.

FPGA Oscilloscope Manual

EE 52 Project Sequence (SOPC Scope Section)

Albert Gural

July 5, 2014

California Institute of Technology

Contents

Chapter 1

User Manual

1.1 System Overview

The FPGA Oscilloscope ® is designed to mimic some of the more used functions of a traditional digital oscilloscope, but in a much smaller package. It has two analog inputs, an 8-bit logic analyzer, hardware buttons and rotary encoders, and a 480x272 RGB display.

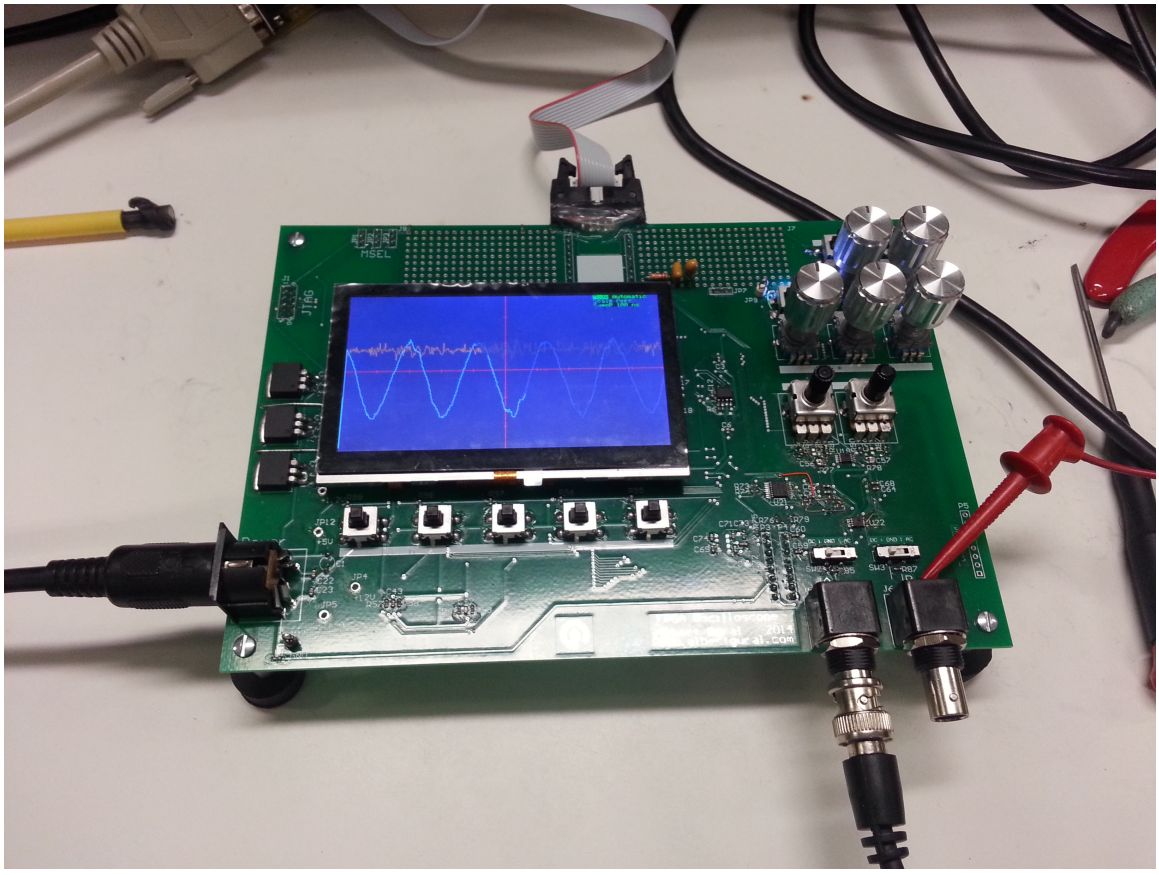

Figure 1.1: FPGA Oscilloscope

The system includes the main oscilloscope board (seen above), the power adapter (which supplies $5V$ and $\pm 12V$), a USB JTAG programmer, and two $1x/10x$ analog probes.

1.2 Setup

This version of the oscilloscope does not come pre-configured with program memory in flash, although the FPGA configuration information is programmed onto a serial flash chip. The set of operations to get the system up and running is thus:

1. Install Quartus web edition (free). Download the program files from `https://github.com/agural/FPGA-Oscilloscope`.
2. Plug in power to the oscilloscope. Connect the USB JTAG connector between the computer and the oscilloscope.
3. Open up the "NIOS II Software Build Tools for Eclipse".
4. Go to `File > New > NIOS II Application and BSP from Template`.
5. Select the SOPC file from `<FPGA Download Directory>/osc`. Name the project whatever you want, and select "Blank Template".
6. Copy code from `<FPGA Download Directory>/osc/software/osc` to your Eclipse project.
7. Compile and download the code to the oscilloscope.

At this point, the oscilloscope should be running. Try connecting a $0.5V$ amplitude sine wave at $100kHz$ to the scope.

1.3 Interface

User control is broken into two main input methods that are mostly redundant with each other (select whichever you prefer): oscilloscope mimic controls and on-screen menu controls.

1.3.1 On-screen Menu

This menu can be used to control all software-controllable aspects of the oscilloscope and is controlled via the five hardware buttons located below the display. The menu appears on the top-right of the display. The left-most button is the menu button. Pressing this toggles the menu on and off. From left to right, the remaining buttons are "Up", "Down", "Left", and "Right". Press "Up" or "Down" to select a menu item (the selected item will be highlighted). For a given menu item, you can change its value by pressing "Left" or "Right". For example, hitting the "Down" button to get delay, then hitting the "Right" button will increase the (positive) delay by a single sample. Pressing and holding the buttons will cause them to auto-repeat.

Here are the menu items and values you can choose from:

- Mode
  - Auto triggering - triggers if it can, or if not, displays the sample after a given timeout.
  - Normal triggering - updates the display only on triggers.
  - One-shot - triggers only once and holds that on the display.
- Scale
  - Scale Axes - Displays just the x-y axes with tick marks at the major divisions.
  - Scale Grid - Displays a grid of the major divisions.
  - None - Displays only the traces (and possibly the menu).
- Sweep - Sets the time per sample for the oscilloscope. Cycles through the available sweep rates (see 1.4 for a list).

- Trigger - Re-arms the one-shot trigger when “Left” or “Right” is pressed.
- Level - Sets the trigger level. It is set up to display a voltage, but this is inaccurate.
- Slope - Either “Left” or “Right” will toggle positive to negative slope.
- Delay - Sets the trigger delay. It can be changed from -480 to $50,000$ samples, in the appropriate time units.

1.3.2 Oscilloscope-mimic Controls

There are two latching pushbutton switches in the top-right of the oscilloscope, two rotary encoders directly below them, and three rotary encoders below these. Below that is two potentiometers, and finally two 3-position slide switches. Since the operation of these is the same as for the menu, the image below should be sufficient.

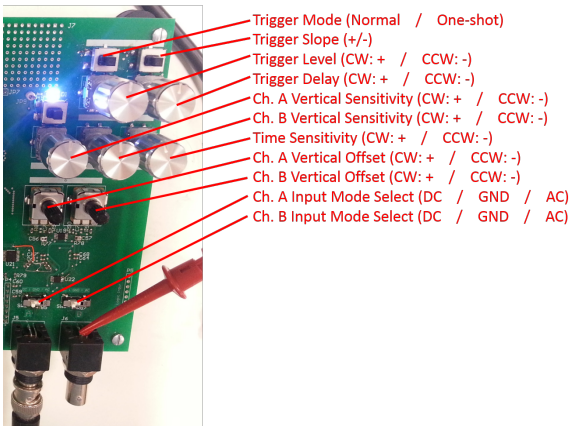

Figure 1.2: Hardware buttons

The potentiometers and slide switches are the only items not do-able by the menu method (they are part of the analog frontend). Vertical sensitivity has not been completed in terms of software, although all the supporting hardware is there. All other functions follow exactly as in the case of the menu.

1.4 Technical Specifications

1.4.1 Physical Specifications

- Processor: NIOS II softcore processor at $36MHz$ on an Altera Cyclone III EP3C25 (240 pin QFP) chip.
- RAM: 64KB x 8bit accessible SRAM.
- ROM: 64KB x 8bit accessible Flash.
- Display: 480x272 RGB display with 24-bit color (only 15-bit hardware-accessible). Also includes a 10-point capacitive touch pad overlay (hardware accessible, but not used in firm/software).
- Buttons: (5) screen menu buttons (momentary), (2) trigger option buttons (latching), (5) rotary encoders, (1) reset button (momentary).
- Ports: $5V$ and $\pm 12V$ power supply input (DIN-5); 10-pin JTAG connector; (2) analog BNC probe inputs; (8) logic analyzer pins (requires user to attach female header wire to it).

1.4.2 Oscilloscope-specific Specifications

- Sample Rates:
  - 5*, 10*, 20*, 50, 100, 200, 500 ns
  - 1, 2, 5, 10, 20, 50, 100, 200, 500 μs
  - 1, 2, 5, 10, 20 ms
- Vertical Sensitivities:
  - $200mV$ (software setup)
  - $25mV$, $50mV$, $100mV$, $200mV$, $400mV$, $800mV$, $1.6V$ (hardware capable)
- Sample Resolution:
  - Channel A/B: 8 bits
  - Logic Analyzer: 1 bit per channel
- Input Voltage Range:
  - $-1V$ to $1V$ (software setup)
  - $-12V$ to $12V$ (hardware capable)
  - $-100V$ to $100V$ (safe input level)
- Trigger Level Resolution:
  - Channel A/B: 7 bits
  - Logic Analyzer: 1 bit per channel
- Trigger Slope: Positive or Negative
- Trigger Delay: -480 samples to $50,000$ samples

*The ADC is only set to sample at the clock frequency, $36MHz$. So at these sample rates, the ADC will be sampled multiple times at the same value, producing a step-like curve.

Chapter 2

Hardware

2.1 Block Diagrams

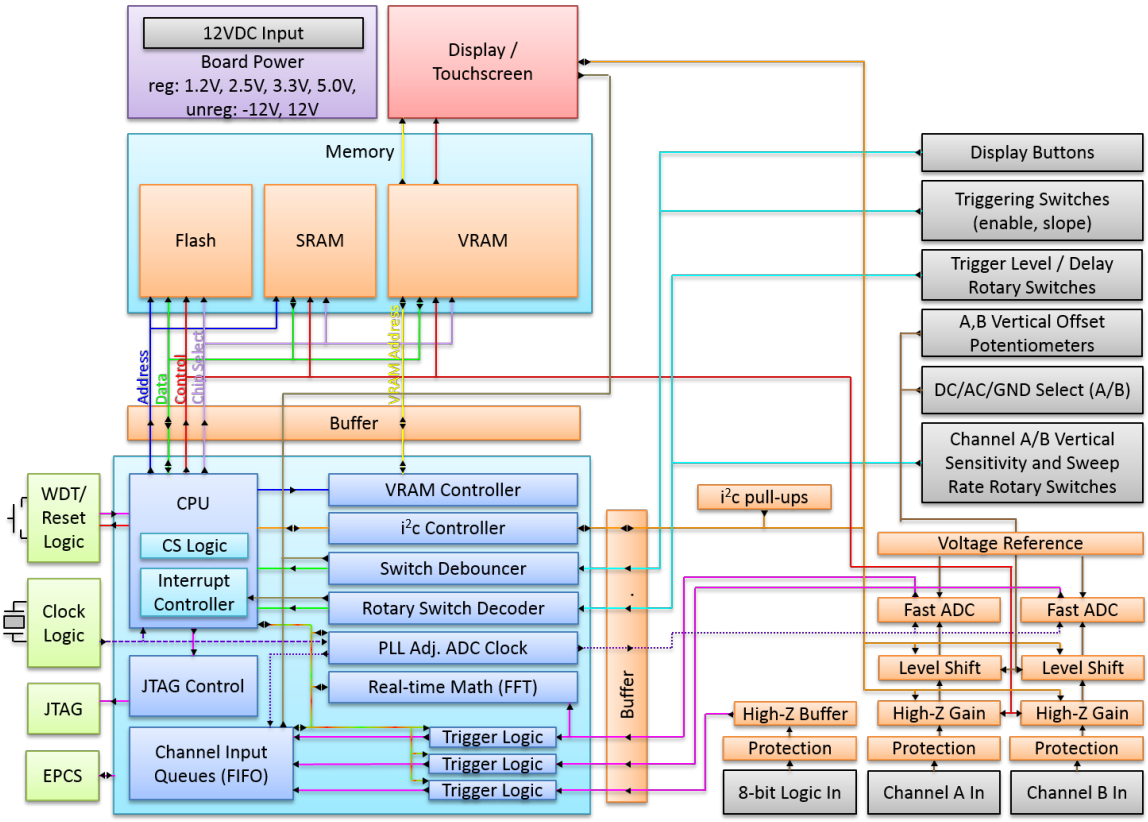

Figure 2.1: Full block diagram

The hardware consists of four main modules: the FPGA (and supporting hardware), the analog input, the memory devices and display, and the switches. On the block diagram above, the FPGA components are in the bottom left with the FPGA itself represented by the large light blue rectangle, the analog is in the bottom right, the memory is in the top left, and the switches are in the top right.

The FPGA itself consists of several separable components, although these will be covered more in the Firmware section (Chapter 3). As a broad overview, the FPGA features a soft-core CPU, a controller for the VRAM and display, logic for handling switch and rotary encoder input, and the entire ADC input and

triggering section. As seen in the block diagram, all of these major components have inputs and outputs to the other major hardware components of the board; they are also all buffered to protect the FPGA.

The analog section consists of two analog channels (A and B) and an 8-bit logic analyzer. While the logic analyzer feeds directly to the FPGA buffer inputs, the two analog channels are fed through an analog front-end before being converted by the ADC. The analog front-end supports a large range of input voltages due to the input protection and voltage scaling op amp circuits. The front-end also contains a few user-controls: the voltage can be shifted using the voltage shift potentiometers; the input mode can be selected between DC, AC, and GND. While the gain is also variable as a function of the vertical sensitivity rotary encoders, the control is indirect (through the FPGA/CPU) as opposed to being a direct part of the analog circuitry.

The memory consists of three separate components: the flash, the SRAM, and the VRAM (plus display). These are connected to the FPGA's soft core CPU as you might expect: address and data buses, as well as chip select and read/write lines. One thing to note is that the VRAM has a separate address bus (9 bits), whereas the flash and SRAM share a 16-bit address bus. The flash memory serves as a non-volatile memory for storing the soft core CPU's operating system. The SRAM serves as external storage for the CPU. The VRAM is memory for the display, discussed in more detail below.

Finally, the user interface is controlled by a number of switches and rotary encoders / potentiometers. Some of these directly control the analog circuitry (potentiometers and mode select slide switches), while others indirectly control the board through FPGA logic (pushbutton switches and rotary encoders). Input from these switches is debounced or interpreted (for rotary encoders), then that simplified information is used to generate an interrupt for the CPU. All debouncing and rotary encoder decoding is done in the FPGA logic (not in the CPU's program).

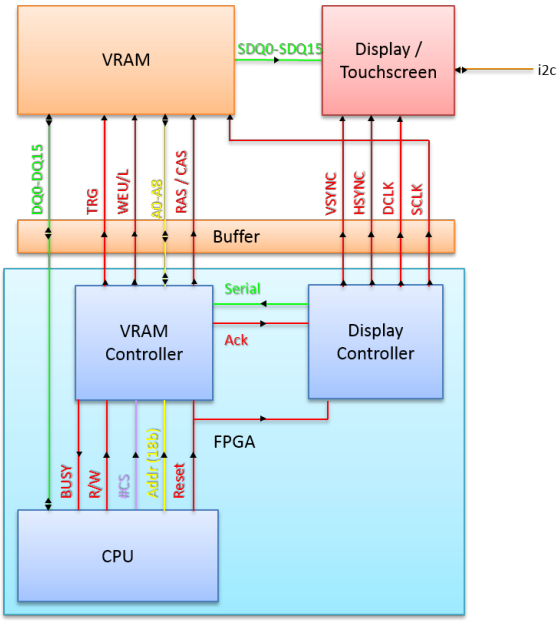

Figure 2.2: Display block diagram

This is a closer look at the VRAM / Display section of the hardware. The large containing blue block represents components internal to the FPGA, and the other blocks represent external components. As usual, there's a buffer to protect the FPGA from damage.

To read/write data to the VRAM (for eventual display by the display), first the CPU tries to write to the VRAM address of interest (18 bits), offset by the VRAM's location in the address map. This actually sends a signal to the VRAM controller (in FPGA logic) that splits the address into the row and column (9 bits each). The VRAM controller sends out the proper signals to the external VRAM chip, allowing the CPU to read or write data. Thus, the VRAM controller and VRAM together form, what looks like to the CPU, a single standard memory component.

With the VRAM loaded with display data, the display should be updated to show what's in VRAM.

This operation is done completely separate from the CPU (all in FPGA logic) with the display controller. The display is a parallel input device which takes three clocks (VSYNC, HSYNC, DCLK), which will be discussed in more detail later. At each of the dot clocks, the parallel input takes a color for what to display at the current dot. Thus, the VRAM must output (SDQ0-15) to the display each pixel as requested - the VRAM is clocked by (SC).

The combination of CPU reads/writes to VRAM and VRAM serial transfers to display completes the operation of the display subcomponent of the hardware.

2.2 FPGA

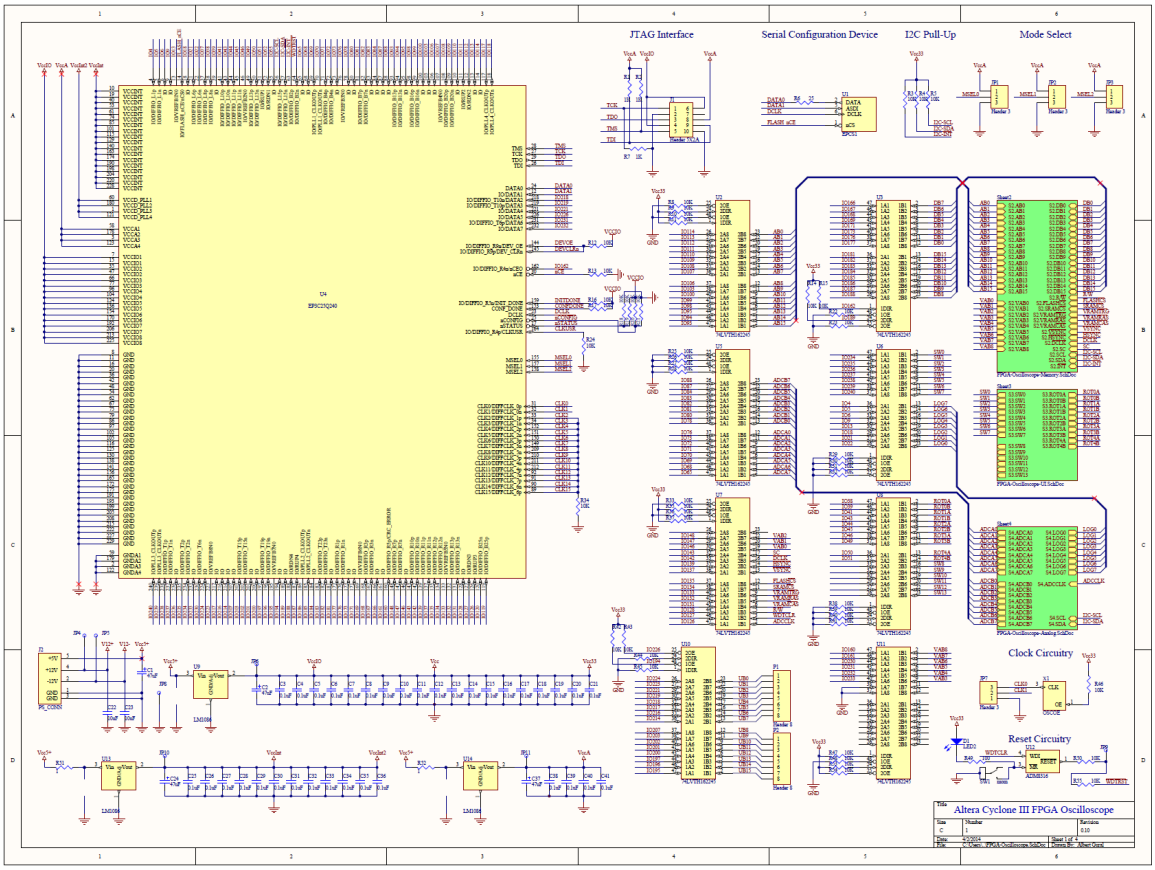

Figure 2.3: FPGA and supporing components schematic diagram

The FPGA is a Quartus EP3C25Q240 Cyclone III FPGA (240-pin quad flat pack) [U4]. Besides various configuration pins, it contains a large number of general purpose I/O. These are connected through buffers [U2,3,5,6,7,8,10,11] to external circuit components.

Power for the entire board is provided by [J2] and contains dual $\pm 12V$ power, as well as a higher current $5V$ line. The FPGA operates on three voltages. The core is powered at $1.2V$ with [U13], the internal analog circuitry is powered at $2.5V$ with [U14], and the I/O banks are powered at $3.3V$ with [U9].

2.3 FPGA Support

There are a few other supporting components specifically for the FPGA. [J1] is a JTAG header that allows configuring the FPGA logic, as well as downloading code for the soft core CPU. [U1] is a serial configuration chip (EPCS16 with 16MB of space) which allows the FPGA to be re-configured after power is lost, without

reprogramming the device through JTAG. [JP1-3] provide mode select capabilities. The board is typically kept in Fast Standard Mode with $3.3V$. [X1] provides a $36MHz$ clock signal. [U12] is a watchdog reset chip that can be used to automatically reset the FPGA (or just the CPU) in the case that the CPU stops operating or reaches an infinite loop.

2.4 Analog Frontend

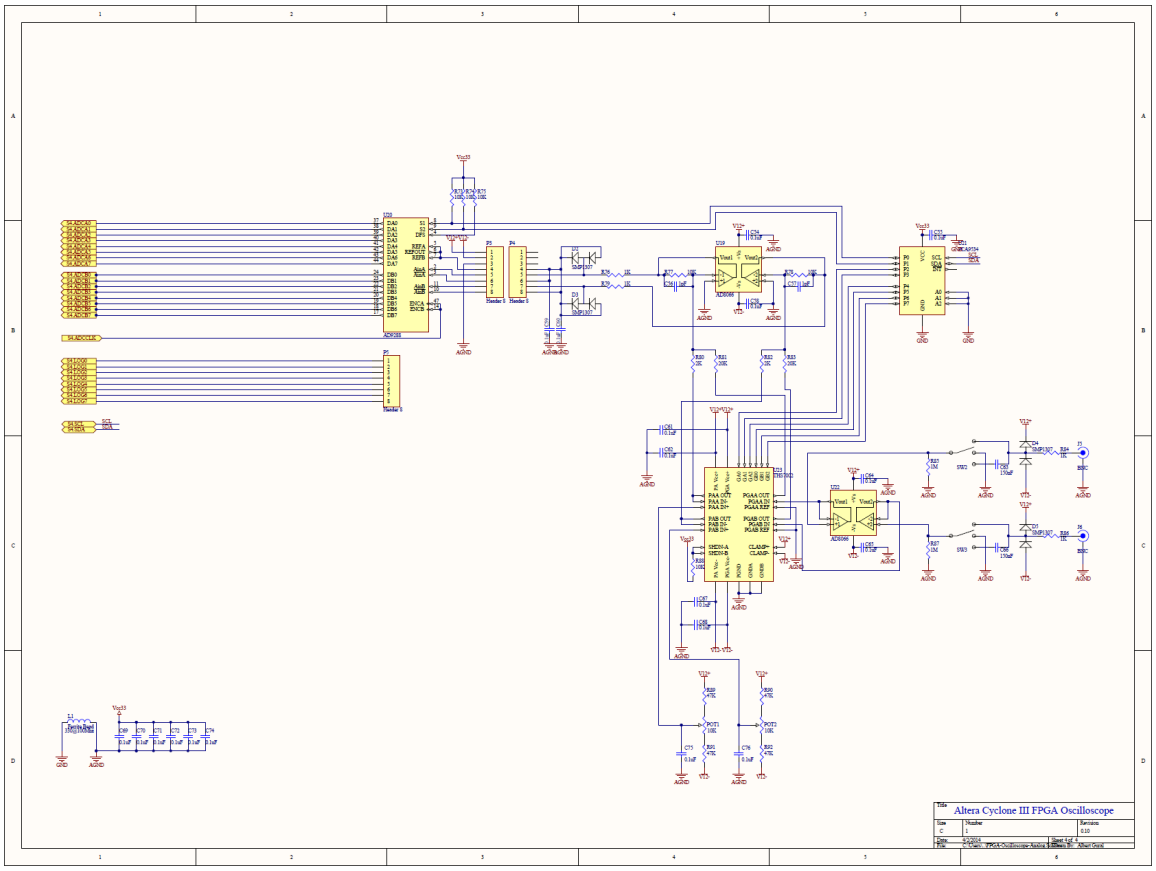

Figure 2.4: Analog section schematic diagram

The analog section contains two channels (A and B) for dual analog input capabilities. Starting from the BNC connector inputs, we can see the full path the signal takes before reaching the ADC to be collected digitally by the FPGA.

BNC connectors [J5-6] take analog input from a standard oscilloscope probe (or a 10x probe). Diode packages [D4-5] provide protection from excessive voltages. [SW2-3] allows for selection of the input mode (between DC/AC/GND). From there, the signals are buffered with [U22], then fed into a programmable gain amplifier (selectable -20dB to 20dB gain) [U23]. [U21] is an extended i2c controllable I/O port which allows for selecting the desired gain on the PGA.

From the PGA, the signal passes through another high-Z buffer [U19], which also provides level shifting (the shift level is provided by potentiometers [POT1-2], which are analog buffered by unused preamplifiers in the PGA [U23]). Finally, the output is again protected by safety diodes before reaching the ADC. Because the ADC has a voltage input swing of $\pm 0.512V$, the $0.7V_F$ diodes [D2-3] can be connected as shown to provide decent protection. They essentially limit the ADC input to within $0.7V$ of the reference voltage.

[U20] is the ADC, which provides 8 bits of data for each of the A and B channels. [P5] is a header for providing 8 bits of logic-analyzer input.

2.5 Analog Frontend Transfer Properties

Effort was put forth to make sure the analog front-end (at least at near-unity gain) had decent transfer function characteristics. Ideally, at least a $5MHz$ bandwidth and no ringing. There were issues, initially, with ringing, as seen in the left of the image below. However, fine-tuning [C56] and [C57] turns out to allow for the results seen in the middle and right.

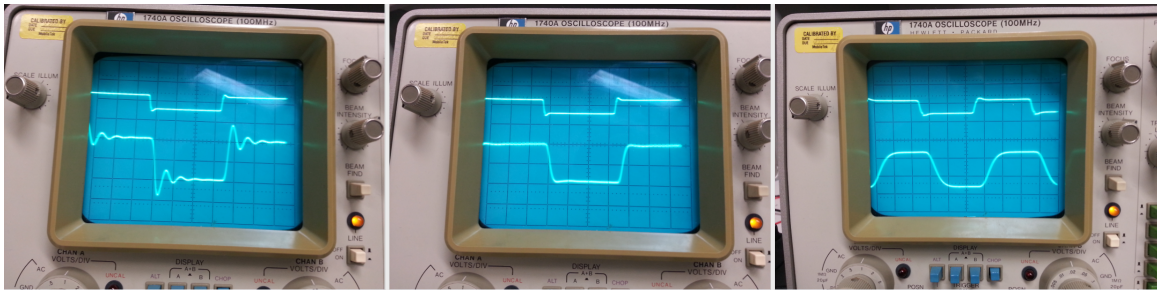

Figure 2.5: Analog frontend step response (input top; output bottom). The left is an underdamped response [C56] $= 0pF$, the middle is well-damped [C56] $= 2pF$, and the right is overdamped [C56] $= 5pF$.

With the $2pF$ values chosen, measurements of the gain and phase were made at a wide range of frequencies. From the step response, we can tell that the circuit has a nice transfer function, but this allows for a more quantitative characterization. Here are the results (DC input mode, -6dB low/mid-band gain):

Table 2.1: Transfer gain and phase at various frequencies

Frequency	Gain	Phase
$0.1Hz$	$-5.7dB$	0^o
$0.2Hz$	$-5.7dB$	0^o
$0.5Hz$	$-5.7dB$	0^o
$1.0Hz$	$-5.7dB$	0^o
$\vdots$	$\vdots$	$\vdots$
$5kHz$	$-5.7dB$	0^o
$10kHz$	$-5.7dB$	0^o
$20kHz$	$-5.7dB$	0^o
$50kHz$	$-5.4dB$	0^o
$100kHz$	$-5.4dB$	0^o
$200kHz$	$-5.4dB$	-7^o
$500kHz$	$-5.4dB$	-14^o
$1MHz$	$-5.2dB$	-29^o
$2MHz$	$-5.4dB$	-54^o
$3MHz$	$-6.0dB$	-84^o
$4MHz$	$-6.7dB$	-112^o
$5MHz$	$-8.0dB$	-140^o
$6MHz$	$-9.4dB$	-167^o
$7MHz$	$-11dB$	-189^o
$8MHz$	$-13dB$	-216^o
$9MHz$	$-15dB$	-235^o
$10MHz$	$-18dB$	-252^o

This gives the following Bode plot:

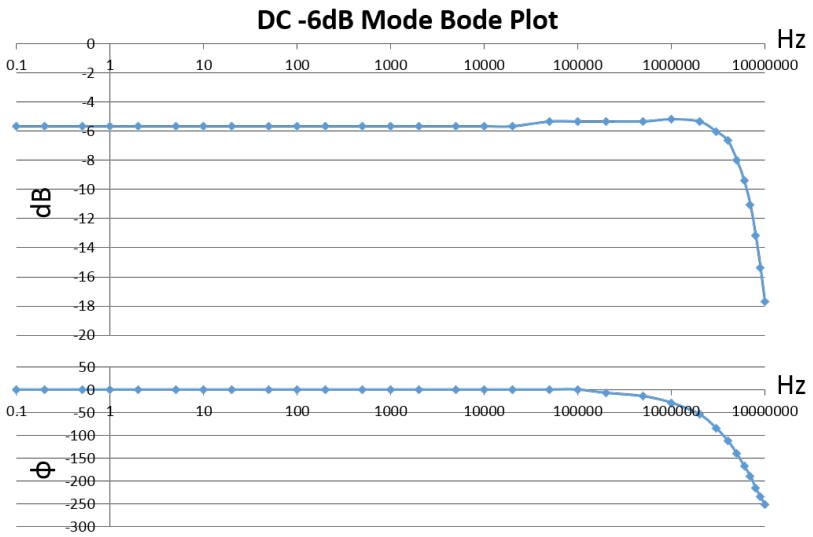

Figure 2.6: Analog input raw Bode plot

This gain and phase function is pretty good. Here's some additional analysis:

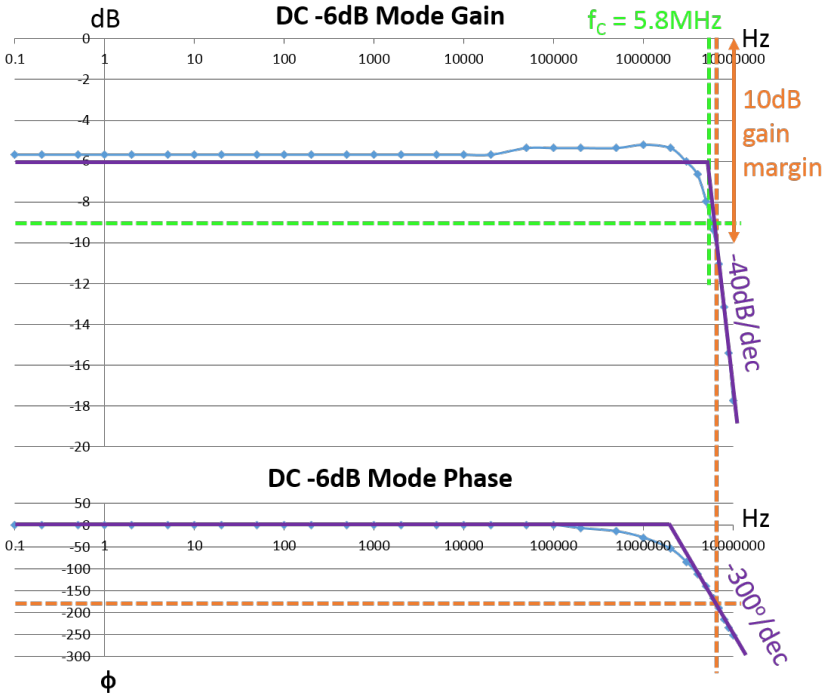

Figure 2.7: Analog input Bode plot with annotations

As we can see, we achieve the desired $5MHz$ bandwidth with a corner frequency at $5.8MHz$.

2.6 ADC and Timing

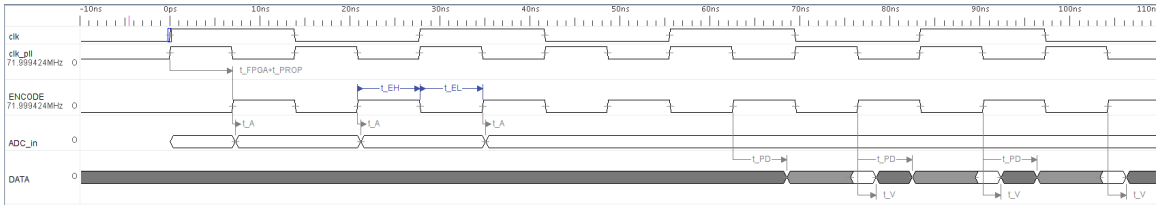

Figure 2.8: ADC timing

ADC timing is fairly straightforward. The FPGA provides a steady ADC clock (limited to a max of $80MHz$), which the ADC then uses to synchronously sample analog input and output the digital value. In the configuration shown, ADC channel A output is synchronized with the clock and channel B output is 180^o offset from that.

Because of the simplicity of timing and the fact that we don't care about small amounts of sample propagation delay, we can simply clock the FPGA sample collection synchronized with the ADC clock to get good data.

2.7 Memory and Display

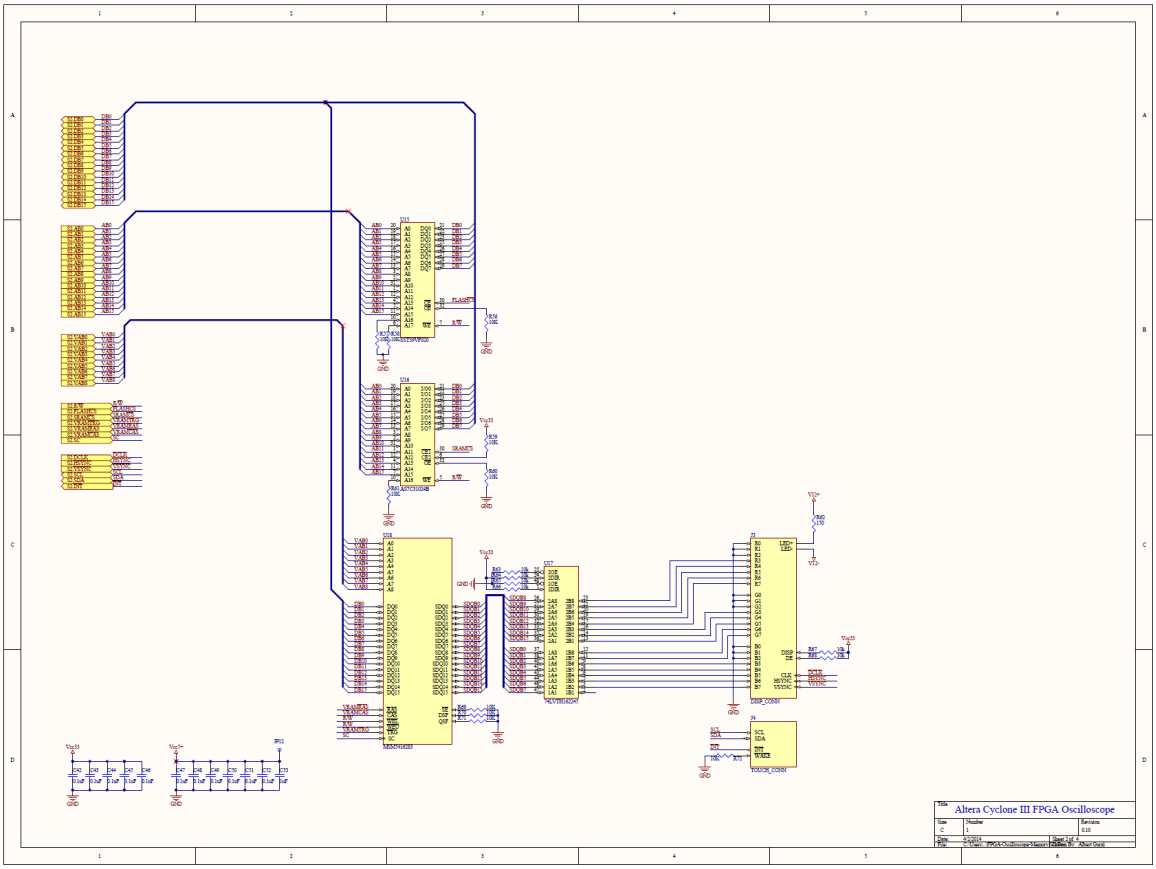

Figure 2.9: Memory schematic diagram

As discussed earlier, flash [U15] and SRAM [U16] are connected in fairly standard configurations. They share a data bus, address bus, and read/write line. Each has a separate chip select input. Both always have outputs enabled since we don't care about saving power.

While VRAM [U18] shares the same data bus and read/write line as flash and SRAM, it has its own 9-bit address line and two separate row/column chip selects. It also has a serial clock pin which outputs the next half-word of data to the SDQ line for the display (it should be clocked whenever we want to send a new pixel to the display). The VRAM operates at $5V$ while the display operates at $3.3V$, so [U17] provides necessary voltage level translation.

The display itself is a 480×272 24-bit RGB display (connector [J3]) with a capacitive touch screen (and a controller that can be communicated with via i2c (connector [J4])). Because VRAM is only 16-bits, the system is only set up to control 5 bits of each color component (15-bit color control). The display is LED backlit (takes $19.2V$), so power is provided from the dual $\pm 12V$ line.

2.8 SRAM and Timing

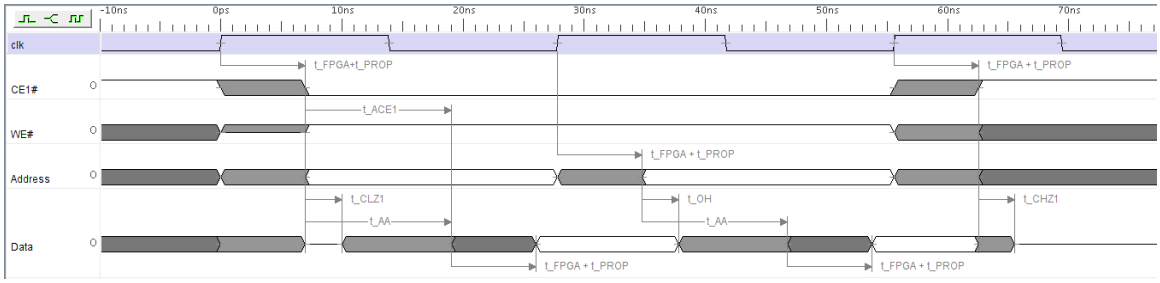

Figure 2.10: SRAM read timing

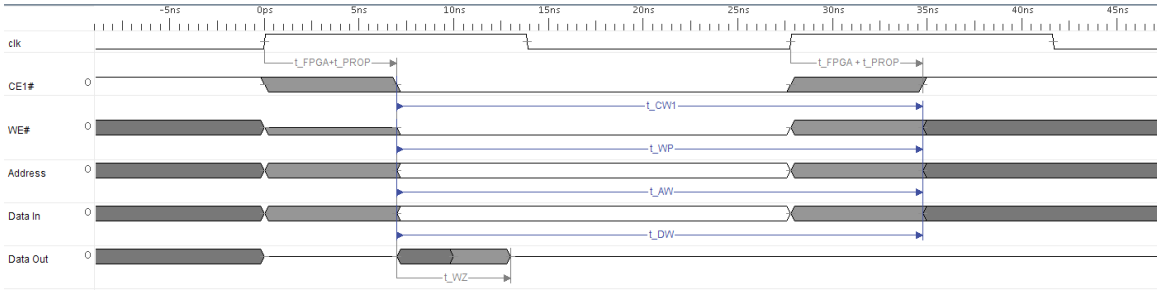

Figure 2.11: SRAM write timing

SRAM timing is shown above. Since the FPGA is clocked with a $36MHz$ source ($27ns$ period), we can see that we can satisfy timing requirements if we have one clock cycle latencies for reading and writing, a one clock setup time, and a zero clock hold time.

2.9 Flash and Timing

Flash timing is shown above. Since the FPGA is clocked with a $36MHz$ source ($27ns$ period), we can see that we can satisfy timing requirements if we have three clock cycle latencies for reading and no latencies for setup and hold times.

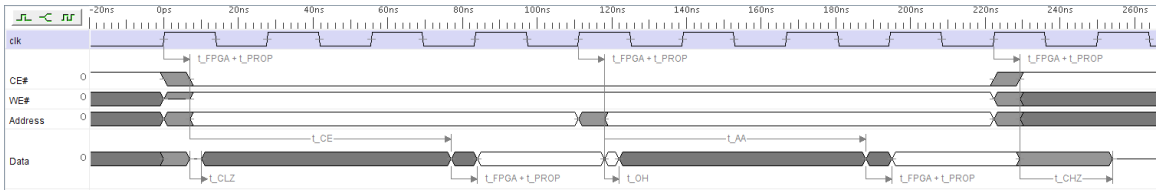

Figure 2.12: Flash read timing

2.10 VRAM and Timing

Since VRAM is controlled by the VRAM controller, timing is important to understand what the VRAM controller must be capable of providing. The CPU, on the other hand, should interact with the VRAM controller using a busy signal. That is, after requesting a read or write, it will wait until the busy signal indicates the data can be read (if reading) or de-asserted (if writing).

The VRAM has four important operational modes. The first two, read and write, provide their obvious functionality. However, because VRAM is a DRAM, it requires period refreshes (refresh cycle). Finally, because it's a video RAM with one port for the CPU to read/write data and one port for the display to receive data, it also requires a serial row transfer cycle. During this cycle, data from a selected row is transferred to a special part of memory so that on serial clocks, the SDQ line fills with data from sequential memory cells in the selected row of memory.

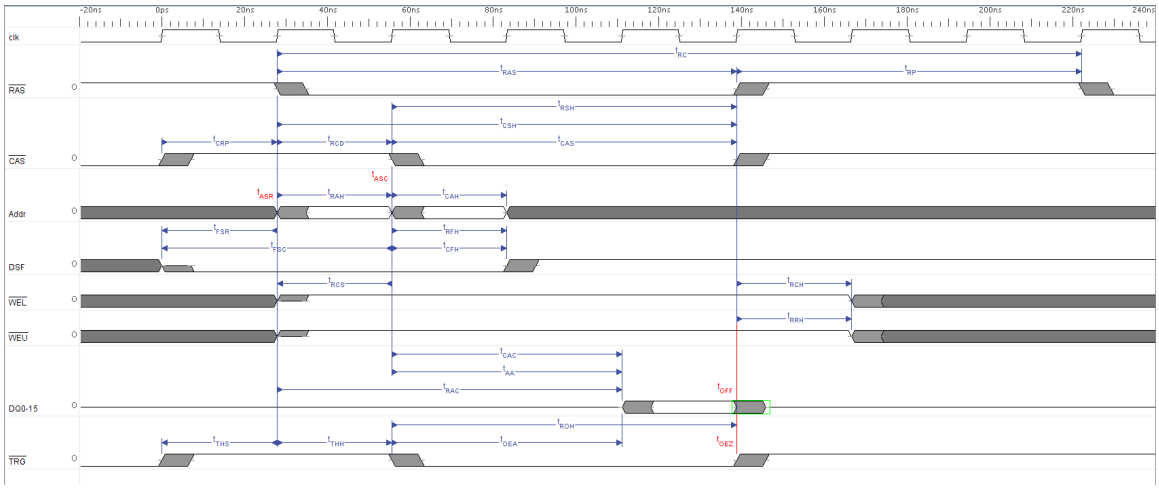

Figure 2.13: VRAM read timing

There are a fair number of signals at play, but the important ones for reading and writing are the address and data buses, the RAS/CAS signals, and obviously the read/write (WE) line.

A refresh cycle is typically very simple. The timing shown above is for a refresh cycle specific to a particular row. There's also a global refresh that refreshes the entire chip very quickly.

The serial row transfer is also fairly simple. Simply provide the row of interest (the row that you want the display to update), and then provide the serial clocks to sequentially output data from that row.

With all of these timings, a state machine can be produced in FPGA logic that provides the necessary signals and delays to trigger the various VRAM modes. See the VRAM state machine VHDL code for more details on specific cycle delays for each of the modes.

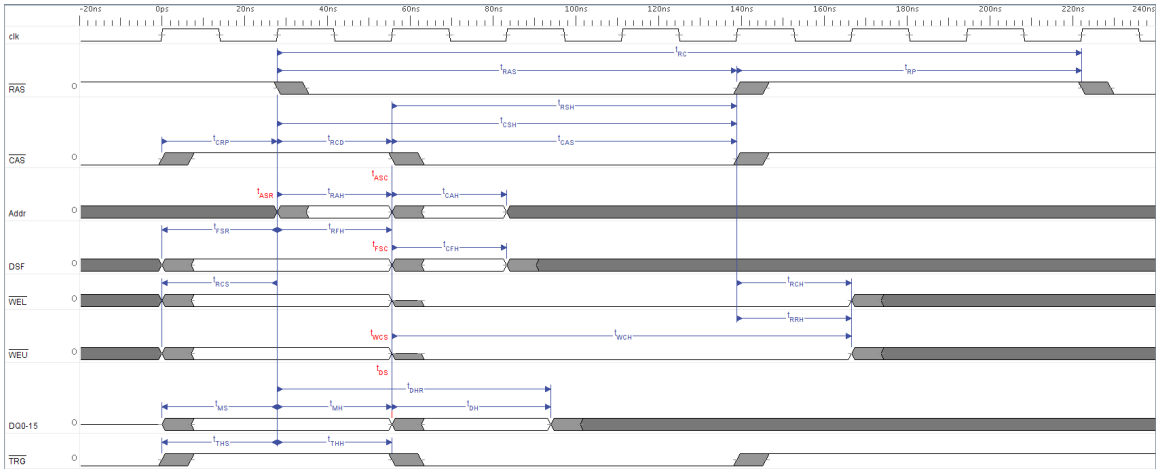

Figure 2.14: VRAM early write timing

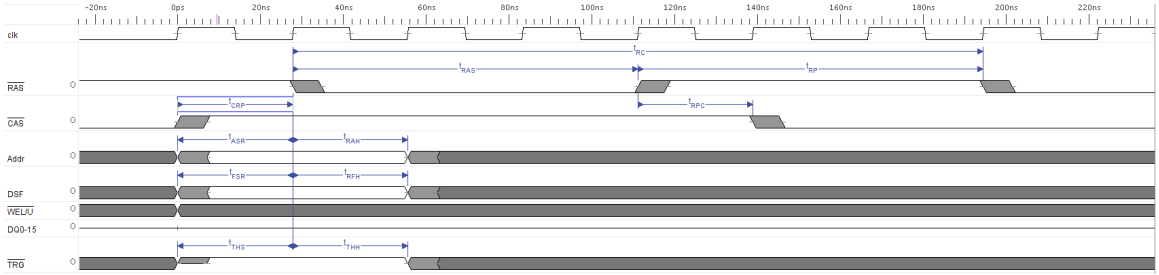

Figure 2.15: VRAM refresh timing

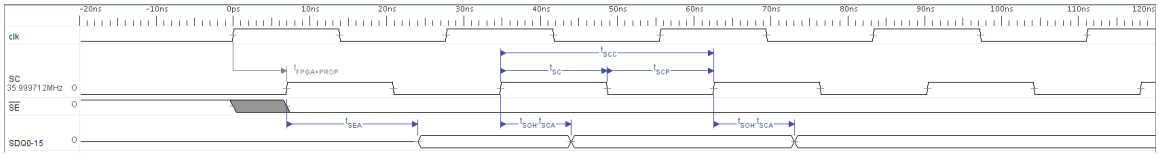

Figure 2.16: VRAM serial row transfer timing

2.11 Display and Timing

The display timing is shown below. The most basic timing is a steady dot clock (DCLK) that is the base pulse of all of the other pulses. During certain portions of the dot clock, each pulse sequences through a row of the display's pixels and any data on the parallel input is read as an RGB value for the display.

For each row to be displayed, there are front and back porches of a few dot clocks, during which nothing is changed on the display (this is shown on the bottom timing diagram). Each row is clocked by the HSYNC pulse. Many of these pulses allow for writing multiple rows of the display (as seen in the middle timing diagram).

Finally, when a screen's worth of rows are covered, the VSYNC pulses (as seen in the top timing diagram). Just as there were front and back porches for the dot clock with respect to a single HSYNC pulse, likewise there are front and back porches for the HSYNC with respect to a single VSYNC pulse. All of these front and back porches together forms an invisible frame around the true image during which the dot clock is active, but no new pixels are drawn to the screen.

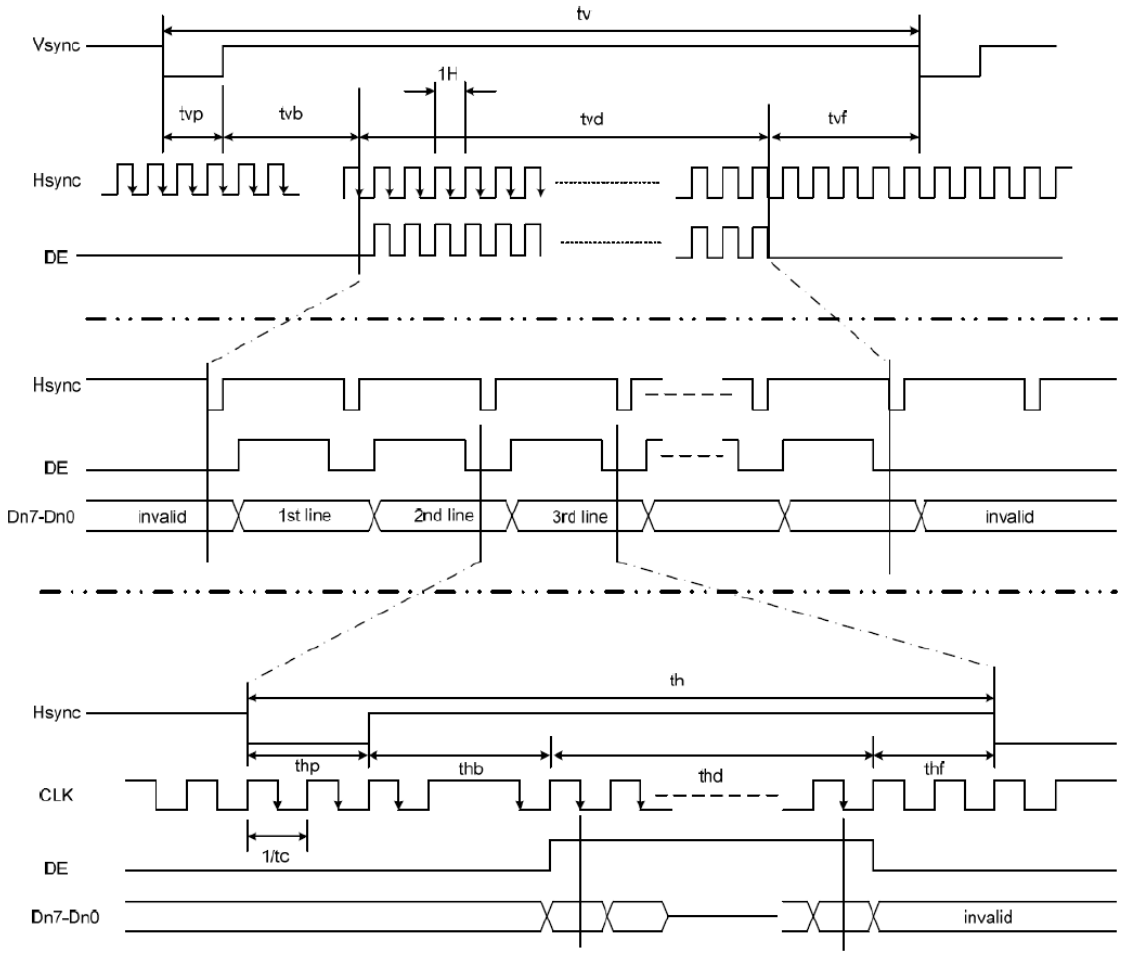

Figure 2.17: Display timing

Parameter	Symbol	Spec.			Unit
		Min.	Typ.	Max.	
Clock cycle	f_{CLK} (1)	-	9	15	MHz
Hsync cycle	1/th	-	17.14	-	KHz
Vsync cycle	1/tv	-	59.94	-	Hz
Horizontal Signal					
Horizontal cycle	th	525	525	605	CLK
Horizontal display period	thd	480	480	480	CLK
Horizontal front porch	thf	2	2	82	CLK
Horizontal pulse width	thp (2)	2	41	41	CLK
Horizontal back porch	thb (2)	2	2	41	CLK
Vertical Signal					
Vertical cycle	tv	285	286	511	H (1)
Vertical display period	tvd	272	272	272	H (1)
Vertical front porch	tvf	1	2	227	H (1)
Vertical pulse width	tvp (2)	1	10	11	H (1)
Vertical back porch	tvb (2)	1	2	11	H (1)

Figure 2.18: Display timing values

2.12 User Input / Switches

There are three different types of switches used on the board. Five momentary pushbutton switches are used as the screen menu select buttons. Two latching pushbutton switches are used for triggering controls (slope and trigger mode). Finally, five rotary encoders are used - two for the channel A/B vertical sensitivities, one for the time sensitivity, and two for trigger delay and level.

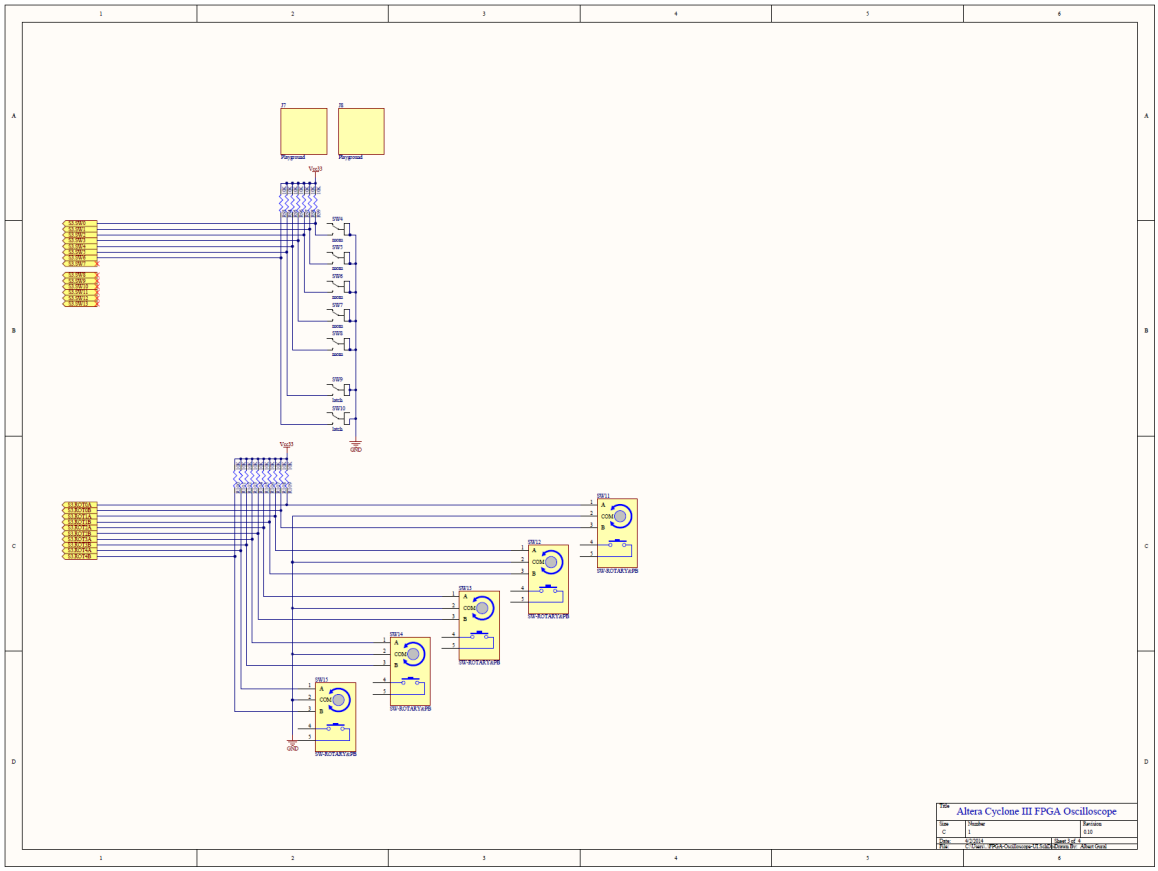

Figure 2.19: Switches schematic diagram

All of the switches are held high by $1k\Omega$ resistors (resistors in the range of $1k\Omega$ are required due to the low input impedance of the buffers). When the switch closes (or during certain phases of the rotary encoder), the switch terminal is brought from V_{CC} to GND. From there, the signals are debounced or decoded in FPGA logic to the soft core CPU.

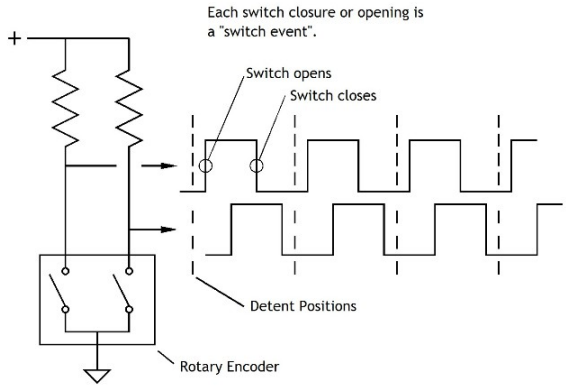

Figure 2.20: Rotary encoder timing

Rotary encoder timing is shown above. It uses a gray code between two outputs (A/B). Thus, the rotation direction can be detected from the pattern seen.

2.13 Board Layout

Refer to the following diagrams to understand the relationship between the raw board layout, the overarching large-scale hardware modules, and the physical part layout. The first diagram is just the raw board layout. Areas in red are top layer copper and areas in blue are bottom layer copper. The next diagram highlights the major hardware modules and buses. Finally, the third diagram overlays components on top, so the overall design can be intuited.

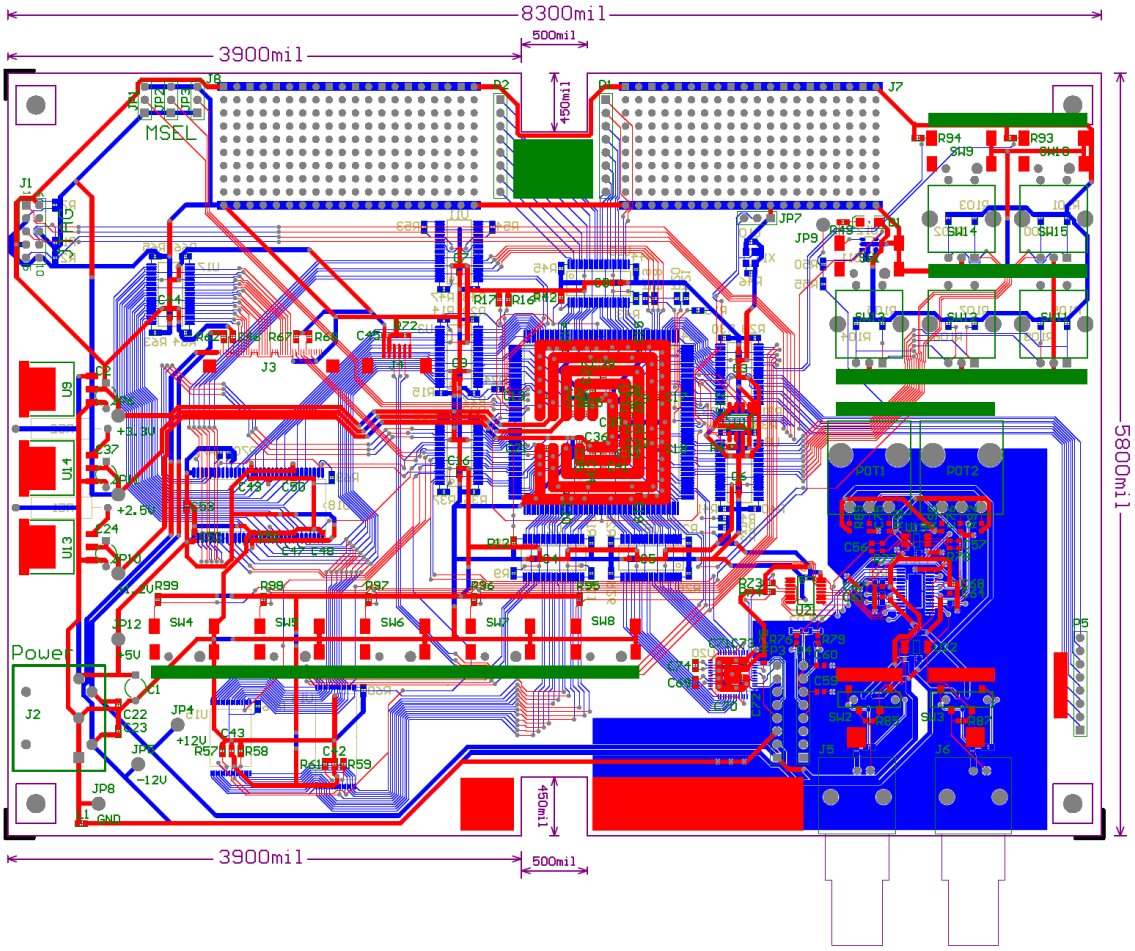

Figure 2.21: Board layout (raw)

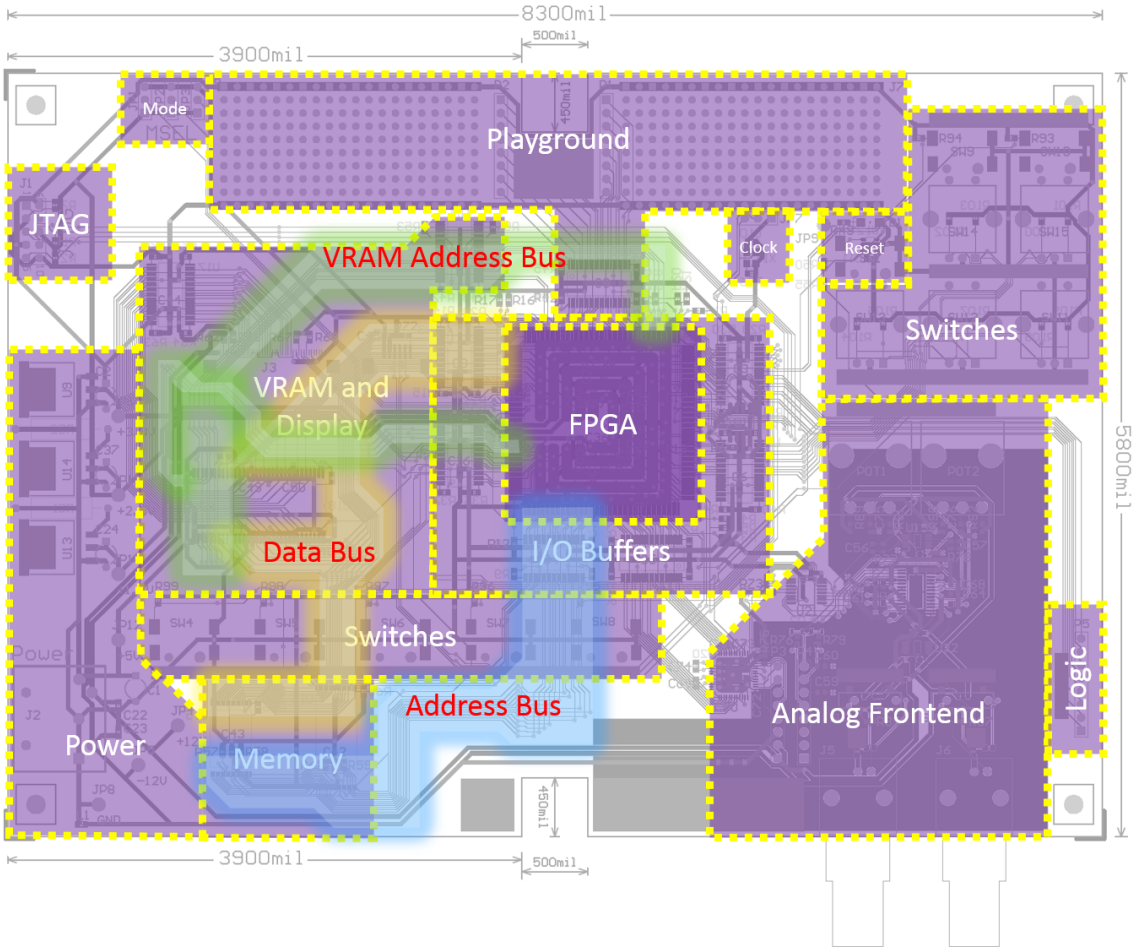

Figure 2.22: Board layout with annotations

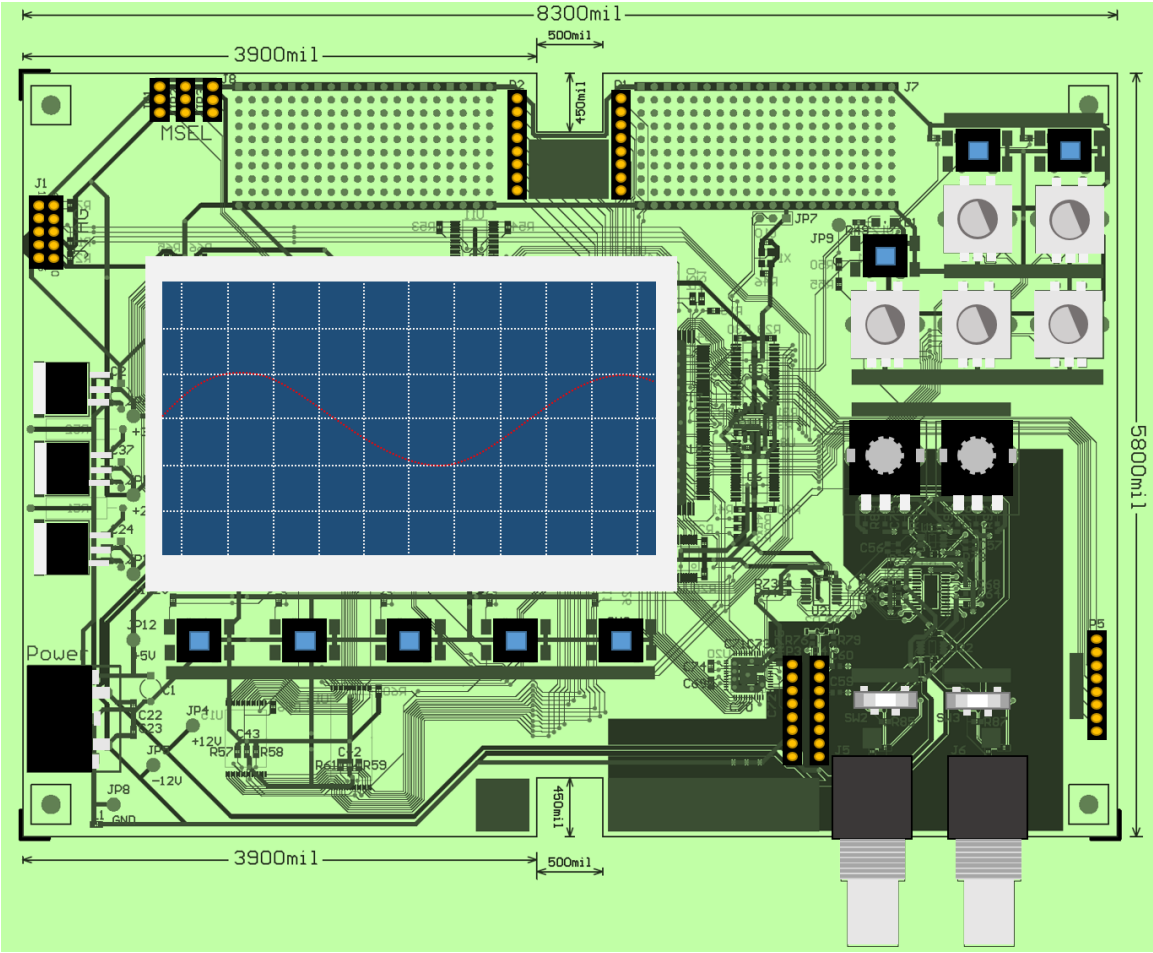

Figure 2.23: Board layout with top part overlay

Chapter 3

FPGA Firmware

3.1 Cyclone III FPGA

The FPGA used on this oscilloscope is Altera's Cyclone III EP3C25Q240 FPGA, with close to 25,000 logic elements. After setting up the chip properly, it can be configured into countless different logic circuits based on the application needs. This is very useful in applications where high speed is important (for example, in taking samples for an oscilloscope), but the quantities that would be produced are too low for ASICs to make sense.

To set up the FPGA for some simple logic circuit, it must be programmed on power-up, either via JTAG or some configuration circuit/chip (the FPGA's memory is volatile, so it requires reprogramming every time it's powered back on). There are a number of ways to create the design for the FPGA logic. One way is to create straight-up VHDL. Another is to create a block diagram file and draw out the logic circuits that way.

Altera provides a piece of software called "Quartus," which manages these different ways of designing the FPGA hardware connections. Quartus in fact provides a number modules that can be placed in a block diagram file. For example (and of particular use to us), a dual-clock FIFO can be placed in a design effortlessly. After creating a design, it can be programmed onto the FPGA.

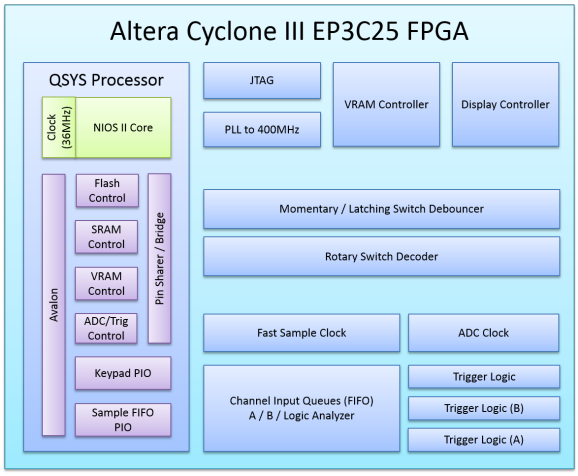

Figure 3.1: FPGA main components

Here's the major components of the FPGA logic. Most broadly, we have a NIOS II CPU, various clocks and timers, VRAM and display controllers, key debouncers and rotary encoder decoders, and finally, the scope sampling logic (FIFO storage, triggering, etc). We'll consider each of these in more detail.

3.2 Clocks

The system is fed an external $36MHz$ signal. This signal feeds direcly into some of the core logic, such as the NIOS II processor. However, in some cases we need a much faster (for sampling) or slower (for human-time-scale delays) clock signal.

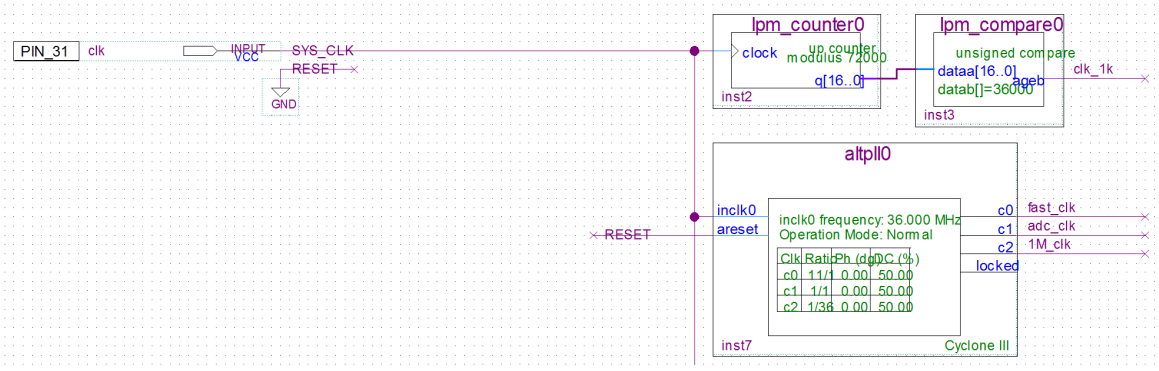

Figure 3.2: FPGA main clocks

A 50% duty cycle $500Hz$ clock (incorrectly named `clk_1k`) is produced by hooking a joint counter/compare to the $36Mhz$ signal. With a counter taken modulo $72,000$, incrementing at $36Mhz$ takes $2ms$. With a compare at $36,000$, we can then get a 50% duty cycle $500Hz$ clock.

For the ADC and FIFO sampling, we want a much higher rate of about $400MHz$ (this is useful for the logic analyzer, even though the ADC can only produce new samples at $80MHz$). To do this, we use an phase-locked loop with an $11\times$ multiplier. This produces a $396MHz$ signal, close enough to $400MHz$. $400MHz$ was chosen because of its easy divisibility by factors of 2 and 5 (the common time scale factors used in scopes).

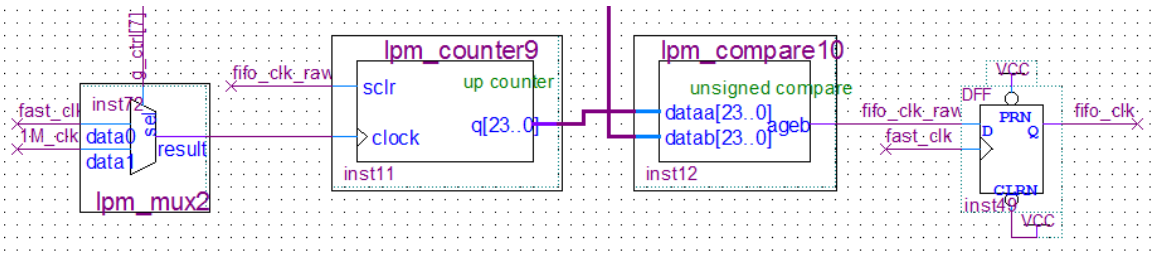

Figure 3.3: FPGA sample clock

We need to be able to modify the rate at which we sample data in our code. To do this, we can attach a counter/compare to the fast $400MHz$ clock, and have the compare value set by the code. The logic above produces the sample clock. Notice that there are two selectable inputs to the counter/compre logic. One is the fast $400MHz$ clock for very high sample rates, and the other is a much slower $1MHz$ clock for slower sample rates. This is necessary because the $400MHz$ clock is less stable, especially with large counters such as a $32-bit$ counter, but has okay operating characteristics with the 24-bit one seen above.

3.3 Sample Handling

From left to right, we have the analog input logic, the sample FIFO and FIFO clock, and the trigger logic. We will look at each of these in detail.

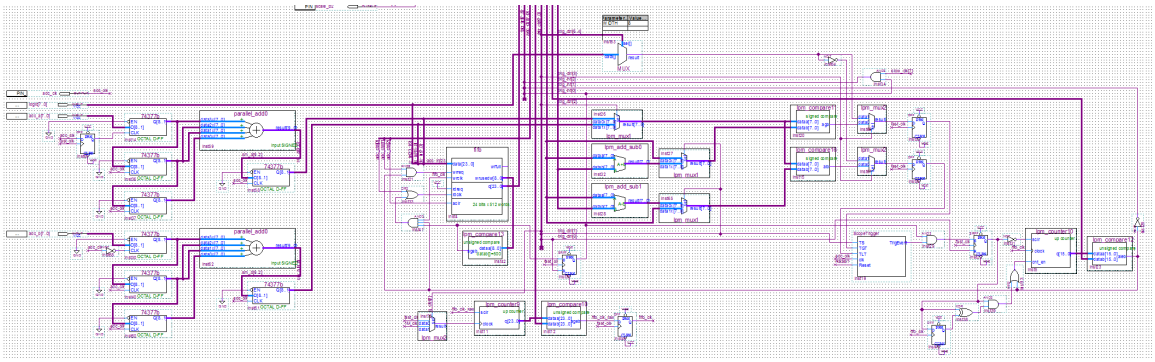

Figure 3.4: FPGA overview of scope sample handling

3.3.1 Analog Input

The signal coming in from the ADC is somewhat noisy due to the suboptimal single-ended design of the analog frontend input to the ADC (it's designed for dual-ended input). Because of that, the signal can be a little noisy. To remedy this issue, the ADC signal is sent through a series of three 8-bit DFFs, and then a very simple 1D time-domain Gaussian blur is done on the signal.

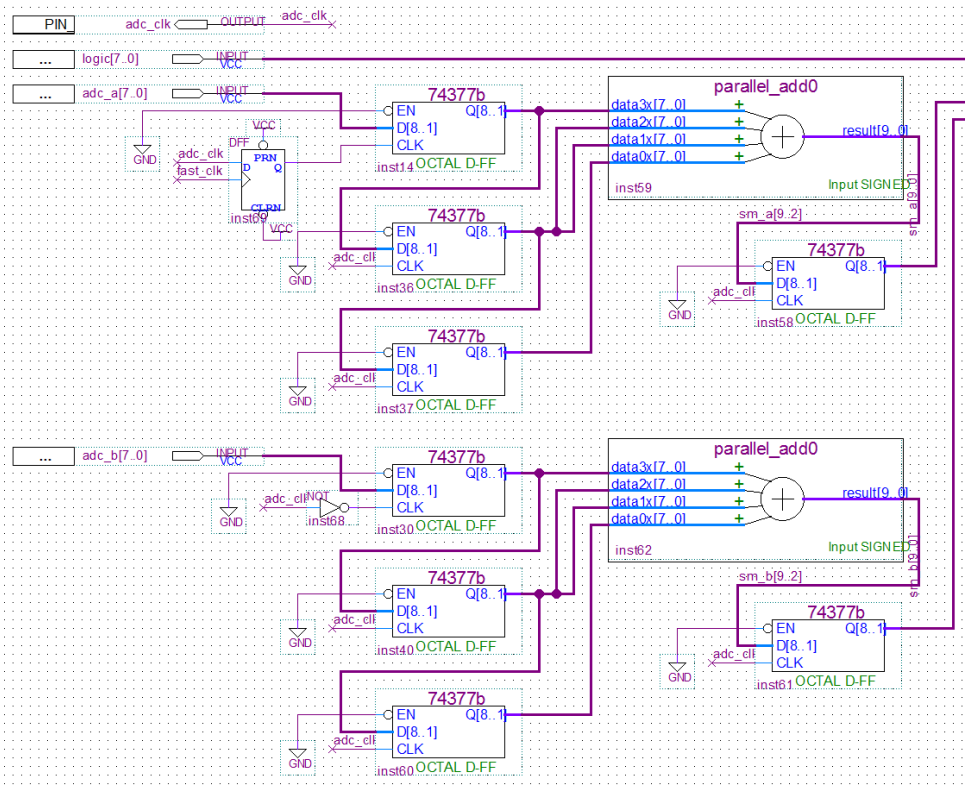

Figure 3.5: FPGA sample input

Also notice that the first DFFs taking the ADC signal are offset by close to 180^o (in the top, there's a $\approx 2.5ns$ delay and in the bottom, a $\approx 12ns$ delay from the adc clock signal). This is because of how the ADC synchronizes its A and B channel outputs.

Output from the result of this logic is fed into the sample FIFO as well as the trigger logic.

3.3.2 FIFO Sampling

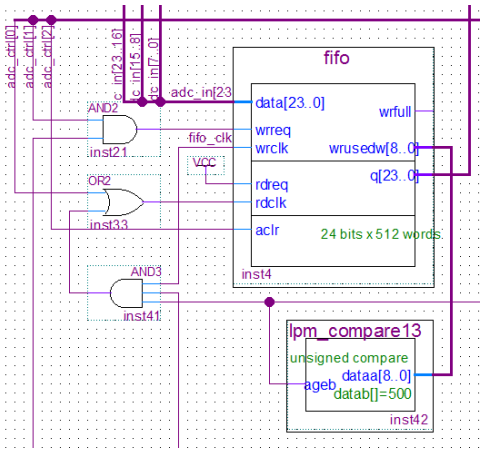

Figure 3.6: FPGA sample FIFO

We are using a dual-clock FIFO in which data can be asynchronously written to (the back of the FIFO) and read from (the front of the FIFO) with different clocks. The way it's set up, the FIFO's write clock is based on the sample clock which can be as high as $400MHz$. Meanwhile, the read clock is bit-banged by the CPU when it wants to get the next 24 bits of data.

The FIFO logic is slightly complicated. First, when the FIFO fills, `lpm_compare13` tells it to automatically start removing data (this allows the FIFO to act like a rotating buffer, always containing the last N samples). Second, the FIFO stops collecting data automatically only when it sees a trigger event. Finally, the CPU has control over whether to allow reading and writing, and it can also clear the FIFO.

3.3.3 Triggering

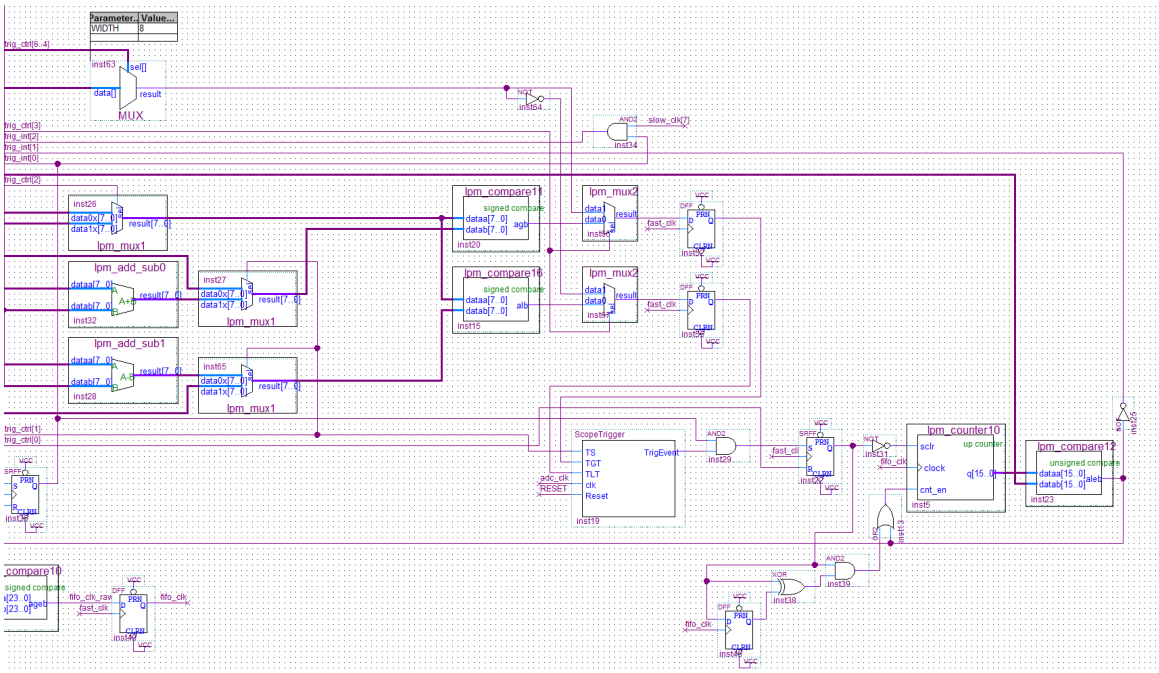

Figure 3.7: FPGA triggering

The function of this logic is to take the sample inputs and detect, using FPGA logic, when a given one of their values crosses a certain threshold, either in the positive or negative direction. This causes a trigger event which can automatically stop the sample FIFO after some specified delay. From here, the FPGA no longer accepts samples from the ADC or logic analyzer and the CPU can collect the FIFO samples. Once the CPU is finished, it can reset the FPGA logic to detect the next trigger event. The CPU provides information such as trigger channel, trigger level, trigger slope, and trigger error signals to control this section. It also provides a reset signal for when it's finished with the current trigger and wants the next one.

In the top left is a couple adders and MUXs. These take the desired trigger level and trigger error and produce dual outputs of the trigger level ± the trigger error. These outputs serve as upper and lower comparisons for the analog signal that's fed in through `lpm_mux1` (which selects which analog channel to trigger on).

Above that is the logic analyzer input which also goes through a MUX to determine which bit (if any) to trigger on. From here, the compare results from `lpm_compare11` and `lpm_compare16` are MUXed with the logic analyzer and its inverse to select which of these signals we're interested in triggering against. Note that just as the analog signal has a lower and upper threshold, the logic analyzer can be fed as an upper threshold and the inverse of it as a lower threshold to provide identical thresholding functionality without requiring too much extra logic.

The selected trigger signals are cleaned with a DFF and then sent to the scope trigger state machine block which handles when to assert a trigger event based on the trigger thresholds `TGT` and `TLT` it sees, as well as the slope `TS`. Once a trigger event is asserted, it sets an SRFF `inst22`, which turns on counter `lpm_counter10`. The logic below just turns off the counter before it loops over. This counter is responsible for the trigger delay, fed in by the CPU as a compare value for `lpm_compare12`. Once the delay is over, the compare block outputs a high signal which sends an interrupt to the CPU and tells the FIFO to stop sampling data.

3.3.4 Trigger State Machine

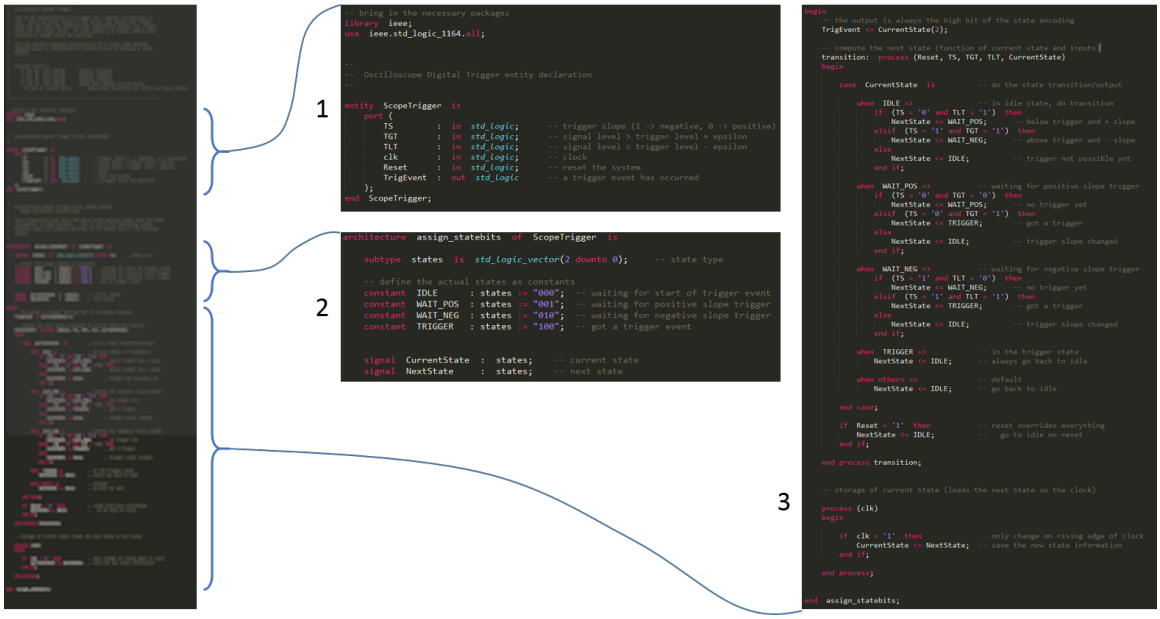

Figure 3.8: FPGA triggering state machine

Shown above is the trigger state machine. It contains three main components, labeled above as (1) the input and output signals, (2) the states, and (3) the transitions. Operation is fairly straightforward and utilizes hysteresis on the two thresholded inputs. If `TGT` goes from less than to greater than, then the signal just underwent a positive slope transition. The same goes for `TLT` for negative slopes.

Notice that in the `IDLE` state, it can only transition to `WAIT_POS` after going below the lower threshold, whereas the upper threshold is needed to cause a trigger event. This is what allows for hysteresis. The same of course holds for `WAIT_NEG`.

3.4 VRAM and Display

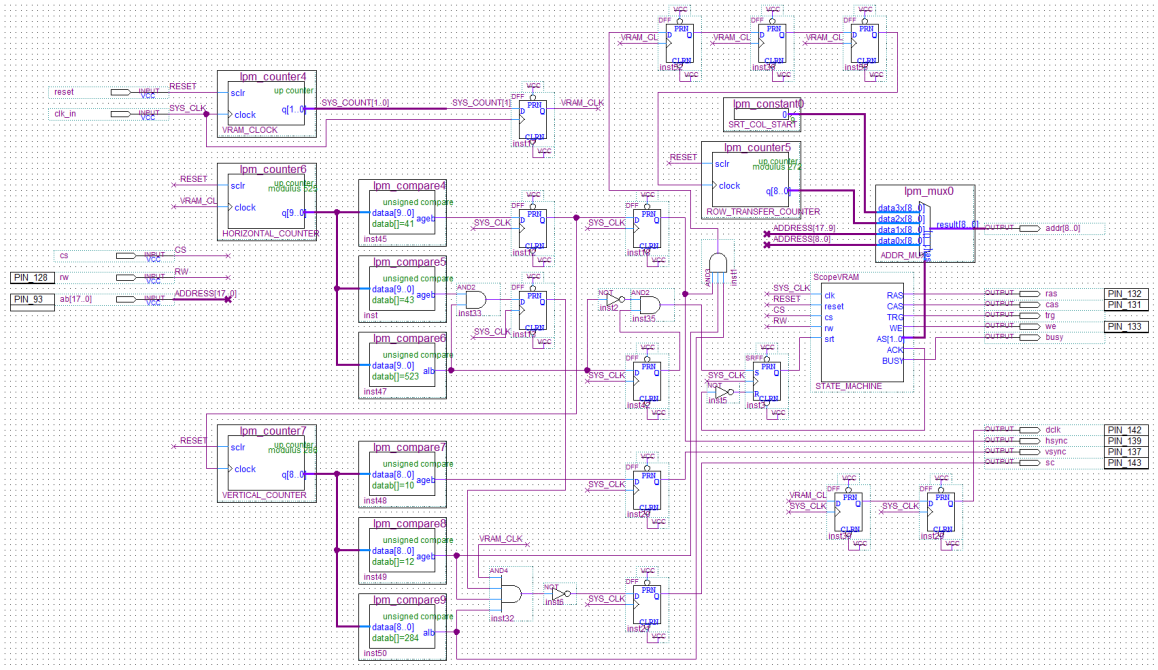

Figure 3.9: FPGA overview of VRAM controller and display controller

The display and VRAM control logic are lumped together due to the similarities in requirements of their timing (although most of the VRAM control is handled by the state machine). Most of the logic seen above controls timing for the display clocks, while the portion in the upper right is for VRAM control. Much of the VRAM control logic shown here controls the 9-bit VRAM address output.

3.4.1 VRAM Controller

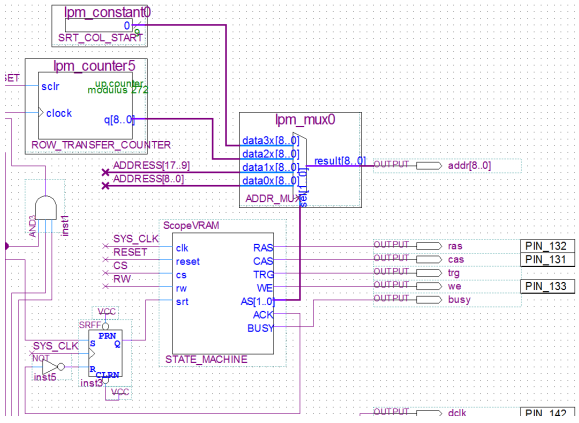

Figure 3.10: FPGA VRAM addressing

The heart of the VRAM controller is the state machine `ScopeVRAM`. It takes the typical memory outputs of the CPU (CS and RW) and uses that for much more advanced control to the VRAM chip. The state machine only returns a `busy` signal to the CPU to indicate completion of the requested task.

Besides the state machine, the other logic useful for the VRAM controller is selecting the address to put out on the VRAM address bits. This is done by MUXing the two halves of the address bits along with a couple other address, controlled by the state machine. The two halves of the address bus are used for specifying row and column in a standard read to or write from the CPU. The other two address are used for serial row transfer cycles and specify the row (output of `lpm_counter5`) and column to start at (`SRT_COL_START`) for a row transfer. The counter is updated and the serial row transfer state triggered right after every row of pixels is written to the display (at the end of a HSYNC cycle).

3.4.2 VRAM State Machine

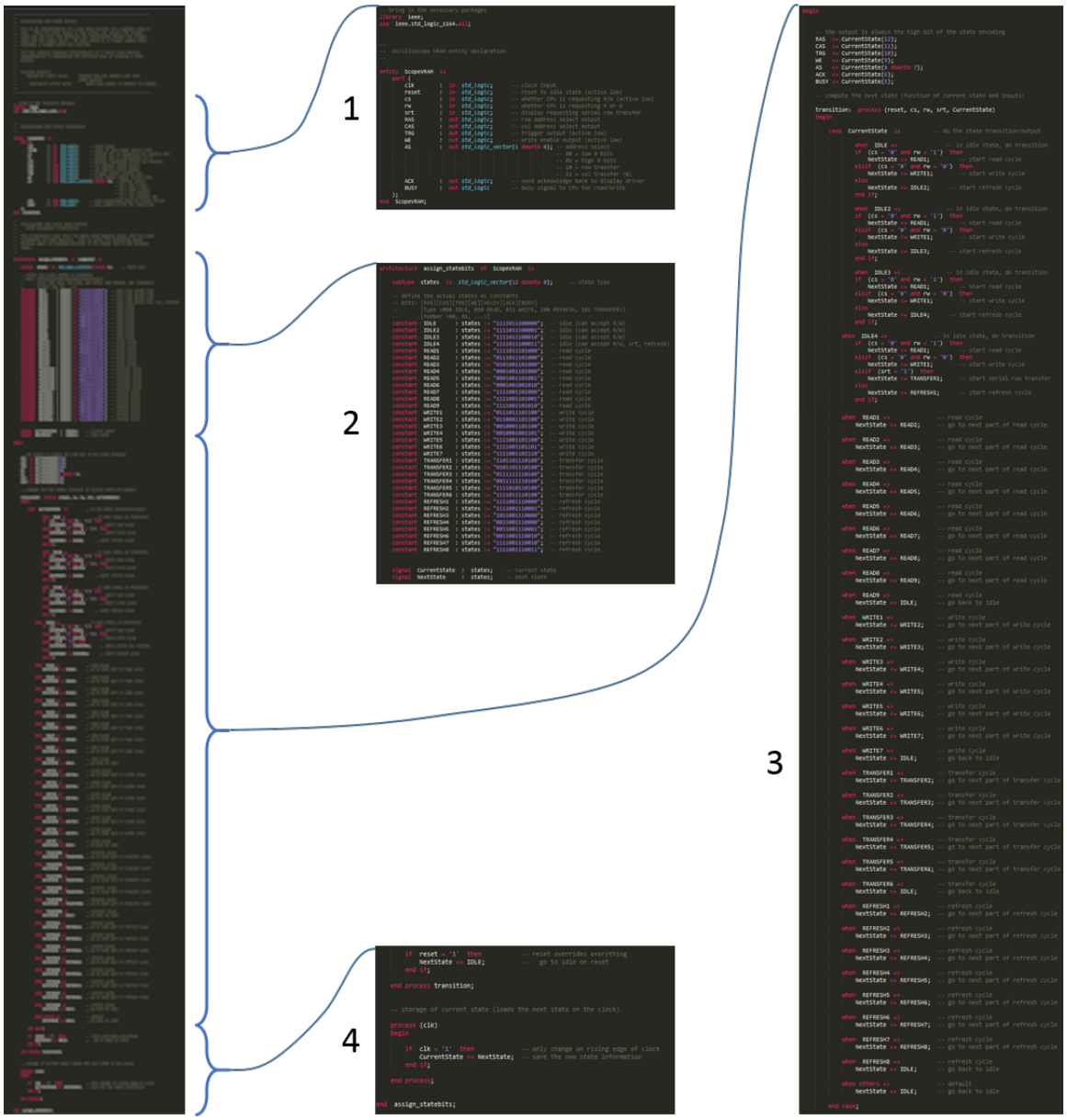

Figure 3.11: FPGA VRAM controller state machine

The state machine is shown above. Just as in the triggering state machine, the VRAM state machine has (1) the inputs and outputs, (2) the states, and (3/4) the transitions. There are a lot of states, but their purpose is really to handle four main operations: read, write, refresh, and serial row transfer. In any given one of those desired operations, there are a number of states which run sequentially and handle the signal timing for the desired operation. In the transitions (3), you can see that the bulk of transitions move from a state in one operation to the next state in that operation (and at the end going back to `IDLE`).

3.4.3 Display Controller

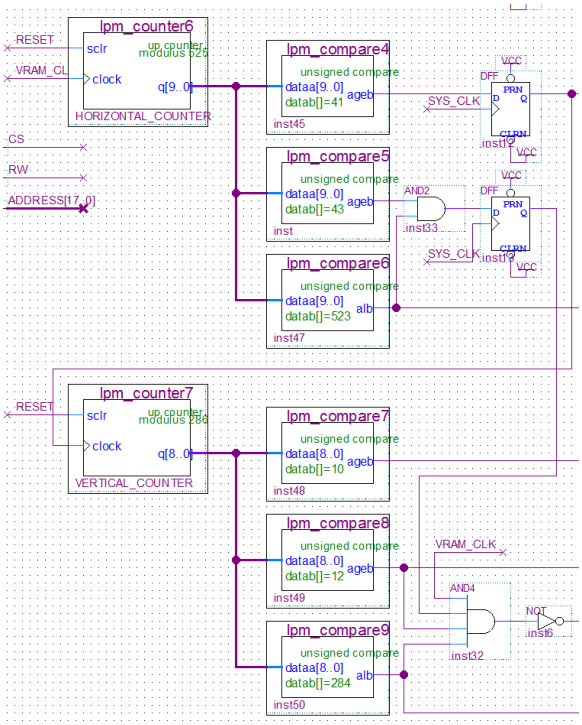

Figure 3.12: FPGA display counter timers

Display timing is shown above. The `VRAM_CLK` operates at about $9MHz$, approximately the dot clock frequency required by the display. `lpm_compare4-6` then take care of a given HSYNC cycle - mainly the HSYNC pulse and the back and front porches. The HSYNC pulse in turn acts as a clock for the VSYNC counter. `lpm_compare7-9` take care of a given VSYNC cycle - mainly the VSYNC pulse and the back and front porches.

3.4.4 Display Timing (Testing)

The tough timing requirements make testing a very good idea. Testing was done in Altera ModelSim with VHDL test code (provided in the appendix). To prove the importance of testing, here's an initial run of the display timing.

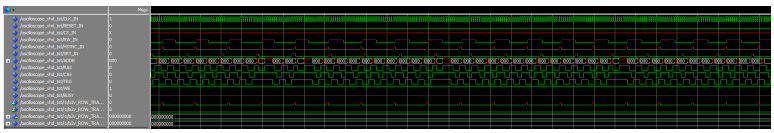

Figure 3.13: Display timing test, problematic start

Notice that large portions of the signal are red, indicating there's a timing problem. In contrast, a successful run should have all green signals. Shown below is a semi-successful startup. The signals start out red due to a conflict in signal outputs (this was fixed), but then everything works properly.

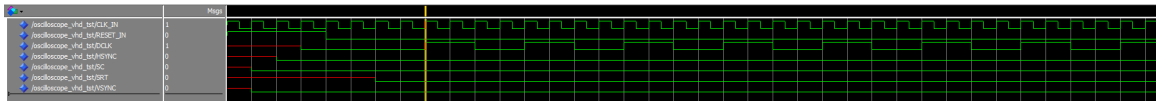

Figure 3.14: Display timing test, semi-successful start

After fixing everything, all of the timing works out properly. Here are the signals at different zoom levels. First, looking at just the pulse of a horizontal cycle. Then looking at the full horizontal cycle. Then, looking at a pulse of a vertical cycle, and finally the full cycle (each period represents one full display refresh).

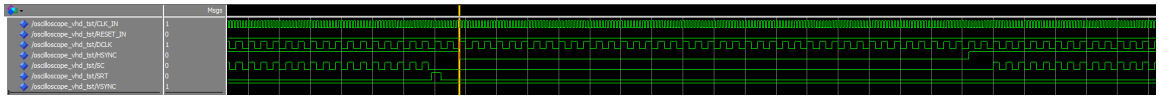

Figure 3.15: Display timing test, successful horizontal pulse

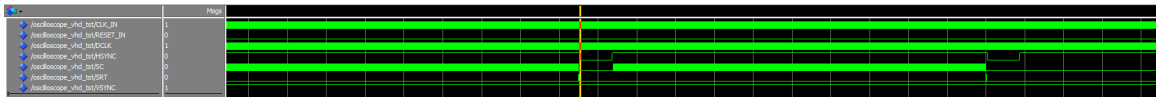

Figure 3.16: Display timing test, successful horizontal cycle

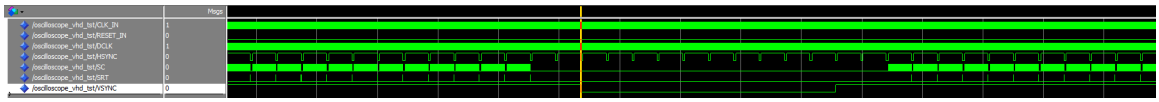

Figure 3.17: Display timing test, successful vertical pulse

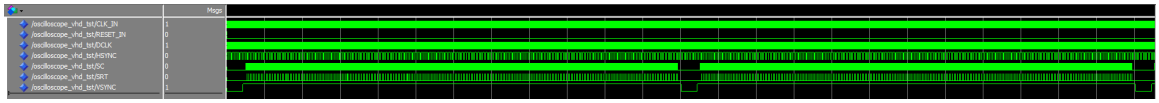

Figure 3.18: Display timing test, successful vertical cycle

We can see that the results are successful:

```
run
run
run
VSIM(paused)> run

VSIM(paused)>
```

Figure 3.19: Display timing test, all tests successful

3.5 Switches

All of the switch debouncing and decoding is done in FPGA logic. All of the switch-handling logic is shown here:

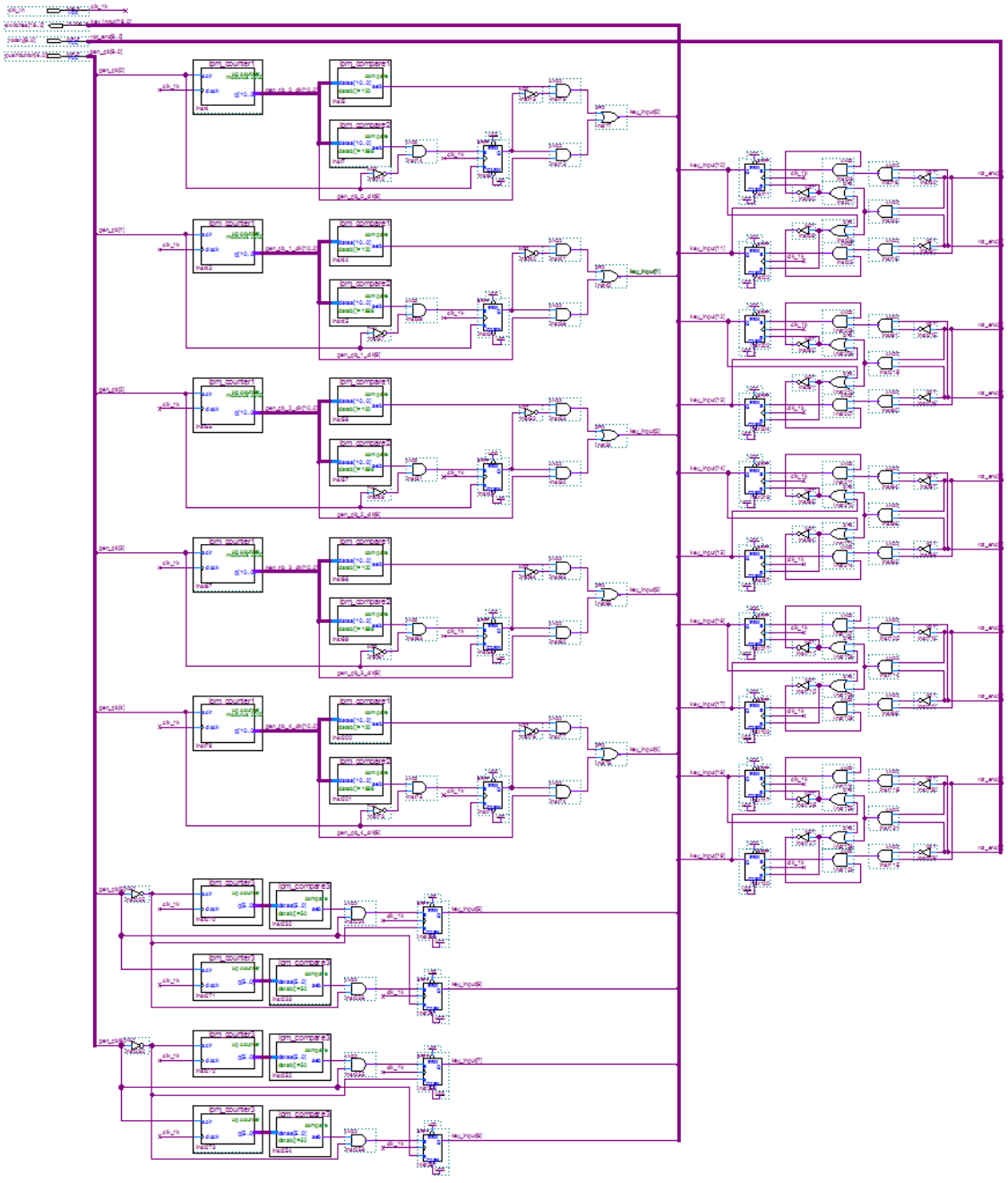

Figure 3.20: FPGA overview of key and rotary encoder input

The five display buttons are handled by logic in the top left, the two latching pushbuttons are handled in the bottom left, and the rotary decoding is handled by the five circuits on the right.

3.5.1 Momentary Pushbutton Input

The following logic circuit shows how the more basic momentary pushbuttons are debounced. When a pushbutton signal (active low) comes in, it unsets the counter clear, which allows the counter to start incrementing. If the button is pressed long enough without a bounce, then the counter will reach the first compare value, generating an interrupt for the CPU. If the switch continues being pressed, the second compare also kicks in and handles automatic switch repetition.

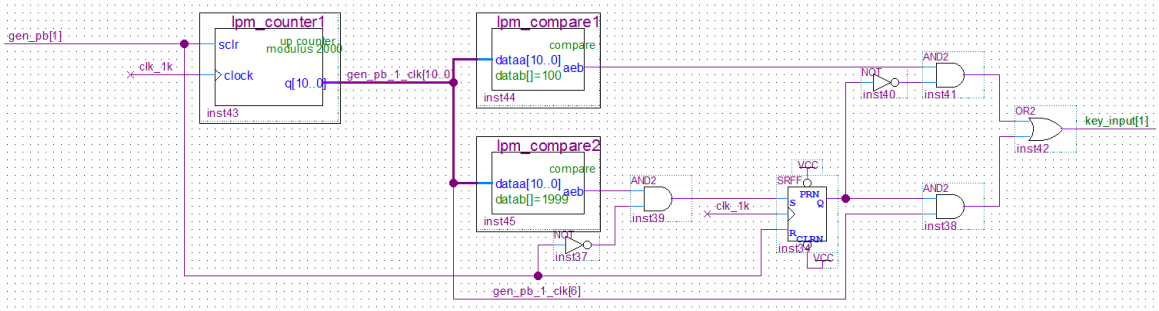

Figure 3.21: FPGA momentary pushbutton input debouncing

3.5.2 Latching Pushbutton Input

The latching pushbuttons are debounced in a similar manner to the momentary ones, with a counter/compare pair. However, there's no need for switch repetition. Instead, there's a need for debouncing the switch in both directions (hence the pair of counter/compares). Closing the pushbutton as well as openning the pushbutton will send separate interrupts to the CPU.

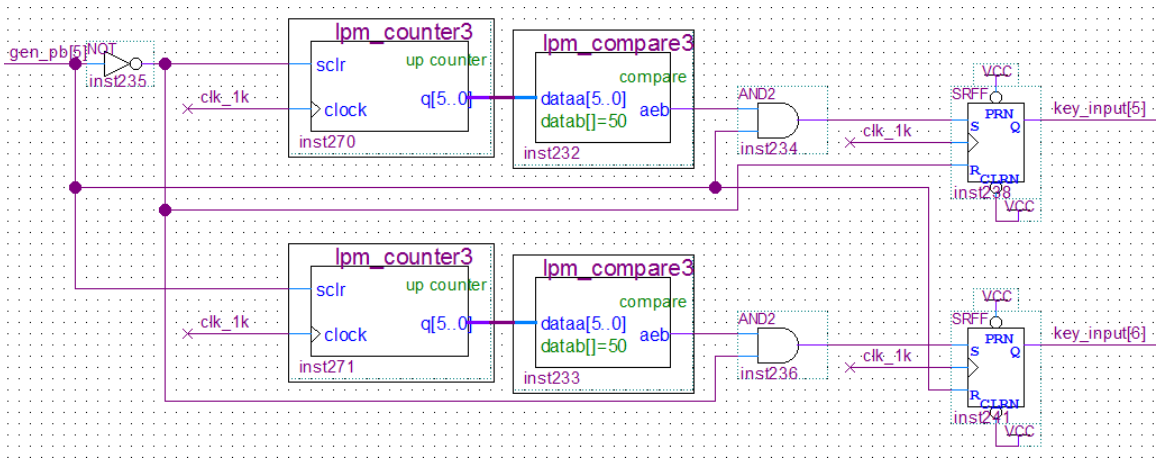

Figure 3.22: FPGA latching pushbutton input debouncing

3.5.3 Rotary Encoder Input

At each detent of the rotary encoder, the A and B lines are held high. During a rotation by one detent, the A and B lines cycle through a grey code pattern depending on which direction the rotation was in. We can see this pattern in the diagram below. One way we can decode the signal, then, is to detect which of $\{A, B\}$ lines go low first (this indicates the rotation direction), and then wait for both of them to return high before going back to the initial state.

The diagram below shows the exact opposite polarity as what's described above, but the function is exactly the same.

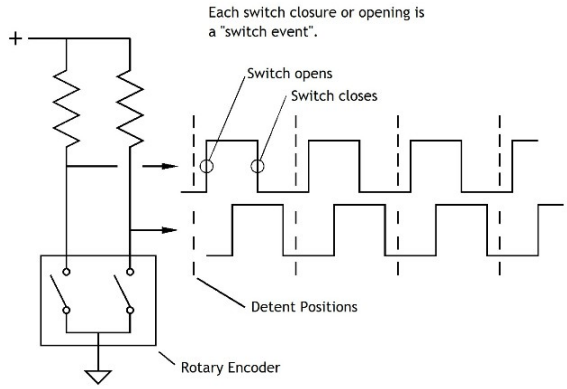

Figure 3.23: Rotary encoder timing

We see that logic in the following circuit. In this circuit, the A and B rotary encoder lines come in from the right, and then interrupts for each of a CW or CCW turn are sent to the CPU on the left. We see the actual logic as described above. Whichever line falls low first sets the appropriate SRFF (triggering either a CW or CCW interrupt), and suppresses the other line from setting the opposing SRFF. Finally, the reset signal is generated when both lines go high again.

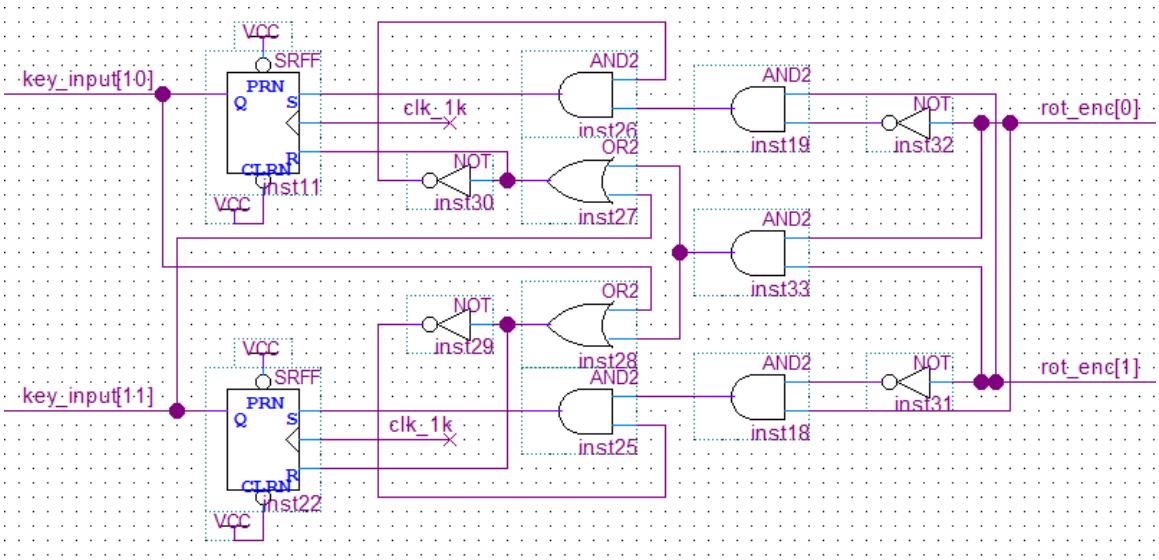

Figure 3.24: FPGA rotary encoder decoding

3.6 NIOS II Processor

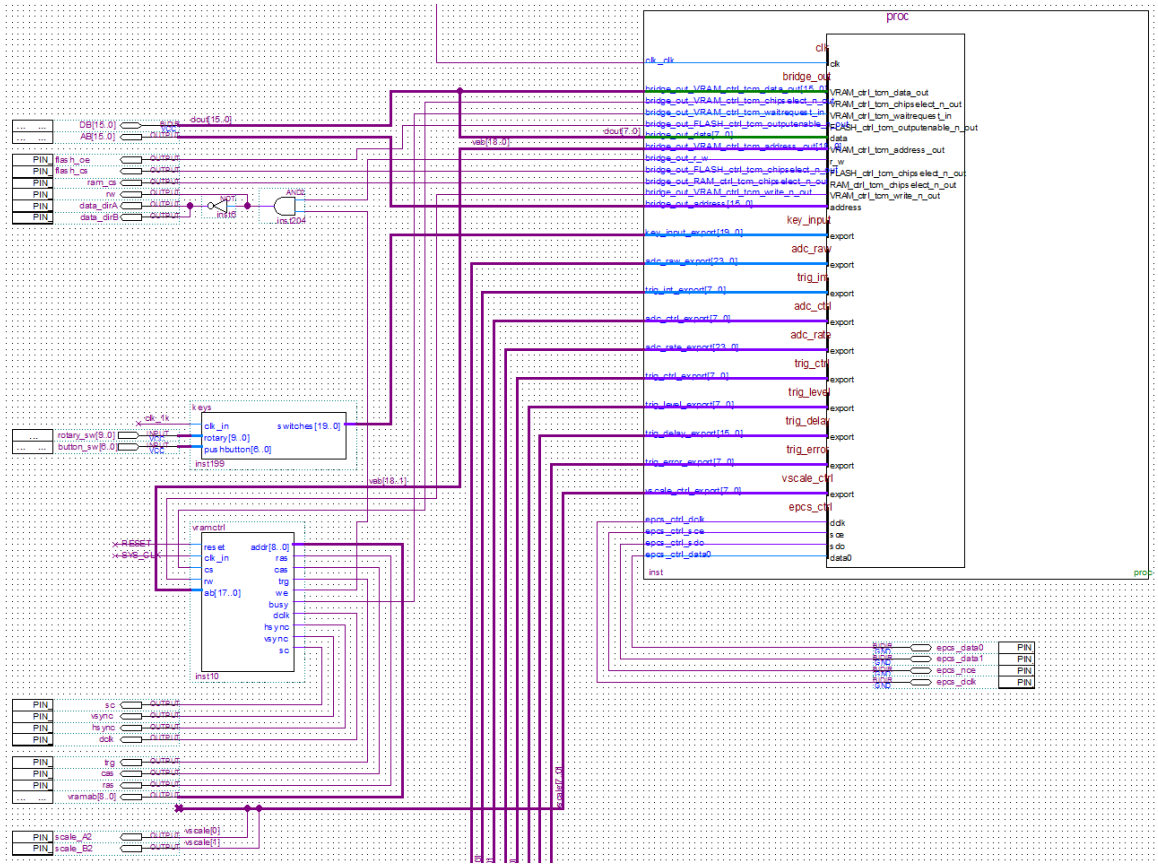

Figure 3.25: FPGA NIOS II processor block

The Altera Cyclone III FPGAs have enough logic gates to support a simple processor, called the NIOS II. It has a RISC architecture with 32 registers and a small instruction set. More information about the NIOS II can be found online.

In the circuit above, the processor and some core peripherals are shown in a single block at the top right. It interfaces with FPGA I/O (at the left of the schematic), as well as the key and VRAM control modules (middle left), EPCS (bottom right), and triggering logic (not shown, below).

3.6.1 Overview of Capabilities

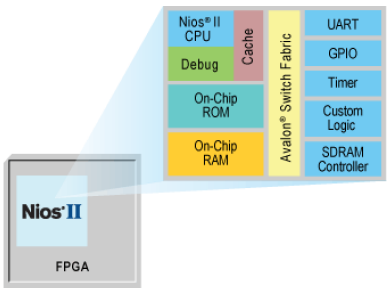

Figure 3.26: NIOS II soft core processor

The processor can be built up using the a subtool of Quartus called QSYS. This system can be used to create not only the core NIOS processor, but also supporting components such as an Avalon bus for managing peripherals, interrupts, input and output to the CPU, etc.

Once the processor is designed in QSYS, it can be generated into a block for use in the logic circuit. In a block diagram file, all inputs/outputs exported in the QSYS screen can be hooked up directly to the other FPGA logic. This allows for seamless integration of the FPGA logic and the soft core CPU.

3.6.2 Peripherals

In addition to the basic NIOS II processor and clock, there are a number of peripherals we can add.

For memory modules which share signals to the outside world, we do this with a Generic Tri-State Controller for each memory module, a Tri-State Conduit Pin Sharer to share certain connections/buses, and finally a Tri-State Conduit Bridge takes the shared symbols and exports them as pins on the processor block.

For simple inputs/outputs between the processor and the outside world, a simple Parallel I/O peripheral can be used. These PIOs can be configured as input, output, and bidirectional. Inputs can trigger interrupts.

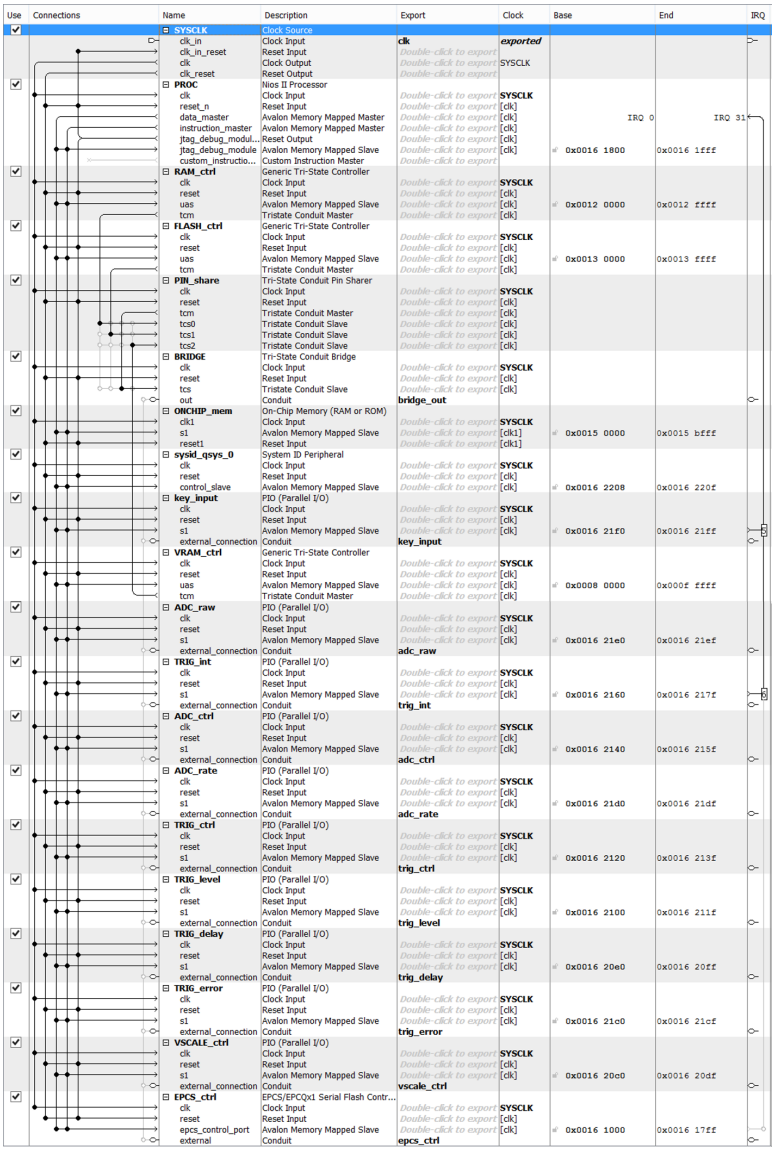

Figure 3.27: NIOS II core peripherals

3.6.3 NIOS II Settings

These are the settings for the NIOS II processor. We're using the NIOS II/s version, which includes instruction caching and hardware divides. Caching is particularly important. Without caching, the processor runs much slower and the LCD refresh rate is very noticeable.

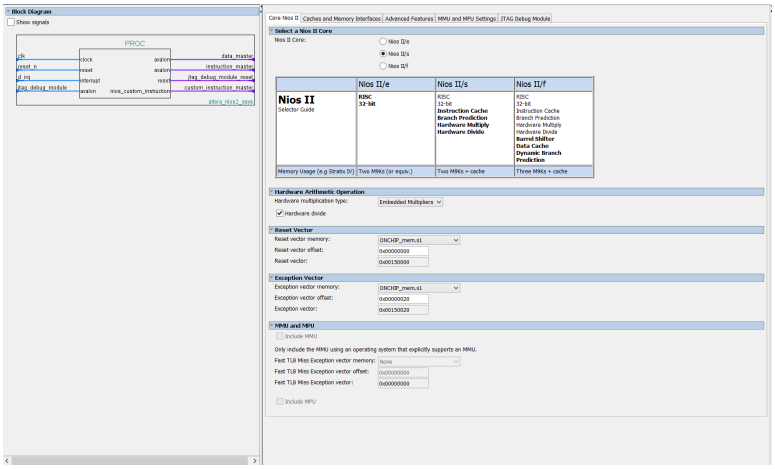

Figure 3.28: NIOS II core settings

3.6.4 Address Memory Map

Finally, with this whole QSYS processor setup, we can generate the addresses for each component, seen below. Once that's done, we generate the entire processor (VHDL file, as well as a block diagram block). Configuration details about the base addresses (including `#defines`) of each of these peripherals is included in the generated processor files. This can be used in the NIOS II code to allow for easy maintenance of the NIOS code, agnostic to changes in the actual base addresses.

	PROC.data_master	PROC.instruction_master
PROC.jtag_debug_module	0x0016 1800 - 0x0016 1fff	0x0016 1800 - 0x0016 1fff
RAM_ctrl.uas	0x0012 0000 - 0x0012 ffff	0x0012 0000 - 0x0012 ffff
FLASH_ctrl.uas	0x0013 0000 - 0x0013 ffff	0x0013 0000 - 0x0013 ffff
ONCHIP_mem.s1	0x0015 0000 - 0x0015 bfff	0x0015 0000 - 0x0015 bfff
sysid_qsys_0.control_slave	0x0016 2208 - 0x0016 220f	0x0016 2208 - 0x0016 220f
key_input.s1	0x0016 21f0 - 0x0016 21ff	0x0016 21f0 - 0x0016 21ff
VRAM_ctrl.uas	0x0008 0000 - 0x000f ffff	0x0008 0000 - 0x000f ffff
ADC_raw.s1	0x0016 21e0 - 0x0016 21ef	0x0016 21e0 - 0x0016 21ef
TRIG_int.s1	0x0016 2160 - 0x0016 217f	0x0016 2160 - 0x0016 217f
ADC_ctrl.s1	0x0016 2140 - 0x0016 215f	0x0016 2140 - 0x0016 215f
ADC_rate.s1	0x0016 21d0 - 0x0016 21df	0x0016 21d0 - 0x0016 21df
TRIG_ctrl.s1	0x0016 2120 - 0x0016 213f	0x0016 2120 - 0x0016 213f
TRIG_level.s1	0x0016 2100 - 0x0016 211f	0x0016 2100 - 0x0016 211f
TRIG_delay.s1	0x0016 20e0 - 0x0016 20ff	0x0016 20e0 - 0x0016 20ff
TRIG_error.s1	0x0016 21c0 - 0x0016 21cf	0x0016 21c0 - 0x0016 21cf
VSCALE_ctrl.s1	0x0016 20c0 - 0x0016 20df	0x0016 20c0 - 0x0016 20df
EPCS_ctrl.epcs_control_port	0x0016 1000 - 0x0016 17ff	0x0016 1000 - 0x0016 17ff

Figure 3.29: NIOS II address memory map

Chapter 4

Software

4.1 Software Overview

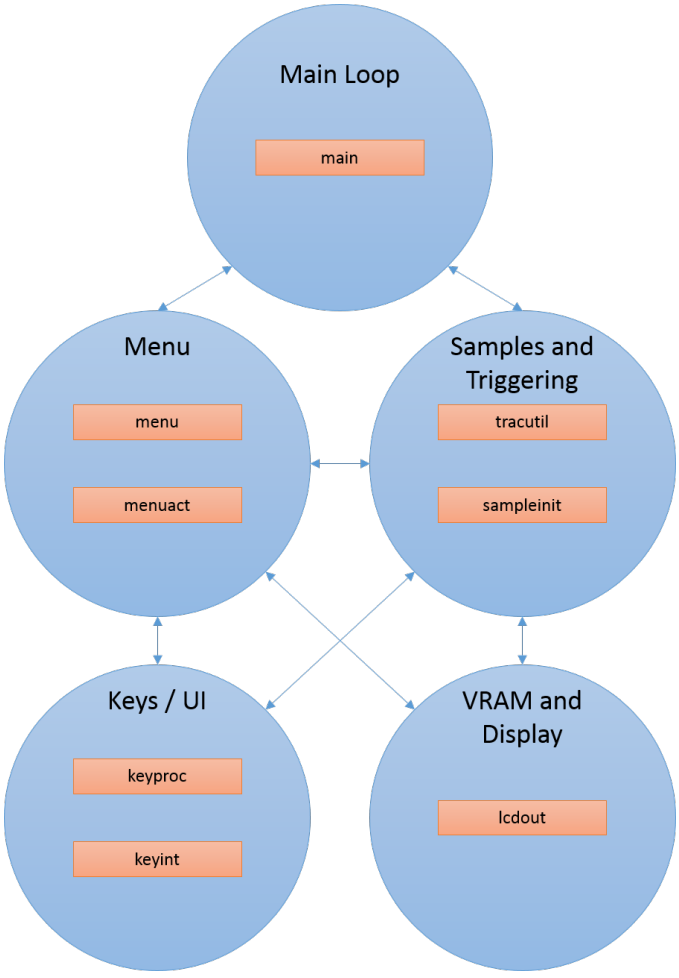

Figure 4.1: Software main components

The software is composed of a number of separate files which can be grouped according to their like purposes. Consider the block diagram above.

The main loop continually checks for key presses and new samples in an infinite loop. The Menu subcomponent is responsible for displaying, updating, and keeping the state of the menu. The Samples and Triggering subcomponent is responsible for both collecting samples (`sampleinit`) and displaying them (`tracutil`). Both of these subcomponents utilize two other subcomponents, "Keys/UI" and "VRAM and Display". Keys is responsible for getting key interrupts and processing them, while VRAM and Display is responsible for handling the basics of writing to the display.

4.2 Main Loop

The main loop continually checks the three following things:

- If ready to get another trace => do the trace
- If ready to display a trace => plot the trace
- If there's a key press => process the key

The only thing the main loop is responsible is installing the interrupt handlers during startup (these are the key interrupt handler and the adc interrupt handler).

4.3 Menu

There are two files of importance - `menu` and `menuact`. `menu` provides functions related to the display of the menu, such as clearing and changing the selection on screen. `menuact` provides functions related to what happens when certain menu actions are triggered, such as increasing the trigger level when so desired.

4.4 Samples and Triggering

Again, there are two files of importance - `tracutil` and `sampleint`. `tracutil` will display a provided sample on the screen. In particular, the function `plot_trace` handles the actual plotting of a new trace. In fact, it takes an array of sample buffers, one for each of the inputs (A, B, Logic analyzer).

On the other hand `sampleint` is responsible for handling interrupts generated by the FPGA logic when a trigger occurs. When this happens, the FIFO sample data is read to a memory buffer, which the CPU (with `tracutil`) can then plot.

4.5 Keys / UI

Once again, there are two files of importance - `keyproc` and `keyint`. `keyproc` takes a given key code and determines what action it should take. This generally includes modifying the menu and changing one of the trigger control values (level, delay, etc).

`keyint` is responsible for handling interrupts that are generated when a keypress event occurs. When handling such an interrupt, `keyint` may either pass on that key code for `keyproc` to take care of, or it may trigger the action itself. Typically, the pushbuttons below the display will trigger `keyint` to pass on the key code to `keyproc`, whereas the hardware mimic buttons trigger direct actions. This is because the hardware mimic buttons should not modify the menu.

4.6 VRAM and Display

There's one file of importance - `lcdout`. This file includes functions for modifying the display, such as clearing, drawing a line, or writing a character or string. These functions are used by `menu` for, for example, writing the menu text. `tracutil` uses VRAM and Display functions (namely, `plot_pixel`) for plotting the trace.

List of Figures

List of Tables

Appendices

Appendix A

Block Diagrams

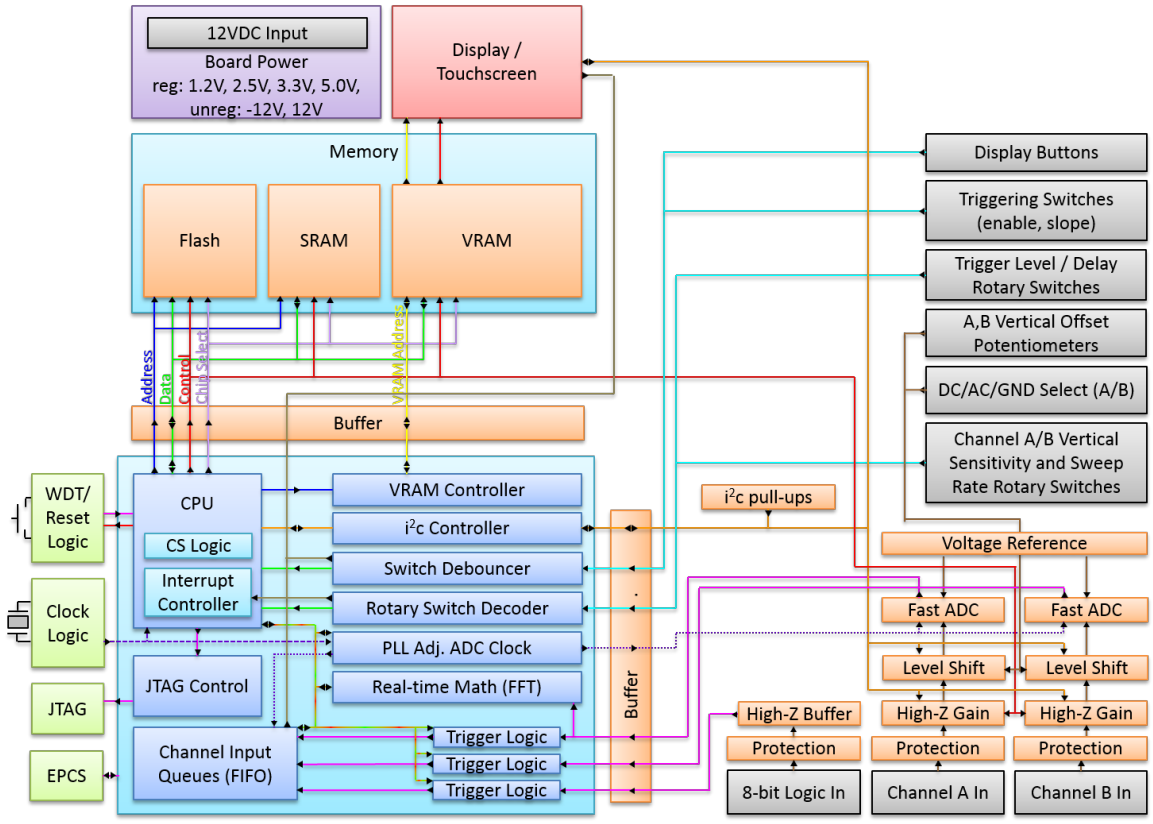

Figure A.1: Full block diagram

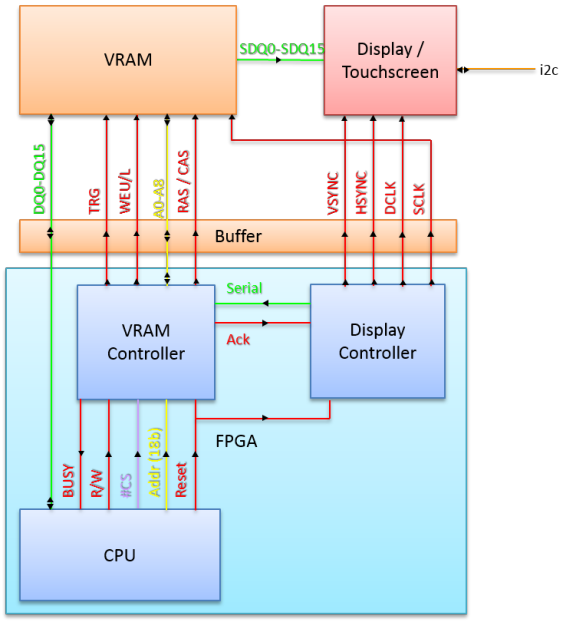

Figure A.2: Display block diagram

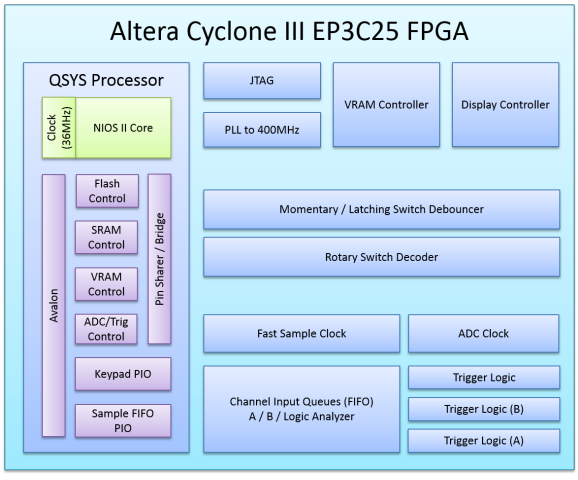

Figure A.3: FPGA main components

Appendix B

Schematic Diagrams

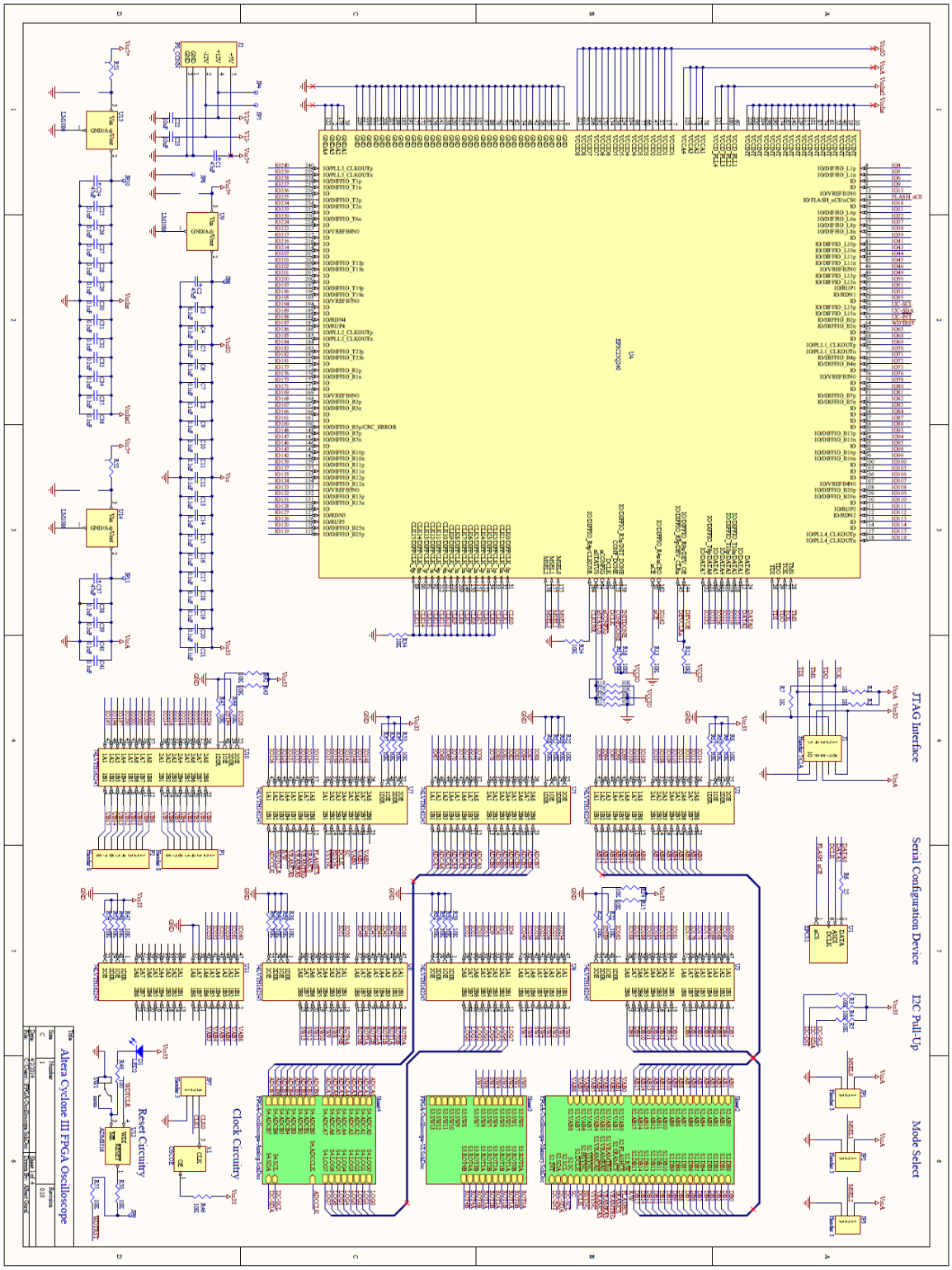

Figure B.1: FPGA and supporing components schematic diagram

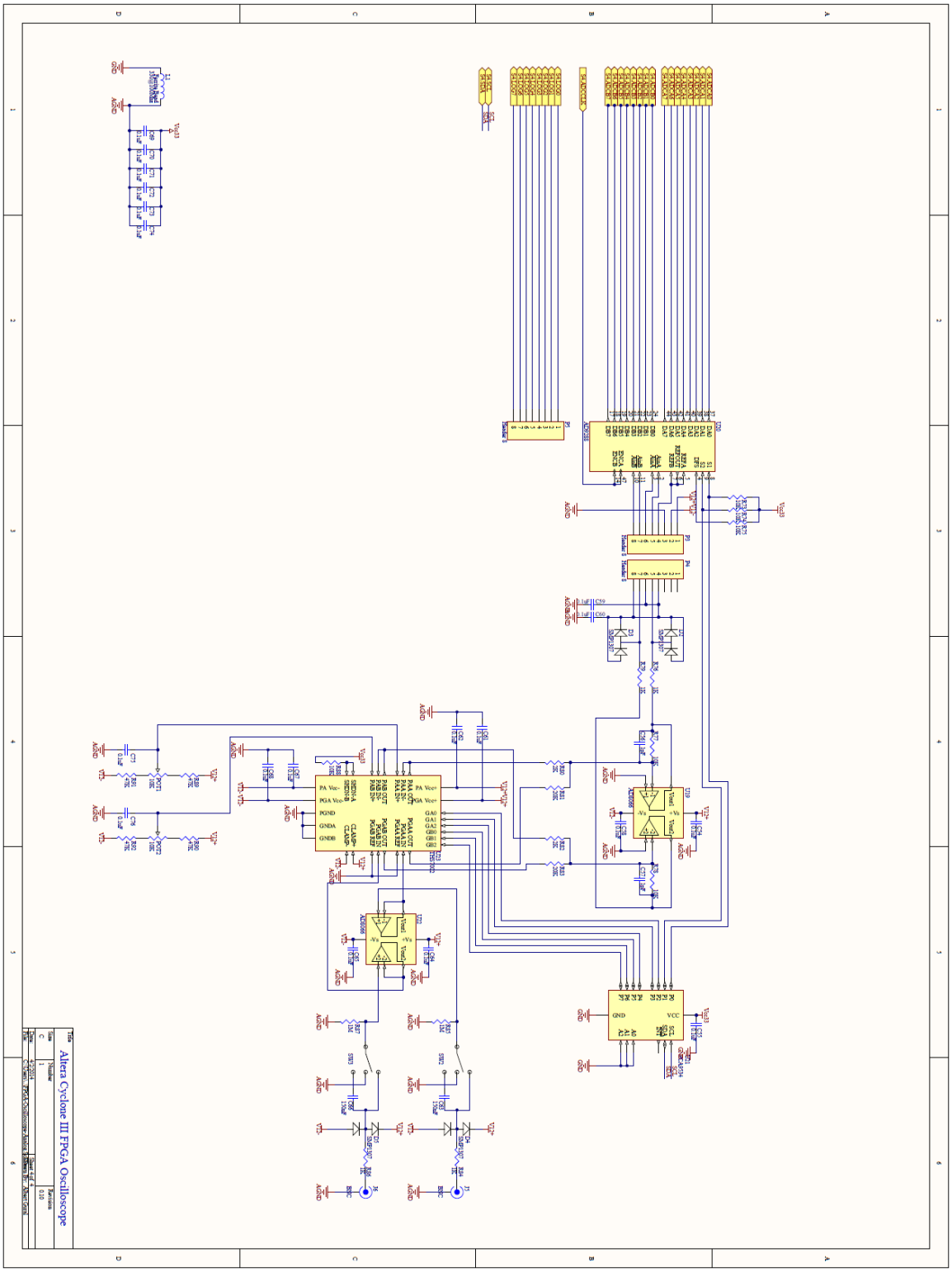

Figure B.2: Analog section schematic diagram

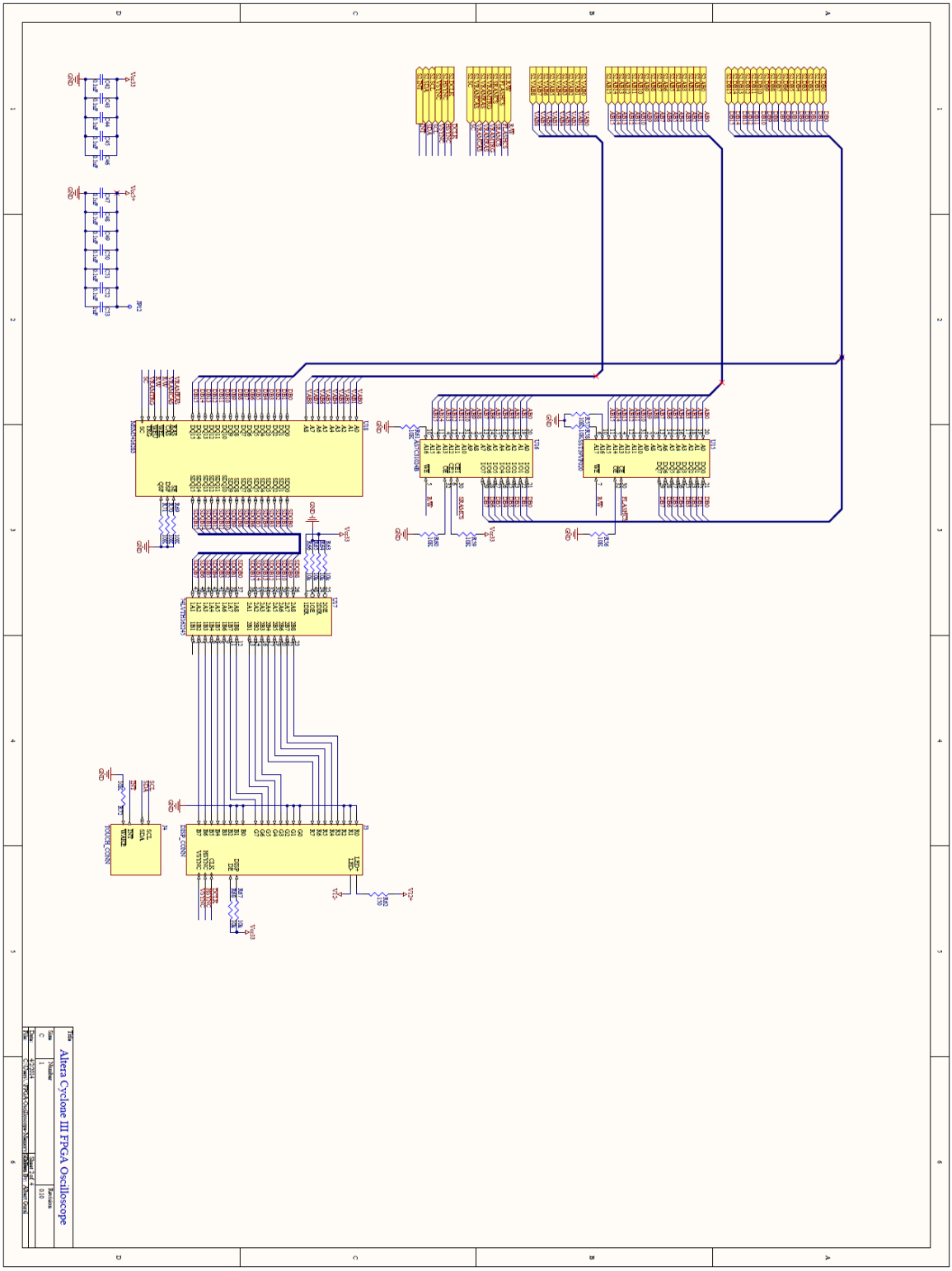

Figure B.3: Memory schematic diagram

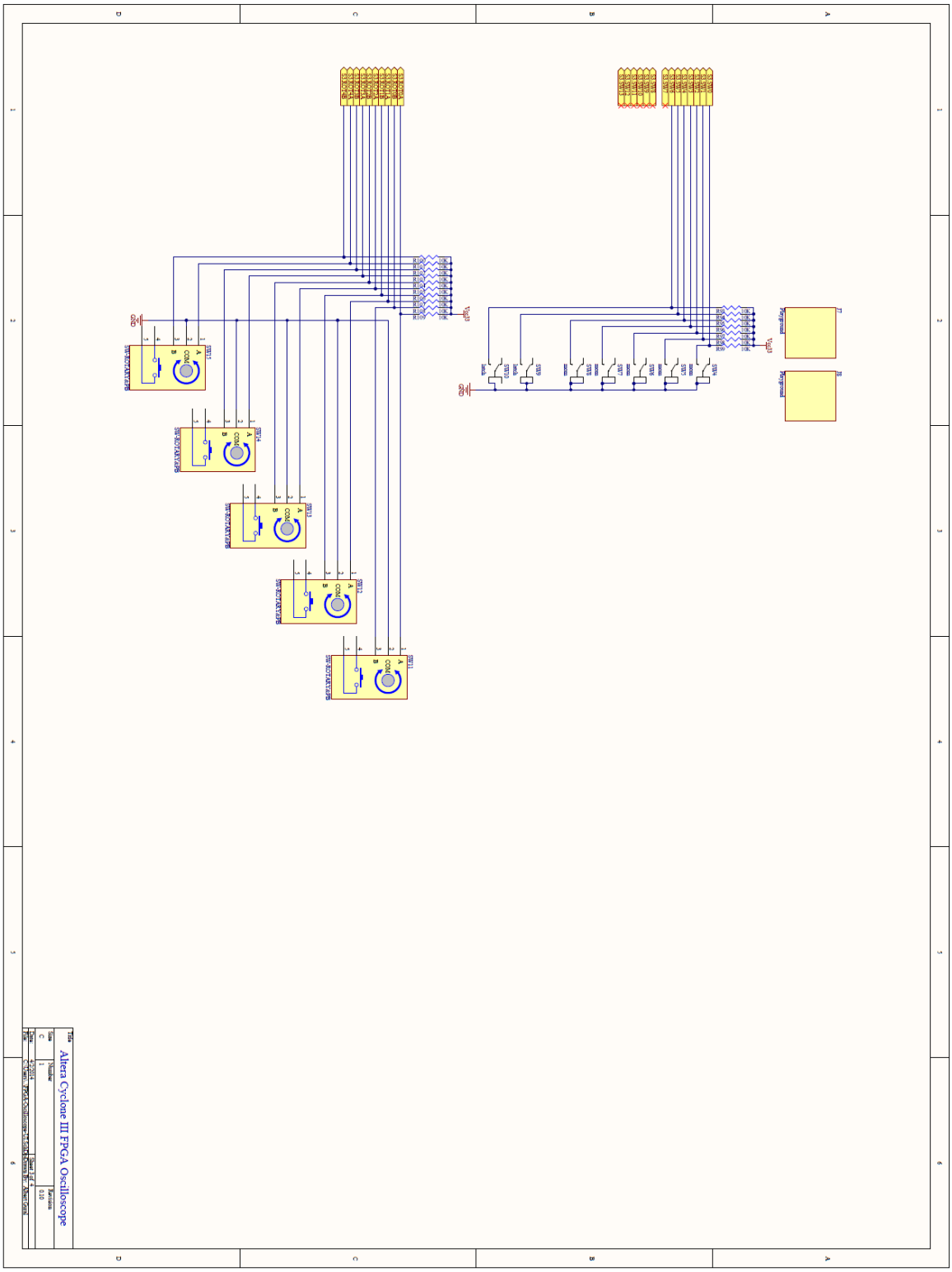

Figure B.4: Switches schematic diagram

Appendix C

Board Layouts

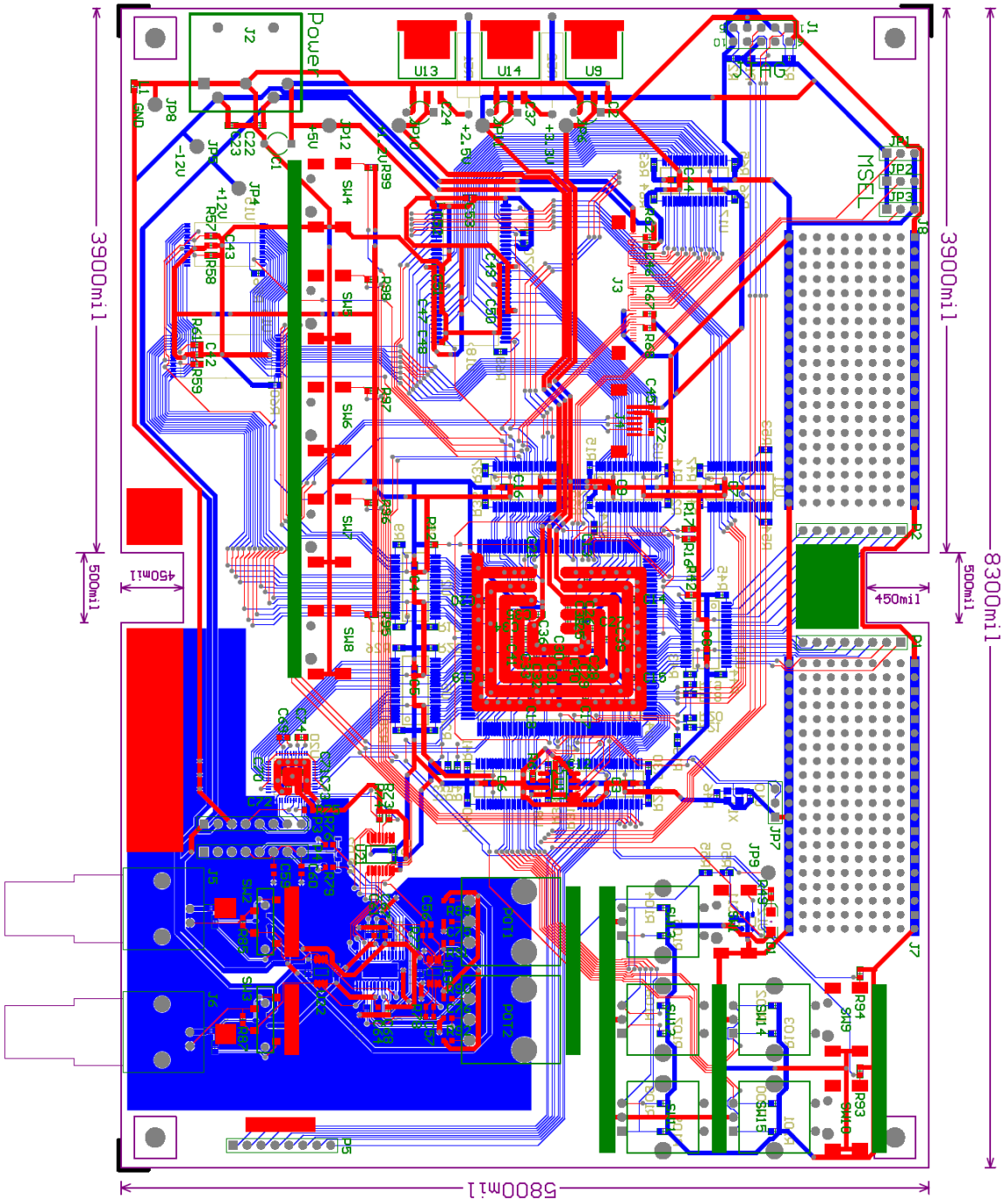

Figure C.1: Board layout (raw)

Appendix D

FPGA Block Diagram Files

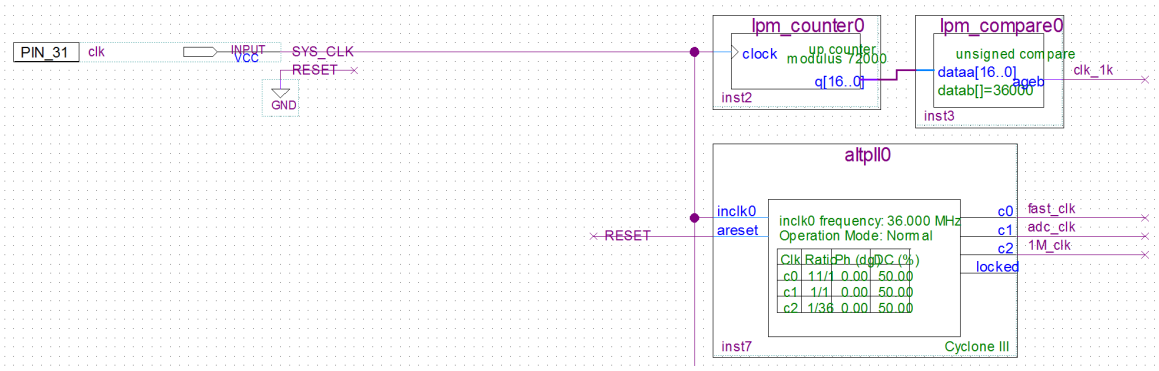

Figure D.1: FPGA main clocks

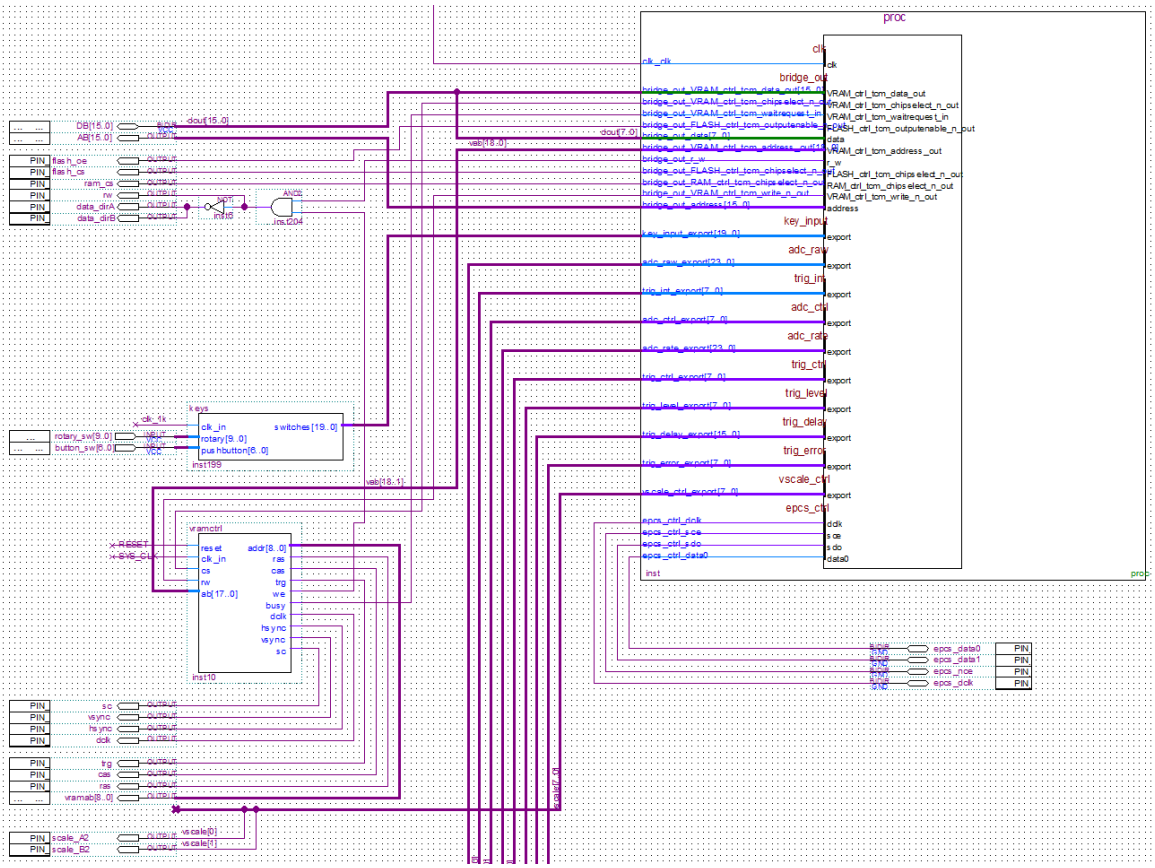

Figure D.2: FPGA NIOS II processor block

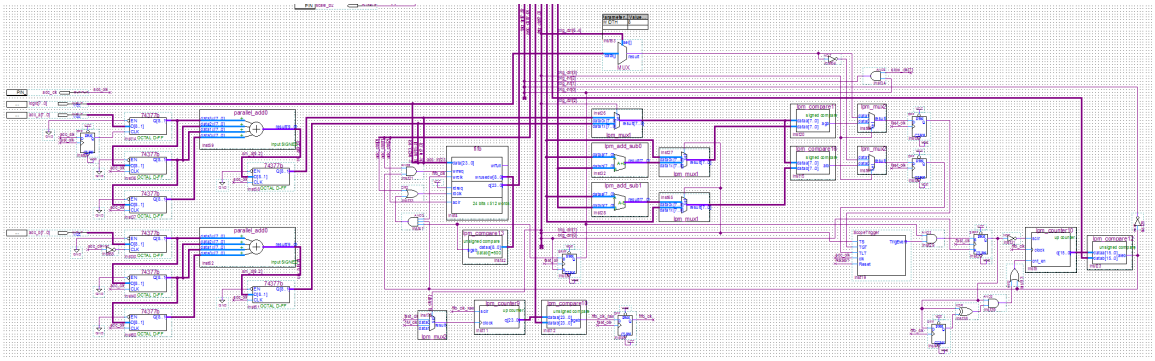

Figure D.3: FPGA overview of scope sample handling

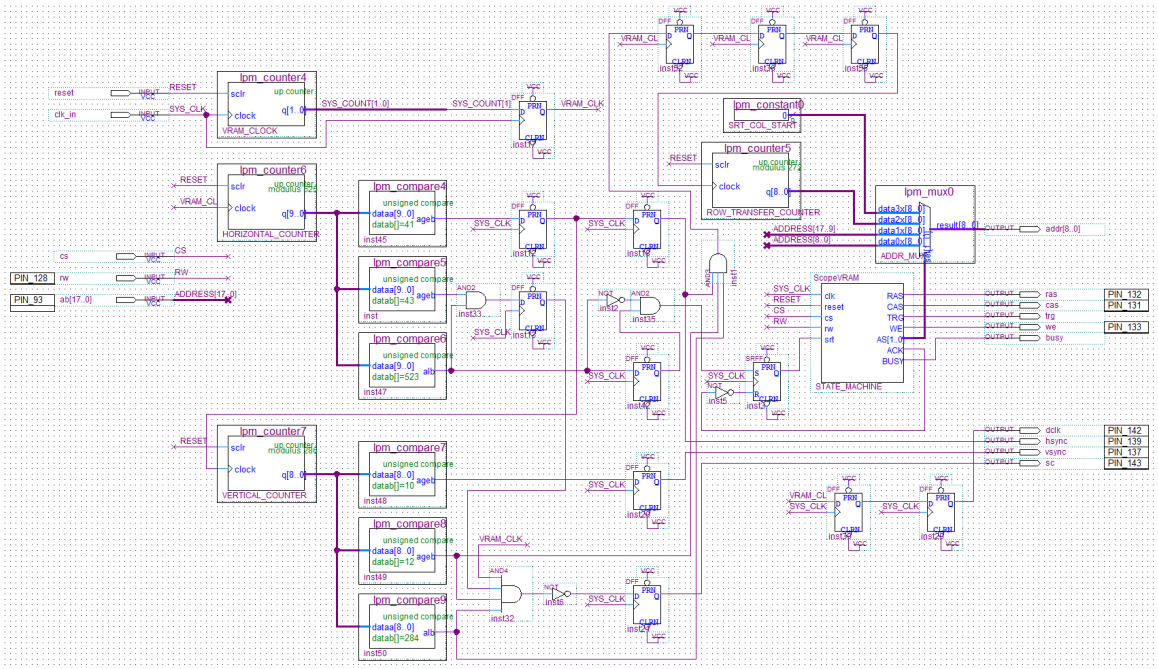

Figure D.4: FPGA overview of VRAM controller and display controller

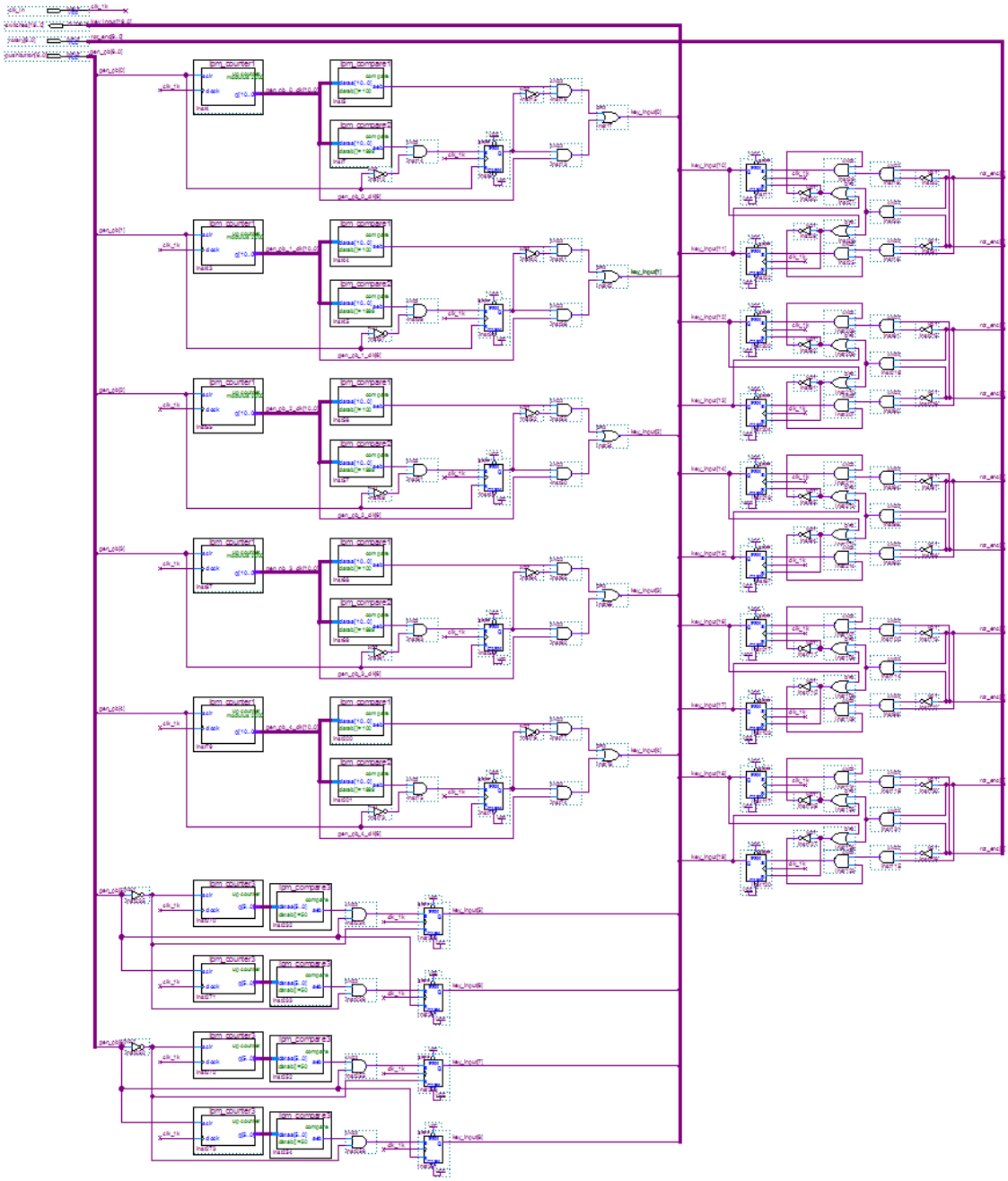

Figure D.5: FPGA overview of key and rotary encoder input

Appendix E

Memory Maps

	PROC.data_master	PROC.instruction_master
PROC.jtag_debug_module	0x0016 1800 - 0x0016 1fff	0x0016 1800 - 0x0016 1fff
RAM_ctrl.uas	0x0012 0000 - 0x0012 ffff	0x0012 0000 - 0x0012 ffff
FLASH_ctrl.uas	0x0013 0000 - 0x0013 ffff	0x0013 0000 - 0x0013 ffff
ONCHIP_mem.s1	0x0015 0000 - 0x0015 bfff	0x0015 0000 - 0x0015 bfff
sysid_qsys_0.control_slave	0x0016 2208 - 0x0016 220f	0x0016 2208 - 0x0016 220f
key_input.s1	0x0016 21f0 - 0x0016 21ff	0x0016 21f0 - 0x0016 21ff
VRAM_ctrl.uas	0x0008 0000 - 0x000f ffff	0x0008 0000 - 0x000f ffff
ADC_raw.s1	0x0016 21e0 - 0x0016 21ef	0x0016 21e0 - 0x0016 21ef
TRIG_int.s1	0x0016 2160 - 0x0016 217f	0x0016 2160 - 0x0016 217f
ADC_ctrl.s1	0x0016 2140 - 0x0016 215f	0x0016 2140 - 0x0016 215f
ADC_rate.s1	0x0016 21d0 - 0x0016 21df	0x0016 21d0 - 0x0016 21df
TRIG_ctrl.s1	0x0016 2120 - 0x0016 213f	0x0016 2120 - 0x0016 213f
TRIG_level.s1	0x0016 2100 - 0x0016 211f	0x0016 2100 - 0x0016 211f
TRIG_delay.s1	0x0016 20e0 - 0x0016 20ff	0x0016 20e0 - 0x0016 20ff
TRIG_error.s1	0x0016 21c0 - 0x0016 21cf	0x0016 21c0 - 0x0016 21cf
VSCALE_ctrl.s1	0x0016 20c0 - 0x0016 20df	0x0016 20c0 - 0x0016 20df
EPCS_ctrl.epcs_control_port	0x0016 1000 - 0x0016 17ff	0x0016 1000 - 0x0016 17ff

Figure E.1: NIOS II address memory map

Appendix F

Control Registers

```
/*
 * system.h - SOPC Builder system and BSP software package information
 *
 * Machine generated for CPU 'PROC' in SOPC Builder design 'proc'
 * SOPC Builder design path: E:/agural/osc/proc.sopcinfo
 *
 * Generated: Sat Jun 14 01:48:11 GMT-08:00 2014
 */

/*
 * DO NOT MODIFY THIS FILE
 *
 * Changing this file will have subtle consequences
 * which will almost certainly lead to a nonfunctioning
 * system. If you do modify this file, be aware that your
 * changes will be overwritten and lost when this file
 * is generated again.
 *
 * DO NOT MODIFY THIS FILE
 */

/*
 * License Agreement
 *
 * Copyright (c) 2008
 * Altera Corporation, San Jose, California, USA.
 * All rights reserved.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and associated documentation files (the "Software"),
 * to deal in the Software without restriction, including without limitation
 * the rights to use, copy, modify, merge, publish, distribute, sublicense,
 * and/or sell copies of the Software, and to permit persons to whom the
 * Software is furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
 * DEALINGS IN THE SOFTWARE.
 *
 * This agreement shall be governed in all respects by the laws of the State
 * of California and by the laws of the United States of America.
 */

#ifndef __SYSTEM_H_
#define __SYSTEM_H_

/* Include definitions from linker script generator */
#include "linker.h"

```

```
57
58 /*
59  * ADC_ctrl configuration
60  *
61  */
62
63 #define ADC_CTRL_BASE 0x162140
64 #define ADC_CTRL_BIT_CLEARING_EDGE_REGISTER 0
65 #define ADC_CTRL_BIT_MODIFYING_OUTPUT_REGISTER 1
66 #define ADC_CTRL_CAPTURE 0
67 #define ADC_CTRL_DATA_WIDTH 8
68 #define ADC_CTRL_DO_TEST_BENCH_WIRING 0
69 #define ADC_CTRL_DRIVEN_SIM_VALUE 0
70 #define ADC_CTRL_EDGE_TYPE "NONE"
71 #define ADC_CTRL_FREQ 36000000
72 #define ADC_CTRL_HAS_IN 0
73 #define ADC_CTRL_HAS_OUT 1
74 #define ADC_CTRL_HAS_TRI 0
75 #define ADC_CTRL_IRQ -1
76 #define ADC_CTRL_IRQ_INTERRUPT_CONTROLLER_ID -1
77 #define ADC_CTRL_IRQ_TYPE "NONE"
78 #define ADC_CTRL_NAME "/dev/ADC_ctrl"
79 #define ADC_CTRL_RESET_VALUE 0
80 #define ADC_CTRL_SPAN 32
81 #define ADC_CTRL_TYPE "altera_avalon_pio"
82 #define ALT_MODULE_CLASS_ADC_ctrl altera_avalon_pio
83
84
85 /*
86  * ADC_rate configuration
87  *
88  */
89
90 #define ADC_RATE_BASE 0x1621d0
91 #define ADC_RATE_BIT_CLEARING_EDGE_REGISTER 0
92 #define ADC_RATE_BIT_MODIFYING_OUTPUT_REGISTER 0
93 #define ADC_RATE_CAPTURE 0
94 #define ADC_RATE_DATA_WIDTH 24
95 #define ADC_RATE_DO_TEST_BENCH_WIRING 0
96 #define ADC_RATE_DRIVEN_SIM_VALUE 0
97 #define ADC_RATE_EDGE_TYPE "NONE"
98 #define ADC_RATE_FREQ 36000000
99 #define ADC_RATE_HAS_IN 0
100 #define ADC_RATE_HAS_OUT 1
101 #define ADC_RATE_HAS_TRI 0
102 #define ADC_RATE_IRQ -1
103 #define ADC_RATE_IRQ_INTERRUPT_CONTROLLER_ID -1
104 #define ADC_RATE_IRQ_TYPE "NONE"
105 #define ADC_RATE_NAME "/dev/ADC_rate"
106 #define ADC_RATE_RESET_VALUE 0
107 #define ADC_RATE_SPAN 16
108 #define ADC_RATE_TYPE "altera_avalon_pio"
109 #define ALT_MODULE_CLASS_ADC_rate altera_avalon_pio
110
111
112 /*
113  * ADC_raw configuration
114  *
115  */
116
117 #define ADC_RAW_BASE 0x1621e0
118 #define ADC_RAW_BIT_CLEARING_EDGE_REGISTER 0
119 #define ADC_RAW_BIT_MODIFYING_OUTPUT_REGISTER 0
120 #define ADC_RAW_CAPTURE 0
121 #define ADC_RAW_DATA_WIDTH 24
122 #define ADC_RAW_DO_TEST_BENCH_WIRING 0
123 #define ADC_RAW_DRIVEN_SIM_VALUE 0
124 #define ADC_RAW_EDGE_TYPE "NONE"
125 #define ADC_RAW_FREQ 36000000
126 #define ADC_RAW_HAS_IN 1
127 #define ADC_RAW_HAS_OUT 0
128 #define ADC_RAW_HAS_TRI 0
129 #define ADC_RAW_IRQ -1
130 #define ADC_RAW_IRQ_INTERRUPT_CONTROLLER_ID -1
131 #define ADC_RAW_IRQ_TYPE "NONE"
132 #define ADC_RAW_NAME "/dev/ADC_raw"
133 #define ADC_RAW_RESET_VALUE 0
134 #define ADC_RAW_SPAN 16
135 #define ADC_RAW_TYPE "altera_avalon_pio"
136 #define ALT_MODULE_CLASS_ADC_raw altera_avalon_pio
137
```

```
138
139 /*
140  * CPU configuration
141  *
142  */
143
144 #define ALT_CPU_ARCHITECTURE "altera_nios2_qsys"
145 #define ALT_CPU_BIG_ENDIAN 0
146 #define ALT_CPU_BREAK_ADDR 0x00161820
147 #define ALT_CPU_CPU_FREQ 36000000u
148 #define ALT_CPU_CPU_ID_SIZE 1
149 #define ALT_CPU_CPU_ID_VALUE 0x00000000
150 #define ALT_CPU_CPU_IMPLEMENTATION "small"
151 #define ALT_CPU_DATA_ADDR_WIDTH 0x15
152 #define ALT_CPU_DCACHE_LINE_SIZE 0
153 #define ALT_CPU_DCACHE_LINE_SIZE_LOG2 0
154 #define ALT_CPU_DCACHE_SIZE 0
155 #define ALT_CPU_EXCEPTION_ADDR 0x00150020
156 #define ALT_CPU_FLUSHDA_SUPPORTED
157 #define ALT_CPU_FREQ 36000000
158 #define ALT_CPU_HARDWARE_DIVIDE_PRESENT 1
159 #define ALT_CPU_HARDWARE_MULTIPLY_PRESENT 1
160 #define ALT_CPU_HARDWARE_MULX_PRESENT 0
161 #define ALT_CPU_HAS_DEBUG_CORE 1
162 #define ALT_CPU_HAS_DEBUG_STUB
163 #define ALT_CPU_HAS_JMPI_INSTRUCTION
164 #define ALT_CPU_ICACHE_LINE_SIZE 32
165 #define ALT_CPU_ICACHE_LINE_SIZE_LOG2 5
166 #define ALT_CPU_ICACHE_SIZE 4096
167 #define ALT_CPU_INST_ADDR_WIDTH 0x15
168 #define ALT_CPU_NAME "PROC"
169 #define ALT_CPU_RESET_ADDR 0x00150000
170
171
172 /*
173  * CPU configuration (with legacy prefix - don't use these anymore)
174  *
175  */
176
177 #define NIOS2_BIG_ENDIAN 0
178 #define NIOS2_BREAK_ADDR 0x00161820
179 #define NIOS2_CPU_FREQ 36000000u
180 #define NIOS2_CPU_ID_SIZE 1
181 #define NIOS2_CPU_ID_VALUE 0x00000000
182 #define NIOS2_CPU_IMPLEMENTATION "small"
183 #define NIOS2_DATA_ADDR_WIDTH 0x15
184 #define NIOS2_DCACHE_LINE_SIZE 0
185 #define NIOS2_DCACHE_LINE_SIZE_LOG2 0
186 #define NIOS2_DCACHE_SIZE 0
187 #define NIOS2_EXCEPTION_ADDR 0x00150020
188 #define NIOS2_FLUSHDA_SUPPORTED
189 #define NIOS2_HARDWARE_DIVIDE_PRESENT 1
190 #define NIOS2_HARDWARE_MULTIPLY_PRESENT 1
191 #define NIOS2_HARDWARE_MULX_PRESENT 0
192 #define NIOS2_HAS_DEBUG_CORE 1
193 #define NIOS2_HAS_DEBUG_STUB
194 #define NIOS2_HAS_JMPI_INSTRUCTION
195 #define NIOS2_ICACHE_LINE_SIZE 32
196 #define NIOS2_ICACHE_LINE_SIZE_LOG2 5
197 #define NIOS2_ICACHE_SIZE 4096
198 #define NIOS2_INST_ADDR_WIDTH 0x15
199 #define NIOS2_RESET_ADDR 0x00150000
200
201
202 /*
203  * Define for each module class mastered by the CPU
204  *
205  */
206
207 #define __ALTERA_AVALON_EPCS_FLASH_CONTROLLER
208 #define __ALTERA_AVALON_ONCHIP_MEMORY2
209 #define __ALTERA_AVALON_PIO
210 #define __ALTERA_AVALON_SYSID_QSYS
211 #define __ALTERA_GENERIC_TRISTATE_CONTROLLER
212 #define __ALTERA_NIOS2_QSYS
213
214
215 /*
216  * EPCS_ctrl configuration
217  *
218  */
```

```
219
220 #define ALT_MODULE_CLASS_EPCS_ctrl altera_avalon_epcs_flash_controller
221 #define EPCS_CTRL_BASE 0x161000
222 #define EPCS_CTRL_IRQ -1
223 #define EPCS_CTRL_IRQ_INTERRUPT_CONTROLLER_ID -1
224 #define EPCS_CTRL_NAME "/dev/EPCS_ctrl"
225 #define EPCS_CTRL_REGISTER_OFFSET 1024
226 #define EPCS_CTRL_SPAN 2048
227 #define EPCS_CTRL_TYPE "altera_avalon_epcs_flash_controller"
228
229
230 /*
231  * FLASH_ctrl configuration
232  *
233  */
234
235 #define ALT_MODULE_CLASS_FLASH_ctrl altera_generic_tristate_controller
236 #define FLASH_CTRL_BASE 0x130000
237 #define FLASH_CTRL_IRQ -1
238 #define FLASH_CTRL_IRQ_INTERRUPT_CONTROLLER_ID -1
239 #define FLASH_CTRL_NAME "/dev/FLASH_ctrl"
240 #define FLASH_CTRL_SIZE 65536
241 #define FLASH_CTRL_SPAN 65536
242 #define FLASH_CTRL_TYPE "altera_generic_tristate_controller"
243
244
245 /*
246  * ONCHIP_mem configuration
247  *
248  */
249
250 #define ALT_MODULE_CLASS_ONCHIP_mem altera_avalon_onchip_memory2
251 #define ONCHIP_MEM_ALLOW_IN_SYSTEM_MEMORY_CONTENT_EDITOR 0
252 #define ONCHIP_MEM_ALLOW_MRAM_SIM_CONTENTS_ONLY_FILE 0
253 #define ONCHIP_MEM_BASE 0x150000
254 #define ONCHIP_MEM_CONTENTS_INFO ""
255 #define ONCHIP_MEM_DUAL_PORT 0
256 #define ONCHIP_MEM_GUI_RAM_BLOCK_TYPE "AUTO"
257 #define ONCHIP_MEM_INIT_CONTENTS_FILE "proc_ONCHIP_mem"
258 #define ONCHIP_MEM_INIT_MEM_CONTENT 1
259 #define ONCHIP_MEM_INSTANCE_ID "NONE"
260 #define ONCHIP_MEM_IRQ -1
261 #define ONCHIP_MEM_IRQ_INTERRUPT_CONTROLLER_ID -1
262 #define ONCHIP_MEM_NAME "/dev/ONCHIP_mem"
263 #define ONCHIP_MEM_NON_DEFAULT_INIT_FILE_ENABLED 0
264 #define ONCHIP_MEM_RAM_BLOCK_TYPE "AUTO"
265 #define ONCHIP_MEM_READ_DURING_WRITE_MODE "DONT_CARE"
266 #define ONCHIP_MEM_SINGLE_CLOCK_OP 0
267 #define ONCHIP_MEM_SIZE_MULTIPLE 1
268 #define ONCHIP_MEM_SIZE_VALUE 49152
269 #define ONCHIP_MEM_SPAN 49152
270 #define ONCHIP_MEM_TYPE "altera_avalon_onchip_memory2"
271 #define ONCHIP_MEM_WRITABLE 1
272
273
274 /*
275  * RAM_ctrl configuration
276  *
277  */
278
279 #define ALT_MODULE_CLASS_RAM_ctrl altera_generic_tristate_controller
280 #define RAM_CTRL_BASE 0x120000
281 #define RAM_CTRL_IRQ -1
282 #define RAM_CTRL_IRQ_INTERRUPT_CONTROLLER_ID -1
283 #define RAM_CTRL_NAME "/dev/RAM_ctrl"
284 #define RAM_CTRL_SIZE 65536
285 #define RAM_CTRL_SPAN 65536
286 #define RAM_CTRL_TYPE "altera_generic_tristate_controller"
287
288
289 /*
290  * System configuration
291  *
292  */
293
294 #define ALT_DEVICE_FAMILY "Cyclone III"
295 #define ALT_ENHANCED_INTERRUPT_API_PRESENT
296 #define ALT_IRQ_BASE NULL
297 #define ALT_LOG_PORT "/dev/null"
298 #define ALT_LOG_PORT_BASE 0x0
299 #define ALT_LOG_PORT_DEV null
```

```
#define ALT_LOG_PORT_TYPE ""
#define ALT_NUM_EXTERNAL_INTERRUPT_CONTROLLERS 0
#define ALT_NUM_INTERNAL_INTERRUPT_CONTROLLERS 1
#define ALT_NUM_INTERRUPT_CONTROLLERS 1
#define ALT_STDERR "/dev/null"
#define ALT_STDERR_BASE 0x0
#define ALT_STDERR_DEV null
#define ALT_STDERR_TYPE ""
#define ALT_STDIN "/dev/null"
#define ALT_STDIN_BASE 0x0
#define ALT_STDIN_DEV null
#define ALT_STDIN_TYPE ""
#define ALT_STDOUT "/dev/null"
#define ALT_STDOUT_BASE 0x0
#define ALT_STDOUT_DEV null
#define ALT_STDOUT_TYPE ""
#define ALT_SYSTEM_NAME "proc"


/*
 * TRIG_ctrl configuration
 *
 */

#define ALT_MODULE_CLASS_TRIG_ctrl altera_avalon_pio
#define TRIG_CTRL_BASE 0x162120
#define TRIG_CTRL_BIT_CLEARING_EDGE_REGISTER 0
#define TRIG_CTRL_BIT_MODIFYING_OUTPUT_REGISTER 1
#define TRIG_CTRL_CAPTURE 0
#define TRIG_CTRL_DATA_WIDTH 8
#define TRIG_CTRL_DO_TEST_BENCH_WIRING 0
#define TRIG_CTRL_DRIVEN_SIM_VALUE 0
#define TRIG_CTRL_EDGE_TYPE "NONE"
#define TRIG_CTRL_FREQ 36000000
#define TRIG_CTRL_HAS_IN 0
#define TRIG_CTRL_HAS_OUT 1
#define TRIG_CTRL_HAS_TRI 0
#define TRIG_CTRL_IRQ -1
#define TRIG_CTRL_IRQ_INTERRUPT_CONTROLLER_ID -1
#define TRIG_CTRL_IRQ_TYPE "NONE"
#define TRIG_CTRL_NAME "/dev/TRIG_ctrl"
#define TRIG_CTRL_RESET_VALUE 0
#define TRIG_CTRL_SPAN 32
#define TRIG_CTRL_TYPE "altera_avalon_pio"


/*
 * TRIG_delay configuration
 *
 */

#define ALT_MODULE_CLASS_TRIG_delay altera_avalon_pio
#define TRIG_DELAY_BASE 0x1620e0
#define TRIG_DELAY_BIT_CLEARING_EDGE_REGISTER 0
#define TRIG_DELAY_BIT_MODIFYING_OUTPUT_REGISTER 1
#define TRIG_DELAY_CAPTURE 0
#define TRIG_DELAY_DATA_WIDTH 16
#define TRIG_DELAY_DO_TEST_BENCH_WIRING 0
#define TRIG_DELAY_DRIVEN_SIM_VALUE 0
#define TRIG_DELAY_EDGE_TYPE "NONE"
#define TRIG_DELAY_FREQ 36000000
#define TRIG_DELAY_HAS_IN 0
#define TRIG_DELAY_HAS_OUT 1
#define TRIG_DELAY_HAS_TRI 0
#define TRIG_DELAY_IRQ -1
#define TRIG_DELAY_IRQ_INTERRUPT_CONTROLLER_ID -1
#define TRIG_DELAY_IRQ_TYPE "NONE"
#define TRIG_DELAY_NAME "/dev/TRIG_delay"
#define TRIG_DELAY_RESET_VALUE 0
#define TRIG_DELAY_SPAN 32
#define TRIG_DELAY_TYPE "altera_avalon_pio"


/*
 * TRIG_error configuration
 *
 */

#define ALT_MODULE_CLASS_TRIG_error altera_avalon_pio
#define TRIG_ERROR_BASE 0x1621c0
#define TRIG_ERROR_BIT_CLEARING_EDGE_REGISTER 0
```

```
#define TRIG_ERROR_BIT_MODIFYING_OUTPUT_REGISTER 0
#define TRIG_ERROR_CAPTURE 0
#define TRIG_ERROR_DATA_WIDTH 8
#define TRIG_ERROR_DO_TEST_BENCH_WIRING 0
#define TRIG_ERROR_DRIVEN_SIM_VALUE 0
#define TRIG_ERROR_EDGE_TYPE "NONE"
#define TRIG_ERROR_FREQ 36000000
#define TRIG_ERROR_HAS_IN 0
#define TRIG_ERROR_HAS_OUT 1
#define TRIG_ERROR_HAS_TRI 0
#define TRIG_ERROR_IRQ -1
#define TRIG_ERROR_IRQ_INTERRUPT_CONTROLLER_ID -1
#define TRIG_ERROR_IRQ_TYPE "NONE"
#define TRIG_ERROR_NAME "/dev/TRIG_error"
#define TRIG_ERROR_RESET_VALUE 0
#define TRIG_ERROR_SPAN 16
#define TRIG_ERROR_TYPE "altera_avalon_pio"


/*
 * TRIG_int configuration
 *
 */

#define ALT_MODULE_CLASS_TRIG_int altera_avalon_pio
#define TRIG_INT_BASE 0x162160
#define TRIG_INT_BIT_CLEARING_EDGE_REGISTER 1
#define TRIG_INT_BIT_MODIFYING_OUTPUT_REGISTER 1
#define TRIG_INT_CAPTURE 1
#define TRIG_INT_DATA_WIDTH 8
#define TRIG_INT_DO_TEST_BENCH_WIRING 0
#define TRIG_INT_DRIVEN_SIM_VALUE 0
#define TRIG_INT_EDGE_TYPE "RISING"
#define TRIG_INT_FREQ 36000000
#define TRIG_INT_HAS_IN 1
#define TRIG_INT_HAS_OUT 0
#define TRIG_INT_HAS_TRI 0
#define TRIG_INT_IRQ 6
#define TRIG_INT_IRQ_INTERRUPT_CONTROLLER_ID 0
#define TRIG_INT_IRQ_TYPE "EDGE"
#define TRIG_INT_NAME "/dev/TRIG_int"
#define TRIG_INT_RESET_VALUE 0
#define TRIG_INT_SPAN 32
#define TRIG_INT_TYPE "altera_avalon_pio"


/*
 * TRIG_level configuration
 *
 */

#define ALT_MODULE_CLASS_TRIG_level altera_avalon_pio
#define TRIG_LEVEL_BASE 0x162100
#define TRIG_LEVEL_BIT_CLEARING_EDGE_REGISTER 0
#define TRIG_LEVEL_BIT_MODIFYING_OUTPUT_REGISTER 1
#define TRIG_LEVEL_CAPTURE 0
#define TRIG_LEVEL_DATA_WIDTH 8
#define TRIG_LEVEL_DO_TEST_BENCH_WIRING 0
#define TRIG_LEVEL_DRIVEN_SIM_VALUE 0
#define TRIG_LEVEL_EDGE_TYPE "NONE"
#define TRIG_LEVEL_FREQ 36000000
#define TRIG_LEVEL_HAS_IN 0
#define TRIG_LEVEL_HAS_OUT 1
#define TRIG_LEVEL_HAS_TRI 0
#define TRIG_LEVEL_IRQ -1
#define TRIG_LEVEL_IRQ_INTERRUPT_CONTROLLER_ID -1
#define TRIG_LEVEL_IRQ_TYPE "NONE"
#define TRIG_LEVEL_NAME "/dev/TRIG_level"
#define TRIG_LEVEL_RESET_VALUE 0
#define TRIG_LEVEL_SPAN 32
#define TRIG_LEVEL_TYPE "altera_avalon_pio"


/*
 * VRAM_ctrl configuration
 *
 */

#define ALT_MODULE_CLASS_VRAM_ctrl altera_generic_tristate_controller
#define VRAM_CTRL_BASE 0x80000
#define VRAM_CTRL_IRQ -1
```

```
#define VRAM_CTRL_IRQ_INTERRUPT_CONTROLLER_ID -1
#define VRAM_CTRL_NAME "/dev/VRAM_ctrl"
#define VRAM_CTRL_SPAN 524288
#define VRAM_CTRL_TYPE "altera_generic_tristate_controller"


/*
 * VSCALE_ctrl configuration
 *
 */

#define ALT_MODULE_CLASS_VSCALE_ctrl altera_avalon_pio
#define VSCALE_CTRL_BASE 0x1620c0
#define VSCALE_CTRL_BIT_CLEARING_EDGE_REGISTER 0
#define VSCALE_CTRL_BIT_MODIFYING_OUTPUT_REGISTER 1
#define VSCALE_CTRL_CAPTURE 0
#define VSCALE_CTRL_DATA_WIDTH 8
#define VSCALE_CTRL_DO_TEST_BENCH_WIRING 0
#define VSCALE_CTRL_DRIVEN_SIM_VALUE 0
#define VSCALE_CTRL_EDGE_TYPE "NONE"
#define VSCALE_CTRL_FREQ 36000000
#define VSCALE_CTRL_HAS_IN 0
#define VSCALE_CTRL_HAS_OUT 1
#define VSCALE_CTRL_HAS_TRI 0
#define VSCALE_CTRL_IRQ -1
#define VSCALE_CTRL_IRQ_INTERRUPT_CONTROLLER_ID -1
#define VSCALE_CTRL_IRQ_TYPE "NONE"
#define VSCALE_CTRL_NAME "/dev/VSCALE_ctrl"
#define VSCALE_CTRL_RESET_VALUE 0
#define VSCALE_CTRL_SPAN 32
#define VSCALE_CTRL_TYPE "altera_avalon_pio"


/*
 * hal configuration
 *
 */

#define ALT_MAX_FD 32
#define ALT_SYS_CLK none
#define ALT_TIMESTAMP_CLK none


/*
 * key_input configuration
 *
 */

#define ALT_MODULE_CLASS_key_input altera_avalon_pio
#define KEY_INPUT_BASE 0x1621f0
#define KEY_INPUT_BIT_CLEARING_EDGE_REGISTER 1
#define KEY_INPUT_BIT_MODIFYING_OUTPUT_REGISTER 0
#define KEY_INPUT_CAPTURE 1
#define KEY_INPUT_DATA_WIDTH 20
#define KEY_INPUT_DO_TEST_BENCH_WIRING 0
#define KEY_INPUT_DRIVEN_SIM_VALUE 0
#define KEY_INPUT_EDGE_TYPE "RISING"
#define KEY_INPUT_FREQ 36000000
#define KEY_INPUT_HAS_IN 1
#define KEY_INPUT_HAS_OUT 0
#define KEY_INPUT_HAS_TRI 0
#define KEY_INPUT_IRQ 5
#define KEY_INPUT_IRQ_INTERRUPT_CONTROLLER_ID 0
#define KEY_INPUT_IRQ_TYPE "EDGE"
#define KEY_INPUT_NAME "/dev/key_input"
#define KEY_INPUT_RESET_VALUE 0
#define KEY_INPUT_SPAN 16
#define KEY_INPUT_TYPE "altera_avalon_pio"


/*
 * sysid_qsys_0 configuration
 *
 */

#define ALT_MODULE_CLASS_sysid_qsys_0 altera_avalon_sysid_qsys
#define SYSID_QSYS_0_BASE 0x162208
#define SYSID_QSYS_0_ID -1431655766
#define SYSID_QSYS_0_IRQ -1
#define SYSID_QSYS_0_IRQ_INTERRUPT_CONTROLLER_ID -1
#define SYSID_QSYS_0_NAME "/dev/sysid_qsys_0"
```

```
543 #define SYSID_QSYS_0_SPAN 8
544 #define SYSID_QSYS_0_TIMESTAMP 1402739081
545 #define SYSID_QSYS_0_TYPE "altera_avalon_sysid_qsys"
546
547 #endif /* __SYSTEM_H_ */
```

Appendix G

VHDL Code

G.1 VRAM State Machine Code

```
1  ----------------------------------------------------------------------------
2  --
3  --  Oscilloscope VRAM State Machine
4  --
5  --  This is an implementation of a VRAM State machine for a digital scope in
6  --  VHDL.  There are three inputs to the system, one selects the trigger
7  --  slope and the other two determine the relationship between the trigger
8  --  level and the signal level.  The only output is a trigger signal which
9  --  indicates a trigger event has occurred.
10 --
11 --  The file contains multiple architectures for a Moore state machine
12 --  implementation to demonstrate the different ways of building a state
13 --  machine.
14 --
15 --
16 --  Revision History:
17 --     2014/02/23 Albert Gural      Created file and updated with VRAM
18 --                                  state machine.
19 --       2014/06/07 Albert Gural        Added idle states to improve r/w speeds.
20 --
21 ----------------------------------------------------------------------------
22
23
24 -- bring in the necessary packages
25 library  ieee;
26 use  ieee.std_logic_1164.all;
27
28
29 --
30 --  Oscilloscope VRAM entity declaration
31 --
32
33 entity  ScopeVRAM  is
34     port (
35         clk         :  in  std_logic;       -- clock input
36         reset       :  in  std_logic;       -- reset to idle state (active low)
37         cs          :  in  std_logic;       -- whether CPU is requesting R/W (active low)
38         rw          :  in  std_logic;       -- whether CPU is requesting R or W
39         srt         :  in  std_logic;       -- display requesting serial row transfer
40         RAS         :  out std_logic;       -- row address select output
41         CAS         :  out std_logic;       -- col address select output
42         TRG         :  out std_logic;       -- trigger output (active low)
43         WE          :  out std_logic;       -- write enable output (active low)
44         AS          :  out std_logic_vector(1 downto 0); -- address select
45                                                          -- 00 = low 9 bits
46                                                          -- 01 = high 9 bits
47                                                          -- 10 = row transfer
48                                                          -- 11 = col transfer (0)
49         ACK         :  out std_logic;       -- send acknowledge back to display driver
50         BUSY        :  out std_logic        -- busy signal to CPU for read/write
51     );
52 end  ScopeVRAM;
53
```

```
54 --
55 --  Oscilloscope VRAM Moore State Machine
56 --     State Assignment Architecture
57 --
58 --  This architecture just shows the basic state machine syntax when the state
59 --  assignments are made manually.  This is useful for minimizing output
60 --  decoding logic and avoiding glitches in the output (due to the decoding
61 --  logic).
62 --
63
64 architecture  assign_statebits  of  ScopeVRAM  is
65
66     subtype  states  is  std_logic_vector(12 downto 0);     -- state type
67
68     -- define the actual states as constants
69     -- bits: [RAS][CAS][TRG][WE][AS<2>][ACK][BUSY]
70     --       [type <000 IDLE, 010 READ, 011 WRITE, 100 REFRESH, 101 TRANSFER>]
71     --       [number <00, 01, ...>]
72     constant  IDLE      : states := "1111011100000";  -- idle (can accept R/W)
73     constant  IDLE2     : states := "1111011100001";  -- idle (can accept R/W)
74     constant  IDLE3     : states := "1111011100010";  -- idle (can accept R/W)
75     constant  IDLE4     : states := "1111011100011";  -- idle (can accept R/W, srt, refresh)
76     constant  READ1     : states := "1111011101000";  -- read cycle
77     constant  READ2     : states := "0111011101000";  -- read cycle
78     constant  READ3     : states := "0101001101000";  -- read cycle
79     constant  READ4     : states := "0001001101000";  -- read cycle
80     constant  READ5     : states := "0001001101001";  -- read cycle
81     constant  READ6     : states := "0001001001010";  -- read cycle
82     constant  READ7     : states := "1111001101000";  -- read cycle
83     constant  READ8     : states := "1111001101001";  -- read cycle
84     constant  READ9     : states := "1111001101010";  -- read cycle
85     constant  WRITE1    : states := "0111011101100";  -- write cycle
86     constant  WRITE2    : states := "0110001101100";  -- write cycle
87     constant  WRITE3    : states := "0010001101100";  -- write cycle
88     constant  WRITE4    : states := "0010001001101";  -- write cycle
89     constant  WRITE5    : states := "1111001101100";  -- write cycle
90     constant  WRITE6    : states := "1111001101101";  -- write cycle
91     constant  WRITE7    : states := "1111001101110";  -- write cycle
92     constant  TRANSFER1 : states := "1101101110100";  -- transfer cycle
93     constant  TRANSFER2 : states := "0101101110100";  -- transfer cycle
94     constant  TRANSFER3 : states := "0111111110100";  -- transfer cycle
95     constant  TRANSFER4 : states := "0011111110100";  -- transfer cycle
96     constant  TRANSFER5 : states := "1111010110100";  -- transfer cycle
97     constant  TRANSFER6 : states := "1111011110100";  -- transfer cycle
98     constant  REFRESH1  : states := "1111001110000";  -- refresh cycle
99     constant  REFRESH2  : states := "1111001110001";  -- refresh cycle
100    constant  REFRESH3  : states := "1011001110000";  -- refresh cycle
101    constant  REFRESH4  : states := "0011001110000";  -- refresh cycle
102    constant  REFRESH5  : states := "0011001110001";  -- refresh cycle
103    constant  REFRESH6  : states := "0011001110010";  -- refresh cycle
104    constant  REFRESH7  : states := "1111001110010";  -- refresh cycle
105    constant  REFRESH8  : states := "1111001110011";  -- refresh cycle
106
107
108    signal  CurrentState  :  states;    -- current state
109    signal  NextState     :  states;    -- next state
110
111 begin
112
113
114    -- the output is always the high bit of the state encoding
115    RAS  <= CurrentState(12);
116    CAS  <= CurrentState(11);
117    TRG  <= CurrentState(10);
118    WE   <= CurrentState(9);
119    AS   <= CurrentState(8 downto 7);
120    ACK  <= CurrentState(6);
121    BUSY <= CurrentState(5);
122
123    -- compute the next state (function of current state and inputs)
124
125    transition:  process (reset, cs, rw, srt, CurrentState)
126    begin
127
128        case  CurrentState  is          -- do the state transition/output
129
130              when  IDLE =>               -- in idle state, do transition
131              if  (cs = '0' and rw = '1')  then
132                  NextState <= READ1;         -- start read cycle
133              elsif  (cs = '0' and rw = '0')  then
134                  NextState <= WRITE1;        -- start write cycle
```

```
135                 else
136                     NextState <= IDLE2;      -- start refresh cycle
137                 end if;
138
139                 when  IDLE2 =>              -- in idle state, do transition
140                 if  (cs = '0' and rw = '1')  then
141                     NextState <= READ1;          -- start read cycle
142                 elsif  (cs = '0' and rw = '0')  then
143                     NextState <= WRITE1;         -- start write cycle
144                 else
145                     NextState <= IDLE3;      -- start refresh cycle
146                 end if;
147
148                 when  IDLE3 =>              -- in idle state, do transition
149                 if  (cs = '0' and rw = '1')  then
150                     NextState <= READ1;          -- start read cycle
151                 elsif  (cs = '0' and rw = '0')  then
152                     NextState <= WRITE1;         -- start write cycle
153                 else
154                     NextState <= IDLE4;      -- start refresh cycle
155                 end if;
156
157             when  IDLE4 =>              -- in idle state, do transition
158                 if  (cs = '0' and rw = '1')  then
159                     NextState <= READ1;          -- start read cycle
160                 elsif  (cs = '0' and rw = '0')  then
161                     NextState <= WRITE1;         -- start write cycle
162                 elsif  (srt = '1')  then
163                     NextState <= TRANSFER1;      -- start serial row transfer
164                 else
165                     NextState <= REFRESH1;       -- start refresh cycle
166                 end if;
167
168             when  READ1 =>              -- read cycle
169                 NextState <= READ2;     -- go to next part of read cycle
170
171             when  READ2 =>              -- read cycle
172                 NextState <= READ3;     -- go to next part of read cycle
173
174             when  READ3 =>              -- read cycle
175                 NextState <= READ4;     -- go to next part of read cycle
176
177             when  READ4 =>              -- read cycle
178                 NextState <= READ5;     -- go to next part of read cycle
179
180             when  READ5 =>              -- read cycle
181                 NextState <= READ6;     -- go to next part of read cycle
182
183             when  READ6 =>              -- read cycle
184                 NextState <= READ7;     -- go to next part of read cycle
185
186             when  READ7 =>              -- read cycle
187                 NextState <= READ8;     -- go to next part of read cycle
188
189             when  READ8 =>              -- read cycle
190                 NextState <= READ9;     -- go to next part of read cycle
191
192             when  READ9 =>              -- read cycle
193                 NextState <= IDLE;      -- go back to idle
194
195             when  WRITE1 =>             -- write cycle
196                 NextState <= WRITE2;    -- go to next part of write cycle
197
198             when  WRITE2 =>             -- write cycle
199                 NextState <= WRITE3;    -- go to next part of write cycle
200
201             when  WRITE3 =>             -- write cycle
202                 NextState <= WRITE4;    -- go to next part of write cycle
203
204             when  WRITE4 =>             -- write cycle
205                 NextState <= WRITE5;    -- go to next part of write cycle
206
207             when  WRITE5 =>             -- write cycle
208                 NextState <= WRITE6;    -- go to next part of write cycle
209
210             when  WRITE6 =>             -- write cycle
211                 NextState <= WRITE7;    -- go to next part of write cycle
212
213             when  WRITE7 =>             -- write cycle
214                 NextState <= IDLE;      -- go back to idle
215
```

```
216             when  TRANSFER1 =>          -- transfer cycle
217                 NextState <= TRANSFER2; -- go to next part of transfer cycle
218
219             when  TRANSFER2 =>          -- transfer cycle
220                 NextState <= TRANSFER3; -- go to next part of transfer cycle
221
222             when  TRANSFER3 =>          -- transfer cycle
223                 NextState <= TRANSFER4; -- go to next part of transfer cycle
224
225             when  TRANSFER4 =>          -- transfer cycle
226                 NextState <= TRANSFER5; -- go to next part of transfer cycle
227
228             when  TRANSFER5 =>          -- transfer cycle
229                 NextState <= TRANSFER6; -- go to next part of transfer cycle
230
231             when  TRANSFER6 =>          -- transfer cycle
232                 NextState <= IDLE;      -- go back to idle
233
234             when  REFRESH1 =>           -- refresh cycle
235                 NextState <= REFRESH2;  -- go to next part of refresh cycle
236
237             when  REFRESH2 =>           -- refresh cycle
238                 NextState <= REFRESH3;  -- go to next part of refresh cycle
239
240             when  REFRESH3 =>           -- refresh cycle
241                 NextState <= REFRESH4;  -- go to next part of refresh cycle
242
243             when  REFRESH4 =>           -- refresh cycle
244                 NextState <= REFRESH5;  -- go to next part of refresh cycle
245
246             when  REFRESH5 =>           -- refresh cycle
247                 NextState <= REFRESH6;  -- go to next part of refresh cycle
248
249             when  REFRESH6 =>           -- refresh cycle
250                 NextState <= REFRESH7;  -- go to next part of refresh cycle
251
252             when  REFRESH7 =>           -- refresh cycle
253                 NextState <= REFRESH8;  -- go to next part of refresh cycle
254
255             when  REFRESH8 =>           -- refresh cycle
256                 NextState <= IDLE;      -- go back to idle
257
258             when others =>              -- default
259                 NextState <= IDLE;      -- go back to idle
260
261         end case;
262
263         if  reset = '1'  then           -- reset overrides everything
264             NextState <= IDLE;          --   go to idle on reset
265         end if;
266
267     end process transition;
268
269
270     -- storage of current state (loads the next state on the clock)
271
272     process (clk)
273     begin
274
275         if  clk = '1'  then             -- only change on rising edge of clock
276             CurrentState <= NextState;  -- save the new state information
277         end if;
278
279     end process;
280
281
282 end  assign_statebits;
```

G.2 Oscilloscope Trigger Code

```
1  ----------------------------------------------------------------------
2  --
3  --  Oscilloscope Digital Trigger
4  --
5  --  This is an implementation of a trigger for a digital oscilloscope in
6  --  VHDL.  There are three inputs to the system, one selects the trigger
7  --  slope and the other two determine the relationship between the trigger
8  --  level and the signal level.  The only output is a trigger signal which
9  --  indicates a trigger event has occurred.
10 --
11 --  The file contains multiple architectures for a Moore state machine
12 --  implementation to demonstrate the different ways of building a state
13 --  machine.
14 --
15 --
16 --  Revision History:
17 --     13 Apr 04  Glen George       Initial revision.
18 --      4 Nov 05  Glen George       Updated comments.
19 --     17 Nov 07  Glen George       Updated comments.
20 --     13 Feb 10  Glen George       Added more example architectures.
21 --     07 Jun 14  Albert Gural      Added basic hysterises to reject LSB noise issues.
22 --
23 ----------------------------------------------------------------------
24
25
26 -- bring in the necessary packages
27 library  ieee;
28 use  ieee.std_logic_1164.all;
29
30
31 --
32 --  Oscilloscope Digital Trigger entity declaration
33 --
34
35 entity  ScopeTrigger  is
36     port (
37         TS          :  in  std_logic;      -- trigger slope (1 -> negative, 0 -> positive)
38         TGT         :  in  std_logic;      -- signal level > trigger level + epsilon
39         TLT         :  in  std_logic;      -- signal level < trigger level - epsilon
40         clk         :  in  std_logic;      -- clock
41         Reset       :  in  std_logic;      -- reset the system
42         TrigEvent   :  out  std_logic      -- a trigger event has occurred
43     );
44 end  ScopeTrigger;
45
46
47 --
48 --  Oscilloscope Digital Trigger Moore State Machine
49 --     State Assignment Architecture
50 --
51 --  This architecture just shows the basic state machine syntax when the state
52 --  assignments are made manually.  This is useful for minimizing output
53 --  decoding logic and avoiding glitches in the output (due to the decoding
54 --  logic).
55 --
56
57 architecture  assign_statebits  of  ScopeTrigger  is
58
59     subtype  states  is  std_logic_vector(2 downto 0);     -- state type
60
61     -- define the actual states as constants
62     constant  IDLE      : states := "000";  -- waiting for start of trigger event
63     constant  WAIT_POS  : states := "001";  -- waiting for positive slope trigger
64     constant  WAIT_NEG  : states := "010";  -- waiting for negative slope trigger
65     constant  TRIGGER   : states := "100";  -- got a trigger event
66
67
68     signal  CurrentState  :  states;    -- current state
69     signal  NextState     :  states;    -- next state
70
71 begin
72     -- the output is always the high bit of the state encoding
73     TrigEvent <= CurrentState(2);
74
75     -- compute the next state (function of current state and inputs)
76     transition:  process (Reset, TS, TGT, TLT, CurrentState)
```

```
77      begin
78
79          case  CurrentState  is          -- do the state transition/output
80
81              when  IDLE =>               -- in idle state, do transition
82                  if  (TS = '0' and TLT = '1')  then
83                      NextState <= WAIT_POS;      -- below trigger and + slope
84                  elsif  (TS = '1' and TGT = '1')  then
85                      NextState <= WAIT_NEG;      -- above trigger and - slope
86                  else
87                      NextState <= IDLE;          -- trigger not possible yet
88                  end if;
89
90              when  WAIT_POS =>           -- waiting for positive slope trigger
91                  if  (TS = '0' and TGT = '0')  then
92                      NextState <= WAIT_POS;      -- no trigger yet
93                  elsif  (TS = '0' and TGT = '1')  then
94                      NextState <= TRIGGER;       -- got a trigger
95                  else
96                      NextState <= IDLE;          -- trigger slope changed
97                  end if;
98
99              when  WAIT_NEG =>           -- waiting for negative slope trigger
100                 if  (TS = '1' and TLT = '0')  then
101                     NextState <= WAIT_NEG;      -- no trigger yet
102                 elsif  (TS = '1' and TLT = '1')  then
103                     NextState <= TRIGGER;       -- got a trigger
104                 else
105                     NextState <= IDLE;          -- trigger slope changed
106                 end if;
107
108             when  TRIGGER =>            -- in the trigger state
109                 NextState <= IDLE;      -- always go back to idle
110
111             when others =>              -- default
112                 NextState <= IDLE;      -- go back to idle
113
114         end case;
115
116         if  Reset = '1'  then           -- reset overrides everything
117             NextState <= IDLE;          --   go to idle on reset
118         end if;
119
120     end process transition;
121
122
123     -- storage of current state (loads the next state on the clock)
124
125     process (clk)
126     begin
127
128         if  clk = '1'  then             -- only change on rising edge of clock
129             CurrentState <= NextState;  -- save the new state information
130         end if;
131
132     end process;
133
134
135 end  assign_statebits;
```

Appendix H

NIOS II Code

H.1 Main Loop

```
/*****************************************************************************/
/*                                                                           */
/*                                  MAINLOOP                                 */
/*                             Main Program Loop                             */
/*                         Digital Oscilloscope Project                      */
/*                                  EE/CS 52                                 */
/*                                                                           */
/*****************************************************************************/

/*
   This file contains the main processing loop (background) for the Digital
   Oscilloscope project.  The only global function included is:
      main - background processing loop

   The local functions included are:
      key_lookup - get a key and look up its keycode

   The locally global variable definitions included are:
      none


   Revision History
      3/8/94   Glen George       Initial revision.
      3/9/94   Glen George       Changed initialized const arrays to static
                 (in addition to const).
      3/9/94   Glen George       Moved the position of the const keyword in
                 declarations of arrays of pointers.
      3/13/94  Glen George       Updated comments.
      3/13/94  Glen George       Removed display_menu call after plot_trace,
                 the plot function takes care of the menu.
      3/17/97  Glen George       Updated comments.
      3/17/97  Glen George       Made key_lookup function static to make it
                 truly local.
      3/17/97  Glen George       Removed KEY_UNUSED and KEYCODE_UNUSED
                 references (no longer used).
      5/27/08  Glen George       Changed code to only check for sample done if
                 it is currently sampling.
*/



/* library include files */
  /* none */

/* local include files */
#include  "interfac.h"
#include  "scopedef.h"
#include  "keyproc.h"
#include  "menu.h"
#include  "tracutil.h"


/* local function declarations */
```

```
54 enum keycode  key_lookup(void);      /* translate key values into keycodes */
55 void key_int_installer(void);
56 void adc_int_installer(void);
57
58
59 /*
60    main
61
62    Description:      This procedure is the main program loop for the Digital
63                      Oscilloscope.  It loops getting keys from the keypad,
64                      processing those keys as is appropriate.  It also handles
65                      starting scope sample collection and updating the LCD
66              screen.
67
68    Arguments:        None.
69    Return Value:     (int) - return code, always 0 (never returns).
70
71    Input:            Keys from the keypad.
72    Output:           Traces and menus to the display.
73
74    Error Handling:   Invalid input is ignored.
75
76    Algorithms:       The function is table-driven.  The processing routines
77                      for each input are given in tables which are selected
78                      based on the context (state) the program is operating in.
79    Data Structures:  Array (process_key) to associate keys with actions
80              (functions to call).
81
82    Global Variables: None.
83
84    Author:           Glen George
85    Last Modified:    May 27, 2008
86 */
87
88 int  main() {
89     key_int_installer();
90     adc_int_installer();
91
92     /* variables */
93     enum keycode        key;            /* an input key */
94
95     enum status         state = MENU_ON;    /* current program state */
96
97     unsigned char      *sample;         /* a captured trace */
98
99     /* key processing functions (one for each system state type and key) */
100    enum status  (* const process_key[NUM_KEYCODES][NUM_STATES])(enum status) =
101       /*   Current System State                          */
102       /*  MENU_ON      MENU_OFF             Input Key  */
103      { {  menu_key,    menu_key    },   /* <Menu>      */
104        {  menu_up,     no_action   },   /* <Up>        */
105        {  menu_down,   no_action   },   /* <Down>      */
106        {  menu_left,   no_action   },   /* <Left>      */
107        {  menu_right,  no_action   },   /* <Right>     */
108        {  no_action,   no_action   } }; /* illegal key */
109
110
111
112    /* first initialize everything */
113    clear_display();        /* clear the display */
114
115    init_trace();       /* initialize the trace routines */
116    init_menu();        /* initialize the menu system */
117
118
119    // infinite loop processing input
120    //int x = 0;
121    while(TRUE)  {
122        //x += 1;
123        // if ready to do a trace, do it
124        if (trace_rdy()) do_trace();
125
126        //check if have a trace to display
127        if (is_sampling() && ((sample = sample_done()) != NULL))  {
128
129            //have a trace - output it
130            plot_trace(sample);
131            //done processing this trace
132            trace_done();
133        }
134
```

```
135         //now check for keypad input
136         if (key_available())  {
137
138             //have keypad input - get the key
139             key = key_lookup();
140
141             //execute processing routine for that key
142             state = process_key[key][state](state);
143         }
144     }
145
146
147     /* done with main (never should get here), return 0 */
148     return  0;
149
150 }
151
152
153
154
155 /*
156    key_lookup
157
158    Description:      This function gets a key from the keypad and translates
159                      the raw keycode to an enumerated keycode for the main
160                      loop.
161
162    Arguments:        None.
163    Return Value:     (enum keycode) - type of the key input on keypad.
164
165    Input:            Keys from the keypad.
166    Output:           None.
167
168    Error Handling:   Invalid keys are returned as KEYCODE_ILLEGAL.
169
170    Algorithms:       The function uses an array to lookup the key types.
171    Data Structures:  Array of key types versus key codes.
172
173    Global Variables: None.
174
175    Author:           Glen George
176    Last Modified:    Mar. 17, 1997
177
178 */
179
180 enum keycode    key_lookup()
181 {
182     /* variables */
183
184     const enum keycode  keycodes[] = /* array of keycodes */
185         {                          /* order must match keys array exactly */
186            KEYCODE_MENU,       /* <Menu>     */   /* also need an extra element */
187            KEYCODE_UP,         /* <Up>       */   /* for unknown key codes */
188            KEYCODE_DOWN,       /* <Down>     */
189            KEYCODE_LEFT,       /* <Left>     */
190            KEYCODE_RIGHT,      /* <Right>    */
191            KEYCODE_ILLEGAL     /* other keys */
192         };
193
194     const int  keys[] =   /* array of key values */
195         {             /* order must match keycodes array exactly */
196            KEY_MENU,     /* <Menu>     */
197            KEY_UP,       /* <Up>       */
198            KEY_DOWN,     /* <Down>     */
199            KEY_LEFT,     /* <Left>     */
200            KEY_RIGHT,    /* <Right>    */
201         };
202
203     int  key;        /* an input key */
204
205     int  i;              /* general loop index */
206
207
208
209     /* get a key */
210     key = getkey();
211
212
213     /* lookup key in keys array */
214     for (i = 0; ((i < (sizeof(keys)/sizeof(int))) && (key != keys[i])); i++);
215
```

```
216
217     /* return the appropriate key type */
218     return  keycodes[i];
219
220 }
```

H.2 Menu

H.2.1 menu

```
/*************************************************************************/
/*                                                                       */
/*                                 MENU.H                                */
/*                              Menu Functions                           */
/*                               Include File                            */
/*                         Digital Oscilloscope Project                  */
/*                                 EE/CS 52                              */
/*                                                                       */
/*************************************************************************/

/*
   This file contains the constants and function prototypes for the functions
   which deal with menus (defined in menu.c) for the Digital Oscilloscope
   project.


   Revision History:
      3/8/94   Glen George       Initial revision.
      3/13/94  Glen George       Updated comments.
      3/13/94  Glen George       Added definitions for SELECTED,
                    OPTION_NORMAL, and OPTION_SELECTED.
*/



#ifndef  __MENU_H__
    #define  __MENU_H__


/* library include files */
  /* none */

/* local include files */
#include  "interfac.h"
#include  "scopedef.h"
#include  "lcdout.h"




/* constants */

/* menu size */
#define  MENU_WIDTH    16          /* menu width (in characters) */
#define  MENU_HEIGHT    7          /* menu height (in characters) */
#define  MENU_SIZE_X   (MENU_WIDTH * HORIZ_SIZE)   /* menu width (in pixels) */
#define  MENU_SIZE_Y   (MENU_HEIGHT * VERT_SIZE)   /* menu height (in pixels) */

/* menu position */
#define  MENU_X     (LCD_WIDTH - MENU_WIDTH - 1)  /* x position (in characters) */
#define  MENU_Y     0                             /* y position (in characters) */
#define  MENU_UL_X (MENU_X * HORIZ_SIZE)          /* x position (in pixels) */
#define  MENU_UL_Y (MENU_Y * VERT_SIZE)           /* y position (in pixels) */

/* menu colors */
#define  SELECTED          REVERSE    /* color for a selected menu entry */
#define  OPTION_SELECTED   NORMAL         /* color for a selected menu entry option */
#define  OPTION_NORMAL     NORMAL         /* color for an unselected menu entry option */

/* number of menu entries */
#define  NO_MENU_ENTRIES   (sizeof(menu) / sizeof(struct menu_item))




/* structures, unions, and typedefs */

/* data for an item in a menu */
struct menu_item  {  const char  *s;         /* string for menu entry */
             int           h_off;     /* horizontal offset of entry */
             int           opt_off;   /* horizontal offset of option setting */
             void       (*display)(int, int, int);  /* option display function */
          };
```

```
74
75
76
77
78 /* function declarations */
79
80 /* menu initialization function */
81 void  init_menu(void);
82
83 /* menu display functions */
84 void  clear_menu(void);        /* clear the menu display */
85 void  display_menu(void);      /* display the menu */
86 void  refresh_menu(void);      /* refresh the menu */
87
88 /* menu update functions */
89 void  reset_menu(void);        /* reset the menu to first entry */
90 void  next_entry(void);        /* go to the next menu entry */
91 void  previous_entry(void);    /* go to the previous menu entry */
92
93 /* menu entry functions */
94 void  menu_entry_left(void);       /* do the <Left> key for the menu entry */
95 void  menu_entry_right(void);      /* do the <Right> key for the menu entry */
96
97
98 #endif
```

```
1  /*****************************************************************************/
2  /*                                                                           */
3  /*                                   MENU                                    */
4  /*                              Menu Functions                               */
5  /*                         Digital Oscilloscope Project                      */
6  /*                                  EE/CS 52                                 */
7  /*                                                                           */
8  /*****************************************************************************/
9
10 /*
11    This file contains the functions for processing menu entries for the
12    Digital Oscilloscope project.  These functions take care of maintaining the
13    menus and handling menu updates for the system.  The functions included
14    are:
15       clear_menu       - remove the menu from the display
16       display_menu     - display the menu
17       init_menu        - initialize menus
18       menu_entry_left  - take care of <Left> key for a menu entry
19       menu_entry_right - take care of <Right> key for a menu entry
20       next_entry       - next menu entry
21       previous_entry   - previous menu entry
22       refresh_menu     - re-display the menu if currently being displayed
23       reset_menu       - reset the current selection to the top of the menu
24
25    The local functions included are:
26       display_entry    - display a menu entry (including option setting)
27
28    The locally global variable definitions included are:
29       menu             - the menu
30       menu_display     - whether or not the menu is currently displayed
31       menu_entry       - the currently selected menu entry
32
33
34    Revision History
35       3/8/94   Glen George       Initial revision.
36       3/9/94   Glen George       Changed position of const keyword in array
37                 declarations involving pointers.
38       3/13/94  Glen George       Updated comments.
39       3/13/94  Glen George       Added display_entry function to output a menu
40                 entry and option setting to the LCD (affects
41                 many functions).
42       3/13/94  Glen George       Changed calls to set_status due to changing
43                      enum scale_status definition.
44       3/13/94  Glen George       No longer clear the menu area before
45                 restoring the trace in clear_menu() (not
46                 needed).
47       3/17/97  Glen George       Updated comments.
48       3/17/97  Glen George       Fixed minor bug in reset_menu().
49       3/17/97  Glen George       When initializing the menu in init_menu(),
50                 set the delay to MIN_DELAY instead of 0 and
51                 trigger to a middle value instead of
52                 MIN_TRG_LEVEL_SET.
```

```
53        5/3/06   Glen George       Changed to a more appropriate constant in
54                                   display_entry().
55        5/3/06   Glen George       Updated comments.
56        5/9/06   Glen George       Changed menus to handle a list for mode and
57                                 scale (move up and down list), instead of
58                       toggling values.
59  */
60
61
62
63  /* library include files */
64    /* none */
65
66  /* local include files */
67  #include  "scopedef.h"
68  #include  "lcdout.h"
69  #include  "menu.h"
70  #include  "menuact.h"
71  #include  "tracutil.h"
72
73
74
75
76  /* local function declarations */
77  void  display_entry(int, int);      /* display a menu entry and its setting */
78
79
80
81
82  /* locally global variables */
83  int  menu_display;           /* TRUE if menu is currently displayed */
84
85  const struct menu_item  menu[] =        /* the menu */
86      { { "Mode",    0, 4, display_mode      },
87        { "Scale",   0, 5, display_scale     },
88        { "Sweep",   0, 5, display_sweep     },
89        { "Trigger", 0, 7, no_display        },
90        { "Level",   2, 7, display_trg_level },
91        { "Slope",   2, 7, display_trg_slope },
92        { "Delay",   2, 7, display_trg_delay },
93      };
94
95  int  menu_entry;            /* currently selected menu entry */
96
97
98
99
100 /*
101    init_menu
102
103    Description:      This function initializes the menu routines.  It sets
104                      the current menu entry to the first entry, indicates the
105               display is off, and initializes the options (and
106               hardware) to normal trigger mode, scale displayed, the
107               fastest sweep rate, a middle trigger level, positive
108               trigger slope, and minimum delay.  Finally, it displays
109               the menu.
110
111    Arguments:        None.
112    Return Value:     None.
113
114    Input:            None.
115    Output:           The menu is displayed.
116
117    Error Handling:   None.
118
119    Algorithms:       None.
120    Data Structures:  None.
121
122    Global Variables: menu_display - reset to FALSE.
123              menu_entry   - reset to first entry (0).
124
125    Author:           Glen George
126    Last Modified:    Mar. 17, 1997
127
128 */
129
130 void  init_menu(void)
131 {
132     /* variables */
133       /* none */
```

```
134
135
136
137     /* set the menu parameters */
138     menu_entry = 0;      /* first menu entry */
139     menu_display = FALSE;   /* menu is not currently displayed (but it will be shortly) */
140
141
142     /* set the scope (option) parameters */
143     set_trigger_mode(NORMAL_TRIGGER);   /* normal triggering */
144     set_scale(SCALE_AXES);      /* scale is axes */
145     set_sweep(0);           /* first sweep rate */
146     set_trg_level((MIN_TRG_LEVEL_SET + MAX_TRG_LEVEL_SET) / 2); /* middle trigger level */
147     set_trg_slope(SLOPE_POSITIVE);  /* positive slope */
148     set_trg_delay(0);       /* default delay */
149
150
151     /* now display the menu */
152     display_menu();
153
154
155     /* done initializing, return */
156     return;
157
158 }
159
160
161
162
163 /*
164    clear_menu
165
166    Description:      This function removes the menu from the display.  The
167                      trace under the menu is restored.  The flag menu_display,
168              is cleared, indicating the menu is no longer being
169              displayed.  Note: if the menu is not currently being
170              displayed this function does nothing.
171
172    Arguments:        None.
173    Return Value:     None.
174
175    Input:            None.
176    Output:           The menu if displayed, is removed and the trace under it
177              is rewritten.
178
179    Error Handling:   None.
180
181    Algorithms:       None.
182    Data Structures:  None.
183
184    Global Variables: menu_display - checked and set to FALSE.
185
186    Author:           Glen George
187    Last Modified:    Mar. 13, 1994
188
189 */
190
191 void  clear_menu(void)
192 {
193     /* variables */
194       /* none */
195
196
197
198     /* check if the menu is currently being displayed */
199 //    if (menu_display)  {
200 //
201 //        /* menu is being displayed - turn it off and restore the trace in that area */
202 //  restore_menu_trace();
203 //    }
204
205
206     /* no longer displaying the menu */
207     menu_display = FALSE;
208
209
210     /* all done, return */
211     return;
212
213 }
214
```

```
215
216
217
218 /*
219    display_menu
220
221    Description:      This function displays the menu.  The trace under the
222                      menu is overwritten (but it was saved).  The flag
223              menu_display, is also set, indicating the menu is
224              currently being displayed.  Note: if the menu is already
225              being displayed this function does not redisplay it.
226
227    Arguments:        None.
228    Return Value:     None.
229
230    Input:            None.
231    Output:           The menu is displayed.
232
233    Error Handling:   None.
234
235    Algorithms:       None.
236    Data Structures:  None.
237
238    Global Variables: menu_display - set to TRUE.
239              menu_entry   - used to highlight currently selected entry.
240
241    Author:           Glen George
242    Last Modified:    Mar. 13, 1994
243
244 */
245
246 void  display_menu(void)
247 {
248     /* variables */
249     int  i;     /* loop index */
250
251
252
253     /* check if the menu is currently being displayed */
254     if (!menu_display)  {
255
256         /* menu is not being displayed - turn it on */
257     /* display it entry by entry */
258     for (i = 0; i < NO_MENU_ENTRIES; i++)  {
259
260         /* display this entry - check if it should be highlighted */
261         if (i == menu_entry)
262             /* currently selected entry - highlight it */
263             display_entry(i, TRUE);
264         else
265             /* not the currently selected entry - "normal video" */
266             display_entry(i, FALSE);
267         }
268     }
269
270
271     /* now are displaying the menu */
272     menu_display = TRUE;
273
274
275     /* all done, return */
276     return;
277
278 }
279
280
281
282
283 /*
284    refresh_menu
285
286    Description:      This function displays the menu if it is currently being
287              displayed.  The trace under the menu is overwritten (but
288              it was already saved).
289
290    Arguments:        None.
291    Return Value:     None.
292
293    Input:            None.
294    Output:           The menu is displayed.
295
```

```
296     Error Handling:   None.
297
298     Algorithms:       None.
299     Data Structures:  None.
300
301     Global Variables: menu_display - determines if menu should be displayed.
302
303     Author:           Glen George
304     Last Modified:    Mar. 8, 1994
305
306 */
307
308 void  refresh_menu(void)
309 {
310     /* variables */
311       /* none */
312
313
314
315     /* check if the menu is currently being displayed */
316     if (menu_display)  {
317
318         /* menu is currently being displayed - need to refresh it */
319     /* do this by turning off the display, then forcing it back on */
320     menu_display = FALSE;
321     display_menu();
322     }
323
324
325     /* refreshed the menu if it was displayed, now return */
326     return;
327
328 }
329
330
331
332
333 /*
334     reset_menu
335
336     Description:      This function resets the current menu selection to the
337                       first menu entry.  If the menu is currently being
338               displayed the display is updated.
339
340     Arguments:        None.
341     Return Value:     None.
342
343     Input:            None.
344     Output:           The menu display is updated if it is being displayed.
345
346     Error Handling:   None.
347
348     Algorithms:       None.
349     Data Structures:  None.
350
351     Global Variables: menu_display - checked to see if menu is displayed.
352               menu_entry   - reset to 0 (first entry).
353
354     Author:           Glen George
355     Last Modified:    Mar. 17, 1997
356
357 */
358
359 void  reset_menu(void)
360 {
361     /* variables */
362       /* none */
363
364
365
366     /* check if the menu is currently being displayed */
367     if (menu_display)  {
368
369         /* menu is being displayed */
370     /* remove highlight from currently selected entry */
371     display_entry(menu_entry, FALSE);
372     }
373
374
375     /* reset the currently selected entry */
376     menu_entry = 0;
```

```
377
378
379     /* finally, highlight the first entry if the menu is being displayed */
380     if (menu_display)
381     display_entry(menu_entry, TRUE);
382
383
384
385     /* all done, return */
386     return;
387
388 }
389
390
391
392
393 /*
394    next_entry
395
396    Description:      This function changes the current menu selection to the
397                      next menu entry.  If the current selection is the last
398              entry in the menu, it is not changed.  If the menu is
399              currently being displayed, the display is updated.
400
401    Arguments:        None.
402    Return Value:     None.
403
404    Input:            None.
405    Output:           The menu display is updated if it is being displayed and
406              the entry selected changes.
407
408    Error Handling:   None.
409
410    Algorithms:       None.
411    Data Structures:  None.
412
413    Global Variables: menu_display - checked to see if menu is displayed.
414              menu_entry   - updated to a new entry (if not at end).
415
416    Author:           Glen George
417    Last Modified:    Mar. 13, 1994
418
419 */
420
421 void  next_entry(void)
422 {
423     /* variables */
424       /* none */
425
426
427
428     /* only update if not at end of the menu */
429     if (menu_entry < (NO_MENU_ENTRIES - 1))  {
430
431         /* not at the end of the menu */
432
433     /* turn off current entry if displaying */
434     if (menu_display)
435             /* displaying menu - turn off currently selected entry */
436         display_entry(menu_entry, FALSE);
437
438     /* update the menu entry to the next one */
439     menu_entry++;
440
441     /* now highlight this entry if displaying the menu */
442     if (menu_display)
443             /* displaying menu - highlight newly selected entry */
444         display_entry(menu_entry, TRUE);
445     }
446
447
448     /* all done, return */
449     return;
450
451 }
452
453
454
455
456 /*
457    previous_entry
```

```
458
459     Description:      This function changes the current menu selection to the
460                       previous menu entry.  If the current selection is the
461               first entry in the menu, it is not changed.  If the menu
462               is currently being displayed, the display is updated.
463
464     Arguments:        None.
465     Return Value:     None.
466
467     Input:            None.
468     Output:           The menu display is updated if it is being displayed and
469               the currently selected entry changes.
470
471     Error Handling:   None.
472
473     Algorithms:       None.
474     Data Structures:  None.
475
476     Global Variables: menu_display - checked to see if menu is displayed.
477               menu_entry   - updated to a new entry (if not at start).
478
479     Author:           Glen George
480     Last Modified:    Mar. 13, 1994
481
482 */
483
484 void  previous_entry(void)
485 {
486     /* variables */
487       /* none */
488
489
490
491     /* only update if not at the start of the menu */
492     if (menu_entry > 0)  {
493
494         /* not at the start of the menu */
495
496     /* turn off current entry if displaying */
497     if (menu_display)
498             /* displaying menu - turn off currently selected entry */
499         display_entry(menu_entry, FALSE);
500
501     /* update the menu entry to the previous one */
502     menu_entry--;
503
504     /* now highlight this entry if displaying the menu */
505     if (menu_display)
506             /* displaying menu - highlight newly selected entry */
507         display_entry(menu_entry, TRUE);
508
509     }
510
511
512     /* all done, return */
513     return;
514
515 }
516
517
518
519
520 /*
521     menu_entry_left
522
523     Description:      This function handles the <Left> key for the current menu
524                       selection.  It does this by doing a table lookup on the
525               current menu selection.
526
527     Arguments:        None.
528     Return Value:     None.
529
530     Input:            None.
531     Output:           The menu display is updated if it is being displayed and
532               the <Left> key causes a change to the display.
533
534     Error Handling:   None.
535
536     Algorithms:       Table lookup is used to determine what to do for the
537               input key.
538     Data Structures:  An array holds the table of key processing routines.
```

```
539
540     Global Variables: menu_entry - used to select the processing function.
541
542     Author:           Glen George
543     Last Modified:    May 9, 2006
544
545 */
546
547 void  menu_entry_left(void)
548 {
549     /* variables */
550
551     /* key processing functions */
552     void  (* const process[])(void) =
553         /*  Mode            Scale             Sweep           Trigger      */
554           {  mode_down,      scale_down,       sweep_down,     trace_rearm,
555              trg_level_down, trg_slope_toggle, trg_delay_down               };
556         /*  Level           Slope             Delay                        */
557
558
559
560     /* invoke the appropriate <Left> key function */
561     process[menu_entry]();
562
563     /* if displaying menu entries, display the new value */
564     /* note: since it is being changed - know this option is selected */
565     if (menu_display)  {
566         menu[menu_entry].display((MENU_X + menu[menu_entry].opt_off),
567                          (MENU_Y + menu_entry), OPTION_SELECTED);
568     }
569
570
571     /* all done, return */
572     return;
573
574 }
575
576
577
578
579 /*
580     menu_entry_right
581
582     Description:      This function handles the <Right> key for the current
583                       menu selection.  It does this by doing a table lookup on
584               the current menu selection.
585
586     Arguments:        None.
587     Return Value:     None.
588
589     Input:            None.
590     Output:           The menu display is updated if it is being displayed and
591               the <Right> key causes a change to the display.
592
593     Error Handling:   None.
594
595     Algorithms:       Table lookup is used to determine what to do for the
596               input key.
597     Data Structures:  An array holds the table of key processing routines.
598
599     Global Variables: menu       - used to display the new menu value.
600               menu_entry - used to select the processing function.
601
602     Author:           Glen George
603     Last Modified:    May 9, 2006
604
605 */
606
607 void  menu_entry_right(void)
608 {
609     /* variables */
610
611     /* key processing functions */
612     void  (* const process[])(void) =
613         /*  Mode          Scale             Sweep           Trigger      */
614           {  mode_up    ,  scale_up,     sweep_up,       trace_rearm,
615              trg_level_up, trg_slope_toggle, trg_delay_up                 };
616         /*  Level         Slope             Delay                        */
617
618
619
```

```
620     /* invoke the appropriate <Right> key function */
621     process[menu_entry]();
622 
623     /* if displaying menu entries, display the new value */
624     /* note: since it is being changed - know this option is selected */
625     if (menu_display)  {
626         menu[menu_entry].display((MENU_X + menu[menu_entry].opt_off),
627                                  (MENU_Y + menu_entry), OPTION_SELECTED);
628     }
629 
630 
631     /* all done, return */
632     return;
633 
634 }
635 
636 
637 
638 
639 /*
640    display_entry
641 
642    Description:      This function displays the passed menu entry and its
643              current option setting.  If the second argument is TRUE
644              it displays them with color SELECTED and OPTION_SELECTED
645              respectively.  If the second argument is FALSE it
646              displays the menu entry with color NORMAL and the option
647              setting with color OPTION_NORMAL.
648 
649    Arguments:        entry (int)    - menu entry to be displayed.
650              selected (int) - whether or not the menu entry is
651                               currently selected (determines the color
652                               with which the entry is output).
653    Return Value:     None.
654 
655    Input:            None.
656    Output:           The menu entry is output to the LCD.
657 
658    Error Handling:   None.
659 
660    Algorithms:       None.
661    Data Structures:  None.
662 
663    Global Variables: menu - used to display the menu entry.
664 
665    Author:           Glen George
666    Last Modified:    Aug. 13, 2004
667 
668 */
669 
670 void  display_entry(int entry, int selected)
671 {
672     /* variables */
673       /* none */
674 
675 
676 
677     /* output the menu entry with the appropriate color */
678     plot_string((MENU_X + menu[entry].h_off), (MENU_Y + entry), menu[entry].s,
679             (selected ? SELECTED : NORMAL));
680     /* also output the menu option with the appropriate color */
681     menu[entry].display((MENU_X + menu[entry].opt_off), (MENU_Y + entry),
682                 (selected ? OPTION_SELECTED : OPTION_NORMAL));
683 
684 
685     /* all done outputting this menu entry - return */
686     return;
687 
688 }
```

H.2.2 menuact

```
1 /*****************************************************************************/
2 /*                                                                           */
3 /*                                 MENUACT.H                                 */
4 /*                           Menu Action Functions                           */
```

```
5  /*                              Include File                                  */
6  /*                      Digital Oscilloscope Project                          */
7  /*                                EE/CS 52                                    */
8  /*                                                                            */
9  /******************************************************************************/
10
11 /*
12    This file contains the constants and function prototypes for the functions
13    which carry out menu actions and display and initialize menu settings for
14    the Digital Oscilloscope project (the functions are defined in menuact.c).
15
16
17    Revision History:
18       3/8/94   Glen George       Initial revision.
19       3/13/94  Glen George       Updated comments.
20       3/13/94  Glen George       Changed definition of enum scale_type (was
21                                   enum scale_status).
22       3/10/95  Glen George       Changed MAX_TRG_LEVEL_SET (maximum trigger
23                                   level) to 127 to match specification.
24       3/17/97  Glen George       Updated comments.
25       5/3/06   Glen George       Updated comments.
26       5/9/06   Glen George       Added a new mode (AUTO_TRIGGER) and a new
27                                   scale (SCALE_GRID).
28       5/9/06   Glen George       Added menu functions for mode and scale to
29                                   move up and down a list instead of just
30                                 toggling the selection.
31       5/9/06   Glen George       Added declaration for the accessor to the
32                                   current trigger mode (get_trigger_mode).
33 */
34
35
36
37 #ifndef  __MENUACT_H__
38     #define  __MENUACT_H__
39
40
41 /* library include files */
42   /* none */
43
44 /* local include files */
45 #include  "interfac.h"
46 #include  "lcdout.h"
47
48
49
50
51 /* constants */
52
53 /* min and max trigger level settings */
54 #define  MIN_TRG_LEVEL_SET   0
55 #define  MAX_TRG_LEVEL_SET   127
56
57 /* number of different sweep rates */
58 #define  NO_SWEEP_RATES     (sizeof(sweep_rates) / sizeof(struct sweep_info))
59
60
61
62
63 /* structures, unions, and typedefs */
64
65 /* types of triggering modes */
66 enum trigger_type  {  NORMAL_TRIGGER,       /* normal triggering */
67                AUTO_TRIGGER,     /* automatic triggering */
68                ONESHOT_TRIGGER       /* one-shot triggering */
69            };
70
71 /* types of displayed scales */
72 enum scale_type    {  SCALE_NONE,       /* no scale is displayed */
73                SCALE_AXES,       /* scale is a set of axes */
74                SCALE_GRID        /* scale is a grid */
75            };
76
77 /* types of trigger slopes */
78 enum slope_type    {  SLOPE_POSITIVE,       /* positive trigger slope */
79                SLOPE_NEGATIVE        /* negative trigger slope */
80            };
81
82 /* sweep rate information */
83 struct sweep_info  {  long int     sample_rate;    /* sample rate */
84                const char  *s;          /* sweep rate string */
85            };
```

```
86
87
88
89
90  /* function declarations */
91
92  /* menu option actions */
93  void   no_menu_action(void);     /* no action to perform */
94  void   mode_down(void);          /* change to the "next" trigger mode */
95  void   mode_up(void);            /* change to the "previous" trigger mode */
96  void   scale_down(void);         /* change to the "next" scale type */
97  void   scale_up(void);           /* change to the "previous" scale type */
98  void   sweep_down(void);         /* decrease the sweep rate */
99  void   sweep_up(void);           /* increase the sweep rate */
100 void   trg_level_down(void);    /* decrease the trigger level */
101 void   trg_level_up(void);      /* increase the trigger level */
102 void   trg_slope_toggle(void);  /* toggle the trigger slope */
103 void   trg_delay_down(void);    /* decrease the trigger delay */
104 void   trg_delay_up(void);      /* increase the trigger delay */
105
106 /* option accessor routines */
107 enum trigger_type  get_trigger_mode(void);  /* get the current trigger mode */
108
109 /* option initialization routines */
110 void   set_trigger_mode(enum trigger_type);  /* set the trigger mode */
111 void   set_trigger_normal(void);             /* normal trigger mode */
112 void   set_trigger_auto(void);               /* auto trigger mode */
113 void   set_trigger_single(void);             /* single trigger mode */
114 void   set_scale(enum scale_type);           /* set the scale type */
115 void   set_sweep(int);                   /* set the sweep rate */
116 void   set_trg_level(int);               /* set the trigger level */
117 void   set_trg_slope(enum slope_type);       /* set the trigger slope */
118 void   set_trg_delay(long int);          /* set the tigger delay */
119
120 /* option display routines */
121 void   no_display(int, int, int);     /* no option setting to display */
122 void   display_mode(int, int, int);       /* display trigger mode */
123 void   display_scale(int, int, int);      /* display the scale type */
124 void   display_sweep(int, int, int);      /* display the sweep rate */
125 void   display_trg_level(int, int, int);  /* display the trigger level */
126 void   display_trg_slope(int, int, int);  /* display the trigger slope */
127 void   display_trg_delay(int, int, int);  /* display the tigger delay */
128
129
130 #endif
```

```
1  /*************************************************************************/
2  /*                                                                       */
3  /*                                 MENUACT                               */
4  /*                           Menu Action Functions                       */
5  /*                        Digital Oscilloscope Project                   */
6  /*                                 EE/CS 52                              */
7  /*                                                                       */
8  /*************************************************************************/
9
10 /*
11    This file contains the functions for carrying out menu actions for the
12    Digital Oscilloscope project.  These functions are invoked when the <Left>
13    or <Right> key is pressed for a menu item.  Also included are the functions
14    for displaying the current menu option selection.  The functions included
15    are:
16       display_mode      - display trigger mode
17       display_scale     - display the scale type
18       display_sweep     - display the sweep rate
19       display_trg_delay - display the tigger delay
20       display_trg_level - display the trigger level
21       display_trg_slope - display the trigger slope
22       get_trigger_mode  - get the current trigger mode
23       mode_down         - go to the "next" trigger mode
24       mode_up           - go to the "previous" trigger mode
25       no_display        - nothing to display for option setting
26       no_menu_action    - no action to perform for <Left> or <Right> key
27       scale_down        - go to the "next" scale type
28       scale_up          - go to the "previous" scale type
29       set_scale         - set the scale type
30       set_sweep         - set the sweep rate
31       set_trg_delay     - set the tigger delay
32       set_trg_level     - set the trigger level
```

```
      set_trg_slope     - set the trigger slope
      set_trigger_mode  - set the trigger mode
      sweep_down        - decrease the sweep rate
      sweep_up          - increase the sweep rate
      trg_delay_down    - decrease the trigger delay
      trg_delay_up      - increase the trigger delay
      trg_level_down    - decrease the trigger level
      trg_level_up      - increase the trigger level
      trg_slope_toggle  - toggle the trigger slope between "+" and "-"

   The local functions included are:
      adjust_trg_delay  - adjust the trigger delay for a new sweep rate
      cvt_num_field     - converts a numeric field value to a string

   The locally global variable definitions included are:
      delay         - current trigger delay
      level         - current trigger level
      scale         - current display scale type
      slope         - current trigger slope
      sweep         - current sweep rate
      sweep_rates   - table of information on possible sweep rates
      trigger_mode  - current triggering mode


   Revision History
      3/8/94   Glen George       Initial revision.
      3/13/94  Glen George       Updated comments.
      3/13/94  Glen George       Changed all arrays of constant strings to be
                      so compiler generates correct code.
      3/13/94  Glen George       Changed scale to type enum scale_type and
                      output the selection as "None" or "Axes".
                    This will allow for easier future expansion.
      3/13/94  Glen George       Changed name of set_axes function (in
                      tracutil.c) to set_display_scale.
      3/10/95  Glen George       Changed calculation of displayed trigger
                      level to use constants MIN_TRG_LEVEL_SET and
                    MAX_TRG_LEVEL_SET to get the trigger level
                    range.
      3/17/97  Glen George       Updated comments.
      5/3/06   Glen George       Changed sweep definitions to include new
                      sweep rates of 100 ns, 200 ns, 500 ns, and
                      1 us and updated functions to handle these
                    new rates.
      5/9/06   Glen George       Added new a triggering mode (automatic
                                 triggering) and a new scale (grid) and
                                 updated functions to implement these options.
      5/9/06   Glen George       Added functions for setting the triggering
                                 mode and scale by going up and down the list
                                 of possibilities instead of just toggling
                                 between one of two possibilities (since there
                    are more than two now).
      5/9/06   Glen George       Added accessor function (get_trigger_mode)
                                 to be able to get the current trigger mode.
*/



/* library include files */
  /* none */

/* local include files */
#include  "interfac.h"
#include  "scopedef.h"
#include  "lcdout.h"
#include  "menuact.h"
#include  "tracutil.h"




/* local function declarations */
void  adjust_trg_delay(int, int);       /* adjust the trigger delay for new sweep */
void  cvt_num_field(long int, char *);  /* convert a number to a string */




/* locally global variables */

/* trace parameters */
enum trigger_type   trigger_mode;   /* current triggering mode */
```

```
enum scale_type     scale;          /* current scale type */
int                 sweep;          /* sweep rate index */
int                 level;          /* current trigger level */
enum slope_type     slope;          /* current trigger slope */
long int            delay;          /* current trigger delay */

/* sweep rate information */
const struct sweep_info  sweep_rates[] =
    { {200000000L, " 5 ns  " },
      {100000000L, " 10 ns " },
      { 50000000L, " 20 ns " },
      { 20000000L, " 50 ns " },
      { 10000000L, " 100 ns" },
      {  5000000L, " 200 ns" },
      {  2000000L, " 500 ns" },
      {  1000000L, " 1 \004s  " },
      {   500000L, " 2 \004s  " },
      {   200000L, " 5 \004s  " },
      {   100000L, " 10 \004s " },
      {    50000L, " 20 \004s " },
      {    20000L, " 50 \004s " },
      {    10000L, " 100 \004s" },
      {     5000L, " 200 \004s" },
      {     2000L, " 500 \004s" },
      {     1000L, " 1 ms  "    },
      {      500L, " 2 ms  "    },
      {      200L, " 5 ms  "    },
      {      100L, " 10 ms "    },
      {       50L, " 20 ms "    } };



/*
   no_menu_action

   Description:      This function handles a menu action when there is nothing
                     to be done.  It just returns.

   Arguments:        None.
   Return Value:     None.

   Input:            None.
   Output:           None.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: None.

   Author:           Glen George
   Last Modified:    Mar. 8, 1994

*/

void  no_menu_action()
{
    /* variables */
      /* none */



    /* nothing to do - return */
    return;

}




/*
   no_display

   Description:      This function handles displaying a menu option's setting
                     when there is nothing to display.  It just returns,
              ignoring all arguments.

   Arguments:        x_pos (int) - x position (in character cells) at which to
                   display the menu option (not used).
```

```
195                y_pos (int) - y position (in character cells) at which to
196                      display the menu option (not used).
197                style (int) - style with which to display the menu option
198                         (not used).
199    Return Value:     None.
200
201    Input:            None.
202    Output:           None.
203
204    Error Handling:   None.
205
206    Algorithms:       None.
207    Data Structures:  None.
208
209    Global Variables: None.
210
211    Author:           Glen George
212    Last Modified:    Mar. 8, 1994
213
214 */
215
216 void  no_display(int x_pos, int y_pos, int style)
217 {
218     /* variables */
219       /* none */
220
221
222
223     /* nothing to do - return */
224     return;
225
226 }
227
228
229
230
231 /*
232    set_trigger_mode
233
234    Description:      This function sets the triggering mode to the passed
235                      value.
236
237    Arguments:        m (enum trigger_type) - mode to which to set the
238                          triggering mode.
239    Return Value:     None.
240
241    Input:            None.
242    Output:           None.
243
244    Error Handling:   None.
245
246    Algorithms:       None.
247    Data Structures:  None.
248
249    Global Variables: trigger_mode - initialized to the passed value.
250
251    Author:           Glen George
252    Last Modified:    Mar. 8, 1994
253
254 */
255
256 void  set_trigger_mode(enum trigger_type m)
257 {
258     /* variables */
259       /* none */
260
261
262
263     /* set the trigger mode */
264     trigger_mode = m;
265
266     /* set the new mode */
267     set_mode(trigger_mode);
268
269
270     /* all done setting the trigger mode - return */
271     return;
272
273 }
274
275
```

```
276
277
278 /*
279    set_trigger_normal
280
281    Description:      This function sets the triggering mode to normal.
282
283    Arguments:        None.
284    Return Value:     None.
285
286    Input:            None.
287    Output:           None.
288
289    Error Handling:   None.
290
291    Algorithms:       None.
292    Data Structures:  None.
293
294    Global Variables: trigger_mode - initialized to the passed value.
295
296    Author:           Albert Gural
297    Last Modified:    Jun. 13, 2014
298
299 */
300
301 void  set_trigger_normal()
302 {
303     /* variables */
304       /* none */
305
306
307
308     /* set the trigger mode */
309     trigger_mode = NORMAL_TRIGGER;
310
311     /* set the new mode */
312     set_mode(trigger_mode);
313
314
315     /* all done setting the trigger mode - return */
316     return;
317
318 }
319
320
321
322
323 /*
324    set_trigger_auto
325
326    Description:      This function sets the triggering mode to auto.
327
328    Arguments:        None.
329    Return Value:     None.
330
331    Input:            None.
332    Output:           None.
333
334    Error Handling:   None.
335
336    Algorithms:       None.
337    Data Structures:  None.
338
339    Global Variables: trigger_mode - initialized to the passed value.
340
341    Author:           Albert Gural
342    Last Modified:    Jun. 13, 2014
343
344 */
345
346 void  set_trigger_auto()
347 {
348     /* variables */
349       /* none */
350
351
352
353     /* set the trigger mode */
354     trigger_mode = AUTO_TRIGGER;
355
356     /* set the new mode */
```

```
357     set_mode(trigger_mode);
358
359
360     /* all done setting the trigger mode - return */
361     return;
362
363 }
364
365
366
367
368 /*
369    set_trigger_single
370
371    Description:      This function sets the triggering mode to single/one-shot.
372
373    Arguments:        None.
374    Return Value:     None.
375
376    Input:            None.
377    Output:           None.
378
379    Error Handling:   None.
380
381    Algorithms:       None.
382    Data Structures:  None.
383
384    Global Variables: trigger_mode - initialized to the passed value.
385
386    Author:           Albert Gural
387    Last Modified:    Jun. 13, 2014
388
389 */
390
391 void  set_trigger_single()
392 {
393     /* variables */
394       /* none */
395
396
397
398     /* set the trigger mode */
399     trigger_mode = ONESHOT_TRIGGER;
400
401     /* set the new mode */
402     set_mode(trigger_mode);
403
404
405     /* all done setting the trigger mode - return */
406     return;
407
408 }
409
410
411
412
413 /*
414    get_trigger_mode
415
416    Description:      This function returns the current triggering mode.
417
418    Arguments:        None.
419    Return Value:     (enum trigger_type) - current triggering mode.
420
421    Input:            None.
422    Output:           None.
423
424    Error Handling:   None.
425
426    Algorithms:       None.
427    Data Structures:  None.
428
429    Global Variables: trigger_mode - value is returned (not changed).
430
431    Author:           Glen George
432    Last Modified:    May 9, 2006
433
434 */
435
436 enum trigger_type  get_trigger_mode()
437 {
```

```
    /* variables */
      /* none */



    /* return the current trigger mode */
    return  trigger_mode;

}



/*
   mode_down

   Description:      This function handles moving down the list of trigger
                     modes.  It changes to the "next" triggering mode and
                     sets that as the current mode.

   Arguments:        None.
   Return Value:     None.

   Input:            None.
   Output:           None.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: trigger_mode - changed to "next" trigger mode.

   Author:           Glen George
   Last Modified:    May 9, 2006

*/

void  mode_down()
{
    /* variables */
      /* none */



    /* move to the "next" triggering mode */
    if (trigger_mode == NORMAL_TRIGGER)
        trigger_mode = AUTO_TRIGGER;
    else if (trigger_mode == AUTO_TRIGGER)
        trigger_mode = ONESHOT_TRIGGER;
    else
        trigger_mode = NORMAL_TRIGGER;

    /* set the new mode */
    set_mode(trigger_mode);


    /* all done with the trigger mode - return */
    return;

}



/*
   mode_up

   Description:      This function handles moving up the list of trigger
                     modes.  It changes to the "previous" triggering mode and
                     sets that as the current mode.

   Arguments:        None.
   Return Value:     None.

   Input:            None.
   Output:           None.

   Error Handling:   None.

   Algorithms:       None.
```

```
519     Data Structures:  None.
520
521     Global Variables: trigger_mode - changed to "previous" trigger mode.
522
523     Author:           Glen George
524     Last Modified:    May 9, 2006
525
526 */
527
528 void  mode_up()
529 {
530     /* variables */
531       /* none */
532
533
534
535     /* move to the "previous" triggering mode */
536     if (trigger_mode == NORMAL_TRIGGER)
537         trigger_mode = ONESHOT_TRIGGER;
538     else if (trigger_mode == AUTO_TRIGGER)
539         trigger_mode = NORMAL_TRIGGER;
540     else
541         trigger_mode = AUTO_TRIGGER;
542
543     /* set the new mode */
544     set_mode(trigger_mode);
545
546
547     /* all done with the trigger mode - return */
548     return;
549
550 }
551
552
553
554
555 /*
556     display_mode
557
558     Description:      This function displays the current triggering mode at the
559                       passed position, in the passed style.
560
561     Arguments:        x_pos (int) - x position (in character cells) at which to
562                     display the trigger mode.
563               y_pos (int) - y position (in character cells) at which to
564                     display the trigger mode.
565               style (int) - style with which to display the trigger
566                         mode.
567     Return Value:     None.
568
569     Input:            None.
570     Output:           The trigger mode is displayed at the passed position on
571               the screen.
572
573     Error Handling:   None.
574
575     Algorithms:       None.
576     Data Structures:  None.
577
578     Global Variables: trigger_mode - determines which string is displayed.
579
580     Author:           Glen George
581     Last Modified:    May 9, 2006
582
583 */
584
585 void  display_mode(int x_pos, int y_pos, int style)
586 {
587     /* variables */
588
589     /* the mode strings (must match enumerated type) */
590     const char * const  modes[] =  {  " Normal    ",
591                                       " Automatic",
592                                       " One-Shot "  };
593
594
595
596     /* display the trigger mode */
597     plot_string(x_pos, y_pos, modes[trigger_mode], style);
598
599
```

```
600     /* all done displaying the trigger mode - return */
601     return;
602
603 }
604
605
606
607
608 /*
609    set_scale
610
611    Description:      This function sets the scale type to the passed value.
612
613    Arguments:        s (enum scale_type) - scale type to which to initialize
614                        the scale status.
615    Return Value:     None.
616
617    Input:            None.
618    Output:           The new trace display is updated with the new scale.
619
620    Error Handling:   None.
621
622    Algorithms:       None.
623    Data Structures:  None.
624
625    Global Variables: scale - initialized to the passed value.
626
627    Author:           Glen George
628    Last Modified:    Mar. 13, 1994
629
630 */
631
632 void  set_scale(enum scale_type s)
633 {
634     /* variables */
635       /* none */
636
637
638
639     /* set the scale type */
640     scale = s;
641
642     /* output the scale appropriately */
643     set_display_scale(scale);
644
645
646     /* all done setting the scale type - return */
647     return;
648
649 }
650
651
652
653
654 /*
655    scale_down
656
657    Description:      This function handles moving down the list of scale
658                      types.  It changes to the "next" type of scale and sets
659              this as the current scale type.
660
661    Arguments:        None.
662    Return Value:     None.
663
664    Input:            None.
665    Output:           The new scale is output to the trace display.
666
667    Error Handling:   None.
668
669    Algorithms:       None.
670    Data Structures:  None.
671
672    Global Variables: scale - changed to the "next" scale type.
673
674    Author:           Glen George
675    Last Modified:    May 9, 2006
676
677 */
678
679 void  scale_down()
680 {
```

```
    /* variables */
      /* none */



    /* change to the "next" scale type */
    if (scale == SCALE_NONE)
        scale = SCALE_AXES;
    else if (scale == SCALE_AXES)
        scale = SCALE_GRID;
    else
        scale = SCALE_NONE;

    /* set the scale type */
    set_display_scale(scale);


    /* all done with toggling the scale type - return */
    return;

}




/*
   scale_up

   Description:      This function handles moving up the list of scale types.
                     It changes to the "previous" type of scale and sets this
             as the current scale type.

   Arguments:        None.
   Return Value:     None.

   Input:            None.
   Output:           The new scale is output to the trace display.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: scale - changed to the "previous" scale type.

   Author:           Glen George
   Last Modified:    May 9, 2006

*/

void  scale_up()
{
    /* variables */
      /* none */



    /* change to the "previous" scale type */
    if (scale == SCALE_NONE)
        scale = SCALE_GRID;
    else if (scale == SCALE_AXES)
        scale = SCALE_NONE;
    else
        scale = SCALE_AXES;

    /* set the scale type */
    set_display_scale(scale);


    /* all done with toggling the scale type - return */
    return;

}




/*
   display_scale

   Description:      This function displays the current scale type at the
```

```
762                     passed position, in the passed style.
763
764     Arguments:          x_pos (int) - x position (in character cells) at which to
765                     display the scale type.
766                 y_pos (int) - y position (in character cells) at which to
767                     display the scale type.
768                 style (int) - style with which to display the scale type.
769     Return Value:       None.
770
771     Input:              None.
772     Output:             The scale type is displayed at the passed position on the
773                 display.
774
775     Error Handling:     None.
776
777     Algorithms:         None.
778     Data Structures:    None.
779
780     Global Variables: scale - determines which string is displayed.
781
782     Author:             Glen George
783     Last Modified:      Mar. 13, 1994
784
785 */
786
787 void  display_scale(int x_pos, int y_pos, int style)
788 {
789     /* variables */
790
791     /* the scale type strings (must match enumerated type) */
792     const char * const  scale_stat[] =  {  " None",
793                                            " Axes",
794                                            " Grid"  };
795
796
797
798     /* display the scale status */
799     plot_string(x_pos, y_pos, scale_stat[scale], style);
800
801
802     /* all done displaying the scale status - return */
803     return;
804
805 }
806
807
808
809
810 /*
811    set_sweep
812
813     Description:        This function sets the sweep rate to the passed value.
814                         The passed value gives the sweep rate to choose from the
815                 list of sweep rates (it gives the list index).
816
817     Arguments:          s (int) - index into the list of sweep rates to which to
818                     set the current sweep rate.
819     Return Value:       None.
820
821     Input:              None.
822     Output:             None.
823
824     Error Handling:     The passed index is not checked for validity.
825
826     Algorithms:         None.
827     Data Structures:    None.
828
829     Global Variables: sweep - initialized to the passed value.
830
831     Author:             Glen George
832     Last Modified:      Mar. 8, 1994
833
834 */
835
836 void  set_sweep(int s)
837 {
838     /* variables */
839     int   sample_size;        /* sample size for this sweep rate */
840
841
842
```

```
    /* set the new sweep rate */
    sweep = s;

    /* set the sweep rate for the hardware */
    sample_size = set_sample_rate(sweep_rates[sweep].sample_rate);
    /* also set the sample size for the trace capture */
    set_trace_size(sample_size);


    /* all done initializing the sweep rate - return */
    return;

}




/*
   sweep_down

   Description:      This function handles decreasing the current sweep rate.
             The new sweep rate (and sample size) is sent to the
             hardware (and trace routines).  If an attempt is made to
             lower the sweep rate below the minimum value it is not
             changed.  This routine also updates the sweep delay based
             on the new sweep rate (to keep the delay time constant).

   Arguments:        None.
   Return Value:     None.

   Input:            None.
   Output:           None.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: sweep - decremented if not already 0.
             delay - increased to keep delay time constant.

   Known Bugs:       The updated delay time is not displayed.  Since the time
             is typically only rounded to the new sample rate, this is
             not a major problem.

   Author:           Glen George
   Last Modified:    Mar. 8, 1994

*/

void  sweep_down()
{
    /* variables */
    int  sample_size;       /* sample size for the new sweep rate */



    /* decrease the sweep rate, if not already the minimum */
    if (sweep > 0)  {
        /* not at minimum, adjust delay for new sweep */
    adjust_trg_delay(sweep, (sweep - 1));
    /* now set new sweep rate */
        sweep--;
    }

    /* set the sweep rate for the hardware */
    sample_size = set_sample_rate(sweep_rates[sweep].sample_rate);
    /* also set the sample size for the trace capture */
    set_trace_size(sample_size);


    /* all done with lowering the sweep rate - return */
    return;

}




/*
   sweep_up
```

```
924
925     Description:      This function handles increasing the current sweep rate.
926               The new sweep rate (and sample size) is sent to the
927               hardware (and trace routines).  If an attempt is made to
928               raise the sweep rate above the maximum value it is not
929               changed.  This routine also updates the sweep delay based
930               on the new sweep rate (to keep the delay time constant).
931
932     Arguments:        None.
933     Return Value:     None.
934
935     Input:            None.
936     Output:           None.
937
938     Error Handling:   None.
939
940     Algorithms:       None.
941     Data Structures:  None.
942
943     Global Variables: sweep - incremented if not already the maximum value.
944               delay - decreased to keep delay time constant.
945
946     Known Bugs:       The updated delay time is not displayed.  Since the time
947               is typically only rounded to the new sample rate, this is
948               not a major problem.
949
950     Author:           Glen George
951     Last Modified:    Mar. 8, 1994
952
953 */
954
955 void  sweep_up()
956 {
957     /* variables */
958     int  sample_size;       /* sample size for the new sweep rate */
959
960
961
962     /* increase the sweep rate, if not already the maximum */
963     if (sweep < (NO_SWEEP_RATES - 1))  {
964         /* not at maximum, adjust delay for new sweep */
965     adjust_trg_delay(sweep, (sweep + 1));
966     /* now set new sweep rate */
967         sweep++;
968     }
969
970     /* set the sweep rate for the hardware */
971     sample_size = set_sample_rate(sweep_rates[sweep].sample_rate);
972     /* also set the sample size for the trace capture */
973     set_trace_size(sample_size);
974
975
976     /* all done with raising the sweep rate - return */
977     return;
978
979 }
980
981
982
983
984 /*
985     display_sweep
986
987     Description:      This function displays the current sweep rate at the
988                       passed position, in the passed style.
989
990     Arguments:        x_pos (int) - x position (in character cells) at which to
991                   display the sweep rate.
992               y_pos (int) - y position (in character cells) at which to
993                   display the sweep rate.
994               style (int) - style with which to display the sweep rate.
995     Return Value:     None.
996
997     Input:            None.
998     Output:           The sweep rate is displayed at the passed position on the
999               display.
1000
1001    Error Handling:   None.
1002
1003    Algorithms:       None.
1004    Data Structures:  None.
```

```
1005
1006    Global Variables: sweep - determines which string is displayed.
1007
1008    Author:           Glen George
1009    Last Modified:    Mar. 8, 1994
1010
1011 */
1012
1013 void  display_sweep(int x_pos, int y_pos, int style)
1014 {
1015     /* variables */
1016       /* none */
1017
1018
1019
1020     /* display the sweep rate */
1021     plot_string(x_pos, y_pos, sweep_rates[sweep].s, style);
1022
1023
1024     /* all done displaying the sweep rate - return */
1025     return;
1026
1027 }
1028
1029
1030
1031
1032 /*
1033    set_trg_level
1034
1035    Description:      This function sets the trigger level to the passed value.
1036
1037    Arguments:        l (int) - value to which to set the trigger level.
1038    Return Value:     None.
1039
1040    Input:            None.
1041    Output:           None.
1042
1043    Error Handling:   The passed value is not checked for validity.
1044
1045    Algorithms:       None.
1046    Data Structures:  None.
1047
1048    Global Variables: level - initialized to the passed value.
1049
1050    Author:           Glen George
1051    Last Modified:    Mar. 8, 1994
1052
1053 */
1054
1055 void  set_trg_level(int l)
1056 {
1057     /* variables */
1058       /* none */
1059
1060
1061
1062     /* set the trigger level */
1063     level = l;
1064
1065     /* set the trigger level in hardware too */
1066     set_trigger(level, slope);
1067
1068
1069     /* all done initializing the trigger level - return */
1070     return;
1071
1072 }
1073
1074
1075
1076
1077 /*
1078    trg_level_down
1079
1080    Description:      This function handles decreasing the current trigger
1081              level.  The new trigger level is sent to the hardware.
1082              If an attempt is made to lower the trigger level below
1083              the minimum value it is not changed.
1084
1085    Arguments:        None.
```

```
1086    Return Value:     None.
1087
1088    Input:            None.
1089    Output:           None.
1090
1091    Error Handling:   None.
1092
1093    Algorithms:       None.
1094    Data Structures:  None.
1095
1096    Global Variables: level - decremented if not already at the minimum value.
1097
1098    Author:           Glen George
1099    Last Modified:    Mar. 8, 1994
1100
1101 */
1102
1103 void  trg_level_down()
1104 {
1105     /* variables */
1106       /* none */
1107
1108
1109
1110     /* decrease the trigger level, if not already the minimum */
1111     if (level > MIN_TRG_LEVEL_SET)
1112         level--;
1113
1114     /* set the trigger level for the hardware */
1115     set_trigger(level, slope);
1116
1117
1118     /* all done with lowering the trigger level - return */
1119     return;
1120
1121 }
1122
1123
1124
1125
1126 /*
1127    trg_level_up
1128
1129    Description:      This function handles increasing the current trigger
1130              level.  The new trigger level is sent to the hardware.
1131              If an attempt is made to raise the trigger level above
1132              the maximum value it is not changed.
1133
1134    Arguments:        None.
1135    Return Value:     None.
1136
1137    Input:            None.
1138    Output:           None.
1139
1140    Error Handling:   None.
1141
1142    Algorithms:       None.
1143    Data Structures:  None.
1144
1145    Global Variables: level - incremented if not already the maximum value.
1146
1147    Author:           Glen George
1148    Last Modified:    Mar. 8, 1994
1149
1150 */
1151
1152 void  trg_level_up()
1153 {
1154     /* variables */
1155       /* none */
1156
1157
1158
1159     /* increase the trigger level, if not already the maximum */
1160     if (level < MAX_TRG_LEVEL_SET)
1161         level++;
1162
1163     /* tell the hardware the new trigger level */
1164     set_trigger(level, slope);
1165
1166
```

```
1167     /* all done raising the trigger level - return */
1168     return;
1169
1170 }
1171
1172
1173
1174
1175 /*
1176    display_trg_level
1177
1178    Description:      This function displays the current trigger level at the
1179                      passed position, in the passed style.
1180
1181    Arguments:        x_pos (int) - x position (in character cells) at which to
1182                    display the trigger level.
1183              y_pos (int) - y position (in character cells) at which to
1184                    display the trigger level.
1185              style (int) - style with which to display the trigger
1186                       level.
1187    Return Value:     None.
1188
1189    Input:            None.
1190    Output:           The trigger level is displayed at the passed position on
1191              the display.
1192
1193    Error Handling:   None.
1194
1195    Algorithms:       None.
1196    Data Structures:  None.
1197
1198    Global Variables: level - determines the value displayed.
1199
1200    Author:           Glen George
1201    Last Modified:    Mar. 10, 1995
1202
1203 */
1204
1205 void  display_trg_level(int x_pos, int y_pos, int style)
1206 {
1207     /* variables */
1208     char      level_str[] = "        "; /* string containing the trigger level */
1209     long int  l;            /* trigger level in mV */
1210
1211
1212
1213     /* compute the trigger level in millivolts */
1214     l = ((long int) MAX_LEVEL - MIN_LEVEL) * level / (MAX_TRG_LEVEL_SET - MIN_TRG_LEVEL_SET) + ↩
         MIN_LEVEL;
1215
1216     /* convert the level to the string (leave first character blank) */
1217     cvt_num_field(l, &level_str[1]);
1218
1219     /* add in the units */
1220     level_str[7] = 'V';
1221
1222
1223     /* now finally display the trigger level */
1224     plot_string(x_pos, y_pos, level_str, style);
1225
1226
1227     /* all done displaying the trigger level - return */
1228     return;
1229
1230 }
1231
1232
1233
1234
1235 /*
1236    set_trg_slope
1237
1238    Description:      This function sets the trigger slope to the passed value.
1239
1240    Arguments:        s (enum slope_type) - trigger slope type to which to set
1241                        the locally global slope.
1242    Return Value:     None.
1243
1244    Input:            None.
1245    Output:           None.
1246
```

```
1247    Error Handling:   None.
1248
1249    Algorithms:       None.
1250    Data Structures:  None.
1251
1252    Global Variables: slope - set to the passed value.
1253
1254    Author:           Glen George
1255    Last Modified:    Mar. 8, 1994
1256
1257 */
1258
1259 void  set_trg_slope(enum slope_type s)
1260 {
1261     /* variables */
1262       /* none */
1263
1264
1265
1266     /* set the slope type */
1267     slope = s;
1268
1269     /* also tell the hardware what the slope is */
1270     set_trigger(level, slope);
1271
1272
1273     /* all done setting the trigger slope - return */
1274     return;
1275
1276 }
1277
1278
1279
1280
1281 /*
1282    trg_slope_toggle
1283
1284    Description:      This function handles toggling (and setting) the current
1285              trigger slope.
1286
1287    Arguments:        None.
1288    Return Value:     None.
1289
1290    Input:            None.
1291    Output:           None.
1292
1293    Error Handling:   None.
1294
1295    Algorithms:       None.
1296    Data Structures:  None.
1297
1298    Global Variables: slope - toggled.
1299
1300    Author:           Glen George
1301    Last Modified:    Mar. 8, 1994
1302
1303 */
1304
1305 void  trg_slope_toggle()
1306 {
1307     /* variables */
1308       /* none */
1309
1310
1311
1312     /* toggle the trigger slope */
1313     if (slope == SLOPE_POSITIVE)
1314         slope = SLOPE_NEGATIVE;
1315     else
1316         slope = SLOPE_POSITIVE;
1317
1318     /* set the new trigger slope */
1319     set_trigger(level, slope);
1320
1321
1322     /* all done with the trigger slope - return */
1323     return;
1324
1325 }
1326
1327
```

```
1328
1329
1330 /*
1331    display_trg_slope
1332
1333    Description:      This function displays the current trigger slope at the
1334                      passed position, in the passed style.
1335
1336    Arguments:        x_pos (int) - x position (in character cells) at which to
1337                    display the trigger slope.
1338              y_pos (int) - y position (in character cells) at which to
1339                    display the trigger slope.
1340              style (int) - style with which to display the trigger
1341                        slope.
1342    Return Value:     None.
1343
1344    Input:            None.
1345    Output:           The trigger slope is displayed at the passed position on
1346              the screen.
1347
1348    Error Handling:   None.
1349
1350    Algorithms:       None.
1351    Data Structures:  None.
1352
1353    Global Variables: slope - determines which string is displayed.
1354
1355    Author:           Glen George
1356    Last Modified:    Mar. 13, 1994
1357
1358 */
1359
1360 void  display_trg_slope(int x_pos, int y_pos, int style)
1361 {
1362     /* variables */
1363
1364     /* the trigger slope strings (must match enumerated type) */
1365     const char * const  slopes[] =  {  " +", " -"  };
1366
1367
1368
1369     /* display the trigger slope */
1370     plot_string(x_pos, y_pos, slopes[slope], style);
1371
1372
1373     /* all done displaying the trigger slope - return */
1374     return;
1375
1376 }
1377
1378
1379
1380
1381 /*
1382    set_trg_delay
1383
1384    Description:      This function sets the trigger delay to the passed value.
1385
1386    Arguments:        d (long int) - value to which to set the trigger delay.
1387    Return Value:     None.
1388
1389    Input:            None.
1390    Output:           None.
1391
1392    Error Handling:   The passed value is not checked for validity.
1393
1394    Algorithms:       None.
1395    Data Structures:  None.
1396
1397    Global Variables: delay - initialized to the passed value.
1398
1399    Author:           Glen George
1400    Last Modified:    Mar. 8, 1994
1401
1402 */
1403
1404 void  set_trg_delay(long int d)
1405 {
1406     /* variables */
1407       /* none */
1408
```

```
1409
1410
1411        /* set the trigger delay */
1412        delay = d;
1413
1414        /* set the trigger delay in hardware too */
1415        set_delay(delay);
1416
1417
1418        /* all done initializing the trigger delay - return */
1419        return;
1420
1421    }
1422
1423
1424
1425
1426    /*
1427       trg_delay_down
1428
1429       Description:      This function handles decreasing the current trigger
1430                 delay.  The new trigger delay is sent to the hardware.
1431                 If an attempt is made to lower the trigger delay below
1432                 the minimum value it is not changed.
1433
1434       Arguments:        None.
1435       Return Value:     None.
1436
1437       Input:            None.
1438       Output:           None.
1439
1440       Error Handling:   None.
1441
1442       Algorithms:       None.
1443       Data Structures:  None.
1444
1445       Global Variables: delay - decremented if not already at the minimum value.
1446
1447       Author:           Glen George
1448       Last Modified:    Mar. 8, 1994
1449
1450    */
1451
1452    void  trg_delay_down()
1453    {
1454        /* variables */
1455          /* none */
1456
1457
1458
1459        /* decrease the trigger delay, if not already the minimum */
1460        if (delay > MIN_DELAY)
1461            delay--;
1462
1463        /* set the trigger delay for the hardware */
1464        set_delay(delay);
1465
1466
1467        /* all done with lowering the trigger delay - return */
1468        return;
1469
1470    }
1471
1472
1473
1474
1475    /*
1476       trg_delay_up
1477
1478       Description:      This function handles increasing the current trigger
1479                 delay.  The new trigger delay is sent to the hardware.
1480                 If an attempt is made to raise the trigger delay above
1481                 the maximum value it is not changed.
1482
1483       Arguments:        None.
1484       Return Value:     None.
1485
1486       Input:            None.
1487       Output:           None.
1488
1489       Error Handling:   None.
```

```
1490
1491    Algorithms:       None.
1492    Data Structures:  None.
1493
1494    Global Variables: delay - incremented if not already the maximum value.
1495
1496    Author:           Glen George
1497    Last Modified:    Mar. 8, 1994
1498
1499 */
1500
1501 void  trg_delay_up()
1502 {
1503     /* variables */
1504       /* none */
1505
1506
1507
1508     /* increase the trigger delay, if not already the maximum */
1509     if (delay < MAX_DELAY)
1510         delay++;
1511
1512     /* tell the hardware the new trigger delay */
1513     set_delay(delay);
1514
1515
1516     /* all done raising the trigger delay - return */
1517     return;
1518
1519 }
1520
1521
1522
1523
1524 /*
1525    adjust_trg_delay
1526
1527    Description:      This function adjusts the trigger delay for a new sweep
1528              rate.  The factor to adjust the delay by is determined
1529              by looking up the sample rates in the sweep_rates array.
1530              If the delay goes out of range, due to the adjustment it
1531              is reset to the maximum or minimum valid value.
1532
1533    Arguments:        old_sweep (int) - old sweep rate (index into sweep_rates
1534                        array).
1535              new_sweep (int) - new sweep rate (index into sweep_rates
1536                        array).
1537    Return Value:     None.
1538
1539    Input:            None.
1540    Output:           None.
1541
1542    Error Handling:   None.
1543
1544    Algorithms:       The delay is multiplied by 10 times the ratio of the
1545              sweep sample rates then divided by 10.  This is done to
1546              avoid floating point arithmetic and integer truncation
1547              problems.
1548    Data Structures:  None.
1549
1550    Global Variables: delay - adjusted based on passed sweep rates.
1551
1552    Known Bugs:       The updated delay time is not displayed.  Since the time
1553              is typically only rounded to the new sample rate, this is
1554              not a major problem.
1555
1556    Author:           Glen George
1557    Last Modified:    Mar. 8, 1994
1558
1559 */
1560
1561 void  adjust_trg_delay(int old_sweep, int new_sweep)
1562 {
1563     /* variables */
1564       /* none */
1565
1566
1567
1568     /* multiply by 10 times the ratio of sweep rates */
1569     delay *= (10 * sweep_rates[new_sweep].sample_rate) / sweep_rates[old_sweep].sample_rate;
1570     /* now divide the factor of 10 back out */
```

```
1571      delay /= 10;
1572
1573      /* make sure delay is not out of range */
1574      if (delay > MAX_DELAY)
1575          /* delay is too large - set to maximum */
1576          delay = MAX_DELAY;
1577      if (delay < MIN_DELAY)
1578          /* delay is too small - set to minimum */
1579      delay = MIN_DELAY;
1580
1581
1582      /* tell the hardware the new trigger delay */
1583      set_delay(delay);
1584
1585
1586      /* all done adjusting the trigger delay - return */
1587      return;
1588
1589  }
1590
1591
1592
1593
1594  /*
1595     display_trg_delay
1596
1597     Description:      This function displays the current trigger delay at the
1598                       passed position, in the passed style.
1599
1600     Arguments:        x_pos (int) - x position (in character cells) at which to
1601                     display the trigger delay.
1602               y_pos (int) - y position (in character cells) at which to
1603                     display the trigger delay.
1604               style (int) - style with which to display the trigger
1605                        delay.
1606     Return Value:     None.
1607
1608     Input:            None.
1609     Output:           The trigger delay is displayed at the passed position on
1610               the display.
1611
1612     Error Handling:   None.
1613
1614     Algorithms:       None.
1615     Data Structures:  None.
1616
1617     Global Variables: delay - determines the value displayed.
1618
1619     Author:           Glen George
1620     Last Modified:    May 3, 2006
1621
1622  */
1623
1624  void  display_trg_delay(int x_pos, int y_pos, int style)
1625  {
1626      /* variables */
1627      char      delay_str[] = "         "; /* string containing the trigger delay */
1628      long int  units_adj;         /* adjustment to get to microseconds */
1629
1630      long int  d;                          /* delay in appropriate units */
1631
1632
1633      /* compute the delay in the appropriate units */
1634      /* have to watch out for overflow, so be careful */
1635      if (sweep_rates[sweep].sample_rate > 10000000L) {
1636          d = delay * (1000000000L / sweep_rates[sweep].sample_rate);
1637          /* need to divide by 1000000 to get milliseconds */
1638          units_adj = 1000000;
1639      } else if (sweep_rates[sweep].sample_rate > 1000000L)  {
1640          /* have a fast sweep rate, could overflow */
1641          /* first compute in units of 100 ns */
1642          d = delay * (10000000L / sweep_rates[sweep].sample_rate);
1643          /* now convert to nanoseconds */
1644          d *= 100L;
1645          /* need to divide by 1000 to get to microseconds */
1646          units_adj = 1000;
1647      } else  {
1648          /* slow sweep rate, don't have to worry about overflow */
1649          d = delay * (1000000L / sweep_rates[sweep].sample_rate);
1650          /* already in microseconds, so adjustment is 1 */
1651          units_adj = 1;
```

```
1652      }
1653
1654      /* convert it to the string (leave first character blank) */
1655      cvt_num_field(d, &delay_str[1]);
1656
1657      /* add in the units */
1658      if (units_adj == 1000000) {
1659          /* delay is in nanoseconds */
1660          delay_str[7] = '\004';
1661          delay_str[8] = 's';
1662      } else if (((d / units_adj) < 1000) && ((d / units_adj) > -1000) && (units_adj == 1000)) {
1663          /* delay is in microseconds */
1664          delay_str[7] = '\004';
1665          delay_str[8] = 's';
1666      } else if (((d / units_adj) < 1000000) && ((d / units_adj) > -1000000)) {
1667          /* delay is in milliseconds */
1668          delay_str[7] = 'm';
1669          delay_str[8] = 's';
1670      } else if (((d / units_adj) < 1000000000) && ((d / units_adj) > -1000000000))  {
1671          /* delay is in seconds */
1672          delay_str[7] = 's';
1673          delay_str[8] = ' ';
1674      } else  {
1675          /* delay is in kiloseconds */
1676          delay_str[7] = 'k';
1677          delay_str[8] = 's';
1678      }
1679
1680
1681      /* now actually display the trigger delay */
1682      plot_string(x_pos, y_pos, delay_str, style);
1683
1684
1685      /* all done displaying the trigger delay - return */
1686      return;
1687
1688  }
1689
1690
1691
1692
1693  /*
1694     cvt_num_field
1695
1696     Description:      This function converts the passed number (numeric field
1697                       value) to a string and returns that in the passed string
1698               reference.  The number may be signed, and a sign (+ or -)
1699               is always generated.  The number is assumed to have three
1700               digits to the right of the decimal point.  Only the four
1701               most significant digits of the number are displayed and
1702               the decimal point is shifted appropriately.  (Four digits
1703               are always generated by the function).
1704
1705     Arguments:        n (long int) - numeric field value to convert.
1706               s (char *)   - pointer to string in which to return the
1707                            converted field value.
1708     Return Value:     None.
1709
1710     Input:            None.
1711     Output:           None.
1712
1713     Error Handling:   None.
1714
1715     Algorithms:       The algorithm used assumes four (4) digits are being
1716               converted.
1717     Data Structures:  None.
1718
1719     Global Variables: None.
1720
1721     Known Bugs:       If the passed long int is the largest negative long int,
1722               the function will display garbage.
1723
1724     Author:           Glen George
1725     Last Modified:    Mar. 8, 1994
1726
1727  */
1728
1729  void  cvt_num_field(long int n, char *s)
1730  {
1731      /* variables */
1732      int  dp = 3;        /* digits to right of decimal point */
```

```
1733      int  d;         /* digit weight (power of 10) */
1734
1735      int  i = 0;         /* string index */
1736
1737
1738
1739      /* first get the sign (and make n positive for conversion) */
1740      if (n < 0)  {
1741          /* n is negative, set sign and convert to positive */
1742      s[i++] = '-';
1743      n = -n;
1744      }
1745      else  {
1746          /* n is positive, set sign only */
1747      s[i++] = '+';
1748      }
1749
1750
1751      /* make sure there are no more than 4 significant digits */
1752      while (n > 9999)  {
1753          /* have more than 4 digits - get rid of one */
1754      n /= 10;
1755      /* adjust the decimal point */
1756      dp--;
1757      }
1758
1759      /* if decimal point is non-positive, make positive */
1760      /* (assume will take care of adjustment with output units in this case) */
1761      while (dp <= 0)
1762         dp += 3;
1763
1764
1765      /* adjust dp to be digits to the right of the decimal point */
1766      /* (assuming 4 digits) */
1767      dp = 4 - dp;
1768
1769
1770      /* finally, loop getting and converting digits */
1771      for (d = 1000; d > 0; d /= 10)  {
1772
1773          /* check if need decimal the decimal point now */
1774      if (dp-- == 0)
1775          /* time for decimal point */
1776          s[i++] = '.';
1777
1778      /* get and convert this digit */
1779      s[i++] = (n / d) + '0';
1780      /* remove this digit from n */
1781      n %= d;
1782      }
1783
1784
1785      /* all done converting the number, return */
1786      return;
1787
1788  }
```

H.3 Interface and Definitions

H.3.1 macros

```
1  ##############################################################################
2  #                                                                            #
3  #                              Convenient Macros                             #
4  #                  Convenient assembly macros for general use                #
5  #                                  EE/CS 52                                  #
6  #                                                                            #
7  ##############################################################################
8
9
10 /*
11  *  Albert Gural
12  *  EE/CS 52
13  *  TA: Dan Pipe-Mazo
14  *
15  *  File Description:   Provides macros for convenience.
16  *
17  *  Revision History:
18  *      05/14/2014  Albert Gural    Wrote initial macros (PUSH, POP, SAVE, RESTORE).
19  *      06/02/2014  Albert Gural    Updated with more macros (STWI, MOVWI).
20  *
21  */
22
23
24 /*
25  *  STWI
26  *
27  *  Description: Acts like sthio, but for word-size values. Also, can be
28  *  directly supplied with the address and immediage value (no need for
29  *  the value to be passed in a register).
30  *
31  *  Arguments:
32  *  addr - immediate that gives the address of the memory to write to
33  *  val  - immediate that gives the value to write to that memory
34  *
35  *  Return Value: (none)
36  *
37  */
38
39 .macro STWI addr, val
40     PUSH    r9
41     PUSH    r10
42
43     MOVWI   r9, \addr
44     MOVWI   r10, \val
45     stwio   r10, (r9)
46
47     POP     r10
48     POP     r9
49 .endm
50
51
52 /*
53  *  MOVWI
54  *
55  *  Description: Acts like movhi, but for word-size values.
56  *
57  *  Arguments:
58  *  reg - register to write value to
59  *  val - value to update register with
60  *
61  *  Return Value: (none)
62  *
63  */
64
65 .macro MOVWI reg, val
66     movhi   \reg, %hi(\val)
67     ori     \reg, \reg, %lo(\val)
68 .endm
69
70
71 /*
72  *  PUSH
73  *
```

```
 *  Description: Pushes the supplied register on to the stack.
 *
 *  Arguments:
 *  reg - The register
 *
 *  Return Value: (none)
 *
 */

.macro PUSH reg
    subi sp, sp, 4
    stw \reg, 0(sp)
.endm


/*
 *  POP
 *
 *  Description: Pops from the stack to the supplied register
 *
 *  Arguments:
 *  reg - The register
 *
 *  Return Value: (none)
 *
 */

.macro POP reg
    ldw \reg, 0(sp)
    addi sp, sp, 4
.endm


/*
 *  SAVE
 *
 *  Description: Sets up the stack frame correctly so that a function can use
 *  the stack, then restore it to the same it was before the function was called.
 *
 *  Arguments: (none)
 *
 *  Return Value: (none)
 *
 */

.macro SAVE
    PUSH    r31
    PUSH    fp
    mov     fp, sp
.endm


/*
 *  RESTORE
 *
 *  Description: Restores the stack frame to the state it was in before the
 *  matching SAVE was called. This requires that the stack pointer not be
 *  tampered with in any non-trivial way (that is, in a way besides standard
 *  push/pop). Also requires that pushes and pops are always balanced correctly.
 *
 *  Arguments: (none)
 *
 *  Return Value: (none)
 *
 */

.macro RESTORE
    mov     sp, fp
    POP     fp
    POP     r31
.endm
```

H.3.2 pio

```
################################################################################
#                                                                              #
```

```
#                                    pio.m                                     #
#                              PIO Include File                                #
#                                  EE/CS 52                                    #
#                                                                              #
################################################################################

/*
* Albert Gural
* EE/CS 52
* TA: Dan Pipe-Mazo
*
* File Description:
*
* Revision History:
*       02/10/2011      Dan Pipe-Mazo   Initial Revision.
*       05/16/2014      Albert Gural    Modified descriptors.
*
*/

.equ    PIO_IRQ_MASK, 8
.equ    PIO_EDGE_CAP, 12
.equ    PIO_OUTSET, 16
.equ    PIO_OUTCLR, 20
```

H.3.3 scopedef

```
/*****************************************************************************/
/*                                                                           */
/*                               SCOPEDEF.H                                  */
/*                           General Definitions                             */
/*                              Include File                                 */
/*                         Digital Oscilloscope Project                      */
/*                                 EE/CS 52                                  */
/*                                                                           */
/*****************************************************************************/

/*
   This file contains the general definitions for the Digital Oscilloscope
   project.  This includes constant and structure definitions along with the
   function declarations for the assembly language functions.


   Revision History:
      3/8/94   Glen George       Initial revision.
      3/13/94  Glen George       Updated comments.
      3/17/97  Glen George       Removed KEYCODE_UNUSED (no longer used).
      5/3/06   Glen George       Added conditional definitions for handling
                                 different architectures.
      5/9/06   Glen George       Updated declaration of start_sample() to
                                 match the new specification.
      5/27/08  Glen George       Added check for __nios__ definition to also
                                 indicate the compilation is for an Altera
                                 NIOS CPU.
*/



#ifndef  __SCOPEDEF_H__
    #define  __SCOPEDEF_H__


/* library include files */
  /* none */

/* local include files */
#include  "interfac.h"
#include  "lcdout.h"




/* constants */

/* general constants */
#define  FALSE        0
#define  TRUE         !FALSE
```

```
#define  NULL         (void *) 0

/* display size (in characters) */
#define  LCD_WIDTH    (SIZE_X / HORIZ_SIZE)
#define  LCD_HEIGHT   (SIZE_Y / VERT_SIZE)




/* macros */

/* let ___nios___ also mean a NIOS compilation */
#ifdef  ___nios___
  #define  NIOS          /* use the standard NIOS defintion */
#endif

/* add the definitions necessary for the Altera NIOS chip */
#ifdef  NIOS
  #define  FLAT_MEMORY       /* use the flat memory model */
#endif


/* if a flat memory model don't need far pointers */
#ifdef  FLAT_MEMORY
  #define  far
#endif




/* structures, unions, and typedefs */

/* program states */
enum status  {  MENU_ON,     /* menu is displayed with the cursor in it */
        MENU_OFF,    /* menu is not displayed - no cursor */
        NUM_STATES   /* number of states */
         };

/* key codes */
enum keycode  {  KEYCODE_MENU,       /* <Menu>      */
             KEYCODE_UP,         /* <Up>        */
             KEYCODE_DOWN,       /* <Down>      */
             KEYCODE_LEFT,       /* <Left>      */
             KEYCODE_RIGHT,      /* <Right>     */
             KEYCODE_ILLEGAL,    /* other keys */
         NUM_KEYCODES        /* number of key codes */
              };




/* function declarations */

/* keypad functions */
unsigned char key_available(void);                  // key is available
int getkey(void);                                   // get a key

/* display functions  */
void clear_display(void);                           // clear the display
void plot_pixel(unsigned int, unsigned int, int);   // output a pixel

/* sampling parameter functions */
int set_sample_rate(long int);                      // set the sample rate
void set_trigger(int, int);                         // set trigger level and slope
void set_delay(long int);                           // set the trigger delay time

/* sampling functions */
void start_sample(int);                             // capture a sample
unsigned char **sample_done(void);                  // sample captured status


#endif
```

H.3.4 interfac

```
/*************************************************************************/
```

```
/*                                                                      */
/*                              INTERFAC.H                              */
/*                         Interface Definitions                        */
/*                            Include File                              */
/*                       Digital Oscilloscope Project                   */
/*                              EE/CS 52                                */
/*                                                                      */
/************************************************************************/

/*
   This file contains the constants for interfacing between the C code and
   the assembly code/hardware for the Digital Oscilloscope project.  This is
   a sample interface file to allow compilation of the .c files.


   Revision History:
      3/8/94   Glen George       Initial revision.
      3/13/94  Glen George       Updated comments.
      3/17/97  Glen George       Added constant MAX_SAMPLE_SIZE and removed
                                 KEY_UNUSED.
*/



#ifndef  __INTERFAC_H__
    #define  __INTERFAC_H__


/* library include files */
  /* none */

/* local include files */
  /* none */




/* constants */

/* keypad constants */
#define  KEY_MENU        0    /* <Menu>       */
#define  KEY_UP          1    /* <Up>         */
#define  KEY_DOWN        2    /* <Down>       */
#define  KEY_LEFT        3    /* <Left>       */
#define  KEY_RIGHT       4    /* <Right>      */
#define  KEY_ILLEGAL     6    /* illegal key */

/* display constants */
#define  SIZE_X          480     /* size in the x dimension */
#define  SIZE_Y          272     /* size in the y dimension */
#define  PIXEL_BLACK     0x0000
#define  PIXEL_WHITE     0xFF7F
#define  PIXEL_GRAY      0x1042
#define  PIXEL_RED       0x1F00
#define  PIXEL_ORANGE    0x7F01
#define  PIXEL_YELLOW    0xFF03
#define  PIXEL_GREEN     0xE003
#define  PIXEL_DGREEN    0xE001
#define  PIXEL_CYAN      0x007F
#define  PIXEL_BLUE      0x007C
#define  PIXEL_PURPLE    0x147C
#define  PIXEL_VIOLET    0x1F7C

#define  PIXEL_BGND      0x001C
#define  PIXEL_A         0xFF7F
#define  PIXEL_B         0x007F
#define  PIXEL_L1        0x1F7C
#define  PIXEL_L2        0x147C
#define  PIXEL_L3        0x007C

/* scope parameters */
#define  MIN_DELAY      -240     /* minimum trigger delay */
#define  MAX_DELAY      50000    /* maximum trigger delay */
#define  MIN_LEVEL      0        /* minimum trigger level (in mV) */
#define  MAX_LEVEL      5000     /* maximum trigger level (in mV) */

/* sampling parameters */
#define  MAX_SAMPLE_SIZE   2400 /* maximum size of a sample (in samples) */

/* useful macros */
#define max(a,b) \
```

```
83      ({ __typeof__ (a) _a = (a); \
84          __typeof__ (b) _b = (b); \
85        _a > _b ? _a : _b; })
86  #define min(a,b) \
87      ({ __typeof__ (a) _a = (a); \
88          __typeof__ (b) _b = (b); \
89        _a < _b ? _a : _b; })
90
91  #endif
```

H.4 Key Processing

H.4.1 keyproc

```
/*****************************************************************/
/*                                                               */
/*                            KEYPROC.H                          */
/*                     Key Processing Functions                  */
/*                          Include File                         */
/*                    Digital Oscilloscope Project               */
/*                            EE/CS 52                           */
/*                                                               */
/*****************************************************************/

/*
   This file contains the constants and function prototypes for the key
   processing functions (defined in keyproc.c) for the Digital Oscilloscope
   project.


   Revision History:
      3/8/94   Glen George       Initial revision.
      3/13/94  Glen George       Updated comments.
*/



#ifndef  __KEYPROC_H__
    #define  __KEYPROC_H__


/* library include files */
  /* none */

/* local include files */
#include  "scopedef.h"




/* constants */
    /* none */



/* structures, unions, and typedefs */
    /* none */



/* function declarations */

enum status  no_action(enum status);      /* nothing to do */

enum status  menu_key(enum status);   /* process the <Menu> key */

enum status  menu_up(enum status);    /* <Up> key in a menu */
enum status  menu_down(enum status);      /* <Down> key in a menu */
enum status  menu_left(enum status);      /* <Left> key in a menu */
enum status  menu_right(enum status);     /* <Right> key in a menu */


#endif
```

```
/*****************************************************************/
/*                                                               */
/*                            KEYPROC                            */
/*                     Key Processing Functions                  */
/*                    Digital Oscilloscope Project               */
/*                            EE/CS 52                           */
/*                                                               */
/*****************************************************************/

```

```
/*
   This file contains the key processing functions for the Digital
   Oscilloscope project.  These functions are called by the main loop of the
   system.  The functions included are:
      menu_down  - process the <Down> key while in a menu
      menu_key   - process the <Menu> key
      menu_left  - process the <Left> key while in a menu
      menu_right - process the <Right> key while in a menu
      menu_up    - process the <Up> key while in a menu
      no_action  - nothing to do

   The local functions included are:
      none

   The locally global variable definitions included are:
      none


   Revision History
      3/8/94   Glen George       Initial revision.
      3/13/94  Glen George       Updated comments.
*/



/* library include files */
  /* none */

/* local include files */
#include  "scopedef.h"
#include  "keyproc.h"
#include  "menu.h"




/*
   no_action

   Description:      This function handles a key when there is nothing to be
                     done.  It just returns.

   Arguments:        cur_state (enum status) - the current system state.
   Return Value:     (enum status) - the new system state (same as current
             state).

   Input:            None.
   Output:           None.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: None.

   Author:           Glen George
   Last Modified:    Mar. 8, 1994

*/

enum status  no_action(enum status cur_state)
{
    /* variables */
      /* none */



    /* return the current state */
    return  cur_state;

}




/*
   menu_key

   Description:      This function handles the <Menu> key.  If the passed
                     state is MENU_ON, the menu is turned off.  If the passed
```

```
91              state is MENU_OFF, the menu is turned on.  The returned
92              state is the "opposite" of the passed state.
93
94      Arguments:          cur_state (enum status) - the current system state.
95      Return Value:       (enum status) - the new system state ("opposite" of the
96              as current state).
97
98      Input:              None.
99      Output:             The menu is either turned on or off.
100
101     Error Handling:     None.
102
103     Algorithms:         None.
104     Data Structures:    None.
105
106     Global Variables:   None.
107
108     Author:             Glen George
109     Last Modified:      Mar. 8, 1994
110
111 */
112
113 enum status  menu_key(enum status cur_state)
114 {
115     /* variables */
116       /* none */
117
118
119
120     /* check if need to turn the menu on or off */
121     if (cur_state == MENU_ON)
122         /* currently the menu is on, turn it off */
123     clear_menu();
124     else
125         /* currently the menu is off, turn it on */
126     display_menu();
127
128
129     /* all done, return the "opposite" of the current state */
130     if (cur_state == MENU_ON)
131         /* state was MENU_ON, change it to MENU_OFF */
132         return  MENU_OFF;
133     else
134         /* state was MENU_OFF, change it to MENU_ON */
135         return  MENU_ON;
136
137 }
138
139
140
141
142 /*
143     menu_up
144
145     Description:        This function handles the <Up> key when in a menu.  It
146                         goes to the previous menu entry and leaves the system
147             state unchanged.
148
149     Arguments:          cur_state (enum status) - the current system state.
150     Return Value:       (enum status) - the new system state (same as current
151             state).
152
153     Input:              None.
154     Output:             The menu display is updated.
155
156     Error Handling:     None.
157
158     Algorithms:         None.
159     Data Structures:    None.
160
161     Global Variables:   None.
162
163     Author:             Glen George
164     Last Modified:      Mar. 8, 1994
165
166 */
167
168 enum status  menu_up(enum status cur_state)
169 {
170     /* variables */
171       /* none */
```

```
172
173
174
175     /* go to the previous menu entry */
176     previous_entry();
177
178
179     /* return the current state */
180     return  cur_state;
181
182 }
183
184
185
186
187 /*
188    menu_down
189
190    Description:      This function handles the <Down> key when in a menu.  It
191                      goes to the next menu entry and leaves the system state
192              unchanged.
193
194    Arguments:        cur_state (enum status) - the current system state.
195    Return Value:     (enum status) - the new system state (same as current
196              state).
197
198    Input:            None.
199    Output:           The menu display is updated.
200
201    Error Handling:   None.
202
203    Algorithms:       None.
204    Data Structures:  None.
205
206    Global Variables: None.
207
208    Author:           Glen George
209    Last Modified:    Mar. 8, 1994
210
211 */
212
213 enum status  menu_down(enum status cur_state)
214 {
215     /* variables */
216       /* none */
217
218
219
220     /* go to the next menu entry */
221     next_entry();
222
223
224     /* return the current state */
225     return  cur_state;
226
227 }
228
229
230
231
232 /*
233    menu_left
234
235    Description:      This function handles the <Left> key when in a menu.  It
236                      invokes the left function for the current menu entry and
237              leaves the system state unchanged.
238
239    Arguments:        cur_state (enum status) - the current system state.
240    Return Value:     (enum status) - the new system state (same as current
241              state).
242
243    Input:            None.
244    Output:           The menu display may be updated.
245
246    Error Handling:   None.
247
248    Algorithms:       None.
249    Data Structures:  None.
250
251    Global Variables: None.
252
```

```
   Author:           Glen George
   Last Modified:    Mar. 8, 1994

*/

enum status  menu_left(enum status cur_state)
{
    /* variables */
      /* none */



    /* invoke the <Left> key function for the current menu entry */
    menu_entry_left();


    /* return the current state */
    return  cur_state;

}




/*
   menu_right

   Description:      This function handles the <Right> key when in a menu.  It
                     invokes the right function for the current menu entry and
             leaves the system state unchanged.

   Arguments:        cur_state (enum status) - the current system state.
   Return Value:     (enum status) - the new system state (same as current
             state).

   Input:            None.
   Output:           The menu display may be updated.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: None.

   Author:           Glen George
   Last Modified:    Mar. 8, 1994

*/

enum status  menu_right(enum status cur_state)
{
    /* variables */
      /* none */



    /* invoke the <Right> key function for the current menu entry */
    menu_entry_right();


    /* return the current state */
    return  cur_state;

}
```

H.4.2 keyint

```
##############################################################################
#                                                                            #
#                           Key Interrupt Handler                            #
#                  Rotary Encoder and Menu Button Routines                   #
#                                EE/CS 52                                    #
#                                                                            #
##############################################################################

```

```
9
10 /*
11  *  Albert Gural
12  *  EE/CS 52
13  *  TA: Dan Pipe-Mazo
14  *
15  *  File Description:  TODO
16  *
17  *  Table of Contents:  TODO
18  *
19  *  Revision History:
20  *      02/09/2012  Dan Pipe-Mazo   Initial Revision.
21  *      05/14/2014  Albert Gural    Begain writing assembly functions to handle
22  *                                  keypress interrupts.
23  *
24  */
25
26  /*  Local Include Files   */
27 #include "macros.m"
28 #include "pio.m"
29 #include "interfac.h"
30 #include "../osc_bsp/system.h"
31
32
33 .section  .text         #start code section
34
35
36 /*
37  *  key_int_installer
38  *
39  *  Description: Installs a callback function for the key interrupts.
40  *
41  *  Arguments: (none)
42  *
43  *  Return Value: (none)
44  *
45  */
46
47 .global key_int_installer
48 .type   key_int_installer, @function
49
50 key_int_installer:
51     SAVE
52
53     # Enable all switch interrupts.
54     movhi   r8, %hi(KEY_INPUT_BASE)
55     ori     r8, r8, %lo(KEY_INPUT_BASE)
56     movhi   r9, %hi(SWITCH_ALL)
57     ori     r9, r9, %lo(SWITCH_ALL)
58     stw     r9, PIO_IRQ_MASK(r8)
59
60     # Install the interrupt handler
61     mov     r4, r0
62     movi    r5, KEY_INPUT_IRQ
63     movhi   r6, %hi(key_handler)
64     ori     r6, r6, %lo(key_handler)
65     mov     r7, r0
66     PUSH    r0
67     call    alt_ic_isr_register
68     POP     r0
69
70 key_int_installer_done:
71     RESTORE
72     ret
73
74
75 /*
76  *  key_handler
77  *
78  *  Description: Callback function for what should happen when a key interrupt occurs.
79  *  For menu keys, the key is saved until automatically requested by the main
80  *  loop code. For the hardware-mimic keys, the desired key action is taken immediately.
81  *
82  *  Arguments: (none)
83  *
84  *  Return Value: (none)
85  *
86  */
87
88 .type key_handler,@function
89
```

```
key_handler:
    SAVE

    # Key should now be available. Update key_press.
    movi    r8, 1
    movia   r9, key_press
    stb     r8, (r9)

    # Clear interrupts.
    movhi   r8, %hi(KEY_INPUT_BASE)
    ori     r8, r8, %lo(KEY_INPUT_BASE)
    stw     r0, PIO_IRQ_MASK(r8)

    # Get the edge capture register.
    movhi   r8, %hi(KEY_INPUT_BASE)
    ori     r8, r8, %lo(KEY_INPUT_BASE)
    ldw     r8, PIO_EDGE_CAP(r8)

    # Check each bit (starting at 0) and see if set.
    movi    r9, 1
    movi    r11, 0

loop_keys:
    and     r10, r8, r9
    bne     r10, r0, key_lookup
    slli    r9, r9, 1
    addi    r11, r11, 1
    br      loop_keys

    # Once the key is found (r11), use the lookup table to set key_value.
key_lookup:
    movia   r8, key_map
    add     r8, r8, r11
    ldb     r8, (r8)

    movia   r10, key_value
    stb     r8, (r10)

    # Do a lookup.
    movia   r4, set_trigger_normal
    movi    r12, 5
    beq     r11, r12, call_action
    movia   r4, set_trigger_single
    movi    r12, 6
    beq     r11, r12, call_action

    movia   r4, trg_slope_toggle
    movi    r12, 7
    beq     r11, r12, call_action
    movia   r4, trg_slope_toggle
    movi    r12, 8
    beq     r11, r12, call_action

    movia   r4, sweep_down
    movi    r12, 10
    beq     r11, r12, call_action
    movia   r4, sweep_up
    movi    r12, 11
    beq     r11, r12, call_action

    movia   r4, trg_level_up
    movi    r12, 16
    beq     r11, r12, call_action
    movia   r4, trg_level_down
    movi    r12, 17
    beq     r11, r12, call_action

    movia   r4, trg_delay_up
    movi    r12, 18
    beq     r11, r12, call_action
    movia   r4, trg_delay_down
    movi    r12, 19
    beq     r11, r12, call_action

key_lookup_cont:
    # Clear the edge capture register (write 1 to clear).
    movhi   r8, %hi(KEY_INPUT_BASE)
    ori     r8, r8, %lo(KEY_INPUT_BASE)
    movhi   r9, %hi(SWITCH_ALL)
    ori     r9, r9, %lo(SWITCH_ALL)
    stw     r9, PIO_EDGE_CAP(r8)
```

```
171
172     # Re-enable interrupts.
173     movhi   r8, %hi(KEY_INPUT_BASE)
174     ori     r8, r8, %lo(KEY_INPUT_BASE)
175     movhi   r9, %hi(SWITCH_ALL)
176     ori     r9, r9, %lo(SWITCH_ALL)
177     stw     r9, PIO_IRQ_MASK(r8)
178
179 key_handler_done:
180     RESTORE
181     ret
182
183 /* For certain keys, no menu action is desired. In these cases, the action
184  * should be directly performed; not stored for later. This function calls a
185  * particular action, saving and restoring the necessary registers. */
186 call_action:
187     PUSH    r8
188     PUSH    r9
189     PUSH    r10
190     PUSH    r11
191     PUSH    r12
192     PUSH    r13
193     PUSH    r14
194     PUSH    r15
195
196     callr   r4
197
198     POP     r15
199     POP     r14
200     POP     r13
201     POP     r12
202     POP     r11
203     POP     r10
204     POP     r9
205     POP     r8
206     br      key_lookup_cont
207
208
209 /*
210  *  key_available
211  *
212  *  Description: Returns whether a key is available.
213  *
214  *  Arguments: (none)
215  *
216  *  Return Value:
217  *  0 - no key available
218  *  1 - key available
219  *
220  */
221
222 .global key_available
223 .type   key_available, @function
224
225 key_available:
226     SAVE
227
228     # Simply return the value in key_press.
229     movia   r2, key_press
230     ldb     r2, (r2)
231
232 key_available_done:
233     RESTORE
234     ret
235
236
237 /*
238  *  get_key
239  *
240  *  Description: Waits until a key is available, then returns that key code.
241  *
242  *  Arguments: (none)
243  *
244  *  Return Value: key code value
245  *
246  */
247
248 .global getkey
249 .type   getkey, @function
250
251 getkey:
```

```
252     SAVE
253
254     # Block until legal key arrives (which is also when key_press = TRUE).
255     movia   r8, key_value
256     ldb     r8, (r8)
257     movi    r9, KEY_ILLEGAL
258     beq     r8, r9, getkey
259
260     # Get return value.
261     movia   r2, key_value
262     ldb     r2, (r2)
263
264     # Update key_value with KEY_ILLEGAL.
265     movia   r10, key_value
266     stb     r9, (r10)
267
268     # Update key_press with FALSE.
269     movia   r10, key_press
270     stb     r0, (r10)
271
272 getkey_done:
273     RESTORE
274     ret
275
276
277 /*
278  *  key_map
279  *
280  *  Description: Lookup table for the key I/O bit to a key code value.
281  *
282  */
283
284 key_map:
285             .byte   KEY_MENU
286             .byte   KEY_UP
287             .byte   KEY_DOWN
288             .byte   KEY_LEFT
289             .byte   KEY_RIGHT
290             .byte   KEY_MENU
291             .byte   KEY_MENU
292             .byte   KEY_MENU
293             .byte   KEY_MENU
294             .byte   KEY_MENU
295             .byte   KEY_MENU
296             .byte   KEY_MENU
297             .byte   KEY_MENU
298             .byte   KEY_MENU
299             .byte   KEY_MENU
300             .byte   KEY_MENU
301             .byte   KEY_MENU
302             .byte   KEY_MENU
303             .byte   KEY_MENU
304             .byte   KEY_MENU
305             .byte   KEY_ILLEGAL
306
307
308 .section  .data     #start data section
309
310 key_press:  .byte   0       # Gives whether a key has been pressed.
311 key_value:  .byte   0       # Gives the value of the pressed key.
```

H.5 Display Processing

H.5.1 lcdout

```
/*****************************************************************************/
/*                                                                           */
/*                                LCDOUT.H                                   */
/*                          LCD Output Functions                             */
/*                             Include File                                  */
/*                       Digital Oscilloscope Project                        */
/*                              EE/CS  52                                    */
/*                                                                           */
/*****************************************************************************/

/*
   This file contains the constants and function prototypes for the LCD output
   functions used in the Digital Oscilloscope project and defined in lcdout.c.


   Revision History:
      3/8/94   Glen George       Initial revision.
      3/13/94  Glen George       Updated comments.
      3/17/97  Glen George       Added enumerated type char_style and updated
                                    function prototypes.
*/



#ifndef  __LCDOUT_H__
    #define  __LCDOUT_H__


/* library include files */
  /* none */

/* local include files */
  /* none */



/* constants */

/* character output styles */

/* size of a character (includes 1 pixel space to the left and below character) */
#define  VERT_SIZE    8          /* vertical size (in pixels -> 7+1) */
#define  HORIZ_SIZE   6          /* horizontal size (in pixels -> 5+1) */



/* structures, unions, and typedefs */

/* character output styles */
enum  char_style  {  NORMAL,     /* "normal video" */
                     REVERSE     /* "reverse video" */
              };



/* function declarations */

void  clear_region(int, int, int, int);       /* clear part of the display */

void  plot_hline(int, int, int);          /* draw a horizontal line */
void  plot_vline(int, int, int);          /* draw a vertical line */

void  plot_char(int, int, char, enum char_style); /* output a character */
void  plot_string(int, int, const char *, enum char_style);  /* output a string */


#endif
```

```
/*************************************************************************/
/*                                                                       */
/*                                 LCDOUT                                */
/*                          LCD Output Functions                         */
/*                       Digital Oscilloscope Project                    */
/*                                EE/CS 52                               */
/*                                                                       */
/*************************************************************************/

/*
   This file contains the functions for doing output to the LCD screen for the
   Digital Oscilloscope project.  The functions included are:
      clear_region - clear a region of the display
      plot_char    - output a character
      plot_hline   - draw a horizontal line
      plot_string  - output a string
      plot_vline   - draw a vertical line

   The local functions included are:
      none

   The locally global variable definitions included are:
      none


   Revision History
      3/8/94   Glen George       Initial revision.
      3/13/94  Glen George       Updated comments.
      3/13/94  Glen George       Simplified code in plot_string function.
      3/17/97  Glen George       Updated comments.
      3/17/97  Glen George       Change plot_char() and plot_string() to use
                                  enum char_style instead of an int value.
      5/27/98  Glen George       Change plot_char() to explicitly declare the
                                  size of the external array to avoid linker
                                  errors.
*/



/* library include files */
  /* none */

/* local include files */
#include  "interfac.h"
#include  "scopedef.h"
#include  "lcdout.h"




/*
   clear_region

   Description:      This function clears the passed region of the display.
                     The region is described by its upper left corner pixel
                     coordinate and the size (in pixels) in each dimension.

   Arguments:        x_ul (int)   - x coordinate of upper left corner of the
                                    region to be cleared.
                     y_ul (int)   - y coordinate of upper left corner of the
                                    region to be cleared.
                     x_size (int) - horizontal size of the region.
                     y_size (int) - vertical size of the region.
   Return Value:     None.

   Input:            None.
   Output:           A portion of the screen is cleared (set to PIXEL_WHITE).

   Error Handling:   No error checking is done on the coordinates.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: None.

   Author:           Glen George
   Last Modified:    Mar. 8, 1994

*/

```

```
void  clear_region(int x_ul, int y_ul, int x_size, int y_size)
{
    /* variables */
    int  x;     /* x coordinate to clear */
    int  y;     /* y coordinate to clear */



    /* loop, clearing the display */
    for (x = x_ul; x < (x_ul + x_size); x++)  {
        for (y = y_ul; y < (y_ul + y_size); y++)  {

        /* clear this pixel */
        plot_pixel(x, y, PIXEL_BGND);
        }
    }


    /* done clearing the display region - return */
    return;

}




/*
   plot_hline

   Description:      This function draws a horizontal line from the passed
                     position for the passed length.  The line is always drawn
                     with the color PIXEL_BLACK.  The position (0,0) is the
             upper left corner of the screen.

   Arguments:        start_x (int) - starting x coordinate of the line.
             start_y (int) - starting y coordinate of the line.
             length (int)  - length of the line (positive for a line
                         to the "right" and negative for a line to
                     the "left").
   Return Value:     None.

   Input:            None.
   Output:           A horizontal line is drawn at the specified position.

   Error Handling:   No error checking is done on the coordinates.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: None.

   Author:           Glen George
   Last Modified:    Mar. 7, 1994

*/

void  plot_hline(int start_x, int start_y, int length)
{
    /* variables */
    int  x;     /* x position while plotting */

    int  init_x;    /* starting x position to plot */
    int  end_x;     /* ending x position to plot */



    /* check if a line to the "right" or "left" */
    if (length > 0)  {

        /* line to the "right" - start at start_x, end at start_x + length */
    init_x = start_x;
    end_x = start_x + length;
    }
    else  {

        /* line to the "left" - start at start_x + length, end at start_x */
    init_x = start_x + length;
    end_x = start_x;
    }


```

```
    /* loop, outputting points for the line (always draw to the "right") */
    for (x = init_x; x < end_x; x++)
        /* plot a point of the line */
    plot_pixel(x, start_y, PIXEL_GREEN);


    /* done plotting the line - return */
    return;

}




/*
   plot_vline

   Description:      This function draws a vertical line from the passed
                     position for the passed length.  The line is always drawn
                     with the color PIXEL_BLACK.  The position (0,0) is the
             upper left corner of the screen.

   Arguments:        start_x (int) - starting x coordinate of the line.
             start_y (int) - starting y coordinate of the line.
             length (int)  - length of the line (positive for a line
                         going "down" and negative for a line
                     going "up").
   Return Value:     None.

   Input:            None.
   Output:           A vertical line is drawn at the specified position.

   Error Handling:   No error checking is done on the coordinates.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: None.

   Author:           Glen George
   Last Modified:    Mar. 7, 1994

*/

void  plot_vline(int start_x, int start_y, int length)
{
    /* variables */
    int  y;     /* y position while plotting */

    int  init_y;    /* starting y position to plot */
    int  end_y;     /* ending y position to plot */



    /* check if an "up" or "down" line */
    if (length > 0)  {

        /* line going "down" - start at start_y, end at start_y + length */
    init_y = start_y;
    end_y = start_y + length;
    }
    else  {

        /* line going "up" - start at start_y + length, end at start_y */
    init_y = start_y + length;
    end_y = start_y;
    }


    /* loop, outputting points for the line (always draw "down") */
    for (y = init_y; y < end_y; y++)
        /* plot a point of the line */
    plot_pixel(start_x, y, PIXEL_GREEN);


    /* done plotting the line - return */
    return;

}
```

```
243
244
245 /*
246    plot_char
247
248    Description:      This function outputs the passed character to the LCD
249                      screen at passed location.  The passed location is given
250                      as a character position with (0,0) being the upper left
251              corner of the screen.  The character can be drawn in
252              "normal video" (black on white) or "reverse video" (white
253              on black).
254
255    Arguments:        pos_x (int)             - x coordinate (in character
256                                cells) of the character.
257              pos_y (int)             - y coordinate (in character
258                                cells) of the character.
259              c (char)                - the character to plot.
260              style (enum char_style) - style with which to plot the
261                                      character (NORMAL or REVERSE).
262    Return Value:     None.
263
264    Input:            None.
265    Output:           A character is output to the LCD screen.
266
267    Error Handling:   No error checking is done on the coordinates or the
268              character (to ensure there is a bit pattern for it).
269
270    Algorithms:       None.
271    Data Structures:  The character bit patterns are stored in an external
272              array.
273
274    Global Variables: None.
275
276    Author:           Glen George
277    Last Modified:    May 27, 2008
278
279 */
280
281 void  plot_char(int pos_x, int pos_y, char c, enum char_style style)
282 {
283     /* variables */
284
285     /* pointer to array of character bit patterns */
286     extern const unsigned char   char_patterns[(VERT_SIZE - 1) * 128];
287
288     int  bits;          /* a character bit pattern */
289
290     int  col;       /* column loop index */
291     int  row;           /* character row loop index */
292
293     int  x;     /* x pixel position for the character */
294     int  y;     /* y pixel position for the character */
295
296
297
298     /* setup the pixel positions for the character */
299     x = pos_x * HORIZ_SIZE;
300     y = pos_y * VERT_SIZE;
301
302
303     /* loop outputting the bits to the screen */
304     for (row = 0; row < VERT_SIZE; row++)  {
305
306         /* get the character bits for this row from the character table */
307     if (row == (VERT_SIZE - 1))
308         /* last row - blank it */
309         bits = 0;
310     else
311         /* in middle of character, get the row from the bit patterns */
312             bits = char_patterns[(c * (VERT_SIZE - 1)) + row];
313
314     /* take care of "normal/reverse video" */
315     if (style == REVERSE)
316         /* invert the bits for "reverse video" */
317         bits = ~bits;
318
319         /* get the bits "in position" (high bit is output first */
320     bits <<= (8 - HORIZ_SIZE);
321
322
323     /* now output the row of the character, pixel by pixel */
```

```
    for (col = 0; col < HORIZ_SIZE; col++)  {

            /* output this pixel in the appropriate color */
        if ((bits & 0x80) == 0)
            /* blank pixel - output in PIXEL_WHITE */
        plot_pixel(x + col, y, PIXEL_BGND);
        else
            /* black pixel - output in PIXEL_BLACK */
        plot_pixel(x + col, y, PIXEL_GREEN);

        /* shift the next bit into position */
        bits <<= 1;
        }


    /* next row - update the y position */
    y++;
    }


    /* all done, return */
    return;

}




/*
   plot_string

   Description:      This function outputs the passed string to the LCD screen
                     at passed location.  The passed location is given as a
                     character position with (0,0) being the upper left corner
            of the screen.  There is no line wrapping, so the entire
            string must fit on the passed line (pos_y).  The string
            can be drawn in "normal video" (black on white) or
            "reverse video" (white on black).

   Arguments:        pos_x (int)             - x coordinate (in character
                                cells) of the start of the
                           string.
            pos_y (int)               - y coordinate (in character
                                cells) of the start of the
                           string.
            s (const char *)        - the string to output.
            style (enum char style) - style with which to plot
                                characters of the string.
   Return Value:     None.

   Input:            None.
   Output:           A string is output to the LCD screen.

   Error Handling:   No checking is done to insure the string is fully on the
            screen (the x and y coordinates and length of the string
            are not checked).

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: None.

   Author:           Glen George
   Last Modified:    Mar. 17, 1997

*/

void  plot_string(int pos_x, int pos_y, const char *s, enum char_style style)
{
    /* variables */
      /* none */



    /* loop, outputting characters from string s */
    while (*s != '\0')

        /* output this character and move to the next character and screen position */
    plot_char(pos_x++, pos_y, *s++, style);

```

```
405     /* all done, return */
406     return;
407
408 }
```

H.5.2 char57

```
1  /*****************************************************************************/
2  /*                                                                           */
3  /*                                  CHAR57                                   */
4  /*                            5x7 Dot Matrix Codes                           */
5  /*                         Digital Oscilloscope Project                      */
6  /*                                EE/CS 52                                   */
7  /*                                                                           */
8  /*****************************************************************************/
9
10 /*
11    This file contains a table of dot matrix patterns for vertically scanned
12    5x7 characters.  The table entries are in ASCII order with 7 bytes per
13    character.  The table starts with 32 special characters (mostly blank
14    characters) then space, the start of the printable ASCII character set.
15    The table is called char_patterns.  In each byte (horizontal row) the
16    leftmost pixel is given by bit 4 and the rightmost by bit 0.
17
18
19    Revision History
20       5/27/08  Glen George       Initial revision (from 3/10/95 version of
21                                  char57.asm).
22 */
23
24
25
26
27 /* library include files */
28   /* none */
29
30 /* local include files */
31   /* none */
32
33
34
35 /* the character pattern table */
36 const unsigned char  char_patterns[] = {
37
38     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x00)            */
39     0x04, 0x0E, 0x15, 0x04, 0x04, 0x04, 0x04,   /* up arrow (0x01)          */
40     0x04, 0x04, 0x04, 0x04, 0x15, 0x0E, 0x04,   /* down arrow (0x02)        */
41     0x00, 0x04, 0x08, 0x1F, 0x08, 0x04, 0x00,   /* left arrow (0x03)        */
42     0x00, 0x11, 0x11, 0x11, 0x1B, 0x14, 0x10,   /* greek u (mu) (0x04)      */
43     0x00, 0x04, 0x02, 0x1F, 0x02, 0x04, 0x00,   /* right arrow (0x05)       */
44     0x00, 0x11, 0x0A, 0x04, 0x0A, 0x11, 0x00,   /* multiply symbol (0x06)   */
45     0x00, 0x04, 0x00, 0x1F, 0x00, 0x04, 0x00,   /* divide symbol (0x07)     */
46     0x04, 0x04, 0x1F, 0x04, 0x04, 0x00, 0x1F,   /* plus/minus symbol (0x08) */
47     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x09)            */
48     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x0A)            */
49     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x0B)            */
50     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x0C)            */
51     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x0D)            */
52     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x0E)            */
53     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x0F)            */
54     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x10)            */
55     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x11)            */
56     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x12)            */
57     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x13)            */
58     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x14)            */
59     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x15)            */
60     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x16)            */
61     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x17)            */
62     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x18)            */
63     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x19)            */
64     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x1A)            */
65     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x1B)            */
66     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x1C)            */
67     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x1D)            */
68     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x1E)            */
69     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* UNUSED (0x1F)            */
```

```
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,   /* space (0x20)             */
    0x04, 0x04, 0x04, 0x04, 0x04, 0x00, 0x04,   /* !                        */
    0x0A, 0x0A, 0x0A, 0x00, 0x00, 0x00, 0x00,   /* "                        */
    0x0A, 0x0A, 0x1F, 0x0A, 0x1F, 0x0A, 0x0A,   /* #                        */
    0x04, 0x0F, 0x14, 0x0E, 0x05, 0x1E, 0x04,   /* $                        */
    0x18, 0x19, 0x02, 0x04, 0x08, 0x13, 0x03,   /* %                        */
    0x08, 0x14, 0x14, 0x08, 0x15, 0x12, 0x0D,   /* &                        */
    0x0C, 0x0C, 0x08, 0x10, 0x00, 0x00, 0x00,   /* '                        */
    0x02, 0x04, 0x08, 0x08, 0x08, 0x04, 0x02,   /* (                        */
    0x08, 0x04, 0x02, 0x02, 0x02, 0x04, 0x08,   /* )                        */
    0x04, 0x15, 0x0E, 0x1F, 0x0E, 0x15, 0x04,   /* *                        */
    0x00, 0x04, 0x04, 0x1F, 0x04, 0x04, 0x00,   /* +                        */
    0x00, 0x00, 0x00, 0x0C, 0x0C, 0x08, 0x10,   /* ,                        */
    0x00, 0x00, 0x00, 0x1F, 0x00, 0x00, 0x00,   /* -                        */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x0C, 0x0C,   /* .                        */
    0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x00,   /* /                        */
    0x0E, 0x11, 0x13, 0x15, 0x19, 0x11, 0x0E,   /* 0                        */
    0x04, 0x0C, 0x04, 0x04, 0x04, 0x04, 0x0E,   /* 1                        */
    0x0E, 0x11, 0x01, 0x0E, 0x10, 0x10, 0x1F,   /* 2                        */
    0x0E, 0x11, 0x01, 0x06, 0x01, 0x11, 0x0E,   /* 3                        */
    0x02, 0x06, 0x0A, 0x12, 0x1F, 0x02, 0x02,   /* 4                        */
    0x1F, 0x10, 0x1E, 0x01, 0x01, 0x11, 0x0E,   /* 5                        */
    0x06, 0x08, 0x10, 0x1E, 0x11, 0x11, 0x0E,   /* 6                        */
    0x1F, 0x01, 0x02, 0x04, 0x08, 0x10, 0x10,   /* 7                        */
    0x0E, 0x11, 0x11, 0x0E, 0x11, 0x11, 0x0E,   /* 8                        */
    0x0E, 0x11, 0x11, 0x0F, 0x01, 0x02, 0x0C,   /* 9                        */
    0x00, 0x0C, 0x0C, 0x00, 0x0C, 0x0C, 0x00,   /* :                        */
    0x0C, 0x0C, 0x00, 0x0C, 0x0C, 0x08, 0x10,   /* ;                        */
    0x02, 0x04, 0x08, 0x10, 0x08, 0x04, 0x02,   /* <                        */
    0x00, 0x00, 0x1F, 0x00, 0x1F, 0x00, 0x00,   /* =                        */
    0x08, 0x04, 0x02, 0x01, 0x02, 0x04, 0x08,   /* >                        */
    0x0E, 0x11, 0x01, 0x02, 0x04, 0x00, 0x04,   /* ?                        */
    0x0E, 0x11, 0x01, 0x0D, 0x15, 0x15, 0x0E,   /* @                        */
    0x04, 0x0A, 0x11, 0x11, 0x1F, 0x11, 0x11,   /* A                        */
    0x1E, 0x09, 0x09, 0x0E, 0x09, 0x09, 0x1E,   /* B                        */
    0x0E, 0x11, 0x10, 0x10, 0x10, 0x11, 0x0E,   /* C                        */
    0x1E, 0x09, 0x09, 0x09, 0x09, 0x09, 0x1E,   /* D                        */
    0x1F, 0x10, 0x10, 0x1C, 0x10, 0x10, 0x1F,   /* E                        */
    0x1F, 0x10, 0x10, 0x1C, 0x10, 0x10, 0x10,   /* F                        */
    0x0F, 0x10, 0x10, 0x13, 0x11, 0x11, 0x0F,   /* G                        */
    0x11, 0x11, 0x11, 0x1F, 0x11, 0x11, 0x11,   /* H                        */
    0x0E, 0x04, 0x04, 0x04, 0x04, 0x04, 0x0E,   /* I                        */
    0x01, 0x01, 0x01, 0x01, 0x01, 0x11, 0x0E,   /* J                        */
    0x11, 0x12, 0x14, 0x18, 0x14, 0x12, 0x11,   /* K                        */
    0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x1F,   /* L                        */
    0x11, 0x1B, 0x15, 0x15, 0x11, 0x11, 0x11,   /* M                        */
    0x11, 0x19, 0x15, 0x13, 0x11, 0x11, 0x11,   /* N                        */
    0x0E, 0x11, 0x11, 0x11, 0x11, 0x11, 0x0E,   /* O                        */
    0x1E, 0x11, 0x11, 0x1E, 0x10, 0x10, 0x10,   /* P                        */
    0x0E, 0x11, 0x11, 0x11, 0x15, 0x12, 0x0D,   /* Q                        */
    0x1E, 0x11, 0x11, 0x1E, 0x14, 0x12, 0x11,   /* R                        */
    0x0E, 0x11, 0x10, 0x0E, 0x01, 0x11, 0x0E,   /* S                        */
    0x1F, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04,   /* T                        */
    0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x0E,   /* U                        */
    0x11, 0x11, 0x11, 0x0A, 0x0A, 0x04, 0x04,   /* V                        */
    0x11, 0x11, 0x11, 0x11, 0x15, 0x1B, 0x11,   /* W                        */
    0x11, 0x11, 0x0A, 0x04, 0x0A, 0x11, 0x11,   /* X                        */
    0x11, 0x11, 0x0A, 0x04, 0x04, 0x04, 0x04,   /* Y                        */
    0x1F, 0x01, 0x02, 0x04, 0x08, 0x10, 0x1F,   /* Z                        */
    0x0E, 0x08, 0x08, 0x08, 0x08, 0x08, 0x0E,   /* [                        */
    0x00, 0x10, 0x08, 0x04, 0x02, 0x01, 0x00,   /* \                        */
    0x0E, 0x02, 0x02, 0x02, 0x02, 0x02, 0x0E,   /* ]                        */
    0x04, 0x0A, 0x11, 0x00, 0x00, 0x00, 0x00,   /* ^                        */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1F,   /* _                        */
    0x06, 0x06, 0x04, 0x02, 0x00, 0x00, 0x00,   /* `                        */
    0x00, 0x00, 0x0E, 0x01, 0x0F, 0x11, 0x0F,   /* a                        */
    0x10, 0x10, 0x16, 0x19, 0x11, 0x19, 0x16,   /* b                        */
    0x00, 0x00, 0x0E, 0x11, 0x10, 0x11, 0x0E,   /* c                        */
    0x01, 0x01, 0x0D, 0x13, 0x11, 0x13, 0x0D,   /* d                        */
    0x00, 0x00, 0x0E, 0x11, 0x1F, 0x10, 0x0E,   /* e                        */
    0x02, 0x05, 0x04, 0x0E, 0x04, 0x04, 0x04,   /* f                        */
    0x0D, 0x13, 0x13, 0x0D, 0x01, 0x11, 0x0E,   /* g                        */
    0x10, 0x10, 0x16, 0x19, 0x11, 0x11, 0x11,   /* h                        */
    0x04, 0x00, 0x0C, 0x04, 0x04, 0x04, 0x0E,   /* i                        */
    0x01, 0x00, 0x01, 0x01, 0x01, 0x11, 0x0E,   /* j                        */
    0x10, 0x10, 0x12, 0x14, 0x18, 0x14, 0x12,   /* k                        */
    0x0C, 0x04, 0x04, 0x04, 0x04, 0x04, 0x0E,   /* l                        */
    0x00, 0x00, 0x1A, 0x15, 0x15, 0x15, 0x15,   /* m                        */
    0x00, 0x00, 0x16, 0x19, 0x11, 0x11, 0x11,   /* n                        */
    0x00, 0x00, 0x0E, 0x11, 0x11, 0x11, 0x0E,   /* o                        */
    0x16, 0x19, 0x11, 0x19, 0x16, 0x10, 0x10,   /* p                        */
```

```
    0x0D, 0x13, 0x11, 0x13, 0x0D, 0x01, 0x01,   /* q                        */
    0x00, 0x00, 0x16, 0x19, 0x10, 0x10, 0x10,   /* r                        */
    0x00, 0x00, 0x0F, 0x10, 0x0E, 0x01, 0x1E,   /* s                        */
    0x04, 0x04, 0x1F, 0x04, 0x04, 0x05, 0x02,   /* t                        */
    0x00, 0x00, 0x11, 0x11, 0x11, 0x13, 0x0D,   /* u                        */
    0x00, 0x00, 0x11, 0x11, 0x11, 0x0A, 0x04,   /* v                        */
    0x00, 0x00, 0x11, 0x11, 0x15, 0x15, 0x0A,   /* w                        */
    0x00, 0x00, 0x11, 0x0A, 0x04, 0x0A, 0x11,   /* x                        */
    0x11, 0x11, 0x11, 0x0F, 0x01, 0x11, 0x0E,   /* y                        */
    0x00, 0x00, 0x1F, 0x02, 0x04, 0x08, 0x1F,   /* z                        */
    0x02, 0x04, 0x04, 0x08, 0x04, 0x04, 0x02,   /* {                        */
    0x04, 0x04, 0x04, 0x00, 0x04, 0x04, 0x04,   /* |                        */
    0x08, 0x04, 0x04, 0x02, 0x04, 0x04, 0x08,   /* }                        */
    0x08, 0x15, 0x02, 0x00, 0x00, 0x00, 0x00,   /* ~                        */
    0x0A, 0x15, 0x0A, 0x15, 0x0A, 0x15, 0x0A    /* DEL (0x7F)               */

};
```

H.6 Trace Processing

H.6.1 tracutil

```
/*****************************************************************/
/*                                                               */
/*                          TRACUTIL.H                           */
/*                    Trace Utility Functions                    */
/*                         Include File                          */
/*                   Digital Oscilloscope Project                */
/*                           EE/CS 52                            */
/*                                                               */
/*****************************************************************/

/*
   This file contains the constants and function prototypes for the trace
   utility functions (defined in tracutil.c) for the Digital Oscilloscope
   project.


   Revision History:
      3/8/94   Glen George       Initial revision.
      3/13/94  Glen George       Updated comments.
      3/13/94  Glen George       Changed name of set_axes function to
                      set_display_scale.
      5/9/06   Glen George       Added the constants for grids and tick marks.
      5/27/08  Glen George       Added is_sampling() function to be able to
                           tell if the system is currently taking a
                     sample.
      6/3/08   Glen George       Removed Y_SCALE_FACTOR - no longer used to
                           fix problems with non-power of 2 display
                     sizes.
*/



#ifndef  __TRACUTIL_H__
    #define  __TRACUTIL_H__


/* library include files */
  /* none */

/* local include files */
#include  "interfac.h"
#include  "menuact.h"




/* constants */

/* plot size */
#define  PLOT_SIZE_X    SIZE_X      /* plot takes entire screen width */
#define  PLOT_SIZE_Y    SIZE_Y      /* plot takes entire screen height */

/* axes position and size */
#define  X_AXIS_START   0           /* starting x position of x-axis */
#define  X_AXIS_END     (PLOT_SIZE_X - 1)  /* ending x position of x-axis */
#define  X_AXIS_POS (PLOT_SIZE_Y / 2)  /* y position of x-axis */
#define  Y_AXIS_START   0           /* starting y position of y-axis */
#define  Y_AXIS_END     (PLOT_SIZE_Y - 1)  /* ending y position of y-axis */
#define  Y_AXIS_POS (PLOT_SIZE_X / 2)  /* x position of y-axis */

/* tick mark and grid constants */
#define  TICK_LEN       5           /* length of axis tick mark */
/* tick mark counts are for a single quadrant, thus total number of tick */
/* marks or grids is twice this number */
#define  X_TICK_CNT     5       /* always 5 tick marks on x axis */
#define  X_TICK_SIZE    (PLOT_SIZE_X / (2 * X_TICK_CNT))   /* distance between tick marks */
#define  Y_TICK_SIZE    X_TICK_SIZE     /* same size as x */
#define  Y_TICK_CNT     (PLOT_SIZE_Y / (2 * Y_TICK_SIZE))  /* number of y tick marks */
#define  X_GRID_START   0           /* starting x position of x grid */
#define  X_GRID_END     (PLOT_SIZE_X - 1)  /* ending x position of x grid */
#define  Y_GRID_START   0           /* starting y position of y-axis */
#define  Y_GRID_END     (PLOT_SIZE_Y - 1)  /* ending y position of y-axis */

```

```
/* maximum size of the save area (in pixels) */
#define  SAVE_SIZE_X    120 /* maximum width */
#define  SAVE_SIZE_Y     16  /* maximum height */




/* structures, unions, and typedefs */
    /* none */




/* function declarations */

/* initialize the trace utility routines */
void  init_trace(void);

/* trace status functions */
void  set_mode(enum trigger_type);  /* set the triggering mode */
int   is_sampling(void);        /* currently trying to take a sample */
int   trace_rdy(void);              /* determine if ready to start a trace */
void  trace_done(void);             /* signal a trace has been completed */
void  trace_rearm(void);            /* re-enable tracing */

/* trace save area functions */
void  clear_saved_areas(void);        /* clears all saved areas */
void  restore_menu_trace(void);           /* restore the trace under menus */
void  set_save_area(int, int, int, int);  /* set an area of a trace to save */
void  restore_trace(void);                /* restore saved area of a trace */

/* set the scale type */
void  set_display_scale(enum scale_type);

/* setup and plot a trace */
void  set_trace_size(int);              /* set the number of samples in a trace */
void  do_trace(void);                   /* start a trace */
void  plot_trace(unsigned char**);      /* plot a trace (sampled data) */


#endif
```

```
/*****************************************************************************/
/*                                                                           */
/*                                  TRACUTIL                                 */
/*                          Trace Utility Functions                          */
/*                        Digital Oscilloscope Project                       */
/*                                  EE/CS 52                                 */
/*                                                                           */
/*****************************************************************************/

/*
   This file contains the utility functions for handling traces (capturing
   and displaying data) for the Digital Oscilloscope project.  The functions
   included are:
      clear_saved_areas  - clear all the save areas
      do_trace           - start a trace
      init_trace         - initialize the trace routines
      plot_trace         - plot a trace (sampled data)
      restore_menu_trace - restore the saved area under the menus
      restore_trace      - restore the saved area of a trace
      set_display_scale  - set the type of displayed scale (and display it)
      set_mode           - set the triggering mode
      set_save_area      - determine an area of a trace to save
      set_trace_size     - set the number of samples in a trace
      trace_done         - inform this module that a trace has been completed
      trace_rdy          - determine if system is ready to start another trace
      trace_rearm        - re-enable tracing (in one-shot triggering mode)

   The local functions included are:
      none

   The locally global variable definitions included are:
      cur_scale    - current scale type
      sample_size  - the size of the sample for the trace
      sampling     - currently doing a sample
      saved_area   - saved trace under a specified area
      saved_axis_x - saved trace under the x lines (axes or grid)
```

```
37        saved_axis_y - saved trace under the y lines (axes or grid)
38        saved_menu   - saved trace under the menu
39        saved_pos_x  - starting position (x coorindate) of area to save
40        saved_pos_y  - starting position (y coorindate) of area to save
41        saved_end_x  - ending position (x coorindate) of area to save
42        saved_end_y  - ending position (y coorindate) of area to save
43        trace_status - whether or not ready to start another trace
44
45
46     Revision History
47        3/8/94   Glen George       Initial revision.
48        3/13/94  Glen George       Updated comments.
49        3/13/94  Glen George       Fixed inversion of signal in plot_trace.
50        3/13/94  Glen George       Added sampling flag and changed the functions
51                                   init_trace, do_trace and trace_done to update
52                  the flag.  Also the function trace_rdy now
53                  uses it.  The function set_mode was updated
54                  to always say a trace is ready for normal
55                  triggering.
56        3/13/94  Glen George       Fixed bug in trace restoring due to operator
57                                   misuse (&& instead of &) in the functions
58                  set_axes, restore_menu_trace, and
59                  restore_trace.
60        3/13/94  Glen George       Fixed bug in trace restoring due to the clear
61                                   function (clear_saved_areas) not clearing all
62                  of the menu area.
63        3/13/94  Glen George       Fixed comparison bug when saving traces in
64                                   plot_trace.
65        3/13/94  Glen George       Changed name of set_axes to set_display_scale
66                                   and the name of axes_state to cur_scale to
67                  more accurately reflect the function/variable
68                  use (especially if add scale display types).
69        3/17/97  Glen George       Updated comments.
70        3/17/97  Glen George       Changed set_display_scale to use plot_hline
71                                   and plot_vline functions to output axes.
72        5/3/06   Glen George       Updated formatting.
73        5/9/06   Glen George       Updated do_trace function to match the new
74                                   definition of start_sample().
75        5/9/06   Glen George       Removed normal_trg variable, its use is now
76                                   handled by the get_trigger_mode() accessor.
77        5/9/06   Glen George       Added tick marks to the axes display.
78        5/9/06   Glen George       Added ability to display a grid.
79        5/27/08  Glen George       Added is_sampling() function to be able to
80                                  tell if the system is currently taking a
81                  sample.
82        5/27/08  Glen George       Changed set_mode() to always turn off the
83                                  sampling flag so samples with the old mode
84                                      setting are ignored.
85        6/3/08   Glen George       Fixed problems with non-power of 2 display
86                  sizes not working.
87    */
88
89
90    /* library include files */
91      /* none */
92
93    /* local include files */
94    #include  "scopedef.h"
95    #include  "lcdout.h"
96    #include  "menu.h"
97    #include  "menuact.h"
98    #include  "tracutil.h"
99
100
101
102
103   /* locally global variables */
104
105   int  trace_status;  /* ready to start another trace */
106
107   int  sampling;           /* currently sampling data */
108
109   int  sample_size;   /* number of data points in a sample */
110
111   enum scale_type  cur_scale; /* current display scale type */
112
113   /* traces (sampled data) saved under the axes */
114   unsigned char  saved_axis_x[2 * Y_TICK_CNT + 1][PLOT_SIZE_X/8]; /* saved trace under x lines */
115   unsigned char  saved_axis_y[2 * X_TICK_CNT + 1][PLOT_SIZE_Y/8]; /* saved trace under y lines */
116
```

```
/* traces (sampled data) saved under the menu */
unsigned char  saved_menu[MENU_SIZE_Y][(MENU_SIZE_X + 7)/8];

/* traces (sampled data) saved under any area */
unsigned char  saved_area[SAVE_SIZE_Y][SAVE_SIZE_X/8]; /* saved trace under any area */
int       saved_pos_x;    /* starting x position of saved area */
int       saved_pos_y;    /* starting y position of saved area */
int       saved_end_x;    /* ending x position of saved area */
int       saved_end_y;    /* ending y position of saved area */

/* saved traces for quick redraw */
int trace_A[PLOT_SIZE_X];
int trace_B[PLOT_SIZE_X];
int trace_L[PLOT_SIZE_X];
int saved_trace_A[PLOT_SIZE_X];
int saved_trace_B[PLOT_SIZE_X];
int saved_trace_L[PLOT_SIZE_X];


/*
   init_trace

   Description:      This function initializes all of the locally global
                     variables used by these routines.  The saved areas are
             set to non-existant with cleared saved data.  Normal
             normal triggering is set, the system is ready for a
             trace, the scale is turned off and the sample size is set
             to the screen size.

   Arguments:        None.
   Return Value:     None.

   Input:            None.
   Output:           None.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: trace_status - set to TRUE.
             sampling     - set to FALSE.
             cur_scale    - set to SCALE_NONE (no displayed scale).
             sample_size  - set to screen size (SIZE_X).
             saved_axis_x - cleared.
             saved_axis_y - cleared.
             saved_menu   - cleared.
             saved_area   - cleared.
             saved_pos_x  - set to off-screen.
             saved_pos_y  - set to off-screen.
             saved_end_x  - set to off-screen.
             saved_end_y  - set to off-screen.

   Author:           Glen George
   Last Modified:    May 9, 2006

*/

void  init_trace()
{
    /* variables */
      /* none */



    /* initialize system status variables */

    /* ready for a trace */
    trace_status = TRUE;

    /* not currently sampling data */
    sampling = FALSE;

    /* turn off the displayed scale */
    cur_scale = SCALE_NONE;

    /* sample size is the screen size */
    sample_size = SIZE_X;


    /* clear save areas */
```

```
199     clear_saved_areas();
200
201     /* also clear the general saved area location variables (off-screen) */
202     saved_pos_x = SIZE_X + 1;
203     saved_pos_y = SIZE_Y + 1;
204     saved_end_x = SIZE_X + 1;
205     saved_end_y = SIZE_Y + 1;
206
207
208     /* done initializing, return */
209     return;
210
211 }
212
213
214
215
216 /*
217    set_mode
218
219    Description:      This function sets the locally global triggering mode
220                      based on the passed value (one of the possible enumerated
221              values).  The triggering mode is used to determine when
222              the system is ready for another trace.  The sampling flag
223                      is also reset so a new sample will be started (if that is
224                      appropriate).
225
226    Arguments:        trigger_mode (enum trigger_type) - the mode with which to
227                             set the triggering.
228    Return Value:     None.
229
230    Input:            None.
231    Output:           None.
232
233    Error Handling:   None.
234
235    Algorithms:       None.
236    Data Structures:  None.
237
238    Global Variables: sampling     - set to FALSE to turn off sampling
239                      trace_status - set to TRUE if not one-shot triggering.
240
241    Author:           Glen George
242    Last Modified:    May 27, 2008
243
244 */
245
246 void  set_mode(enum trigger_type trigger_mode)
247 {
248     /* variables */
249       /* none */
250
251
252
253     /* if not one-shot triggering - ready for trace too */
254     trace_status = (trigger_mode != ONESHOT_TRIGGER);
255
256
257     /* turn off the sampling flag so will start a new sample */
258     sampling = FALSE;
259
260
261     /* all done, return */
262     return;
263
264 }
265
266
267
268
269 /*
270    is_sampling
271
272    Description:      This function determines whether the system is currently
273                      taking a sample or not.  This is just the value of the
274              sampling flag.
275
276    Arguments:        None.
277    Return Value:     (int) - the current sampling status (TRUE if currently
278              trying to take a sample, FALSE otherwise).
279
```

```
280     Input:              None.
281     Output:             None.
282
283     Error Handling:     None.
284
285     Algorithms:         None.
286     Data Structures:    None.
287
288     Global Variables:   sampling - determines if taking a sample or not.
289
290     Author:             Glen George
291     Last Modified:      May 27, 2008
292
293 */
294
295 int  is_sampling()
296 {
297     /* variables */
298       /* none */
299
300
301
302     /* currently sampling if sampling flag is set */
303     return  sampling;
304
305 }
306
307
308
309
310 /*
311     trace_rdy
312
313     Description:        This function determines whether the system is ready to
314                         start another trace.  This is determined by whether or
315               not the system is still sampling (sampling flag) and if
316               it is ready for another trace (trace_status flag).
317
318     Arguments:          None.
319     Return Value:       (int) - the current trace status (TRUE if ready to do
320               another trace, FALSE otherwise).
321
322     Input:              None.
323     Output:             None.
324
325     Error Handling:     None.
326
327     Algorithms:         None.
328     Data Structures:    None.
329
330     Global Variables:   sampling     - determines if ready for another trace.
331               trace_status - determines if ready for another trace.
332
333     Author:             Glen George
334     Last Modified:      Mar. 13, 1994
335
336 */
337
338 int  trace_rdy()
339 {
340     /* variables */
341       /* none */
342
343
344
345     /* ready for another trace if not sampling and trace is ready */
346     return  (!sampling && trace_status);
347
348 }
349
350
351
352
353 /*
354     trace_done
355
356     Description:        This function is called to indicate a trace has been
357                         completed.  If in normal triggering mode this means the
358               system is ready for another trace.
359
360     Arguments:          None.
```

```
361     Return Value:     None.
362
363     Input:            None.
364     Output:           None.
365
366     Error Handling:   None.
367
368     Algorithms:       None.
369     Data Structures:  None.
370
371     Global Variables: trace_status - may be set to TRUE.
372               sampling     - set to FALSE.
373
374     Author:           Glen George
375     Last Modified:    May 9, 2006
376
377 */
378
379 void  trace_done()
380 {
381     /* variables */
382       /* none */
383
384
385
386     /* done with a trace - if retriggering, ready for another one */
387     if (get_trigger_mode() != ONESHOT_TRIGGER)
388         /* in a retriggering mode - set trace_status to TRUE (ready) */
389     trace_status = TRUE;
390
391     /* no longer sampling data */
392     sampling = FALSE;
393
394
395     /* done so return */
396     return;
397
398 }
399
400
401
402
403 /*
404     trace_rearm
405
406     Description:      This function is called to rearm the trace.  It sets the
407                       trace status to ready (TRUE).  It is used to rearm the
408               trigger in one-shot mode.
409
410     Arguments:        None.
411     Return Value:     None.
412
413     Input:            None.
414     Output:           None.
415
416     Error Handling:   None.
417
418     Algorithms:       None.
419     Data Structures:  None.
420
421     Global Variables: trace_status - set to TRUE.
422
423     Author:           Glen George
424     Last Modified:    Mar. 8, 1994
425
426 */
427
428 void  trace_rearm()
429 {
430     /* variables */
431       /* none */
432
433
434
435     /* rearm the trace - set status to ready (TRUE) */
436     trace_status = TRUE;
437
438
439     /* all done - return */
440     return;
441
```

```
}



/*
   set_trace_size

   Description:      This function sets the locally global sample size to the
                     passed value.  This is used to scale the data when
             plotting a trace.

   Arguments:        size (int) - the trace sample size.
   Return Value:     None.

   Input:            None.
   Output:           None.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: sample_size - set to the passed value.

   Author:           Glen George
   Last Modified:    Mar. 8, 1994

*/

void  set_trace_size(int size)
{
    /* variables */
      /* none */



    /* set the locally global sample size */
    sample_size = size;


    /* all done, return */
    return;

}




/*
   set_display_scale

   Description:      This function sets the displayed scale type to the passed
             argument.  If the scale is turned on, it draws it.  If it
             is turned off (SCALE_NONE), it restores the saved trace
             under the scale.  Scales can be axes with tick marks
                     (SCALE_AXES) or a grid (SCALE_GRID).

   Arguments:        scale (scale_type) - new scale type.
   Return Value:     None.

   Input:            None.
   Output:           Either a scale is output or the trace under the old scale
             is restored.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: cur_scale    - set to the passed value.
             saved_axis_x - used to restore trace data under x-axis.
             saved_axis_y - used to restore trace data under y-axis.

   Author:           Glen George
   Last Modified:    May 9, 2006

*/

void  set_display_scale(enum scale_type scale)
{
```

```
523     /* variables */
524     int  p;             /* x or y coordinate */
525
526     int  i;     /* loop indices */
527     int  j;
528
529
530
531     /* whenever change scale type, need to clear out previous scale */
532     /* unnecessary if going to SCALE_GRID or from SCALE_NONE or not changing the scale */
533     if ((scale != SCALE_GRID) && (cur_scale != SCALE_NONE) && (scale != cur_scale))  {
534
535         /* need to restore the trace under the lines (tick, grid, or axis) */
536
537     /* go through all points on horizontal lines */
538     for (j = -Y_TICK_CNT; j <= Y_TICK_CNT; j++)  {
539
540         /* get y position of the line */
541         p = X_AXIS_POS + j * Y_TICK_SIZE;
542         /* make sure it is in range */
543         if (p >= PLOT_SIZE_Y)
544             p = PLOT_SIZE_Y - 1;
545         if (p < 0)
546             p = 0;
547
548         /* look at entire horizontal line */
549         for (i = 0; i < PLOT_SIZE_X; i++)  {
550             /* check if this point is on or off (need to look at bits) */
551         if ((saved_axis_x[j + Y_TICK_CNT][i / 8] & (0x80 >> (i % 8))) == 0)
552             /* saved pixel is off */
553             plot_pixel(i, p, PIXEL_BGND);
554         else
555             /* saved pixel is on */
556             plot_pixel(i, p, PIXEL_RED);
557         }
558     }
559
560     /* go through all points on vertical lines */
561     for (j = -X_TICK_CNT; j <= X_TICK_CNT; j++)  {
562
563         /* get x position of the line */
564         p = Y_AXIS_POS + j * X_TICK_SIZE;
565         /* make sure it is in range */
566         if (p >= PLOT_SIZE_X)
567             p = PLOT_SIZE_X - 1;
568         if (p < 0)
569             p = 0;
570
571         /* look at entire vertical line */
572         for (i = 0; i < PLOT_SIZE_Y; i++)  {
573             /* check if this point is on or off (need to look at bits) */
574         if ((saved_axis_y[j + X_TICK_CNT][i / 8] & (0x80 >> (i % 8))) == 0)
575             /* saved pixel is off */
576             plot_pixel(p, i, PIXEL_BGND);
577         else
578             /* saved pixel is on */
579             plot_pixel(p, i, PIXEL_RED);
580         }
581     }
582     }
583
584
585     /* now handle the scale type appropriately */
586     switch (scale)  {
587
588         case SCALE_AXES:    /* axes for the scale */
589         case SCALE_GRID:    /* grid for the scale */
590
591                     /* draw x lines (grid or tick marks) */
592                 for (i = -Y_TICK_CNT; i <= Y_TICK_CNT; i++)  {
593
594                 /* get y position of the line */
595                 p = X_AXIS_POS + i * Y_TICK_SIZE;
596                 /* make sure it is in range */
597                 if (p >= PLOT_SIZE_Y)
598                     p = PLOT_SIZE_Y - 1;
599                 if (p < 0)
600                     p = 0;
601
602                 /* should we draw a grid, an axis, or a tick mark */
603                 if (scale == SCALE_GRID)
```

```
                    /* drawing a grid line */
                        plot_hline(X_GRID_START, p, (X_GRID_END - X_GRID_START));
                else if (i == 0)
                    /* drawing the x axis */
                        plot_hline(X_AXIS_START, p, (X_AXIS_END - X_AXIS_START));
                else
                    /* must be drawing a tick mark */
                        plot_hline((Y_AXIS_POS - (TICK_LEN / 2)), p, TICK_LEN);
                }

                    /* draw y lines (grid or tick marks) */
                for (i = -X_TICK_CNT; i <= X_TICK_CNT; i++)  {

                /* get x position of the line */
                p = Y_AXIS_POS + i * X_TICK_SIZE;
                /* make sure it is in range */
                if (p >= PLOT_SIZE_X)
                    p = PLOT_SIZE_X - 1;
                    if (p < 0)
                    p = 0;

                /* should we draw a grid, an axis, or a tick mark */
                if (scale == SCALE_GRID)
                    /* drawing a grid line */
                        plot_vline(p, Y_GRID_START, (Y_GRID_END - Y_GRID_START));
                else if (i == 0)
                    /* drawing the y axis */
                        plot_vline(p, Y_AXIS_START, (Y_AXIS_END - Y_AXIS_START));
                else
                    /* must be drawing a tick mark */
                        plot_vline(p, (X_AXIS_POS - (TICK_LEN / 2)), TICK_LEN);
                }

                /* done with the axes */
                break;

        case SCALE_NONE:    /* there is no scale */
                /* already restored plot so nothing to do */
                break;

    }


    /* now remember the new (now current) scale type */
    cur_scale = scale;


    /* scale is taken care of, return */
    return;
}




/*
   clear_saved_areas

   Description:      This function clears all the saved areas (for saving the
                     trace under the axes, menus, and general areas).

   Arguments:        None.
   Return Value:     None.

   Input:            None.
   Output:           None.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: saved_axis_x - cleared.
             saved_axis_y - cleared.
             saved_menu   - cleared.
             saved_area   - cleared.

   Author:           Glen George
   Last Modified:    May 9, 2006

*/
```

```
685
686 void  clear_saved_areas()
687 {
688     /* variables */
689     int  i;     /* loop indices */
690     int  j;
691
692
693
694     /* clear x-axis and y-axis save areas */
695     for (j = 0; j <= (2 * Y_TICK_CNT); j++)
696         for (i = 0; i < (SIZE_X / 8); i++)
697             saved_axis_x[j][i] = 0;
698     for (j = 0; j <= (2 * X_TICK_CNT); j++)
699         for (i = 0; i < (SIZE_Y / 8); i++)
700             saved_axis_y[j][i] = 0;
701
702     /* clear the menu save ares */
703     for (i = 0; i < MENU_SIZE_Y; i++)
704         for (j = 0; j < ((MENU_SIZE_X + 7) / 8); j++)
705         saved_menu[i][j] = 0;
706
707     /* clear general save area */
708     for (i = 0; i < SAVE_SIZE_Y; i++)
709         for (j = 0; j < (SAVE_SIZE_X / 8); j++)
710         saved_area[i][j] = 0;
711
712
713     /* done clearing the saved areas - return */
714     return;
715
716 }
717
718
719
720
721 /*
722    restore_menu_trace
723
724    Description:      This function restores the trace under the menu when the
725                      menus are turned off.  (The trace was previously saved.)
726
727    Arguments:        None.
728    Return Value:     None.
729
730    Input:            None.
731    Output:           The trace under the menu is restored to the LCD screen.
732
733    Error Handling:   None.
734
735    Algorithms:       None.
736    Data Structures:  None.
737
738    Global Variables: saved_menu - used to restore trace data under the menu.
739
740    Author:           Glen George
741    Last Modified:    Mar. 13, 1994
742
743 */
744
745 void  restore_menu_trace()
746 {
747     /* variables */
748     int  bit_position;  /* position of bit to restore (in saved data) */
749     int  bit_offset;    /* offset (in bytes) of bit within saved row */
750
751     int  x;     /* loop indices */
752     int  y;
753
754
755
756     /* loop, restoring the trace under the menu */
757     for (y = MENU_UL_Y; y < (MENU_UL_Y + MENU_SIZE_Y); y++)  {
758
759         /* starting a row - initialize bit position */
760     bit_position = 0x80;    /* start at high-order bit in the byte */
761     bit_offset = 0;     /* first byte of the row */
762
763         for (x = MENU_UL_X; x < (MENU_UL_X + MENU_SIZE_X); x++)  {
764
765         /* check if this point is on or off (need to look at bits) */
```

```
766         if ((saved_menu[y - MENU_UL_Y][bit_offset] & bit_position) == 0)
767             /* saved pixel is off */
768         plot_pixel(x, y, PIXEL_BGND);
769         else
770             /* saved pixel is on */
771         plot_pixel(x, y, PIXEL_RED);
772 
773         /* move to the next bit position */
774         bit_position >>= 1;
775         /* check if moving to next byte */
776         if (bit_position == 0)  {
777             /* now on high bit of next byte */
778         bit_position = 0x80;
779         bit_offset++;
780         }
781         }
782     }
783 
784 
785     /* restored menu area - return */
786     return;
787 
788 }
789 
790 
791 
792 
793 /*
794    set_save_area
795 
796    Description:      This function sets the position and size of the area to
797                      be saved when traces are drawn.  It also clears any data
798              currently saved.
799 
800    Arguments:        pos_x (int)  - x position of upper left corner of the
801                     saved area.
802              pos_y (int)  - y position of upper left corner of the
803                         saved area.
804              size_x (int) - horizontal size of the saved area.
805              size_y (int) - vertical size of the saved area.
806    Return Value:     None.
807 
808    Input:            None.
809    Output:           None.
810 
811    Error Handling:   None.
812 
813    Algorithms:       None.
814    Data Structures:  None.
815 
816    Global Variables: saved_area  - cleared.
817              saved_pos_x - set to passed value.
818              saved_pos_y - set to passed value.
819              saved_end_x - computed from passed values.
820              saved_end_y - computed from passed values.
821 
822    Author:           Glen George
823    Last Modified:    Mar. 8, 1994
824 
825 */
826 
827 void  set_save_area(int pos_x, int pos_y, int size_x, int size_y)
828 {
829     /* variables */
830     int  x;     /* loop indices */
831     int  y;
832 
833 
834 
835     /* just setup all the locally global variables from the passed values */
836     saved_pos_x = pos_x;
837     saved_pos_y = pos_y;
838     saved_end_x = pos_x + size_x;
839     saved_end_y = pos_y + size_y;
840 
841 
842     /* clear the save area */
843     for (y = 0; y < SAVE_SIZE_Y; y++)  {
844         for (x = 0; x < (SAVE_SIZE_X / 8); x++)  {
845         saved_area[y][x] = 0;
846         }
```

```
847     }
848
849
850     /* setup the saved area - return */
851     return;
852
853 }
854
855
856
857
858 /*
859    restore_trace
860
861    Description:      This function restores the trace under the set saved
862                      area.  (The area was previously set and the trace was
863              previously saved.)
864
865    Arguments:        None.
866    Return Value:     None.
867
868    Input:            None.
869    Output:           The trace under the saved ares is restored to the LCD.
870
871    Error Handling:   None.
872
873    Algorithms:       None.
874    Data Structures:  None.
875
876    Global Variables: saved_area  - used to restore trace data.
877              saved_pos_x - gives starting x position of saved area.
878              saved_pos_y - gives starting y position of saved area.
879              saved_end_x - gives ending x position of saved area.
880              saved_end_y - gives ending y position of saved area.
881
882    Author:           Glen George
883    Last Modified:    Mar. 13, 1994
884
885 */
886
887 void  restore_trace()
888 {
889     /* variables */
890     int  bit_position;  /* position of bit to restore (in saved data) */
891     int  bit_offset;    /* offset (in bytes) of bit within saved row */
892
893     int  x;     /* loop indices */
894     int  y;
895
896
897
898     /* loop, restoring the saved trace */
899     for (y = saved_pos_y; y < saved_end_y; y++)  {
900
901         /* starting a row - initialize bit position */
902     bit_position = 0x80;    /* start at high-order bit in the byte */
903     bit_offset = 0;     /* first byte of the row */
904
905         for (x = saved_pos_x; x < saved_end_x; x++)  {
906
907         /* check if this point is on or off (need to look at bits) */
908         if ((saved_area[y - saved_pos_y][bit_offset] & bit_position) == 0)
909             /* saved pixel is off */
910         plot_pixel(x, y, PIXEL_BGND);
911         else
912             /* saved pixel is on */
913         plot_pixel(x, y, PIXEL_RED);
914
915         /* move to the next bit position */
916         bit_position >>= 1;
917         /* check if moving to next byte */
918         if (bit_position == 0)  {
919             /* now on high bit of next byte */
920         bit_position = 0x80;
921         bit_offset++;
922         }
923         }
924     }
925
926
927     /* restored the saved area - return */
```

```
928     return;
929
930 }
931
932
933
934
935 /*
936    do_trace
937
938    Description:      This function starts a trace.  It starts the hardware
939                      sampling data (via a function call) and sets the trace
940              ready flag (trace_status) to FALSE and the sampling flag
941              (sampling) to TRUE.
942
943    Arguments:        None.
944    Return Value:     None.
945
946    Input:            None.
947    Output:           None.
948
949    Error Handling:   None.
950
951    Algorithms:       None.
952    Data Structures:  None.
953
954    Global Variables: trace_status - set to FALSE (not ready for another trace).
955              sampling     - set to TRUE (doing a sample now).
956
957    Author:           Glen George
958    Last Modified:    Mar. 13, 1994
959
960 */
961
962 void  do_trace()
963 {
964     /* variables */
965       /* none */
966
967
968
969     /* start up the trace */
970     /* indicate whether using automatic triggering or not */
971     start_sample(get_trigger_mode() == AUTO_TRIGGER);
972
973     /* now not ready for another trace (currently doing one) */
974     trace_status = FALSE;
975
976     /* and are currently sampling data */
977     sampling = TRUE;
978
979
980     /* trace is going, return */
981     return;
982
983 }
984
985
986
987
988 /*
989    plot_trace
990
991    Description:      This function plots the passed traces.  The traces are
992                      assumed to contain sample_size points of sampled data.
993              Any points falling within any of the save areas are also
994              saved by this routine.  The data is also scaled to be
995              within the range of the entire screen. Function assumes 24-bits of
996                  trace data per sample (8 bits for each of the two analog inputs
997                  the logic analyzer).
998
999    Arguments:        sample (unsigned char **) - sample to plot.
1000   Return Value:     None.
1001
1002   Input:            None.
1003   Output:           The sample is plotted on the screen.
1004
1005   Error Handling:   None.
1006
1007   Algorithms:       If there are more sample points than screen width the
1008             sample is plotted with multiple points per horizontal
```

```
1009            position.
1010    Data Structures:  None.
1011
1012    Global Variables: cur_scale    - determines type of scale to plot.
1013              sample_size  - determines size of passed sample.
1014              saved_axis_x - stores trace under x-axis.
1015              saved_axis_y - stores trace under y-axis.
1016              saved_menu   - stores trace under the menu.
1017              saved_area   - stores trace under the saved area.
1018              saved_pos_x  - determines location of saved area.
1019              saved_pos_y  - determines location of saved area.
1020              saved_end_x  - determines location of saved area.
1021              saved_end_y  - determines location of saved area.
1022
1023    Author:           Glen George
1024    Last Modified:    May 9, 2006
1025
1026 */
1027
1028 void  plot_trace(unsigned char **sample)
1029 {
1030     /* variables */
1031     int x = 0;                          /* current x position to plot */
1032     int x_pos = (PLOT_SIZE_X / 2);  /* "fine" x position for multiple point plotting */
1033
1034     int yA;                             /* y position of point to plot */
1035     int yB;                             /* y position of point to plot */
1036
1037     int p;                              /* an x or y coordinate */
1038
1039     int i;                              /* loop indices */
1040     int j;
1041     int ii;
1042     int jj;
1043
1044     unsigned char *sample_A = sample[0];
1045     unsigned char *sample_B = sample[1];
1046     unsigned char *sample_L = sample[2];
1047
1048     // Iterate through the points.
1049     for (i = 0; i < sample_size; i++) {
1050         // Get the new screen coordinates.
1051         trace_A[i] = 255 - sample_A[i] + 8;
1052         trace_B[i] = 255 - sample_B[i] + 8;
1053         trace_L[i] = sample_L[i];
1054
1055         // Clear the analog trace.
1056         if (i > 0) {
1057             for (j = min(saved_trace_A[i-1], saved_trace_A[i]); j <= max(saved_trace_A[i-1], ↩
                    saved_trace_A[i]); j++) {
1058                 plot_pixel(i, j, PIXEL_BGND);
1059             }
1060             for (j = min(saved_trace_B[i-1], saved_trace_B[i]); j <= max(saved_trace_B[i-1], ↩
                    saved_trace_B[i]); j++) {
1061                 plot_pixel(i, j, PIXEL_BGND);
1062             }
1063         }
1064
1065         // Clear the logic analyzer trace.
1066         unsigned char cur_log = saved_trace_L[i];
1067         for (j = 0; j < 8; j++) {
1068             if (cur_log & 1) {
1069                 plot_pixel(i, 270 - 5 * j - 3, PIXEL_BGND);
1070             } else {
1071                 plot_pixel(i, 270 - 5 * j, PIXEL_BGND);
1072             }
1073             cur_log = cur_log >> 1;
1074         }
1075
1076         //Draw the analog trace.
1077         if (i > 1) {
1078             for (j = min(trace_A[i-1], trace_A[i]); j <= max(trace_A[i-1], trace_A[i]); j++) {
1079                 plot_pixel(i, j, PIXEL_A);
1080             }
1081             for (j = min(trace_B[i-1], trace_B[i]); j <= max(trace_B[i-1], trace_B[i]); j++) {
1082                 plot_pixel(i, j, PIXEL_B);
1083             }
1084         }
1085
1086         // Draw the logic analyzer trace.
1087         cur_log = trace_L[i];
```

```
1088            for (j = 0; j < 8; j++) {
1089                if (cur_log & 1) {
1090                    if (j % 2) plot_pixel(i, 270 - 5 * j - 3, PIXEL_L1);
1091                    else plot_pixel(i, 270 - 5 * j - 3, PIXEL_L2);
1092                } else {
1093                    if (j % 2) plot_pixel(i, 270 - 5 * j, PIXEL_L1);
1094                    else plot_pixel(i, 270 - 5 * j, PIXEL_L2);
1095                }
1096                plot_pixel(i, 270 - 5 * j - 1, PIXEL_L3);
1097                plot_pixel(i, 270 - 5 * j - 2, PIXEL_L3);
1098                cur_log = cur_log >> 1;
1099            }
1100
1101        }
1102
1103        // Update the saved trace arrays.
1104        for (i = 0; i < sample_size; i++) {
1105            saved_trace_A[i] = trace_A[i];
1106            saved_trace_B[i] = trace_B[i];
1107            saved_trace_L[i] = trace_L[i];
1108        }
1109
1110        // Output the scale if need be.
1111        set_display_scale(cur_scale);
1112
1113        // Replace menu if need be.
1114        refresh_menu();
1115
1116        /* done with plot, return */
1117        return;
1118
1119    }
```

H.6.2 sampleint

```
1  ##############################################################################
2  #                                                                            #
3  #                                sampleint.S                                 #
4  #                      Handles new samples and triggering                    #
5  #                                  EE/CS 52                                  #
6  #                                                                            #
7  ##############################################################################
8
9
10 /*
11  *  Albert Gural
12  *  EE/CS 52
13  *  TA: Dan Pipe-Mazo
14  *
15  *  File Description:   This file contains functions for handling
16  *  interrupts generated by new triggered samples, and provides
17  *  the accessor functions for the main C code to access the sampled
18  *  data. Samples are stored in a char** buffer first indexed by which
19  *  input (A, B, logic analyzer), then indexed by the samples.
20  *
21  *  Revision History:
22  *      02/09/2012  Dan Pipe-Mazo   Initial Revision.
23  *      05/14/2014  Albert Gural    Began writing adc code assembly.
24  *      06/02/2014  Albert Gural    Added accessor functions for C code interface.
25  *
26  */
27
28  /*  Local Include Files   */
29 #include "macros.m"
30 #include "pio.m"
31 #include "../osc_bsp/system.h"
32
33 .section  .text         #start code section
34
35
36 /*
37  *  adc_int_installer
38  *
39  *  Description: Installs the interrupt handler for trigger interrupts.
40  *
41  *  Arguments: (none)
```

```
 *
 *  Return Value: (none)
 *
 */

.global adc_int_installer
.type   adc_int_installer, @function

adc_int_installer:
    SAVE

    # Install the interrupt handler
    mov     r4, r0
    movi    r5, 6
    MOVWI   r6, adc_int_handler
    mov     r7, r0
    PUSH    r0
    call    alt_ic_isr_register
    POP     r0

    # Clear the edge capture register (write 1 to clear).
    MOVWI   r8, TRIG_INT_BASE
    MOVWI   r9, 0xFFFFFFFF
    stw     r9, PIO_EDGE_CAP(r8)

adc_int_installer_done:
    RESTORE
    ret


/*
 *  adc_int_handler
 *
 *  Description: Handles a trigger interrupt. When a trigger occurs, we need
 *  to move all the data from the FIFO onto a local buffer (which can then be
 *  used by the C code in higher level functions).
 *
 *  Arguments: (none)
 *
 *  Return Value: (none)
 *
 */

.global adc_int_handler
.type adc_int_handler, @function

adc_int_handler:
    SAVE

    # Clear interrupts.
    MOVWI   r8, TRIG_INT_BASE
    stw     r0, PIO_IRQ_MASK(r8)

    # Get the edge capture register.
    ldw     r9, PIO_EDGE_CAP(r8)

    # Pause FIFO write.
    STWI    ADC_CTRL_BASE, 0x00

    # Loop variable to clear front of fifo
    mov     r10, r0

    # Set specially-designed delay offsets based on which fifo source clock is being used.
    movia   r8, fifo_clk_src
    ldb     r9, (r8)
    beq     r9, r0, adc_int_handler_fast_clk_offset

adc_int_handler_1M_clk_offset:
    movi    r13, 15
    br      adc_int_handler_clear_front

adc_int_handler_fast_clk_offset:
    movi    r13, 22

adc_int_handler_clear_front:
    # Bitbang clock pulse.
    STWI    ADC_CTRL_BASE, 0x01
    STWI    ADC_CTRL_BASE, 0x00

    # Keep going until 480 good remaining points (for display)
    addi    r10, r10, 1
```

```
123     bltu    r10, r13, adc_int_handler_clear_front
124
125     movia   r8, sample_buffer_A
126     mov     r10, r0
127
128 adc_int_handler_loop:
129     # Bitbang clock pulse.
130     STWI    ADC_CTRL_BASE, 0x01
131     STWI    ADC_CTRL_BASE, 0x00
132
133     # Get ch. A, ch. B, and logic data.
134     MOVWI   r11, ADC_RAW_BASE
135     ldwio   r12, (r11)
136
137     # r12 for ch. A, r13 for ch. B, r14 for logic.
138     srli    r13, r12, 8
139     srli    r14, r13, 8
140     movui   r15, 128
141
142     # Convert analog channels to non-signed values.
143     add     r12, r12, r15
144     add     r13, r13, r15
145
146     # Keep only the bottom byte.
147     andi    r12, r12, 0xFF
148     andi    r13, r13, 0xFF
149     andi    r14, r14, 0xFF
150
151     # CH. A
152     # Retrieve the current buffer contents.
153     # Then update the current buffer with the new value.
154     movia   r8, sample_buffer_A
155     add     r8, r8, r10
156     stb     r12, (r8)
157
158     # CH. B
159     # Retrieve the current buffer contents.
160     # Then update the current buffer with the new value.
161     movia   r8, sample_buffer_B
162     add     r8, r8, r10
163     stb     r13, (r8)
164
165     # LOGIC ANALYZER
166     # Retrieve the current buffer contents.
167     # Then update the current buffer with the new value.
168     movia   r8, sample_buffer_L
169     add     r8, r8, r10
170     stb     r14, (r8)
171
172     addi    r10, r10, 1
173     movi    r15, 480
174     bltu    r10, r15, adc_int_handler_loop
175
176     # Sample done.
177     movia   r8, sample_complete
178     movi    r9, 1
179     stb     r9, (r8)
180
181 adc_int_handler_done:
182
183     RESTORE
184     ret
185
186
187
188 /*
189  *  clear_display
190  *
191  *  Description: Clears the display.
192  *
193  *  Arguments: (none)
194  *
195  *  Return Value: (none)
196  *
197  */
198
199 .global clear_display
200 .type clear_display, @function
201
202 clear_display:
203     SAVE
```

```
204
205     # Get display address and background color.
206     MOVWI   r8, VRAM_CTRL_BASE
207     movui   r9, 272
208     slli    r9, r9, 10
209     add     r9, r8, r9
210     movui   r15, 0x001C
211
212     # Loop over all screen pixels, clearing them.
213 clear_display_loop:
214     sthio   r15, (r8)
215     addi    r8, r8, 2
216     bltu    r8, r9, clear_display_loop
217
218 clear_display_done:
219     RESTORE
220     ret
221
222
223
224 /*
225  *  plot_pixel
226  *
227  *  Description: Plots a pixel of the specified color at the specified location.
228  *
229  *  Arguments:
230  *  r4 - The x coordinate (from left)
231  *  r5 - The y coordinate (from top)
232  *  r6 - The color
233  *
234  *  Return Value: (none)
235  *
236  */
237
238 .global plot_pixel
239 .type plot_pixel, @function
240
241 plot_pixel:
242     SAVE
243
244     push    r4
245     push    r5
246
247     # Each row takes 1024 bytes, so shift row var by 10 bits.
248     slli    r5, r5, 10
249     # Add twice to account for 16-bit VRAM storage.
250     add     r5, r5, r4
251     add     r5, r5, r4
252     # Now get the absolute address.
253     MOVWI   r4, VRAM_CTRL_BASE
254     add     r5, r5, r4
255     # Store the color.
256     sth     r6, (r5)
257
258     pop     r5
259     pop     r4
260
261 plot_pixel_done:
262     RESTORE
263     ret
264
265
266 /*
267  *  set_sample_rate
268  *
269  *  Description: Given the desired sample frequency, computes what compare value
270  *  value would be needed on the FIFO sample clock to get that frequency. The
271  *  computed value is sent to the control register.
272  *
273  *  Arguments:
274  *  r4 - The desired frequency (Hz)
275  *
276  *  Return Value: (none)
277  *
278  */
279
280 .global set_sample_rate
281 .type set_sample_rate, @function
282
283 set_sample_rate:
284     SAVE
```

```
285
286     MOVWI   r9, 100000
287     bleu    r4, r9, slow_sample_rate
288
289 fast_sample_rate:
290     # Divide fastest sample rate by desired sample rate
291     # to get number of ticks to pause (not collect samples)
292     # between collecting samples.
293     MOVWI   r8, ADC_RATE_BASE
294     MOVWI   r9, 400000000
295     divu    r9, r9, r4
296     subi    r9, r9, 1
297     stw     r9, (r8)
298
299     # Set fast clock for FIFO clock counter.
300     MOVWI   r8, TRIG_CTRL_BASE
301     MOVWI   r9, 0x00000080
302     stw     r9, PIO_OUTCLR(r8)
303
304     # Update fifo source flag.
305     movia   r8, fifo_clk_src
306     stb     r0, (r8)
307
308     br      set_sample_rate_done
309
310 slow_sample_rate:
311     # Divide fastest sample rate by desired sample rate
312     # to get number of ticks to pause (not collect samples)
313     # between collecting samples.
314     MOVWI   r8, ADC_RATE_BASE
315     MOVWI   r9, 1000000
316     divu    r9, r9, r4
317     subi    r9, r9, 1
318     stw     r9, (r8)
319
320     # Set 1MHz clock for FIFO clock counter.
321     MOVWI   r8, TRIG_CTRL_BASE
322     MOVWI   r9, 0x00000080
323     stw     r9, PIO_OUTSET(r8)
324
325     # Update fifo source flag.
326     movia   r8, fifo_clk_src
327     movi    r9, 1
328     stb     r9, (r8)
329
330 set_sample_rate_done:
331     # Always return 480 samples.
332     movui   r2, 480
333
334     RESTORE
335     ret
336
337
338 /*
339  *  set_trigger
340  *
341  *  Description: Sets the trigger level (7 bit precision) and slope.
342  *
343  *  Arguments:
344  *  r4 - Trigger level
345  *  r5 - Slope
346  *
347  *  Return Value: (none)
348  *
349  */
350
351 .global set_trigger
352 .type set_trigger, @function
353
354 set_trigger:
355     SAVE
356
357     # Convert [0 to 127] to signed 8-bit [-127 to 127].
358     # Then update trigger level.
359     MOVWI   r8, TRIG_LEVEL_BASE
360     slli    r9, r4, 1
361     subi    r9, r9, 127
362     andi    r9, r9, 0xFF
363     stw     r9, (r8)
364
365     # Slope is second bit of TRIG_CTRL. Modify the given
```

```
366     # argument, then update slope.
367     beq     r5, r0, set_trigger_slope_pos
368     movi    r8, 0x02
369     MOVWI   r9, TRIG_CTRL_BASE
370     stw     r8, PIO_OUTSET(r9)
371     br      set_trigger_done
372
373 set_trigger_slope_pos:
374     movi    r8, 0x02
375     MOVWI   r9, TRIG_CTRL_BASE
376     stw     r8, PIO_OUTCLR(r9)
377
378 set_trigger_done:
379     RESTORE
380     ret
381
382
383 /*
384  *  set_delay
385  *
386  *  Description: Sets the delay (positive or negative)
387  *
388  *  Arguments:
389  *  r4 - Delay in samples
390  *
391  *  Return Value: (none)
392  *
393  */
394
395 .global set_delay
396 .type set_delay, @function
397
398 set_delay:
399     SAVE
400
401     # Simply set the delay (240 offset means delay 0 is in the middle).
402     addi    r9, r4, 240
403     MOVWI   r8, TRIG_DELAY_BASE
404     stw     r9, (r8)
405
406 set_delay_done:
407     RESTORE
408     ret
409
410
411 /*
412  *  start_sample
413  *
414  *  Description: Prepares the FPGA logic for finding the next trigger event.
415  *
416  *  Arguments:
417  *  r4 - Type of trigger (0 for normal, 1 for auto-trigger)
418  *
419  *  Return Value: (none)
420  *
421  */
422
423 .global start_sample
424 .type start_sample, @function
425
426 start_sample:
427     SAVE
428
429     # Clear the edge capture register (write 1 to clear).
430     MOVWI   r8, TRIG_INT_BASE
431     MOVWI   r9, 0xFFFFFFFF
432     stw     r9, PIO_EDGE_CAP(r8)
433
434     # Enable trigger interrupts.
435     MOVWI   r9, 0x00000002
436     # If auto-trigger, enable time-outs as well.
437     slli    r8, r4, 2
438     add     r9, r8, r9
439
440     MOVWI   r8, TRIG_INT_BASE
441     stw     r9, PIO_IRQ_MASK(r8)
442
443     # Clear FIFO, turn on FIFO write.
444     STWI    ADC_CTRL_BASE, 0x04
445     STWI    ADC_CTRL_BASE, 0x02
446
```

```
    # Restart trigger counter [ch. A], [+ slope]
    MOVWI   r8, TRIG_CTRL_BASE
    movi    r9, 1
    stw     r9, PIO_OUTSET(r8)
    stw     r9, PIO_OUTCLR(r8)

start_sample_done:
    RESTORE
    ret


/*
 *  sample_done
 *
 *  Description: Returns a pointer to the sample buffer, which is an array
 *  of an array of samples. If the sample is not done, returns null.
 *
 *  Arguments: (none)
 *
 *  Return Value: Pointer to the buffer, or null if the sample is not finished
 *
 */

.global sample_done
.type sample_done, @function

sample_done:
    SAVE

    # Test if sample occurred.
    movia   r8, sample_complete
    ldb     r9, (r8)
    beq     r9, r0, sample_null

    # If so, reset sample complete variable and return map to buffers.
    stb     r0, (r8)
    movia   r2, sample_map
    br      sample_done_done

sample_null:
    # Otherwise, return null.
    mov     r2, r0

sample_done_done:
    RESTORE
    ret


/*
 * Contains the three sample buffers. In this way, sample_map is an array of an array of samples,
 * or a char**. The first index [0..2] gives which input we want, while the second index [0..479]
 * gives which sample we want.
 *
 */

sample_map:
    .word   sample_buffer_A
    .word   sample_buffer_B
    .word   sample_buffer_L


.section  .data     #start data section

sample_complete:    .word   0       # 0 = still trying to get sample; 1 = sample done.
fifo_clk_src:       .word   0       # 0 = fast_clk, 1 = 1M_clk. This is useful for fine-tuning the trigger delay.
sample_buffer_A:    .skip   480     # Buffer stores display-worth of ADC samples (ch. A).
sample_buffer_B:    .skip   480     # Buffer stores display-worth of ADC samples (ch. B).
sample_buffer_L:    .skip   480     # Buffer stores display-worth of ADC samples (Logic).
```

Appendix I

Test Code

I.1 Display Timing with ModelSim

```
1  -- Copyright (C) 1991-2013 Altera Corporation
2  -- Your use of Altera Corporation's design tools, logic functions
3  -- and other software and tools, and its AMPP partner logic
4  -- functions, and any output files from any of the foregoing
5  -- (including device programming or simulation files), and any
6  -- associated documentation or information are expressly subject
7  -- to the terms and conditions of the Altera Program License
8  -- Subscription Agreement, Altera MegaCore Function License
9  -- Agreement, or other applicable license agreement, including,
10 -- without limitation, that your use is for the sole purpose of
11 -- programming logic devices manufactured by Altera and sold by
12 -- Altera or its authorized distributors.  Please refer to the
13 -- applicable agreement for further details.
14
15 -- ***************************************************************************
16 -- This file contains a Vhdl test bench template that is freely editable to
17 -- suit user's needs .Comments are provided in each section to help the user
18 -- fill out necessary details.
19 -- ***************************************************************************
20 -- Generated on "03/04/2014 13:51:33"
21
22 -- Vhdl Test Bench template for design  :  oscilloscope
23 --
24 -- Simulation tool : ModelSim-Altera (VHDL)
25 --
26
27 LIBRARY ieee;
28 USE ieee.std_logic_1164.all;
29 USE ieee.numeric_std.all;
30 USE ieee.std_logic_misc.all;
31 USE ieee.math_real.all;
32
33 ENTITY oscilloscope_vhd_tst IS
34 END oscilloscope_vhd_tst;
35
36 ARCHITECTURE oscilloscope_arch OF oscilloscope_vhd_tst IS
37     -- constants
38     -- signals
39     SIGNAL CLK_IN    : STD_LOGIC;
40     SIGNAL RESET_IN  : STD_LOGIC;
41     SIGNAL CS_IN     : STD_LOGIC;
42     SIGNAL RW_IN     : STD_LOGIC;
43     SIGNAL HSYNC_IN  : STD_LOGIC;
44     SIGNAL SRT_IN    : STD_LOGIC;
45     SIGNAL ADDR      : STD_LOGIC_VECTOR(8 DOWNTO 0);
46     SIGNAL RAS       : STD_LOGIC;
47     SIGNAL CAS       : STD_LOGIC;
48     SIGNAL TRG       : STD_LOGIC;
49     SIGNAL WE        : STD_LOGIC;
50     SIGNAL BUSY      : STD_LOGIC;
51
52     COMPONENT oscilloscope
53         PORT (
```

```
54              CLK_IN      : IN  STD_LOGIC;
55              RESET_IN    : IN  STD_LOGIC;
56              CS_IN       : IN  STD_LOGIC;
57              RW_IN       : IN  STD_LOGIC;
58              HSYNC_IN    : IN  STD_LOGIC;
59              SRT_IN      : IN  STD_LOGIC;
60              ADDR        : OUT STD_LOGIC_VECTOR(8 DOWNTO 0);
61              RAS         : OUT STD_LOGIC;
62              CAS         : OUT STD_LOGIC;
63              TRG         : OUT STD_LOGIC;
64              WE          : OUT STD_LOGIC;
65              BUSY        : OUT STD_LOGIC
66          );
67      END COMPONENT;
68
69      BEGIN
70          i1 : oscilloscope
71          PORT MAP (
72              -- list connections between master ports and signals
73              CLK_IN      => CLK_IN,
74              RESET_IN    => RESET_IN,
75              CS_IN       => CS_IN,
76              RW_IN       => RW_IN,
77              HSYNC_IN    => HSYNC_IN,
78              SRT_IN      => SRT_IN,
79              ADDR        => ADDR,
80              RAS         => RAS,
81              CAS         => CAS,
82              TRG         => TRG,
83              WE          => WE,
84              BUSY        => BUSY
85          );
86
87
88      clock : PROCESS
89          -- variable declarations
90          variable go : integer := 1;
91          --constant syshalfclk : integer := 50;
92          BEGIN
93              if (go = 1) then
94                  CLK_IN <= '1';
95                  wait for 50 ps;
96                  CLK_IN <= '0';
97                  wait for 50 ps;
98              end if;
99      END PROCESS clock;
100
101
102     init : PROCESS
103         -- variable declarations
104         BEGIN
105                 -- code that executes only once
106                 RESET_IN <= '1';
107                 wait for 100 ps;
108                 RESET_IN <= '0';
109         WAIT;
110     END PROCESS init;
111
112
113     always : PROCESS
114         -- optional sensitivity list
115         -- (        )
116         -- variable declarations
117
118         BEGIN
119             -- code executes for every event on sensitivity list
120             CS_IN <= '1';
121             RW_IN <= '0';
122             HSYNC_IN <= '0';
123             SRT_IN <= '0';
124             wait for 1 ps;
125
126             -- Standard read cycle
127             CS_IN <= '0';
128             RW_IN <= '1';
129             assert(RAS = '1' and CAS = '1' and TRG = '1' and BUSY = '1')
130                 report "READ0: Output signal to VRAM different from expected.";
131
132             wait for 200 ps;
133             assert(RAS = '0' and CAS = '1' and TRG = '1' and WE = '1' and BUSY = '1')
134                 report "READ1: Output signal to VRAM different from expected.";
```

```
135             assert(ADDR = "101010101") report "READ1: Address different from expected.";
136
137             wait for 100 ps;
138             assert(RAS = '0' and CAS = '0' and TRG = '0' and WE = '1' and BUSY = '1')
139                 report "READ2: Output signal to VRAM different from expected.";
140             assert(ADDR = "001100110") report "READ2: Address different from expected.";
141
142             wait for 100 ps;
143             assert(RAS = '0' and CAS = '0' and TRG = '0' and WE = '1' and BUSY = '1')
144                 report "READ3: Output signal to VRAM different from expected.";
145
146             wait for 100 ps;
147             assert(RAS = '0' and CAS = '0' and TRG = '0' and WE = '1' and BUSY = '0')
148                 report "READ4: Output signal to VRAM different from expected.";
149
150             wait for 100 ps;
151             assert(RAS = '1' and CAS = '1' and TRG = '1' and WE = '1' and BUSY = '1')
152                 report "READ5: Output signal to VRAM different from expected.";
153
154             wait for 100 ps;
155             assert(RAS = '1' and CAS = '1' and TRG = '1' and WE = '1' and BUSY = '1')
156                 report "READ6: Output signal to VRAM different from expected.";
157
158             wait for 100 ps;
159             assert(RAS = '1' and CAS = '1' and TRG = '1' and WE = '1' and BUSY = '1')
160                 report "READ7: Output signal to VRAM different from expected.";
161
162             wait for 100 ps;
163
164             -- Standard write cycle
165             CS_IN <= '0';
166             RW_IN <= '0';
167             assert(RAS = '1' and CAS = '1' and TRG = '1' and BUSY = '1') report "WRITE0: Output ↩
                    signal to VRAM different from expected.";
168
169             wait for 100 ps;
170             assert(RAS = '0' and CAS = '1' and TRG = '1' and WE = '1' and BUSY = '0')
171                 report "WRITE1: Output signal to VRAM different from expected.";
172             assert(ADDR = "101010101") report "WRITE1: Address different from expected.";
173
174             wait for 100 ps;
175             assert(RAS = '0' and CAS = '0' and TRG = '0' and WE = '0' and BUSY = '1')
176                 report "WRITE2: Output signal to VRAM different from expected.";
177             assert(ADDR = "001100110") report "WRITE2: Address different from expected.";
178
179             wait for 100 ps;
180             assert(RAS = '0' and CAS = '0' and TRG = '0' and WE = '0' and BUSY = '1')
181                 report "WRITE3: Output signal to VRAM different from expected.";
182
183             wait for 100 ps;
184             assert(RAS = '0' and CAS = '0' and TRG = '0' and WE = '0' and BUSY = '1')
185                 report "WRITE4: Output signal to VRAM different from expected.";
186
187             wait for 100 ps;
188             assert(RAS = '1' and CAS = '1' and TRG = '1' and WE = '0' and BUSY = '1')
189                 report "WRITE5: Output signal to VRAM different from expected.";
190
191             wait for 100 ps;
192             assert(RAS = '1' and CAS = '1' and TRG = '1' and WE = '1' and BUSY = '1')
193                 report "WRITE6: Output signal to VRAM different from expected.";
194
195             wait for 100 ps;
196             assert(RAS = '1' and CAS = '1' and TRG = '1' and WE = '1' and BUSY = '1')
197                 report "WRITE7: Output signal to VRAM different from expected.";
198
199             -- Standard serial transfer cycle
200             CS_IN <= '1';
201             RW_IN <= '1';
202             SRT_IN <= '1';
203
204         WAIT;
205     END PROCESS always;
206
207 END oscilloscope_arch;
```

I.2 VRAM Testing

```
################################################################################
#                                                                              #
#                                VRAM Test Code                                #
#                              Test code for VRAM                              #
#                                  EE/CS 52                                    #
#                                                                              #
################################################################################


/*
 *  Albert Gural
 *  EE/CS 52
 *  TA: Dan Pipe-Mazo
 *
 *  File Description:   Tests VRAM (minimally)
 *
 *  Revision History:
 *      02/09/2012  Dan Pipe-Mazo   Initial Revision.
 *      05/14/2014  Albert Gural    Begain writing testcode assembly.
 *
 */

 /*  Local Include Files   */
#include "macros.m"
#include "../osc_bsp/system.h"

.section  .text         #start code section

/*
 *  test_vram
 *
 *  Description: Tests the VRAM in a couple different scenarios:
 *   - sequentially write/reading each value
 *   - sequentially writing all values then immediately reading all values
 *   - writing all values, waiting, then reading all values
 *
 *  Arguments: (none)
 *
 *  Return Value: (none)
 *
 */

.global test_vram
.type test_vram,@function

test_vram:
    SAVE

    movhi   r8, %hi(VRAM_CTRL_BASE)
    ori     r8, r8, %lo(VRAM_CTRL_BASE)
    movhi   r9, %hi(VRAM_CTRL_SPAN)
    ori     r9, r9, %lo(VRAM_CTRL_SPAN)
    add     r9, r8, r9
    movui   r12, 0x7000 # Test value to write to VRAM

    # Quick W/R test
    call    write_read_all

    # Slower W/R test
    call    write_all
    call    read_all

    # Place a breakpoint here and wait as long as necessary (1 minute or more).
    nop
    call    read_all # Verify there is no corrputed data.

# Failed tests
test_vram_fail:
    nop
    br      test_vram_fail

# Succeeded tests
test_vram_done:
    RESTORE
    ret

```

```
77
78
79  write_all:
80      mov     r10, r8
81      mov     r11, r0
82
83  write_all_loop:
84      # Write to each element sequentially
85      sthio   r11, (r10)
86      addi    r10, r10, 2
87      addi    r11, r11, 1
88      bgeu    r10, r9, write_all_done
89      bgeu    r11, r12, write_all_reset_cnt
90      br      write_all_loop
91
92  write_all_reset_cnt:
93      mov     r11, r0
94      br      write_all_loop
95
96  write_all_done:
97      ret
98
99
100
101 read_all:
102     mov     r10, r8
103     mov     r11, r0
104
105 read_all_loop:
106     # Read and verify each element sequentially
107     ldhio   r13, (r10)
108     bne     r13, r11, test_vram_fail
109     addi    r10, r10, 2
110     addi    r11, r11, 1
111     bgeu    r10, r9, read_all_done
112     bgeu    r11, r12, read_all_reset_cnt
113     br      read_all_loop
114
115 read_all_reset_cnt:
116     mov     r11, r0
117     br      write_all_loop
118
119 read_all_done:
120     ret
```

I.3 Display Testing

```
################################################################################
#                                                                              #
#                                 Display Test Code                            #
#                              Test code for display                           #
#                                    EE/CS 52                                  #
#                                                                              #
################################################################################


/*
 *  Albert Gural
 *  EE/CS 52
 *  TA: Dan Pipe-Mazo
 *
 *  File Description:   Tests the display with a simple 4-square pattern and border
 *
 *  Revision History:
 *      02/09/2012  Dan Pipe-Mazo   Initial Revision.
 *      05/14/2014  Albert Gural    Begain writing testcode assembly.
 *
 */

 /*  Local Include Files   */
#include "../osc_bsp/system.h"

.section  .text         #start code section

/*
 *  test_disp
 *
 *  Description: Draws four rectangles in a 2x2 configuration, then draws a border.
 *  This tests for correct color being displayed, as well as no display timing issues.
 *
 *  This is a quick and dirty implementation, not meant to be well-commented.
 *
 *  Arguments: (none)
 *
 *  Return Value: (none)
 *
 */

.global test_disp
.type test_disp,@function

test_disp:
    movhi   r8, %hi(VRAM_CTRL_BASE)
    ori     r8, r8, %lo(VRAM_CTRL_BASE)
    mov     r11, r0
    mov     r12, r0

disp:
    movi    r13, 136
    bltu    r12, r13, draw_top
    br      draw_bot
disp_cont:
    addi    r8, r8, 2
    addi    r11, r11, 1
    movi    r13, 272
    bgeu    r12, r13, draw_border
    movi    r13, 0x0200
    bgeu    r11, r13, disp_reset
    br      disp

draw_top:
    movi    r13, 240
    bltu    r11, r13, draw_red
    br      draw_yellow

draw_bot:
    movi    r13, 240
    bltu    r11, r13, draw_green
    br      draw_blue

draw_red:
    movi    r9, 0x1F00
    sthio   r9, (r8)
```

```
77      br      disp_cont
78
79  draw_yellow:
80      movui   r9, 0xFF03
81      sthio   r9, (r8)
82      br      disp_cont
83
84  draw_green:
85      movui   r9, 0xE003
86      sthio   r9, (r8)
87      br      disp_cont
88
89  draw_blue:
90      movi    r9, 0x007C
91      sthio   r9, (r8)
92      br      disp_cont
93
94  draw_border:
95      movi    r9, 0x1F7C
96      movi    r12, 272
97      movi    r13, 480
98
99  draw_border_top:
100     movhi   r8, %hi(VRAM_CTRL_BASE)
101     ori     r8, r8, %lo(VRAM_CTRL_BASE)
102     addi    r8, r8, 0x0400
103     mov     r11, r0
104 dbt_loop:
105     sthio   r9, (r8)
106     addi    r8, r8, 2
107     addi    r11, r11, 1
108     bgeu    r11, r13, draw_border_bot
109     br      dbt_loop
110
111 draw_border_bot:
112     movhi   r8, %hi(VRAM_CTRL_BASE)
113     ori     r8, r8, %lo(VRAM_CTRL_BASE)
114     movhi   r14, %hi(277504)
115     ori     r14, r14, %lo(277504)
116     add     r8, r8, r14
117     mov     r11, r0
118 dbb_loop:
119     sthio   r9, (r8)
120     addi    r8, r8, 2
121     addi    r11, r11, 1
122     bgeu    r11, r13, draw_border_left
123     br      dbb_loop
124
125 draw_border_left:
126     movhi   r8, %hi(VRAM_CTRL_BASE)
127     ori     r8, r8, %lo(VRAM_CTRL_BASE)
128     mov     r11, r0
129 dbl_loop:
130     sthio   r9, (r8)
131     addi    r8, r8, 0x0400
132     addi    r11, r11, 1
133     bgeu    r11, r12, draw_border_right
134     br      dbl_loop
135
136 draw_border_right:
137     movhi   r8, %hi(VRAM_CTRL_BASE)
138     ori     r8, r8, %lo(VRAM_CTRL_BASE)
139     add     r8, r8, r13
140     add     r8, r8, r13
141     subi    r8, r8, 1
142     mov     r11, r0
143 dbr_loop:
144     sthio   r9, (r8)
145     addi    r8, r8, 0x0400
146     addi    r11, r11, 1
147     bgeu    r11, r12, test_disp_done
148     br      dbr_loop
149
150 disp_reset:
151     mov     r11, r0
152     addi    r12, r12, 1
153     br      disp
154
155 test_disp_done:
156     ret
```