by physically rotating [**R1**] by approximately ninety degrees.

FPGA Configuration

In the original schematics, INIT_DONE was left floating on a header. This signal was tied high through a 10kΩ resistor to VCCIO so that the FPGA could be programmed properly.

In the original schematics, nCE was left floating on a header. This signal was tied low to GND through a wire so that the FPGA could be programmed properly.

Reset

In the original schematics, the RESET signal on the reset chip was solely connected to the FPGA. This signal was also connected through a wire to CONF_DONE in order to have the RESET signal actually reset the system.

Resistor Values for Input Signals

The 10kΩ resistors used for the pushbutton and encoder signals appeared to be too large for the impedance of the buffer, and therefore the signal did not appear to be high when it should have been, so these resistors were replaced with 1kΩ resistors.

Shottky Diode Direction

The direction of diodes [**D1**] and [**D2**] was the opposite of what it should have been. This was fixed in implementation by rotating the diodes.

Nonpermanent Changes

Re-routed Data Bus

The data bus did not always behave as expected after going through the data buffer [**U17**]. Sometimes the signals would show the correct behavior, but otherwise the signals would be high or low unexpectedly. In the interest of saving time, the data bus was re-routed to IO pins 30-37 and then wire-wrapped to the correct PCB location.

Re-routed Memory Control Signals

All the memory control signals passing through the buffer [**U13**] (chip selects for RAM and ROM, the read/write signal, and address line sixteen) did not always act as designed. As with the data bus, these signals would sometimes show the correct behavior, but could also be unexpectedly high or low at other times. The signals were reassigned to IO pins on the same buffer as follows and then wire-wrapped to the memory chips.

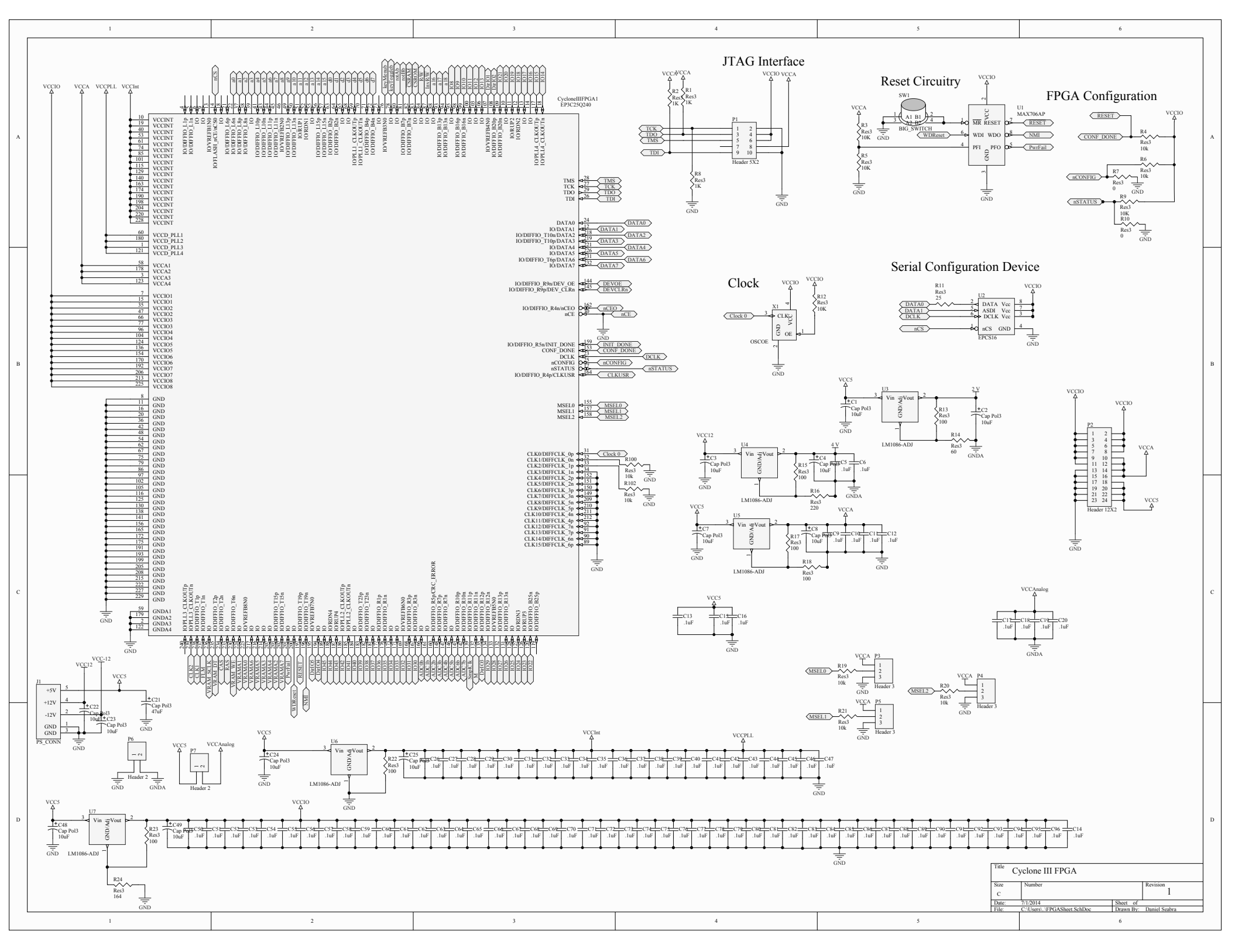
Signal Name/Description	New Location
RAM Chip Select	IO Pin 10
ROM Chip Select	IO Pin 11
Address Line 16	IO Pin 8
Read/Write Signal	IO Pin 9

Table 13 - Remapped Signals on Board

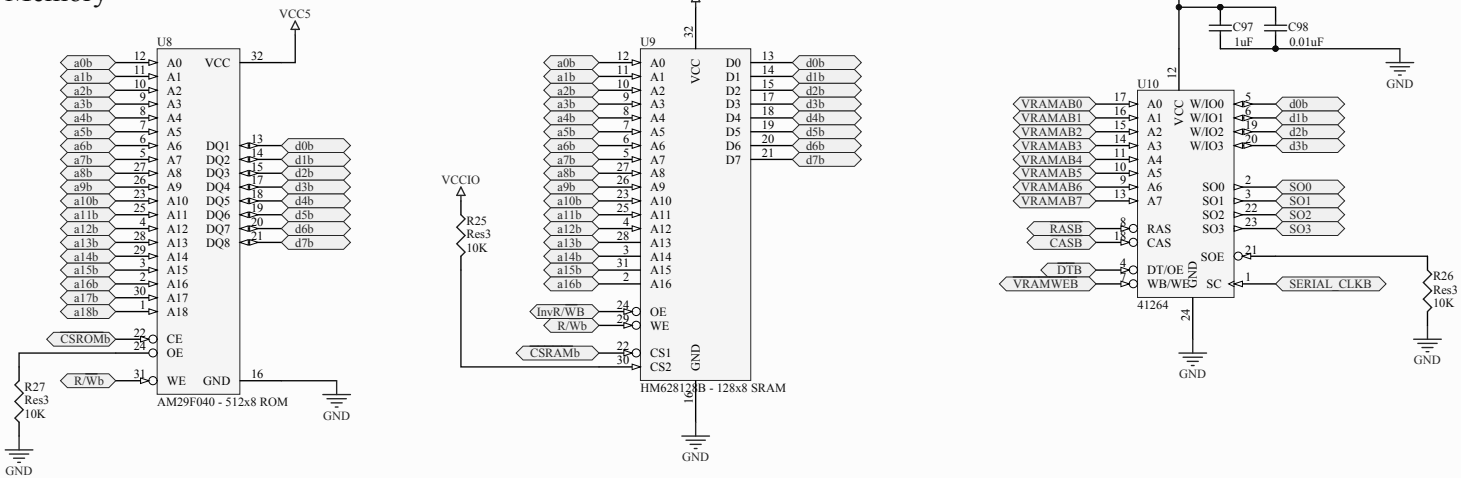
Permanent Omissions

The analog scaling front-end portion of the project was skipped in order to save time. Thus, resistor [**R98**] was left unconnected and the header [**P8**] was shorted. This effectively sends the signal from the BNC connector to the ADC while keeping the 0-4V Shottky diode protection.

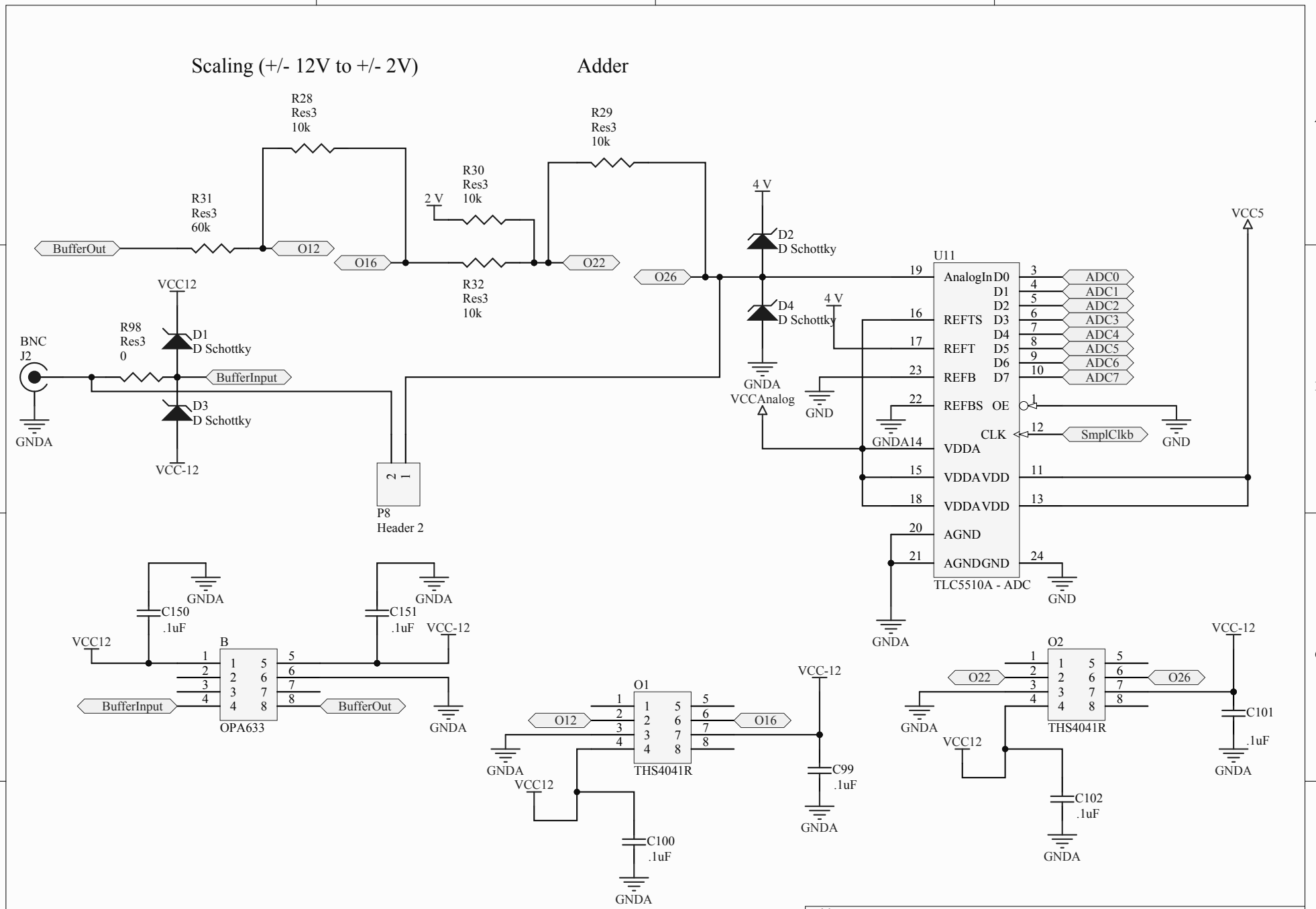
Appendix A - Revised Schematics


JTAG Interface
Reset Circuitry
FPGA Configuration
Clock
Serial Configuration Device
CycloneIIIFPGA1
EP3C25Q240
Header 5X2
BIG_SWITCH
MAX706AP
EPCS16
LM1086-ADJ
PS_CONN
Header 2
Header 3
Header 12X2
Cyclone III FPGA
Size
C
Number
Revision
1
Date:
7/1/2014
Sheet of
File:
C:\Users\...\FPGASheet.SchDoc
Drawn By:
Daniel Seabra

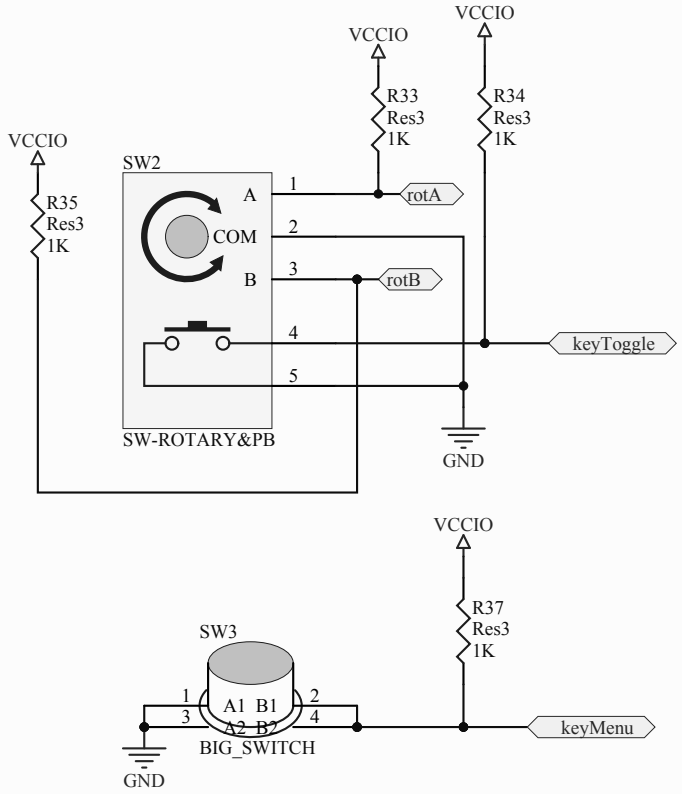
Memory

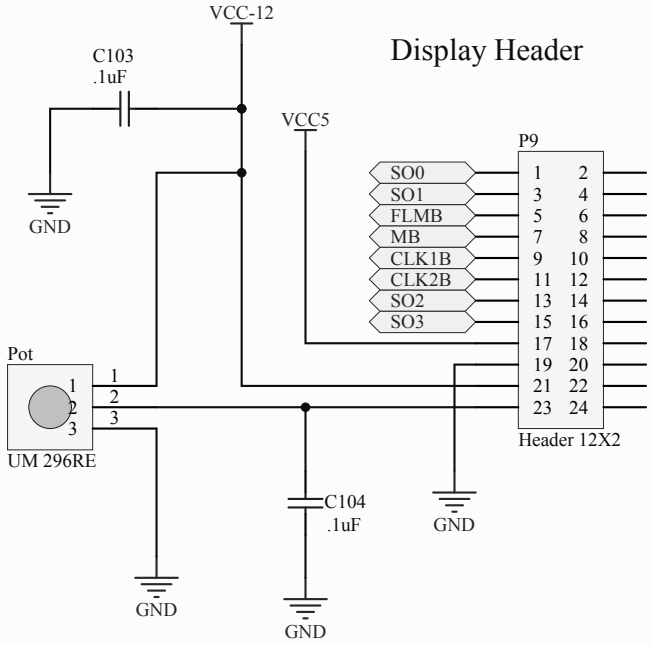


Title	Scope Memory	
Size: B	Number:	Revision: 1
Date: 7/1/2014		Sheet of
File: C:\Users\..\ScopeMemory.SchDoc		Drawn By: Daniel Seabra

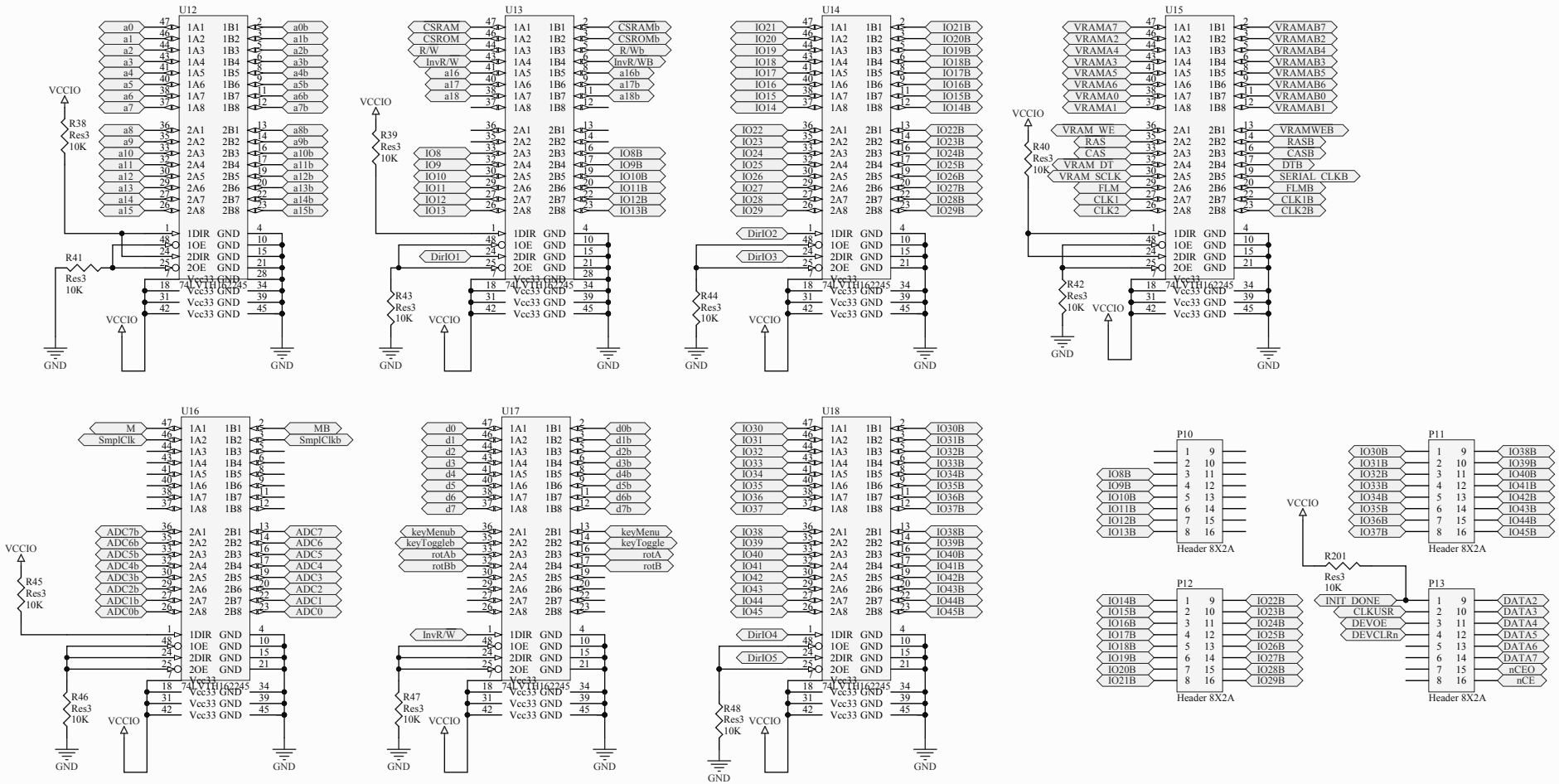


Title	Analog Front End	
Size: A	Number:	Revision: 1
Date: 7/1/2014		Sheet of
File: C:\Users\..\FrontEnd.SchDoc		Drawn By: Daniel Seabra



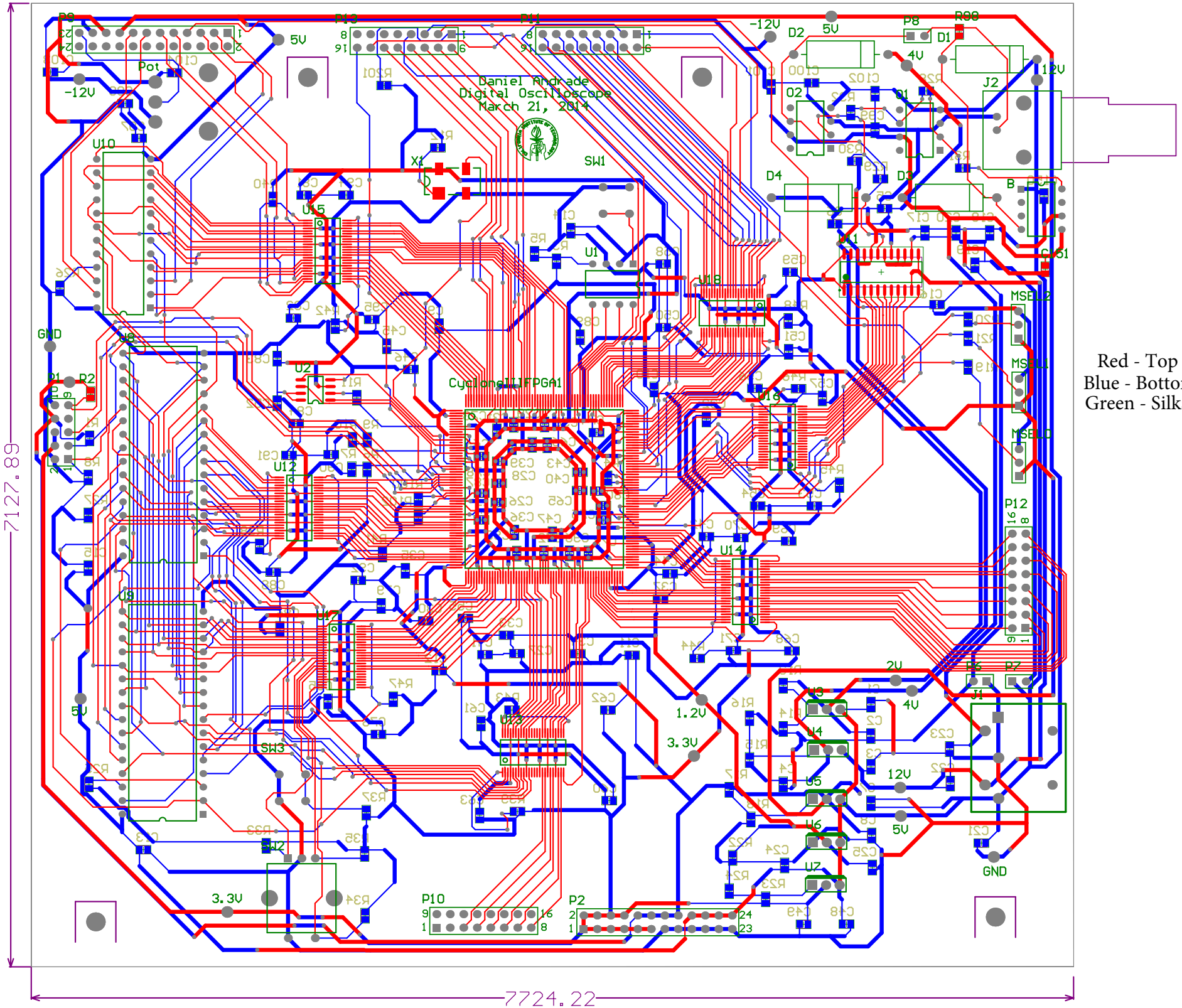


Title: Peripheals		
Size: A	Number:	Revision: 1
Date: 7/1/2014		Sheet of
File: C:\Users\..\Peripheals.SchDoc		Drawn By: Daniel Seabra

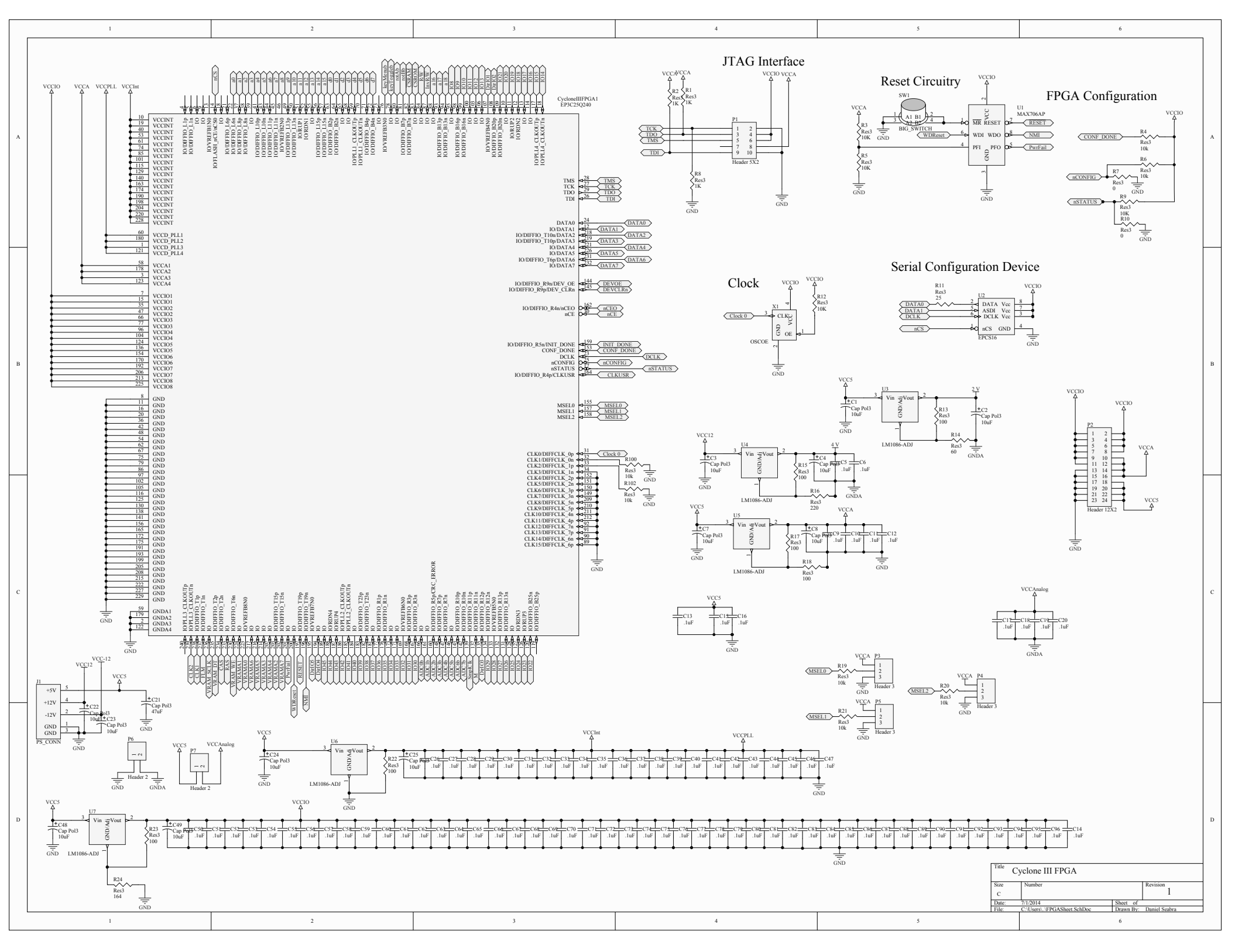


Title: Buffers		
Size: B	Number:	Revision:
Date: 7/1/2014		Sheet of
File: C:\Users\..\Buffers.SchDoc		Drawn By: Daniel Seabra

Appendix B - Altium PCB Layout

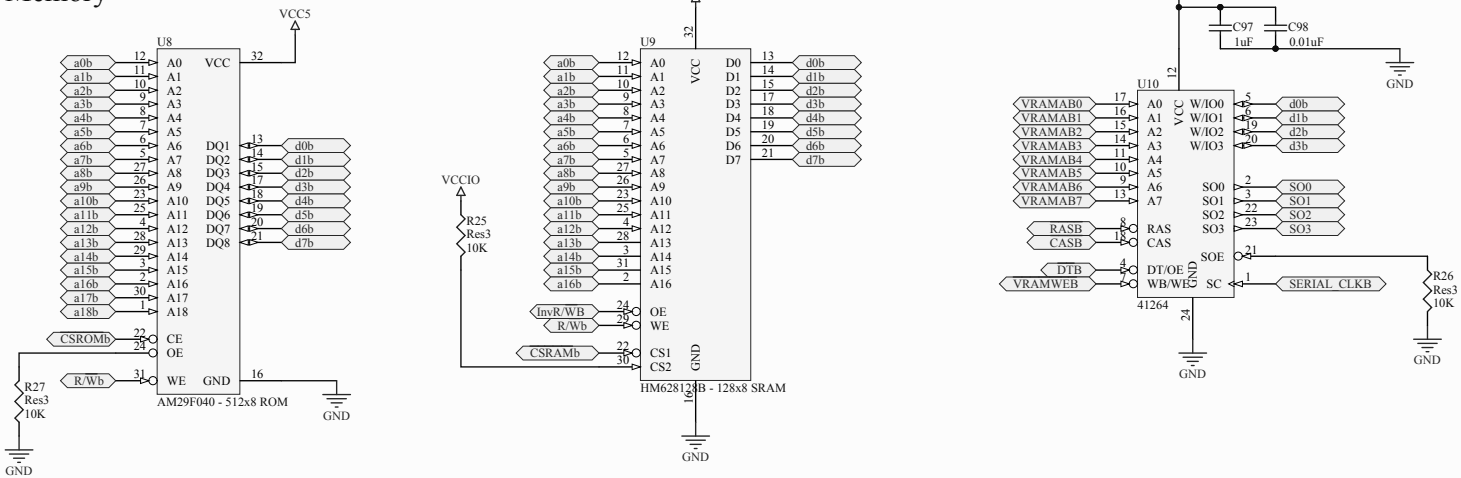

Daniel Andrade
Digital Oscilloscope
March 21, 2014
CycloneIIFPGA1
7127.89
7724.22
Red - Top Layer
Blue - Bottom Layer
Green - Silkscreen

Appendix C - Original Schematics

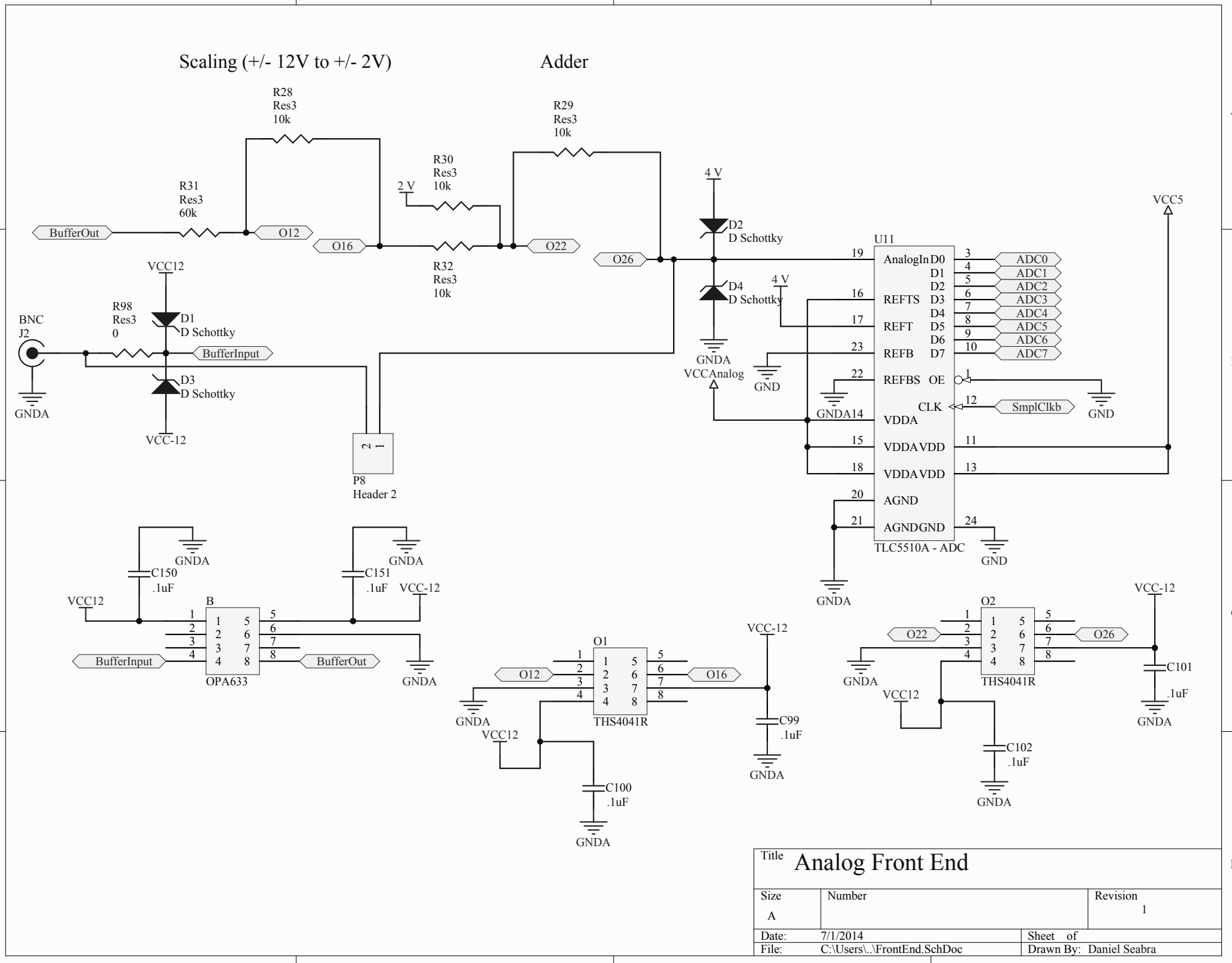


Title	Cyclone III FPGA	
Size: C	Number	Revision: 1
Date: 7/1/2014		Sheet of
File: C:\Users\...\FPGASheet.SchDoc		Drawn By: Daniel Seabra

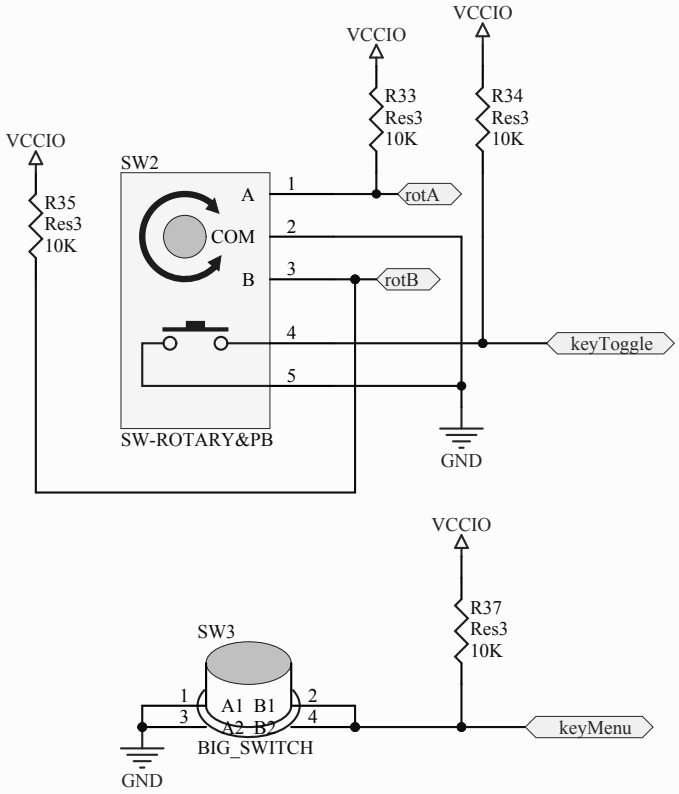
Memory

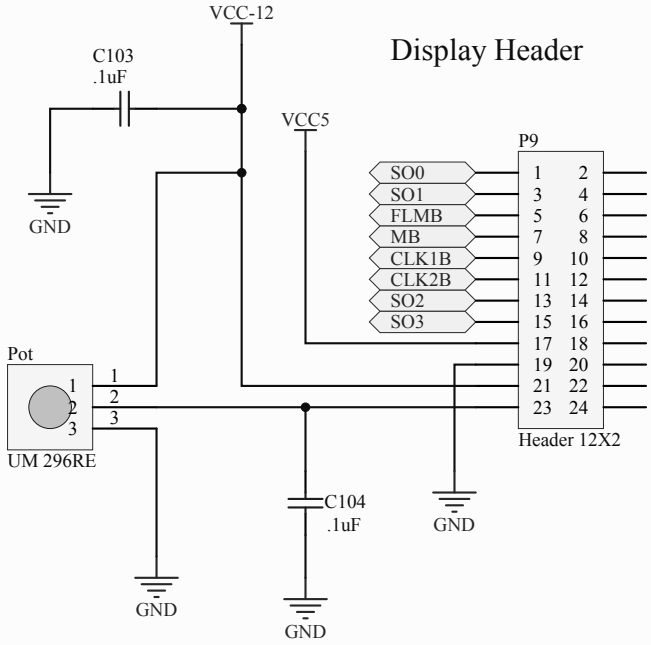


Title	Scope Memory	
Size: B	Number:	Revision: 1
Date: 7/1/2014		Sheet of
File: C:\Users\..\ScopeMemory.SchDoc		Drawn By: Daniel Seabra

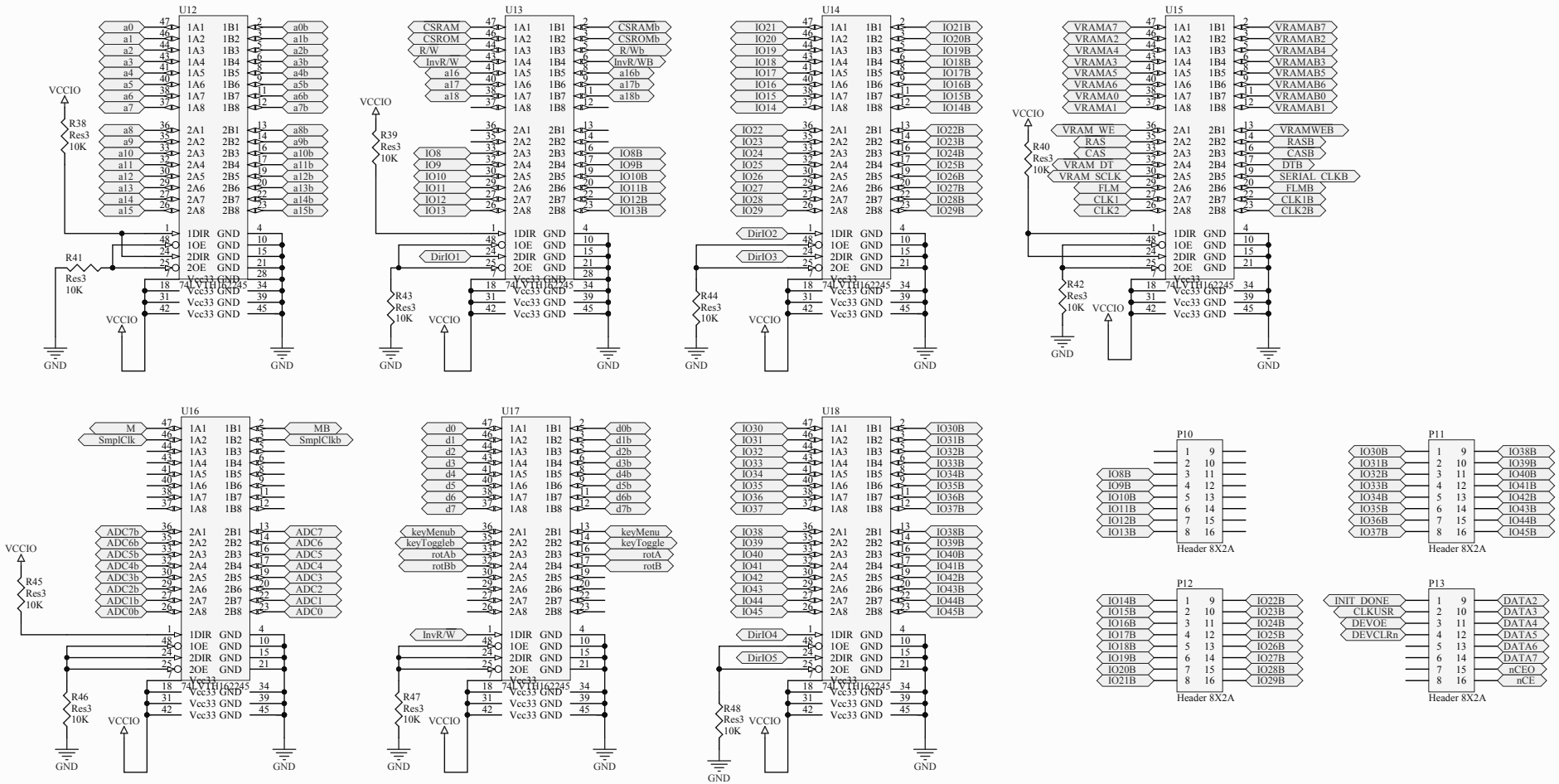


Title	Analog Front End	
Size: A	Number:	Revision: 1
Date: 7/1/2014		Sheet of
File: C:\Users\..\FrontEnd.SchDoc		Drawn By: Daniel Seabra



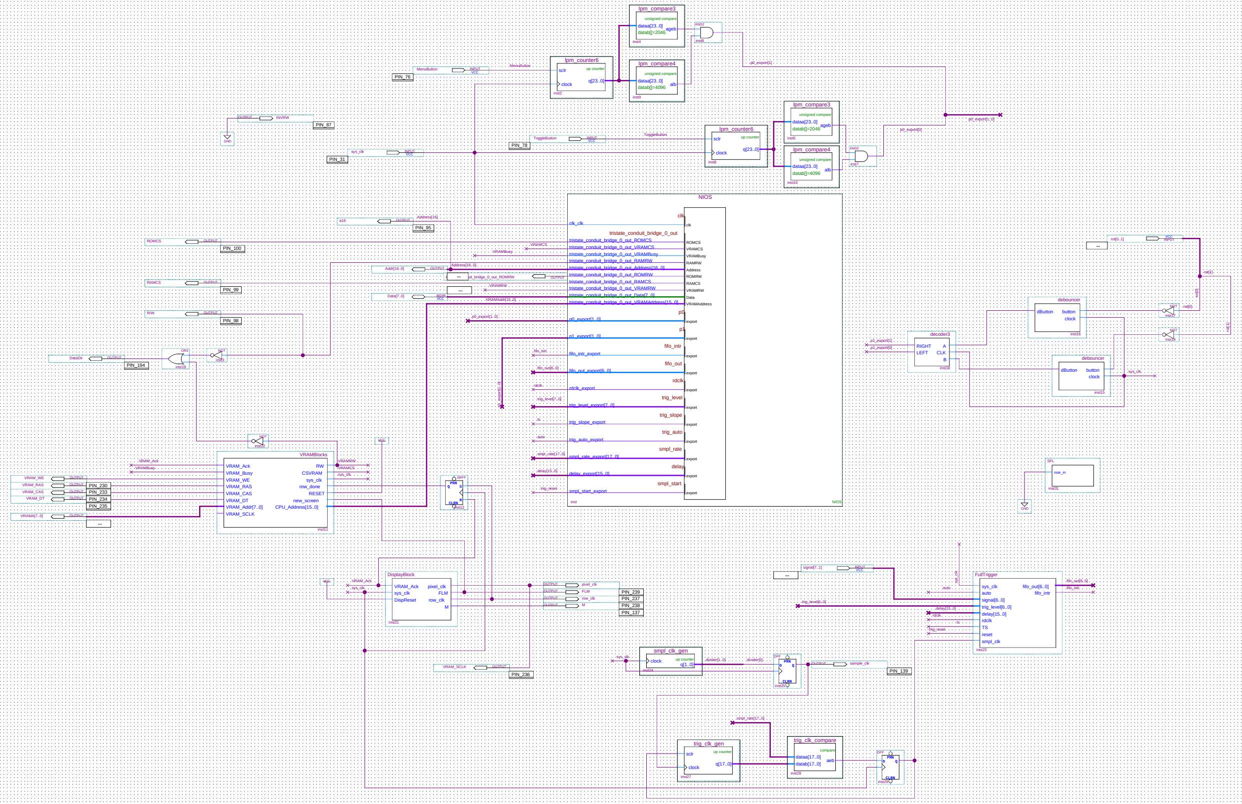


Title: Peripheals		
Size: A	Number:	Revision: 1
Date: 7/1/2014		Sheet of
File: C:\Users\..\Peripheals.SchDoc		Drawn By: Daniel Seabra

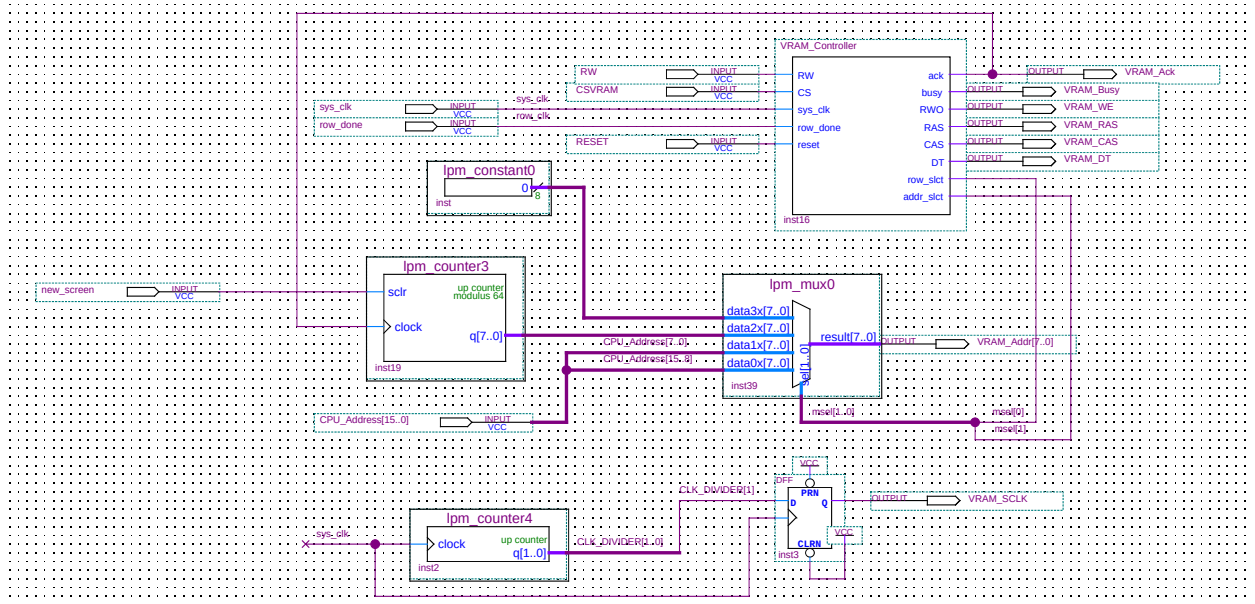


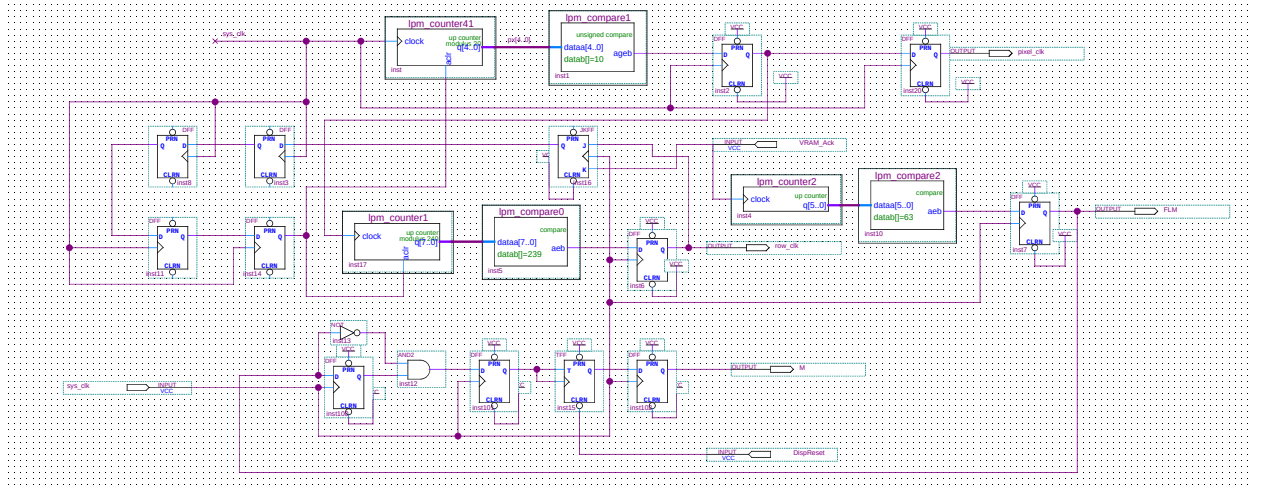
Title: Buffers		
Size: B	Number:	Revision:
Date: 7/1/2014		Sheet of
File: C:\Users\..\Buffers.SchDoc		Drawn By: Daniel Seabra

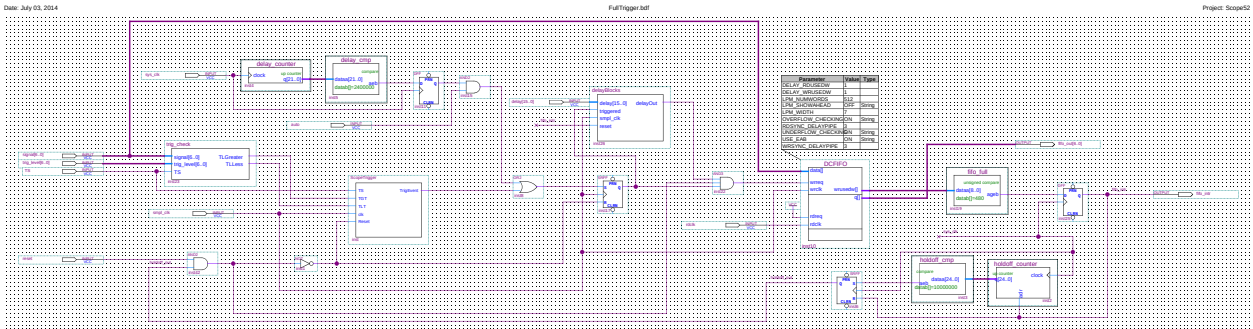
Appendix D - Quartus Block Diagrams

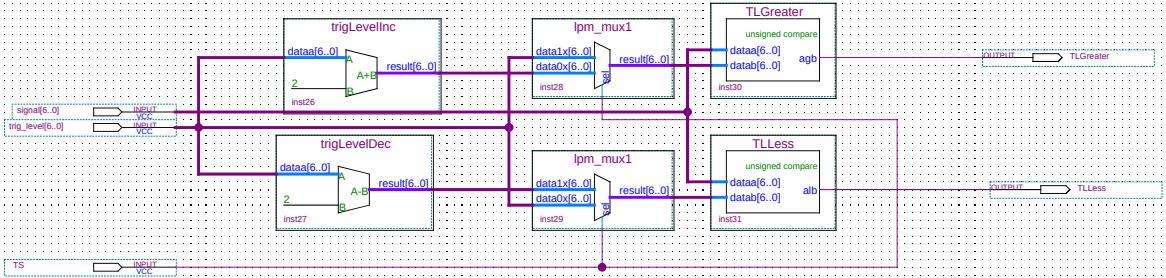


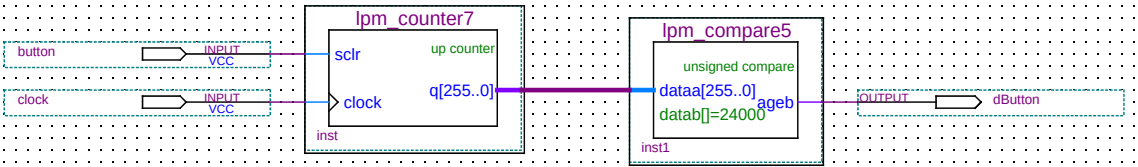
lpm_compare3
lpm_compare4
lpm_counter6
NIOS
VRAMBlocks
DisplayBlock
FullTrigger
smpl_clk_gen
trig_clk_gen
trig_clk_compare
decoder3
debouncer
PIN_76
PIN_87
PIN_78
PIN_31
PIN_95
PIN_100
PIN_99
PIN_98
PIN_194
PIN_230
PIN_233
PIN_234
PIN_235
PIN_239
PIN_237
PIN_238
PIN_137
PIN_236
PIN_139

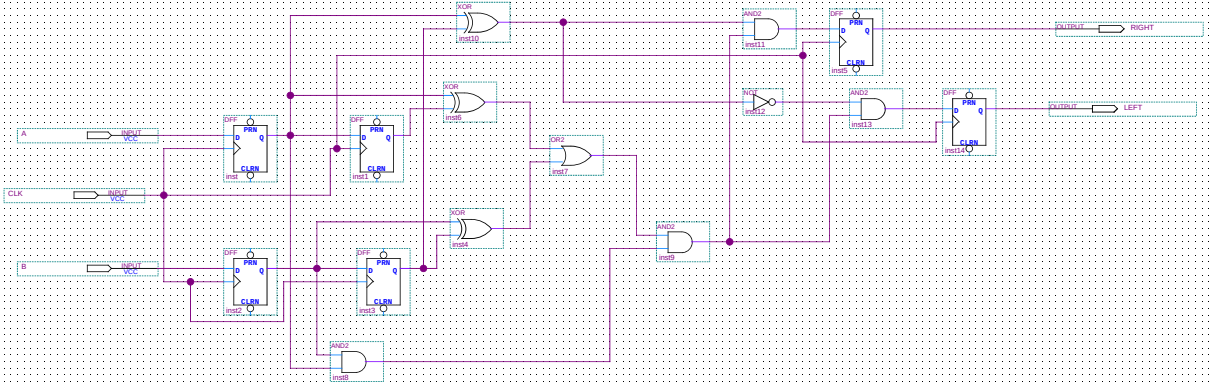

VRAM_Controller
RW
CS
sys_clk
row_done
reset
ack
busy
RWO
RAS
CAS
DT
row_slct
addr_slct
inst16
INPUT
VCC
OUTPUT
VRAM_Ack
VRAM_Busy
VRAM_WE
VRAM_RAS
VRAM_CAS
VRAM_DT
CSVRAM
RESET
lpm_constant0
inst
lpm_counter3
sclr
clock
up counter
modulus 64
q[7..0]
inst19
new_screen
lpm_mux0
data3x[7..0]
data2x[7..0]
data1x[7..0]
data0x[7..0]
sel[1..0]
result[7..0]
inst39
VRAM_Addr[7..0]
CPU_Address[7..0]
CPU_Address[15..8]
CPU_Address[15..0]
msel[1..0]
msel[0]
msel[1]
OFF
PRN
D
Q
CLRN
inst3
VRAM_SCLK
CLK_DIVIDER[1]
lpm_counter4
up counter
CLK_DIVIDER[1..0]
inst2


lpm_counter41
clock
up counter
modulus[4..0]
q[4..0]
inst
lpm_compare1
unsigned compare
dataa[4..0]
datab[]=10
ageb
inst1
sys_clk
INPUT
VCC
DFF
PRN
CLRN
OUTPUT
pixel_clk
inst20
VRAM_Ack
lpm_counter2
clock
up counter
q[5..0]
inst4
lpm_compare2
compare
dataa[5..0]
datab[]=63
aeb
inst10
FLM
row_clk
inst8
lpm_counter1
clock
up counter
modulus[7..0]
q[7..0]
inst17
lpm_compare0
compare
dataa[7..0]
datab[]=239
aeb
inst5
inst11
inst14
inst16
inst3
inst7
inst13
AND2
inst12
inst15
inst18
inst19
M
DispReset


delay_counter
clock
sr counter
q[21..0]
aclr
delay_cmp
compare
dataa[21..0]
datab[21..0]=2400000
aeb
sig_clk
clk
CLKIN
CLKOUT
sig_sync
sig_sync_d
sig_check
sigreg[0..0]
sig_level[0..0]
TS
TLGreater
TLLess
sig_level
TS
TRIGOUT
trigger_out
trigger_out_d
TS
TRIGOUT
N_T
clk
reset
trigger
trigger_out
trigger_out_d
delay_counter
clock
delay[15..0]
triggered
armpt_clk
reset
delayOut
delayOut
armpt_clk
DCFIFO
wrreq
wrclk
data[8..0]
wrusedw[]
rdreq
rdclk
q[8..0]
rdusedw[]
data[]
q[]
wrusedw[]
rdclk
rdreq
Blo_full
data[8..0]
wrclk
data[]=480
full
wrfull
Blo_full
clk
CLKIN
CLKOUT
Blo_full
wrfull
Blo_full
fulloff_cmp
compare
dataa[24..0]
datab[]=10000000
aeb
fulloff_counter
counter
clock
q[24..0]
aeb
fulloff_counter
clock
q[24..0]
aeb
Parameter
Value
Type
DELAY_REGISTERS
5
DELAY_WIDTH
16
LPM_NUMWORDS
512
LPM_SHOWAHEAD
OFF
String
LPM_WIDTH
9
OVERFLOW_CHECKING
ON
String
RDSYNC_DELAYPIPE
3
UNDERFLOW_CHECKING
ON
String
USE_EAB
ON
String
WRSYNC_DELAYPIPE
3


trigLevelInc
dataa[6..0]
A
A+B
result[6..0]
2
B
inst26
lpm_mux1
data1x[6..0]
data0x[6..0]
result[6..0]
sel
inst28
TLGreater
unsigned compare
dataa[6..0]
datab[6..0]
agb
inst30
TLGreater
signa[6..0]
trig_level[6..0]
input
VCC
trigLevelDec
dataa[6..0]
A
A-B
result[6..0]
2
B
inst27
lpm_mux1
data1x[6..0]
data0x[6..0]
result[6..0]
sel
inst29
TLLess
unsigned compare
dataa[6..0]
datab[6..0]
alb
inst31
TLLess
TS
input
VCC


button
INPUT
VCC
clock
INPUT
VCC
lpm_counter7
sclr
clock
up counter
q[255..0]
inst
lpm_compare5
unsigned compare
dataa[255..0]
datab[]=24000
ageb
inst1
OUTPUT
dButton

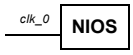

A
INPUT
VCC
CLK
INPUT
VCC
B
INPUT
VCC
DFF
PRN
D
Q
CLRN
inst
inst1
inst2
inst3
XOR
inst10
inst6
inst4
OR2
inst7
AND2
inst8
inst9
inst11
inst13
NOT
inst12
OUTPUT
RIGHT
LEFT

Appendix E - Custom NIOS 2 Datasheet

NIOS

2014.06.19.21:26:38 Datasheet

Overview



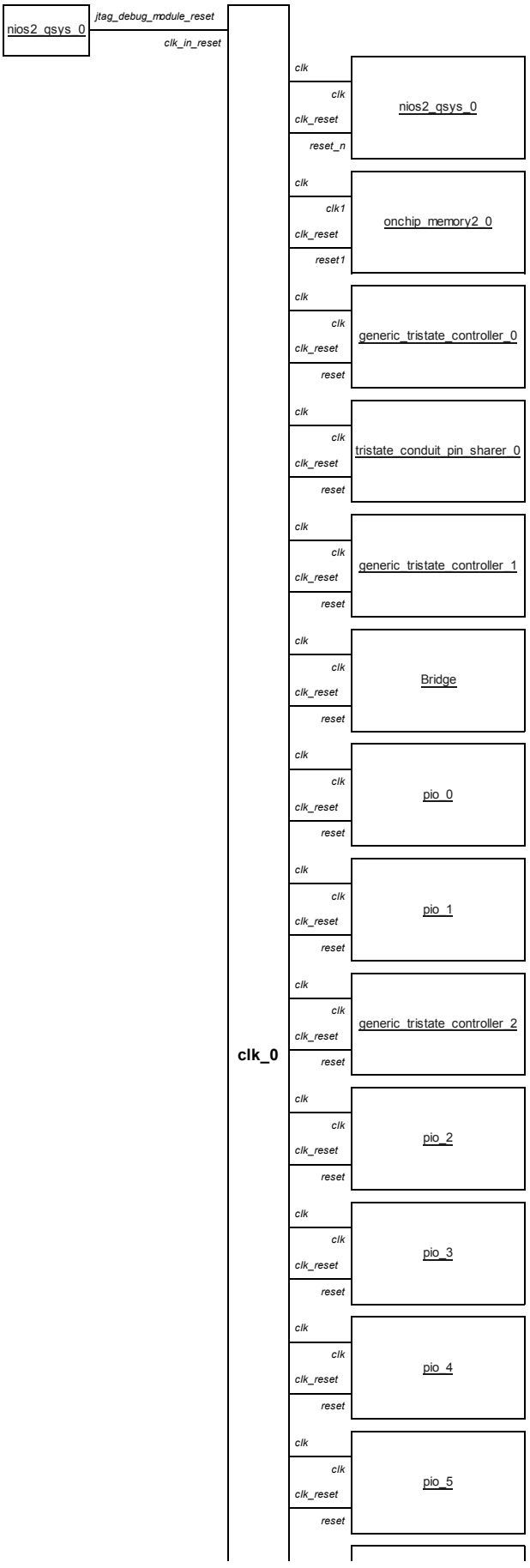
Processor
  nios2_qsys_0 Nios II 13.1
All Components
  nios2_qsys_0 altera_nios2_qsys 13.1
  onchip_memory2_0 altera_avalon_onchip_memory2 13.1
  generic_tristate_controller_0 altera_generic_tristate_controller 13.1
  generic_tristate_controller_1 altera_generic_tristate_controller 13.1
  pio_0 altera_avalon_pio 13.1
  pio_1 altera_avalon_pio 13.1
  generic_tristate_controller_2 altera_generic_tristate_controller 13.1
  pio_2 altera_avalon_pio 13.1
  pio_3 altera_avalon_pio 13.1
  pio_4 altera_avalon_pio 13.1
  pio_5 altera_avalon_pio 13.1
  pio_6 altera_avalon_pio 13.1
  pio_7 altera_avalon_pio 13.1
  pio_8 altera_avalon_pio 13.1
  pio_9 altera_avalon_pio 13.1
  pio_10 altera_avalon_pio 13.1

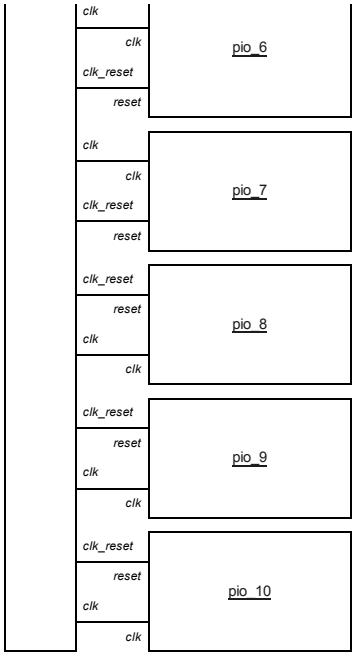
Memory Map

	nios2_qsys_0	
	data_master	*instruction_master*
nios2_qsys_0		
jtag_debug_module	0x000c0800	0x000c0800
onchip_memory2_0		
s1	0x000b0000	0x000b0000
generic_tristate_controller_0		
uas	0x00060000	0x00060000
generic_tristate_controller_1		
uas	0x00040000	0x00040000
pio_0		
s1	0x000c1150	0x000c1150
pio_1		
s1	0x000c1140	0x000c1140
generic_tristate_controller_2		
uas	0x00090000	0x00090000
pio_2		
s1	0x000c1130	0x000c1130
pio_3		
s1	0x000c1120	0x000c1120
pio_4		
s1	0x000c1110	0x000c1110
pio_5		
s1	0x000c1100	0x000c1100
pio_6		
s1	0x000c10f0	0x000c10f0
pio_7		
s1	0x000c10e0	0x000c10e0
pio_8		
s1	0x000c10d0	0x000c10d0
pio_9		
s1	0x000c10c0	0x000c10c0
pio_10		
s1	0x000c10b0	0x000c10b0

clk_0

clock_source v13.1





Parameters

clockFrequency	**50000000**
clockFrequencyKnown	**true**
inputClockFrequency	**0**
resetSynchronousEdges	**NONE**
deviceFamily	**UNKNOWN**
generateLegacySim	**false**

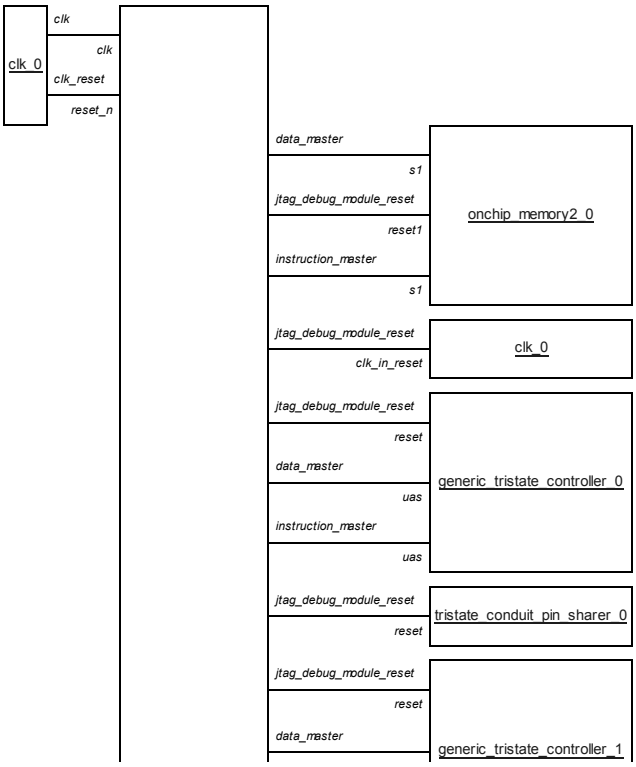
Software Assignments

(none)

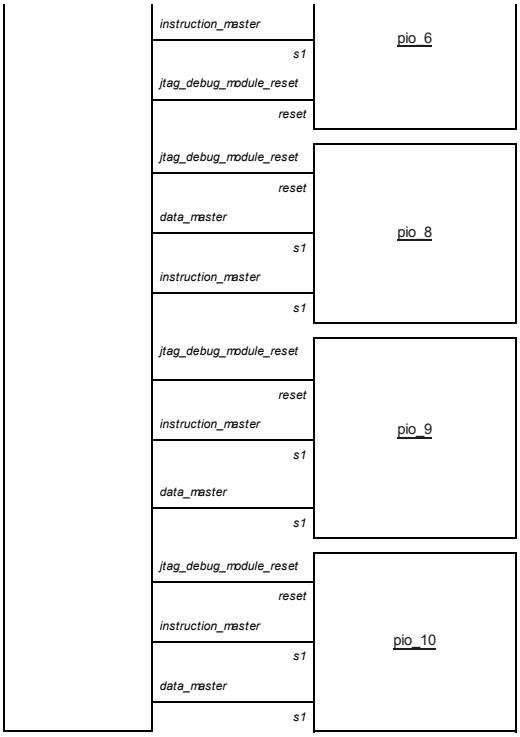
---

nios2_qsys_0

altera_nios2_qsys v13.1



	uas	
	instruction_master	
	uas	
	jtag_debug_module_reset	Bridge
	reset	
	jtag_debug_module_reset	pio_0
	reset	
	data_master	
	s1	
	instruction_master	
	s1	
	d_irq	
	irq	
	jtag_debug_module_reset	pio_1
	reset	
	data_master	
	s1	
	instruction_master	
	s1	
	d_irq	
	irq	
	data_master	generic_tristate_controller_2
	uas	
	instruction_master	
	uas	
	jtag_debug_module_reset	
	reset	
nios2_qsys_0	d_irq	pio_7
	irq	
	data_master	
	s1	
	instruction_master	
	s1	
	jtag_debug_module_reset	
	reset	
	data_master	pio_2
	s1	
	instruction_master	
	s1	
	jtag_debug_module_reset	
	reset	
	data_master	pio_3
	s1	
	instruction_master	
	s1	
	jtag_debug_module_reset	
	reset	
	data_master	pio_4
	s1	
	instruction_master	
	s1	
	jtag_debug_module_reset	
	reset	
	data_master	pio_5
	s1	
	instruction_master	
	s1	
	jtag_debug_module_reset	
	reset	
	data_master	
	s1	



Parameters

setting_showUnpublishedSettings	**false**
setting_showInternalSettings	**false**
setting_preciseSlaveAccessErrorException	**false**
setting_preciseIllegalMemAccessException	**false**
setting_preciseDivisionErrorException	**false**
setting_performanceCounter	**false**
setting_illegalMemAccessDetection	**false**
setting_illegalInstructionsTrap	**true**
setting_fullWaveformSignals	**false**
setting_extraExceptionInfo	**false**
setting_exportPCB	**false**
setting_debugSimGen	**false**
setting_clearXBitsLDNonBypass	**true**
setting_bit31BypassDCache	**true**
setting_bigEndian	**false**
setting_export_large_RAMs	**false**
setting_asic_enabled	**false**
setting_asic_synopsys_translate_on_off	**false**
setting_oci_export_jtag_signals	**false**
setting_bhtIndexPcOnly	**false**
setting_avalonDebugPortPresent	**false**
setting_alwaysEncrypt	**true**
setting_allowFullAddressRange	**false**
setting_activateTrace	**true**
setting_activateTrace_user	**false**
setting_activateTestEndChecker	**false**
setting_ecc_sim_test_ports	**false**
setting_activateMonitors	**true**
setting_activateModelChecker	**false**
setting_HDLSimCachesCleared	**true**
setting_HBreakTest	**false**
setting_breakslaveoveride	**false**

muldiv_divider	**true**
mpu_useLimit	**false**
mpu_enabled	**false**
mmu_enabled	**false**
mmu_autoAssignTlbPtrSz	**true**
manuallyAssignCpuID	**true**
debug_triggerArming	**true**
debug_embeddedPLL	**true**
debug_debugReqSignals	**false**
debug_assignJtagInstanceID	**false**
dcache_omitDataMaster	**false**
cpuReset	**false**
is_hardcopy_compatible	**false**
setting_shadowRegisterSets	**0**
mpu_numOfInstRegion	**8**
mpu_numOfDataRegion	**8**
mmu_TLBMissExcOffset	**0**
debug_jtagInstanceID	**0**
resetOffset	**0**
exceptionOffset	**32**
cpuID	**0**
cpuID_stored	**0**
breakOffset	**32**
userDefinedSettings	
resetSlave	**generic_tristate_controller_1.uas**
mmu_TLBMissExcSlave	**None**
exceptionSlave	**generic_tristate_controller_1.uas**
breakSlave	**nios2_qsys_0.jtag_debug_module**
setting_perfCounterWidth	**32**
setting_interruptControllerType	**Internal**
setting_branchPredictionType	**Automatic**
setting_bhtPtrSz	**8**
muldiv_multiplierType	**EmbeddedMulFast**
mpu_minInstRegionSize	**12**
mpu_minDataRegionSize	**12**
mmu_uitlbNumEntries	**4**
mmu_udtlbNumEntries	**6**
mmu_tlbPtrSz	**7**
mmu_tlbNumWays	**16**
mmu_processIDNumBits	**8**
impl	**Fast**
icache_size	**4096**
icache_tagramBlockType	**Automatic**
icache_ramBlockType	**Automatic**
icache_numTCIM	**0**
icache_burstType	**None**
dcache_bursts	**false**
dcache_victim_buf_impl	**ram**
debug_level	**Level1**
debug_OCIOnchipTrace	**_128**
dcache_size	**0**
dcache_tagramBlockType	**Automatic**
dcache_ramBlockType	**Automatic**
dcache_numTCDM	**0**
dcache_lineSize	**32**
setting_exportvectors	**false**
setting_ecc_present	**false**
setting_ic_ecc_present	**true**
setting_rf_ecc_present	**true**

setting_mmu_ecc_present	**true**
setting_dc_ecc_present	**false**
setting_itcm_ecc_present	**false**
setting_dtcm_ecc_present	**false**
regfile_ramBlockType	**Automatic**
ocimem_ramBlockType	**Automatic**
mmu_ramBlockType	**Automatic**
bht_ramBlockType	**Automatic**
resetAbsoluteAddr	**262144**
exceptionAbsoluteAddr	**262176**
breakAbsoluteAddr	**788512**
mmu_TLBMissExcAbsAddr	**0**
dcache_bursts_derived	**false**
dcache_size_derived	**0**
dcache_lineSize_derived	**32**
translate_on	**"synthesis translate_on"**
translate_off	**"synthesis translate_off"**
instAddrWidth	**20**
dataAddrWidth	**20**
tightlyCoupledDataMaster0AddrWidth	**1**
tightlyCoupledDataMaster1AddrWidth	**1**
tightlyCoupledDataMaster2AddrWidth	**1**
tightlyCoupledDataMaster3AddrWidth	**1**
tightlyCoupledInstructionMaster0AddrWidth	**1**
tightlyCoupledInstructionMaster1AddrWidth	**1**
tightlyCoupledInstructionMaster2AddrWidth	**1**
tightlyCoupledInstructionMaster3AddrWidth	**1**
instSlaveMapParam	**<address-map><slave name='generic_tristate_controller_1.uas' start='0x40000' end='0x60000' /><slave name='generic_tristate_controller_0.uas' start='0x60000' end='0x80000' /><slave name='generic_tristate_controller_2.uas' start='0x90000' end='0xA0000' /><slave name='onchip_memory2_0.s1' start='0xB0000' end='0xB03E8' /><slave name='nios2_qsys_0.jtag_debug_module' start='0xC0800' end='0xC1000' /><slave name='pio_10.s1' start='0xC10B0' end='0xC10C0' /><slave name='pio_9.s1' start='0xC10C0' end='0xC10D0' /><slave name='pio_8.s1' start='0xC10D0' end='0xC10E0' /><slave name='pio_7.s1' start='0xC10E0' end='0xC10F0' /><slave name='pio_6.s1' start='0xC10F0' end='0xC1100' /><slave name='pio_5.s1' start='0xC1100' end='0xC1110' /><slave name='pio_4.s1' start='0xC1110' end='0xC1120' /><slave name='pio_3.s1' start='0xC1120' end='0xC1130' /><slave name='pio_2.s1' start='0xC1130' end='0xC1140' /><slave name='pio_1.s1' start='0xC1140' end='0xC1150' /><slave name='pio_0.s1' start='0xC1150' end='0xC1160' /></address-map>**
dataSlaveMapParam	**<address-map><slave name='generic_tristate_controller_1.uas' start='0x40000' end='0x60000' /><slave name='generic_tristate_controller_0.uas' start='0x60000' end='0x80000' /><slave name='generic_tristate_controller_2.uas' start='0x90000' end='0xA0000' /><slave name='onchip_memory2_0.s1' start='0xB0000' end='0xB03E8' /><slave name='nios2_qsys_0.jtag_debug_module' start='0xC0800' end='0xC1000' /><slave name='pio_10.s1' start='0xC10B0' end='0xC10C0' /><slave name='pio_9.s1' start='0xC10C0' end='0xC10D0' /><slave name='pio_8.s1' start='0xC10D0' end='0xC10E0' /><slave name='pio_7.s1' start='0xC10E0' end='0xC10F0' /><slave name='pio_6.s1' start='0xC10F0' end='0xC1100' /><slave name='pio_5.s1' start='0xC1100' end='0xC1110' /><slave name='pio_4.s1' start='0xC1110' end='0xC1120' /><slave name='pio_3.s1' start='0xC1120' end='0xC1130' /><slave name='pio_2.s1' start='0xC1130' end='0xC1140' /><slave name='pio_1.s1' start='0xC1140' end='0xC1150' /><slave name='pio_0.s1' start='0xC1150' end='0xC1160' /></address-map>**
clockFrequency	**50000000**
deviceFamilyName	**CYCLONEIII**
internalIrqMaskSystemInfo	**7**
customInstSlavesSystemInfo	**<info/>**
	ADDRESS_STALL 1 ADVANCED_INFO 0 ANY_QFP 0 CELL_LEVEL_BACK_ANNOTATION_DISABLED 0 COMPILER_SUPPORT 1 DSP 0 DSP_SHIFTER_BLOCK 0 DUMP_ASM_LAB_BITS_FOR_POWER 1 EMUL 1 ENABLE_ADVANCED_IO_ANALYSIS_GUI_FEATURES 1 ENABLE_PIN_PLANNER 0 ENGINEERING_SAMPLE 0 EPCS 1 ESB 0 FAKE1 0 FAKE2 0 FAKE3 0 FAMILY_LEVEL_INSTALLATION_ONLY 1 FASTEST 0 FINAL_TIMING_MODEL 0 FITTER_USE_FALLING_EDGE_DELAY 1 GENERATE_DC_ON_CURRENT_WARNING_FOR_INTERNAL_CLAMPING_DIODE 0 HARDCOPY 0 HAS_18_BIT_MULTS 0 HAS_ACE_SUPPORT 1 HAS_ACTIVE_PARALLEL_FLASH_SUPPORT 0 HAS_ADJUSTABLE_OUTPUT_IO_TIMING_MEAS_POINT 1 HAS_ADVANCED_IO_INVERTED_CORNER 0 HAS_ADVANCED_IO_POWER_SUPPORT 1 HAS_ADVANCED_IO_TIMING_SUPPORT 1 HAS_ALM_SUPPORT 0 HAS_ATOM_AND_ROUTING_POWER_MODELED_TOGETHER 0 HAS_AUTO_DERIVE_CLOCK_UNCERTAINTY_SUPPORT 0 HAS_AUTO_FIT_SUPPORT 1 HAS_BALANCED_OPT_TECHNIQUE_SUPPORT 1 HAS_BENEFICIAL_SKEW_SUPPORT 1 HAS_BITLEVEL_DRIVE_STRENGTH_CONTROL 1 HAS_BSDL_FILE_GENERATION 0 HAS_CDB_RE_NETWORK_PRESERVATION_SUPPORT 0 HAS_CGA_SUPPORT 1 HAS_CHECK_NETLIST_SUPPORT 0 HAS_CLOCK_REGION_CHECKER_ENABLED 1 HAS_CORE_JUNCTION_TEMP_DERATING 0 HAS_CROSSTALK_SUPPORT 0 HAS_CUSTOM_REGION_SUPPORT 1 HAS_DAP_JTAG_FROM_HPS 0 HAS_DATA_DRIVEN_ACVQ_HSSI_SUPPORT 0 HAS_DDB_FDI_SUPPORT 0

deviceFeaturesSystemInfo	HAS_DESIGN_ANALYZER_SUPPORT 1 HAS_DETAILED_IO_RAIL_POWER_MODEL 1 HAS_DETAILED_LEIM_STATIC_POWER_MODEL 1 HAS_DETAILED_LE_POWER_MODEL 1 HAS_DETAILED_ROUTING_MUX_STATIC_POWER_MODEL 1 HAS_DETAILED_THERMAL_CIRCUIT_PARAMETER_SUPPORT 1 HAS_DEVICE_MIGRATION_SUPPORT 1 HAS_DIAGONAL_MIGRATION_SUPPORT 0 HAS_EMIF_TOOLKIT_SUPPORT 0 HAS_ERROR_DETECTION_SUPPORT 0 HAS_FAMILY_VARIANT_MIGRATION_SUPPORT 0 HAS_FANOUT_FREE_NODE_SUPPORT 1 HAS_FAST_FIT_SUPPORT 1 HAS_FITTER_EARLY_TIMING_ESTIMATE_SUPPORT 1 HAS_FITTER_ECO_SUPPORT 1 HAS_FIT_NETLIST_OPT_RETIME_SUPPORT 1 HAS_FIT_NETLIST_OPT_SUPPORT 1 HAS_FORMAL_VERIFICATION_SUPPORT 1 HAS_FPGA_XCHANGE_SUPPORT 1 HAS_FSAC_LUTRAM_REGISTER_PACKING_SUPPORT 0 HAS_FULL_DAT_MIN_TIMING_SUPPORT 1 HAS_FULL_INCREMENTAL_DESIGN_SUPPORT 1 HAS_FUNCTIONAL_SIMULATION_SUPPORT 1 HAS_FUNCTIONAL_VERILOG_SIMULATION_SUPPORT 0 HAS_FUNCTIONAL_VHDL_SIMULATION_SUPPORT 0 HAS_GLITCH_FILTERING_SUPPORT 1 HAS_HARDCOPYII_SUPPORT 0 HAS_HC_READY_SUPPORT 0 HAS_HIGH_SPEED_LOW_POWER_TILE_SUPPORT 0 HAS_HOLD_TIME_AVOIDANCE_ACROSS_CLOCK_SPINE_SUPPORT 1 HAS_HSPICE_WRITER_SUPPORT 1 HAS_HSSI_POWER_CALCULATOR 0 HAS_IBISO_WRITER_SUPPORT 1 HAS_ICD_DATA_IP 0 HAS_INCREMENTAL_DAT_SUPPORT 1 HAS_INCREMENTAL_SYNTHESIS_SUPPORT 1 HAS_INTERFACE_PLANNER_SUPPORT 0 HAS_IO_ASSIGNMENT_ANALYSIS_SUPPORT 1 HAS_IO_DECODER 0 HAS_IO_PLACEMENT_OPTIMIZATION_SUPPORT 1 HAS_IO_SMART_RECOMPILE_SUPPORT 0 HAS_JITTER_SUPPORT 1 HAS_JTAG_SLD_HUB_SUPPORT 1 HAS_LIMITED_TCL_FITTER_SUPPORT 0 HAS_LOGICAL_FLOORPLANNER_SUPPORT 0 HAS_LOGIC_LOCK_SUPPORT 1 HAS_MICROPROCESSOR 0 HAS_MIF_SMART_COMPILE_SUPPORT 1 HAS_MINMAX_TIMING_MODELING_SUPPORT 1 HAS_MIN_TIMING_ANALYSIS_SUPPORT 1 HAS_MUX_RESTRUCTURE_SUPPORT 1 HAS_NEW_HC_FLOW_SUPPORT 0 HAS_NEW_SERDES_MAX_RESOURCE_COUNT_REPORTING_SUPPORT 1 HAS_NEW_VPR_SUPPORT 1 HAS_NONSOCKET_TECHNOLOGY_MIGRATION_SUPPORT 0 HAS_NO_HARDBLOCK_PARTITION_SUPPORT 0 HAS_NO_JTAG_USERCODE_SUPPORT 0 HAS_OPERATING_SETTINGS_AND_CONDITIONS_REPORTING_SUPPORT 1 HAS_PAD_LOCATION_ASSIGNMENT_SUPPORT 0 HAS_PARTIAL_RECONFIG_SUPPORT 0 HAS_PASSIVE_PARALLEL_SUPPORT 0 HAS_PHYSICAL_NETLIST_OUTPUT 0 HAS_PHYSICAL_ROUTING_SUPPORT 0 HAS_PIN_SPECIFIC_VOLTAGE_SUPPORT 1 HAS_PLDM_REF_SUPPORT 1 HAS_POWER_BINNING_LIMITS_DATA 0 HAS_POWER_ESTIMATION_SUPPORT 1 HAS_PRELIMINARY_CLOCK_UNCERTAINTY_NUMBERS 0 HAS_PRE_FITTER_FPP_SUPPORT 0 HAS_PRE_FITTER_LUTRAM_NETLIST_CHECKER_ENABLED 0 HAS_PVA_SUPPORT 1 HAS_RAPID_RECOMPILE_SUPPORT 0 HAS_RCF_SUPPORT 1 HAS_RCF_SUPPORT_FOR_DEBUGGING 0 HAS_RED_BLACK_SEPARATION_SUPPORT 0 HAS_RE_LEVEL_TIMING_GRAPH_SUPPORT 1 HAS_RISEFALL_DELAY_SUPPORT 1 HAS_SIGNAL_PROBE_SUPPORT 1 HAS_SIGNAL_TAP_SUPPORT 1 HAS_SIMULATOR_SUPPORT 1 HAS_SPLIT_IO_SUPPORT 1 HAS_SPLIT_LC_SUPPORT 1 HAS_STRICT_PRESERVATION_SUPPORT 0 HAS_SYNTH_FSYN_NETLIST_OPT_SUPPORT 1 HAS_SYNTH_NETLIST_OPT_RETIME_SUPPORT 1 HAS_SYNTH_NETLIST_OPT_SUPPORT 1 HAS_TCL_FITTER_SUPPORT 0 HAS_TECHNOLOGY_MIGRATION_SUPPORT 0 HAS_TEMPLATED_REGISTER_PACKING_SUPPORT 1 HAS_TIME_BORROWING_SUPPORT 0 HAS_TIMING_DRIVEN_SYNTHESIS_SUPPORT 1 HAS_TIMING_INFO_SUPPORT 1 HAS_TIMING_OPERATING_CONDITIONS 1 HAS_TIMING_SIMULATION_SUPPORT 1 HAS_TITAN_BASED_MAC_REGISTER_PACKER_SUPPORT 0 HAS_U2B2_SUPPORT 0 HAS_USER_HIGH_SPEED_LOW_POWER_TILE_SUPPORT 0 HAS_USE_FITTER_INFO_SUPPORT 1 HAS_VCCPD_POWER_RAIL 0 HAS_VERTICAL_MIGRATION_SUPPORT 1 HAS_VIEWDRAW_SYMBOL_SUPPORT 0 HAS_VIO_SUPPORT 1 HAS_VIRTUAL_DEVICES 0 HAS_WYSIWYG_DFFEAS_SUPPORT 1 HAS_XIBISO_WRITER_SUPPORT 0 IFP_USE_LEGACY_IO_CHECKER 0 INCREMENTAL_DESIGN_SUPPORTS_COMPATIBLE_CONSTRAINTS 1 INSTALLED 0 INTERNAL_POF_SUPPORT_ENABLED 0 INTERNAL_USE_ONLY 0 ISSUE_MILITARY_TEMPERATURE_WARNING 0 IS_CONFIG_ROM 0 IS_DEFAULT_FAMILY 0 IS_HARDCOPY_FAMILY 0 IS_HBGA_PACKAGE 0 IS_HIGH_CURRENT_PART 0 IS_LOW_POWER_PART 0 IS_SDM_ONLY_PACKAGE 0 LVDS_IO 0 M10K_MEMORY 0 M144K_MEMORY 0 M20K_MEMORY 0 M4K_MEMORY 0 M512_MEMORY 0 M9K_MEMORY 1 MLAB_MEMORY 0 MRAM_MEMORY 0 NOT_LISTED 0 NOT_MIGRATABLE 0 NO_FITTER_DELAY_CACHE_GENERATED 0 NO_PIN_OUT 0 NO_POF 0 NO_RPE_SUPPORT 0 NO_SUPPORT_FOR_LOGICLOCK_CONTENT_BACK_ANNOTATION 1 NO_SUPPORT_FOR_STA_CLOCK_UNCERTAINTY_CHECK 0 NO_TDC_SUPPORT 0 POSTFIT_BAK_DATABASE_EXPORT_ENABLED 1 POSTMAP_BAK_DATABASE_EXPORT_ENABLED 1 PROGRAMMER_SUPPORT 1 QFIT_IN_DEVELOPMENT 0 QMAP_IN_DEVELOPMENT 0 RAM_LOGICAL_NAME_CHECKING_IN_CUT_ENABLED 1 REPORTS_METASTABILITY_MTBF 1 REQUIRES_INSTALLATION_PATCH 0 REQUIRES_LIST_OF_TEMPERATURE_AND_VOLTAGE_OPERATING_CONDITIONS 1 RESERVES_SIGNAL_PROBE_PINS 0 RESOLVE_MAX_FANOUT_EARLY 1 RESOLVE_MAX_FANOUT_LATE 0 RESPECTS_FIXED_SIZED_LOCKED_LOCATION_LOGICLOCK 1 RESTRICTED_USER_SELECTION 0 RISEFALL_SUPPORT_IS_HIDDEN 0 SHOW_HIDDEN_FAMILY_IN_PROGRAMMER 0 STRICT_TIMING_DB_CHECKS 0 SUPPORTS_ADDITIONAL_OPTIONS_FOR_UNUSED_IO 0 SUPPORTS_CRC 1 SUPPORTS_DIFFERENTIAL_AIOT_BOARD_TRACE_MODEL 1 SUPPORTS_DSP_BALANCING_BACK_ANNOTATION 0 SUPPORTS_GENERATION_OF_EARLY_POWER_ESTIMATOR_FILE 1 SUPPORTS_GLOBAL_SIGNAL_BACK_ANNOTATION 0 SUPPORTS_MAC_CHAIN_OUT_ADDER 0 SUPPORTS_RAM_PACKING_BACK_ANNOTATION 0 SUPPORTS_REG_PACKING_BACK_ANNOTATION 0 SUPPORTS_SIGNALPROBE_REGISTER_PIPELINING 1 SUPPORTS_SINGLE_ENDED_AIOT_BOARD_TRACE_MODEL 1 SUPPORTS_USER_MANUAL_LOGIC_DUPLICATION 1 TMV_RUN_CUSTOMIZABLE_VIEWER 1 TMV_RUN_INTERNAL_DETAILS 1 TMV_RUN_INTERNAL_DETAILS_ON_IO 0 TMV_RUN_INTERNAL_DETAILS_ON_IOBUF 1 TMV_RUN_INTERNAL_DETAILS_ON_LCELL 0 TMV_RUN_INTERNAL_DETAILS_ON_LRAM 0 TRANSCEIVER_3G_BLOCK 0 TRANSCEIVER_6G_BLOCK 0 USES_ACV_FOR_FLED 1 USES_ADB_FOR_BACK_ANNOTATION 1 USES_ALTERA_LNSIM 0 USES_ASIC_ROUTING_POWER_CALCULATOR 0 USES_DATA_DRIVEN_PLL_COMPUTATION_UTIL 1 USES_DEV 1 USES_ICP_FOR_ECO_FITTER 0 USES_LIBERTY_TIMING 0 USES_POWER_SIGNAL_ACTIVITIES 1 USES_THIRD_GENERATION_TIMING_MODELS_TIS 1 USES_U2B2_TIMING_MODELS 0 USE_ADVANCED_IO_POWER_BY_DEFAULT 1

Software Assignments

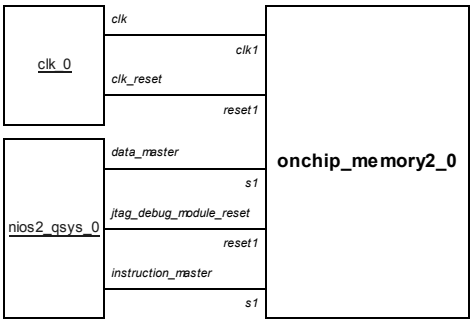
BIG_ENDIAN	**0**
BREAK_ADDR	**0x000c0820**
CPU_FREQ	**50000000u**
CPU_ID_SIZE	**1**
CPU_ID_VALUE	**0x00000000**
CPU_IMPLEMENTATION	**"fast"**
DATA_ADDR_WIDTH	**20**
DCACHE_LINE_SIZE	**0**
DCACHE_LINE_SIZE_LOG2	**0**
DCACHE_SIZE	**0**
EXCEPTION_ADDR	**0x00040020**
FLUSHDA_SUPPORTED	

	USE_ADVANCED_IO_TIMING_BY_DEFAULT 1 USE_BASE_FAMILY_DDB_PATH 0 USE_OCT_AUTO_CALIBRATION 0 USE_RELAX_IO_ASSIGNMENT_RULES 1 USE_RISEFALL_ONLY 1 USE_SEPARATE_LIST_FOR_TECH_MIGRATION 0 USE_SINGLE_COMPILER_PASS_PLL_MIF_FILE_WRITER 1 USE_TITAN_IO_BASED_IO_REGISTER_PACKER_UTIL 0 USING_28NM_OR_OLDER_TIMING_METHODOLOGY 1 WYSIWYG_BUS_WIDTH_CHECKING_IN_CUT_ENABLED 1
tightlyCoupledDataMaster0MapParam	
tightlyCoupledDataMaster1MapParam	
tightlyCoupledDataMaster2MapParam	
tightlyCoupledDataMaster3MapParam	
tightlyCoupledInstructionMaster0MapParam	
tightlyCoupledInstructionMaster1MapParam	
tightlyCoupledInstructionMaster2MapParam	
tightlyCoupledInstructionMaster3MapParam	
deviceFamily	**UNKNOWN**
generateLegacySim	**false**

FLUSHDA_SUPPORTED	
HARDWARE_DIVIDE_PRESENT	**1**
HARDWARE_MULTIPLY_PRESENT	**1**
HARDWARE_MULX_PRESENT	**0**
HAS_DEBUG_CORE	**1**
HAS_DEBUG_STUB	
HAS_ILLEGAL_INSTRUCTION_EXCEPTION	
HAS_JMPI_INSTRUCTION	
ICACHE_LINE_SIZE	**32**
ICACHE_LINE_SIZE_LOG2	**5**
ICACHE_SIZE	**4096**
INST_ADDR_WIDTH	**20**
NUM_OF_SHADOW_REG_SETS	**0**
RESET_ADDR	**0x00040000**

onchip_memory2_0

altera_avalon_onchip_memory2 v13.1



Parameters

allowInSystemMemoryContentEditor	**false**
blockType	**AUTO**
dataWidth	**8**
dualPort	**false**
initMemContent	**true**
initializationFileName	**onchip_mem.hex**
instanceID	**NONE**
memorySize	**1000**
readDuringWriteMode	**DONT_CARE**
simAllowMRAMContentsFile	**false**
simMemInitOnlyFilename	**0**
singleClockOperation	**false**
slave1Latency	**1**
slave2Latency	**1**
useNonDefaultInitFile	**false**
useShallowMemBlocks	**false**
writable	**true**
ecc_enabled	**false**
autoInitializationFileName	**NIOS_onchip_memory2_0**
deviceFamily	**CYCLONEIII**
	ADDRESS_STALL 1 ADVANCED_INFO 0 ANY_QFP 0 CELL_LEVEL_BACK_ANNOTATION_DISABLED 0 COMPILER_SUPPORT 1 DSP 0 DSP_SHIFTER_BLOCK 0 DUMP_ASM_LAB_BITS_FOR_POWER 1 EMUL 1 ENABLE_ADVANCED_IO_ANALYSIS_GUI_FEATURES 1 ENABLE_PIN_PLANNER 0 ENGINEERING_SAMPLE 0 EPCS 1 ESB 0 FAKE1 0 FAKE2 0 FAKE3 0 FAMILY_LEVEL_INSTALLATION_ONLY 1 FASTEST 0 FINAL_TIMING_MODEL 0 FITTER_USE_FALLING_EDGE_DELAY 1 GENERATE_DC_ON_CURRENT_WARNING_FOR_INTERNAL_CLAMPING_DIODE 0 HARDCOPY 0 HAS_18_BIT_MULTS 0 HAS_ACE_SUPPORT 1 HAS_ACTIVE_PARALLEL_FLASH_SUPPORT 0

deviceFeatures	HAS_ADJUSTABLE_OUTPUT_IO_TIMING_MEAS_POINT 1 HAS_ADVANCED_IO_INVERTED_CORNER 0 HAS_ADVANCED_IO_POWER_SUPPORT 1 HAS_ADVANCED_IO_TIMING_SUPPORT 1 HAS_ALM_SUPPORT 0 HAS_ATOM_AND_ROUTING_POWER_MODELED_TOGETHER 0 HAS_AUTO_DERIVE_CLOCK_UNCERTAINTY_SUPPORT 0 HAS_AUTO_FIT_SUPPORT 1 HAS_BALANCED_OPT_TECHNIQUE_SUPPORT 1 HAS_BENEFICIAL_SKEW_SUPPORT 1 HAS_BITLEVEL_DRIVE_STRENGTH_CONTROL 1 HAS_BSDL_FILE_GENERATION 0 HAS_CDB_RE_NETWORK_PRESERVATION_SUPPORT 0 HAS_CGA_SUPPORT 1 HAS_CHECK_NETLIST_SUPPORT 0 HAS_CLOCK_REGION_CHECKER_ENABLED 1 HAS_CORE_JUNCTION_TEMP_DERATING 0 HAS_CROSSTALK_SUPPORT 0 HAS_CUSTOM_REGION_SUPPORT 1 HAS_DAP_JTAG_FROM_HPS 0 HAS_DATA_DRIVEN_ACVQ_HSSI_SUPPORT 0 HAS_DDB_FDI_SUPPORT 0 HAS_DESIGN_ANALYZER_SUPPORT 1 HAS_DETAILED_IO_RAIL_POWER_MODEL 1 HAS_DETAILED_LEIM_STATIC_POWER_MODEL 1 HAS_DETAILED_LE_POWER_MODEL 1 HAS_DETAILED_ROUTING_MUX_STATIC_POWER_MODEL 1 HAS_DETAILED_THERMAL_CIRCUIT_PARAMETER_SUPPORT 1 HAS_DEVICE_MIGRATION_SUPPORT 1 HAS_DIAGONAL_MIGRATION_SUPPORT 0 HAS_EMIF_TOOLKIT_SUPPORT 0 HAS_ERROR_DETECTION_SUPPORT 0 HAS_FAMILY_VARIANT_MIGRATION_SUPPORT 0 HAS_FANOUT_FREE_NODE_SUPPORT 1 HAS_FAST_FIT_SUPPORT 1 HAS_FITTER_EARLY_TIMING_ESTIMATE_SUPPORT 1 HAS_FITTER_ECO_SUPPORT 1 HAS_FIT_NETLIST_OPT_RETIME_SUPPORT 1 HAS_FIT_NETLIST_OPT_SUPPORT 1 HAS_FORMAL_VERIFICATION_SUPPORT 1 HAS_FPGA_XCHANGE_SUPPORT 1 HAS_FSAC_LUTRAM_REGISTER_PACKING_SUPPORT 0 HAS_FULL_DAT_MIN_TIMING_SUPPORT 1 HAS_FULL_INCREMENTAL_DESIGN_SUPPORT 1 HAS_FUNCTIONAL_SIMULATION_SUPPORT 1 HAS_FUNCTIONAL_VERILOG_SIMULATION_SUPPORT 0 HAS_FUNCTIONAL_VHDL_SIMULATION_SUPPORT 0 HAS_GLITCH_FILTERING_SUPPORT 1 HAS_HARDCOPYII_SUPPORT 0 HAS_HC_READY_SUPPORT 0 HAS_HIGH_SPEED_LOW_POWER_TILE_SUPPORT 0 HAS_HOLD_TIME_AVOIDANCE_ACROSS_CLOCK_SPINE_SUPPORT 1 HAS_HSPICE_WRITER_SUPPORT 1 HAS_HSSI_POWER_CALCULATOR 0 HAS_IBISO_WRITER_SUPPORT 1 HAS_ICD_DATA_IP 0 HAS_INCREMENTAL_DAT_SUPPORT 1 HAS_INCREMENTAL_SYNTHESIS_SUPPORT 1 HAS_INTERFACE_PLANNER_SUPPORT 0 HAS_IO_ASSIGNMENT_ANALYSIS_SUPPORT 1 HAS_IO_DECODER 0 HAS_IO_PLACEMENT_OPTIMIZATION_SUPPORT 1 HAS_IO_SMART_RECOMPILE_SUPPORT 0 HAS_JITTER_SUPPORT 1 HAS_JTAG_SLD_HUB_SUPPORT 1 HAS_LIMITED_TCL_FITTER_SUPPORT 0 HAS_LOGICAL_FLOORPLANNER_SUPPORT 0 HAS_LOGIC_LOCK_SUPPORT 1 HAS_MICROPROCESSOR 0 HAS_MIF_SMART_COMPILE_SUPPORT 1 HAS_MINMAX_TIMING_MODELING_SUPPORT 1 HAS_MIN_TIMING_ANALYSIS_SUPPORT 1 HAS_MUX_RESTRUCTURE_SUPPORT 1 HAS_NEW_HC_FLOW_SUPPORT 0 HAS_NEW_SERDES_MAX_RESOURCE_COUNT_REPORTING_SUPPORT 1 HAS_NEW_VPR_SUPPORT 1 HAS_NONSOCKET_TECHNOLOGY_MIGRATION_SUPPORT 0 HAS_NO_HARDBLOCK_PARTITION_SUPPORT 0 HAS_NO_JTAG_USERCODE_SUPPORT 0 HAS_OPERATING_SETTINGS_AND_CONDITIONS_REPORTING_SUPPORT 1 HAS_PAD_LOCATION_ASSIGNMENT_SUPPORT 0 HAS_PARTIAL_RECONFIG_SUPPORT 0 HAS_PASSIVE_PARALLEL_SUPPORT 0 HAS_PHYSICAL_NETLIST_OUTPUT 0 HAS_PHYSICAL_ROUTING_SUPPORT 0 HAS_PIN_SPECIFIC_VOLTAGE_SUPPORT 1 HAS_PLDM_REF_SUPPORT 1 HAS_POWER_BINNING_LIMITS_DATA 0 HAS_POWER_ESTIMATION_SUPPORT 1 HAS_PRELIMINARY_CLOCK_UNCERTAINTY_NUMBERS 0 HAS_PRE_FITTER_FPP_SUPPORT 0 HAS_PRE_FITTER_LUTRAM_NETLIST_CHECKER_ENABLED 0 HAS_PVA_SUPPORT 1 HAS_RAPID_RECOMPILE_SUPPORT 0 HAS_RCF_SUPPORT 1 HAS_RCF_SUPPORT_FOR_DEBUGGING 0 HAS_RED_BLACK_SEPARATION_SUPPORT 0 HAS_RE_LEVEL_TIMING_GRAPH_SUPPORT 1 HAS_RISEFALL_DELAY_SUPPORT 1 HAS_SIGNAL_PROBE_SUPPORT 1 HAS_SIGNAL_TAP_SUPPORT 1 HAS_SIMULATOR_SUPPORT 1 HAS_SPLIT_IO_SUPPORT 1 HAS_SPLIT_LC_SUPPORT 1 HAS_STRICT_PRESERVATION_SUPPORT 0 HAS_SYNTH_FSYN_NETLIST_OPT_SUPPORT 1 HAS_SYNTH_NETLIST_OPT_RETIME_SUPPORT 1 HAS_SYNTH_NETLIST_OPT_SUPPORT 1 HAS_TCL_FITTER_SUPPORT 0 HAS_TECHNOLOGY_MIGRATION_SUPPORT 0 HAS_TEMPLATED_REGISTER_PACKING_SUPPORT 1 HAS_TIME_BORROWING_SUPPORT 0 HAS_TIMING_DRIVEN_SYNTHESIS_SUPPORT 1 HAS_TIMING_INFO_SUPPORT 1 HAS_TIMING_OPERATING_CONDITIONS 1 HAS_TIMING_SIMULATION_SUPPORT 1 HAS_TITAN_BASED_MAC_REGISTER_PACKER_SUPPORT 0 HAS_U2B2_SUPPORT 0 HAS_USER_HIGH_SPEED_LOW_POWER_TILE_SUPPORT 0 HAS_USE_FITTER_INFO_SUPPORT 1 HAS_VCCPD_POWER_RAIL 0 HAS_VERTICAL_MIGRATION_SUPPORT 1 HAS_VIEWDRAW_SYMBOL_SUPPORT 0 HAS_VIO_SUPPORT 1 HAS_VIRTUAL_DEVICES 0 HAS_WYSIWYG_DFFEAS_SUPPORT 1 HAS_XIBISO_WRITER_SUPPORT 0 IFP_USE_LEGACY_IO_CHECKER 0 INCREMENTAL_DESIGN_SUPPORTS_COMPATIBLE_CONSTRAINTS 1 INSTALLED 0 INTERNAL_POF_SUPPORT_ENABLED 0 INTERNAL_USE_ONLY 0 ISSUE_MILITARY_TEMPERATURE_WARNING 0 IS_CONFIG_ROM 0 IS_DEFAULT_FAMILY 0 IS_HARDCOPY_FAMILY 0 IS_HBGA_PACKAGE 0 IS_HIGH_CURRENT_PART 0 IS_LOW_POWER_PART 0 IS_SDM_ONLY_PACKAGE 0 LVDS_IO 0 M10K_MEMORY 0 M144K_MEMORY 0 M20K_MEMORY 0 M4K_MEMORY 0 M512_MEMORY 0 M9K_MEMORY 1 MLAB_MEMORY 0 MRAM_MEMORY 0 NOT_LISTED 0 NOT_MIGRATABLE 0 NO_FITTER_DELAY_CACHE_GENERATED 0 NO_PIN_OUT 0 NO_POF 0 NO_RPE_SUPPORT 0 NO_SUPPORT_FOR_LOGICLOCK_CONTENT_BACK_ANNOTATION 1 NO_SUPPORT_FOR_STA_CLOCK_UNCERTAINTY_CHECK 0 NO_TDC_SUPPORT 0 POSTFIT_BAK_DATABASE_EXPORT_ENABLED 1 POSTMAP_BAK_DATABASE_EXPORT_ENABLED 1 PROGRAMMER_SUPPORT 1 QFIT_IN_DEVELOPMENT 0 QMAP_IN_DEVELOPMENT 0 RAM_LOGICAL_NAME_CHECKING_IN_CUT_ENABLED 1 REPORTS_METASTABILITY_MTBF 1 REQUIRES_INSTALLATION_PATCH 0 REQUIRES_LIST_OF_TEMPERATURE_AND_VOLTAGE_OPERATING_CONDITIONS 1 RESERVES_SIGNAL_PROBE_PINS 0 RESOLVE_MAX_FANOUT_EARLY 1 RESOLVE_MAX_FANOUT_LATE 0 RESPECTS_FIXED_SIZED_LOCKED_LOCATION_LOGICLOCK 1 RESTRICTED_USER_SELECTION 0 RISEFALL_SUPPORT_IS_HIDDEN 0 SHOW_HIDDEN_FAMILY_IN_PROGRAMMER 0 STRICT_TIMING_DB_CHECKS 0 SUPPORTS_ADDITIONAL_OPTIONS_FOR_UNUSED_IO 0 SUPPORTS_CRC 1 SUPPORTS_DIFFERENTIAL_AIOT_BOARD_TRACE_MODEL 1 SUPPORTS_DSP_BALANCING_BACK_ANNOTATION 0 SUPPORTS_GENERATION_OF_EARLY_POWER_ESTIMATOR_FILE 1 SUPPORTS_GLOBAL_SIGNAL_BACK_ANNOTATION 0 SUPPORTS_MAC_CHAIN_OUT_ADDER 0 SUPPORTS_RAM_PACKING_BACK_ANNOTATION 0 SUPPORTS_REG_PACKING_BACK_ANNOTATION 0 SUPPORTS_SIGNALPROBE_REGISTER_PIPELINING 1 SUPPORTS_SINGLE_ENDED_AIOT_BOARD_TRACE_MODEL 1

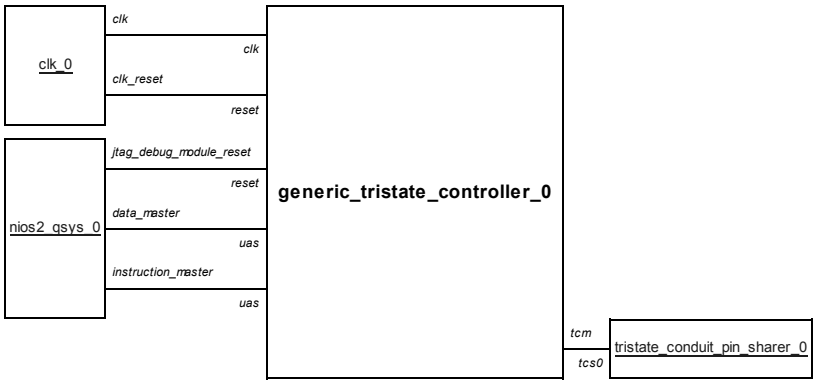
	SUPPORTS_SINGLE_ENDED_AIOT_BOARD_TRACE_MODEL 1 SUPPORTS_USER_MANUAL_LOGIC_DUPLICATION 1 TMV_RUN_CUSTOMIZABLE_VIEWER 1 TMV_RUN_INTERNAL_DETAILS 1 TMV_RUN_INTERNAL_DETAILS_ON_IO 0 TMV_RUN_INTERNAL_DETAILS_ON_IOBUF 1 TMV_RUN_INTERNAL_DETAILS_ON_LCELL 0 TMV_RUN_INTERNAL_DETAILS_ON_LRAM 0 TRANSCEIVER_3G_BLOCK 0 TRANSCEIVER_6G_BLOCK 0 USES_ACV_FOR_FLED 1 USES_ADB_FOR_BACK_ANNOTATION 1 USES_ALTERA_LNSIM 0 USES_ASIC_ROUTING_POWER_CALCULATOR 0 USES_DATA_DRIVEN_PLL_COMPUTATION_UTIL 1 USES_DEV 1 USES_ICP_FOR_ECO_FITTER 0 USES_LIBERTY_TIMING 0 USES_POWER_SIGNAL_ACTIVITIES 1 USES_THIRD_GENERATION_TIMING_MODELS_TIS 1 USES_U2B2_TIMING_MODELS 0 USE_ADVANCED_IO_POWER_BY_DEFAULT 1 USE_ADVANCED_IO_TIMING_BY_DEFAULT 1 USE_BASE_FAMILY_DDB_PATH 0 USE_OCT_AUTO_CALIBRATION 0 USE_RELAX_IO_ASSIGNMENT_RULES 1 USE_RISEFALL_ONLY 1 USE_SEPARATE_LIST_FOR_TECH_MIGRATION 0 USE_SINGLE_COMPILER_PASS_PLL_MIF_FILE_WRITER 1 USE_TITAN_IO_BASED_IO_REGISTER_PACKER_UTIL 0 USING_28NM_OR_OLDER_TIMING_METHODOLOGY 1 WYSIWYG_BUS_WIDTH_CHECKING_IN_CUT_ENABLED 1
derived_set_addr_width	**10**
derived_set_data_width	**8**
derived_gui_ram_block_type	**Automatic**
derived_is_hardcopy	**false**
derived_init_file_name	**NIOS_onchip_memory2_0.hex**
generateLegacySim	**false**

Software Assignments

ALLOW_IN_SYSTEM_MEMORY_CONTENT_EDITOR	**0**
ALLOW_MRAM_SIM_CONTENTS_ONLY_FILE	**0**
CONTENTS_INFO	**""**
DUAL_PORT	**0**
GUI_RAM_BLOCK_TYPE	**AUTO**
INIT_CONTENTS_FILE	**NIOS_onchip_memory2_0**
INIT_MEM_CONTENT	**1**
INSTANCE_ID	**NONE**
NON_DEFAULT_INIT_FILE_ENABLED	**0**
RAM_BLOCK_TYPE	**AUTO**
READ_DURING_WRITE_MODE	**DONT_CARE**
SINGLE_CLOCK_OP	**0**
SIZE_MULTIPLE	**1**
SIZE_VALUE	**1000**
WRITABLE	**1**

generic_tristate_controller_0

altera_generic_tristate_controller v13.1



Parameters

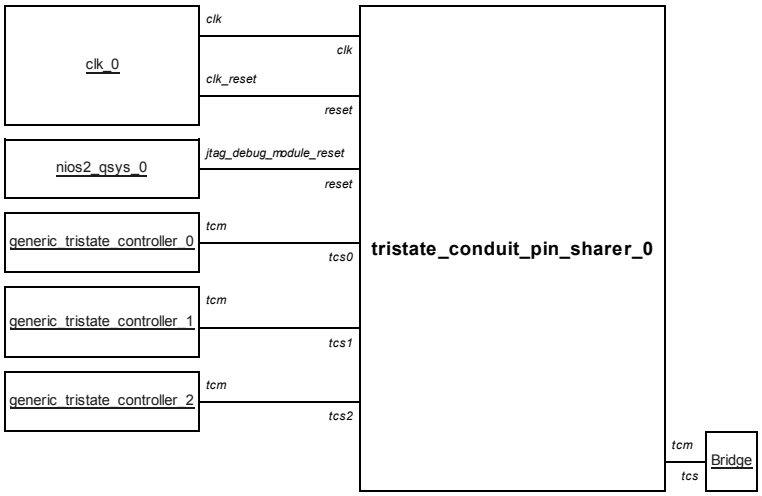
TCM_ADDRESS_W	**17**
TCM_DATA_W	**8**
TCM_BYTEENABLE_W	**1**
TCM_READ_WAIT	**4**
TCM_WRITE_WAIT	**3**
TCM_SETUP_WAIT	**0**
TCM_DATA_HOLD	**0**
TCM_MAX_PENDING_READ_TRANSACTIONS	**1**
TCM_TURNAROUND_TIME	**2**
TCM_TIMING_UNITS	**1**
TCM_READLATENCY	**2**
TCM_SYMBOLS_PER_WORD	**1**
USE_READDATA	**1**
USE_WRITEDATA	**1**
USE_READ	**0**
USE_WRITE	**1**
USE_BEGINTRANSFER	**0**
USE_BYTEENABLE	**0**
USE_CHIPSELECT	**1**
USE_LOCK	**0**
USE_ADDRESS	**1**
USE_WAITREQUEST	**0**
USE_WRITEBYTEENABLE	**0**
USE_OUTPUTENABLE	**0**
USE_RESETREQUEST	**0**
USE_IRQ	**0**
USE_RESET_OUTPUT	**0**
ACTIVE_LOW_READ	**0**
ACTIVE_LOW_LOCK	**0**
ACTIVE_LOW_WRITE	**1**
ACTIVE_LOW_CHIPSELECT	**1**
ACTIVE_LOW_BYTEENABLE	**0**
ACTIVE_LOW_OUTPUTENABLE	**0**
ACTIVE_LOW_WRITEBYTEENABLE	**0**
ACTIVE_LOW_WAITREQUEST	**0**
ACTIVE_LOW_BEGINTRANSFER	**0**
ACTIVE_LOW_RESETREQUEST	**0**
ACTIVE_LOW_IRQ	**0**
ACTIVE_LOW_RESET_OUTPUT	**0**
CHIPSELECT_THROUGH_READLATENCY	**0**
IS_MEMORY_DEVICE	**1**
MODULE_ASSIGNMENT_KEYS	**embeddedsw.configuration.hwClassnameDriverSupportList**
MODULE_ASSIGNMENT_VALUES	**altera_avalon_lan91c111\,altera_avalon_cfi_flash**
INTERFACE_ASSIGNMENT_KEYS	
INTERFACE_ASSIGNMENT_VALUES	
CLOCK_RATE	**50000000**
AUTO_CLK_CLOCK_DOMAIN	**1**
AUTO_CLK_RESET_DOMAIN	**1**
AUTO_TRISTATECONDUIT_MASTERS	
AUTO_DEVICE_FAMILY	**CYCLONEIII**
AUTO_DEVICE	**EP3C25Q240C8**
deviceFamily	**Cyclone III**
generateLegacySim	**false**

Software Assignments

(none)

tristate_conduit_pin_sharer_0

altera_tristate_conduit_pin_sharer v13.1



Parameters

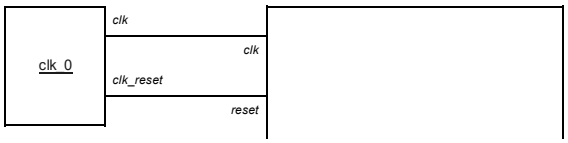
INTERFACE_INFO	**<info><slave name="tcs0"><master name="generic_tristate_controller_0.tcm"><pin role="" width="1" type="Invalid" output_name="" output_enable_name="" input_name="" /><pin r width="8" type="Bidirectional" output_name="tcm_data_out" output_enable_name="tcm_data_outen" input_name="tcm_data_in" /><pin role="chipselect_n" width="1" type="Output" input_name="" /><pin role="address" width="17" type="Output" output_name="tcm_address_out" output_enable_name="" input_name="" /><pin role="write_n" width="1" type="Outp width="1" type="Output" output_name="tcm_chipselect_n_out" output_enable_name="" input_name="" /></master></slave><slave name="tcs2"><master name="generic_tristate_co role="address" width="16" type="Output" output_name="tcm_address_out" output_enable_name="" input_name="" /><pin role="write_n" width="1" type="Output" output_name="tcm output_name="tcm_chipselect_n_out" output_enable_name="" input_name="" /></master></slave></info>**
NUM_INTERFACES	**3**
MODULE_ORIGIN_LIST	**generic_tristate_controller_2.tcm,generic_tristate_controller_2.tcm,generic_tristate_controller_2.tcm,generic_tristate_controller_2.tcm,generic_tristate_controller_2.tcm,generic_trista**
SIGNAL_ORIGIN_LIST	**address,waitrequest,write_n,data,chipselect_n,address,write_n,data,chipselect_n,address,write_n,data,chipselect_n**
SIGNAL_ORIGIN_TYPE	**Output,Input,Output,Bidirectional,Output,Output,Output,Bidirectional,Output,Output,Output,Bidirectional,Output**
SIGNAL_ORIGIN_WIDTH	**16,1,1,8,1,17,1,8,1,17,1,8,1**
SHARED_SIGNAL_LIST	**VRAMAddress,VRAMBusy,VRAMRW,Data,VRAMCS,Address,ROMRW,Data,ROMCS,Address,RAMRW,Data,RAMCS**
SIGNAL_OUTPUT_NAMES	**tcm_address_out,,tcm_write_n_out,tcm_data_out,tcm_chipselect_n_out,tcm_address_out,tcm_write_n_out,tcm_data_out,tcm_chipselect_n_out,tcm_address_out,tcm_write_n_out,tcr**
SIGNAL_INPUT_NAMES	**,tcm_waitrequest_in,,tcm_data_in,,,,tcm_data_in,,,,tcm_data_in**
SIGNAL_OUTPUT_ENABLE_NAMES	**,,,tcm_data_outen,,,,tcm_data_outen,,,,tcm_data_outen**
REALTIME_MODULE_ORIGIN_LIST	**generic_tristate_controller_2.tcm,generic_tristate_controller_2.tcm,generic_tristate_controller_2.tcm,generic_tristate_controller_2.tcm,generic_tristate_controller_2.tcm,generic_trista**
REALTIME_SIGNAL_ORIGIN_LIST	**address,waitrequest,write_n,data,chipselect_n,address,write_n,data,chipselect_n,address,write_n,data,chipselect_n**
REALTIME_SHARED_SIGNAL_LIST	**VRAMAddress,VRAMBusy,VRAMRW,Data,VRAMCS,Address,ROMRW,Data,ROMCS,Address,RAMRW,Data,RAMCS**
AUTO_CLK_CLOCK_RATE	**50000000**
AUTO_CLK_CLOCK_DOMAIN	**1**
AUTO_CLK_RESET_DOMAIN	**1**
AUTO_TRISTATECONDUIT_MASTERS	**<info><slave name="tcs0"><master name="generic_tristate_controller_0.tcm"><port role="write_n_out" direction="output" width="1" /><port role="chipselect_n_out" direction="outp direction="output" width="1" /><port role="data_in" direction="input" width="8" /></master></slave><slave name="tcs1"><master name="generic_tristate_controller_1.tcm"><port ro width="17" /><port role="data_out" direction="output" width="8" /><port role="data_outen" direction="output" width="1" /><port role="data_in" direction="input" width="8" /></master width="1" /><port role="request" direction="output" width="1" /><port role="grant" direction="input" width="1" /><port role="address_out" direction="output" width="16" /><port role=**
AUTO_DEVICE_FAMILY	**CYCLONEIII**
AUTO_DEVICE	**EP3C25Q240C8**
deviceFamily	**Cyclone III**
generateLegacySim	**false**

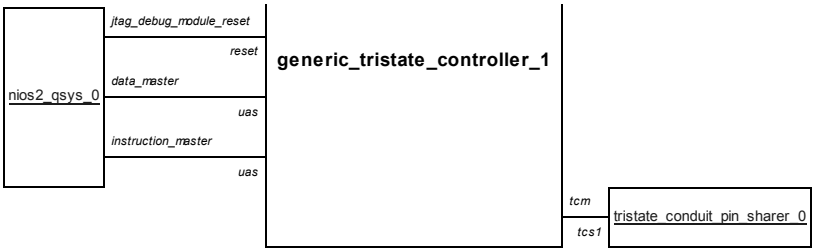
Software Assignments

(none)

generic_tristate_controller_1

altera_generic_tristate_controller v13.1




nios2_qsys_0
jtag_debug_module_reset
reset
data_master
uas
instruction_master
uas
generic_tristate_controller_1
tcm
tcs1
tristate_conduit_pin_sharer_0

Parameters

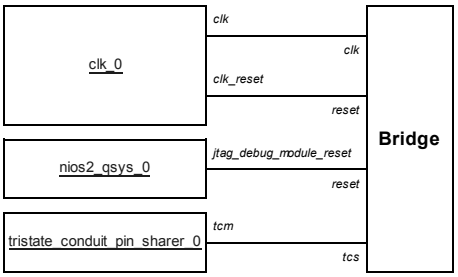
TCM_ADDRESS_W	**17**
TCM_DATA_W	**8**
TCM_BYTEENABLE_W	**1**
TCM_READ_WAIT	**4**
TCM_WRITE_WAIT	**0**
TCM_SETUP_WAIT	**0**
TCM_DATA_HOLD	**0**
TCM_MAX_PENDING_READ_TRANSACTIONS	**1**
TCM_TURNAROUND_TIME	**2**
TCM_TIMING_UNITS	**1**
TCM_READLATENCY	**2**
TCM_SYMBOLS_PER_WORD	**1**
USE_READDATA	**1**
USE_WRITEDATA	**1**
USE_READ	**0**
USE_WRITE	**1**
USE_BEGINTRANSFER	**0**
USE_BYTEENABLE	**0**
USE_CHIPSELECT	**1**
USE_LOCK	**0**
USE_ADDRESS	**1**
USE_WAITREQUEST	**0**
USE_WRITEBYTEENABLE	**0**
USE_OUTPUTENABLE	**0**
USE_RESETREQUEST	**0**
USE_IRQ	**0**
USE_RESET_OUTPUT	**0**
ACTIVE_LOW_READ	**0**
ACTIVE_LOW_LOCK	**0**
ACTIVE_LOW_WRITE	**1**
ACTIVE_LOW_CHIPSELECT	**1**
ACTIVE_LOW_BYTEENABLE	**0**
ACTIVE_LOW_OUTPUTENABLE	**0**
ACTIVE_LOW_WRITEBYTEENABLE	**0**
ACTIVE_LOW_WAITREQUEST	**0**
ACTIVE_LOW_BEGINTRANSFER	**0**
ACTIVE_LOW_RESETREQUEST	**0**
ACTIVE_LOW_IRQ	**0**
ACTIVE_LOW_RESET_OUTPUT	**0**
CHIPSELECT_THROUGH_READLATENCY	**0**
IS_MEMORY_DEVICE	**1**
MODULE_ASSIGNMENT_KEYS	**embeddedsw.configuration.hwClassnameDriverSupportList**
MODULE_ASSIGNMENT_VALUES	**altera_avalon_lan91c111\,altera_avalon_cfi_flash**
INTERFACE_ASSIGNMENT_KEYS	
INTERFACE_ASSIGNMENT_VALUES	
CLOCK_RATE	**50000000**
AUTO_CLK_CLOCK_DOMAIN	**1**
AUTO_CLK_RESET_DOMAIN	**1**
AUTO_TRISTATECONDUIT_MASTERS	
AUTO_DEVICE_FAMILY	**CYCLONEIII**
AUTO_DEVICE	**EP3C25Q240C8**
deviceFamily	**Cyclone III**
generateLegacySim	**false**

Software Assignments

(none)

Bridge

altera_tristate_conduit_bridge v13.1



Parameters

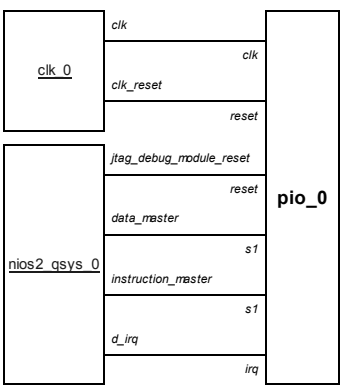
INTERFACE_INFO	**<info><slave name="tcs"><master name="tristate_conduit_pin_sharer_0.tcm"> <pin role="" width="1" type="Invalid" output_name="" output_enable_name="" input_name="" /><pin role="Data" width="8" type="Bidirectional" output_name="Data" output_enable_name="Data_outen" input_name="Data_in" /><pin role="RAMCS" width="1" type="Output" output_name="RAMCS" output_enable_name="" input_name="" /> <pin role="VRAMAddress" width="16" type="Output" output_name="VRAMAddress" output_enable_name="" input_name="" /> <pin role="VRAMBusy" width="1" type="Input" output_name="" output_enable_name="" input_name="VRAMBusy" /><pin role="VRAMCS" width="1" type="Output" output_name="VRAMCS" output_enable_name="" input_name="" /> <pin role="Address" width="17" type="Output" output_name="Address" output_enable_name="" input_name="" /> <pin role="ROMRW" width="1" type="Output" output_name="ROMRW" output_enable_name="" input_name="" /> <pin role="RAMRW" width="1" type="Output" output_name="RAMRW" output_enable_name="" input_name="" /> <pin role="VRAMRW" width="1" type="Output" output_name="VRAMRW" output_enable_name="" input_name="" /> <pin role="ROMCS" width="1" type="Output" output_name="ROMCS" output_enable_name="" input_name="" /> </master></slave></info>**
AUTO_CLK_CLOCK_RATE	**50000000**
AUTO_DEVICE_FAMILY	**CYCLONEIII**
deviceFamily	**Cyclone III**
generateLegacySim	**false**

Software Assignments

(none)

pio_0

altera_avalon_pio v13.1



Parameters

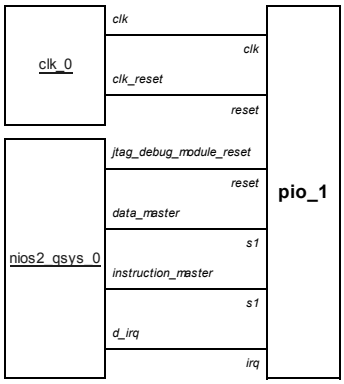
bitClearingEdgeCapReg	**false**
bitModifyingOutReg	**false**
captureEdge	**true**
direction	**Input**
edgeType	**RISING**
generateIRQ	**true**
irqType	**EDGE**
resetValue	**0**
simDoTestBenchWiring	**false**
simDrivenValue	**0**
width	**2**
clockRate	**50000000**
derived_has_tri	**false**
derived_has_out	**false**
derived_has_in	**true**
derived_do_test_bench_wiring	**false**
derived_capture	**true**
derived_edge_type	**RISING**
derived_irq_type	**EDGE**
derived_has_irq	**true**
deviceFamily	**UNKNOWN**
generateLegacySim	**false**

Software Assignments

BIT_CLEARING_EDGE_REGISTER	**0**
BIT_MODIFYING_OUTPUT_REGISTER	**0**
CAPTURE	**1**
DATA_WIDTH	**2**
DO_TEST_BENCH_WIRING	**0**
DRIVEN_SIM_VALUE	**0**
EDGE_TYPE	**RISING**
FREQ	**50000000**
HAS_IN	**1**
HAS_OUT	**0**
HAS_TRI	**0**
IRQ_TYPE	**EDGE**
RESET_VALUE	**0**

pio_1

altera_avalon_pio v13.1



Parameters

bitClearingEdgeCapReg	**false**
bitModifyingOutReg	**false**
captureEdge	**true**
direction	**Input**
edgeType	**ANY**
generateIRQ	**true**
irqType	**EDGE**
resetValue	**0**
simDoTestBenchWiring	**false**
simDrivenValue	**0**
width	**2**
clockRate	**50000000**
derived_has_tri	**false**
derived_has_out	**false**
derived_has_in	**true**
derived_do_test_bench_wiring	**false**
derived_capture	**true**
derived_edge_type	**ANY**
derived_irq_type	**EDGE**
derived_has_irq	**true**
deviceFamily	**UNKNOWN**
generateLegacySim	**false**

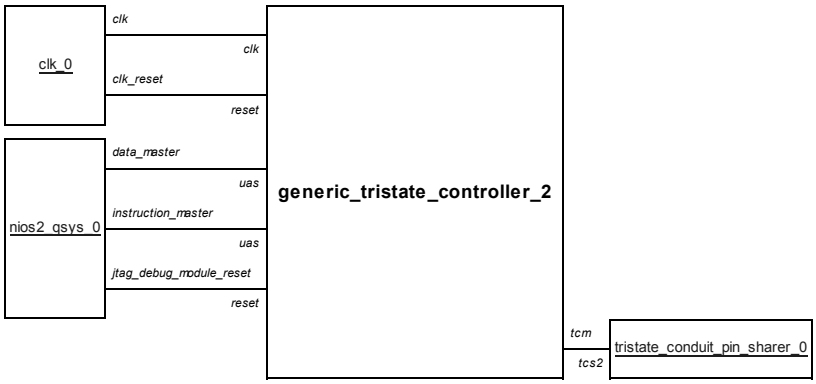
Software Assignments

BIT_CLEARING_EDGE_REGISTER	**0**
BIT_MODIFYING_OUTPUT_REGISTER	**0**
CAPTURE	**1**
DATA_WIDTH	**2**
DO_TEST_BENCH_WIRING	**0**
DRIVEN_SIM_VALUE	**0**
EDGE_TYPE	**ANY**
FREQ	**50000000**
HAS_IN	**1**
HAS_OUT	**0**
HAS_TRI	**0**
IRQ_TYPE	**EDGE**
RESET_VALUE	**0**

---

generic_tristate_controller_2

altera_generic_tristate_controller v13.1



Parameters

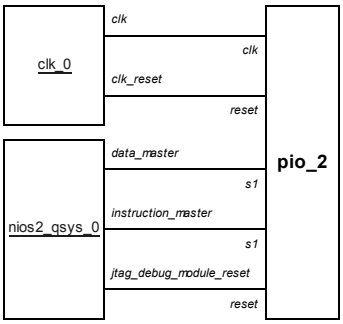
TCM_ADDRESS_W	**16**
TCM_DATA_W	**8**
TCM_BYTEENABLE_W	**1**
TCM_READ_WAIT	**0**
TCM_WRITE_WAIT	**0**
TCM_SETUP_WAIT	**0**
TCM_DATA_HOLD	**0**
TCM_MAX_PENDING_READ_TRANSACTIONS	**1**
TCM_TURNAROUND_TIME	**2**
TCM_TIMING_UNITS	**1**
TCM_READLATENCY	**2**
TCM_SYMBOLS_PER_WORD	**1**
USE_READDATA	**1**
USE_WRITEDATA	**1**
USE_READ	**0**
USE_WRITE	**1**
USE_BEGINTRANSFER	**0**
USE_BYTEENABLE	**0**
USE_CHIPSELECT	**1**
USE_LOCK	**0**
USE_ADDRESS	**1**
USE_WAITREQUEST	**1**
USE_WRITEBYTEENABLE	**0**
USE_OUTPUTENABLE	**0**
USE_RESETREQUEST	**0**
USE_IRQ	**0**
USE_RESET_OUTPUT	**0**
ACTIVE_LOW_READ	**0**
ACTIVE_LOW_LOCK	**0**
ACTIVE_LOW_WRITE	**1**
ACTIVE_LOW_CHIPSELECT	**1**
ACTIVE_LOW_BYTEENABLE	**0**
ACTIVE_LOW_OUTPUTENABLE	**0**
ACTIVE_LOW_WRITEBYTEENABLE	**0**
ACTIVE_LOW_WAITREQUEST	**0**
ACTIVE_LOW_BEGINTRANSFER	**0**
ACTIVE_LOW_RESETREQUEST	**0**
ACTIVE_LOW_IRQ	**0**
ACTIVE_LOW_RESET_OUTPUT	**0**
CHIPSELECT_THROUGH_READLATENCY	**0**
IS_MEMORY_DEVICE	**1**
MODULE_ASSIGNMENT_KEYS	**embeddedsw.configuration.hwClassnameDriverSupportList**
MODULE_ASSIGNMENT_VALUES	**altera_avalon_lan91c111\,altera_avalon_cfi_flash**
INTERFACE_ASSIGNMENT_KEYS	
INTERFACE_ASSIGNMENT_VALUES	
CLOCK_RATE	**50000000**
AUTO_CLK_CLOCK_DOMAIN	**1**
AUTO_CLK_RESET_DOMAIN	**1**
AUTO_TRISTATECONDUIT_MASTERS	
AUTO_DEVICE_FAMILY	**CYCLONEIII**
AUTO_DEVICE	**EP3C25Q240C8**
deviceFamily	**Cyclone III**
generateLegacySim	**false**

Software Assignments

(none)

pio_2

altera_avalon_pio v13.1



Parameters

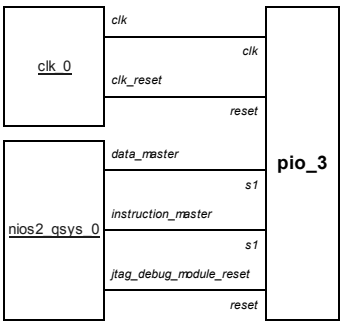
bitClearingEdgeCapReg	**false**
bitModifyingOutReg	**false**
captureEdge	**false**
direction	**Output**
edgeType	**RISING**
generateIRQ	**false**
irqType	**LEVEL**
resetValue	**0**
simDoTestBenchWiring	**false**
simDrivenValue	**0**
width	**1**
clockRate	**50000000**
derived_has_tri	**false**
derived_has_out	**true**
derived_has_in	**false**
derived_do_test_bench_wiring	**false**
derived_capture	**false**
derived_edge_type	**NONE**
derived_irq_type	**NONE**
derived_has_irq	**false**
deviceFamily	**UNKNOWN**
generateLegacySim	**false**

Software Assignments

BIT_CLEARING_EDGE_REGISTER	**0**
BIT_MODIFYING_OUTPUT_REGISTER	**0**
CAPTURE	**0**
DATA_WIDTH	**1**
DO_TEST_BENCH_WIRING	**0**
DRIVEN_SIM_VALUE	**0**
EDGE_TYPE	**NONE**
FREQ	**50000000**
HAS_IN	**0**
HAS_OUT	**1**
HAS_TRI	**0**
IRQ_TYPE	**NONE**
RESET_VALUE	**0**

pio_3

altera_avalon_pio v13.1



Parameters

bitClearingEdgeCapReg	**false**
bitModifyingOutReg	**false**
captureEdge	**false**
direction	**Output**
edgeType	**RISING**
generateIRQ	**false**
irqType	**LEVEL**
resetValue	**0**
simDoTestBenchWiring	**false**
simDrivenValue	**0**
width	**1**
clockRate	**50000000**
derived_has_tri	**false**
derived_has_out	**true**
derived_has_in	**false**
derived_do_test_bench_wiring	**false**
derived_capture	**false**
derived_edge_type	**NONE**
derived_irq_type	**NONE**
derived_has_irq	**false**
deviceFamily	**UNKNOWN**
generateLegacySim	**false**

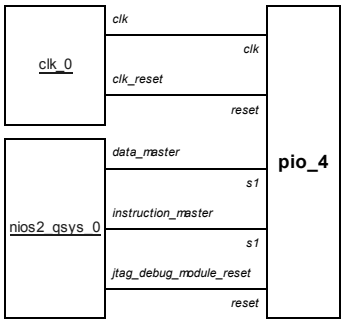
Software Assignments

BIT_CLEARING_EDGE_REGISTER	**0**
BIT_MODIFYING_OUTPUT_REGISTER	**0**
CAPTURE	**0**
DATA_WIDTH	**1**
DO_TEST_BENCH_WIRING	**0**
DRIVEN_SIM_VALUE	**0**
EDGE_TYPE	**NONE**
FREQ	**50000000**
HAS_IN	**0**
HAS_OUT	**1**
HAS_TRI	**0**
IRQ_TYPE	**NONE**
RESET_VALUE	**0**

---

pio_4

altera_avalon_pio v13.1



Parameters

bitClearingEdgeCapReg	**false**
bitModifyingOutReg	**false**
captureEdge	**false**
direction	**Output**
edgeType	**RISING**
generateIRQ	**false**
irqType	**LEVEL**
resetValue	**0**
simDoTestBenchWiring	**false**
simDrivenValue	**0**
width	**8**
clockRate	**50000000**
derived_has_tri	**false**
derived_has_out	**true**
derived_has_in	**false**
derived_do_test_bench_wiring	**false**
derived_capture	**false**
derived_edge_type	**NONE**
derived_irq_type	**NONE**
derived_has_irq	**false**
deviceFamily	**UNKNOWN**
generateLegacySim	**false**

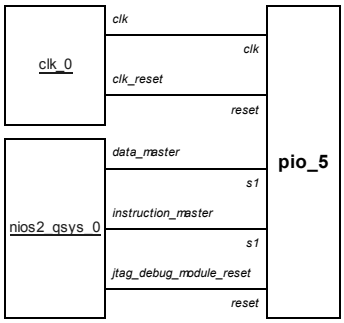
Software Assignments

BIT_CLEARING_EDGE_REGISTER	**0**
BIT_MODIFYING_OUTPUT_REGISTER	**0**
CAPTURE	**0**
DATA_WIDTH	**8**
DO_TEST_BENCH_WIRING	**0**
DRIVEN_SIM_VALUE	**0**
EDGE_TYPE	**NONE**
FREQ	**50000000**
HAS_IN	**0**
HAS_OUT	**1**
HAS_TRI	**0**
IRQ_TYPE	**NONE**
RESET_VALUE	**0**

---

pio_5

altera_avalon_pio v13.1



Parameters

bitClearingEdgeCapReg	**false**
bitModifyingOutReg	**false**
captureEdge	**false**
direction	**Output**
edgeType	**RISING**
generateIRQ	**false**
irqType	**LEVEL**
resetValue	**0**
simDoTestBenchWiring	**false**
simDrivenValue	**0**
width	**1**
clockRate	**50000000**
derived_has_tri	**false**
derived_has_out	**true**
derived_has_in	**false**
derived_do_test_bench_wiring	**false**
derived_capture	**false**
derived_edge_type	**NONE**
derived_irq_type	**NONE**
derived_has_irq	**false**
deviceFamily	**UNKNOWN**
generateLegacySim	**false**

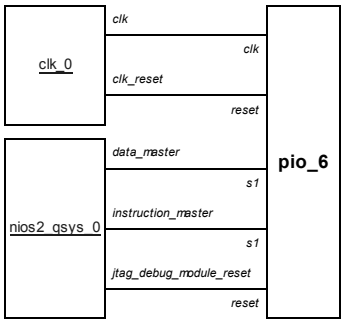
Software Assignments

BIT_CLEARING_EDGE_REGISTER	**0**
BIT_MODIFYING_OUTPUT_REGISTER	**0**
CAPTURE	**0**
DATA_WIDTH	**1**
DO_TEST_BENCH_WIRING	**0**
DRIVEN_SIM_VALUE	**0**
EDGE_TYPE	**NONE**
FREQ	**50000000**
HAS_IN	**0**
HAS_OUT	**1**
HAS_TRI	**0**
IRQ_TYPE	**NONE**
RESET_VALUE	**0**

---

pio_6

altera_avalon_pio v13.1



Parameters

bitClearingEdgeCapReg	**false**
bitModifyingOutReg	**false**
captureEdge	**false**
direction	**Input**
edgeType	**RISING**
generateIRQ	**false**
irqType	**LEVEL**
resetValue	**0**
simDoTestBenchWiring	**false**
simDrivenValue	**0**
width	**7**
clockRate	**50000000**
derived_has_tri	**false**
derived_has_out	**false**
derived_has_in	**true**
derived_do_test_bench_wiring	**false**
derived_capture	**false**
derived_edge_type	**NONE**
derived_irq_type	**NONE**
derived_has_irq	**false**
deviceFamily	**UNKNOWN**
generateLegacySim	**false**

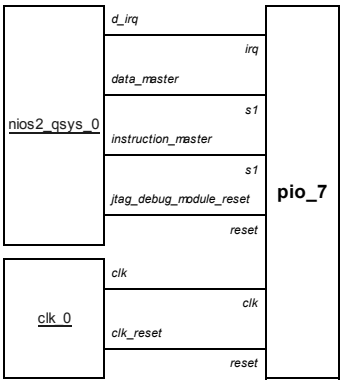
Software Assignments

BIT_CLEARING_EDGE_REGISTER	**0**
BIT_MODIFYING_OUTPUT_REGISTER	**0**
CAPTURE	**0**
DATA_WIDTH	**7**
DO_TEST_BENCH_WIRING	**0**
DRIVEN_SIM_VALUE	**0**
EDGE_TYPE	**NONE**
FREQ	**50000000**
HAS_IN	**1**
HAS_OUT	**0**
HAS_TRI	**0**
IRQ_TYPE	**NONE**
RESET_VALUE	**0**

---

pio_7

altera_avalon_pio v13.1



Parameters

bitClearingEdgeCapReg	**false**
bitModifyingOutReg	**false**
captureEdge	**false**
direction	**Input**
edgeType	**RISING**
generateIRQ	**true**
irqType	**LEVEL**
resetValue	**0**
simDoTestBenchWiring	**false**
simDrivenValue	**0**
width	**1**
clockRate	**50000000**
derived_has_tri	**false**
derived_has_out	**false**
derived_has_in	**true**
derived_do_test_bench_wiring	**false**
derived_capture	**false**
derived_edge_type	**NONE**
derived_irq_type	**LEVEL**
derived_has_irq	**true**
deviceFamily	**UNKNOWN**
generateLegacySim	**false**

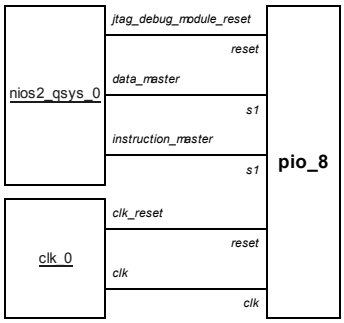
Software Assignments

BIT_CLEARING_EDGE_REGISTER	**0**
BIT_MODIFYING_OUTPUT_REGISTER	**0**
CAPTURE	**0**
DATA_WIDTH	**1**
DO_TEST_BENCH_WIRING	**0**
DRIVEN_SIM_VALUE	**0**
EDGE_TYPE	**NONE**
FREQ	**50000000**
HAS_IN	**1**
HAS_OUT	**0**
HAS_TRI	**0**
IRQ_TYPE	**LEVEL**
RESET_VALUE	**0**

---

pio_8

altera_avalon_pio v13.1



Parameters

bitClearingEdgeCapReg	**false**
bitModifyingOutReg	**false**
captureEdge	**false**
direction	**Output**
edgeType	**RISING**
generateIRQ	**false**
irqType	**LEVEL**
resetValue	**0**
simDoTestBenchWiring	**false**
simDrivenValue	**0**
width	**18**
clockRate	**50000000**
derived_has_tri	**false**
derived_has_out	**true**
derived_has_in	**false**
derived_do_test_bench_wiring	**false**
derived_capture	**false**
derived_edge_type	**NONE**
derived_irq_type	**NONE**
derived_has_irq	**false**
deviceFamily	**UNKNOWN**
generateLegacySim	**false**

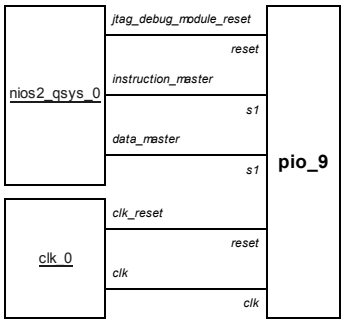
Software Assignments

BIT_CLEARING_EDGE_REGISTER	**0**
BIT_MODIFYING_OUTPUT_REGISTER	**0**
CAPTURE	**0**
DATA_WIDTH	**18**
DO_TEST_BENCH_WIRING	**0**
DRIVEN_SIM_VALUE	**0**
EDGE_TYPE	**NONE**
FREQ	**50000000**
HAS_IN	**0**
HAS_OUT	**1**
HAS_TRI	**0**
IRQ_TYPE	**NONE**
RESET_VALUE	**0**

---

pio_9

altera_avalon_pio v13.1



Parameters

bitClearingEdgeCapReg	**false**
bitModifyingOutReg	**false**
captureEdge	**false**
direction	**Output**
edgeType	**RISING**
generateIRQ	**false**
irqType	**LEVEL**
resetValue	**0**
simDoTestBenchWiring	**false**
simDrivenValue	**0**
width	**16**
clockRate	**50000000**
derived_has_tri	**false**
derived_has_out	**true**
derived_has_in	**false**
derived_do_test_bench_wiring	**false**
derived_capture	**false**
derived_edge_type	**NONE**
derived_irq_type	**NONE**
derived_has_irq	**false**
deviceFamily	**UNKNOWN**
generateLegacySim	**false**

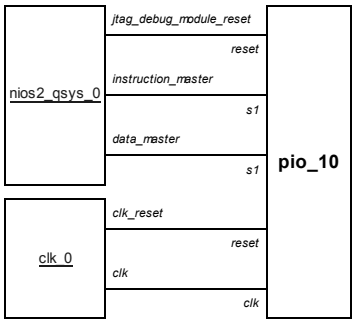
Software Assignments

BIT_CLEARING_EDGE_REGISTER	**0**
BIT_MODIFYING_OUTPUT_REGISTER	**0**
CAPTURE	**0**
DATA_WIDTH	**16**
DO_TEST_BENCH_WIRING	**0**
DRIVEN_SIM_VALUE	**0**
EDGE_TYPE	**NONE**
FREQ	**50000000**
HAS_IN	**0**
HAS_OUT	**1**
HAS_TRI	**0**
IRQ_TYPE	**NONE**
RESET_VALUE	**0**

---

pio_10

altera_avalon_pio v13.1



Parameters

bitClearingEdgeCapReg	**false**
bitModifyingOutReg	**false**
captureEdge	**false**
direction	**Output**
edgeType	**RISING**
generateIRQ	**false**
irqType	**LEVEL**
resetValue	**0**
simDoTestBenchWiring	**false**
simDrivenValue	**0**
width	**1**
clockRate	**50000000**
derived_has_tri	**false**
derived_has_out	**true**
derived_has_in	**false**
derived_do_test_bench_wiring	**false**
derived_capture	**false**
derived_edge_type	**NONE**
derived_irq_type	**NONE**
derived_has_irq	**false**
deviceFamily	**UNKNOWN**
generateLegacySim	**false**

Software Assignments

BIT_CLEARING_EDGE_REGISTER	**0**
BIT_MODIFYING_OUTPUT_REGISTER	**0**
CAPTURE	**0**
DATA_WIDTH	**1**
DO_TEST_BENCH_WIRING	**0**
DRIVEN_SIM_VALUE	**0**
EDGE_TYPE	**NONE**
FREQ	**50000000**
HAS_IN	**0**
HAS_OUT	**1**
HAS_TRI	**0**
IRQ_TYPE	**NONE**
RESET_VALUE	**0**

Appendix F - Quartus Pin Planner

Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard
a16	Output	PIN_95	4	B4_N0	PIN_95	3.3-V LVCMOS
Addr[16]	Output	PIN_88	3	B3_N0	PIN_88	3.3-V LVCMOS
Addr[15]	Output	PIN_57	2	B2_N0	PIN_57	3.3-V LVCMOS
Addr[14]	Output	PIN_56	2	B2_N0	PIN_56	3.3-V LVCMOS
Addr[13]	Output	PIN_55	2	B2_N0	PIN_55	3.3-V LVCMOS
Addr[12]	Output	PIN_52	2	B2_N0	PIN_52	3.3-V LVCMOS
Addr[11]	Output	PIN_51	2	B2_N0	PIN_51	3.3-V LVCMOS
Addr[10]	Output	PIN_50	2	B2_N0	PIN_50	3.3-V LVCMOS
Addr[9]	Output	PIN_49	2	B2_N0	PIN_49	3.3-V LVCMOS
Addr[8]	Output	PIN_46	2	B2_N0	PIN_46	3.3-V LVCMOS
Addr[7]	Output	PIN_45	2	B2_N0	PIN_45	3.3-V LVCMOS
Addr[6]	Output	PIN_44	2	B2_N0	PIN_44	3.3-V LVCMOS
Addr[5]	Output	PIN_43	2	B2_N0	PIN_43	3.3-V LVCMOS
Addr[4]	Output	PIN_41	2	B2_N0	PIN_41	3.3-V LVCMOS
Addr[3]	Output	PIN_39	2	B2_N0	PIN_39	3.3-V LVCMOS
Addr[2]	Output	PIN_38	2	B2_N0	PIN_38	3.3-V LVCMOS
Addr[1]	Output	PIN_37	2	B2_N0	PIN_37	3.3-V LVCMOS
Addr[0]	Output	PIN_22	1	B1_N0	PIN_22	
altera_reserved_tck	Input				PIN_27	3.3-V LVCMOS
altera_reserved_tdi	Input				PIN_26	3.3-V LVCMOS
altera_reserved_tdo	Output				PIN_29	3.3-V LVCMOS
altera_reserved_tms	Input				PIN_28	3.3-V LVCMOS
Data[7]	Bidir	PIN_181	7	B7_N0	PIN_181	3.3-V LVCMOS
Data[6]	Bidir	PIN_177	6	B6_N0	PIN_177	3.3-V LVCMOS
Data[5]	Bidir	PIN_176	6	B6_N0	PIN_176	3.3-V LVCMOS
Data[4]	Bidir	PIN_173	6	B6_N0	PIN_173	3.3-V LVCMOS
Data[3]	Bidir	PIN_171	6	B6_N0	PIN_171	3.3-V LVCMOS
Data[2]	Bidir	PIN_169	6	B6_N0	PIN_169	3.3-V LVCMOS
Data[1]	Bidir	PIN_168	6	B6_N0	PIN_168	3.3-V LVCMOS
Data[0]	Bidir	PIN_167	6	B6_N0	PIN_167	3.3-V LVCMOS
DataDir	Output	PIN_194	7	B7_N0	PIN_194	3.3-V LVCMOS
FLM	Output	PIN_237	8	B8_N0	PIN_237	3.3-V LVCMOS
INVRW	Output	PIN_87	3	B3_N0	PIN_87	3.3-V LVCMOS
M	Output	PIN_137	5	B5_N0	PIN_137	3.3-V LVCMOS
MenuButton	Input	PIN_76	3	B3_N0	PIN_76	3.3-V LVCMOS
pixel_clk	Output	PIN_239	8	B8_N0	PIN_239	3.3-V LVCMOS
R/W	Output	PIN_98	4	B4_N0	PIN_98	3.3-V LVCMOS
RAMCS	Output	PIN_99	4	B4_N0	PIN_99	3.3-V LVCMOS

ROMCS	Output	PIN_100	4	B4_N0	PIN_100	3.3-V LVCMOS
rot[0]	Input	PIN_80	3	B3_N0	PIN_80	3.3-V LVCMOS
rot[1]	Input	PIN_81	3	B3_N0	PIN_81	3.3-V LVCMOS
row_clk	Output	PIN_238	8	B8_N0	PIN_238	3.3-V LVCMOS
sample_clk	Output	PIN_139	5	B5_N0	PIN_139	3.3-V LVCMOS
signal[7]	Input	PIN_142	5	B5_N0	PIN_142	3.3-V LVCMOS
signal[6]	Input	PIN_143	5	B5_N0	PIN_143	3.3-V LVCMOS
signal[5]	Input	PIN_146	5	B5_N0	PIN_146	3.3-V LVCMOS
signal[4]	Input	PIN_147	5	B5_N0	PIN_147	3.3-V LVCMOS
signal[3]	Input	PIN_148	5	B5_N0	PIN_148	3.3-V LVCMOS
signal[2]	Input	PIN_160	6	B6_N0	PIN_160	3.3-V LVCMOS
signal[1]	Input	PIN_161	6	B6_N0	PIN_161	3.3-V LVCMOS
sys_clk	Input	PIN_31	1	B1_N0	PIN_31	
ToggleButton	Input	PIN_78	3	B3_N0	PIN_78	3.3-V LVCMOS
bridge_out_ROMRW	Output				PIN_21	
VRAM_CAS	Output	PIN_234	8	B8_N0	PIN_234	3.3-V LVCMOS
VRAM_DT	Output	PIN_235	8	B8_N0	PIN_235	3.3-V LVCMOS
VRAM_RAS	Output	PIN_233	8	B8_N0	PIN_233	3.3-V LVCMOS
VRAM_SCLK	Output	PIN_236	8	B8_N0	PIN_236	3.3-V LVCMOS
VRAM_WE	Output	PIN_230	8	B8_N0	PIN_230	3.3-V LVCMOS
VRAMA[7]	Output	PIN_202	7	B7_N0	PIN_202	3.3-V LVCMOS
VRAMA[6]	Output	PIN_217	8	B8_N0	PIN_217	3.3-V LVCMOS
VRAMA[5]	Output	PIN_216	8	B8_N0	PIN_216	3.3-V LVCMOS
VRAMA[4]	Output	PIN_207	7	B7_N0	PIN_207	3.3-V LVCMOS
VRAMA[3]	Output	PIN_214	8	B8_N0	PIN_214	3.3-V LVCMOS
VRAMA[2]	Output	PIN_203	7	B7_N0	PIN_203	3.3-V LVCMOS
VRAMA[1]	Output	PIN_224	8	B8_N0	PIN_224	3.3-V LVCMOS
VRAMA[0]	Output	PIN_223	8	B8_N0	PIN_223	3.3-V LVCMOS

Appendix G - Code for Interfacing with Hardware

```
 1 /*************************************************************************/
 2 /*                                                                       */
 3 /*                                MAINLOOP                               */
 4 /*                            Main Program Loop                          */
 5 /*                        Digital Oscilloscope Project                   */
 6 /*                                 EE/CS 52                              */
 7 /*                                                                       */
 8 /*************************************************************************/
 9
10 /*
11    This file contains the main processing loop (background) for the Digital
12    Oscilloscope project.  The only global function included is:
13       main - background processing loop
14
15    The local functions included are:
16       key_lookup - get a key and look up its keycode
17
18    The locally global variable definitions included are:
19       none
20
21
22    Revision History
23       3/8/94   Glen George       Initial revision.
24       3/9/94   Glen George       Changed initialized const arrays to static
25                  (in addition to const).
26       3/9/94   Glen George       Moved the position of the const keyword in
27                  declarations of arrays of pointers.
28       3/13/94  Glen George       Updated comments.
29       3/13/94  Glen George       Removed display_menu call after plot_trace,
30                  the plot function takes care of the menu.
31       3/17/97  Glen George       Updated comments.
32       3/17/97  Glen George       Made key_lookup function static to make it
33                  truly local.
34       3/17/97  Glen George       Removed KEY_UNUSED and KEYCODE_UNUSED
35                  references (no longer used).
36       5/27/08  Glen George       Changed code to only check for sample done if
37                  it is currently sampling.
38
39       5/16/14  Daniel Andrade    Included code to initialize the keypad.
40       6/10/14  Daniel Andrade    Updated with triggering code.
41 */
42
43
44
45 /* library include files */
46   /* none */
47
48 /* local include files */
49 #include  "interfac.h"
50 #include  "scopedef.h"
51 #include  "keyproc.h"
52 #include  "menu.h"
53 #include  "tracutil.h"
54 #include  "unistd.h"
55
56
57
```

```
 58
 59 /* local function declarations */
 60 enum keycode  key_lookup(void);      /* translate key values into keycodes */
 61
 62
 63 /* defined in assembly, keypad2.S */
 64 extern void keypad_init();
 65 extern unsigned char key_available();
 66 extern int getkey();
 67
 68 /* display.S */
 69 extern void clear_display();
 70
 71 /* trigger.S */
 72 extern void trigger_init();
 73
 74 /*
 75    main
 76
 77    Description:      This procedure is the main program loop for the Digital
 78                      Oscilloscope.  It loops getting keys from the keypad,
 79                      processing those keys as is appropriate.  It also handles
 80                      starting scope sample collection and updating the LCD
 81              screen.
 82
 83    Arguments:        None.
 84    Return Value:     (int) - return code, always 0 (never returns).
 85
 86    Input:            Keys from the keypad.
 87    Output:           Traces and menus to the display.
 88
 89    Error Handling:   Invalid input is ignored.
 90
 91    Algorithms:       The function is table-driven.  The processing routines
 92                      for each input are given in tables which are selected
 93                      based on the context (state) the program is operating in.
 94    Data Structures:  Array (process_key) to associate keys with actions
 95              (functions to call).
 96
 97    Global Variables: None.
 98
 99    Author:           Glen George
100    Last Modified:    May 27, 2008
101
102 */
103
104 int  main()
105 {
106     /* variables */
107     enum keycode        key;            /* an input key */
108
109     enum status         state = MENU_ON;    /* current program state */
110
111     unsigned char      *sample;         /* a captured trace */
112
113     /* key processing functions (one for each system state type and key) */
114     static enum status  (* const process_key[NUM_KEYCODES][NUM_STATES])(enum status) =
```

```
115        /*   Current System State                          */
116        /*  MENU_ON      MENU_OFF             Input Key  */
117       { {  menu_key,    menu_key    },   /* <Menu>      */
118         {  menu_up,     no_action   },   /* <Up>        */
119         {  menu_down,   no_action   },   /* <Down>      */
120         {  menu_left,   no_action   },   /* <Left>      */
121         {  menu_right,  no_action   },   /* <Right>     */
122         {  no_action,   no_action   } }; /* illegal key */
123
124
125
126     /* first initialize everything */
127     clear_display();        /* clear the display */
128
129     keypad_init();      /* initialize the keypad */
130     init_trace();       /* initialize the trace routines */
131     init_menu();        /* initialize the menu system */
132     trigger_init();
133
134
135
136     /* infinite loop processing input */
137     while(TRUE)  {
138
139         /* check if ready to do a trace */
140     if (trace_rdy())
141         /* ready for a trace - do it */
142         do_trace();
143
144
145     /* check if have a trace to display */
146     if (is_sampling() && ((sample = sample_done()) != NULL))  {
147
148         /* have a trace - output it */
149         plot_trace(sample);
150         /* done processing this trace */
151         trace_done();
152     }
153
154
155     /* now check for keypad input */
156     if (key_available())  {
157
158         /* have keypad input - get the key */
159         key = key_lookup();
160
161         /* execute processing routine for that key */
162         state = process_key[key][state](state);
163     }
164     }
165
166
167     /* done with main (never should get here), return 0 */
168     return  0;
169
170 }
171
```

```
172
173 /*`
174    key_lookup
175
176    Description:      This function gets a key from the keypad and translates
177                      the raw keycode to an enumerated keycode for the main
178                      loop.
179
180    Arguments:        None.
181    Return Value:     (enum keycode) - type of the key input on keypad.
182
183    Input:            Keys from the keypad.
184    Output:           None.
185
186    Error Handling:   Invalid keys are returned as KEYCODE_ILLEGAL.
187
188    Algorithms:       The function uses an array to lookup the key types.
189    Data Structures:  Array of key types versus key codes.
190
191    Global Variables: None.
192
193    Author:           Glen George
194    Last Modified:    Mar. 17, 1997
195
196 */
197
198 enum keycode  key_lookup()
199 {
200     /* variables */
201
202     const static enum keycode  keycodes[] = /* array of keycodes */
203         {                           /* order must match keys array exactly */
204            KEYCODE_MENU,      /* <Menu>     */   /* also need an extra element */
205        KEYCODE_UP,        /* <Up>       */   /* for unknown key codes */
206        KEYCODE_DOWN,      /* <Down>     */
207        KEYCODE_LEFT,      /* <Left>     */
208        KEYCODE_RIGHT,     /* <Right>    */
209        KEYCODE_ILLEGAL    /* other keys */
210         };
211
212     const static int  keys[] =   /* array of key values */
213         {            /* order must match keycodes array exactly */
214            KEY_MENU,    /* <Menu>     */
215        KEY_UP,      /* <Up>       */
216        KEY_DOWN,    /* <Down>     */
217        KEY_LEFT,    /* <Left>     */
218        KEY_RIGHT,   /* <Right>    */
219         };
220
221     int  key;       /* an input key */
222
223     int  i;             /* general loop index */
224
225
226
227     /* get a key */
228     key = getkey();
```

```
229
230
231     /* lookup key in keys array */
232     for (i = 0; ((i < (sizeof(keys)/sizeof(int))) && (key != keys[i])); i++);
233
234
235     /* return the appropriate key type */
236     return  keycodes[i];
237
238 }
239
```

```
 1
 2 ###############################################################################
 3 #                                                                             #
 4 #                                 Interfac.m                                  #
 5 #                             Keypad Include File                             #
 6 #                                 EE/CS 52                                    #
 7 #                                                                             #
 8 ###############################################################################
 9
10 /*
11 * Daniel Andrade
12 * EE/CS 52
13 * TA: Dan Pipe-Mazo
14 *
15 * File Description: Peripheals/PIO constants for keypad handling.
16 *
17 * Revision History:
18 *       05/10/2014      Daniel Andrade      Constants for peripheals
19 *       05/18/2014      Daniel Andrade      Constants for the encoders
20 *
21 */
22
23
24 .equ    periphBaseKey,     0x000c1150;      # Base address of peripheals for keys
25 .equ    periphBaseRot,     0x000c1140;      # Base address of peripheals for encoders
26 .equ    edgeCaptOffset,    0xC;             # Beginning of edge capture register
27 .equ    maskRegOffset,     0x8;             # Begging of mask register
28 .equ    unmaskMask,        0x3;             # Unmask the interrupts with this mask
29 .equ    toggleMask,        0xFFFFFFFE;      # Used to clear toggle bit in PIO
30 .equ    menuMask,          0xFFFFFFFD;      # Used to clear menu bit in PIO
31 .equ    encoderPos,        0x3;             # Detent position
32
33 .equ    KEY_MENU,           0;              # <Menu>
34 .equ    KEY_UP,             1;              # <Up>
35 .equ    KEY_DOWN,           2;              # <Down>
36 .equ    KEY_LEFT,           3;              # <Left>
37 .equ    KEY_RIGHT,          4;              # <Right>
```

```
1 ##############################################################################
2 #                                                                            #
3 #                                  Keypad2                                   #
4 #                    Rotary Encoder and Menu Button Routines                 #
5 #                                  EE/CS 52                                  #
6 #                                                                            #
7 ##############################################################################
8
9
10 /*
11  *  Daniel Andrade
12  *  EE/CS 52
13  *  TA: Dan Pipe-Mazo
14  *
15  *  File Description:  Keypad functions (accessors and handlers).
16  *
17  *  Table of Contents:
18  *     keypad_init:           Initialization function for the keypad
19  *     key_interrupt_handler: Interrupt handler for keys
20  *     rot_interrupt_handler: Interrupt handler for encoders
21  *     key_available:         Returns whether a key has been pressed
22  *     getkey:                Returns which key was pressed last
23  *
24  *  Revision History:
25  *      05/10/2014      Daniel Andrade      Initial Revision
26  *      05/11/2014      Daniel Andrade      Some touch-up before compiling
27  *      05/18/2014      Daniel Andrade      Rotary encoder code
28  *      05/19/2014      Daniel Andrade      New code for hardware decoded
29  *                                            encoder.
30  *
31  */
32
33  /*  Local Include Files   */
34 .include "interfac.m"
35 .include "macros.m"
36 .include "general.m"
37 #include "system.h"
38
39 .section  .text         # Start code section
40
41  /*
42  *  keypad_init
43  *
44  *  Description: Initializes the keypad shared variables and interrupts. Install
45  *               the interrupt handlers for the keypad.
46  *
47  *  Operation:   Initializes the shared variables to known values. Calls the
48  *               function alt_ic_isr_register() to install the handlers and
49  *               unmasks the interrupts in the mask register.
50  *
51  *  Arguments:          None.
52  *
53  *  Return Value:       None.
54  *
55  *  Local Variables:    TODO.
56  *
57  *  Shared Variables:   'button_pressed' - Flag variable: whether a button
```

```
 58 *                          has been pressed since last check (write)
 59 *                      'current_button' - Last button pressed (write)
 60 *                      'toggled' - Whether encoder is toggled or not  (write)
 61 *                      TODO: Rot variables
 62 *
 63 *  Global Variables:   None.
 64 *
 65 *  Input:              None.
 66 *
 67 *  Output:             None.
 68 *
 69 *  Error Handling:     None.
 70 *
 71 *  Limitations:        None.
 72 *
 73 *  Algorithms:         None.
 74 *  Data Structures:    None.
 75 *
 76 *  Registers Changed:  None.
 77 *
 78 *  Revision History:
 79 *      05/10/2014    Daniel Andrade    initial revision
 80 *
 81 */
 82 .global keypad_init
 83 .type keypad_init,@function
 84
 85 keypad_init:
 86     BEGINF
 87
 88     PUSH    r16             # Push the used registers (all callee-saved)
 89     PUSH    r17
 90     PUSH    r18
 91
 92 keypad_init_vars:
 93     movia   r16, button_pressed          # Zero is a non-breaking value for
 94     stb     r0, 0(r16)                   #  all of these, so just initialize to
 95     movia   r16, current_button          #  that to keep it simple.
 96     stb     r0, 0(r16)
 97     movia   r16, toggled
 98     movi    r15, 1
 99     stb     r15, 0(r16)
100
101 keypad_init_handlers:
102     mov     r4, r0                       # Install the first handler
103     mov     r5, r0                       # Key handler has IRQ of 0
104     movia   r6, key_interrupt_handler    # Address to handler function
105     mov     r7, r0                       # No data to pass
106     PUSH    r0                           # And flags should always be zero
107     call    alt_ic_isr_register
108     POP     r16
109
110     mov     r4, r0                       # Install the second handler
111     movi    r5, 1                        # Key handler has IRQ of 1
112     movia   r6, rot_interrupt_handler    # Address to handler function
113     mov     r7, r0                       # No data to pass
114     PUSH    r0                           # And flags should always be zero
```

```
115     call    alt_ic_isr_register
116     POP     r16
117
118 keypad_unmask_bits:
119     movia   r16, periphBaseKey          # Unmask the interrupts for the keys
120     movi    r17, unmaskMask
121     stwio   r17, maskRegOffset(r16)
122
123     movia   r16, periphBaseRot
124     stwio   r0,  edgeCaptOffset(r16)
125
126     movia   r16, periphBaseRot          # Unmask the interrupts for encoders
127     movi    r17, unmaskMask
128     stwio   r17, maskRegOffset(r16)
129
130     POP     r18
131     POP     r17
132     POP     r16
133
134     ENDF
135
136 /*
137  *  key_interrupt_handler
138  *
139  *  Description: The interrupt handler for a key press. This handler does not
140  *               handle rotary encoder itself, but it does handle the
141  *               pushbutton on the rotary encoder. Prioritizes the menu key
142  *               over the toggle key if both keys are pressed at the same time.
143  *
144  *  Operation:   Checks the PIO register to determine which button has been
145  *               pressed and sets the shared variable 'current_button' to the
146  *               menu key if it was pressed. If the toggle key was pressed,
147  *               then toggles the toggled variable. Sets the flag variable
148  *               'button_pressed' if the menu button was pressed.
149  *
150  *  Arguments:          None.
151  *
152  *  Return Value:       None (clears the edge capture register appropriately).
153  *
154  *  Local Variables:    None.
155  *
156  *  Shared Variables:   'button_pressed' - Flag variable: whether a button
157  *                           has been pressed since last check (write)
158  *                      'current_button' - Last button pressed (write)
159  *                      'toggled' - Whether encoder is toggled or not  (write)
160  *
161  *  Global Variables:   None.
162  *
163  *  Input:              None (although interrupts are caused by button presses).
164  *
165  *  Output:             None.
166  *
167  *  Error Handling:     None.
168  *
169  *  Limitations:        Only keeps track of last button pressed, so multiple,
170  *                      quick button presses will be ignored.
171  *                      Multiple button presses are ignored,
```

```
172 *
173 *  Algorithms:         None.
174 *  Data Structures:    None.
175 *
176 *  Registers Changed:  None.
177 *
178 *  Revision History:
179 *     5/10/2014    Daniel Andrade  initial revision
180 */
181
182 .global key_interrupt_handler
183 .type key_interrupt_handler,@function
184
185 key_interrupt_handler:
186     BEGINF
187
188     PUSH    r16             # Push the used registers (all callee-saved)
189     PUSH    r17
190     PUSH    r18
191
192 KIH_check:
193     movhi   r16, %hi(periphBaseKey)     # Get the value of the edge capture
194     ori     r16, r16, %lo(periphBaseKey) #   register.
195     ldwio   r17, edgeCaptOffset(r16)
196     movi    r16, 2
197     bgeu    r17, r16, KIH_menu_press   # Anything that's not one is a menu press
198
199 KIH_toggle_press:
200     movia   r16, toggled               # Otherwise we press the toggle key
201     ldbu    r18, 0(r16)                # Get the value of the toggle variable
202     movi    r17, 0xFFFFFFFE            #    and invert it by NORing it.
203     nor     r18, r18, r17
204     stb     r18, 0(r16)
205     movi    r17, toggleMask            # Will be used to toggle the bit in the
206     br      KIH_clean_up               #    edge capture register.
207
208 KIH_menu_press:
209     movia   r16, button_pressed        # Indicate a button was pressed
210     movi    r18, TRUE
211     stb     r18, 0(r16)
212
213     movia   r16, current_button        # If it's a menu key press, just move
214     movi    r18, KEY_MENU              #   KEY_MENU into the current_button
215     stb     r18, 0(r16)                #   variable.
216
217     movi    r17, menuMask              # Will be used to toggle the bit in the
218                                        #    edge capture register.
219 KIH_clean_up:
220     movhi   r16, %hi(periphBaseKey)     # Don't forget to clear the register bit
221     ori     r16, r16, %lo(periphBaseKey) #    by anding it with the correct mask
222     ldwio   r18, edgeCaptOffset(r16)
223     and     r18, r18, r17
224     stwio   r18,  edgeCaptOffset(r16)
225
226     POP     r18                        # Now just clean up used registers
227     POP     r17
228     POP     r16
```

```
229
230     ENDF
231
232 /*
233  *  rot_interrupt_handler
234  *
235  *  Description: The interrupt handler for rotary encoders. Gets the LEFT/RIGHT
236  *               or UP/DOWN key presses.
237  *               Does not handle the pushbutton on the encoder.
238  *
239  *  Operation:   Checks the PIO edge capture register to determine whether the
240  *               encoder moved left or right. Does no decoding; that's done
241  *               in hardware. If the 'toggled' variable is FALSE, it moves
242  *               LEFT/RIGHT, otherwise it moves UP/DOWN.
243  *
244  *  Arguments:          None.
245  *
246  *  Return Value:       None (clears the edge capture register appropriately).
247  *
248  *  Local Variables:    None.
249  *
250  *  Shared Variables:   'button_pressed' - Flag variable: whether a button
251  *                           has been pressed since last check (write)
252  *                      'current_button' - Last button pressed (write)
253  *                      'toggled' - Whether encoder is toggled or not  (read)
254  *
255  *  Global Variables:   None.
256  *
257  *  Input:              None (although interrupts are caused by encoder moving).
258  *
259  *  Output:             None.
260  *
261  *  Error Handling:     None.
262  *
263  *  Limitations:        Only keeps track of last button pressed, so multiple,
264  *                      quick button presses will be ignored.
265  *                      Multiple button presses are ignored,
266  *
267  *  Algorithms:         None.
268  *  Data Structures:    None.
269  *
270  *  Registers Changed:  None.
271  *
272  *  Revision History:
273  *     5/11/2014    Daniel Andrade  initial revision
274  *     05/19/2014   Daniel Andrade  rewrote code; encoders now decoded in
275  *                                    hardware.
276  */
277
278 .global rot_interrupt_handler
279 .type rot_interrupt_handler,@function
280
281 rot_interrupt_handler:
282     BEGINF
283
284     PUSH    r16                         # Push the used registers
285     PUSH    r17                         #   (all callee-saved).
```

```
286     PUSH    r18
287
288 RIH_get_value:
289     movhi   r16, %hi(periphBaseRot)      # Get the value of the data
290     ori     r16, r16, %lo(periphBaseRot) #   register to see what is high
291     ldwio   r17, edgeCaptOffset(r16)
292
293 RIH_get_toggle:
294     movia   r16, button_pressed          # Indicate that a button was pressed
295     movi    r18, TRUE
296     stb     r18, 0(r16)
297
298     movia   r16, toggled                 # Check to see if encoder is toggled.
299     ldbu    r16, (r16)
300     movi    r18, TRUE
301     beq     r16, r18, RIH_toggled_dir    # If it is, then jump. Otherwise,
302                                          # can just continue.
303
304 RIH_utoggled_dir:
305     movi    r18, 2                       # If greater than or equal to 2,
306     bgeu    r17, r18, RIH_right          #    encoder moved right.
307
308 RIH_left:
309     movi    r18, KEY_LEFT                # Moved left
310     br      RIH_store_last
311
312 RIH_right:
313     movi    r18, KEY_RIGHT               # Moved right
314     br      RIH_store_last
315
316 RIH_toggled_dir:
317     movi    r18, 2                       # If greater than or equal to 2,
318     bgeu    r17, r18, RIH_down           #    encoder down.
319
320 RIH_up:
321     movi    r18, KEY_UP                  # Moved up
322     br      RIH_store_last
323
324 RIH_down:
325     movi    r18, KEY_DOWN                # Moved down
326
327 RIH_store_last:
328     movia   r16, current_button          # Store the pressed key in
329     stb     r18, (r16)                   #   current_button.
330
331 RIH_clean_up:
332     movhi   r16, %hi(periphBaseRot)      # Clear register bits to stop
333     ori     r16, r16, %lo(periphBaseRot) #   interrupting (clear everything).
334     stwio   r0,  edgeCaptOffset(r16)
335
336     POP     r18
337     POP     r17
338     POP     r16
339
340     ENDF
341
342 /*
```

```
343 *  key_available
344 *
345 *  Description:  Returns TRUE if a key is available, FALSE otherwise.
346 *
347 *  Operation:    Returns the value of 'button_pressed.'
348 *
349 *  Arguments:    None.
350 *
351 *  Return Value:        TRUE if a key is available, FALSE otherwise.
352 *
353 *  Local Variables:     None.
354 *
355 *  Shared Variables:    'button_pressed' - whether a button has been pressed.
356 *                             (read)
357 *
358 *  Global Variables:    None.
359 *
360 *  Input:               None.
361 *
362 *  Output:              None.
363 *
364 *  Error Handling:      None.
365 *
366 *  Limitations:         None (but see limitations on handlers).
367 *
368 *  Algorithms:          None.
369 *  Data Structures:     None.
370 *
371 *  Registers Changed:   r4 - returned value.
372 *
373 *  Revision History:
374 *      05/11/2014    Daniel Andrade     initial revision
375 *
376 */
377
378 .global key_available
379 .type key_available,@function
380
381 key_available:
382     BEGINF
383
384     PUSH    r16             # Push the used registers (callee-saved)
385
386     movia   r16, button_pressed
387     mov     r2, r0
388     ldbu    r2, 0(r16)
389
390 KeyAvailableDone:
391     POP     r16
392
393     ENDF
394
395 /*
396 *  getkey
397 *
398 *  Description:  Returns the key value that was pressed last.
399 *
```

```
400 *  Operation:    Returns the value stored in 'current_button'.
401 *
402 *  Arguments:           None.
403 *
404 *  Return Value:        Returns the value stored in 'current_button'
405 *
406 *  Local Variables:     None.
407 *
408 *  Shared Variables:    'current_button' - Last button pressed (read)
409 *
410 *  Global Variables:    None.
411 *
412 *  Input:               None.
413 *
414 *  Output:              None.
415 *
416 *  Error Handling:      None.
417 *
418 *  Limitations:         None (see limitations on handlers).
419 *
420 *  Algorithms:          None.
421 *  Data Structures:     None.
422 *
423 *  Registers Changed:   r4 - Return value.
424 *
425 *  Revision History:
426 *      05/10/2014      Daniel Andrade      initial revision
427 *      05/11/2014      Daniel Andrade      forgot to clear button_pressed here
428 *
429 */
430
431 .global getkey
432 .type getkey,@function
433
434 getkey:
435     BEGINF
436
437     PUSH    r16                     # Push the used registers (callee-saved)
438
439     movia   r16, current_button     # Move current button into return value
440     mov     r2, r0
441     ldbu    r2, 0(r16)
442
443     movia   r16, button_pressed     # Now there's no button pressed
444     stb     r0, 0(r16)
445
446 getkeydone:
447     POP     r16
448
449     ENDF
450
451
452 .section  .data     #start data section
453 button_pressed:     .byte   0             # Whether a button is pressed
454 current_button:     .byte   0             # Which button was pressed last
455 toggled:            .byte   0             # Is the rotary encoder toggled?
456
```

457
458

```
1 ##############################################################################
2 #                                                                            #
3 #                              display.m                                     #
4 #                          Display Include File                              #
5 #                               EE/CS 52                                     #
6 #                                                                            #
7 ##############################################################################
8
9 /*
10 * Daniel Andrade
11 * EE/CS 52
12 * TA: Dan Pipe-Mazo
13 *
14 * File Description: Constants for Display
15 *
16 * Revision History:
17 *       05/24/2014      Daniel Andrade      Initial revision
18 *
19 */
20
21 .equ    colsPerRow,          256;      # Columns per row of VRAM
22 .equ    pixPerQH,            240;      # Pixels per horizontal quadrant of display
23 .equ    pixPerQV,             64;      # Pixels per vertical quadrant of display
24 .equ    PX_Q1,                 1;      # Bit mask for quadrant one
25 .equ    PX_Q2,                 2;      # Bit mask for quadrant two
26 .equ    PX_Q3,                 4;      # Bit mask for quadrant three
27 .equ    PX_Q4,                 8;      # Bit mask for quardant four
28
```

```
1 ##############################################################################
2 #                                                                            #
3 #                                  Display                                   #
4 #                               Display Routines                             #
5 #                                  EE/CS 52                                  #
6 #                                                                            #
7 ##############################################################################
8
9
10 /*
11  *  Daniel Andrade
12  *  EE/CS 52
13  *  TA: Dan Pipe-Mazo
14  *
15  *  File Description:  Code for changing pixels on the display.
16  *
17  *  Table of Contents:
18  *     clear_display(): Clears the display (makes all the pixels white)
19  *     plot_pixel(int x, int y, int p): Writes a pixel to the display at (x, y)
20  *                                      Argument p specifies color of pixel.
21  *
22  *  Revision History:
23  *      06/10/2014      Daniel Andrade      Initial Revision.
24  *      06/14/2014      Daniel Andrade      'Fixed' hardware issues in code for
25  *                                            plotting pixels.
26  *
27  */
28
29 .include "vramtest.m"
30 .include "macros.m"
31 .include "display.m"
32 #include "interfac.h"
33
34 /*
35  *  clear_display
36  *
37  *  Description: Clears the display - writes white pixels to all values
38  *
39  *  Operation:   Writes zeroes to all values in VRAM.
40  *
41  *  Arguments:          None.
42  *
43  *  Return Value:       None.
44  *
45  *  Local Variables:    None.
46  *
47  *  Shared Variables:   None.
48  *
49  *  Global Variables:   None.
50  *
51  *  Input:              None.
52  *
53  *  Output:             None.
54  *
55  *  Error Handling:     None.
56  *
57  *  Limitations:        None.
```

```
 58 *
 59 *  Algorithms:         None.
 60 *  Data Structures:    None.
 61 *
 62 *  Registers Changed:  r8 - r9 (these are supposedly caller saved).
 63 *
 64 *  Revision History:
 65 *      06/10/2014    Daniel Andrade    initial revision
 66 *
 67 *
 68 */
 69
 70 .global clear_display
 71 .type clear_display,@function
 72
 73 clear_display:
 74     BEGINF
 75
 76     movhi   r8, %hi(vramBase)             # Start writing at the base address
 77     ori     r8, r8, %lo(vramBase)
 78
 79     movhi   r9, %hi(vramEndAddr)
 80     ori     r9, r9, %lo(vramEndAddr)
 81
 82 CD_loop:
 83     bgtu    r8, r9, CD_done               # Done if we reached end address
 84
 85     stb     r0, (r8)                      # Store zero at each byte and
 86     addi    r8, r8, byteSize              #  move on to next byte
 87
 88     br      CD_loop
 89
 90 CD_done:
 91     ENDF
 92
 93 /*
 94 *  plot_pixel
 95 *
 96 *  Description: Writes a pixel to a position on the display.
 97 *               Position is passed in binary with (0, 0) being the top
 98 *               left corner.
 99 *
100 *  Operation:   Gets column number and row number from (x, y) by dividing
101 *               by the width and height of a quadrant (pixPerQH and pixPerQV).
102 *               Uses the quotient to shift a mask to the right value.
103 *               Fixes hardware problem of having the rows offset by two by
104 *               shifting all the rows up except for the top two rows, which
105 *               are shifted down.
106 *               Finally checks color value of pixel and writes it to the
107 *               right quadrant by using an AND or OR operation, depeding on
108 *               the color of the pixel.
109 *
110 *  Arguments:          x - x position of pixel (0 to WIDTH)
111 *                      y - y position of pixel (0 to HEIGHT)
112 *                      p - Color of pixel:
113 *                          PIXEL_BLACK: A black pixel
114 *                          PIXEL_WHITE: A white pixel
```

```
115 *
116 *  Return Value:       None.
117 *
118 *  Local Variables:    None.
119 *
120 *  Shared Variables:   None
121 *
122 *  Global Variables:   None.
123 *
124 *  Input:              None.
125 *
126 *  Output:             Outputs pixel to the display (writes to VRAM).
127 *
128 *  Error Handling:     None.
129 *
130 *  Limitations:        None.
131 *
132 *  Algorithms:         None.
133 *  Data Structures:    None.
134 *
135 *  Registers Changed:  r8 - r15 (these are supposedly caller saved).
136 *
137 *  Revision History:
138 *      06/10/2014    Daniel Andrade    initial revision
139 *      06/15/2014    Daniel Andrade    'fixed' hardware issues
140 *
141 *
142 */
143
144 .global plot_pixel
145 .type plot_pixel,@function
146
147 plot_pixel:
148     BEGINF
149
150 PP_init:
151     movhi   r8, %hi(vramBase)               # Load VRAM base address
152     ori     r8, r8, %lo(vramBase)
153
154 PP_get_column:
155     movi    r9, pixPerQH                    # Get the column number (remainder)
156     divu    r10, r4, r9                     #  and amount to shift by
157     mul     r12, r10, r9                    #  (quotient).
158     sub     r12, r4, r12
159
160 PP_get_row:
161     movi    r9, pixPerQV                    # Get the row number (remainder)
162     divu    r11, r5, r9                     #   and amount to shift by
163     mul     r13, r11, r9                    #   quotient.
164     sub     r13, r5, r13
165
166 PP_fix_hardware_Y:                          # Rows are offset by two, so fix
167     movi    r9, 1                           #  that in software.
168     mov     r15, r13
169     bgt     r13, r9, PP_fix_hardware_shift  # Check if row is greater than one
170                                             # If so, then we need to move
171     movi    r9, 62                          #  to the bottom row.
```

```
172     add     r13, r9, r13
173     br      PP_get_address
174
175 PP_fix_hardware_shift:
176     movi    r9, 2                           # Otherwise, just shift it by two
177     sub     r13, r15, r9                    #  upwards.
178
179 PP_get_address:
180     muli    r13, r13, colsPerRow            # Now that row and col are fixed
181     add     r13, r12, r13                   #  get the address where we want
182     add     r8, r8, r13                     #  to store this pixel.
183
184 PP_shift_value:
185     slli    r9, r10, 1                      # X quadrant is top bit
186     add     r9, r9, r11                     # This is total amount to shift by
187
188     movi    r13, 1                          # Shift this bit to get it to the
189     sll     r13, r13, r9                    #  right quadrant.
190
191 PP_check_color:
192     ldbu    r9, (r8)                        # Get the old value store here.
193
194     movi    r14, PIXEL_WHITE                # Check to see what color
195     bne     r6, r14, PP_plot_black          #  to write.
196     nor     r13, r13, r0                    # If it's white, we need to invert
197     and     r13, r13, r9                    # it and AND it with old value
198
199     br      plotEnd
200
201 PP_plot_black:
202     or      r13, r13, r9                    # Otherwise just OR it with old value
203
204 plotEnd:
205     stb     r13, (r8)                       # And store the new value
206
207     ENDF
208
209
```

```
1 ###############################################################################
2 #                                                                             #
3 #                                  Trigger                                    #
4 #                              Triggering Routines                            #
5 #                                   EE/CS 52                                  #
6 #                                                                             #
7 ###############################################################################
8
9 /*
10  *  Daniel Andrade
11  *  EE/CS 52
12  *  TA: Dan Pipe-Mazo
13  *
14  *  File Description: Constants for the triggering/sampling code.
15  *
16  *
17  *  Revision History:
18  *      06/14/2014    Daniel Andrade        Initial Revision
19  *
20  */
21
22  .equ   clkFreq,          12000000;         # Frequency of ADC clock
23  .equ   sampleSize,            480;         # How many points to take per sample
24  .equ   startSample,             1;         # Constant to write to PIO for starting
25
26  .equ   rateAddress,     0x000c10d0;        # PIO address for sample rate
27  .equ   tSAddress,       0x000c1120;        # PIO address for trigger slope
28  .equ   tLAddress,       0x000c1110;        # PIO address for trigger level
29  .equ   startAddress,    0x000c10b0;        # PIO address for trigger reset
30  .equ   autoAddress,     0x000c1130;        # PIO address for auto setting on
31                                             #  trigger.
32  .equ   rdClkAddress,    0x000c1100;        # PIO address for FIFO read clock
33  .equ   fifoDataAddress, 0x000c10f0;        # PIO address for FIFO data
34  .equ   delayAddress,    0x000c10c0;        # PIO address for trigger delay
35  .equ   fifoIntrAddress, 0x000c10e0;        # PIO address for FIFO interrupts
36
37  .equ   triggerIRQ,              2;         # IRQ value for FIFO interrupts.
38
```

```
1 ##############################################################################
2 #                                                                            #
3 #                                  Trigger                                   #
4 #                              Triggering Routines                           #
5 #                                   EE/CS 52                                 #
6 #                                                                            #
7 ##############################################################################
8
9 /*
10  *  Daniel Andrade
11  *  EE/CS 52
12  *  TA: Dan Pipe-Mazo
13  *
14  *  File Description:  Code used for triggering/sampling from the ADC.
15  *                     Interfaces heavily with FPGA hardware.
16  *
17  *  Table of Contents:
18  *     int trigger_init():                Intalls the handler for FIFO data
19  *                                          and initializes trigger.
20  *     int set_sample_rate(long int rate): Sets sample rate. Returns the
21  *                                          number of sample to be taken.
22  *     set_trigger(int level, int slope):  Sets the level and slope of the
23  *                                          trigger.
24  *     set_delay(long int delay):         Sets the delay on the scope.
25  *     start_sample(int auto):            Start sampling.
26  *     unsigned char* sample_done():      Returns pointer to sample data.
27  *
28  *  Revision History:
29  *      06/14/2014      Daniel Andrade      Initial Revision
30  *      06/15/2014      Daniel Andrade      Some debugging on handler
31  *
32  */
33
34  .include "trigger.m"
35  .include "macros.m"
36  .include "interfac.m"
37  .include "general.m"
38
39 /*
40  *  trigger_init
41  *
42  *  Description:  Interrupt handler installer for the FIFO data interrupt.
43  *
44  *  Operation:    Installs the interrupt handler, unmasks it, and resets the
45  *                trigger.
46  *
47  *  Arguments:          None.
48  *
49  *  Return Value:       None.
50  *
51  *  Local Variables:    None.
52  *
53  *  Shared Variables:   None.
54  *
55  *  Global Variables:   None.
56  *
57  *  Input:              None.
```

```
 58 *
 59 *  Output:              PIO for trigger reset.
 60 *
 61 *  Error Handling:      None.
 62 *
 63 *  Limitations:         None.
 64 *
 65 *  Algorithms:          None.
 66 *  Data Structures:     None.
 67 *
 68 *  Registers Changed:   r8-r9
 69 *
 70 *  Revision History:
 71 *      06/14/2014      Daniel Andrade      initial revision
 72 *
 73 */
 74
 75 .global trigger_init
 76 .type trigger_init,@function
 77
 78 trigger_init:
 79     BEGINF
 80     movi    r8, triggerIRQ
 81
 82 install_handler:
 83     mov     r4, r0                      # Install the first handler
 84     mov     r5, r8                      # Trigger handler has IRQ of triggerIRQ
 85     movia   r6, trigger_intr            # Address to handler function
 86     mov     r7, r0                      # No data to pass
 87     PUSH    r0                          # And flags should always be zero
 88     call    alt_ic_isr_register
 89     POP     r8
 90
 91 unmaskBit:
 92     movia   r8, fifoIntrAddress         # Unmask the interrupt for fifo
 93     movi    r9, 1
 94     stwio   r9, maskRegOffset(r8)
 95
 96 resetInit:
 97     movhi   r8, %hi(startAddress)       # Trigger should start unarmed
 98     ori     r8, r8, %lo(startAddress)
 99     stb     r0, (r8)
100
101     ENDF
102
103 /*
104 *  set_sample_rate
105 *
106 *  Description:  Sets the sample rate (rate at which data is stored in FIFO).
107 *
108 *  Operation:    Divides passed frequency by clock frequency and puts this
109 *                value in a PIO.
110 *
111 *  Arguments:           long int rate - Frequency at which to take samples.
112 *
113 *  Return Value:        Samples to be taken. Always equal to sampleSize.
114 *
```

```
 *  Local Variables:     None.
 *
 *  Shared Variables:    None.
 *
 *  Global Variables:    None.
 *
 *  Input:               None.
 *
 *  Output:              PIO for sample rate.
 *
 *  Error Handling:      None.
 *
 *  Limitations:         None.
 *
 *  Algorithms:          None.
 *  Data Structures:     None.
 *
 *  Registers Changed:   r4      - Return value.
 *                       r8 - r9
 *
 *  Revision History:
 *      06/14/2014      Daniel Andrade      initial revision
 *
 */

.global set_sample_rate
.type set_sample_rate,@function

set_sample_rate:
    BEGINF

    movhi   r8, %hi(clkFreq)            # Divide by clock frequency to get
    ori     r8, r8, %lo(clkFreq)        #  number of clocks.

    divu    r8, r8, r4

    movhi   r9, %hi(rateAddress)
    ori     r9, r9, %lo(rateAddress)

    stwio   r8, (r9)                    # And store this in a PIO.

    movi    r2, sampleSize              # Return number of points (sampleSize)

    ENDF

/*
 *  set_trigger
 *
 *  Description:  Sets the trigger parameters (slope and level).
 *
 *  Operation:    Writes to PIOs to set these in hardware.
 *
 *  Arguments:          int level - Trigger level (from 0 to 127 inclusive)
 *                      int slope - Trigger slope (1 for negative,
 *                                                 0 for positive)
 *
 *  Return Value:       None.
```

```
172 *
173 *  Local Variables:     None.
174 *
175 *  Shared Variables:    None.
176 *
177 *  Global Variables:    None.
178 *
179 *  Input:               None.
180 *
181 *  Output:              PIOs.
182 *
183 *  Error Handling:      None.
184 *
185 *  Limitations:         None.
186 *
187 *  Algorithms:          None.
188 *  Data Structures:     None.
189 *
190 *  Registers Changed:   r8
191 *
192 *  Revision History:
193 *      06/14/2014      Daniel Andrade      initial revision
194 *
195 */
196
197 .global set_trigger
198 .type set_trigger,@function
199
200 set_trigger:
201     BEGINF
202
203     movhi   r8, %hi(tLAddress)
204     ori     r8, r8, %lo(tLAddress)
205
206     stbio   r4, (r8)                    # Store trigger level
207
208     movhi   r8, %hi(tSAddress)
209     ori     r8, r8, %lo(tSAddress)
210
211     stwio   r5, (r8)                    # And store trigger slope
212
213     ENDF
214
215
216 /*
217 *  set_delay
218 *
219 *  Description:  Sets the trigger delay.
220 *
221 *  Operation:    Writes to PIOs.
222 *
223 *  Arguments:           long int delay - Desired delay (in samples)
224 *
225 *  Return Value:        None.
226 *
227 *  Local Variables:     None.
228 *
```

```
229 *  Shared Variables:    None.
230 *
231 *  Global Variables:    None.
232 *
233 *  Input:               None.
234 *
235 *  Output:              PIO.
236 *
237 *  Error Handling:      None.
238 *
239 *  Limitations:         16 bits of precision on delay - maximum delay of
240 *                         2^16 samples.
241 *
242 *  Algorithms:          None.
243 *  Data Structures:     None.
244 *
245 *  Registers Changed:   r8
246 *
247 *  Revision History:
248 *      06/14/2014      Daniel Andrade      initial revision
249 *
250 */
251
252 .global set_delay
253 .type set_delay,@function
254
255 set_delay:
256     BEGINF
257
258     movhi   r8, %hi(delayAddress)
259     ori     r8, r8, %lo(delayAddress)
260
261     stwio   r4, (r8)
262
263     ENDF
264
265
266 /*
267 *  start_sample
268 *
269 *  Description:  Starts sampling (arms the trigger)
270 *
271 *  Operation:    Writes to PIOs - one for triggering type and one for arming
272 *                the trigger.
273 *
274 *  Arguments:           int auto - auto or normal trigger
275 *
276 *  Return Value:        None.
277 *
278 *  Local Variables:     None.
279 *
280 *  Shared Variables:    None.
281 *
282 *  Global Variables:    None.
283 *
284 *  Input:               None.
285 *
```

```
286 *  Output:              PIOs.
287 *
288 *  Error Handling:      None.
289 *
290 *  Limitations:         None.
291 *
292 *  Algorithms:          None.
293 *  Data Structures:     None.
294 *
295 *  Registers Changed:   r8 - r9
296 *
297 *  Revision History:
298 *      06/14/2014      Daniel Andrade      initial revision
299 *
300 */
301
302 .global start_sample
303 .type start_sample,@function
304
305 start_sample:
306     BEGINF
307
308     movhi   r8, %hi(autoAddress)
309     ori     r8, r8, %lo(autoAddress)
310     stbio   r4, (r8)                    # Set trigger type (auto or normal)
311
312     movhi   r8, %hi(startAddress)
313     ori     r8, r8, %lo(startAddress)
314
315     movi    r9, startSample             # Arm the trigger
316     stbio   r9, (r8)
317
318     ENDF
319
320
321 /*
322 *  sample_done
323 *
324 *  Description:  Polling function to see if sampling is done.
325 *
326 *  Operation:    Checks to see if sampleSize samples have been sampled, and,
327 *                if so, it returns a pointer to the buffer with the samples.
328 *
329 *  Arguments:           int auto - auto or normal trigger
330 *
331 *  Return Value:        NULL if sampling not done, otherwise pointer to
332 *                         data buffer with samples.
333 *
334 *  Local Variables:     None.
335 *
336 *  Shared Variables:    sample_buffer - Buffer of collected samples from FIFO
337 *                                       (read).
338 *                       sample_flag   - Marks whether enough samples have been
339 *                                       collected (read).
340 *
341 *  Global Variables:    None.
342 *
```

```
343 *  Input:               None.
344 *
345 *  Output:              None.
346 *
347 *  Error Handling:      None.
348 *
349 *  Limitations:         None.
350 *
351 *  Algorithms:          None.
352 *  Data Structures:     None.
353 *
354 *  Registers Changed:   r8 - r9
355 *
356 *  Revision History:
357 *      06/14/2014      Daniel Andrade      initial revision
358 *
359 */
360
361 .global sample_done
362 .type sample_done,@function
363
364 sample_done:
365     BEGINF
366
367     movia   r8, sample_flag
368     ldbuio  r8, (r8)
369     movi    r9, FALSE
370
371     beq     r8, r9, sample_not_done        # Check to see if actually done
372
373 sample_really_done:
374     movia   r2, sample_buffer              # If so, then return pointer
375
376     movia   r8, sample_flag                # Reset the sample done flag
377     movi    r9, FALSE
378     stb     r9, (r8)
379
380     br      sample_done_done
381
382 sample_not_done:
383     mov     r2, r0                         # Otherwise return NULL
384
385 sample_done_done:
386     ENDF
387
388
389 /*
390 *  trigger_intr
391 *
392 *  Description:  Interrupt handler for the sampling hardware.
393 *
394 *  Operation:    Disarms the trigger, then loops through every element of
395 *                the FIFO and stores it in a buffer. This is done by setting
396 *                the read clock PIO high and low repeatedly.
397 *
398 *  Arguments:           None.
399 *
```

```
400 *  Return Value:        None.
401 *
402 *  Local Variables:     None.
403 *
404 *  Shared Variables:    sample_buffer - Buffer of collected samples from FIFO
405 *                                       (write).
406 *                       sample_flag   - Marks whether enough samples have been
407 *                                       collected (write).
408 *
409 *  Global Variables:    None.
410 *
411 *  Input:               None.
412 *
413 *  Output:              PIO to disarm trigger.
414 *
415 *  Error Handling:      None.
416 *
417 *  Limitations:         None.
418 *
419 *  Algorithms:          None.
420 *  Data Structures:     None.
421 *
422 *  Registers Changed:   None (registers were pushed).
423 *
424 *  Revision History:
425 *      06/14/2014      Daniel Andrade      initial revision
426 *      06/15/2014      Daniel Andrade      extensive debugging, disarmed trigger.
427 *
428 */
429
430 .global trigger_intr
431 .type trigger_intr,@function
432
433 trigger_intr:
434     BEGINF
435
436     PUSH    r16
437     PUSH    r17
438     PUSH    r18
439     PUSH    r19
440
441 TI_read_init:
442     movhi   r8, %hi(startAddress)
443     ori     r8, r8, %lo(startAddress)
444
445     stbio   r0, (r8)                    # Disarm trigger
446
447     movia   r18, sample_buffer          # Load buffer address
448     movi    r19, sampleSize
449     add     r19, r19, r18               # Stopping point
450
451 TI_read_loop:
452     movhi   r16, %hi(rdClkAddress)      # Set the read clock high to read
453     ori     r16, r16, %lo(rdClkAddress) #  from FIFO.
454
455     movi    r17, 1
456     stbio   r17, (r16)
```

```
457
458     movhi   r16, %hi(fifoDataAddress)     # Load FIFO address
459     ori     r16, r16, %lo(fifoDataAddress)
460
461     ldbuio  r17, (r16)                    # Load data from FIFO
462
463     movhi   r16, %hi(rdClkAddress)
464     ori     r16, r16, %lo(rdClkAddress)
465
466     stbio   r0, (r16)                     # Done reading, set read clock low
467
468     slli    r17, r17, 1                   # Shift because we threw away bottom
469                                           #  bit.
470     stb     r17, (r18)                    # Store FIFO data in buffer
471
472     addi    r18, r18, 1                   # Go to next element in buffer
473
474     beq     r18, r19, TI_end              # We collected all samples
475     br      TI_read_loop                  # Otherwise read next value
476
477 TI_end:
478     movia   r16, sample_flag              # Sample is done
479     movi    r17, TRUE
480     stb     r17, (r16)
481
482     POP     r19
483     POP     r18
484     POP     r17
485     POP     r16
486
487     ENDF
488
489 .section .data
490 sample_buffer:      .space  sampleSize          # Buffer for sample data
491 sample_flag:        .byte   0                   # Whether sample is done
492
493
```