

# **EE/CS 52 SoPC Digital Oscilloscope**

**Technical Manual**

Santiago Navonne

June 2014

# Contents

<b>1. Introduction</b>	<b>3</b>
<b>2. Hardware</b>	<b>3</b>
2.1. System Overview . . . . .	3
2.2. FPGA . . . . .	10
2.2.1. NIOS Processor . . . . .	14
2.2.2. Triggering and FIFO . . . . .	16
2.2.3. Debouncer . . . . .	22
2.2.4. Decoder . . . . .	24
2.2.5. VRAM Controller . . . . .	26
2.2.6. Display Controller . . . . .	45
2.3. Reset Logic . . . . .	54
2.4. Clock Logic . . . . .	54
2.5. JTAG Interface . . . . .	54
2.6. Power Supply . . . . .	54
2.7. Buffers . . . . .	56
2.8. Memory . . . . .	58
2.8.1. RAM . . . . .	58
2.8.2. ROM . . . . .	64
2.8.3. VRAM . . . . .	67
2.9. Display . . . . .	69
2.10. Touch Screen Controller . . . . .	69
2.11. Rotary Encoders . . . . .	70
2.12. Analog Interface . . . . .	72
2.12.1. Analog Front-End . . . . .	74
2.12.2. Analog to Digital Converter . . . . .	76
2.13. Revision History . . . . .	79
<b>3. Software</b>	<b>80</b>
3.1. System Overview . . . . .	80
3.2. Initialization and Main Loop . . . . .	80
3.3. User Interface . . . . .	80
3.4. Trace Processing . . . . .	82
3.5. Key Hardware Interface . . . . .	82
3.6. Display Hardware Interface . . . . .	82
3.7. Analog Hardware Interface . . . . .	83
<b>Appendices</b>	<b>85</b>
<b>A. Original Documents</b>	<b>85</b>
<b>B. Software Code</b>	<b>93</b>

# 1. Introduction

This document describes the system workings and details of the EE/CS 52 System-on-Programmable-Chip (SoPC) Digital Oscilloscope.

The guide describes first the hardware, and then the software, giving for both a system overview, followed by a detailed description of every part of the system. In Appendix A, the schematics and printed circuit board used in the original prototype can be found.

## 2. Hardware

This section explains how the system's hardware works, from the system overview to the detailed description of each element. The interactions of components are described, and detailed schematics, timing diagrams, and board layouts are provided.

### 2.1. System Overview

The highest level illustration of the structure of the system is provided in the block diagram of Figure 1. The parts colored blue are created within the FPGA component ( $U2$ ), while the parts colored red are outside components.

The central component of the system is the NIOS II CPU, a soft-core device generated within the FPGA. A NIOS II/f processor is used upon power-up, and can be upgraded to the faster NIOS II/s by connecting the device to a computer. Within the NIOS CPU are included the chip select decoding and interrupt control logic sections. The chip select decoding logic uses the address bus to activate the chip select control line of the device being accessed, if it requires one. The interrupt controller processes interrupt control signals from hardware devices and makes them available to software procedures.

The display controller, also included within the FPGA, controls the VRAM serial clock and all of the display timing signals, updating the VRAM serial data bus as needed to ensure that data is correctly shown on the display. The debouncers and decoders take signals from the user input sections of the system (i.e. rotary encoders and push-button switches), and process them to translate them into interrupt signals for the processor. These signals are accessed through a Parallel IO (PIO) interface. The triggering logic is configured through a PIO interface, and reads the signal output by the ADC, determining the correct moment to trigger based on triggering mode, level, slope, and delay parameters. The component then instructs the FIFO to start writing samples as necessary. The First-In First-Out (FIFO) data structure stores samples to be processed by the CPU. The FIFO starts being filled when instructed by the trigger, and once full transmits this signal to both the trigger, which disables the trigger signal, and the CPU, which goes ahead and processes the samples.

Outside the FPGA but closely related is the reset logic, which generates a reset signal for the NIOS CPU on power-up, power failure, and when requested by the user. Similarly, the clock logic is an outside component that generates a constant, 38 MHz clock signal used throughout the system. A JTAG connector and interface is used to program and debug the FPGA and NIOS CPU.

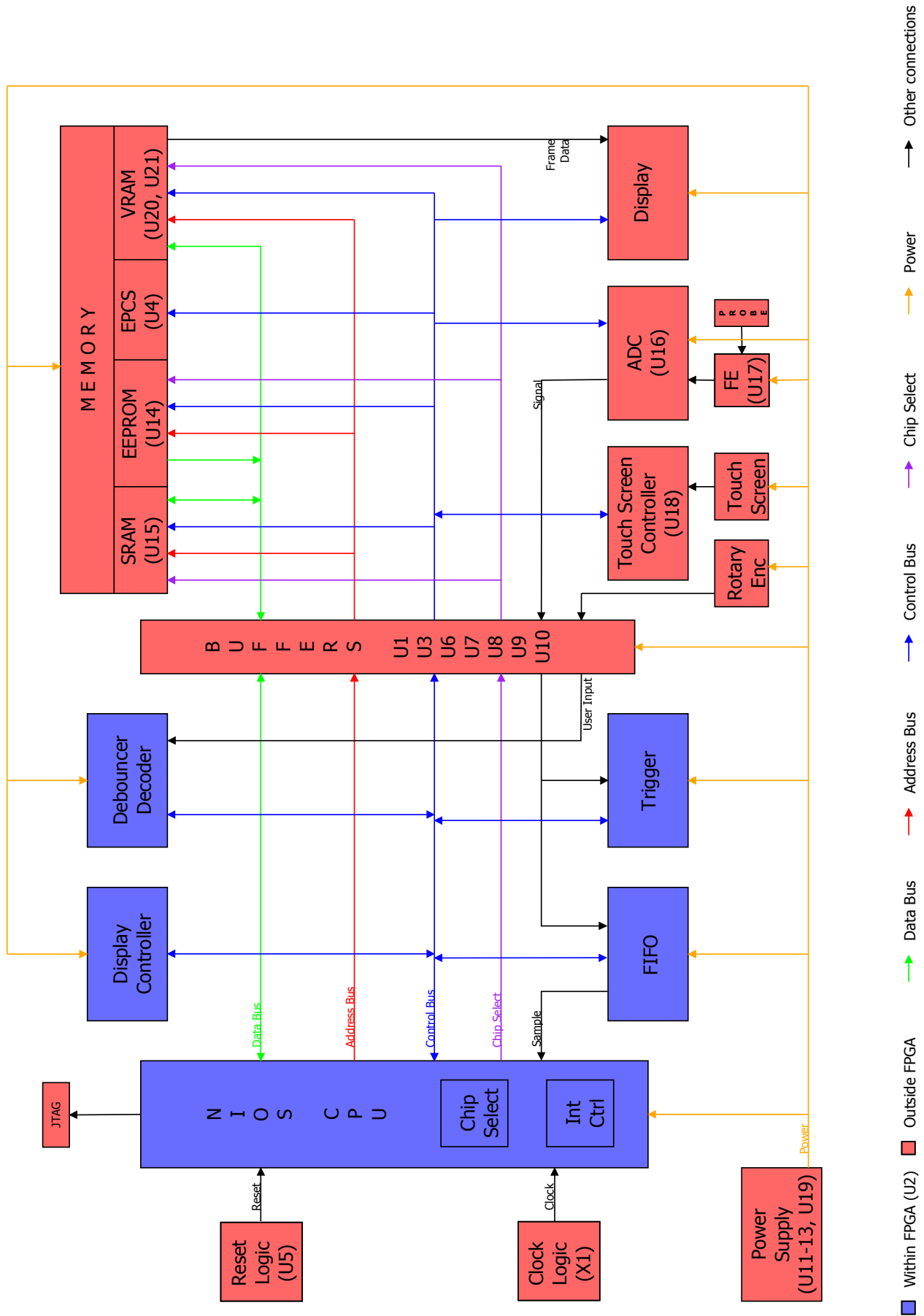


Figure 1: Highest level block diagram of the SoPC Oscilloscope system. The diagram is described in Section 2.1.

Every signal exiting the FPGA, with the sole exception of the I<sup>2</sup>C bus, is buffered. These chips provide a layer of protection for the FPGA, as well as more driving power and voltage flexibility for the components being operated.

Four memory devices are used by the system. The Serial Configuration device (EPCS) is used to store the FPGA that is loaded upon start-up. The device uses dedicated serial control signals to communicate with the FPGA. The Static Random Access Memory (SRAM) device is the system's volatile memory, used to store the software's variables, stack, and other uninitialized memory. The device shares the data and address buses with the other memory devices, and is selected and controlled by a small set of exclusive signals from the CPU. The Electrically Erasable Programmable Read-Only Memory (EEPROM) device is the system's non-volatile memory, used to store code and constants. This device also shares the data and address buses with the other memory devices, and it is, too, selected and controlled by a small set of exclusive signals from the CPU. Two Video RAM (VRAM) devices are used as a buffer for the frames being shown on the display. Data is put there by the CPU via the VRAM controller, and subsequently extracted serially and shown on the LCD by the display controller. This device shares the data bus with other memory devices, while the address bus is exclusive from the controller to the device. A set of VRAM specific control signals is also exclusive to this device.

The display is controlled by timing signals from the display controller, which, synchronized with the VRAM controller, ensure that pixels are output over a dedicated bus between display and VRAM at the right moment to be shown in the correct region of the display.

An Analog Front-End (AFE) scales and shifts signals from the oscilloscope probe as needed to prepare them for input into the Analog-to-Digital Converter (ADC). This device, in turn, reads the samples and converts them into digital values, which are then directly relayed to the triggering logic and FIFO within the FPGA. The ADC is also clocked by a control signal routed through the FPGA.

Two rotary encoders with momentary push-button switches provide the user-input interface of the system: the devices are connected to the FPGA and then the decoders and debouncers, which filter process the signals before making them available to the CPU. Finally, a touch screen controller, currently unimplemented, communicates with the CPU over an I<sup>2</sup>C bus and a dedicated interrupt line used to identify touch screen events. This line is made available to the CPU through a PIO interface. The controller uses an analog interface to drive and read the touch screen.

All these elements, after being designed in detail and their connections finalized, are physically placed on a printed circuit board. The front of the board is illustrated in Figure 2, where each section is highlighted in a different color: the FPGA and related components are colored red; the buffers are green; the memory devices are yellow; the power supply circuitry is blue; the analog interface is orange; the display connector is pink; and the rotary encoders are brown. The components without any highlighting are prototyping and debugging holes and pins, unused in the final design. The back of the board is shown in figure 3. Note that no components other than capacitors and resistors are placed on the back of the board, and thus nothing is highlighted.

The memory map of the system is shown in figure 4. The *JTAG* device is used for debugging

purposes the; *trig\_period*, *trig\_level*, *fifo\_data*, *fifo\_full*, *fifo\_delay*, and *trig\_ctrl* are parallel IO devices used to interface with the triggering logic. Each of these locations in memory contains several registers necessary for the interface to function. *pio\_0* is another parallel IO device, this time used to interface with the rotary encoders and push buttons. Finally, *ram*, *rom*, and *vram* are the memory devices described above. Note that only memory devices have assigned chip select signals, since the other components do not require them; furthermore, note that the VRAM chip select signal does not exist in hardware external to the FPGA.

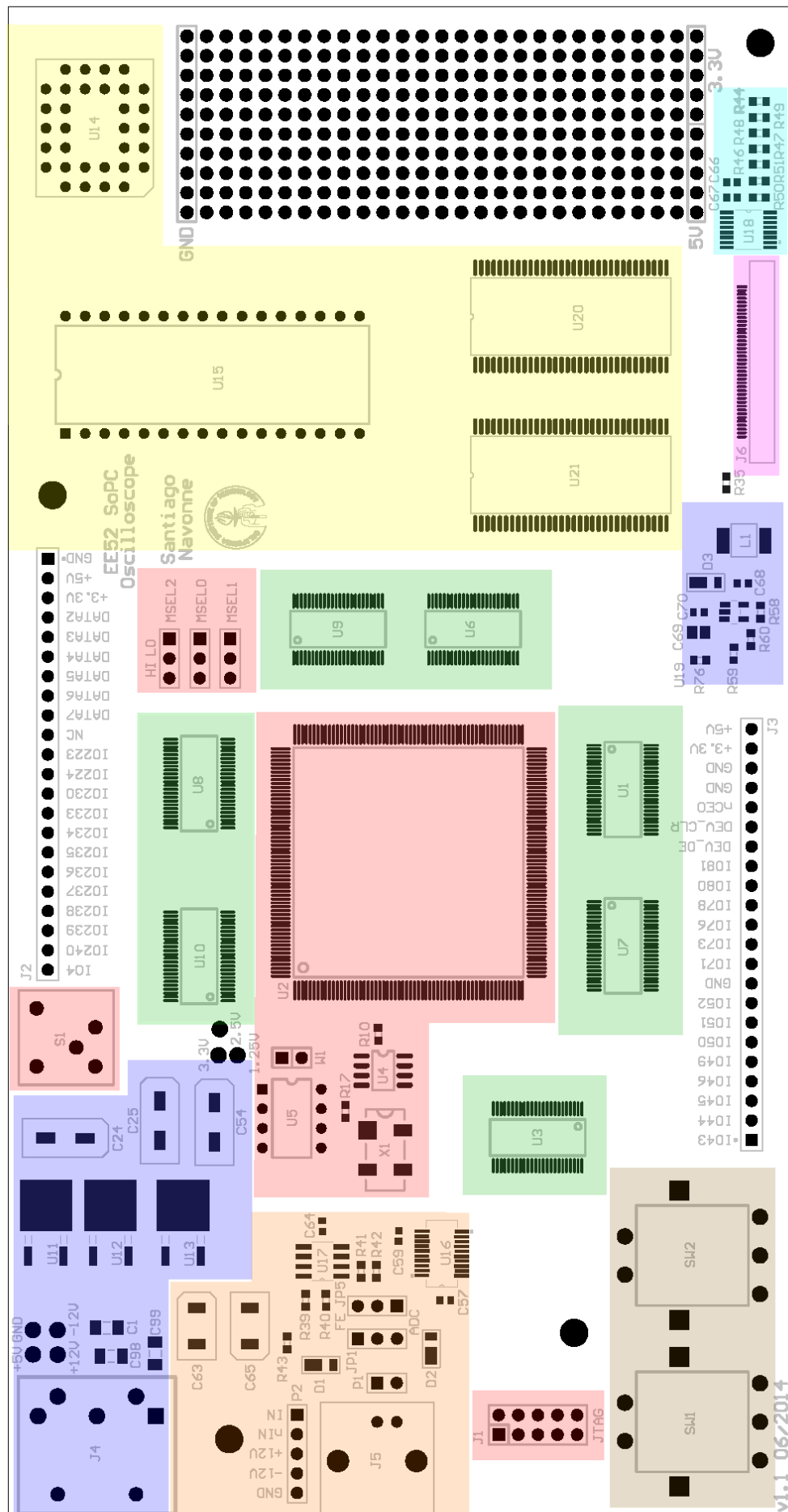


Figure 2: Front side of the system’s Printed Circuit Board (PCB). The color each section is highlighted and identifies the corresponding block.

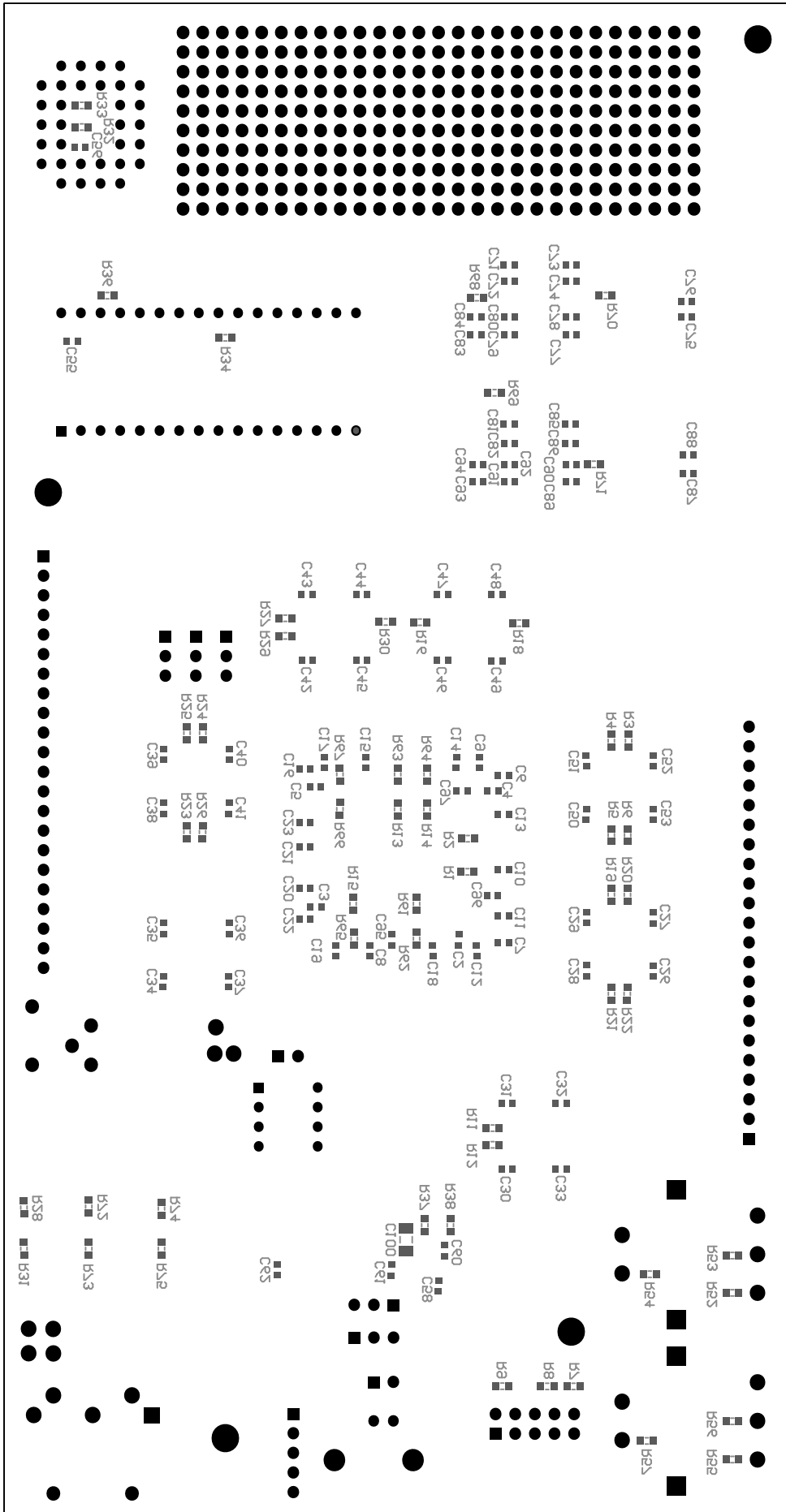


Figure 3: Back side of the system's Printed Circuit Board (PCB). Note that this side is only used for routing and placement of passive parts, such as resistors and capacitors.



Device	Address Range	Size (b...	...	View
fifo_data	0x00241140 - 0x0024114F	16		jtag
fifo_full	0x00241130 - 0x0024113F	16		
jtag	0x00241180 - 0x00241187	8	..	trig period
pio_0	0x002410A0 - 0x002410BF	32		trig level
ram	0x00220000 - 0x0023FFFF	131072	..	fifo data
rom	0x00180000 - 0x001FFFFF	524288	..	fifo full
trig_ctrl	0x00241060 - 0x0024107F	32		trig delay
trig_delay	0x00241120 - 0x0024112F	16		
trig_level	0x00241150 - 0x0024115F	16		pio 0
trig_period	0x00241160 - 0x0024116F	16		
vram	0x00000000 - 0x000FFFFF	1048576	..	trig ctrl
				ram <b>CS1</b>
				rom <b>CS0</b>
				vram

Figure 4: Memory map of the system within the NIOS processor, with associated chip select lines and addresses.

## 2.2. FPGA

An Altera EP3C25Q240 Cyclone III Field Programmable Gate Array (FPGA), *U2*, is the central unit of the system. The component and its associated parts are at the center of the PCB, highlighted in red in Figure 2. Its connections at a system's level are illustrated in the schematic of Figure 5. The device is programmed and debugged through a JTAG interface. After debugging, the final design is loaded upon power-on from the serial memory device *U4* through lines *DATA0* and *DATA1*, clocked by *DCLK* and configured through *nCS0*; the debugging and design loading configuration is determined by the *MSEL2..0* jumpers *JP2*, *JP3*, and *JP4*; for the correct operation of the device, *MSEL2* and *MSEL0* should be jumped on the HI position, while *MSEL1* should be configured to the LO position. Pins 100 and 103 are pulled high through *R1* and *R2* to act as the I<sup>2</sup>C bus lines *SDA* and *SCL*. Configuration lines *INIT\_DONE*, *CONF\_DONE*, and *nSTATUS* are pulled high, while *nCE* and *CLKUSR* are tied low. Configuration lines *DEV\_OE*, *DEV\_CLRn*, and *nCEO* can be left floating, and are therefore connected to break-out pins in *J3*.

The device contains the system's CPU, as well as all of the logic needed to process analog signals, debounce keys, decode rotary encoders, control the VRAM, and control the display. The logic was designed using the Altera tool chain: Quartus and QSys.

Within the FPGA, several components interact with one another. Figure 6 illustrates these components and their interactions.

The top left section of the diagram constitutes the user input section. Rotary encoder channels A (*ROT1A*, *ROT2A*) and B (*ROT1B*, *ROT2B*), as well as the push-button lines (*PUSH1*, *PUSH2*), for both devices, are input on pins 57, 63-65, 68, 69. The inputs are processed through debouncers (*DBC1*, *DBC2*) and decoders (*DEC1*, *DEC2*), which filter the signals, generating events as appropriate at their outputs. These events are collected into a bus, that is input to the NIOS processor (*CPU1*) at the *switches\_in* port of *PIO\_0*.

The middle left section of the diagram is the devices triggering logic. The signal from the ADC (*SIG7..0*) is input on pins 9, 13, 18, 21, 37-39, 41; and then connected to the *DATA7..0* input of the triggering block (*TRIG1*). The trigger's general clock (*GCLK*) is connected to the system clock input at pin 31 (*CLK*). All the other lines of the trigger block are connected to their PIO counterpart at the processor. Note that all the control lines (*SLOPE*, *AUTO\_TRIG*, *FIFO\_WE*, *READ*, *RESET*) are collected into a single bus that is then connected to the processor's *trig\_ctrl* PIO interface. The ADC's sample clock on pin 72 (*ACLK*) is clocked at a constant 38 MHz through the system clock (*CLK*).

The bottom left block, *SR1*, is a necessary component for the functioning of the system's serial memory device (*U4*).

In the bottom right, we recognize the VRAM (*VRAM1*) and display (*DISP1*) controllers. These components' inputs are connected to the processor (*CPU1*), with the exception of the interconnected serial row update request (*UREQ*) and acknowledge (*UACK*) signals. The VRAM receives its own dedicated address bus shifted right twice to turn 4-byte addressing into single-word addressing (*vaddr\_out19..2* to *A17..9* and *A8..0*). The dedicated chip select (*CS*) and shared write enable (*WE*) signals are operated as for any other generic controller. All clocks are connected to the system clock. Finally, these two components output all the necessary timing

signals for the display and VRAM devices on pins 82, 83, 87, 88, 93-95, 98. Additionally, the VRAM controller outputs a *RDY* signal, converted to an active-high *WAIT* signal for the processor through inverter *NOT*, to regulate the VRAM access cycles, which use a variable number of wait states.

In the top right, we see the system's NIOS processor (*CPU1*). Apart from the previously described connections, the devices shared address (*ADDR\_BUS18..0*) and data (*DATA\_BUS23..0*) buses for the ROM and RAM devices are output on pins 114, 117-120, 126-128, 131-135, 137, 139, 142, 143, 146-148, 160, 161, 166, 167, 181-189, 194-197, 200-203, 217. The write enable signal (*WE*) is output to the RAM on pin 168, and is also used as the direction signal for the data bus buffers (*U6*, *U9*) on pin 207 (*DATA\_DIR*). The RAM and ROM's dedicated chip select signals (*CS1*, *CS0*) are output on pins 171 and 173, respectively.

Finally note that I<sup>2</sup>C bus lines *I2C\_SDA* and *I2C\_SCL* on pins 103 and 100, as well as the *PENIRQ* input on pin 70, are left unconnected since the touch screen interface remains unimplemented. Also note that the display enable line (*DISP*) is tied high to permanently enable the device, and that buffer *U10* is configured as always enabled, output by pulling *U10\_DIR* on pin 214 and *U10\_OE* on pin 216 low.

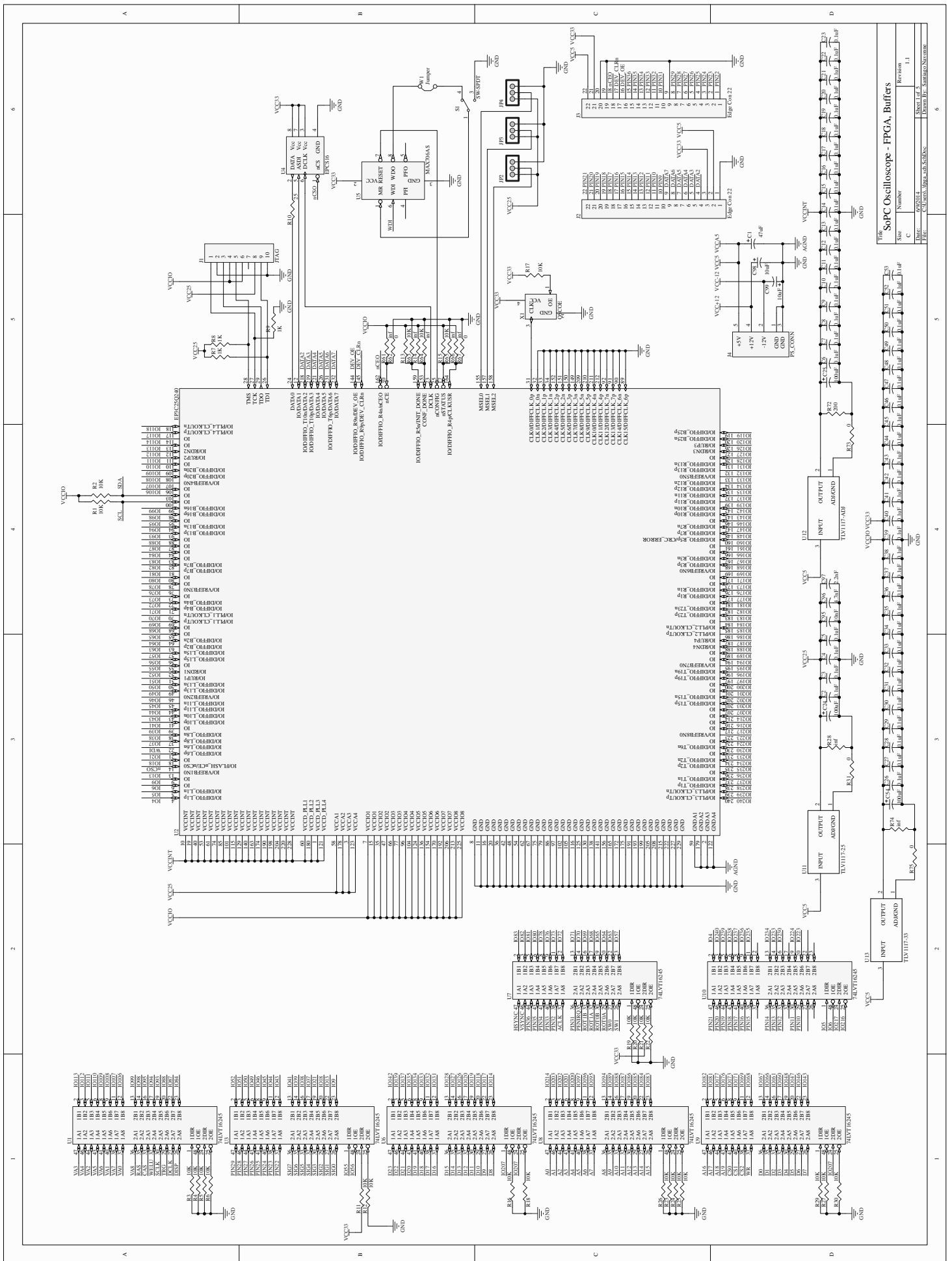


Figure 5: Schematic of the FPGA and related components. Also included are the buffers and power supply. The document is described in Section 2.2.



13

### **2.2.1. NIOS Processor**

The NIOS processor used in the project is generated with Altera's QSys. Table 1 summarizes all the components included within the synthesized device.

Note that the Interrupt Controller and Chip Select blocks are automatically implemented by the Altera tool chain, and are therefore not described in this document; these are compiled together with the SoPC design in QSys.

Device	Type	Description
clk_0	Clock Source	System's main clock, generated at 38 MHz.
nios	NIOS II Processor	NIOS II/f Central Processing Unit, host of all of the system's software.
ram	Generic Tri-State Controller	RAM device controller, used to store volatile data such as variables and the stack.
rom	Generic Tri-State Controller	ROM device controller, used to store non-volatile data such as code and constants.
vram	Generic Tri-State Controller	VRAM device controller, used as a frame buffer to output data to the display.
pin_sharer	Tri-State Conduit Pin Sharer	Component that combines all data and address lines into a single bus.
bridge	Tri-State Conduit Bridge	Device that outputs the memory devices' shared and exclusive lines to FPGA pins.
jtag	JTAG UART	Debugging interface for standard output.
pio_0	PIO (Parallel I/O)	6-bit debounced and decoded rotary encoder and push button inputs. Bits 0 and 1 are the left and right rotation events of rotary encoder 2. Bits 2 and 3 are the left and right rotation events of rotary encoder 1. Bits 4 and 5 are the push-button press events of rotary encoder push-buttons 2 and 1, respectively. All of these bits generate interrupts on their falling edges.
trig_period	PIO (Parallel I/O)	Output for the 32-bit sampling period of the analog interface (time between samples, in number of 38 MHz clock cycles). Individual bit set and clear registers are disabled for this component.
trig_level	PIO (Parallel I/O)	Output for the configured level, as an 8-bit value where 0 is the most negative value, and 255 the most positive value. Individual bit set and clear registers are disabled for this component.
fifo_data	PIO (Parallel I/O)	Input for the current 8-bit sample being output by the FIFO data structure.
trig_ctrl	PIO (Parallel I/O)	Output for the triggering logic control values. Bit 0 is an active-high auto trigger enable signal; bit 1 is the slope bit, 1 for negative slope and 0 for negative slope; bit 2 is the active-low write enable signal for the FIFO, which is enable when a sample is started and disabled when it is completed; bit 3 is the FIFO's read clock; bit 4 is an active-high reset signal for the triggering logic. Individual bit set and clear registers are enabled for this component.
fifo_full	PIO (Parallel I/O)	1-bit input for the interrupt signal indicating that the FIFO is full. Interrupts are generated on rising edges of the line.
trig_delay	PIO (Parallel I/O)	Output for the 32-bit trigger delay value, in number of sample times where the minimum valid value is 1. Note that the value to be output is actually the desired value minus one. Individual bit set and clear registers are disabled for this component.

Table 1: Central Processing Unit and related devices configured within the QSys part of the FPGA design.

### 2.2.2. Triggering and FIFO

The triggering logic and FIFO of component *TRIG1* acquire samples from the analog interface as instructed by the processor, and then make them available on a serial data structure until the CPU is ready to process them. The block structure of this component is illustrated in figure 7.

The very top section of the diagram divides the system clock (*GCLK*) down to the sample clock requested by the CPU. The requested value is input as the 32-bit duration of the sample clock in number of system clock cycles (*CLK\_PERIOD31..0*). A counter (*CT1*) counts system clocks, and the output count is then compared in *CMP1* to the required period divided by two (shifted right twice in *W1*): when the count is less than the compare value, the output of *CMP1* is high; when the count is greater, the output is low. The counter is cleared by *CMP4* when the count reaches the clock period or when the triggering logic is reset (*G8*), effectively generating a 50

Manual trigger events are generated in the middle section of the block diagram. Here, the sample from the ADC (*DATA7..0*) is sent through three chained delayed flip-flops (TDFF2) to remove any glitches, and then compared to the desired trigger level (input at *LEVEL7..0*). *CMP2* thus generates the *TL* and *TEQ* signals required by the ScopeTrigger state machine, instantiated in *TRIG1*. This component, which also takes the slope (*SLOPE*), sample clock, and reset signal (*RESET*), generates a trigger event (*TrigEvent*) when the input signal intersects the desired trigger level with the requested slope. The component is part of the EE/CS 52 library, and its code is provided in the next few pages for reference.



```

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 --      01 Mar 14   Santiago Navonne Removed unnecessary architectures.
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         TEQ         : in std_logic;      -- signal and trigger levels equal
39         TLT         : in std_logic;      -- signal level < trigger level
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
73
74     -- the output is always the high bit of the state encoding
75     TrigEvent <= CurrentState(2);
76
77
78     -- compute the next state (function of current state and inputs)
79
80     transition: process (Reset, TS, TEQ, TLT, CurrentState)
81     begin
82
83         case CurrentState is           -- do the state transition/output
84
85             when IDLE =>               -- in idle state, do transition
86                 if (TS = '0' and TLT = '1' and TEQ = '0') then
87                     NextState <= WAIT_POS;    -- below trigger and + slope
88                 elsif (TS = '1' and TLT = '0' and TEQ = '0') then
89                     NextState <= WAIT_NEG;    -- above trigger and - slope
90                 else
91                     NextState <= IDLE;        -- trigger not possible yet
92                 end if;
93
94             when WAIT_POS =>           -- waiting for positive slope trigger
95                 if (TS = '0' and TLT = '1') then
96                     NextState <= WAIT_POS;    -- no trigger yet
97                 elsif (TS = '0' and TLT = '0') then
98                     NextState <= TRIGGER;    -- got a trigger
99                 else
100                     NextState <= IDLE;        -- trigger slope changed
101                 end if;
102
103             when WAIT_NEG =>           -- waiting for negative slope trigger
104                 if (TS = '1' and TLT = '0' and TEQ = '0') then
105                     NextState <= WAIT_NEG;    -- no trigger yet
106                 elsif (TS = '1' and (TLT = '1' or TEQ = '1')) then
107                     NextState <= TRIGGER;    -- got a trigger
108                 else
109                     NextState <= IDLE;        -- trigger slope changed
110                 end if;
111
112             when TRIGGER =>            -- in the trigger state
113                 NextState <= IDLE;          -- always go back to idle
114
115             when others =>
116                 NextState <= IDLE;
117
118         end case;
119
120         if Reset = '1' then           -- reset overrides everything
121             NextState <= IDLE;        -- go to idle on reset
122         end if;
123
124     end process transition;
125
126
127     -- storage of current state (loads the next state on the clock)
128
129     process (clk)
130     begin
131
132         if clk = '1' then             -- only change on rising edge of clock
133             CurrentState <= NextState; -- save the new state information
134         end if;
135
136     end process;

```

```
137  
138  
139 end assign_statebits;  
140
```

Trigger events generated by *TRIG1* are then sent through the delay logic. Since the events only last one clock (i.e. only a pulse is generated), *TrigEvent* is used to set J/K flip flop *FF2*. The output of *FF2* enables a counter (*CT2*) that counts sample clocks. When the number of sample clocks from the trigger event reaches the requested 32-bit delay (*DELAY31..0*), *CMP3*'s output goes high. This output clears *FF2*, disabling the counter until the next trigger event, and *CT2*, resetting the counter. Note that *CT2* is also reset on *RESET* events. *CMP3*'s output is therefore a single sample-clock long pulse that is low at all times, except for *DELAY31..0* sample clocks after a *TrigEvent*.

The bottom section of the diagram uses the so far generated delayed trigger events and other settings from the CPU to correctly fill the FIFO (*FIFO1*). When automatic triggering is disabled (*AUTO\_TRIG* low), counter *CT8* gets constantly cleared and is therefore “bypassed.” If writing to the FIFO is disabled because no sample has been started (*FIFO\_WE* high), J/K flip-flop *FF1* will be cleared, and its output, *FIFO1*'s *wrreq*, will be disabled; no data will thus be written to the FIFO. If writing to the FIFO is enabled (*FIFO\_WE* low), *FF1* will be set through *G4* whenever a delayed trigger event (output of *CMP3*) is received. Once *FF1* is set, *wrreq* becomes enabled, and samples from the ADC (*DATA7..0*), sent through three additional delayed flip-flops (*TDF1*) for pipelining, are written to the FIFO. When the FIFO becomes full, *G5* clears *FF1*, disabling writing to *FIFO1*. The FIFO full signal also acts as an interrupt for the processor through output *FIFO\_FULL*, prompting it to read the completed sample. The sample is read by first disabling writing (sending *FIFO\_WE* low), and then bit-banging the read clock (*READ*). Since the read enable line (*rdreq*) is permanently enabled, every time the *READ* line transitions from low to high a new sample is output from *FIFO1* onto *SAMPLE7..0*. *RESET* signals clear *FIFO1*.

When automatic triggering is enabled (*AUTO\_TRIG* high), the logic described above still applies with one addition: counter *CT8* is enabled as long as writing to the FIFO is enabled too (*FIFO\_WE* low), and the FIFO is not currently being written to (input to *wrreq* low). When the counter reaches 380,000, a timeout designed to count 10 ms on the 38 MHz system clock, comparator *CMP12* transitions to high, causing the FIFO to start being written to (*wrreq* sent high). This in turns disables counting in *CT8*, which causes the output of *CMP12* to stay high until writing is disabled (that is, until the samples are read). This mechanism effectively forces the generation of a trigger 10 ms after a sample is started, if no regular trigger was generated before then.

In a typical interaction, the processor will configure all triggering settings, and send the *FIFO\_WE* line low to start the sample. When a sample is completed, the *FIFO\_FULL* line will exhibit a rising edge. The processor must thus disable the *FIFO\_WE* line (send it high), and clock the *READ* line appropriately to extract all 512 samples from *FIFO1*. Any time settings are changed, the processor should reset the triggering logic by pulsing the *RESET* line high. The line may be maintained high while settings are being changed.



### 2.2.3. Debouncer

Two debouncers (*DBC1*, *DBC2*) are used to filter the input from the two rotary encoder push-buttons. Figure 8 illustrates the structure of the debouncer component.

The component takes the signal from the push-button, *BUTTON*, and a debouncing clock, *DEBOUNCE\_CLK*, as inputs. *BUTTON* is assumed to be active-low to support pulled-up switches that are grounded upon activation.

As long as *BUTTON* is high, counter *CT3* will keep getting cleared, and its count enable line will be active; the counter will therefore not count. When *BUTTON* goes low, the counter will start counting *DEBOUNCE\_CLK*. When this value reaches 380,000, the output of *CMP4* will go high. The compare constant was chosen to generate a 10 ms delay on the 38 MHz clock used in the system, and input into *DEBOUNCE\_CLK*. *CMP4*'s output is then inverted to create an active-low signal that is used to prevent *CT3* from counting (and therefore wrapping around and debouncing the signal again) and output on line *DEBOUNCED*.

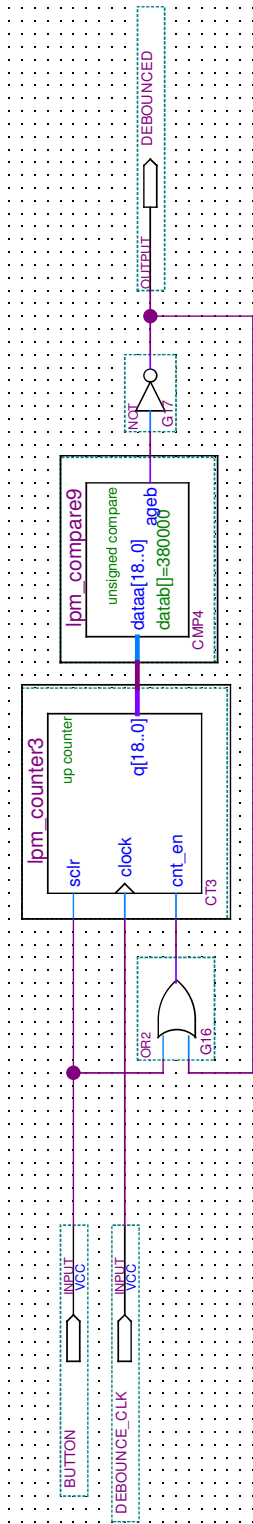


Figure 8: Block diagram of the push-button debouncer component within the FPGA. The document is described in Section 2.2.3.

#### 2.2.4. Decoder

Two decoders (*DEC1*, *DEC2*) are used to decode the input from the two rotary encoders' rotation. Figure 9 illustrates the structure of the decoder component.

The component takes the *A* and *B* signals from the rotary encoder, which is assumed to have detents only on *A* and *B* active (high), and a decoding clock, *DECODE\_CLK*, as inputs. The bottom part of the block diagram generates an enable signal, while the top part determines the direction of rotation.

To determine whether the encoder was turned (i.e. to generate an enable signal "clock"), an S/R flip-flop, *FF4*, is used. *FF4* is set when both *A* and *B* are high, that is when the encoder finds itself at a detent. *FF4* is reset when both *A* and *B* are low, that is when the encoder is between detents. Since rotary encoders only bounce between adjacent positions, the set and reset signals on *FF4* will not bounce. The output of *FF4* is an active-high enable signal.

To determine the direction of rotation, *A* is XOR'd in *G9* with the previous clock's *B*, saved through delayed flip-flop *FF3*. The output of *G9* will be high if the encoder was turned clockwise, and low if the encoder was turned counter-clockwise, due to the order in which positions occur within the encoder. The output is then fed to delayed flip-flop *FF5* for pipelining, and to delayed flip-flop *FF6*, which is clocked on the above described enable signal, to latch the direction only when a detent is reached. The output of *FF6* is thus directly NAND'd with the enable signal in *G13* to generate the active-low clockwise rotation interrupt, *RIGHT*, and inverted through *G12* and then NAND'd with the enable signal in *G15* to generate the active-low counter-clockwise rotation interrupt, *LEFT*. Note that *RIGHT* and *LEFT* will be high most of the time, and exhibit a falling edge when the encoder is turned in the corresponding direction.



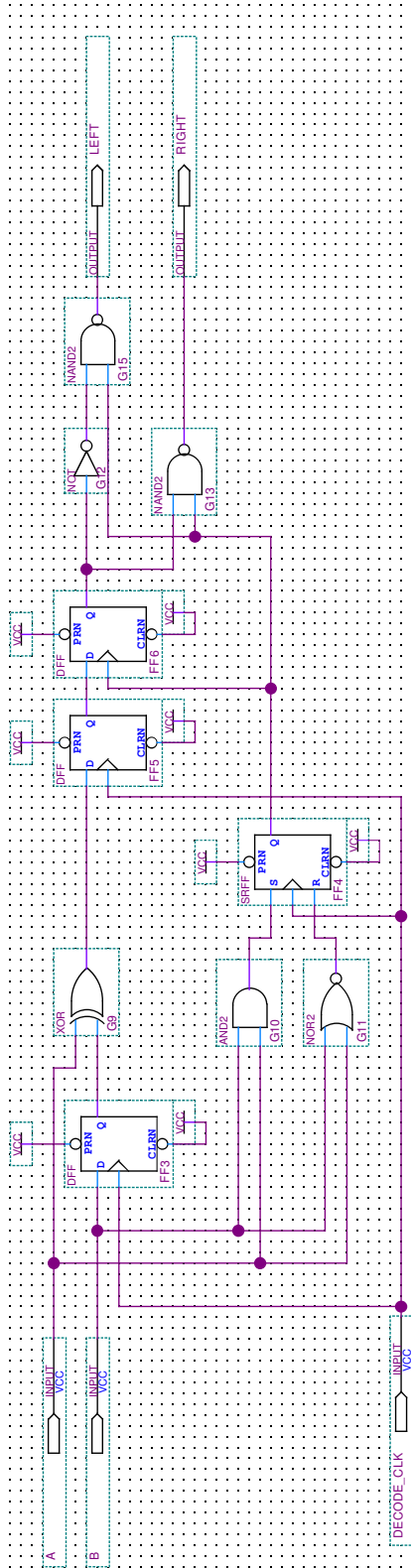


Figure 9: Block diagram of the rotary encoder decoder component within the FPGA. The document is described in Section 2.2.

### 2.2.5. VRAM Controller

The VRAM controller mediates interactions between the processor and the Video RAM. The processor can thus interact with the controller as if it were a regular memory device with variable wait states (i.e., an access is completed when the *WAIT* line goes low), and the controller generate the signals actually needed by the VRAM chips. Additionally, the VRAM controller performs row updates when requested by the display controller. A generic block diagram of the interactions between processor, VRAM controller, and display controller can be seen in Figure 10. The VRAM controller is illustrated in more detail in the block diagram of Figure 11.

The component takes a 18-bit address bus,  $A[17..0]$ , an active-low write enable signal, *WE*, an active low chip select signal, *CS*, an active high serial update request signal, *UREQ*, an active high reset signal, *RESET*, and a clock, *CLK*.

The address is divided into a row address,  $A[17..9]$ , and a column address,  $A[8..0]$ . These are muxed, together with a row address generated by *CT4* and a blank column address, in *MUX1*, allowing the VRAM control to output the correct address onto the VRAM address bus,  $VADDR[8..0]$ , as needed. *CT4* counts every time a new row transfer is requested, and wraps around the number of rows in the display, 272: this effectively causes the row address used for row updates to sequentially go through the whole display.

The bulk of the controller's logic is implemented in VHDL, using a Moore state machine. The next few pages provide the code of this state machine component. The state machine starts in the idle state (*IDLE*), where all the output signals are maintained at their neutral levels. If a row update is being requested (*UREQ* active), the state machine transitions into a row update cycle (*SERIAL1..6*) to ensure that the display controller is provided with a new row of data before it needs to start outputting at the end of its row porches. If no row update is being requested, but a read or write is being requested (*CS* active), the state machine transitions into the corresponding read (*READ1..6*) or write (*WRITE1..5*) cycle. If no memory access is being requested either, the controller performs a memory refresh by transitioning into the refresh (*REFRESH1..6*) cycle. Note that after each cycle, the state machine transitions back into the idle state.

```

1  -----
2  --
3  -- SoPC Oscilloscope VRAM Controller
4  --
5  -- Implementation of the VRAM Controller for the SoPC Oscilloscope project.
6  -- The state machine generates the necessary timing signals for the VRAM
7  -- based on the needs of the CPU and display controller. Additionally,
8  -- it refreshes the VRAM as necessary whenever no other cycle is being
9  -- performed.
10 -- The inputs to the system determine what action needs to be performed:
11 -- cs+we requests a read or a write, while ureq requests a SAM row update.
12 -- The system then outputs the necessary timing signals, and a rdy/uack
13 -- signal to notify the sender of the end of the cycle.
14 -- The state machine is implemented using a Moore state machine and a state
15 -- assignment architecture.
16 --
17 --
18 -- Revision History:
19 --     13 Apr 04   Glen George           Initial template.
20 --     20 Feb 14   Santiago Navonne     Initial revision.
21 --
22  -----
23
24
25 -- bring in the necessary packages
26 library ieee;
27 use ieee.std_logic_1164.all;
28
29
30 --
31 -- Oscilloscope VRAM Controller entity declaration
32 --
33
34 entity VRAMCtrl is
35     port (
36         we      : in  std_logic;      -- read / not write
37         cs      : in  std_logic;      -- chip select
38         ureq    : in  std_logic;      -- serial row update request
39         clk     : in  std_logic;      -- clock
40         Reset   : in  std_logic;      -- reset the system
41         ras     : out std_logic;      -- RAS timing signal
42         cas     : out std_logic;      -- CAS timing signal
43         trg     : out std_logic;      -- transfer/read signal
44         welu    : out std_logic;      -- write signal
45         asrc    : out std_logic;      -- address source selection
46         arow    : out std_logic;      -- address row/column selection
47         uack    : out std_logic;      -- serial row update acknowledge
48         rdy     : out std_logic;      -- read/write acknowledge
49     );
50 end VRAMCtrl;
51
52
53
54 --
55 -- Oscilloscope VRAM Controller Moore State Machine
56 --
57
58 architecture assign_statebits of VRAMCtrl is
59
60     subtype states is std_logic_vector(10 downto 0);    -- state type
61
62     -- define the actual states as constants
63
64     -- bits are: RAS CAS TRG WE ASRC AROW UACK RDY ID[2..0]
65     constant IDLE      : states := "1111110000"; -- waiting for events
66
67     constant READ1     : states := "0111110000"; -- read state 1
68     constant READ2     : states := "0101110000"; -- read state 2

```

```

69  constant READ3      : states := "00011000000"; -- read state 3
70  constant READ4      : states := "00011101000"; -- read state 4
71  constant READ5      : states := "11111100001"; -- read state 5
72  constant READ6      : states := "11111100010"; -- read state 6
73
74  constant WRITE1     : states := "01111100001"; -- write state 1
75  constant WRITE2     : states := "01101000000"; -- write state 2
76  constant WRITE3     : states := "00101001000"; -- write state 3
77  constant WRITE4     : states := "11111100011"; -- write state 4
78  constant WRITE5     : states := "11111100100"; -- write state 5
79
80  constant SERIAL1    : states := "11010100000"; -- serial transfer state 1
81  constant SERIAL2    : states := "01010100000"; -- serial transfer state 2
82  constant SERIAL3    : states := "01110000000"; -- serial transfer state 3
83  constant SERIAL4    : states := "00110000000"; -- serial transfer state 4
84  constant SERIAL5    : states := "11111110000"; -- serial transfer state 5
85  constant SERIAL6    : states := "11111100101"; -- serial transfer state 6
86
87  constant REFRESH1   : states := "10111100000"; -- refresh state 1
88  constant REFRESH2   : states := "00111100001"; -- refresh state 2
89  constant REFRESH3   : states := "00111100010"; -- refresh state 3
90  constant REFRESH4   : states := "00111100011"; -- refresh state 4
91  constant REFRESH5   : states := "11111100110"; -- refresh state 5
92  constant REFRESH6   : states := "11111100111"; -- refresh state 6
93
94  signal CurrentState : states; -- current state
95  signal NextState    : states; -- next state
96
97  begin
98
99
100  -- the output is always the 8 highest bits of the encoding
101  ras <= CurrentState(10);
102  cas <= CurrentState(9);
103  trg <= CurrentState(8);
104  welu <= CurrentState(7);
105  asrc <= CurrentState(6);
106  arow <= CurrentState(5);
107  uack <= CurrentState(4);
108  rdy <= CurrentState(3);
109
110
111  -- compute the next state (function of current state and inputs)
112
113  transition: process (Reset, ureq, we, cs, CurrentState)
114  begin
115
116      case CurrentState is -- do the state transition/output
117
118          -- transition from idle
119          when IDLE => -- in idle state, do transition
120              if (ureq = '1') then
121                  NextState <= SERIAL1; -- serial update request has priority
122              elsif (cs = '0' and we = '1') then
123                  NextState <= READ1; -- read request
124              elsif (cs = '0' and we = '0') then
125                  NextState <= WRITE1; -- write request
126              else
127                  NextState <= REFRESH1; -- nothing to do; refresh
128              end if;
129
130          -- read cycle
131          when READ1 => -- continue read cycle
132              NextState <= READ2;
133
134          when READ2 => -- continue read cycle
135              NextState <= READ3;
136

```

```

137     when READ3 =>                -- continue read cycle
138         NextState <= READ4;
139
140     when READ4 =>                -- continue read cycle
141         NextState <= READ5;
142
143     when READ5 =>                -- continue read cycle
144         NextState <= READ6;
145
146     when READ6 =>                -- end read cycle
147         NextState <= IDLE;
148
149 -- write cycle
150     when WRITE1 =>               -- continue write cycle
151         NextState <= WRITE2;
152
153     when WRITE2 =>               -- continue write cycle
154         NextState <= WRITE3;
155
156     when WRITE3 =>               -- continue write cycle
157         NextState <= WRITE4;
158
159     when WRITE4 =>               -- continue write cycle
160         NextState <= WRITE5;
161
162     when WRITE5 =>               -- end write cycle
163         NextState <= IDLE;
164
165 -- serial update cycle
166     when SERIAL1 =>              -- continue serial cycle
167         NextState <= SERIAL2;
168
169     when SERIAL2 =>              -- continue serial cycle
170         NextState <= SERIAL3;
171
172     when SERIAL3 =>              -- continue serial cycle
173         NextState <= SERIAL4;
174
175     when SERIAL4 =>              -- continue serial cycle
176         NextState <= SERIAL5;
177
178     when SERIAL5 =>              -- continue serial cycle
179         NextState <= SERIAL6;
180
181     when SERIAL6 =>              -- end serial cycle
182         NextState <= IDLE;
183
184 -- refresh cycle
185     when REFRESH1 =>             -- continue refresh cycle
186         NextState <= REFRESH2;
187
188     when REFRESH2 =>             -- continue refresh cycle
189         NextState <= REFRESH3;
190
191     when REFRESH3 =>             -- continue refresh cycle
192         NextState <= REFRESH4;
193
194     when REFRESH4 =>             -- continue refresh cycle
195         NextState <= REFRESH5;
196
197     when REFRESH5 =>             -- continue refresh cycle
198         NextState <= REFRESH6;
199
200     when REFRESH6 =>             -- end refresh cycle
201         NextState <= IDLE;
202
203     when OTHERS =>               -- default; needed for compilation
204         NextState <= IDLE;

```

```

205
206         end case;
207
208         if Reset = '1' then           -- reset overrides everything
209             NextState <= IDLE;         -- go to idle on reset
210         end if;
211
212     end process transition;
213
214
215     -- storage of current state (loads the next state on the clock)
216
217     process (clk)
218     begin
219
220         if clk = '1' then             -- only change on rising edge of clock
221             CurrentState <= NextState; -- save the new state information
222         end if;
223
224     end process;
225
226
227 end assign_statebits;
228

```

The transitions within the read, write, serial, and refresh cycles are those of typical RAS/CAS DRAM access cycles, shown in the timing diagrams of Figures 12-15 and Tables 2-5, and output the necessary signals for the VRAM controller (*RAS*, *CAS*, *TRG*, *WEL/U*), an active-high row update acknowledge signal (*UACK*) to the display controller when the row update has been completed, an active-high ready signal (*RDY*) to the processor when a read or write cycle has been completed (and, in case of the read cycle, valid data will be present on the data bus for 1 clock following the activation of the signal), and the control signals for the address multiplexer (*MUX*) that determine what section of which address bus should be output: *ASRC* is high when the processor's address bus should be output, and low when the row update address should be output, *AROW* is high when the row address should be output, and low when the column address should.

Note that the controller does not specify a number of wait states, and requires the processor to wait for the *RDY* signal to go high before completing the access cycle instead.

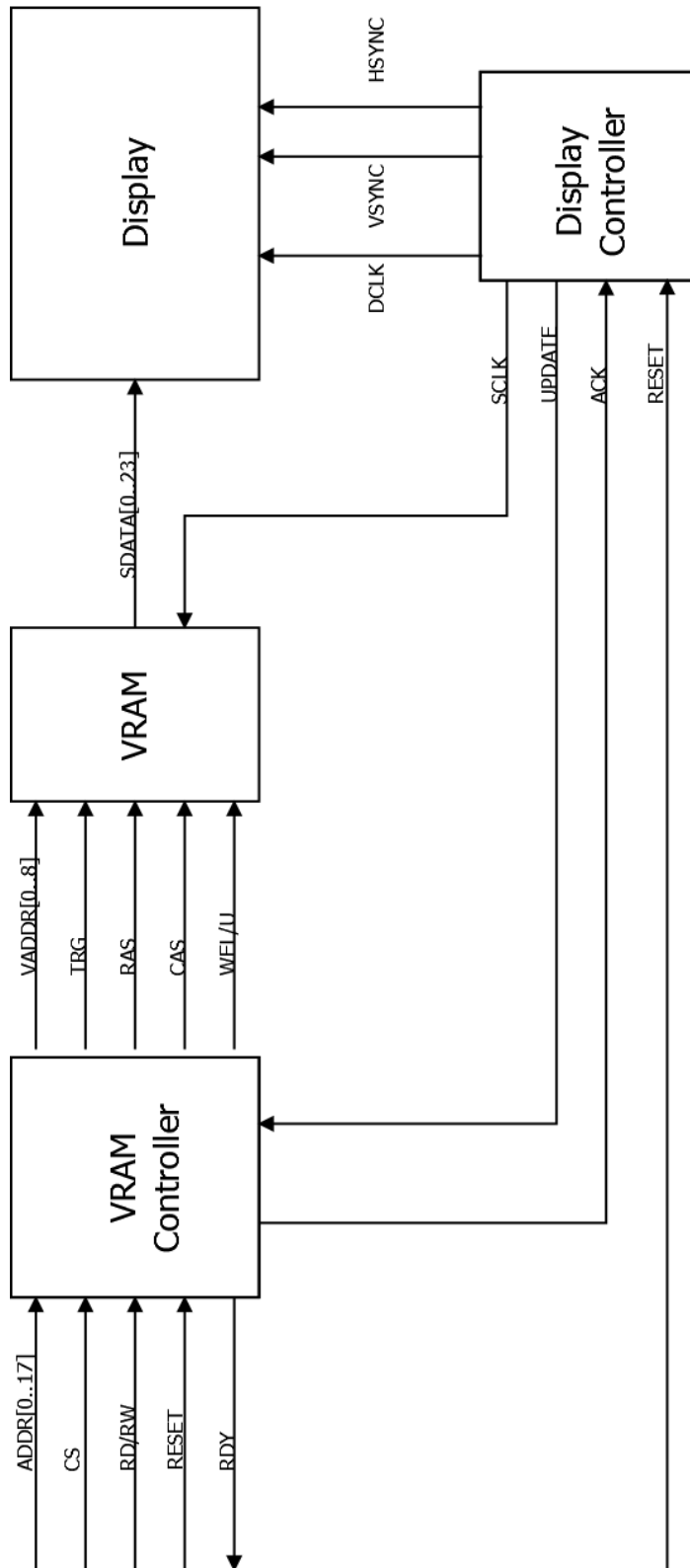


Figure 10: High level, summarizing block diagram of the VRAM and display interface. The VRAM controller listens to the CPU's generic memory controller commands, and generates the necessary timing lines for the VRAM device. The display controller periodically updates the serial row on the VRAM device, outputting new data to the display, while simultaneously generating the necessary timing signals.



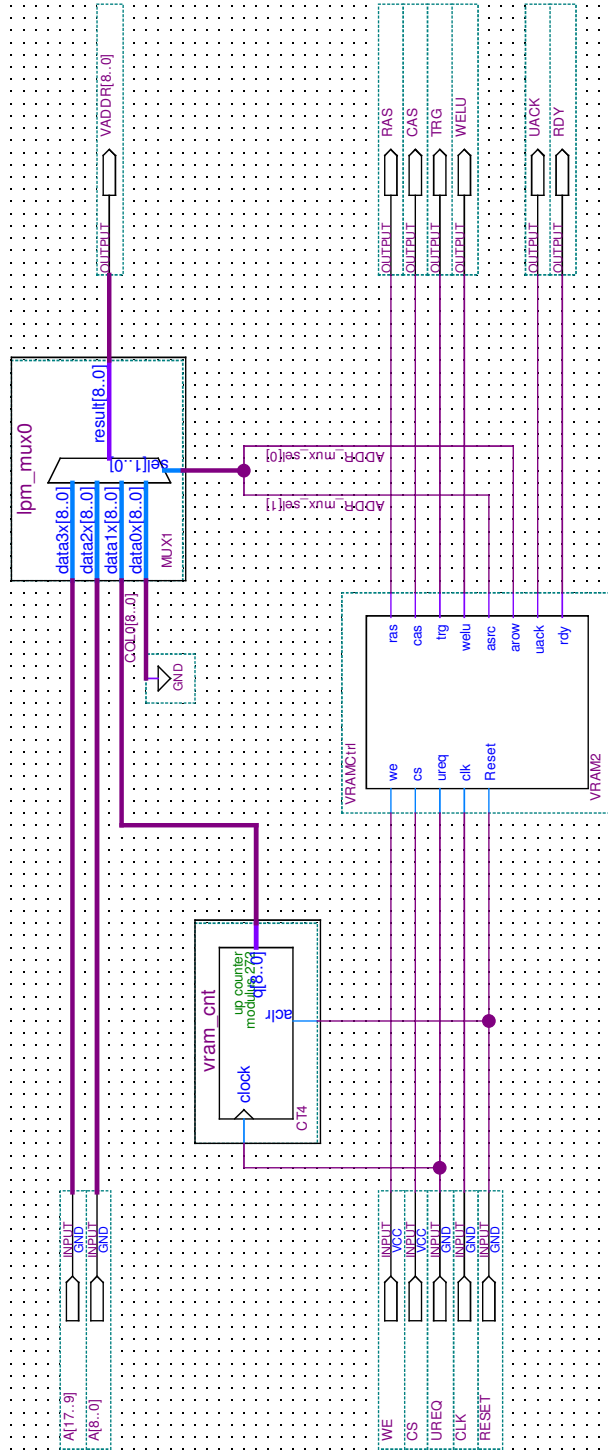


Figure 11: Block diagram of the VRAM controller component within the FPGA. The document is described in Section 2.2.5.

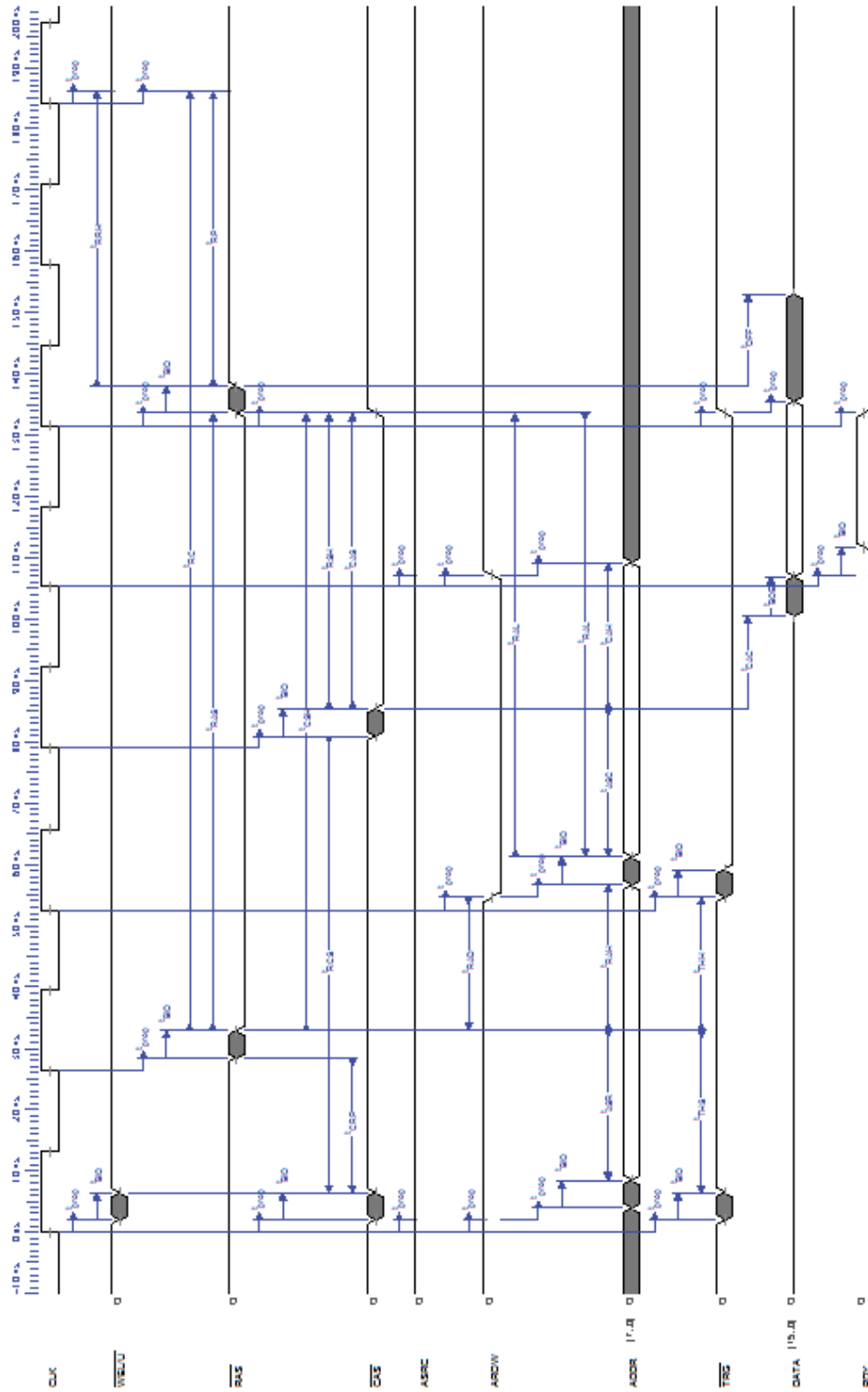


Figure 12: Timing diagram of the read cycle of the VRAM device, described in Section 2.2.5.

General Data				
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX
$t_{prop}$		Propagation Delay		2ns
$t_{BIO}$		Buffer I/O Delay		4.5ns
$t_{BOE}$		Buffer output enable		6.5ns
$t_{RAS}$		RAS Pulse Width	50ns	
$t_{CAS}$		CAS Pulse Width	10ns	
$t_{CSH}$		CAS Hold Time	45ns	
$t_{RC}$		Read/Write Cycle Time	104ns	
$t_{RP}$		RAS Precharge Time	40ns	
$t_{RSH}$		RAS Hold Time	15ns	
$t_{RAH}$		Row Address Hold Time	10ns	
$t_{ASR}$		Row Address Setup	0ns	
$t_{CAH}$		Column Address Hold Time	10ns	
$t_{ASC}$		Column Address Setup Time	0ns	
$t_{THH}$		TRG High Hold Time	10ns	
$t_{THS}$		TRG High Setup Time	0ns	
$t_{CAC}$		CAS Output Time		15ns
$t_{OFF}$		Data Bus Off Time		15ns
$t_{RRH}$		Read Command Hold Time (from RAS)	0ns	
$t_{RAL}$		Column Address to RAS Lead Time	30ns	

Table 2: Table of constraints of the read cycle of the VRAM device, shown in Figure 12 and described in Section 2.2.5.

General Data continued...

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOTES
$t_{RCH}$		Read Command Hold Time (from CAS)	0ns		
$t_{RCS}$		Read Command Setup Time	0ns		
$t_{CRP}$		CAS to RAS Precharge Time	5ns		
$t_{RAL}$		Column Address to RAS Lead Time	30ns		
$t_{RAD}$		RAS to Column Address Delay Time	12ns		



General Data				
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX
$t_{prop}$		Propagation Delay		2ns
$t_{BIO}$		Buffer I/O Delay		4.5ns
$t_{BOE}$		Buffer output enable		6.5ns
$t_{RAS}$		RAS Pulse Width	50ns	
$t_{CAS}$		CAS Pulse Width	10ns	
$t_{CSH}$		CAS Hold Time	45ns	
$t_{RC}$		Read/Write Cycle Time	104ns	
$t_{RP}$		RAS Precharge Time	40ns	
$t_{RSH}$		RAS Hold Time	15ns	
$t_{RAH}$		Row Address Hold Time	10ns	
$t_{ASR}$		Row Address Setup	0ns	
$t_{ASC}$		Column Address Setup Time	0ns	
$t_{CAH}$		Column Address Hold Time	10ns	
$t_{THS}$		TRG High Setup Time	0ns	
$t_{THH}$		TRG High Hold Time	10ns	
$t_{CRP}$		CAS to RAS Precharge Time	5ns	
$t_{WSR}$		WE Setup Time	0ns	
$t_{RWH}$		WE Hold Time	10ns	
$t_{WCS}$		Write Command Setup Time	0ns	

Table 3: Table of constraints of the write cycle of the VRAM device, shown in Figure 13 and described in Section 2.2.5.

General Data continued...

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOTES
$t_{AR}$		Column Address Hold Time (from RAS)	50ns		
$t_{WCH}$		Write Command Hold Time	10ns		
$t_{RAL}$		Column Address to RAS Lead Time	30ns		
$t_{WP}$		Write Command Pulse Width	10ns		
$t_{WCR}$		Write Command Hold Time (from RAS)	50ns		
$t_{RWL}$		Write Command to RAS Lead Time	15ns		
$t_{CWL}$		Write Command to CAS Lead Time	15ns		
$t_{DS}$		Data Setup Time	0ns		
$t_{DH}$		Data Hold Time	10ns		
$t_{DHR}$		Data Hold Time (from RAS)	50ns		

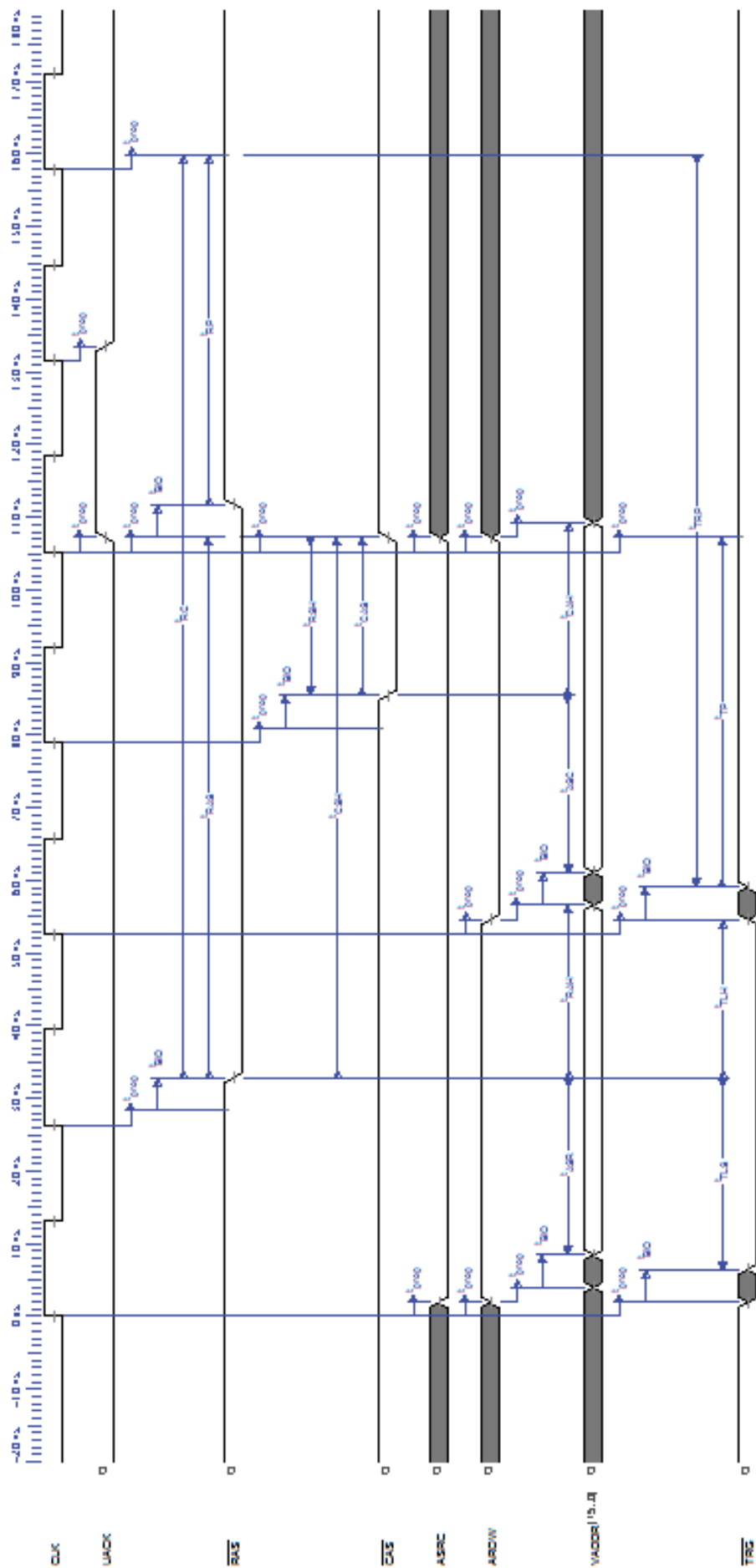


Figure 14: Timing diagram of the serial row transfer cycle of the VRAM device, described in Section 2.2.5.



General Data				
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX
$t_{prop}$		Propagation Delay		2ns
$t_{BIO}$		Buffer Input/Output Delay		4.5ns
$t_{RAS}$		RAS Pulse Width	60ns	
$t_{RC}$		Read/Write Cycle Time	104ns	
$t_{RP}$		RAS Precharge Time	40ns	
$t_{CSH}$		CAS Hold Time	45ns	
$t_{RSH}$		RAS Hold Time	15ns	
$t_{RCD}$		RAS to CAS Delay Time	15ns	42ns
$t_{CAS}$		CAS Pulse Width	10ns	10000ns
$t_{AR}$		Column Address Hold Time (from RAS)	50ns	
$t_{RAD}$		RAS to Column Address Delay Time	12ns	
$t_{RAL}$		Column Address to RAS Lead Time	30ns	
$t_{ASR}$		Row Address Setup	0ns	
$t_{RAH}$		Row Address Hold Time	10ns	
$t_{ASC}$		Column Address Setup Time	0ns	
$t_{CAH}$		Column Address Hold Time	10ns	
$t_{TLS}$		TRG Low Setup Time	0ns	
$t_{TLH}$		TRG Low Hold Time	10ns	

Table 4: Table of constraints of the serial row transfer cycle of the VRAM device, shown in Figure 14 and described in Section 2.2.5.

General Data continued...

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOTES
$t_{RSD}$		RAS to First SC Delay Time	60ns		
$t_{TRP}$		TRG to RAS Precharge Time	40ns		
$t_{TP}$		TRG Precharge Time	20ns		
$t_{TSD}$		TRG to First SC Delay Time	10ns		

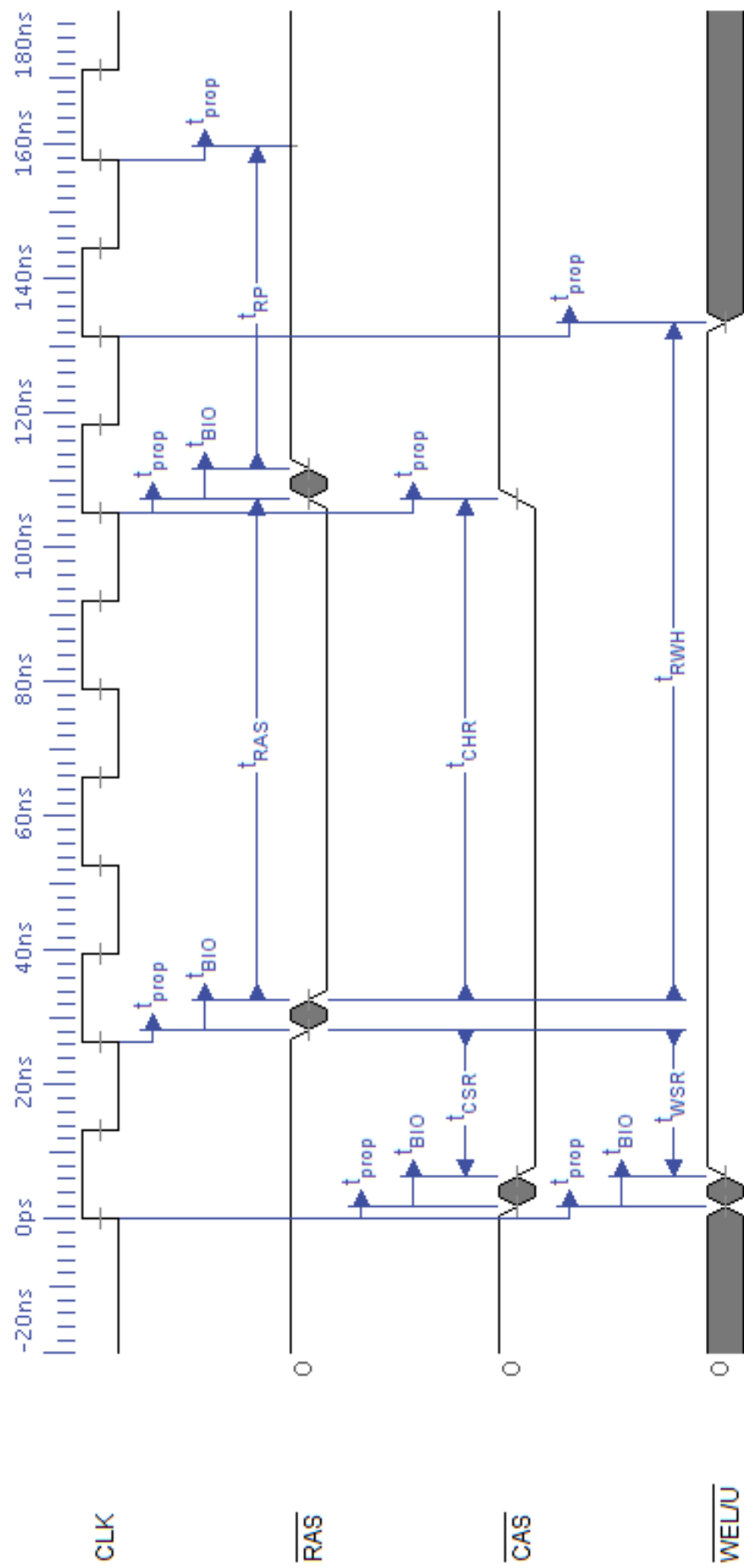


Figure 15: Timing diagram of the refresh cycle of the VRAM device, described in Section 2.2.5.

General Data					
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOTES
$t_{prop}$		Propagation Delay		2ns	
$t_{BIO}$		Buffer Input/Output delay		4.5ns	
$t_{RP}$		RAS Precharge Time	40ns		
$t_{RAS}$		RAS Pulse Width	50ns	10000ns	
$t_{RC}$		Read/Write Cycle Time	104ns		
$t_{RPC}$		RAS Precharge to CAS Active	0ns		
$t_{CSR}$		CAS Setup Time	5ns		
$t_{CHR}$		CAS Hold Time	10ns		
$t_{WSR}$		WE Setup Time	0ns		
$t_{RWH}$		WE Hold Time	10ns		

Table 5: Table of constraints of the refresh cycle of the VRAM device, shown in Figure 15 and described in Section 2.2.5.

### 2.2.6. Display Controller

The display controller outputs data from the VRAM onto the display at a very fast rate, ensuring that the most up-to-date version of the video data is constantly being displayed. A generic block diagram of the interactions between processor, VRAM controller, and display controller can be seen in Figure 10. The display controller is illustrated in more detail in the block diagram of Figure 16.

The controller takes a clock, *CLK*, a reset signal, *RESET*, and a VRAM controller row update acknowledge signal, *UACK*, as inputs. The clock is immediately divided by two by clocking a mod-2 counter, *CT5*, with it, and taking the high bit of the output. The bit is sent through delayed flip-flop *FF10* to remove any glitches.

All the necessary timing signals required by the display are generated using a combination of counters, comparators, and flip-flops. The outline of an interaction is shown in Figure 17; the detailed timing of every transition is shown in the timing diagram of Figure 18 and Table 6.

The display pixel clock, *DCLK*, must always run. It is thus generated from the divided clock, pipelined through delayed flip-flop *FF11* for synchronization.

The frame clock, *VSYNC*, is also always running, going low at the beginning of every frame, staying low for the *VSYNC* pulse period of 10 *HSYNC*s, and then going high and staying high for the remainder of the frame. Each cycle lasts a total of 286 *HSYNC*s. This structure is achieved with counter *CT6*, which counts the number of clocks per frame, and comparator *CMP5*, which determines the moment *VSYNC* should transition. The signal is then pipelined through delayed flip-flop *FF12* to synchronize it with the rest of the controller.

The row clock (*HSYNC*) is also constantly running, going low at the beginning of a row, staying low for the *HSYNC* pulse period of *DCLK*s, and then going high and staying high for the remainder of the row. Each cycle lasts a total of 525 *SCLK*s. This structure is achieved with counter *CT7*, which counts the number of clocks per row, and comparator *CMP9*, which determines the moment *HSYNC* should transition. The signal is then pipelined through two delayed flip-flops, *FF14* and *FF15*, to ensure synchronization with the other signals in the controller.

The serial clock, *SCLK*, is used to shift pixels out of the VRAM. The signal must only run during the display period of each row in order to output the correct region of the VRAM. To achieve this result, *CT7* is used in combination with comparators *CMP10* and *CMP11*, which determine the bounds of within a row where the clock should run (effectively excluding the horizontal front porch of 2 *DCLK*s and the horizontal back porch of 2 *DCLK*s). These conditions are then ANDed in *G20* with the output of comparators *CMP67* and *CMP7*, which use row-counter *CT6* to only enable *SCLK* during the display part of the frame, effectively excluding the the vertical front porch of 2 *SCLK*s and the vertical back porch of 2 *SCLK*s.

The outputs of *CMP6* and *CMP7* are also used to generate a row update request signal for the VRAM controller at the end of the display period within each row. As long as the we're within the display portion of the frame (i.e. between row porches), *G18* will set S/R flip-flop *FF13* when the row clock reaches the end of the display portion of a row, indicated by comparator *CMP8* based on the row clock of *CT7*. The output *FF13* is used as a row update

signal (*UREQ*); the flip-flop is reset when the VRAM controller confirms the completion of the row update (*UACK*).

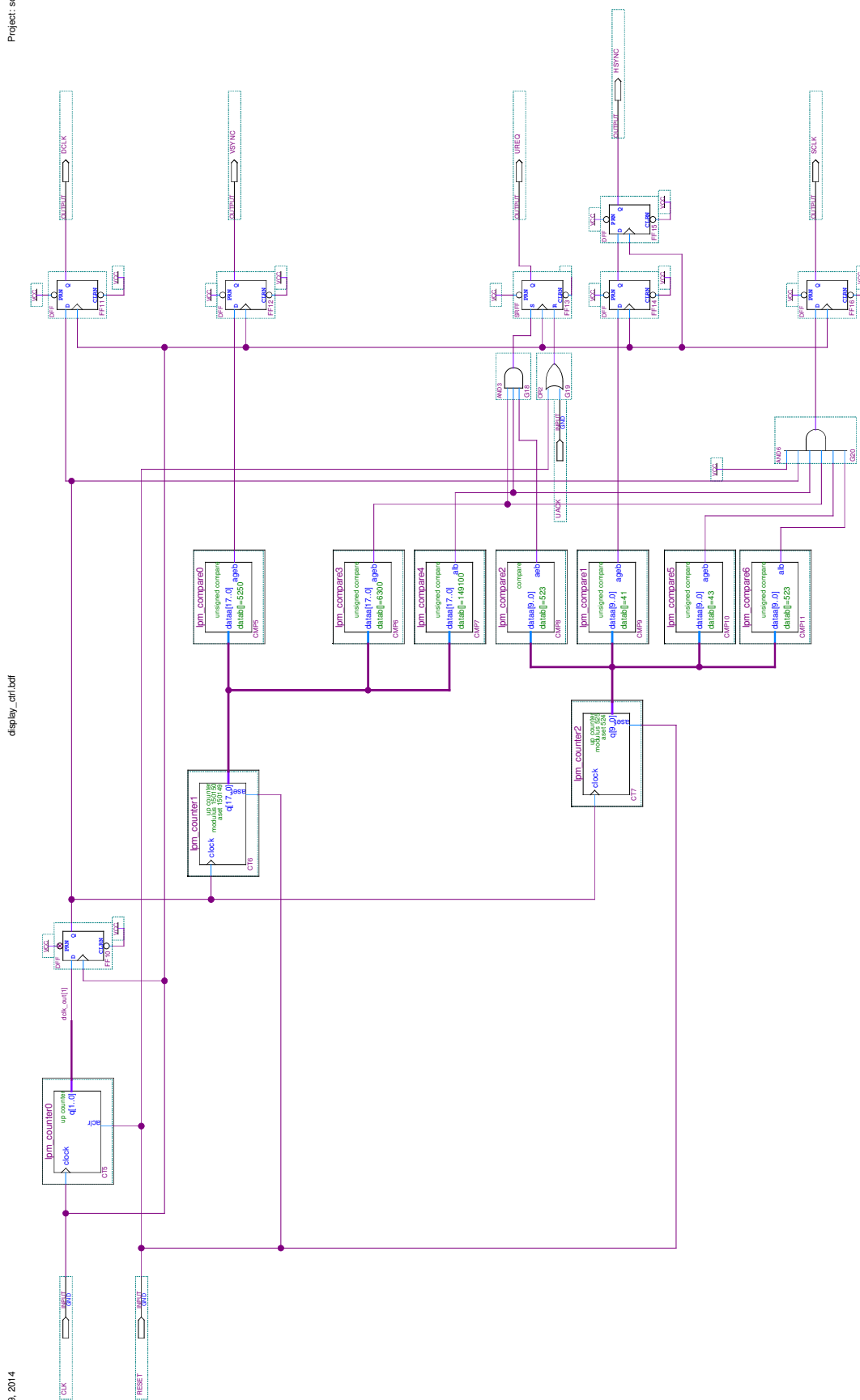
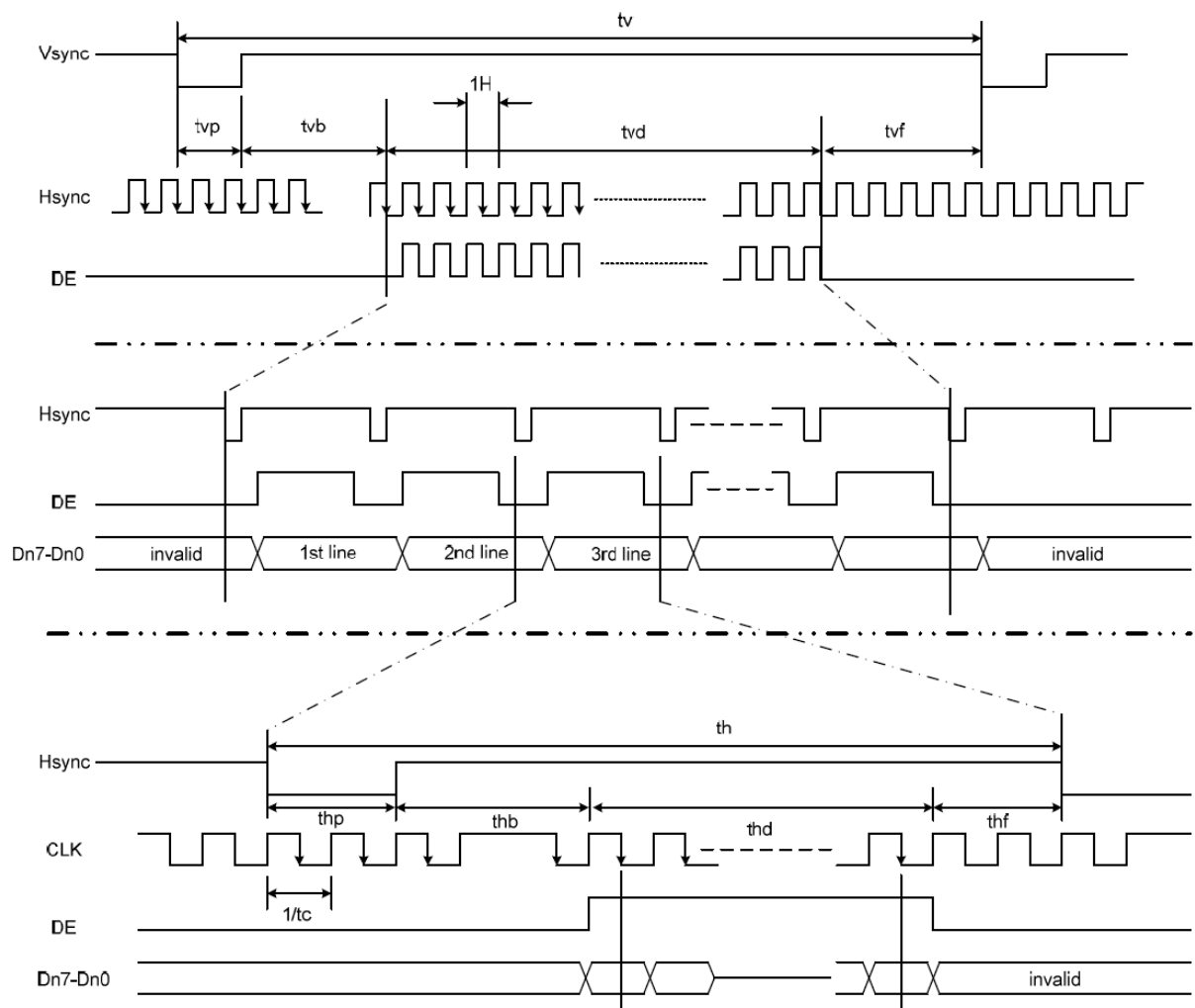


Figure 16: Block diagram of the display controller component within the FPGA. The document is described in Section 2.2.6.



[7]

Figure 17: Summary of the display's frame cycle structure. Source: HX8257 LCD driver datasheet.



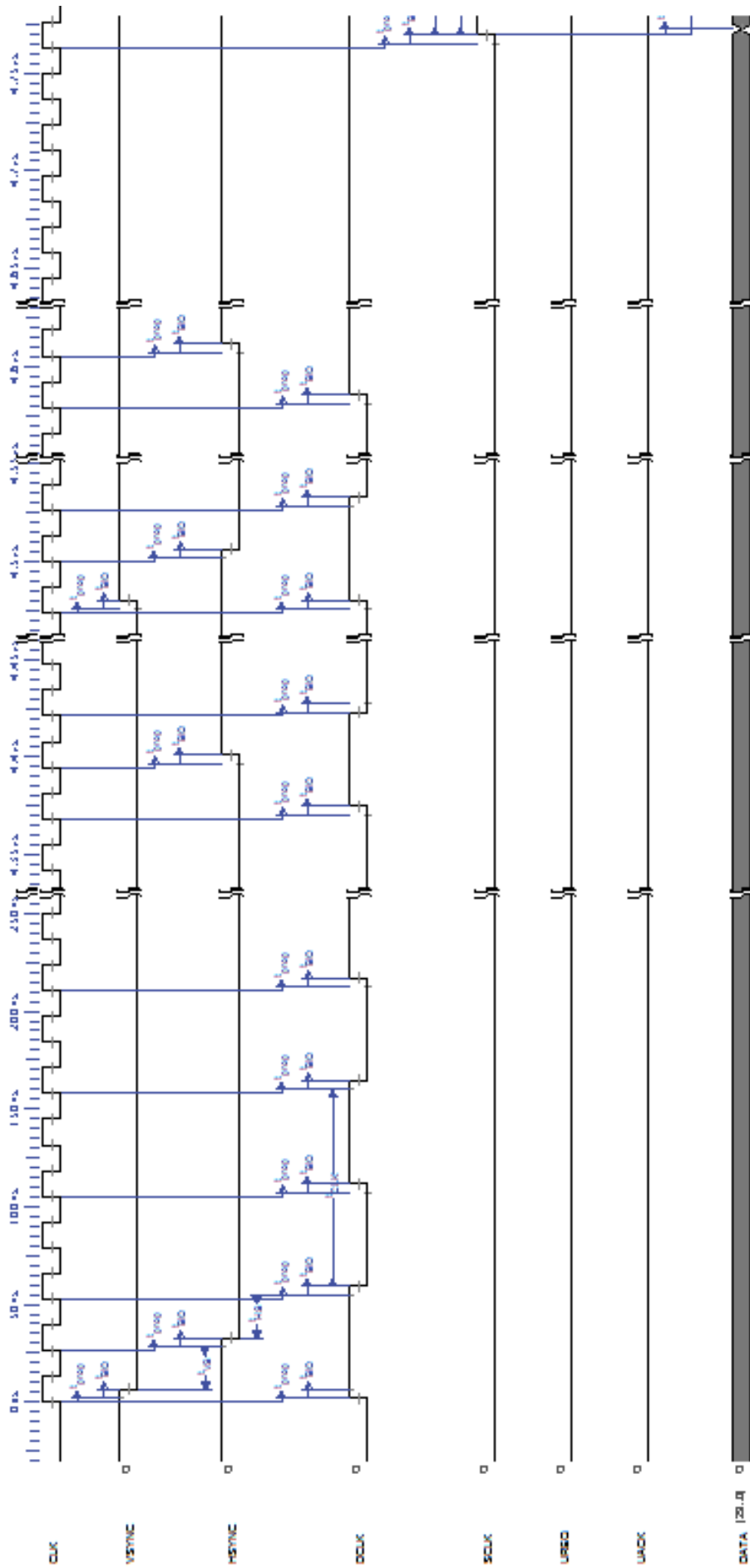


Figure 18: Timing diagram of the display cycle, described in Section 2.2.6.





General Data				
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX
$t_{BIO}$		Buffer I/O Delay		4.5ns
$t_{prop}$		FPGA Propagation Delay		2ns
$t_{CLK}$		DCLK Period	66.7ns	
H		HSync Cycle - 2100		
$t_{VP}$		VSyn Pulse Width - 10 H		
$t_{VB}$		VSyn Back Porch - 2 H		
$t_{VD}$		VSyn Display Period - 272 H		
$t_{VF}$		VSyn Front Porch - 2 H		
$t_{HP}$		HSyn Pulse Width - 164 CLKS	260ns	4.33us
$t_{HB}$		HSyn Back Porch - 8 CLKS		
$t_{HD}$		HSyn Display Period - 1920 CLKS		
$t_{HF}$		HSyn Front Porch - 8 CLKS		
$t_{SCA}$		VRAM Access Time (from SC)		15ns
$t_{SOH}$		VRAM Serial Output Hold Time (rom SC)	3ns	
$t_{DS}$		Data Setup Time	10ns	
$t_{DH}$		Data Hold Time	10ns	
$t_{VS}$		VSyn Setup Time	10ns	
$t_{HS}$		HSyn Setup Time	10ns	

Table 6: Table of constraints of a display cycle, shown in Figure 18 and described in Section 2.2.6.

General Data continued...

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOTES
$t_{SC}$		Serial Clock Pulse Width	5ns		
$t_{SCC}$		Serial Clock Period	18ns		

### 2.3. Reset Logic

A MAX706AS reset chip, *U5* (illustrated in Figure 5), is used to provide power-on reset and manual reset functionality. The chip is closely connected to the FPGA, and therefore highlighted in red in Figure 2. The device natively provides the power-on and power-loss reset functionality, while switch *S1* adds the manual reset functionality by allowing the manual reset pin *MR* to be pulled low: users can thus reset the system by pressing and releasing *S1*. Jumper *W1* can be used to enable watchdog timer expiration reset when shorted: in this configuration, FPGA pin 22 (line *WDI*) would need to transition from low to high or high to low every 1.6s at most to avoid reset. *U5*'s reset output line is connected to *U2*'s *nCONFIG* line, which is an active-low reset input to the FPGA. For the correct operation of the system, *W1* should be left unjumped.

### 2.4. Clock Logic

The FPGA is clocked using a 38 MHz SG363 oscillator, *X1*, illustrated in Figure 5. The device is closely connected to the FPGA, and therefore highlighted in red in Figure 2. Its output enable line is tied high through *R17* to always enable the clock. Its output is connected to the *U2*'s *CLK0* input, and therefore acts as the FPGA's main clock. Every other clock line, *CLK1* through *CLK15*, is tied low and therefore disabled.

### 2.5. JTAG Interface

The FPGA can be programmed and debugged using a JTAG interface, through JTAG connector *J1*, illustrated in Figure 5. Lines *TMS*, *TCK*, *TDO*, and *TDI* control the interface. Pull-up and pull-down resistors *R7*, *R8*, and *R9* are installed as needed by the JTAG interface. The interface is closely related to the FPGA, and therefore highlighted in red in Figure 2.

### 2.6. Power Supply

A +5 V / +12 V / -12 V external power supply is expected to be used with this system. The different power lines are connected through DIN-5 connector *J4*, illustrated in Figure 5. Each one of them is immediately filtered using capacitors *C1*, *C98*, and *C99*. The 5 V line, capable of providing the most current, is then regulated to create the various rails. The 3.3 V source is regulated by a TLV1117-33 regulator, *U13*. The 2.5 V source is regulated by a TLV1117-25 regulator, *U11*. The 1.25 V source is regulated by a TLV1117-ADJ adjustable regulator, *U12*: the output selection resistors *R72* and *R73* are chosen to output the minimum possible voltage, 1.25 V. Each one of the regulators has a 100  $\mu$ F capacitor at the output (*C24*, *C25*, *C54*). The other listed bypass capacitors (*C2-C23*, *C26-C53*, *C95-C97*) are placed as close as possible to every power pin of *U1-U3* and *U6-U10* that uses that regulator, and are sized as required by the component itself.

Additionally, the +5 V line is also regulated to +20 V to power the display's backlight LEDs, as illustrated in Figure 23. This is performed using an LMR62014 boost regulator. Resistors *R59* and *R76* are chosen to select the correct output voltage, while inductor *L1* and capacitor

*C69* are selected to obtain the best output waveform characteristics.

The power supply circuitry is highlighted in blue in Figure 2. Note that all bypass capacitors are placed on the backside of the board, drawn in Figure 3.

## 2.7. Buffers

Every FPGA pin that can be sent through a buffer is; thus, every buffer is placed between the FPGA, *U2*, and some other chip(s). This provides protection against over-voltage, converts every voltage into 3.3 V, and provides better current characteristics, allowing more devices to be driven with the each line. Seven 74LVT16245 buffers, *U1*, *U3*, and *U6-U10* (illustrated in Figure 5), are used to this goal. The delay introduced by these devices is assumed throughout the project to be less than 2 ns.

Buffer *U8* relays the low 16 bits of the address bus (*A15..0*) output from *U2* on both ports; therefore, both of its ports are configured as output (B→A) by pulling the direction pins *1DIR* and *2DIR* low through *R26* and *R24*. The outputs are always enabled, and thus *1OE* and *2OE* are also pulled low through *R23* and *R25*.

Buffer *U9* relays the remaining 4 bits of the address bus (*A19..16*), as well as chip select signals *CS2..0*, write enable signal *WR*, and the bottom 8 bits of the data bus *D0..7*. Port A contains only output signals (B→A), and is therefore configured as always-enabled, output-only by pulling *1DIR* and *1OE* low through *R29* and *R27*. The data bus is always enabled by tying *2OE* low through *R30*; however, it is bidirectional, and its direction (*2DIR*) is therefore controlled by an FPGA output line that mediates the data bus, *IO207*.

Buffer *U6* connects the remaining 16 bits of the data bus (*D23..D8*). These are always-enabled, bidirectional as described above. Its output enable lines, *1OE* and *2OE* are therefore tied low through *R16* and *R18*, while the direction lines are controller by FPGA output line *IO207*.

Buffer *U1* relays VRAM and display signals. It transmits the video address bus (*VA8..0*), VRAM control signals (*RAS*, *CAS*, *WEL/U*, *SCLK*, *TRG*, and *DCLK*), and the display enable signal (*DISP*). Since all the signals are always-enabled, output-only (B→A), the direction and output enable control lines (*1DIR*, *2DIR*, *1OE*, *2OE*) are pulled low through resistors *R3-R6*.

Buffer *U3* carries the ADC output *SIG7..0* to the FPGA on port 2: the relative direction and output enable lines are therefore tied high and low respectively, through *R11* and *R12*. Port 1 connects eight unused FPGA pins. These pins are made available on break-out header *J3*. To allow the configuration of the direction of these lines, the direction and output enable pins are made available to *U2* on pins *IO55* and *IO56*, respectively.

Buffer *U7* bridges the user input lines from the rotary encoders (*SW0,1*; *ROT0A,B*; *ROT1A,B*) and the interrupt line from the touch screen controller (*PENIRQ*), as well as an unused pin made available on *J3*, on port 2. These lines are configured as always-enabled, input-only (A→B) by tying port 2 direction line *2DIR* high through *R21* and output enable line *2OE* low through *R22*. Port 1 connects display timing lines *HSYNC* and *VSYNC*, ADC clock *ACLK*, and five unused pins; all of these lines are configured always-enabled, output-only (B→A) by tying both port 1 control lines low through *R19,20*.

Buffer *U10* connects 12 unused FPGA pins to break-out pins in *J2*. The pins are divided between port 1, with seven connections, and port 2, with five. The direction and output of both ports can be configured by using FPGA pins *IO5,6* and *216,217*.

The buffers are highlighted in green in Figure 2. Note that all bypass capacitors are placed



on the back side of the board, drawn in Figure 3.

## 2.8. Memory

The system uses three memory devices, in addition to the previously mentioned serial ROM used by the FPGA. A Random Access Memory (RAM) chip is used as volatile memory for the NIOS processor. A Read-Only Memory (ROM) device is used for the storage of non-volatile constants and code for the NIOS processor. Two Video RAM (VRAM) chips are used as a frame buffer for the display: the NIOS processor loads frame data into the memory device, which is subsequently read by the display controller and output to the display.

The memory devices are highlighted in yellow in Figure 2. Note that all bypass capacitors are placed on the backside of the board, drawn in Figure 3.

### 2.8.1. RAM

A HM628128B 128 Kword x 8-bit RAM chip, *U15*, constitutes the system's volatile storage. The device's connections are illustrated in Figure 19. The device is accessible by the processor at addresses 0x220000-0x23FFFF, as shown in Figure 4.

The chip is connected to the bottom 17 bits of the address bus (*A16..0*) and the bottom 8 bits of the data bus (*D7..0*), which are then routed through buffers *U8* and *U9* to *U2*, the FPGA. Note that both buses are shared between multiple devices. Also note that since this is a volatile memory device, the alignment of the address and data lines does not matter; therefore, both buses are “shuffled” on their interface with the chip to simplify routing on the PCB.

*U15* is selected by using active-low line *CS1* uniquely, which is then routed through buffer *U9* and into *U2*; the active-high counterpart is tied high through *R36* to render it unnecessary. The chip is configured as always-enabled by pulling the output-enable line low through *R34*. The processor selects whether it's reading or writing to the chip by modulating the active-low write-enable line, *WR*. This line is bridged by buffer *U9* and then routed into the FPGA, *U2*.

The interactions between the processor and the memory device, which follow a generic memory controller interaction model, are illustrated in the timing diagrams of Figures 20-21 and Tables 7-8. Note from the diagrams that the device requires 3 wait states when reading, and 3 wait states when writing.

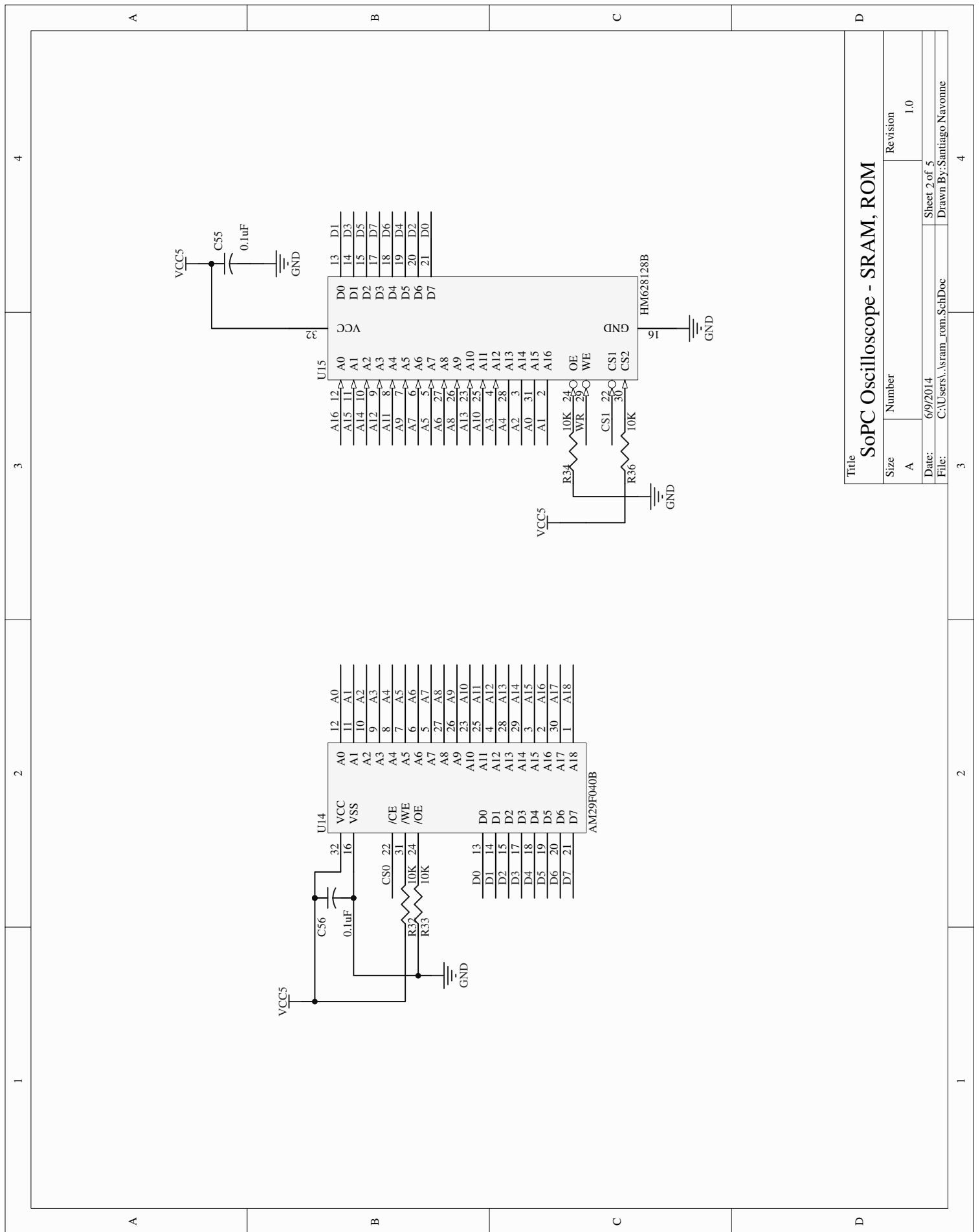


Figure 19: Schematic of the RAM and ROM memory devices. Further details are provided in Sections 2.8.1 and 2.8.2.

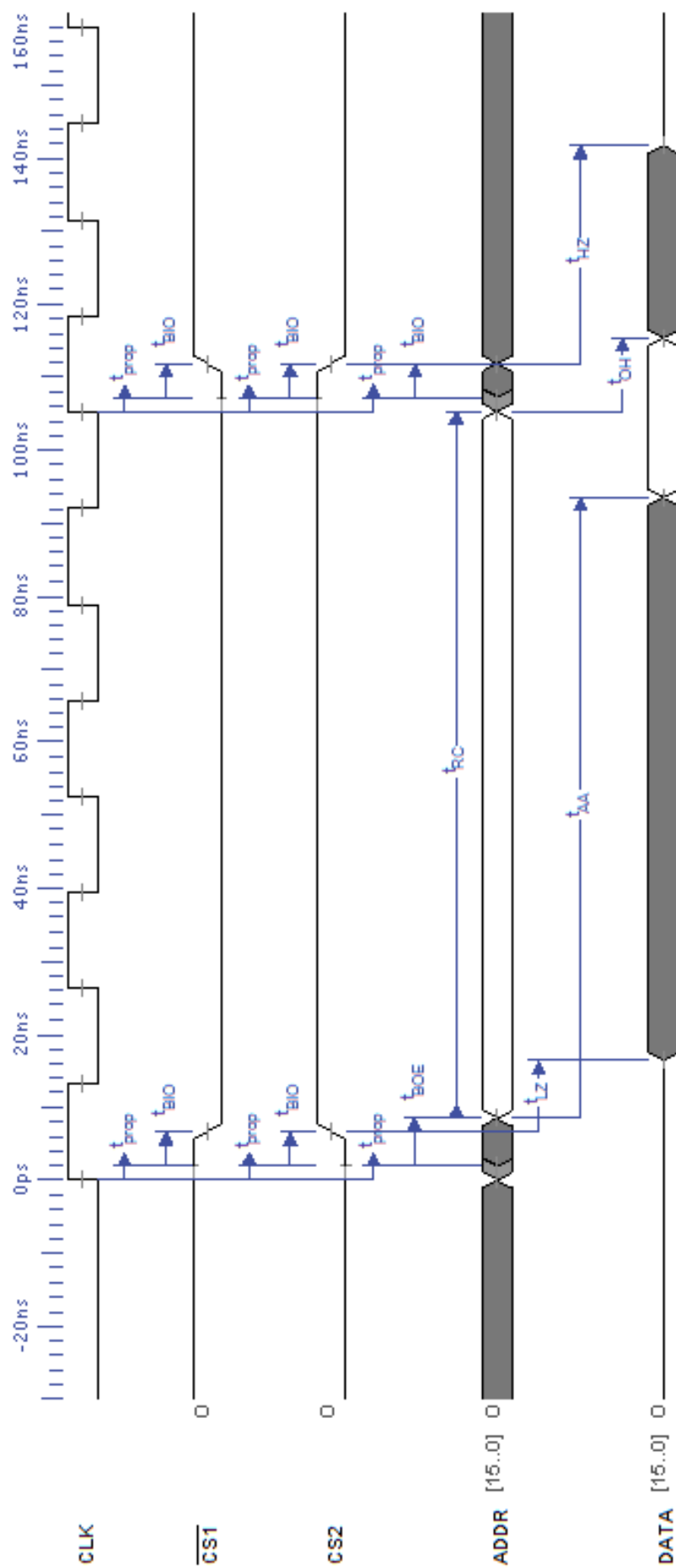


Figure 20: Timing diagram of the read cycle of the RAM device, described in Section 2.8.1.

General Data				
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX
$t_{prop}$		Propagation delay of FPGA	0ns	2ns
$t_{BIO}$		Buffer input-output delay		4.5ns
$t_{BOE}$		Buffer output enable		6.5ns
$t_{LZ}$		Chip selection to output enable		10ns
$t_{AA}$		Address access time		85ns
$t_{RC}$		Read cycle time	85ns	
$t_{HZ}$		Chip deselection to output high-Z		30ns
$t_{OH}$		Output hold from address change	10ns	

Table 7: Table of constraints of a display cycle, shown in Figure 20 and described in Section 2.8.1.

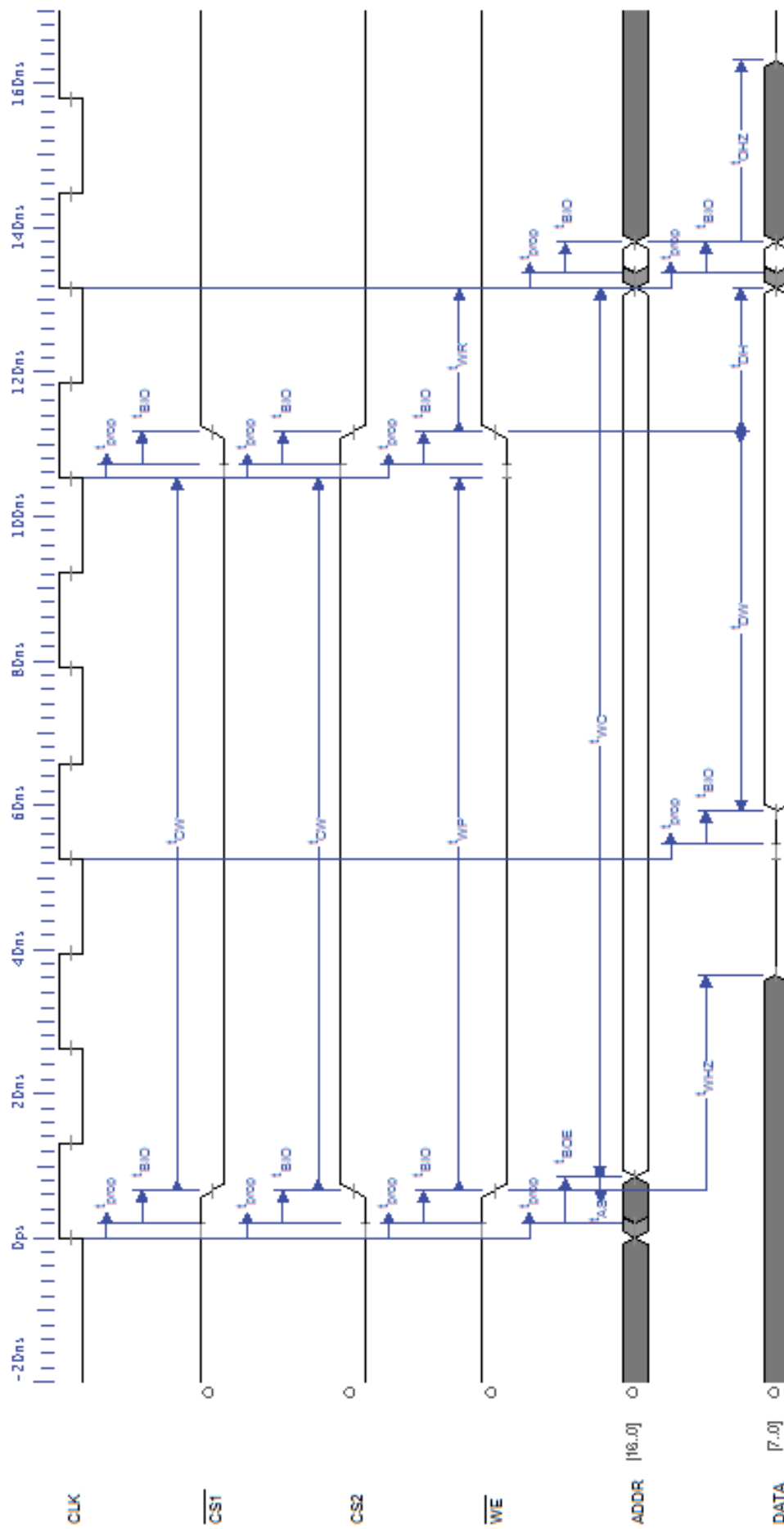


Figure 21: Timing diagram of the write cycle of the RAM device, described in Section 2.8.2.

General Data				
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX
$t_{prop}$		FPGA Propagation Delay	0	2ns
$t_{BIO}$		Buffer Input-Output Delay		4.5ns
$t_{BOE}$		Buffer Output Enable Delay		6.5ns
$t_{CW}$		Chip selection to end of write (CS1, CS2 hold time)	75ns	
$t_{WC}$		Write cycle time	85ns	
$t_{WP}$		Write pulse width ( $\overline{WE}$ hold time)	55ns	
$t_{AS}$		Address setup time	0ns	
$t_{WHZ}$		Write output to high-Z (don't drive data during this delay)		30ns
$t_{DW}$		Data to write setup time	30ns	
$t_{DH}$		Data hold from write time	0ns	
$t_{WR}$		Write recovery time	0ns	
$t_{OHZ}$		Output disable to output high-Z delay		25ns

Table 8: Table of constraints of the write cycle of the RAM device, shown in Figure 21 and described in Section 2.8.1.

### 2.8.2. ROM

A AM29F040B 512 Kword x 8-bit ROM chip, *U14*, constitutes the system's non-volatile storage. The device's connections are shown in Figure 19. The device is accessible by the processor at addresses 0x180000-0x1FFFFFF, as shown in Figure 4.

The chip is connected to the bottom 18 bits of the address bus (*A17..0*) and the bottom 8 bits of the data bus (*D7..0*), which are then routed through buffers *U8* and *U9* to *U2*, the FPGA. Note that both buses are shared between multiple memory devices.

*U14* is selected by using active-low signal *CS0* uniquely, which is then routed through buffer *U9* and into *U2*. The chip is configured as always-enabled by pulling the active-low output-enable line low through *R33*. Writing to the device is always disabled, since the device is read-only, by pulling the active-low write-enable line high through *R32*.

The interactions between the processor and the memory device, which follow a generic memory controller interaction model, are illustrated in the timing diagram of Figure 22 and Table 9. Note from the diagram that the device requires 5 wait states when reading.



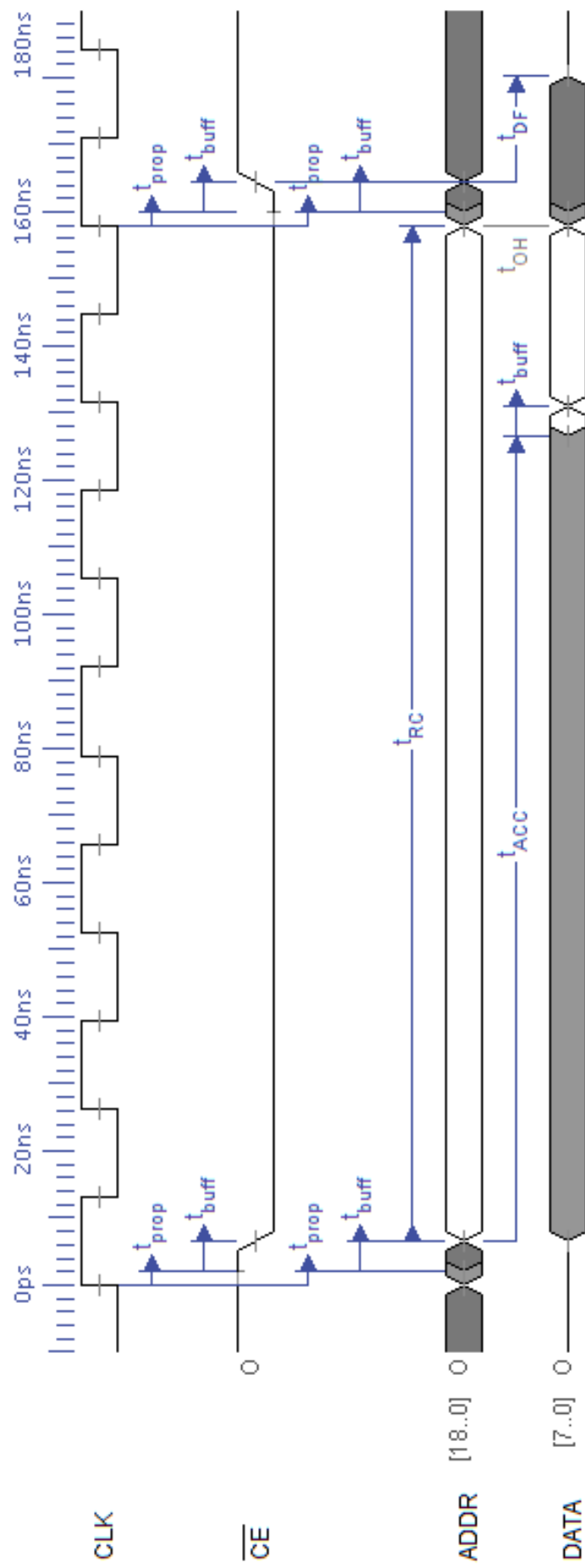


Figure 22: Timing diagram of the read cycle of the ROM device, described in Section 2.8.2.

General Data				
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX
$t_{prop}$		Propagation delay of FPGA	0ns	2ns
$t_{buff}$		Buffer output delay		4.5ns
$t_{ACC}$		Address to Data Valid delay		120ns
$t_{CE}$		Chip Enable to Data Valid delay		120ns
$t_{RC}$		Address stable hold time	120ns	
$t_{DF}$		Chip enable to output high-Z delay		16ns
$t_{OH}$		Output hold time from address/ $\overline{CE}$	0ns	0

Table 9: Table of constraints of a display cycle, shown in Figure 22 and described in Section 2.8.2.

### 2.8.3. VRAM

Two MSM5416283 512 Kword x 16-bit VRAM chips, *U20* and *U21*, are connected “in parallel” to form a single virtual 512 Kword x 32-bit memory device. The devices’ connections are illustrated in Figure 23. The device is accessible by the processor at addresses 0x00000-0xFFFFF, as shown in Figure 4.

In this configuration, the whole video address bus (*VA8..0*) is shared between the two devices, and bridged through buffer *U1* into the FPGA, *U2*. The data bus, on the other hand, is split: *U21* is connected to the bottom 16 bits of the data bus (*U15..0*), while *U20* is connected to the top 8 bits (*U23..16*). Note that eight of the data lines at *U20* are left unconnected, since only 24 bits of data are used. The data bus is shared with other memory devices, and relayed by buffers *U6* and *U9* into *U2*, the FPGA.

The serial outputs of both memory devices are output to the display on connector *J6*. The bottom byte of *U21* (*SDQ7..0*) is used as the red channel in the display (*R7..0*); the top byte of *U21* (*SDQ15..8*) is used as the green channel in the display (*G7..0*); the top byte in *U20* (*SDQ15..8*) is the blue channel in the display (*B7..0*).

With the parallel configuration of the two chips, *U20* and *U21* both share the same signals for *SCLK*, *TRG*, *CAS*, *RAS*, and the combination of *WEL* and *WEU*, *WEL/U*. These signals are routed through buffer *U1* into the FPGA, *U2*.

On both chips, *SOE* is tied low through *R68* and *R69* to permanently enable the serial interface of the devices. *DSF* is tied low through *R70* and *R71* to disable special functions. *QSF* is unused, and thus left floating.

The interactions between the processor and the VRAM are mediated by the VRAM controller, and are thus described in Section 2.2.5. As far as the processor is concerned, it can access the device following a generic memory controller interaction model with variable wait states and wait signal *!WAIT*.

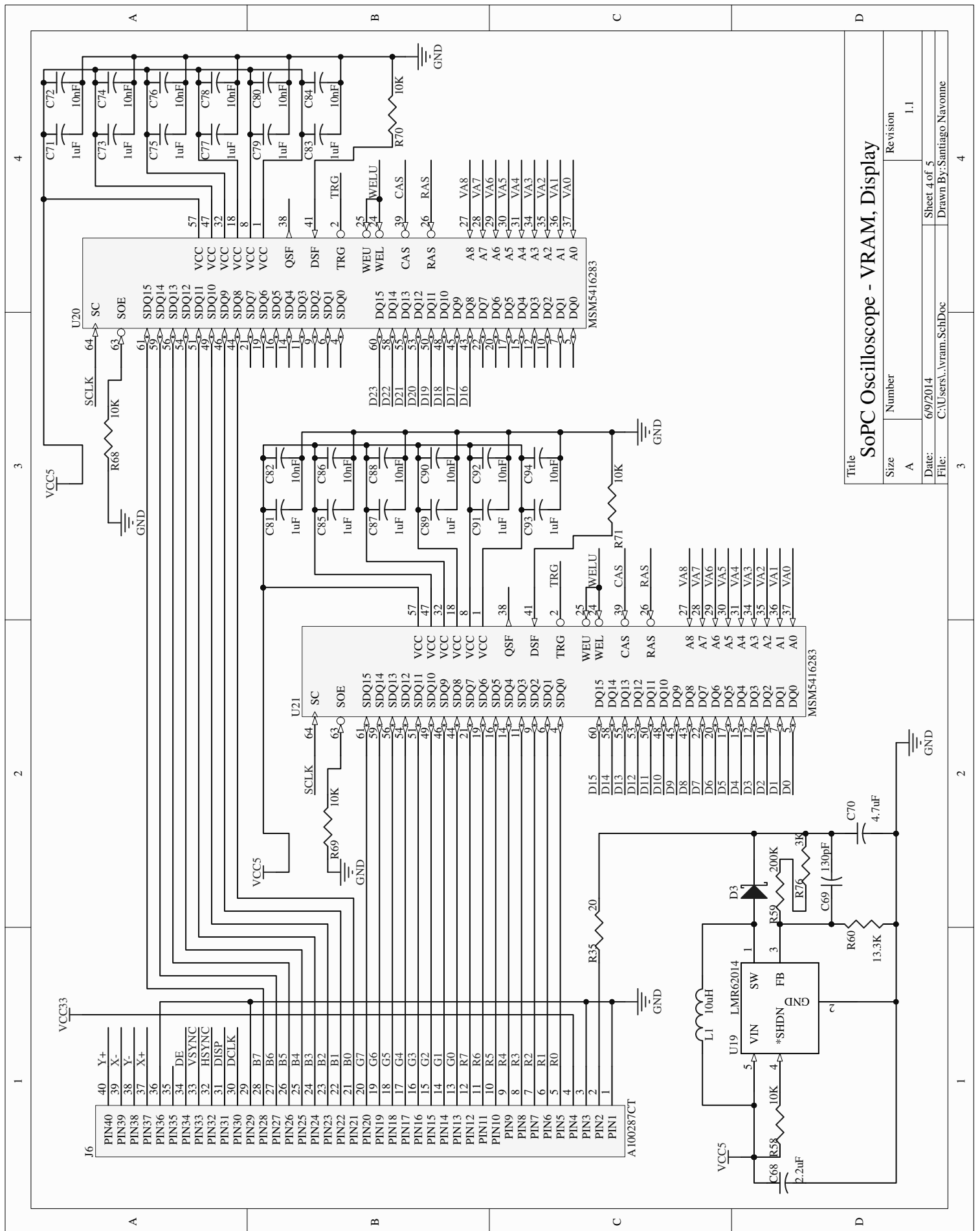


Figure 23: Schematic of the VRAM memory devices and display interface. Also shown is the section of the power supply used to power the display's LEDs. Further details are provided in Section 2.8.3.

## 2.9. Display

A NHD-4.3-480272-EF-ATXL#-T 4.3 inch, 480x270 pixel, color, touch screen display is used in the system. The display uses a HX8257 LCD driver, which is controlled by the display controller described in Section 2.2.6.

The display is connected through Molex connector *J6*, as shown in Figure 23. The 24 data lines for the three colors output by the VRAM memory devices (*R0..8*, *G0..8*, *B0..8*) are output on pins 5 through 28. The display control signals output by the display controller (*DCLK*, *DISP*, *HSYNC*, *VSYNC*), routed through buffers *U1* and *U7*, are output on pins 30 through 33. Pin 43 is used for the *DE* signal, which is unused and therefore left floating.

Pins 37 through 40 are connected to the display's touch screen, and are therefore used by the touch screen controller, *U18*. Pin 2 is used to drive the display backlight's LEDs, and is therefore tied to the +20 V power supply through a current-limiting 20  $\Omega$  resistor.

The display connector is highlighted in pink in Figure 2.

## 2.10. Touch Screen Controller

The display's touch screen is interfaced with using a TSC2003 touch screen controller. The connections of this device are shown in Figure 24. Note that the device remains unimplemented in the system's software, and therefore its hardware is not fully tested.

The *X+*, *Y+*, *X-*, *Y-* lines are connected to the corresponding pins on the display connector, *J6*. The I<sup>2</sup>C bus lines, *SCL* and *SDA*, are connected directly to the FPGA and into a currently unimplemented I<sup>2</sup>C controller. Lines *A1,0* are tied low to configure the address of the device on the I<sup>2</sup>C bus. *\*PENIRQ* goes low when the screen is touched, signaling a touch screen event to the processor; this line is therefore relayed by buffer *U7* into the FPGA, *U2*, and then made accessible to the processor as PIO.

The full functionality of the chip is not used, and therefore monitoring pins *VBAT1,2* and *IN1,2* are simply tied low through resistors *R50,51* and *R46,47*.

The touch screen controller is highlighted in cyan in Figure 2. Note that all bypass capacitors are placed on the backside of the board, drawn in Figure 3.

## 2.11. Rotary Encoders

Two rotary encoders with temporary push-buttons are used to provide the main user input interface for the system. The devices' connections are illustrated in Figure 24. Once debounced and decoded within the FPGA, their signals are made available to the NIOS processor on the *PIO\_0* interface, at addresses 0x2410A0-0x2410BF.

Each one of the signals (*ROT0A,B*; *ROT1A,B*; *SW0,1*) is pulled high through resistors *R52-57*. As the rotary encoders are turned, the rotation signals are shorted to the *COM* line, which is tied to ground. Similarly, as either push-button is pressed, the *SW* lines are shorted to ground. The signals are then debounced and decoded as described in Sections 2.2.3 and 2.2.4, making user input available to the processor.

The rotary encoders are highlighted in brown in Figure 2. Note that the pull-up resistors are placed on the backside of the board, drawn in Figure 3.

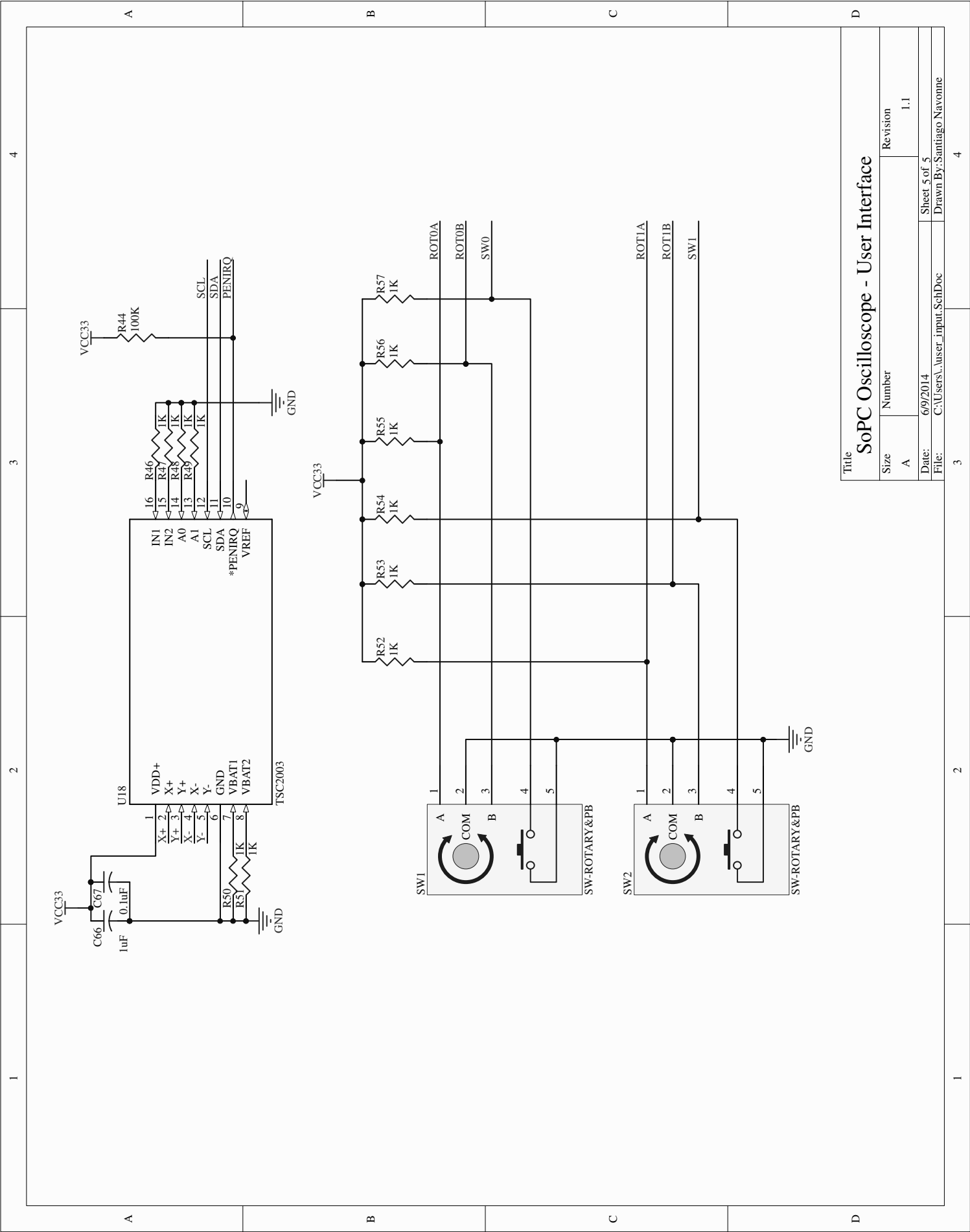


Figure 24: Schematic of the user input section of the system. This includes the rotary encoders, described in Section 2.11, and the touch screen controller, described in Section 2.10.

## 2.12. Analog Interface

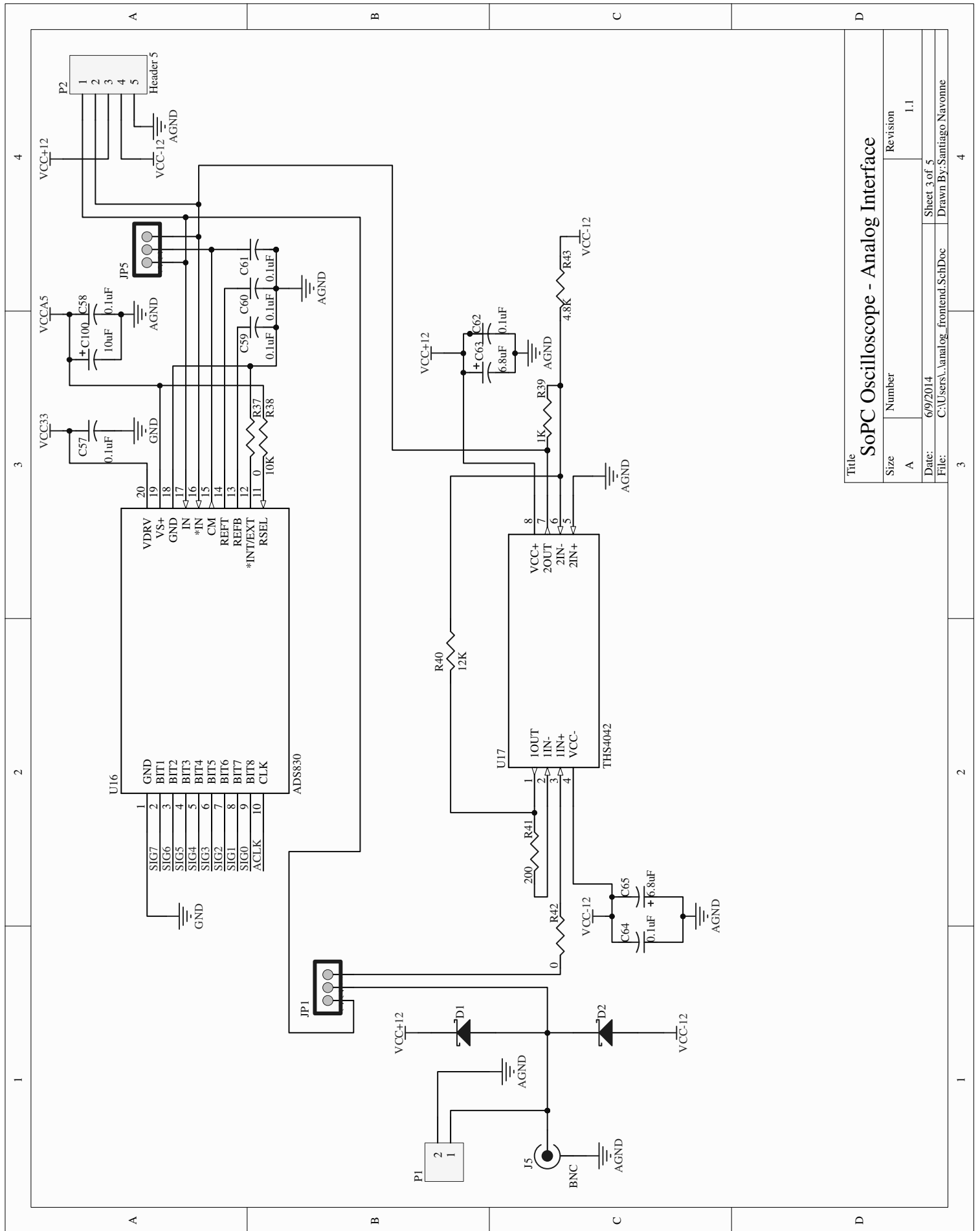
Analog samples are acquired from the probe through the analog interface here described, and then made available to the processor via the triggering mechanism described in Section 2.2.2. The analog interface schematic is illustrated in Figure 25.

The signal is acquired through the probe connected to BNC connector *J5*. An alternative connector is provided through two-pin header *P1*. The positive line is then connected to +12 V and -12 V through Schottky diodes, ensuring that no voltages outside that range will ever reach components forward of this point. Note that this functionality remains untested.

*JP1* provides an easy means of selecting whether to scale and shift the signal using the Analog Front-End (Section 2.12.1), or skip the section altogether and input the signal directly to the Analog-to-Digital Converter (Section 2.12.2). For the correct operation of the system, *JP1* should be configured to the FE position.

The analog interface is highlighted in orange in Figure 2. Note that all bypass capacitors are placed on the backside of the board, drawn in Figure 3. Ground planes are placed below the region in order to reduce noise.





Title		SoPC Oscilloscope - Analog Interface	
Size	Number	Revision	
A		1.1	
Date:	6/9/2014	Sheet 3 of 5	
File:	C:\Users\Navonne\Documents\frontend.SchDoc	Drawn By: Santiago Navonne	

Figure 25: Schematic of the system's analog interface, described in Section 2.12. This includes the analog front-end and the analog-to-digital converter (ADC).

### 2.12.1. Analog Front-End

The system incorporates an analog front-end that scales and shifts the signal to allow for an increased voltage range. Thanks to this section of the circuit, the system is able to accept signals from -10 V to +10 V. These voltages are thus scaled to the Analog-to-Digital Converter's (ADC) voltage swing ( $\pm 1$  V), and shifted to its common mode voltage (+2.5 V).

To this end, the operational amplifier circuit of Figure 26 is used. A THS4042 165 MHz, dual op-amp chip was used. The first stage in this circuit, which corresponds to pins *1IN+*, *1IN-*, and *1OUT* at *U17*, and resistor *R41* in Figure 25, is a simply buffer, used to provide high-impedance to the input circuit: it is vital that the oscilloscope do not disturb the circuit being measured. The second stage, made up of pins *2IN-*, *2IN+*, and *2OUT* at *U17*, and resistors *R40*, *R39*, and *R43*, is a shifting-scaling stage. This part of the circuit adds one twelfth of the input signal (scaling it down from  $\pm 12$  V to  $\pm 1$  V) to  $-12\text{V}/4.8 = -2.5\text{V}$  (shifting it down to -2.5 V CM), and inverts the result. The output is the mirror image of the input signal ("negative" the signal), scaled down to one twelfth, and shifted to +2.5 V CM. This is exactly the input required by the inverted signal input of the ADC.

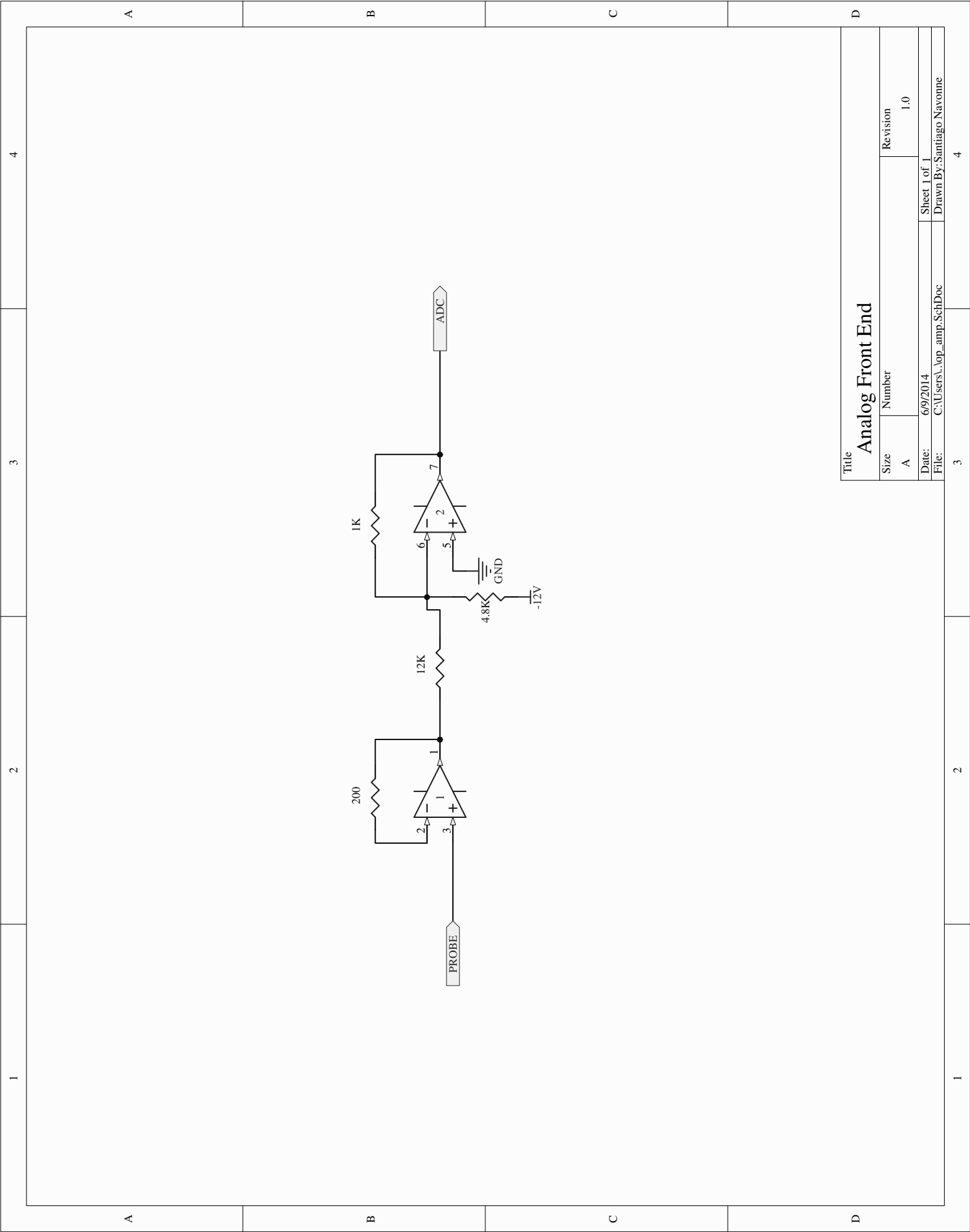


Figure 26: Symbol schematic of the operational amplifier scaling and shifting circuit used in the system’s analog front-end, and described in Section 2.12.1.

### 2.12.2. Analog to Digital Converter

The analog to digital converter takes either the input signal or the scaled output of the op-amp circuit of Section 2.12.1, and converts it to a digital value. The digital value is output on lines *SIG7..0*, which are then bridged through buffer *U3* into the FPGA, *U2*. The digital conversion is clocked by *ACLK*, which is output by *U2* through buffer *U7*. The timing of a ADC clocking interaction is shown in Figure 27 and Table 10.

The input signal is placed on either the regular input pin *IN* or the inverted input line *\*IN*. The other pin must be tied to the common mode pin *CM* using jumper *JP5*. For the correct operation of the system, *JP5* must be configured to the FE position. A 5-pin header, *P2*, is provided to allow for the substitution of the analog front-end with an alternative circuit. The *\*INT/EXT* configuration line is tied low to select the internal reference, and *RSEL* is tied high through *R38*.

Note that the digital side of the chip is placed outside of the analog region on the PCB of Figure 2, and that the device is therefore placed at the edge of the analog region. The analog ground is separated from the digital ground.

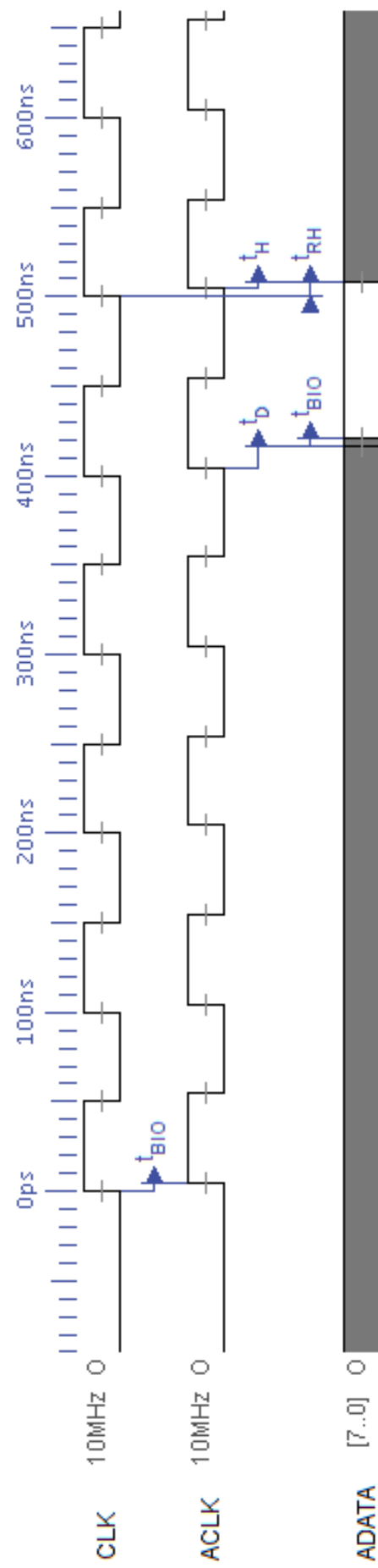


Figure 27: Timing diagram of an ADC sampling clock, described in Section 2.12.2.

General Data					
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOTES
$t_D$		New data delay after fourth clock		12ns	
$t_H$		Data hold after fifth clock	3.9ns		
$t_{BIO}$		Buffer Input/Output delay		4.5ns	
$t_{RH}$		Read cycle hold requirement	0ns		

Table 10: Table of constraint for the ADC sampling clock cycle, shown in Figure 27 and described in Section 2.12.2.

### 2.13. Revision History

Table 11 provides a summary of the revisions made to the original design.

Date	Revision	Document(s)	Changes
March 2014	1.0	All	Initial revision.
June 2014	1.1	User Input Schematic	Changed rotary encoder pull-up resistors to 1k $\Omega$ .
			Added pull-up resistor on PENIRQ line.
		FPGA Schematic	Corrected JTAG connector wiring.
			Fixed error in routing of A0.
		PCB	Changed voltage regulators' footprints.
		Analog Inter-face Schematic	Changed value of pull-down resistor at ADC.

Table 11: Revision history.

## 3. Software

This section describes how the system's software works, from the system overview to the detailed description of each element. The roles and interactions of the various part of the program are described. The actual code for the program is provided in Appendix B.

### 3.1. System Overview

The core of the system's software constitutes in the EE/CS 52 SoPC Oscilloscope software library, written in C, with minor modifications. The hardware interface procedures are written in NIOS assembly, and are specific to the hardware used in the system. The block and file structure of the project is shown in Figure 28.

When the system is started, the stack is automatically set up with other initialization, and *main* is run, within *mainloop.c*. This function, performs all required initialization and runs the system. The main loop makes use of several procedures: a set of user interface procedures found in files *lcdout.c*, *keyproc.c*, *menu.c*, and *menuact.c* allow the system to interact with users through the UI; trace utility procedures in *tracutil.c* process acquired signals; key handling procedures in *keys.s* interface with the software to identify key presses and user actions; display controlling procedures in *display.s* provide a layer of abstraction for the specific LCD used; and analog interface procedures in *trigger.s* control the analog and triggering interface as needed.

Note that each file has a header file of the same name, but *.h* extension, associated with it, where functions are declared and constants defined. Additionally, *scopedefs.h* provides general project constants for C files, while *general.h* provides general constants for assembly files. *interfac.h* defines constants specific to the system's hardware interface. Finally, *system.h* contains memory map and IRQ number definitions, and is automatically generated by the Altera toolchain. The used version is also provided in Appendix B.

### 3.2. Initialization and Main Loop

When the system is started as after general initialization is performed by NIOS toolchain functions, *main* is called within *mainloop.c*, part of the Oscilloscope library. This function performs all additional initialization and runs the system. The procedure was modified to initialize the keys handler and the analog/triggering interface on start. For more details on the procedures within *mainloop.c*, please refer to the EE/CS 52 Oscilloscope library documentation.

### 3.3. User Interface

In order to interface with users, a number of functions are provided in the Oscilloscope library. *lcdout.c* provides procedures used to output data to the LCD; *keyproc.c* has functions that process the various available keys; *menu.c* contains the functions for processing menu entries; *menuact.c* includes the functions for carrying out menu actions. All of these functions utilize the hardware abstraction procedures provided in the assembly files. *lcdout.c* was modified to support user interface colors, and highlighted characters (used in the menu); *menuact.c* was modified to support a faster sweep rate, and to change the sweep rate values to the actual values obtained by dividing the system clock. For more details on the procedures within the user interface files, please refer to the EE/CS 52 Oscilloscope library documentation.



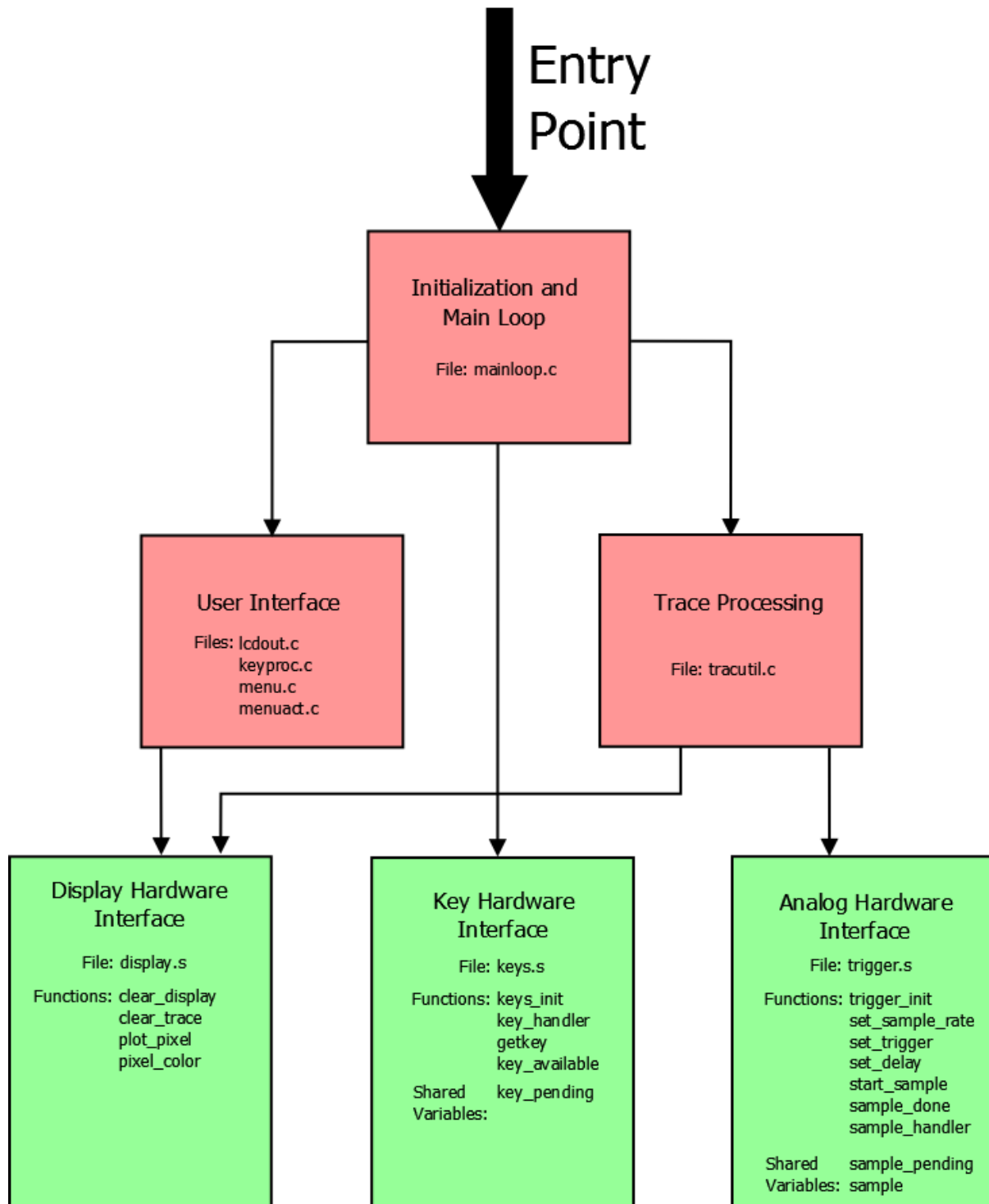


Figure 28: Block diagram of the SoPC Oscilloscope system's software. Blocks in red are part of the EE/CS 52 Oscilloscope library; blocks in green are hardware-specific and implemented for this system. The diagram is described in Section 3.1.

### 3.4. Trace Processing

In order to draw traces to the screen, capturing and outputting the acquired data, procedures within *tracutil.c*, provided by the Oscilloscope library, are used. This library file was modified to change the UI display colors (trace), and clear only the trace instead of the whole display when refreshing the sample, obtaining a better behavior of the display. These procedures employ analog and triggering hardware assembly procedures to perform the required actions. For more details on the procedures within *tracutil.c*, please refer to the EE/CS 52 Oscilloscope library documentation.

### 3.5. Key Hardware Interface

To interface with the system's push-buttons and rotary encoders, abstracted as keys, a set of procedures are provided in *keys.s*. These procedures translate user input actions into key values as needed, making them available to the system's software. The procedures use shared variable *curr\_key* to store a pending key press, if any is available.

*keys\_init* performs all the necessary initialization for the keys interface. It initializes shared variable *curr\_key* and sets up interrupts as necessary, preparing the interface for use.

*key\_handler* is executed whenever a key press interrupts occurs after initialization. The function identifies the key pressed, and saves it in buffer *curr\_key* to make it available to *getkey* and *key\_available*, and thus outside functions.

*getkey* returns the currently pending key press if one is available (i.e. present in the *curr\_key* buffer). If none is available, the function blocks in a busy loop.

Finally, *key\_available* checks whether a key press is pending (i.e. there's a valid value in *curr\_key*). This procedure is normally called before *getkey* to avoid blocking.

For more details about the operation of these procedures, please refer to their definitions in *keys.s*.

### 3.6. Display Hardware Interface

To interface with the system's color display, a set of procedures are provided in *display.s*. These procedures communicate with the VRAM device as needed to control the display as requested. Note that the display requires no formal initialization; however, *clear\_display* should be called at the start, since the image initially displayed is undefined.

*clear\_display* completely clears the display, making every pixel in it black. This function should be called after initialization, since the image initially shown on the display is undefined.

*clear\_trace* only clears the trace pixels on the display, that is pixels that are the color of the trace or of the cursor; the procedure turns these pixels black. The function is currently unused, but is still provided as it can simplify the implementation of additional features, such as a cursor.

*plot\_pixel* changes the color of one pixel at a given location to an RGB value.

Finally, *pixel\_color* accesses the color, as an RGB value, currently being displayed at a given location. This procedure is currently unused, but is still provided as it can simplify the implementation of additional features, such as a cursor.

For more details about the operation of these procedures, please refer to their definitions in *keys.s*.

### 3.7. Analog Hardware Interface

To control the analog and triggering interface, starting and acquiring samples and configuring the trigger parameters, a set of procedures are provided in *trigger.s*. These procedures output the necessary signals to the triggering logic, causing it to acquire samples as needed. Additionally, they transfer samples to the main code when they are requested after completion. The procedures use shared variable *sample\_pending* to keep track of the currently started sample, as well as buffer *sample* to extract the samples from the FIFO and return them to the caller.

*trigger\_init* performs all necessary initialization for the analog and triggering interface. it initializes shared variable *sample\_pending*, sets up interrupts as needed, and resets the triggering logic, preparing the interface for use. The triggering logic is also reset.

*set\_sample\_rate* allows the caller to configure the sampling rate to any positive number of samples per second less than or equal to the system clock divided by two (19.5 Msamples per second). The frequency is translated to a number of system clocks per sample, and the value is then output to the triggering hardware block. The triggering logic is then reset. The number of samples that will be acquired at the configured rate is then returned, but note that this value is always the same, and equal to the width of the display (which must be less than or equal to 512 per hardware limitations).

*set\_trigger* configures the trigger level and slope. The trigger level is translated to the corresponding level in the correct range, and then its value is output together with the slope to the triggering hardware block. The triggering logic is then reset.

*set\_delay* configures the trigger delay to any positive, unsigned 32-bit number of samples less than  $2^{32} - 2$ . The value is corrected to take into account any hardware limitations (delay must be positive), and the value is then output to the triggering hardware block. The triggering logic is then reset.

*start\_sample* starts a new sample with the previously configured settings. The sample can trigger automatically or manually, as configured via the procedure's argument. The sample is thus started in hardware.

*sample\_done* checks whether the previously started data sample has been completed and is thus available by reading shared variable *sample\_pending*. If the sample has been completed, it is at this point extracted from the FIFO and returned in a buffer, while variable *sample\_pending* is reset. If no sample is available, a null pointer is provided.

Finally, *sample\_handler* is executed any time the FIFO finished being filled. The procedure simply updates shared variable *sample\_pending* to indicate that there is a sample that can be extracted by *sample\_done*.

# Appendices

## A. Original Documents

Figures 29, 30, 31, 32, 33 show the schematic of the original design; Figures 34, 35 show the resulting PCB used in the prototype. Note that the RAM and ROM section is identical to the one described in Sections 2.8.1 and 2.8.2. The changes made and their reasons are summarized below.

After noticing that the footprints used for the voltage regulators (*U11-13*) in revision 1.0 were incorrect (power and ground pins were switched), these were corrected in revision 1.1. The bottom bit of the address bus (*A0*) but was incorrectly routed through a buffer direction pin; the line was thus re-routed in the newest revision. There was a mistake in the wiring of the JTAG connector (*J1*), where pins 9 and 10 were switched.

Additionally, the values of some resistors had to be changed after noticing too big voltage drops across them: the pull-up resistors at the rotary encoders (*R52-57*) had to be decreased from  $10\text{k}\Omega$  to  $1\text{k}\Omega$ , and the pull-down resistor at the ADC's *\*INT/EXT* pin (*R37*) had to be shorted.

Finally, a missing pull-up resistor had to be added to the touch screen controller's PENIRQ line in order to keep it from floating when not active (*R44*).

Figure 29: Original revision of the FPGA and related components schematic.

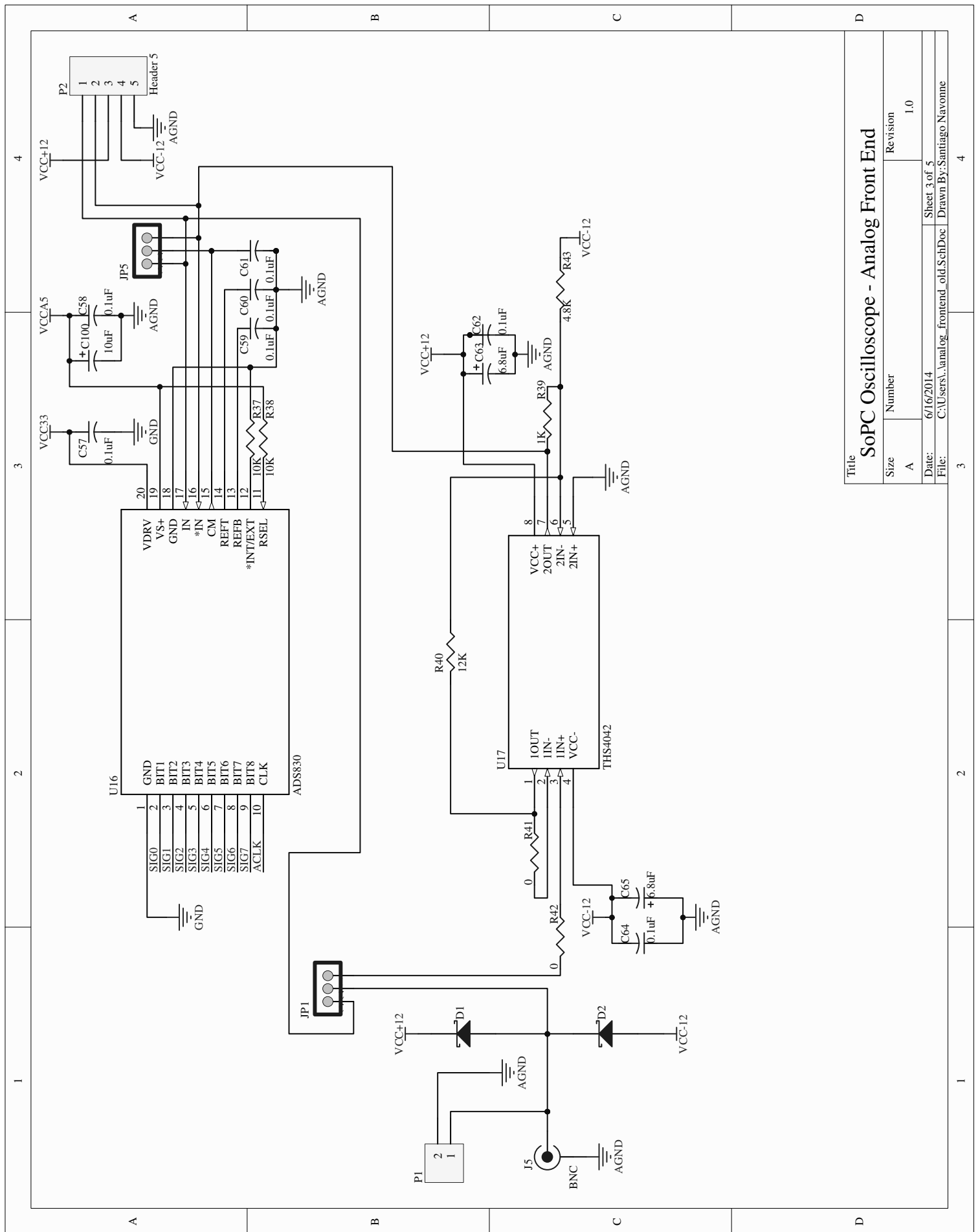


Figure 30: Original revision of the analog interface schematic.

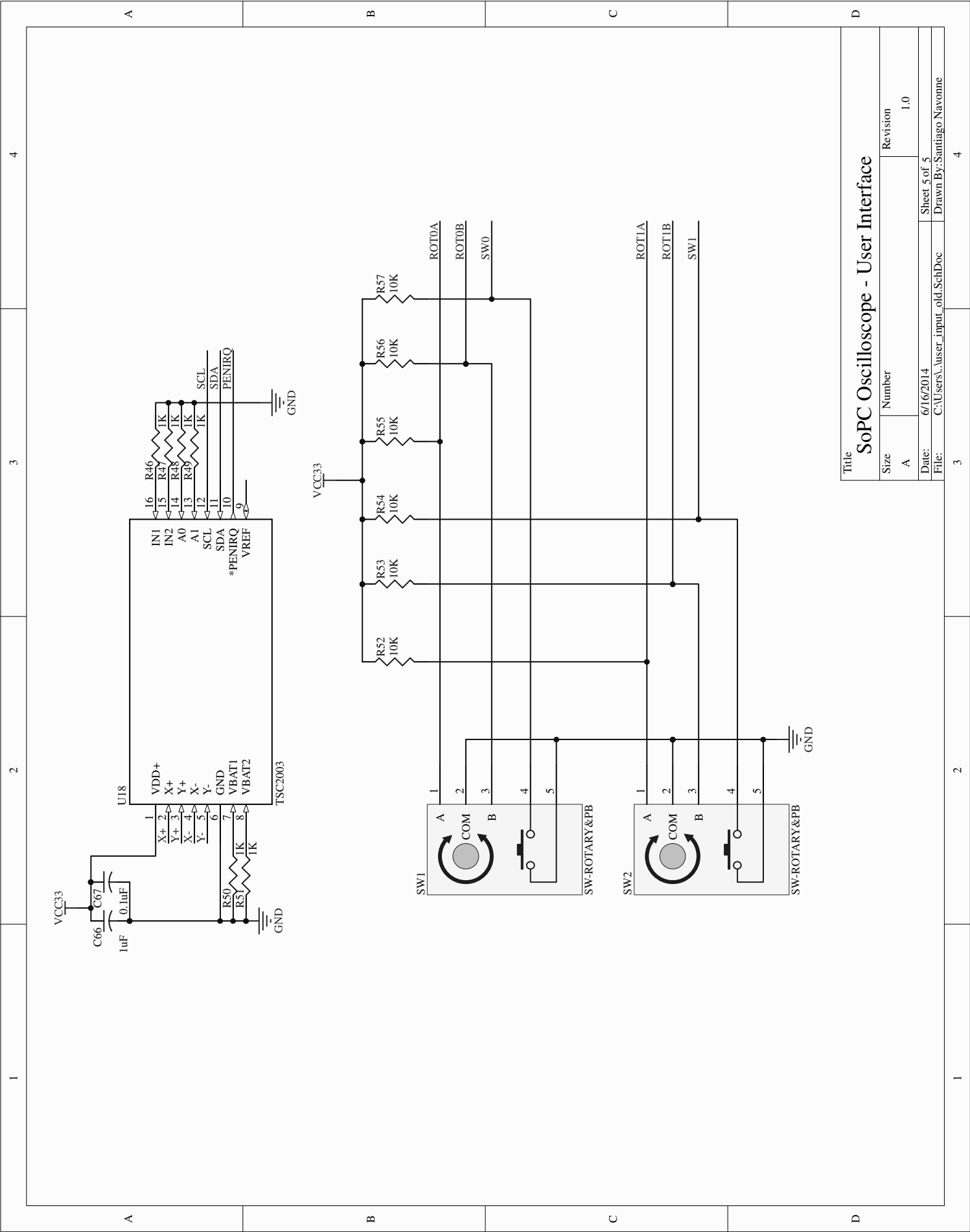
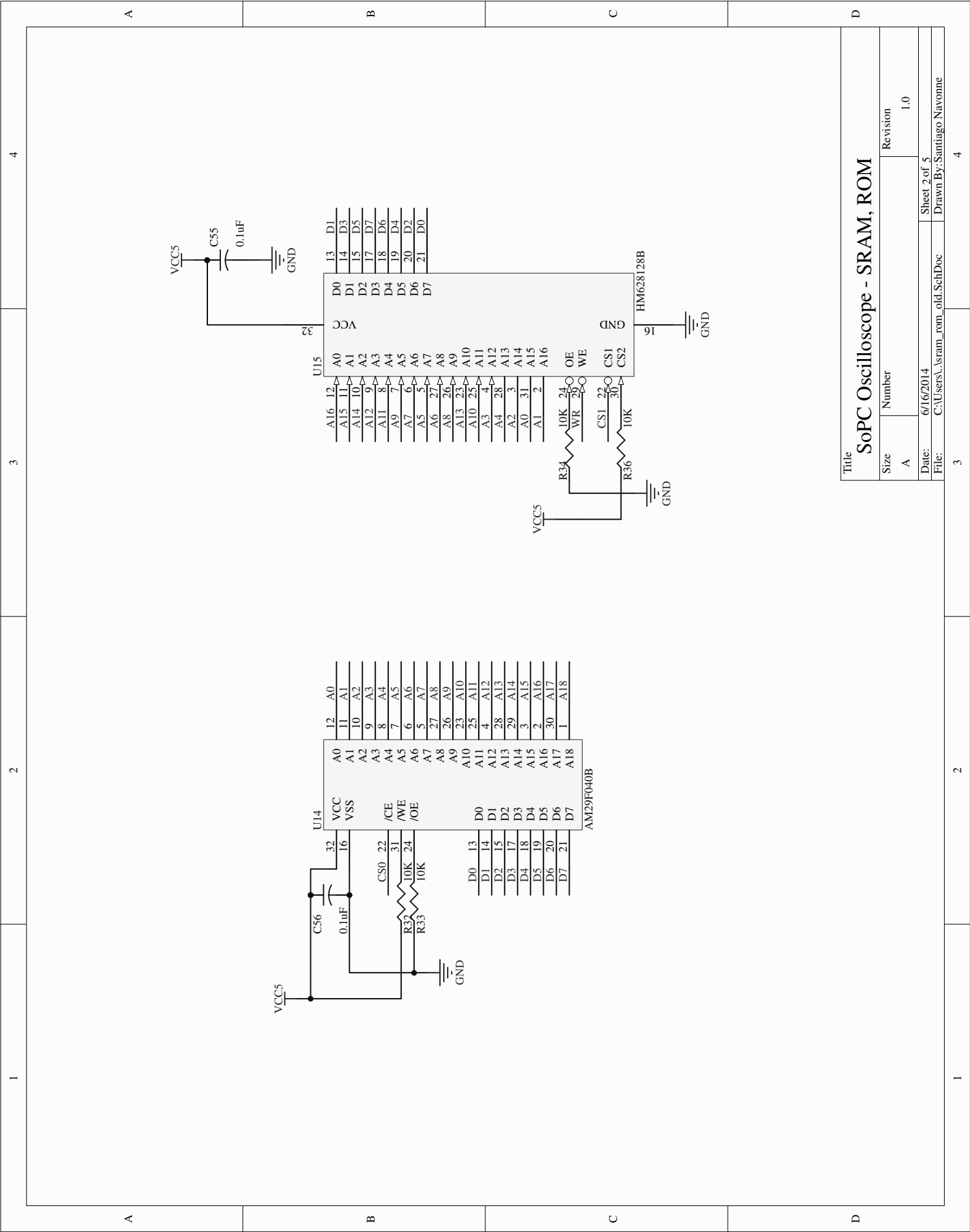


Figure 31: Original revision of the user input components schematic.





Title			SoPC Oscilloscope - SRAM, ROM	
Size	Number	Revision		1.0
A				
Date:	6/16/2014	Sheet 2 of 5		
File:	C:\Users\Navonne\Documents\SoPC Oscilloscope - SRAM, ROM	Drawn By: Santiago Navonne		

Figure 32: Original revision of the SRAM and ROM devices schematic.



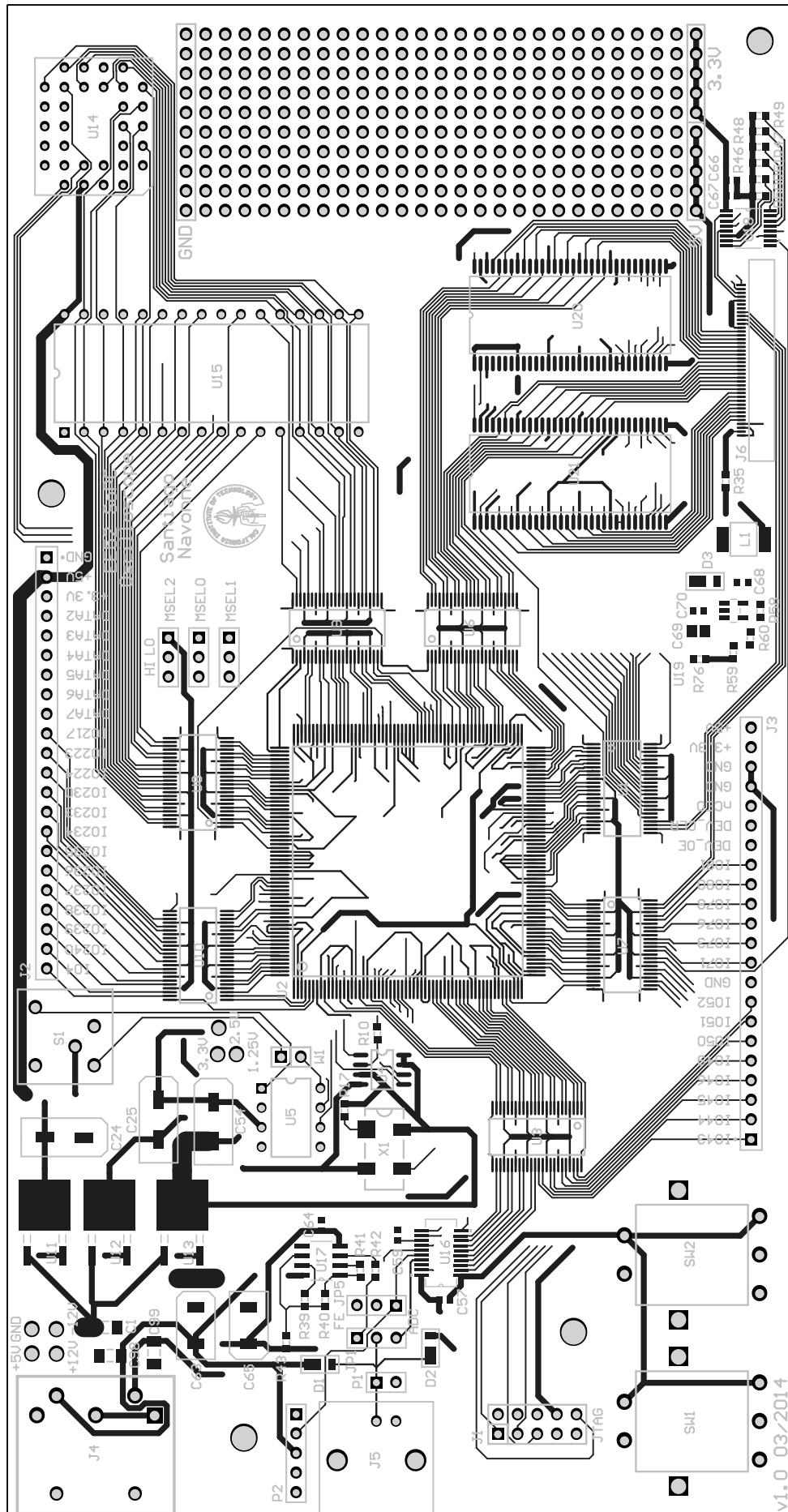


Figure 34: Front side of the original revision of the system's Printed Circuit Board (PCB).

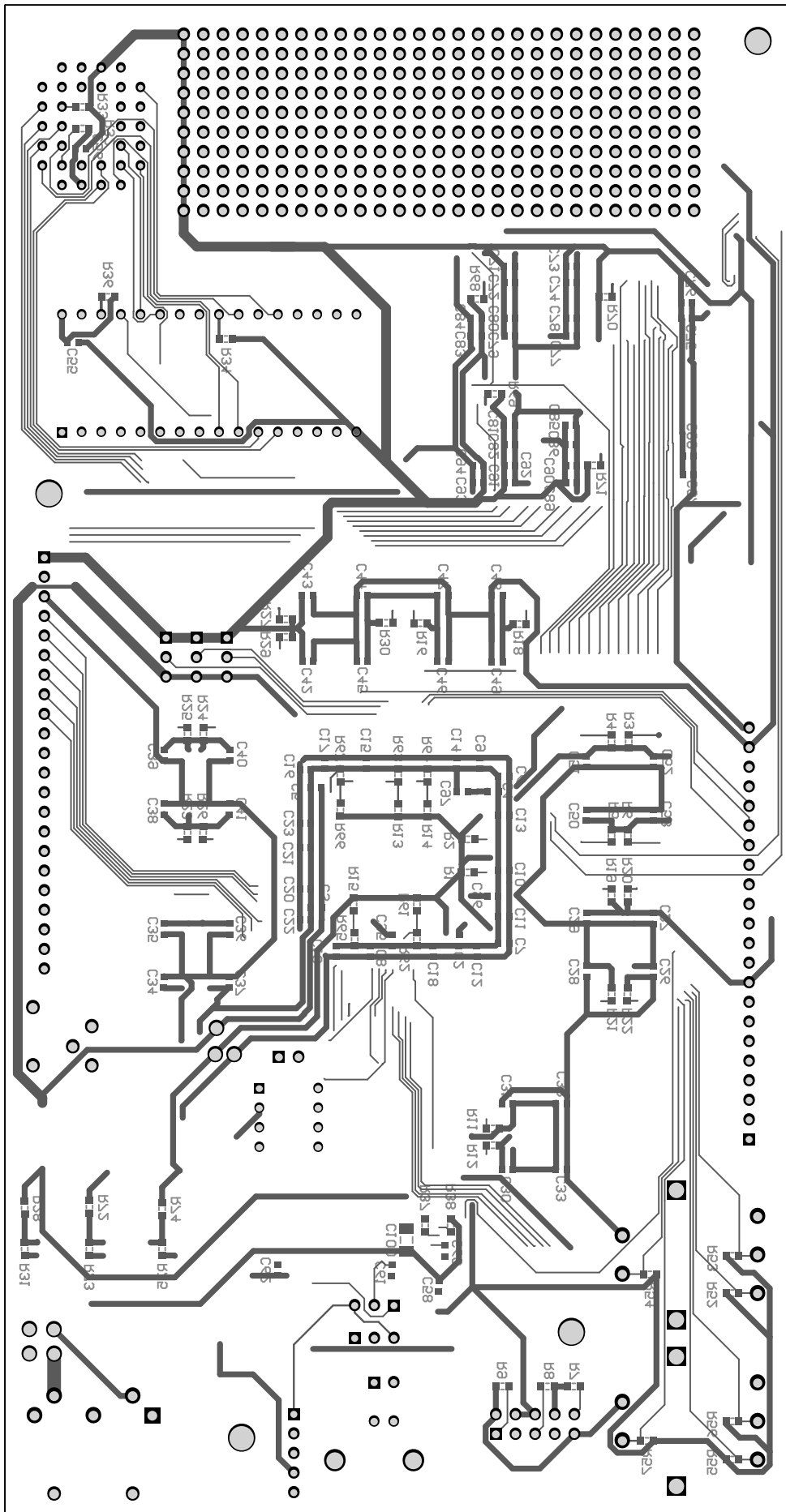


Figure 35: Back side of the original revision of the system's Printed Circuit Board (PCB).

## **B. Software Code**

In this appendix, all the code contained within the program's software is provided. Table 12 shows a quick overview of the various files and their contents for quick reference, in alphabetical order.

display.s	Display hardware interface and control routines.	95
display.h	Display hardware interface constants.	99
general.h	General project constants for assembly files.	100
interfac.h	Hardware interface constants for library functions.	101
keyproc.c	Key processing functions.	102
keyproc.h	Key processing function prototypes and constants.	107
keys.s	Key and rotary encoder hardware interface and control routines.	108
keys.h	Key and rotary encoder hardware interface constants.	113
lcdout.c	LCD output functions.	114
lcdout.h	LCD output function prototypes and constants.	120
mainloop.c	Main processing loop and initialization.	121
menu.c	Menu entry processing functions.	125
menu.h	Menu entry processing function prototypes and constants.	135
menuact.c	Menu action execution functions.	137
menuact.h	Menu action execution function prototypes and constants.	160
scopedef.h	General project definitions for C files.	162
tracutil.c	Trace handling utility functions.	164
tracutil.h	Trace handling utility function prototypes and constants.	179
trigger.s	Data sampling and triggering hardware interface and control routines.	181
trigger.h	Data sampling and triggering hardware interface constants.	190
system.h	SOPC Builder system and BSP software package information.	191

Table 12: Table of Contents for system's software code.

```

1  /*****
2  /*
3  /*          DISPLAY.S          */
4  /*          Display Interface Functions          */
5  /*          Digital Oscilloscope Project          */
6  /*          EE/CS 52          */
7  /*          Santiago Navonne          */
8  /*          */
9  /*****/
10
11 /*
12  Display interface and control routines for the EE/CS 52 Digital Oscilloscope
13  project. Function definitions are included in this file, and are laid out
14  as follows:
15  - clear_display: Completely clears the display;
16  - clear_trace: Clears the pixels on the display that are the color of the
17  trace;
18  - plot_pixel: Changes the color of the pixel at a given location;
19  - pixel_color: Accesses the color of the pixel currently being displayed at
20  a given location.
21
22
23  Revision History:
24  6/3/14 Santiago Navonne Initial revision.
25  */
26
27 #include "general.h"
28 #include "system.h"
29 #include "interfac.h"
30 #include "display.h"
31
32
33 .section .text /* Code starts here */
34
35
36 /*
37  * clear_display
38  *
39  * Description: This procedure clears the display, setting the color of every
40  * pixel to black immediately.
41  *
42  * Operation: The procedure loops through every pixel in the display-mapped
43  * region of the VRAM, storing 0 (black; clear pixel) into every
44  * location.
45  *
46  * Arguments: None.
47  *
48  * Return Value: None.
49  *
50  * Local Variables: None.
51  *
52  * Shared Variables: None.
53  *
54  * Global Variables: None.
55  *
56  * Input: None.
57  *
58  * Output: Clears every pixel on the display (changes color to black).
59  *
60  * Error Handling: None.
61  *
62  * Limitations: None.
63  *
64  * Algorithms: None.
65  * Data Structures: None.
66  *
67  * Registers Changed: r8, r9, r10, r11, r12.
68  *
69  * Revision History:
70  * 6/03/14 Santiago Navonne Initial revision.
71  *
72  */
73 .global clear_display
74 clear_display: /* clear the whole display */
75     MOVHI r8, %hi(VRAM_BASE) /* start at base of VRAM */

```

```

76     ORI      r8, r8, %lo(VRAM_BASE)
77     MOVI     r9, SIZE_X           /* and will loop through all columns */
78     MOVI     r10, SIZE_Y          /* and rows */
79     MOV      r11, r0              /* starting at coordinates (0, 0) */
80     MOV      r12, r0              /* (top left corner) */
81
82 row_loop:                                /* go through an entire row */
83     STWIO     r0, (r8)             /* first clear the current pixel */
84     ADDI      r8, r8, WORD_SIZE    /* then go to next column */
85     ADDI      r11, r11, 1          /* also incrementing the index */
86     BLT       r11, r9, row_loop    /* and if we're still within display, repeat */
87
88 next_row:                                /* move to next row */
89     ADDI      r8, r8, REMAINDER     /* add the remainder to finish up a VRAM row */
90     MOV       r11, r0              /* reset the column index */
91     ADDI      r12, r12, 1          /* and increment the row index */
92     BLT       r12, r10, row_loop   /* if we're still within display, repeat */
93
94     RET                               /* all done, so return */
95
96 /*
97 * clear_trace
98 *
99 * Description:          This procedure clears the trace from the display, changing the
100 *                      color of every pixel that is currently the trace or cursor color
101 *                      to black.
102 *
103 * Operation:           The procedure loops through every pixel in the display-mapped
104 *                      region of the VRAM. For every location, if the current value
105 *                      matches either trace or cursor colors (both part of the trace)
106 *                      the pixel is cleared by storing 0 into that memory location.
107 *
108 * Arguments:           None.
109 *
110 * Return Value:        None.
111 *
112 * Local Variables:    None.
113 *
114 * Shared Variables:    None.
115 *
116 * Global Variables:    None.
117 *
118 * Input:               None.
119 *
120 * Output:              Clears every trace pixel on the display (sets color to black).
121 *
122 * Error Handling:      None.
123 *
124 * Limitations:         None.
125 *
126 * Algorithms:          None.
127 * Data Structures:     None.
128 *
129 * Registers Changed:   r8, r9, r10, r11, r12, r14, r15.
130 *
131 * Revision History:
132 *     6/03/14   Santiago Navonne       Initial revision.
133 *
134 */
135     .global clear_trace_old
136 clear_trace_old:                                /* clear all trace pixels on display */
137     MOVHI     r8, %hi(VRAM_BASE) /* start at base of VRAM */
138     ORI       r8, r8, %lo(VRAM_BASE)
139     MOVHI     r13, %hi(PIXEL_TRACE) /* load colors that will be cleared */
140     ORI       r13, r13, %lo(PIXEL_TRACE)
141     MOVHI     r14, %hi(PIXEL_CURSOR) /* which are trace and cursor */
142     ORI       r14, r14, %lo(PIXEL_CURSOR)
143     MOVI      r9, SIZE_X           /* will loop through all columns */
144     MOVI      r10, SIZE_Y          /* and all rows */
145     MOV       r11, r0              /* starting at (0, 0) */
146     MOV       r12, r0              /* (top left corner) */
147
148 trace_check:                                /* check if current pixel is part of trace */
149     LDWIO     r15, (r8)             /* read value from VRAM */
150     BEQ       r13, r15, trace_clear /* definitely clear if color is trace color */

```



```

151
152 cursor_check:                /* check if current pixel is part of cursor */
153     BNE     r14, r15, trace_row_loop /* also clear if part of cursor */
154
155 trace_clear:                  /* pixel is part of trace or cursor */
156     STWIO   r0, (r8)          /* so clear it */
157
158 trace_row_loop:              /* done with current pixel */
159     ADDI    r8, r8, WORD_SIZE /* so go to next */
160     ADDI    r11, r11, 1        /* and also increment column index */
161     BLT     r11, r9, trace_check /* if still within display, repeat */
162
163 trace_next_row:              /* done with current row */
164     ADDI    r8, r8, REMAINDER /* add remainder to finish up VRAM row */
165     MOV     r11, r0            /* reset column index */
166     ADDI    r12, r12, 1        /* and increment row index */
167     BLT     r12, r10, trace_check /* if still within display, repeat */
168
169     RET                                /* all done, so return */
170
171
172 /*
173 * plot_pixel
174 *
175 * Description:      This procedure changes the color to the pixel at the passed x, y
176 *                  coordinates, where the top left corner is (0, 0), to the passed
177 *                  color. Colors are specified with a 24-bit value, where the bottom
178 *                  8 bits represent the amount of blue, the following 8 the amount
179 *                  of green, and the next 8 the amount of red.
180 *
181 * Operation:        The function simply translates the x and y coordinates into a VRAM
182 *                  address by setting the top bits to the offset of the VRAM, and ORing
183 *                  in the shifted row and column indices. Then, it stores the passed
184 *                  color value at that address.
185 *
186 * Arguments:        x - x coordinate of the pixel, where leftmost column is 0 (r4).
187 *                  y - y coordinate of the pixel, where top row is 0 (r5).
188 *                  color - 24-bit value with RGB color the pixel should change to (r6).
189 *
190 * Return Value:     None.
191 *
192 * Local Variables:  None.
193 *
194 * Shared Variables: None.
195 *
196 * Global Variables: None.
197 *
198 * Input:            None.
199 *
200 * Output:           Changes the color of one pixel on the display.
201 *
202 * Error Handling:    None.
203 *
204 * Limitations:      None.
205 *
206 * Algorithms:       None.
207 * Data Structures:   None.
208 *
209 * Registers Changed: r8, r9, r10.
210 *
211 * Revision History:
212 *     6/03/14   Santiago Navonne   Initial revision.
213 *
214 */
215
216 .global plot_pixel
217 plot_pixel:                /* draw a pixel of the specified color */
218     MOVHI   r8, %hi(VRAM_BASE) /* find pixel location by first going to VRAM base */
219     ORI     r8, r8, %lo(VRAM_BASE)
220     MOVI    r9, ROW_ADDR_SHIFT /* shift the row to the row part of the address */
221     SLL     r9, r5, r9
222     MOVI    r10, COL_ADDR_SHIFT /* and the column to the column part */
223     SLL     r10, r4, r10
224     OR      r8, r8, r9          /* OR row, column, and VRAM base together */
225     OR      r8, r8, r10         /* to create final pixel address */
226     STWIO   r6, (r8)          /* and finally save passed color value to that address */

```

```

226
227     RET                                /* all done, so return */
228
229 /*
230 * pixel_color
231 *
232 * Description:      This procedure returns the color of the pixel at the passed x, y
233 *                  coordinates, where the top left corner is (0, 0). Colors are
234 *                  specified with a 24-bit RGB value, where the bottom 8 bits
235 *                  represent the amount of blue, the following 8 the amount of green,
236 *                  and the next 8 the amount of red.
237 *
238 * Operation:       The function simply translates the x and y coordinates into a VRAM
239 *                  address by setting the top bits to the offset of the VRAM, and ORing
240 *                  in the shifted row and column indeces. Then, it loads the color word
241 *                  from VRAM and returns it in r2.
242 *
243 * Arguments:       x - x coordinate of the pixel, where leftmost column is 0 (r4).
244 *                  y - y coordinate of the pixel, where top row is 0 (r5).
245 *
246 * Return Value:    color - 24-bit value with RGB color of requested pixel, or NO_TRACE
247 *                  if no trace was found at the requested coordinate(r2).
248 *
249 * Local Variables: None.
250 *
251 * Shared Variables: None.
252 *
253 * Global Variables: None.
254 *
255 * Input:           None.
256 *
257 * Output:          None.
258 *
259 * Error Handling:  None.
260 *
261 * Limitations:     None.
262 *
263 * Algorithms:      None.
264 * Data Structures: None.
265 *
266 * Registers Changed: r8, r9, r10, r2.
267 *
268 * Revision History:
269 *     6/03/14   Santiago Navonne   Initial revision.
270 *
271 */
272
273 .global pixel_color
274 pixel_color:                                /* read a pixel from display */
275     MOVHI    r8, %hi(VRAM_BASE) /* find pixel location by first going to VRAM base */
276     ORI      r8, r8, %lo(VRAM_BASE)
277     MOVI     r9, ROW_ADDR_SHIFT /* shift the row to the row part of the address */
278     SLL      r9, r5, r9
279     MOVI     r10, COL_ADDR_SHIFT /* and the column to the column part */
280     SLL      r10, r4, r10
281     OR       r8, r8, r9          /* OR row, column, and VRAM base together */
282     OR       r8, r8, r10         /* to create final pixel address */
283     LDWIO    r2, (r8)           /* and finally read color value from that address */
284
285     RET                                /* storing it in return register */

```

```

1  /*****
2  /*
3  /*          DISPLAY.H
4  /*          Display Interface Definitions
5  /*          Include File
6  /*          Digital Oscilloscope Project
7  /*          EE/CS 52
8  /*          Santiago Navonne
9  /*
10 /*****/
11
12 /*
13  This file contains the constants for the display interface routines. The
14  file includes hardware constants related to the memory layout of the display
15  are in the VRAM.
16
17
18  Revision History:
19      6/3/14  Santiago Navonne  Initial revision.
20  */
21
22 /* VRAM-related constants */
23 #define ROW_SIZE      512
24 #define REMAINDER      (ROW_SIZE-SIZE_X)*WORD_SIZE
25 #define ROW_ADDR_SHIFT 11
26 #define COL_ADDR_SHIFT 2
27

```

```

1  /*****
2  /*
3  /*          GENERAL.H          */
4  /*          General Assembly Definitions          */
5  /*          Include File          */
6  /*          Digital Oscilloscope Project          */
7  /*          EE/CS 52          */
8  /*          Santiago Navonne          */
9  /*          */
10 /*****
11
12 /*
13     This file contains general constants for the assembly functions within the
14     EE/CS 52 Digital Oscilloscope project.
15
16
17     Revision History:
18         5/30/14  Santiago Navonne  Initial revision.
19 */
20
21 /* General constants */
22 #define FALSE      0          /* Zero is false */
23 #define TRUE       1          /* Non-zero is true */
24 #define WORD_SIZE  4          /* A word is 4 bytes */
25 #define NEG_WORD_SIZE -4      /* Include negative to facilitate subtraction */
26
27 /* PIO register constants */
28 #define EDGE_CAP_OF 3*WORD_SIZE /* Offset of edge capture PIO register */
29 #define INTMASK_OF  2*WORD_SIZE /* Offset of interrupt mask PIO register */
30 #define ENABLE_ALL  0b00111111 /* Enable interrupts from all six sources */
31

```

```

1  /*****
2  /*
3  /*          INTERFAC.H
4  /*          Interface Definitions
5  /*          Include File
6  /*          Digital Oscilloscope Project
7  /*          EE/CS 52
8  /*
9  /*****
10
11 /*
12 This file contains the constants for interfacing between the C code and
13 the assembly code/hardware for the Digital Oscilloscope project.
14
15
16 Revision History:
17     3/8/94    Glen George    Initial revision.
18     3/13/94   Glen George    Updated comments.
19     3/17/97   Glen George    Added constant MAX_SAMPLE_SIZE and removed
20                               KEY_UNUSED.
21     5/14/14   Santiago Navonne Changed keypad codes.
22     6/01/14   Santiago Navonne Changed scope and sampling parameters.
23     6/03/14   Santiago Navonne Changed and added display parameters.
24 */
25
26
27
28 #ifndef __INTERFAC_H
29     #define __INTERFAC_H__
30
31
32 /* library include files */
33 /* none */
34
35 /* local include files */
36 /* none */
37
38
39
40
41 /* constants */
42
43 /* keypad constants */
44 #define KEY_MENU      1    /* <Menu>
45 #define KEY_UP        2    /* <Up>
46 #define KEY_DOWN      3    /* <Down>
47 #define KEY_LEFT      4    /* <Left>
48 #define KEY_RIGHT     5    /* <Right>
49 #define KEY_ILLEGAL   6    /* illegal key */
50
51 /* display constants */
52 #define SIZE_X         480    /* size in the x dimension */
53 #define SIZE_Y         272    /* size in the y dimension */
54 #define PIXEL_CLEAR    0x00000000 /* pixel off is black */
55 #define PIXEL_LINE     0x001B3830 /* lines are gray */
56 #define PIXEL_TEXT_H   0x00FFFFFF /* highlighted text is white */
57 #define PIXEL_TRACE    0x0000A000 /* trace is green */
58 #define PIXEL_TEXT_N   0x001B3830 /* normal text is gray */
59 #define PIXEL_CURSOR   0x00A00000 /* cursor is red */
60 #define NO_TRACE       0xFFFFFFFF /* no trace found */
61
62 /* scope parameters */
63 #define MIN_DELAY      0        /* minimum trigger delay */
64 #define MAX_DELAY      0xFFFFFFFF /* maximum trigger delay */
65 #define MIN_LEVEL      -12000   /* minimum trigger level (in mV) */
66 #define MAX_LEVEL      12000    /* maximum trigger level (in mV) */
67
68 /* sampling parameters */
69 #define MAX_SAMPLE_SIZE 512     /* maximum size of a sample (in samples) */
70
71
72 #endif
73

```

```

1  /*****
2  /*
3  /*                                KEYPROC                                */
4  /*                                Key Processing Functions                    */
5  /*                                Digital Oscilloscope Project                */
6  /*                                EE/CS 52                                    */
7  /*
8  /*****/
9
10 /*
11  This file contains the key processing functions for the Digital
12  Oscilloscope project. These functions are called by the main loop of the
13  system. The functions included are:
14      menu_down - process the <Down> key while in a menu
15      menu_key  - process the <Menu> key
16      menu_left - process the <Left> key while in a menu
17      menu_right - process the <Right> key while in a menu
18      menu_up   - process the <Up> key while in a menu
19      no_action - nothing to do
20
21  The local functions included are:
22      none
23
24  The locally global variable definitions included are:
25      none
26
27
28  Revision History
29      3/8/94   Glen George      Initial revision.
30      3/13/94 Glen George      Updated comments.
31 */
32
33
34
35 /* library include files */
36 /* none */
37
38 /* local include files */
39 #include "scopedef.h"
40 #include "keyproc.h"
41 #include "menu.h"
42
43
44
45
46 /*
47  no_action
48
49  Description:      This function handles a key when there is nothing to be
50                    done. It just returns.
51
52  Arguments:      cur_state (enum status) - the current system state.
53  Return Value:   (enum status) - the new system state (same as current
54                    state).
55
56  Input:          None.
57  Output:         None.
58
59  Error Handling:  None.
60
61  Algorithms:     None.
62  Data Structures: None.
63
64  Global Variables: None.
65
66  Author:         Glen George
67  Last Modified:  Mar. 8, 1994
68
69 */
70
71 enum status no_action(enum status cur_state)
72 {
73     /* variables */
74     /* none */
75

```

```

76
77
78     /* return the current state */
79     return cur_state;
80
81 }
82
83
84
85
86 /*
87 menu_key
88
89 Description:      This function handles the <Menu> key. If the passed
90                   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
253 Author:           Glen George
254 Last Modified:     Mar. 8, 1994
255
256 */
257
258 enum status menu_left(enum status cur_state)
259 {
260     /* variables */
261     /* none */
262
263
264
265     /* invoke the <Left> key function for the current menu entry */
266     menu_entry_left();
267
268
269     /* return the current state */
270     return cur_state;
271 }
272
273
274
275
276
277 /*
278 menu_right
279
280 Description:      This function handles the <Right> key when in a menu. It
281                   invokes the right function for the current menu entry and
282                   leaves the system state unchanged.
283
284 Arguments:        cur_state (enum status) - the current system state.
285 Return Value:      (enum status) - the new system state (same as current
286                   state).
287
288 Input:            None.
289 Output:           The menu display may be updated.
290
291 Error Handling:    None.
292
293 Algorithms:        None.
294 Data Structures:   None.
295
296 Global Variables: None.
297
298 Author:           Glen George
299 Last Modified:     Mar. 8, 1994
300

```

```

301 */
302
303 enum status  menu_right(enum status cur_state)
304 {
305     /* variables */
306     /* none */
307
308
309
310     /* invoke the <Right> key function for the current menu entry */
311     menu_entry_right();
312
313
314     /* return the current state */
315     return  cur_state;
316
317 }
318

```

```

1  /*****
2  /*
3  /*          KEYPROC.H
4  /*          Key Processing 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 key
13  processing functions (defined in keyproc.c) for the Digital Oscilloscope
14  project.
15
16
17  Revision History:
18      3/8/94   Glen George      Initial revision.
19      3/13/94  Glen George      Updated comments.
20  */
21
22
23
24 #ifndef __KEYPROC_H__
25     #define __KEYPROC_H__
26
27
28 /* library include files */
29 /* none */
30
31 /* local include files */
32 #include "scopedef.h"
33
34
35
36
37 /* constants */
38 /* none */
39
40
41
42
43 /* structures, unions, and typedefs */
44 /* none */
45
46
47
48
49 /* function declarations */
50
51 enum status  no_action(enum status);      /* nothing to do */
52
53 enum status  menu_key(enum status);      /* process the <Menu> key */
54
55 enum status  menu_up(enum status);      /* <Up> key in a menu */
56 enum status  menu_down(enum status);      /* <Down> key in a menu */
57 enum status  menu_left(enum status);      /* <Left> key in a menu */
58 enum status  menu_right(enum status);      /* <Right> key in a menu */
59
60
61 #endif
62

```

```

1  /*****
2  /*
3  /*          KEYS.S
4  /*          Key handlers
5  /*          Digital Oscilloscope Project
6  /*          EE/CS 52
7  /*          Santiago Navonne
8  /*
9  /*****/
10
11 /*
12 Key and rotary encoder control routines for the EE/CS 52 Digital Oscilloscope
13 project. Function definitions are included in this file, and are laid out
14 as follows:
15   - keys_init: Initializes the key handler's shared variables, and enables
16                 interrupts from the required sources, effectively preparing
17                 the user input section for use;
18   - keys_handler: Handles key press (and rotary encoder turn) interrupts;
19   - getkey: Returns the currently pending user action, blocking if none is
20             available.
21   - key_available: Checks whether a user action is currently pending.
22
23
24 Revision History:
25   5/7/14 Santiago Navonne Initial revision.
26   5/14/14 Santiago Navonne Added additional documentation.
27   6/7/14 Santiago Navonne Changed up/down rotation direction.
28 */
29
30 /* Includes */
31 #include "general.h" /* General constants */
32 #include "system.h" /* Base addresses */
33 #include "interfac.h" /* Software interface definitions */
34 #include "keys.h" /* Local constants */
35
36
37 /* Variables */
38 .section .data /* No alignment necessary: variables are bytes */
39 curr_key: .byte 0 /* Current pending key; 0 if no key available */
40
41 .section .text /* Code starts here */
42
43 /*
44 * keys_init
45 *
46 * Description: This procedure initializes the internal state of the key/
47 *              user input handling system, preparing any shared variables
48 *              for use and configuring interrupts. This function should be
49 *              called in order to start accepting user input.
50 *
51 * Operation: This procedure initializes any shared variables to their
52 *              default states:
53 *              - curr_key: value of the currently pending key (default: 0).
54 *              Additionally, the function registers the key press handler
55 *              as the default interrupt handler for key presses using the HAL
56 *              API alt_ic_isr_register, and finally unmask all interrupts by
57 *              writing to the corresponding PIO register.
58 *
59 * Arguments: None.
60 *
61 * Return Value: None.
62 *
63 * Local Variables: None.
64 *
65 * Shared Variables: - curr_key (write only).
66 *
67 * Global Variables: None.
68 *
69 * Input: None.
70 *
71 * Output: None.
72 *
73 * Error Handling: None.
74 *
75 * Limitations: None.

```

```

76 *
77 * Algorithms:      None.
78 * Data Structures: None.
79 *
80 * Registers Changed: r4, r5, r6, r7, r8, r9.
81 *
82 * Revision History:
83 *   5/7/14   Santiago Navonne   Initial revision.
84 *   5/14/14  Santiago Navonne   Added additional documentation.
85 *
86 */
87 .global keys_init
88 keys_init:
89     ADDI    sp, sp, NEG_WORD_SIZE /* push return address */
90     STW     ra, (sp)
91
92     MOVIA   r9, curr_key          /* no key (r0) available at start */
93     STB     r0, (r9)             /* so store it into variable curr_key */
94
95     MOVHI   r8, %hi(PIO_0_BASE)  /* write to the PIO registers */
96     ORI     r8, r8, %lo(PIO_0_BASE)
97     MOVI    r9, ENABLE_ALL       /* the ENABLE_ALL value */
98     STBIO   r9, EDGE_CAP_OF(r8)  /* sending general EOI to clear ints */
99
100    MOV      r4, r0               /* argument ic_id is ignored */
101    MOVI     r5, PIO_0_IRQ        /* second arg is IRQ num */
102    MOVIA    r6, keys_handler    /* third arg is int handler */
103    MOV      r7, r0              /* fourth arg is data struct (null) */
104    ADDI     sp, sp, NEG_WORD_SIZE /* fifth arg goes on stack */
105    STW      r0, (sp)            /* and is ignored (so 0) */
106    CALL     alt_ic_isr_register /* finally, call setup function */
107    ADDI     sp, sp, WORD_SIZE    /* clean up stack after call */
108
109    LDW      ra, (sp)             /* pop return address */
110    ADDI     sp, sp, WORD_SIZE
111
112    STBIO    r9, INTMASK_OF(r8)  /* enable (unmask) interrupts */
113
114    RET                                           /* and finally return */
115
116
117 /*
118 * keys_handler
119 *
120 * Description:      This procedure handles hardware interrupts generated by
121 *                   key presses and rotary encoder steps. Every time one of
122 *                   these fires, the shared variable containing the currently
123 *                   pending key is updated to indicate a key press. Note that
124 *                   previously pending key presses are overwritten by this
125 *                   function.
126 *                   The function is designed to support only one key press
127 *                   at a time; its behavior in the event of simultaneous key
128 *                   presses is undefined.
129 *
130 * Operation:        When called, the function first reads the edge capture
131 *                   register of the user input PIO interface to figure out
132 *                   which interrupt fired. It compares the read value to all
133 *                   the known constants, translating it into a key ID. Unknown
134 *                   values, which are caused by simultaneous key presses,
135 *                   are handled in the else case.
136 *                   After the key press is decoded, the identification code is
137 *                   saved to the shared variable curr_key.
138 *                   Note that the procedure uses multiple comparisons and not
139 *                   a jump table in order to save space; furthermore, the
140 *                   interrupt register value is not simply used as a key
141 *                   identifier to prevent simultaneous key presses from
142 *                   breaking the system.
143 *
144 * Arguments:         None.
145 *
146 * Return Value:      None.
147 *
148 * Local Variables:   None.
149 *
150 *

```

```

151 * Shared Variables: - curr_key: currently pending key press code (read/write).
152 *
153 * Global Variables: None.
154 *
155 * Input:           Key presses and rotary encoder turns from the user interface.
156 *
157 * Output:          None.
158 *
159 * Error Handling:   If multiple keys are pressed at once, the function's
160 *                  behavior is undefined.
161 *
162 * Limitations:      Only one simultaneous key press is accepted.
163 *                  Any previously recognized but not yet polled key presses
164 *                  are lost (overwritten) when a new event is received.
165 *
166 * Algorithms:       None.
167 * Data Structures:  None.
168 *
169 * Registers Changed: et.
170 *
171 * Revision History:
172 *   5/7/14   Santiago Navonne   Initial revision.
173 *   5/14/14  Santiago Navonne   Added additional documentation.
174 *
175 */
176 .global keys_handler
177 keys_handler:
178     ADDI    sp, sp, NEG_WORD_SIZE /* save r8 */
179     STW     r8, (sp)
180
181     MOVHI   et, %hi(PIO_0_BASE) /* fetch PIO edge capture register */
182     ORI     et, et, %lo(PIO_0_BASE)
183     LDBIO   r8, EDGE_CAP_OF(et)
184
185     STBIO   r8, EDGE_CAP_OF(et) /* and write back to send EOI */
186                                     /* figure out what interrupt fired */
187     MOVI    et, PUSH1_MASK /* check if it was pushbutton 1 */
188     BEQ     r8, et, keys_handler_push1
189     MOVI    et, PUSH2_MASK /* check if it was pushbutton 2 */
190     BEQ     r8, et, keys_handler_push2
191     MOVI    et, ROT1R_MASK /* check if it was rotary enc 1 right */
192     BEQ     r8, et, keys_handler_rot1r
193     MOVI    et, ROT1L_MASK /* check if it was rotary enc 1 left */
194     BEQ     r8, et, keys_handler_rot1l
195     MOVI    et, ROT2R_MASK /* check if it was rotary enc 2 right */
196     BEQ     r8, et, keys_handler_rot2r
197     JMPI    keys_handler_rot2l /* else it must be rotary enc 2 left */
198
199 keys_handler_push1: /* handle pushbutton 1 ints */
200     MOVI    et, KEY_MENU /* translates into menu key */
201     JMPI    keys_handler_done
202
203 keys_handler_push2: /* handle pushbutton 2 ints */
204     MOVI    et, KEY_MENU /* translates into menu key */
205     JMPI    keys_handler_done
206
207 keys_handler_rot1r: /* handle rotary enc 1 right ints */
208     MOVI    et, KEY_DOWN /* translates into down key */
209     JMPI    keys_handler_done
210
211 keys_handler_rot1l: /* handle rotary enc 1 left ints */
212     MOVI    et, KEY_UP /* translates into up key */
213     JMPI    keys_handler_done
214
215 keys_handler_rot2r: /* handle rotary enc 2 right ints */
216     MOVI    et, KEY_RIGHT /* translates into right key */
217     JMPI    keys_handler_done
218
219 keys_handler_rot2l: /* handle rotary enc 2 left ints */
220     MOVI    et, KEY_LEFT /* translates into left key */
221     JMPI    keys_handler_done
222
223 keys_handler_done: /* handling completed */
224     MOVIA   r8, curr_key /* save to curr_key */
225     STB     et, (r8) /* the processed key */

```

```

226
227     LDW      r8, (sp)          /* restore r8 */
228     ADDI     sp, sp, WORD_SIZE
229     RET                               /* all done */
230
231
232
233 /*
234 *  getkey
235 *
236 *  Description:      This procedure returns the identifier of the last pressed,
237 *                   unpolled key, as described in interfac.h.
238 *                   If no key press is pending, the function blocks.
239 *                   (To ensure non-blocking behavior, getkey calls should be
240 *                   preceded by key_available calls.)
241 *
242 *  Operation:       The function first fetches the value stored in curr_key and
243 *                   compares it to 0, which would indicate that there isn't
244 *                   actually any pending key press. In no key press is pending,
245 *                   the function keeps fetching the value until it is not 0.
246 *                   When the value is not 0, the function clears the value of
247 *                   curr_key (to delete the now reported press) and returns
248 *                   the retrieved value.
249 *
250 *  Arguments:       None.
251 *
252 *  Return Value:    key (r2) - ID code of the pending key, as defined in
253 *                   interfac.h.
254 *
255 *  Local Variables: None.
256 *
257 *  Shared Variables: - curr_key: currently pending key press code (read/write).
258 *
259 *  Global Variables: None.
260 *
261 *  Input:           None.
262 *
263 *  Output:          None.
264 *
265 *  Error Handling:  If no key is available, the function blocks until a key
266 *                   is pressed.
267 *
268 *  Limitations:     None.
269 *
270 *  Algorithms:      None.
271 *  Data Structures: None.
272 *
273 *  Registers Changed: r2, r8.
274 *
275 *  Revision History:
276 *      5/7/14    Santiago Navonne    Initial revision.
277 *      5/14/14   Santiago Navonne    Added additional documentation.
278 *
279 */
280     .global getkey
281 getkey:
282     MOVIA     r8, curr_key      /* return current pending key */
283     LDB       r2, (r8)
284     BEQ       r0, r2, getkey    /* if there is no key (curr_key == r0), block */
285
286     STB       r0, (r8)         /* clear current key */
287     RET                               /* return with current pending key in r2 */
288
289
290
291 /*
292 *  key_available
293 *
294 *  Description:      This procedure checks whether a key has been pressed and
295 *                   is available for polling. The function returns true
296 *                   (non-zero) if there's a key available, and non-zero if no
297 *                   key has been pressed.
298 *                   This function should be called before using getkey to avoid
299 *                   blocking.
300 *

```

```

301 * Operation:          The function simply returns the value stored in the shared
302 *                    variable curr_key, taking advantage of the fact that this
303 *                    value is zero if no key is available, and non-zero otherwise.
304 *
305 * Arguments:          None.
306 *
307 * Return Value:       key_available (r2) - true (non-zero) if a key press is
308 *                    available, false (zero) otherwise.
309 *
310 * Local Variables:    None.
311 *
312 * Shared Variables:   - curr_key: currently pending key press code (read only).
313 *
314 * Global Variables:   None.
315 *
316 * Input:              Key presses and rotary encoder turns from the user interface.
317 *
318 * Output:             None.
319 *
320 * Error Handling:      None.
321 *
322 * Limitations:        None.
323 *
324 * Algorithms:         None.
325 * Data Structures:    None.
326 *
327 * Registers Changed:  r2, r8.
328 *
329 * Revision History:
330 *   5/7/14    Santiago Navonne    Initial revision.
331 *   5/14/14   Santiago Navonne    Added additional documentation.
332 *
333 */
334 .globl key_available
335 key_available:
336     MOVIA    r8, curr_key        /* return current pending key */
337     LDB      r2, (r8)            /* will be zero (FALSE) if no key is pending */
338
339     RET                          /* return with boolean in r2 */
340
341
342

```



```

1  /*****
2  /*
3  /*          KEYS.H
4  /*          Key Handlers Definitions
5  /*          Include File
6  /*          Digital Oscilloscope Project
7  /*          EE/CS 52
8  /*          Santiago Navonne
9  /*
10 /*****
11
12 /*
13 This file contains the constants for the key press and rotary encoder
14 handler routines. The file includes interrupt masks used to determine the
15 source of interrupts; offsets of the PIO registers.
16
17
18 Revision History:
19     5/7/14 Santiago Navonne Initial revision.
20     5/14/14 Santiago Navonne Added additional documentation.
21 */
22
23 /* Interrupt masks */
24 #define PUSH1_MASK    0b00100000 /* Pushbutton 1 mask */
25 #define PUSH2_MASK    0b00010000 /* Pushbutton 2 mask */
26 #define ROT1R_MASK    0b00000100 /* Rotary encoder 1, right mask */
27 #define ROT1L_MASK    0b00001000 /* Rotary encoder 1, left mask */
28 #define ROT2R_MASK    0b00000001 /* Rotary encoder 2, right mask */
29 #define ROT2L_MASK    0b00000010 /* Rotary encoder 2, left mask */
30

```

```

1  /*****
2  /*
3  /*          LCDOUT          */
4  /*          LCD Output Functions          */
5  /*          Digital Oscilloscope Project          */
6  /*          EE/CS 52          */
7  /*
8  /*****
9
10 /*
11 This file contains the functions for doing output to the LCD screen for the
12 Digital Oscilloscope project. The functions included are:
13     clear_region - clear a region of the display
14     plot_char   - output a character
15     plot_hline  - draw a horizontal line
16     plot_string - output a string
17     plot_vline  - draw a vertical line
18     plot_cursor - plot the cursor
19
20 The local functions included are:
21     none
22
23 The locally global variable definitions included are:
24     none
25
26
27 Revision History
28     3/8/94   Glen George   Initial revision.
29     3/13/94  Glen George   Updated comments.
30     3/13/94  Glen George   Simplified code in plot_string function.
31     3/17/97  Glen George   Updated comments.
32     3/17/97  Glen George   Change plot_char() and plot_string() to use
33                             enum char_style instead of an int value.
34     5/27/98  Glen George   Change plot_char() to explicitly declare the
35                             size of the external array to avoid linker
36                             errors.
37     6/3/14   Santiago Navonne Changed UI display colors, added support for
38                             highlighted characters.
39 */
40
41
42
43 /* library include files */
44 /* none */
45
46 /* local include files */
47 #include "interfac.h"
48 #include "scopedef.h"
49 #include "lcdout.h"
50
51
52 extern int pixel_color(int, int);
53
54
55
56 /*
57     clear_region
58
59 Description:      This function clears the passed region of the display.
60                   The region is described by its upper left corner pixel
61                   coordinate and the size (in pixels) in each dimension.
62
63 Arguments:      x_ul (int) - x coordinate of upper left corner of the
64                   region to be cleared.
65                   y_ul (int) - y coordinate of upper left corner of the
66                   region to be cleared.
67                   x_size (int) - horizontal size of the region.
68                   y_size (int) - vertical size of the region.
69 Return Value:    None.
70
71 Input:          None.
72 Output:         A portion of the screen is cleared (set to PIXEL_CLEAR).
73
74 Error Handling:  No error checking is done on the coordinates.
75

```

```

76 Algorithms:      None.
77 Data Structures: None.
78
79 Global Variables: None.
80
81 Author:          Glen George
82 Last Modified:   June 03, 2014
83
84 */
85
86 void clear_region(int x_ul, int y_ul, int x_size, int y_size)
87 {
88     /* variables */
89     int x;        /* x coordinate to clear */
90     int y;        /* y coordinate to clear */
91
92
93
94     /* loop, clearing the display */
95     for (x = x_ul; x < (x_ul + x_size); x++) {
96         for (y = y_ul; y < (y_ul + y_size); y++) {
97
98             /* clear this pixel */
99             plot_pixel(x, y, PIXEL_CLEAR);
100         }
101     }
102
103
104     /* done clearing the display region - return */
105     return;
106 }
107
108
109
110
111
112 /*
113 plot_hline
114
115 Description:      This function draws a horizontal line from the passed
116                   position for the passed length. The line is always drawn
117                   with the color PIXEL_LINE. The position (0,0) is the
118                   upper left corner of the screen.
119
120 Arguments:      start_x (int) - starting x coordinate of the line.
121                 start_y (int) - starting y coordinate of the line.
122                 length (int) - length of the line (positive for a line
123                             to the "right" and negative for a line to
124                             the "left").
125 Return Value:   None.
126
127 Input:          None.
128 Output:         A horizontal line is drawn at the specified position.
129
130 Error Handling: No error checking is done on the coordinates.
131
132 Algorithms:     None.
133 Data Structures: None.
134
135 Global Variables: None.
136
137 Author:         Glen George
138 Last Modified:  June 03, 2014
139
140 */
141
142 void plot_hline(int start_x, int start_y, int length)
143 {
144     /* variables */
145     int x;        /* x position while plotting */
146
147
148     int init_x;    /* starting x position to plot */
149     int end_x;     /* ending x position to plot */
150

```

```

151
152  /* check if a line to the "right" or "left" */
153  if (length > 0) {
154
155      /* line to the "right" - start at start_x, end at start_x + length */
156      init_x = start_x;
157      end_x = start_x + length;
158  }
159  else {
160
161      /* line to the "left" - start at start_x + length, end at start_x */
162      init_x = start_x + length;
163      end_x = start_x;
164  }
165
166
167  /* loop, outputting points for the line (always draw to the "right") */
168  for (x = init_x; x < end_x; x++)
169      /* plot a point of the line */
170      plot_pixel(x, start_y, PIXEL_LINE);
171
172
173  /* done plotting the line - return */
174  return;
175
176 }
177
178
179
180
181 /*
182 plot_vline
183
184 Description:      This function draws a vertical line from the passed
185                  position for the passed length. The line is always drawn
186                  with the color PIXEL_LINE. The position (0,0) is the
187                  upper left corner of the screen.
188
189 Arguments:      start_x (int) - starting x coordinate of the line.
190                  start_y (int) - starting y coordinate of the line.
191                  length (int) - length of the line (positive for a line
192                              going "down" and negative for a line
193                              going "up").
194 Return Value:   None.
195
196 Input:          None.
197 Output:         A vertical line is drawn at the specified position.
198
199 Error Handling: No error checking is done on the coordinates.
200
201 Algorithms:     None.
202 Data Structures: None.
203
204 Global Variables: None.
205
206 Author:        Glen George
207 Last Modified:  June 03, 2014
208
209 */
210
211 void plot_vline(int start_x, int start_y, int length)
212 {
213     /* variables */
214     int y;      /* y position while plotting */
215
216     int init_y; /* starting y position to plot */
217     int end_y;  /* ending y position to plot */
218
219
220
221     /* check if an "up" or "down" line */
222     if (length > 0) {
223
224         /* line going "down" - start at start_y, end at start_y + length */
225         init_y = start_y;

```

```

226     end_y = start_y + length;
227 }
228 else {
229
230     /* line going "up" - start at start_y + length, end at start_y */
231     init_y = start_y + length;
232     end_y = start_y;
233 }
234
235
236 /* loop, outputting points for the line (always draw "down") */
237 for (y = init_y; y < end_y; y++)
238     /* plot a point of the line */
239     plot_pixel(start_x, y, PIXEL_LINE);
240
241
242 /* done plotting the line - return */
243 return;
244
245 }
246
247
248
249
250 /*
251 plot_char
252
253 Description:      This function outputs the passed character to the LCD
254                   screen at passed location. The passed location is given
255                   as a character position with (0,0) being the upper left
256                   corner of the screen. The character can be drawn in
257                   "normal video" (gray on black), "reverse video" (black
258                   on gray), or highlighted (white on black).
259
260 Arguments:      pos_x (int)          - x coordinate (in character
261                                   cells) of the character.
262                 pos_y (int)          - y coordinate (in character
263                                   cells) of the character.
264                 c (char)             - the character to plot.
265                 style (enum char_style) - style with which to plot the
266                                   character (NORMAL or REVERSE).
267
268 Return Value:    None.
269
270 Input:          None.
271 Output:         A character is output to the LCD screen.
272
273 Error Handling:  No error checking is done on the coordinates or the
274                 character (to ensure there is a bit pattern for it).
275
276 Algorithms:     None.
277 Data Structures: The character bit patterns are stored in an external
278                 array.
279
280 Global Variables: None.
281
282 Author:         Glen George
283 Last Modified:  June 03, 2014
284 */
285
286 void plot_char(int pos_x, int pos_y, char c, enum char_style style)
287 {
288     /* variables */
289
290     /* pointer to array of character bit patterns */
291     extern const unsigned char char_patterns[(VERT_SIZE - 1) * 128];
292
293     int bits;          /* a character bit pattern */
294
295     int col;           /* column loop index */
296     int row;           /* character row loop index */
297
298     int x;             /* x pixel position for the character */
299     int y;             /* y pixel position for the character */
300

```

```

301     int color = PIXEL_TEXT_N; /* pixel drawing color */
302
303
304
305     /* setup the pixel positions for the character */
306     x = pos_x * HORIZ_SIZE;
307     y = pos_y * VERT_SIZE;
308
309
310     /* loop outputting the bits to the screen */
311     for (row = 0; row < VERT_SIZE; row++) {
312
313         /* get the character bits for this row from the character table */
314         if (row == (VERT_SIZE - 1))
315             /* last row - blank it */
316             bits = 0;
317         else
318             /* in middle of character, get the row from the bit patterns */
319             bits = char_patterns[(c * (VERT_SIZE - 1)) + row];
320
321         /* take care of "normal/reverse video" */
322         if (style == REVERSE)
323             /* invert the bits for "reverse video" */
324             bits = ~bits;
325         if (style == HIGHLIGHTED)
326             color = PIXEL_TEXT_H;
327
328         /* get the bits "in position" (high bit is output first */
329         bits <= (8 - HORIZ_SIZE);
330
331
332         /* now output the row of the character, pixel by pixel */
333         for (col = 0; col < HORIZ_SIZE; col++) {
334
335             /* output this pixel in the appropriate color */
336             if ((bits & 0x80) == 0)
337                 /* blank pixel - output in PIXEL_CLEAR */
338                 plot_pixel(x + col, y, PIXEL_CLEAR);
339             else
340                 /* black pixel - output in PIXEL_TEXT */
341                 plot_pixel(x + col, y, color);
342
343             /* shift the next bit into position */
344             bits <= 1;
345         }
346
347
348         /* next row - update the y position */
349         y++;
350     }
351
352
353     /* all done, return */
354     return;
355 }
356
357
358
359
360
361 /*
362 plot_string
363
364 Description:      This function outputs the passed string to the LCD screen
365                  at passed location. The passed location is given as a
366                  character position with (0,0) being the upper left corner
367                  of the screen. There is no line wrapping, so the entire
368                  string must fit on the passed line (pos_y). The string
369                  can be drawn in "normal video" (black on white) or
370                  "reverse video" (white on black).
371
372 Arguments:      pos_x (int)          - x coordinate (in character
373                  cells) of the start of the
374                  string.
375                  pos_y (int)        - y coordinate (in character

```

```

376         cells) of the start of the
377         string.
378     s (const char *) - the string to output.
379     style (enum char style) - style with which to plot
380         characters of the string.
381 Return Value:      None.
382
383 Input:             None.
384 Output:            A string is output to the LCD screen.
385
386 Error Handling:     No checking is done to insure the string is fully on the
387                     screen (the x and y coordinates and length of the string
388                     are not checked).
389
390 Algorithms:        None.
391 Data Structures:    None.
392
393 Global Variables:  None.
394
395 Author:            Glen George
396 Last Modified:     Mar. 17, 1997
397
398 */
399
400 void plot_string(int pos_x, int pos_y, const char *s, enum char_style style)
401 {
402     /* variables */
403     /* none */
404
405
406
407     /* loop, outputting characters from string s */
408     while (*s != '\0')
409
410         /* output this character and move to the next character and screen position */
411         plot_char(pos_x++, pos_y, *s++, style);
412
413
414     /* all done, return */
415     return;
416
417 }
418

```

```

1  /*****
2  /*
3  /*          LCDOUT.H          */
4  /*          LCD Output 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 LCD output
13 functions used in the Digital Oscilloscope project and defined in lcdout.c.
14
15
16 Revision History:
17 3/8/94   Glen George   Initial revision.
18 3/13/94  Glen George   Updated comments.
19 3/17/97  Glen George   Added enumerated type char_style and updated
20                        function prototypes.
21 6/3/14   Santiago Navonne Added highlighted character style.
22 */
23
24
25
26
27 #ifndef __LCDOUT_H__
28 #define __LCDOUT_H__
29
30
31 /* library include files */
32 /* none */
33
34 /* local include files */
35 /* none */
36
37
38
39
40 /* constants */
41
42 /* character output styles */
43
44 /* size of a character (includes 1 pixel space to the left and below character) */
45 #define VERT_SIZE 8 /* vertical size (in pixels -> 7+1) */
46 #define HORIZ_SIZE 6 /* horizontal size (in pixels -> 5+1) */
47
48
49
50
51 /* structures, unions, and typedefs */
52
53 /* character output styles */
54 enum char_style { NORMAL, /* "normal video" */
55                 REVERSE, /* "reverse video" */
56                 HIGHLIGHTED /* highlighted text */
57 };
58
59
60
61
62 /* function declarations */
63
64 void clear_region(int, int, int, int); /* clear part of the display */
65
66 void plot_hline(int, int, int); /* draw a horizontal line */
67 void plot_vline(int, int, int); /* draw a vertical line */
68
69 void plot_char(int, int, char, enum char_style); /* output a character */
70 void plot_string(int, int, const char *, enum char_style); /* output a string */
71
72 int plot_cursor(int, int); /* draws the cursor on the trace */
73
74
75 #endif

```



```

1  /*****
2  /*
3  /*          MAINLOOP          */
4  /*          Main Program Loop          */
5  /*          Digital Oscilloscope Project          */
6  /*          EE/CS 52          */
7  /*
8  /*****
9
10 /*
11 This file contains the main processing loop (background) for the Digital
12 Oscilloscope project. The only global function included is:
13     main - background processing loop
14
15 The local functions included are:
16     key_lookup - get a key and look up its keycode
17
18 The locally global variable definitions included are:
19     none
20
21
22 Revision History
23     3/8/94    Glen George      Initial revision.
24     3/9/94    Glen George      Changed initialized const arrays to static
25                          (in addition to const).
26     3/9/94    Glen George      Moved the position of the const keyword in
27                          declarations of arrays of pointers.
28     3/13/94   Glen George      Updated comments.
29     3/13/94   Glen George      Removed display_menu call after plot_trace,
30                          the plot function takes care of the menu.
31     3/17/97   Glen George      Updated comments.
32     3/17/97   Glen George      Made key_lookup function static to make it
33                          truly local.
34     3/17/97   Glen George      Removed KEY_UNUSED and KEYCODE_UNUSED
35                          references (no longer used).
36     5/27/08   Glen George      Changed code to only check for sample done if
37                          it is currently sampling.
38     6/03/14   Santiago Navonne Added initialization code.
39     6/11/14   Santiago Navonne Added sleep time between draws.
40 */
41
42
43
44 /* library include files */
45 #include "unistd.h"
46
47 /* local include files */
48 #include "interfac.h"
49 #include "scopedef.h"
50 #include "keyproc.h"
51 #include "menu.h"
52 #include "tracutil.h"
53
54
55
56
57 /* local function declarations */
58 static enum keycode key_lookup(void);          /* translate key values into keycodes */
59
60
61
62
63 /*
64     main
65
66     Description:      This procedure is the main program loop for the Digital
67                      Oscilloscope. It loops getting keys from the keypad,
68                      processing those keys as is appropriate. It also handles
69                      starting scope sample collection and updating the LCD
70                      screen. Additionally, it initializes the triggering logic
71                      and key interface.
72
73     Arguments:       None.
74     Return Value:    (int) - return code, always 0 (never returns).
75

```

```

76 Input:          Keys from the keypad.
77 Output:         Traces and menus to the display.
78
79 Error Handling:  Invalid input is ignored.
80
81 Algorithms:      The function is table-driven. The processing routines
82                  for each input are given in tables which are selected
83                  based on the context (state) the program is operating in.
84 Data Structures: Array (process_key) to associate keys with actions
85                  (functions to call).
86
87 Global Variables: None.
88
89 Author:          Glen George
90 Last Modified:   June 11, 2014
91
92 */
93
94 int main()
95 {
96     /* initialize keys, triggering */
97     keys_init();
98     trigger_init();
99
100    /* variables */
101    enum keycode      key;          /* an input key */
102
103    enum status
104    state = MENU_ON;    /* current program state */
105
106    unsigned char *sample;          /* a captured trace */
107
108    /* key processing functions (one for each system state type and key) */
109    static enum status (* const process_key[NUM_KEYCODES][NUM_STATES])(enum status) =
110    /* Current System State */
111    /* MENU_ON    MENU_OFF    Input Key */
112    { { menu_key,    menu_key    }, /* <Menu> */
113      { menu_up,    no_action   }, /* <Up> */
114      { menu_down,  no_action   }, /* <Down> */
115      { menu_left,  no_action   }, /* <Left> */
116      { menu_right, no_action   }, /* <Right> */
117      { no_action,  no_action   } }; /* illegal key */
118
119
120
121    /* first initialize everything */
122    clear_display();    /* clear the display */
123
124    init_trace();       /* initialize the trace routines */
125    init_menu();        /* initialize the menu system */
126
127
128    /* infinite loop processing input */
129    while(TRUE) {
130
131        /* check if ready to do a trace */
132        if (trace_rdy())
133            /* ready for a trace - do it */
134            do_trace();
135
136
137        /* check if have a trace to display */
138        if (is_sampling() && ((sample = sample_done()) != NULL)) {
139
140            /* have a trace - output it */
141            plot_trace(sample);
142
143            /* sleep for some time to reduce blinking of display */
144            /*usleep(DRAW_INTERVAL);
145
146            /* done processing this trace */
147            trace_done();
148        }
149    }
150

```

```

151  /* now check for keypad input */
152  if (key_available()) {
153
154      /* have keypad input - get the key */
155      key = key_lookup();
156
157      /* execute processing routine for that key */
158      state = process_key[key][state](state);
159  }
160  }
161
162
163  /* done with main (never should get here), return 0 */
164  return 0;
165
166  }
167
168
169
170
171  /*
172  key_lookup
173
174  Description:      This function gets a key from the keypad and translates
175                    the raw keycode to an enumerated keycode for the main
176                    loop.
177
178  Arguments:        None.
179  Return Value:     (enum keycode) - type of the key input on keypad.
180
181  Input:            Keys from the keypad.
182  Output:           None.
183
184  Error Handling:   Invalid keys are returned as KEYCODE_ILLEGAL.
185
186  Algorithms:       The function uses an array to lookup the key types.
187  Data Structures:  Array of key types versus key codes.
188
189  Global Variables: None.
190
191  Author:           Glen George
192  Last Modified:    Mar. 17, 1997
193
194  */
195
196  static enum keycode key_lookup()
197  {
198      /* variables */
199
200      const static enum keycode keycodes[] = /* array of keycodes */
201      {
202          /* order must match keys array exactly */
203          KEYCODE_MENU, /* <Menu> */ /* also need an extra element */
204          KEYCODE_UP, /* <Up> */ /* for unknown key codes */
205          KEYCODE_DOWN, /* <Down> */
206          KEYCODE_LEFT, /* <Left> */
207          KEYCODE_RIGHT, /* <Right> */
208          KEYCODE_ILLEGAL /* other keys */
209      };
210
211      const static int keys[] = /* array of key values */
212      {
213          /* order must match keycodes array exactly */
214          KEY_MENU, /* <Menu> */
215          KEY_UP, /* <Up> */
216          KEY_DOWN, /* <Down> */
217          KEY_LEFT, /* <Left> */
218          KEY_RIGHT, /* <Right> */
219      };
220
221      int key; /* an input key */
222
223      int i; /* general loop index */
224
225      /* get a key */

```

```
226     key = getkey();
227
228
229     /* lookup key in keys array */
230     for (i = 0; ((i < (sizeof(keys)/sizeof(int))) && (key != keys[i])); i++);
231
232
233     /* return the appropriate key type */
234     return keycodes[i];
235
236 }
237
```

```

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 static void display_entry(int, int);      /* display a menu entry and its setting */
78
79
80
81
82 /* locally global variables */
83 static int menu_display;                  /* TRUE if menu is currently displayed */
84
85 const static 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 static 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(MIN_DELAY);        /* minimum 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     static 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     static 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 static void display_entry(int entry, int selected)
671 {
672     /* variables */
673     /* none */
674
675

```

```

676
677
678 /* output the menu entry with the appropriate color */
679 plot_string((MENU_X + menu[entry].h_off), (MENU_Y + entry), menu[entry].s,
680             (selected ? SELECTED : NORMAL));
681 /* also output the menu option with the appropriate color */
682 menu[entry].display((MENU_X + menu[entry].opt_off), (MENU_Y + entry),
683                     (selected ? OPTION_SELECTED : OPTION_NORMAL));
684
685 /* all done outputting this menu entry - return */
686 return;
687
688 }
689

```

```

1  /*****
2  /*
3  /*          MENU.H
4  /*          Menu 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 deal with menus (defined in menu.c) for the Digital Oscilloscope
14 project.
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      Added definitions for SELECTED,
21                          OPTION_NORMAL, and OPTION_SELECTED.
22 6/03/14  Santiago Navonne Changed selected menu and option style to HIGHLIGHTED.
23 */
24
25
26
27 #ifndef __MENU_H__
28 #define __MENU_H__
29
30
31 /* library include files */
32 /* none */
33
34 /* local include files */
35 #include "interfac.h"
36 #include "scopedef.h"
37 #include "lcdout.h"
38
39
40
41
42 /* constants */
43
44 /* menu size */
45 #define MENU_WIDTH 16          /* menu width (in characters) */
46 #define MENU_HEIGHT 7         /* menu height (in characters) */
47 #define MENU_SIZE_X (MENU_WIDTH * HORIZ_SIZE) /* menu width (in pixels) */
48 #define MENU_SIZE_Y (MENU_HEIGHT * VERT_SIZE) /* menu height (in pixels) */
49
50 /* menu position */
51 #define MENU_X (LCD_WIDTH - MENU_WIDTH - 1) /* x position (in characters) */
52 #define MENU_Y 0 /* y position (in characters) */
53 #define MENU_UL_X (MENU_X * HORIZ_SIZE) /* x position (in pixels) */
54 #define MENU_UL_Y (MENU_Y * VERT_SIZE) /* y position (in pixels) */
55
56 /* menu colors */
57 #define SELECTED HIGHLIGHTED /* color for a selected menu entry */
58 #define OPTION_SELECTED HIGHLIGHTED /* color for a selected menu entry option */
59 #define OPTION_NORMAL NORMAL /* color for an unselected menu entry option */
60
61 /* number of menu entries */
62 #define NO_MENU_ENTRIES (sizeof(menu) / sizeof(struct menu_item))
63
64
65
66
67 /* structures, unions, and typedefs */
68
69 /* data for an item in a menu */
70 struct menu_item { const char *s; /* string for menu entry */
71                  int h_off; /* horizontal offset of entry */
72                  int opt_off; /* horizontal offset of option setting */
73                  void (*display)(int, int, int); /* option display function */
74                  };
75

```

```

76
77
78
79 /* function declarations */
80
81 /* menu initialization function */
82 void init_menu(void);
83
84 /* menu display functions */
85 void clear_menu(void); /* clear the menu display */
86 void display_menu(void); /* display the menu */
87 void refresh_menu(void); /* refresh the menu */
88
89 /* menu update functions */
90 void reset_menu(void); /* reset the menu to first entry */
91 void next_entry(void); /* go to the next menu entry */
92 void previous_entry(void); /* go to the previous menu entry */
93
94 /* menu entry functions */
95 void menu_entry_left(void); /* do the <Left> key for the menu entry */
96 void menu_entry_right(void); /* do the <Right> key for the menu entry */
97
98
99 #endif
100

```



```

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 trigger 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 trigger delay
32     set_trg_level     - set the trigger level
33     set_trg_slope     - set the trigger slope
34     set_trigger_mode  - set the trigger mode
35     sweep_down        - decrease the sweep rate
36     sweep_up          - increase the sweep rate
37     trg_delay_down    - decrease the trigger delay
38     trg_delay_up      - increase the trigger delay
39     trg_level_down    - decrease the trigger level
40     trg_level_up      - increase the trigger level
41     trg_slope_toggle  - toggle the trigger slope between "+" and "-"
42
43 The local functions included are:
44     adjust_trg_delay  - adjust the trigger delay for a new sweep rate
45     cvt_num_field     - converts a numeric field value to a string
46
47 The locally global variable definitions included are:
48     delay             - current trigger delay
49     level              - current trigger level
50     scale              - current display scale type
51     slope              - current trigger slope
52     sweep              - current sweep rate
53     sweep_rates        - table of information on possible sweep rates
54     trigger_mode       - current triggering mode
55
56
57 Revision History
58     3/8/94   Glen George   Initial revision.
59     3/13/94  Glen George   Updated comments.
60     3/13/94  Glen George   Changed all arrays of constant strings to be
61                             static so compiler generates correct code.
62     3/13/94  Glen George   Changed scale to type enum scale_type and
63                             output the selection as "None" or "Axes".
64                             This will allow for easier future expansion.
65     3/13/94  Glen George   Changed name of set_axes function (in
66                             tracutil.c) to set_display_scale.
67     3/10/95  Glen George   Changed calculation of displayed trigger
68                             level to use constants MIN_TRG_LEVEL_SET and
69                             MAX_TRG_LEVEL_SET to get the trigger level
70                             range.
71     3/17/97  Glen George   Updated comments.
72     5/3/06   Glen George   Changed sweep definitions to include new
73                             sweep rates of 100 ns, 200 ns, 500 ns, and
74                             1 us and updated functions to handle these
75                             new rates.

```

```

76      5/9/06    Glen George      Added new a triggering mode (automatic
77                                     triggering) and a new scale (grid) and
78                                     updated functions to implement these options.
79      5/9/06    Glen George      Added functions for setting the triggering
80                                     mode and scale by going up and down the list
81                                     of possibilities instead of just toggling
82                                     between one of two possibilities (since there
83                                     are more than two now).
84      5/9/06    Glen George      Added accessor function (get_trigger_mode)
85                                     to be able to get the current trigger mode.
86      6/6/14    Santiago Navonne  Added fastest sweep rate and changed their
87                                     values to reflect actual possible rates.
88      6/11/14   Santiago Navonne  Modified delay set function to support faster
89                                     sweep rates.
90  */
91
92
93
94  /* library include files */
95  /* none */
96
97  /* local include files */
98  #include "interfac.h"
99  #include "scopedef.h"
100 #include "lcdout.h"
101 #include "menuact.h"
102 #include "tracutil.h"
103
104
105
106
107  /* local function declarations */
108  static void adjust_trg_delay(int, int);      /* adjust the trigger delay for new sweep */
109  static void cvt_num_field(long int, char *); /* convert a number to a string */
110
111
112
113
114  /* locally global variables
115
116  /* trace parameters */
117  static enum trigger_type trigger_mode; /* current triggering mode */
118  static enum scale_type scale; /* current scale type */
119  static int sweep; /* sweep rate index */
120  static int level; /* current trigger level */
121  static enum slope_type slope; /* current trigger slope */
122  static long int delay; /* current trigger delay */
123
124  /* sweep rate information */
125  static const struct sweep_info sweep_rates[] =
126  { { 19000000L, " 52 ns " },
127    { 9500000L, " 104 ns" },
128    { 4750000L, " 208 ns" },
129    { 2000000L, " 500 ns" },
130    { 1000000L, " 1 \004s " },
131    { 500000L, " 2 \004s " },
132    { 200000L, " 5 \004s " },
133    { 100000L, " 10 \004s " },
134    { 50000L, " 20 \004s " },
135    { 20000L, " 50 \004s " },
136    { 10000L, " 100 \004s" },
137    { 5000L, " 200 \004s" },
138    { 2000L, " 500 \004s" },
139    { 1000L, " 1 ms " },
140    { 500L, " 2 ms " },
141    { 200L, " 5 ms " },
142    { 100L, " 10 ms " },
143    { 50L, " 20 ms " } };
144
145
146
147
148  /*
149  no_menu_action
150

```

```

151 Description:      This function handles a menu action when there is nothing
152                   to be done.  It just returns.
153
154 Arguments:        None.
155 Return Value:     None.
156
157 Input:            None.
158 Output:           None.
159
160 Error Handling:    None.
161
162 Algorithms:        None.
163 Data Structures:   None.
164
165 Global Variables: None.
166
167 Author:           Glen George
168 Last Modified:    Mar. 8, 1994
169
170 */
171
172 void no_menu_action()
173 {
174     /* variables */
175     /* none */
176
177
178     /* nothing to do - return */
179     return;
180
181 }
182
183
184
185
186
187 /*
188 no_display
189
190 Description:      This function handles displaying a menu option's setting
191                   when there is nothing to display.  It just returns,
192                   ignoring all arguments.
193
194 Arguments:        x_pos (int) - x position (in character cells) at which to
195                   display the menu option (not used).
196                   y_pos (int) - y position (in character cells) at which to
197                   display the menu option (not used).
198                   style (int) - style with which to display the menu option
199                   (not used).
200 Return Value:     None.
201
202 Input:            None.
203 Output:           None.
204
205 Error Handling:    None.
206
207 Algorithms:        None.
208 Data Structures:   None.
209
210 Global Variables: None.
211
212 Author:           Glen George
213 Last Modified:    Mar. 8, 1994
214
215 */
216
217 void no_display(int x_pos, int y_pos, int style)
218 {
219     /* variables */
220     /* none */
221
222
223     /* nothing to do - return */
224     return;
225

```

```

226 }
227 }
228
229
230
231
232 /*
233  set_trigger_mode
234
235  Description:      This function sets the triggering mode to the passed
236                   value.
237
238  Arguments:       m (enum trigger_type) - mode to which to set the
239                   triggering mode.
240  Return Value:    None.
241
242  Input:           None.
243  Output:          None.
244
245  Error Handling:   None.
246
247  Algorithms:      None.
248  Data Structures: None.
249
250  Global Variables: trigger_mode - initialized to the passed value.
251
252  Author:          Glen George
253  Last Modified:   Mar. 8, 1994
254
255 */
256
257 void set_trigger_mode(enum trigger_type m)
258 {
259     /* variables */
260     /* none */
261
262
263
264     /* set the trigger mode */
265     trigger_mode = m;
266
267     /* set the new mode */
268     set_mode(trigger_mode);
269
270
271     /* all done setting the trigger mode - return */
272     return;
273 }
274
275
276
277
278
279 /*
280  get_trigger_mode
281
282  Description:      This function returns the current triggering mode.
283
284  Arguments:       None.
285  Return Value:    (enum trigger_type) - current triggering mode.
286
287  Input:           None.
288  Output:          None.
289
290  Error Handling:   None.
291
292  Algorithms:      None.
293  Data Structures: None.
294
295  Global Variables: trigger_mode - value is returned (not changed).
296
297  Author:          Glen George
298  Last Modified:   May 9, 2006
299
300 */

```

```

301
302 enum trigger_type  get_trigger_mode()
303 {
304     /* variables */
305     /* none */
306
307
308
309     /* return the current trigger mode */
310     return  trigger_mode;
311
312 }
313
314
315
316
317 /*
318     mode_down
319
320     Description:      This function handles moving down the list of trigger
321                      modes.  It changes to the "next" triggering mode and
322                      sets that as the current mode.
323
324     Arguments:       None.
325     Return Value:    None.
326
327     Input:           None.
328     Output:          None.
329
330     Error Handling:   None.
331
332     Algorithms:      None.
333     Data Structures:  None.
334
335     Global Variables: trigger_mode - changed to "next" trigger mode.
336
337     Author:          Glen George
338     Last Modified:   May 9, 2006
339
340 */
341
342 void  mode_down()
343 {
344     /* variables */
345     /* none */
346
347
348
349     /* move to the "next" triggering mode */
350     if (trigger_mode == NORMAL_TRIGGER)
351         trigger_mode = AUTO_TRIGGER;
352     else if (trigger_mode == AUTO_TRIGGER)
353         trigger_mode = ONESHOT_TRIGGER;
354     else
355         trigger_mode = NORMAL_TRIGGER;
356
357     /* set the new mode */
358     set_mode(trigger_mode);
359
360
361     /* all done with the trigger mode - return */
362     return;
363
364 }
365
366
367
368
369 /*
370     mode_up
371
372     Description:      This function handles moving up the list of trigger
373                      modes.  It changes to the "previous" triggering mode and
374                      sets that as the current mode.
375

```

```

376 Arguments:      None.
377 Return Value:   None.
378
379 Input:          None.
380 Output:         None.
381
382 Error Handling:  None.
383
384 Algorithms:     None.
385 Data Structures: None.
386
387 Global Variables: trigger_mode - changed to "previous" trigger mode.
388
389 Author:         Glen George
390 Last Modified:  May 9, 2006
391
392 */
393
394 void mode_up()
395 {
396     /* variables */
397     /* none */
398
399
400
401     /* move to the "previous" triggering mode */
402     if (trigger_mode == NORMAL_TRIGGER)
403         trigger_mode = ONESHOT_TRIGGER;
404     else if (trigger_mode == AUTO_TRIGGER)
405         trigger_mode = NORMAL_TRIGGER;
406     else
407         trigger_mode = AUTO_TRIGGER;
408
409     /* set the new mode */
410     set_mode(trigger_mode);
411
412
413     /* all done with the trigger mode - return */
414     return;
415 }
416
417
418
419
420
421 /*
422 display_mode
423
424 Description:      This function displays the current triggering mode at the
425                   passed position, in the passed style.
426
427 Arguments:        x_pos (int) - x position (in character cells) at which to
428                   display the trigger mode.
429                   y_pos (int) - y position (in character cells) at which to
430                   display the trigger mode.
431                   style (int) - style with which to display the trigger
432                   mode.
433 Return Value:     None.
434
435 Input:           None.
436 Output:          The trigger mode is displayed at the passed position on
437                   the screen.
438
439 Error Handling:   None.
440
441 Algorithms:      None.
442 Data Structures: None.
443
444 Global Variables: trigger_mode - determines which string is displayed.
445
446 Author:         Glen George
447 Last Modified:  May 9, 2006
448
449 */
450

```

```

451 void display_mode(int x_pos, int y_pos, int style)
452 {
453     /* variables */
454
455     /* the mode strings (must match enumerated type) */
456     const static char * const modes[] = { " Normal ",
457                                           " Automatic",
458                                           " One-Shot " };
459
460
461
462     /* display the trigger mode */
463     plot_string(x_pos, y_pos, modes[trigger_mode], style);
464
465
466     /* all done displaying the trigger mode - return */
467     return;
468 }
469
470
471
472
473
474 /*
475 set_scale
476
477 Description:      This function sets the scale type to the passed value.
478
479 Arguments:      s (enum scale_type) - scale type to which to initialize
480                  the scale status.
481 Return Value:    None.
482
483 Input:          None.
484 Output:         The new trace display is updated with the new scale.
485
486 Error Handling:  None.
487
488 Algorithms:     None.
489 Data Structures: None.
490
491 Global Variables: scale - initialized to the passed value.
492
493 Author:         Glen George
494 Last Modified:   Mar. 13, 1994
495
496 */
497
498 void set_scale(enum scale_type s)
499 {
500     /* variables */
501     /* none */
502
503
504
505     /* set the scale type */
506     scale = s;
507
508     /* output the scale appropriately */
509     set_display_scale(scale);
510
511
512     /* all done setting the scale type - return */
513     return;
514 }
515
516
517
518
519
520 /*
521 scale_down
522
523 Description:      This function handles moving down the list of scale
524                  types. It changes to the "next" type of scale and sets
525                  this as the current scale type.

```

```

526
527 Arguments:      None.
528 Return Value:   None.
529
530 Input:          None.
531 Output:         The new scale is output to the trace display.
532
533 Error Handling:  None.
534
535 Algorithms:     None.
536 Data Structures: None.
537
538 Global Variables: scale - changed to the "next" scale type.
539
540 Author:         Glen George
541 Last Modified:  May 9, 2006
542
543 */
544
545 void scale_down()
546 {
547     /* variables */
548     /* none */
549
550
551
552     /* change to the "next" scale type */
553     if (scale == SCALE_NONE)
554         scale = SCALE_AXES;
555     else if (scale == SCALE_AXES)
556         scale = SCALE_GRID;
557     else
558         scale = SCALE_NONE;
559
560     /* set the scale type */
561     set_display_scale(scale);
562
563
564     /* all done with toggling the scale type - return */
565     return;
566 }
567
568
569
570
571
572 /*
573 scale_up
574
575 Description:      This function handles moving up the list of scale types.
576                  It changes to the "previous" type of scale and sets this
577                  as the current scale type.
578
579 Arguments:        None.
580 Return Value:     None.
581
582 Input:            None.
583 Output:           The new scale is output to the trace display.
584
585 Error Handling:    None.
586
587 Algorithms:        None.
588 Data Structures:   None.
589
590 Global Variables: scale - changed to the "previous" scale type.
591
592 Author:           Glen George
593 Last Modified:    May 9, 2006
594
595 */
596
597 void scale_up()
598 {
599     /* variables */
600     /* none */

```



```

601
602
603
604     /* change to the "previous" scale type */
605     if (scale == SCALE_NONE)
606         scale = SCALE_GRID;
607     else if (scale == SCALE_AXES)
608         scale = SCALE_NONE;
609     else
610         scale = SCALE_AXES;
611
612     /* set the scale type */
613     set_display_scale(scale);
614
615
616     /* all done with toggling the scale type - return */
617     return;
618
619 }
620
621
622
623
624 /*
625     display_scale
626
627     Description:      This function displays the current scale type at the
628                       passed position, in the passed style.
629
630     Arguments:      x_pos (int) - x position (in character cells) at which to
631                       display the scale type.
632                       y_pos (int) - y position (in character cells) at which to
633                       display the scale type.
634                       style (int) - style with which to display the scale type.
635     Return Value:    None.
636
637     Input:          None.
638     Output:         The scale type is displayed at the passed position on the
639                       display.
640
641     Error Handling:  None.
642
643     Algorithms:     None.
644     Data Structures: None.
645
646     Global Variables: scale - determines which string is displayed.
647
648     Author:         Glen George
649     Last Modified:   Mar. 13, 1994
650
651 */
652
653 void display_scale(int x_pos, int y_pos, int style)
654 {
655     /* variables */
656
657     /* the scale type strings (must match enumerated type) */
658     const static char * const scale_stat[] = { " None",
659                                                " Axes",
660                                                " Grid" };
661
662
663
664     /* display the scale status */
665     plot_string(x_pos, y_pos, scale_stat[scale], style);
666
667
668     /* all done displaying the scale status - return */
669     return;
670
671 }
672
673
674
675

```

```

676  /*
677  set_sweep
678
679  Description:      This function sets the sweep rate to the passed value.
680                   The passed value gives the sweep rate to choose from the
681                   list of sweep rates (it gives the list index).
682
683  Arguments:       s (int) - index into the list of sweep rates to which to
684                   set the current sweep rate.
685  Return Value:    None.
686
687  Input:           None.
688  Output:          None.
689
690  Error Handling:   The passed index is not checked for validity.
691
692  Algorithms:      None.
693  Data Structures: None.
694
695  Global Variables: sweep - initialized to the passed value.
696
697  Author:          Glen George
698  Last Modified:   Mar. 8, 1994
699
700  */
701
702  void set_sweep(int s)
703  {
704      /* variables */
705      int sample_size;          /* sample size for this sweep rate */
706
707
708
709      /* set the new sweep rate */
710      sweep = s;
711
712      /* set the sweep rate for the hardware */
713      sample_size = set_sample_rate(sweep_rates[sweep].sample_rate);
714      /* also set the sample size for the trace capture */
715      set_trace_size(sample_size);
716
717
718      /* all done initializing the sweep rate - return */
719      return;
720  }
721
722
723
724
725
726  /*
727  sweep_down
728
729  Description:      This function handles decreasing the current sweep rate.
730                   The new sweep rate (and sample size) is sent to the
731                   hardware (and trace routines). If an attempt is made to
732                   lower the sweep rate below the minimum value it is not
733                   changed. This routine also updates the sweep delay based
734                   on the new sweep rate (to keep the delay time constant).
735
736  Arguments:       None.
737  Return Value:    None.
738
739  Input:           None.
740  Output:          None.
741
742  Error Handling:   None.
743
744  Algorithms:      None.
745  Data Structures: None.
746
747  Global Variables: sweep - decremented if not already 0.
748                   delay - increased to keep delay time constant.
749
750  Known Bugs:      The updated delay time is not displayed. Since the time

```

```

751         is typically only rounded to the new sample rate, this is
752         not a major problem.
753
754     Author:      Glen George
755     Last Modified: Mar. 8, 1994
756
757 */
758
759 void sweep_down()
760 {
761     /* variables */
762     int sample_size;          /* sample size for the new sweep rate */
763
764
765     /* decrease the sweep rate, if not already the minimum */
766     if (sweep > 0) {
767         /* not at minimum, adjust delay for new sweep */
768         adjust_trg_delay(sweep, (sweep - 1));
769         /* now set new sweep rate */
770         sweep--;
771     }
772
773     /* set the sweep rate for the hardware */
774     sample_size = set_sample_rate(sweep_rates[sweep].sample_rate);
775     /* also set the sample size for the trace capture */
776     set_trace_size(sample_size);
777
778
779     /* all done with lowering the sweep rate - return */
780     return;
781
782 }
783
784
785
786
787
788 /*
789 sweep_up
790
791 Description:      This function handles increasing the current sweep rate.
792                   The new sweep rate (and sample size) is sent to the
793                   hardware (and trace routines). If an attempt is made to
794                   raise the sweep rate above the maximum value it is not
795                   changed. This routine also updates the sweep delay based
796                   on the new sweep rate (to keep the delay time constant).
797
798 Arguments:      None.
799 Return Value:   None.
800
801 Input:          None.
802 Output:         None.
803
804 Error Handling:  None.
805
806 Algorithms:     None.
807 Data Structures: None.
808
809 Global Variables: sweep - incremented if not already the maximum value.
810                   delay - decreased to keep delay time constant.
811
812 Known Bugs:      The updated delay time is not displayed. Since the time
813                   is typically only rounded to the new sample rate, this is
814                   not a major problem.
815
816 Author:      Glen George
817 Last Modified: Mar. 8, 1994
818
819 */
820
821 void sweep_up()
822 {
823     /* variables */
824     int sample_size;          /* sample size for the new sweep rate */
825

```

```

826
827
828     /* increase the sweep rate, if not already the maximum */
829     if (sweep < (NO_SWEEP_RATES - 1)) {
830         /* not at maximum, adjust delay for new sweep */
831         adjust_trg_delay(sweep, (sweep + 1));
832         /* now set new sweep rate */
833         sweep++;
834     }
835
836     /* set the sweep rate for the hardware */
837     sample_size = set_sample_rate(sweep_rates[sweep].sample_rate);
838     /* also set the sample size for the trace capture */
839     set_trace_size(sample_size);
840
841
842     /* all done with raising the sweep rate - return */
843     return;
844 }
845
846
847
848
849
850 /*
851 display_sweep
852
853 Description:      This function displays the current sweep rate at the
854                   passed position, in the passed style.
855
856 Arguments:      x_pos (int) - x position (in character cells) at which to
857                   display the sweep rate.
858                   y_pos (int) - y position (in character cells) at which to
859                   display the sweep rate.
860                   style (int) - style with which to display the sweep rate.
861 Return Value:    None.
862
863 Input:          None.
864 Output:         The sweep rate is displayed at the passed position on the
865                   display.
866
867 Error Handling:  None.
868
869 Algorithms:     None.
870 Data Structures: None.
871
872 Global Variables: sweep - determines which string is displayed.
873
874 Author:         Glen George
875 Last Modified:  Mar. 8, 1994
876
877 */
878
879 void display_sweep(int x_pos, int y_pos, int style)
880 {
881     /* variables */
882     /* none */
883
884
885
886     /* display the sweep rate */
887     plot_string(x_pos, y_pos, sweep_rates[sweep].s, style);
888
889
890     /* all done displaying the sweep rate - return */
891     return;
892 }
893
894
895
896
897
898 /*
899 set_trg_level
900

```

```

901 Description:      This function sets the trigger level to the passed value.
902
903 Arguments:      1 (int) - value to which to set the trigger level.
904 Return Value:   None.
905
906 Input:          None.
907 Output:         None.
908
909 Error Handling:  The passed value is not checked for validity.
910
911 Algorithms:     None.
912 Data Structures: None.
913
914 Global Variables: level - initialized to the passed value.
915
916 Author:         Glen George
917 Last Modified:  Mar. 8, 1994
918
919 */
920
921 void set_trg_level(int l)
922 {
923     /* variables */
924     /* none */
925
926
927     /* set the trigger level */
928     level = l;
929
930
931     /* set the trigger level in hardware too */
932     set_trigger(level, slope);
933
934
935     /* all done initializing the trigger level - return */
936     return;
937 }
938
939
940
941
942
943 /*
944 trg_level_down
945
946 Description:      This function handles decreasing the current trigger
947                   level. The new trigger level is sent to the hardware.
948                   If an attempt is made to lower the trigger level below
949                   the minimum value it is not changed.
950
951 Arguments:        None.
952 Return Value:     None.
953
954 Input:            None.
955 Output:           None.
956
957 Error Handling:    None.
958
959 Algorithms:       None.
960 Data Structures:  None.
961
962 Global Variables: level - decremented if not already at the minimum value.
963
964 Author:           Glen George
965 Last Modified:    Mar. 8, 1994
966
967 */
968
969 void trg_level_down()
970 {
971     /* variables */
972     /* none */
973
974
975

```

```

976     /* decrease the trigger level, if not already the minimum */
977     if (level > MIN_TRG_LEVEL_SET)
978         level--;
979
980     /* set the trigger level for the hardware */
981     set_trigger(level, slope);
982
983
984     /* all done with lowering the trigger level - return */
985     return;
986
987 }
988
989
990
991
992 /*
993     trg_level_up
994
995     Description:      This function handles increasing the current trigger
996                       level.  The new trigger level is sent to the hardware.
997                       If an attempt is made to raise the trigger level above
998                       the maximum value it is not changed.
999
1000     Arguments:       None.
1001     Return Value:    None.
1002
1003     Input:           None.
1004     Output:          None.
1005
1006     Error Handling:   None.
1007
1008     Algorithms:      None.
1009     Data Structures: None.
1010
1011     Global Variables: level - incremented if not already the maximum value.
1012
1013     Author:          Glen George
1014     Last Modified:    Mar. 8, 1994
1015 */
1016
1017 void trg_level_up()
1018 {
1019     /* variables */
1020     /* none */
1021
1022
1023
1024
1025     /* increase the trigger level, if not already the maximum */
1026     if (level < MAX_TRG_LEVEL_SET)
1027         level++;
1028
1029     /* tell the hardware the new trigger level */
1030     set_trigger(level, slope);
1031
1032
1033     /* all done raising the trigger level - return */
1034     return;
1035
1036 }
1037
1038
1039
1040
1041 /*
1042     display_trg_level
1043
1044     Description:      This function displays the current trigger level at the
1045                       passed position, in the passed style.
1046
1047     Arguments:       x_pos (int) - x position (in character cells) at which to
1048                       display the trigger level.
1049                       y_pos (int) - y position (in character cells) at which to
1050                       display the trigger level.

```

```

1051         style (int) - style with which to display the trigger
1052             level.
1053 Return Value:     None.
1054
1055 Input:           None.
1056 Output:          The trigger level is displayed at the passed position on
1057                 the display.
1058
1059 Error Handling:   None.
1060
1061 Algorithms:      None.
1062 Data Structures: None.
1063
1064 Global Variables: level - determines the value displayed.
1065
1066 Author:          Glen George
1067 Last Modified:   Mar. 10, 1995
1068
1069 */
1070
1071 void display_trg_level(int x_pos, int y_pos, int style)
1072 {
1073     /* variables */
1074     char    level_str[] = "          "; /* string containing the trigger level */
1075     long int l;          /* trigger level in mV */
1076
1077
1078     /* compute the trigger level in millivolts */
1079     l = ((long int) MAX_LEVEL - MIN_LEVEL) * level / (MAX_TRG_LEVEL_SET - MIN_TRG_LEVEL_SET) + MIN_LEV
1080
1081     /* convert the level to the string (leave first character blank) */
1082     cvt_num_field(l, &level_str[1]);
1083
1084     /* add in the units */
1085     level_str[7] = 'V';
1086
1087
1088     /* now finally display the trigger level */
1089     plot_string(x_pos, y_pos, level_str, style);
1090
1091
1092     /* all done displaying the trigger level - return */
1093     return;
1094
1095 }
1096
1097
1098
1099
1100
1101 /*
1102 set_trg_slope
1103
1104 Description:      This function sets the trigger slope to the passed value.
1105
1106 Arguments:        s (enum slope_type) - trigger slope type to which to set
1107                 the locally global slope.
1108 Return Value:     None.
1109
1110 Input:           None.
1111 Output:          None.
1112
1113 Error Handling:   None.
1114
1115 Algorithms:      None.
1116 Data Structures: None.
1117
1118 Global Variables: slope - set to the passed value.
1119
1120 Author:          Glen George
1121 Last Modified:   Mar. 8, 1994
1122
1123 */
1124
1125 void set_trg_slope(enum slope_type s)

```

```

1126 {
1127     /* variables */
1128     /* none */
1129
1130
1131
1132     /* set the slope type */
1133     slope = s;
1134
1135     /* also tell the hardware what the slope is */
1136     set_trigger(level, slope);
1137
1138
1139     /* all done setting the trigger slope - return */
1140     return;
1141 }
1142
1143
1144
1145
1146
1147 /*
1148     trg_slope_toggle
1149
1150     Description:      This function handles toggling (and setting) the current
1151                      trigger slope.
1152
1153     Arguments:       None.
1154     Return Value:    None.
1155
1156     Input:           None.
1157     Output:          None.
1158
1159     Error Handling:   None.
1160
1161     Algorithms:       None.
1162     Data Structures:  None.
1163
1164     Global Variables: slope - toggled.
1165
1166     Author:          Glen George
1167     Last Modified:   Mar. 8, 1994
1168 */
1169
1170
1171 void trg_slope_toggle()
1172 {
1173     /* variables */
1174     /* none */
1175
1176
1177
1178     /* toggle the trigger slope */
1179     if (slope == SLOPE_POSITIVE)
1180         slope = SLOPE_NEGATIVE;
1181     else
1182         slope = SLOPE_POSITIVE;
1183
1184     /* set the new trigger slope */
1185     set_trigger(level, slope);
1186
1187
1188     /* all done with the trigger slope - return */
1189     return;
1190 }
1191
1192
1193
1194
1195
1196 /*
1197     display_trg_slope
1198
1199     Description:      This function displays the current trigger slope at the
1200                      passed position, in the passed style.

```



```

1201
1202 Arguments:      x_pos (int) - x position (in character cells) at which to
1203                  display the trigger slope.
1204                  y_pos (int) - y position (in character cells) at which to
1205                  display the trigger slope.
1206                  style (int) - style with which to display the trigger
1207                  slope.
1208 Return Value:    None.
1209
1210 Input:           None.
1211 Output:          The trigger slope is displayed at the passed position on
1212                  the screen.
1213
1214 Error Handling:   None.
1215
1216 Algorithms:      None.
1217 Data Structures: None.
1218
1219 Global Variables: slope - determines which string is displayed.
1220
1221 Author:          Glen George
1222 Last Modified:   Mar. 13, 1994
1223
1224 */
1225
1226 void display_trg_slope(int x_pos, int y_pos, int style)
1227 {
1228     /* variables */
1229
1230     /* the trigger slope strings (must match enumerated type) */
1231     const static char * const slopes[] = { " +", " -" };
1232
1233
1234
1235     /* display the trigger slope */
1236     plot_string(x_pos, y_pos, slopes[slope], style);
1237
1238
1239     /* all done displaying the trigger slope - return */
1240     return;
1241 }
1242
1243
1244
1245
1246
1247 /*
1248 set_trg_delay
1249
1250 Description:      This function sets the trigger delay to the passed value.
1251
1252 Arguments:        d (long int) - value to which to set the trigger delay.
1253 Return Value:     None.
1254
1255 Input:            None.
1256 Output:           None.
1257
1258 Error Handling:    The passed value is not checked for validity.
1259
1260 Algorithms:        None.
1261 Data Structures:   None.
1262
1263 Global Variables:  delay - initialized to the passed value.
1264
1265 Author:           Glen George
1266 Last Modified:    Mar. 8, 1994
1267
1268 */
1269
1270 void set_trg_delay(long int d)
1271 {
1272     /* variables */
1273     /* none */
1274
1275

```

```

1276
1277     /* set the trigger delay */
1278     delay = d;
1279
1280     /* set the trigger delay in hardware too */
1281     set_delay(delay);
1282
1283
1284     /* all done initializing the trigger delay - return */
1285     return;
1286
1287 }
1288
1289
1290
1291
1292 /*
1293     trg_delay_down
1294
1295     Description:      This function handles decreasing the current trigger
1296                      delay.  The new trigger delay is sent to the hardware.
1297                      If an attempt is made to lower the trigger delay below
1298                      the minimum value it is not changed.
1299
1300     Arguments:       None.
1301     Return Value:    None.
1302
1303     Input:           None.
1304     Output:          None.
1305
1306     Error Handling:  None.
1307
1308     Algorithms:      None.
1309     Data Structures: None.
1310
1311     Global Variables: delay - decremented if not already at the minimum value.
1312
1313     Author:          Glen George
1314     Last Modified:   Mar. 8, 1994
1315 */
1316
1317 void trg_delay_down()
1318 {
1319     /* variables */
1320     /* none */
1321
1322
1323
1324
1325     /* decrease the trigger delay, if not already the minimum */
1326     if (delay > MIN_DELAY)
1327         delay--;
1328
1329     /* set the trigger delay for the hardware */
1330     set_delay(delay);
1331
1332
1333     /* all done with lowering the trigger delay - return */
1334     return;
1335
1336 }
1337
1338
1339
1340
1341 /*
1342     trg_delay_up
1343
1344     Description:      This function handles increasing the current trigger
1345                      delay.  The new trigger delay is sent to the hardware.
1346                      If an attempt is made to raise the trigger delay above
1347                      the maximum value it is not changed.
1348
1349     Arguments:       None.
1350     Return Value:    None.

```

```

1351
1352     Input:          None.
1353     Output:         None.
1354
1355     Error Handling:  None.
1356
1357     Algorithms:      None.
1358     Data Structures: None.
1359
1360     Global Variables: delay - incremented if not already the maximum value.
1361
1362     Author:          Glen George
1363     Last Modified:   Mar. 8, 1994
1364
1365 */
1366
1367 void trg_delay_up()
1368 {
1369     /* variables */
1370     /* none */
1371
1372
1373
1374     /* increase the trigger delay, if not already the maximum */
1375     if (delay < MAX_DELAY)
1376         delay++;
1377
1378     /* tell the hardware the new trigger delay */
1379     set_delay(delay);
1380
1381
1382     /* all done raising the trigger delay - return */
1383     return;
1384
1385 }
1386
1387
1388
1389
1390 /*
1391     adjust_trg_delay
1392
1393     Description:      This function adjusts the trigger delay for a new sweep
1394                       rate. The factor to adjust the delay by is determined
1395                       by looking up the sample rates in the sweep_rates array.
1396                       If the delay goes out of range, due to the adjustment it
1397                       is reset to the maximum or minimum valid value.
1398
1399     Arguments:        old_sweep (int) - old sweep rate (index into sweep_rates
1400                               array).
1401                       new_sweep (int) - new sweep rate (index into sweep_rates
1402                               array).
1403     Return Value:      None.
1404
1405     Input:             None.
1406     Output:            None.
1407
1408     Error Handling:    None.
1409
1410     Algorithms:        The delay is multiplied by 10 times the ratio of the
1411                       sweep sample rates then divided by 10. This is done to
1412                       avoid floating point arithmetic and integer truncation
1413                       problems.
1414     Data Structures:   None.
1415
1416     Global Variables:  delay - adjusted based on passed sweep rates.
1417
1418     Known Bugs:        The updated delay time is not displayed. Since the time
1419                       is typically only rounded to the new sample rate, this is
1420                       not a major problem.
1421
1422     Author:            Glen George
1423     Last Modified:     Mar. 8, 1994
1424
1425 */

```

```

1426
1427 static void adjust_trg_delay(int old_sweep, int new_sweep)
1428 {
1429     /* variables */
1430     /* none */
1431
1432
1433
1434     /* multiply by 10 times the ratio of sweep rates */
1435     delay *= (10 * sweep_rates[new_sweep].sample_rate) / sweep_rates[old_sweep].sample_rate;
1436     /* now divide the factor of 10 back out */
1437     delay /= 10;
1438
1439     /* make sure delay is not out of range */
1440     if (delay > MAX_DELAY)
1441         /* delay is too large - set to maximum */
1442         delay = MAX_DELAY;
1443     if (delay < MIN_DELAY)
1444         /* delay is too small - set to minimum */
1445         delay = MIN_DELAY;
1446
1447
1448     /* tell the hardware the new trigger delay */
1449     set_delay(delay);
1450
1451
1452     /* all done adjusting the trigger delay - return */
1453     return;
1454 }
1455
1456
1457
1458
1459
1460 /*
1461 display_trg_delay
1462
1463 Description:      This function displays the current trigger delay at the
1464                  passed position, in the passed style.
1465
1466 Arguments:      x_pos (int) - x position (in character cells) at which to
1467                  display the trigger delay.
1468                  y_pos (int) - y position (in character cells) at which to
1469                  display the trigger delay.
1470                  style (int) - style with which to display the trigger
1471                  delay.
1472 Return Value:    None.
1473
1474 Input:           None.
1475 Output:          The trigger delay is displayed at the passed position on
1476                  the display.
1477
1478 Error Handling:  None.
1479
1480 Algorithms:      None.
1481 Data Structures: None.
1482
1483 Global Variables: delay - determines the value displayed.
1484
1485 Author:          Glen George
1486 Last Modified:   June 11, 2014
1487
1488 */
1489
1490 void display_trg_delay(int x_pos, int y_pos, int style)
1491 {
1492     /* variables */
1493     char    delay_str[] = "          "; /* string containing the trigger delay */
1494     long int units_adj; /* adjustment to get to microseconds */
1495
1496     long int d; /* delay in appropriate units */
1497     float    temp_d; /* delay in float to avoid overflows */
1498
1499     /* compute the delay in the appropriate units */
1500     /* have to watch out for overflow, so use float temp */

```

```

1501     if (sweep_rates[sweep].sample_rate > 1000000L) {
1502         /* have a fast sweep rate */
1503         /* first compute with float to avoid overflow */
1504         temp_d = delay * (1000000000L / sweep_rates[sweep].sample_rate);
1505
1506         /* now convert to int */
1507         d = (int) temp_d;
1508         /* need to divide by 1000 to get to microseconds */
1509         units_adj = 1000;
1510     }
1511     else {
1512         /* slow sweep rate, don't have to worry about overflow */
1513         d = delay * (1000000L / sweep_rates[sweep].sample_rate);
1514         /* already in microseconds, so adjustment is 1 */
1515         units_adj = 1;
1516     }
1517
1518     /* convert it to the string (leave first character blank) */
1519     cvt_num_field(d, &delay_str[1]);
1520
1521     /* add in the units */
1522     if (((d / units_adj) < 1000) && ((d / units_adj) > -1000) && (units_adj == 1000)) {
1523         /* delay is in microseconds */
1524         delay_str[7] = '\004';
1525         delay_str[8] = 's';
1526     }
1527     else if (((d / units_adj) < 1000000) && ((d / units_adj) > -1000000)) {
1528         /* delay is in milliseconds */
1529         delay_str[7] = 'm';
1530         delay_str[8] = 's';
1531     }
1532     else if (((d / units_adj) < 1000000000) && ((d / units_adj) > -1000000000)) {
1533         /* delay is in seconds */
1534         delay_str[7] = 's';
1535         delay_str[8] = ' ';
1536     }
1537     else {
1538         /* delay is in kiloseconds */
1539         delay_str[7] = 'k';
1540         delay_str[8] = 's';
1541     }
1542
1543
1544     /* now actually display the trigger delay */
1545     plot_string(x_pos, y_pos, delay_str, style);
1546
1547
1548     /* all done displaying the trigger delay - return */
1549     return;
1550 }
1551
1552
1553
1554
1555
1556 /*
1557 cvt_num_field
1558
1559 Description:      This function converts the passed number (numeric field
1560                   value) to a string and returns that in the passed string
1561                   reference. The number may be signed, and a sign (+ or -)
1562                   is always generated. The number is assumed to have three
1563                   digits to the right of the decimal point. Only the four
1564                   most significant digits of the number are displayed and
1565                   the decimal point is shifted appropriately. (Four digits
1566                   are always generated by the function).
1567
1568 Arguments:      n (long int) - numeric field value to convert.
1569                 s (char *) - pointer to string in which to return the
1570                           converted field value.
1571
1572 Return Value:    None.
1573
1574 Input:          None.
1575 Output:         None.
1576
1577

```

```

1576 Error Handling:    None.
1577
1578 Algorithms:        The algorithm used assumes four (4) digits are being
1579                    converted.
1580 Data Structures:    None.
1581
1582 Global Variables:   None.
1583
1584 Known Bugs:         If the passed long int is the largest negative long int,
1585                    the function will display garbage.
1586
1587 Author:             Glen George
1588 Last Modified:      Mar. 8, 1994
1589
1590 */
1591
1592 static void  cvt_num_field(long int n, char *s)
1593 {
1594     /* variables */
1595     int  dp = 3;          /* digits to right of decimal point */
1596     int  d;              /* digit weight (power of 10) */
1597
1598     int  i = 0;          /* string index */
1599
1600
1601
1602     /* first get the sign (and make n positive for conversion) */
1603     if (n < 0) {
1604         /* n is negative, set sign and convert to positive */
1605         s[i++] = '-';
1606         n = -n;
1607     }
1608     else {
1609         /* n is positive, set sign only */
1610         s[i++] = '+';
1611     }
1612
1613
1614     /* make sure there are no more than 4 significant digits */
1615     while (n > 9999) {
1616         /* have more than 4 digits - get rid of one */
1617         n /= 10;
1618         /* adjust the decimal point */
1619         dp--;
1620     }
1621
1622     /* if decimal point is non-positive, make positive */
1623     /* (assume will take care of adjustment with output units in this case) */
1624     while (dp <= 0)
1625         dp += 3;
1626
1627
1628     /* adjust dp to be digits to the right of the decimal point */
1629     /* (assuming 4 digits) */
1630     dp = 4 - dp;
1631
1632
1633     /* finally, loop getting and converting digits */
1634     for (d = 1000; d > 0; d /= 10) {
1635
1636         /* check if need decimal the decimal point now */
1637         if (dp-- == 0)
1638             /* time for decimal point */
1639             s[i++] = '.';
1640
1641         /* get and convert this digit */
1642         s[i++] = (n / d) + '0';
1643         /* remove this digit from n */
1644         n %= d;
1645     }
1646
1647
1648     /* all done converting the number, return */
1649     return;
1650

```

1651 }  
1652

```

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_scale(enum scale_type); /* set the scale type */
112 void set_sweep(int); /* set the sweep rate */
113 void set_trg_level(int); /* set the trigger level */
114 void set_trg_slope(enum slope_type); /* set the trigger slope */
115 void set_trg_delay(long int); /* set the trigger delay */
116
117 /* option display routines */
118 void no_display(int, int, int); /* no option setting to display */
119 void display_mode(int, int, int); /* display trigger mode */
120 void display_scale(int, int, int); /* display the scale type */
121 void display_sweep(int, int, int); /* display the sweep rate */
122 void display_trg_level(int, int, int); /* display the trigger level */
123 void display_trg_slope(int, int, int); /* display the trigger slope */
124 void display_trg_delay(int, int, int); /* display the trigger delay */
125
126
127 #endif
128

```

```

1  /*****
2  /*
3  /*          SCOPEDEF.H          */
4  /*          General Definitions  */
5  /*          Include File        */
6  /*          Digital Oscilloscope Project
7  /*          EE/CS 52            */
8  /*
9  /*****/
10
11 /*
12  This file contains the general definitions for the Digital Oscilloscope
13  project. This includes constant and structure definitions along with the
14  function declarations for the assembly language functions.
15
16
17  Revision History:
18      3/8/94   Glen George      Initial revision.
19      3/13/94  Glen George      Updated comments.
20      3/17/97  Glen George      Removed KEYCODE_UNUSED (no longer used).
21      5/3/06   Glen George      Added conditional definitions for handling
22                                different architectures.
23      5/9/06   Glen George      Updated declaration of start_sample() to
24                                match the new specification.
25      5/27/08  Glen George      Added check for __nios__ definition to also
26                                indicate the compilation is for an Altera
27                                NIOS CPU.
28      6/03/14  Santiago Navonne Added cursor text area, and NO_TRACE value.
29  */
30
31
32
33 #ifndef __SCOPEDEF_H
34 #define __SCOPEDEF_H
35
36
37 /* library include files */
38 /* none */
39
40 /* local include files */
41 #include "interfac.h"
42 #include "lcdout.h"
43
44
45
46
47 /* constants */
48
49 /* general constants */
50 #define FALSE      0
51 #define TRUE       !FALSE
52 #define NULL       (void *) 0
53
54 /* display size (in characters) */
55 #define LCD_WIDTH   (SIZE_X / HORIZ_SIZE)
56 #define LCD_HEIGHT  (SIZE_Y / VERT_SIZE)
57
58 /* cursor area */
59 #define CURSOR_STR_X      5
60 #define CURSOR_STR_Y      5
61 #define CURSOR_STR_W     100
62 #define CURSOR_STR_H      7
63
64
65
66 /* macros */
67
68 /* let __nios__ also mean a NIOS compilation */
69 #ifdef __nios__
70 #define NIOS          /* use the standard NIOS definition */
71 #endif
72
73 /* add the definitions necessary for the Altera NIOS chip */
74 #ifdef NIOS
75 #define FLAT_MEMORY    /* use the flat memory model */

```

```

76 #endif
77
78
79 /* if a flat memory model don't need far pointers */
80 #ifdef FLAT_MEMORY
81     #define far
82 #endif
83
84
85
86
87 /* structures, unions, and typedefs */
88
89 /* program states */
90 enum status { MENU_ON, /* menu is displayed with the cursor in it */
91             MENU_OFF, /* menu is not displayed - no cursor */
92             NUM_STATES /* number of states */
93             };
94
95 /* key codes */
96 enum keycode { KEYCODE_MENU, /* <Menu> */
97              KEYCODE_UP, /* <Up> */
98              KEYCODE_DOWN, /* <Down> */
99              KEYCODE_LEFT, /* <Left> */
100             KEYCODE_RIGHT, /* <Right> */
101             KEYCODE_ILLEGAL, /* other keys */
102             NUM_KEYCODES /* number of key codes */
103             };
104
105
106
107
108 /* function declarations */
109
110 /* keypad functions */
111 unsigned char key_available(void); /* key is available */
112 int getkey(void); /* get a key */
113
114 /* display functions */
115 void clear_display(void); /* clear the display */
116 void plot_pixel(unsigned int, unsigned int, int); /* output a pixel */
117
118 /* sampling parameter functions */
119 int set_sample_rate(long int); /* set the sample rate */
120 void set_trigger(int, int); /* set trigger level and slope */
121 void set_delay(long int); /* set the trigger delay time */
122
123 /* sampling functions */
124 void start_sample(int); /* capture a sample */
125 unsigned char *sample_done(void); /* sample captured status */
126
127
128 #endif
129

```

```

1  /*****
2  /*
3  /*          TRACUTIL
4  /*          Trace Utility Functions
5  /*          Digital Oscilloscope Project
6  /*          EE/CS 52
7  /*
8  /*****/
9
10 /*
11 This file contains the utility functions for handling traces (capturing
12 and displaying data) for the Digital Oscilloscope project. The functions
13 included are:
14     clear_saved_areas - clear all the save areas
15     do_trace          - start a trace
16     init_trace        - initialize the trace routines
17     plot_trace        - plot a trace (sampled data)
18     restore_menu_trace - restore the saved area under the menus
19     restore_trace     - restore the saved area of a trace
20     set_display_scale - set the type of displayed scale (and display it)
21     set_mode          - set the triggering mode
22     set_save_area     - determine an area of a trace to save
23     set_trace_size    - set the number of samples in a trace
24     trace_done        - inform this module that a trace has been completed
25     trace_rdy         - determine if system is ready to start another trace
26     trace_rearm       - re-enable tracing (in one-shot triggering mode)
27
28 The local functions included are:
29     none
30
31 The locally global variable definitions included are:
32     cur_scale - current scale type
33     sample_size - the size of the sample for the trace
34     sampling - currently doing a sample
35     saved_area - saved trace under a specified area
36     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 coordinate) of area to save
40     saved_pos_y - starting position (y coordinate) of area to save
41     saved_end_x - ending position (x coordinate) of area to save
42     saved_end_y - ending position (y coordinate) 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     6/3/14    Santiago Navonne Changed UI display colors; changed plot_trace
88                                     to clear just trace instead of whole display.
89 */
90
91
92
93 /* library include files */
94 /* none */
95
96 /* local include files */
97 #include "scopedef.h"
98 #include "lcdout.h"
99 #include "menu.h"
100 #include "menuact.h"
101 #include "tracutil.h"
102
103
104
105
106 /* locally global variables */
107
108 static int  trace_status;    /* ready to start another trace */
109
110 static int  sampling;        /* currently sampling data */
111
112 static int  sample_size;     /* number of data points in a sample */
113
114 static int  old_sample[SIZE_X]; /* sample currently being displayed */
115
116 static enum scale_type  cur_scale; /* current display scale type */
117
118 /* traces (sampled data) saved under the axes */
119 static unsigned char  saved_axis_x[2 * Y_TICK_CNT + 1][PLOT_SIZE_X/8]; /* saved trace under x lines */
120 static unsigned char  saved_axis_y[2 * X_TICK_CNT + 1][PLOT_SIZE_Y/8]; /* saved trace under y lines */
121
122 /* traces (sampled data) saved under the menu */
123 static unsigned char  saved_menu[MENU_SIZE_Y][(MENU_SIZE_X + 7)/8];
124
125 /* traces (sampled data) saved under any area */
126 static unsigned char  saved_area[SAVE_SIZE_Y][SAVE_SIZE_X/8]; /* saved trace under any area */
127 static int  saved_pos_x;    /* starting x position of saved area */
128 static int  saved_pos_y;    /* starting y position of saved area */
129 static int  saved_end_x;    /* ending x position of saved area */
130 static int  saved_end_y;    /* ending y position of saved area */
131
132
133
134
135 /*
136     init_trace
137
138     Description:      This function initializes all of the locally global
139                       variables used by these routines. The saved areas are
140                       set to non-existent with cleared saved data. Normal
141                       normal triggering is set, the system is ready for a
142                       trace, the scale is turned off and the sample size is set
143                       to the screen size.
144
145     Arguments:        None.
146     Return Value:     None.
147
148     Input:            None.
149     Output:           None.
150

```

```

151 Error Handling:    None.
152
153 Algorithms:       None.
154 Data Structures:  None.
155
156 Global Variables: trace_status - set to TRUE.
157                 sampling      - set to FALSE.
158                 cur_scale     - set to SCALE_NONE (no displayed scale).
159                 sample_size   - set to screen size (SIZE_X).
160                 saved_axis_x  - cleared.
161                 saved_axis_y  - cleared.
162                 saved_menu    - cleared.
163                 saved_area    - cleared.
164                 saved_pos_x   - set to off-screen.
165                 saved_pos_y   - set to off-screen.
166                 saved_end_x   - set to off-screen.
167                 saved_end_y   - set to off-screen.
168
169 Author:           Glen George
170 Last Modified:    May 9, 2006
171
172 */
173
174 void init_trace()
175 {
176     /* variables */
177     /* none */
178
179
180
181     /* initialize system status variables */
182
183     /* ready for a trace */
184     trace_status = TRUE;
185
186     /* not currently sampling data */
187     sampling = FALSE;
188
189     /* turn off the displayed scale */
190     cur_scale = SCALE_NONE;
191
192     /* sample size is the screen size */
193     sample_size = SIZE_X;
194
195
196     /* clear save areas */
197     clear_saved_areas();
198
199     /* also clear the general saved area location variables (off-screen) */
200     saved_pos_x = SIZE_X + 1;
201     saved_pos_y = SIZE_Y + 1;
202     saved_end_x = SIZE_X + 1;
203     saved_end_y = SIZE_Y + 1;
204
205
206     /* done initializing, return */
207     return;
208 }
209
210
211
212
213
214 /*
215 set_mode
216
217 Description:      This function sets the locally global triggering mode
218                  based on the passed value (one of the possible enumerated
219                  values). The triggering mode is used to determine when
220                  the system is ready for another trace. The sampling flag
221                  is also reset so a new sample will be started (if that is
222                  appropriate).
223
224 Arguments:        trigger_mode (enum trigger_type) - the mode with which to
225                  set the triggering.

```

```

226 Return Value:      None.
227
228 Input:             None.
229 Output:            None.
230
231 Error Handling:     None.
232
233 Algorithms:         None.
234 Data Structures:    None.
235
236 Global Variables:  sampling      - set to FALSE to turn off sampling
237                   trace_status - set to TRUE if not one-shot triggering.
238
239 Author:            Glen George
240 Last Modified:     May 27, 2008
241
242 */
243
244 void set_mode(enum trigger_type trigger_mode)
245 {
246     /* variables */
247     /* none */
248
249
250
251     /* if not one-shot triggering - ready for trace too */
252     trace_status = (trigger_mode != ONESHOT_TRIGGER);
253
254
255     /* turn off the sampling flag so will start a new sample */
256     sampling = FALSE;
257
258
259     /* all done, return */
260     return;
261 }
262
263
264
265
266
267 /*
268 is_sampling
269
270 Description:      This function determines whether the system is currently
271                  taking a sample or not. This is just the value of the
272                  sampling flag.
273
274 Arguments:        None.
275 Return Value:     (int) - the current sampling status (TRUE if currently
276                  trying to take a sample, FALSE otherwise).
277
278 Input:            None.
279 Output:           None.
280
281 Error Handling:    None.
282
283 Algorithms:        None.
284 Data Structures:   None.
285
286 Global Variables:  sampling - determines if taking a sample or not.
287
288 Author:            Glen George
289 Last Modified:     May 27, 2008
290
291 */
292
293 int is_sampling()
294 {
295     /* variables */
296     /* none */
297
298
299
300     /* currently sampling if sampling flag is set */

```

```

301     return  sampling;
302
303 }
304
305
306
307
308 /*
309     trace_rdy
310
311     Description:      This function determines whether the system is ready to
312                      start another trace.  This is determined by whether or
313                      not the system is still sampling (sampling flag) and if
314                      it is ready for another trace (trace_status flag).
315
316     Arguments:       None.
317     Return Value:    (int) - the current trace status (TRUE if ready to do
318                      another trace, FALSE otherwise).
319
320     Input:           None.
321     Output:          None.
322
323     Error Handling:  None.
324
325     Algorithms:      None.
326     Data Structures: None.
327
328     Global Variables: sampling      - determines if ready for another trace.
329                      trace_status - determines if ready for another trace.
330
331     Author:          Glen George
332     Last Modified:   Mar. 13, 1994
333
334 */
335
336 int  trace_rdy()
337 {
338     /* variables */
339     /* none */
340
341
342
343     /* ready for another trace if not sampling and trace is ready */
344     return  (!sampling && trace_status);
345
346 }
347
348
349
350
351 /*
352     trace_done
353
354     Description:      This function is called to indicate a trace has been
355                      completed.  If in normal triggering mode this means the
356                      system is ready for another trace.
357
358     Arguments:       None.
359     Return Value:    None.
360
361     Input:           None.
362     Output:          None.
363
364     Error Handling:  None.
365
366     Algorithms:      None.
367     Data Structures: None.
368
369     Global Variables: trace_status - may be set to TRUE.
370                      sampling      - set to FALSE.
371
372     Author:          Glen George
373     Last Modified:   May 9, 2006
374
375 */

```



```

376
377 void trace_done()
378 {
379     /* variables */
380     /* none */
381
382
383
384     /* done with a trace - if retriggering, ready for another one */
385     if (get_trigger_mode() != ONESHOT_TRIGGER)
386         /* in a retriggering mode - set trace_status to TRUE (ready) */
387         trace_status = TRUE;
388
389     /* no longer sampling data */
390     sampling = FALSE;
391
392
393     /* done so return */
394     return;
395
396 }
397
398
399
400
401 /*
402 trace_rearm
403
404 Description:      This function is called to rearm the trace. It sets the
405                  trace status to ready (TRUE). It is used to rearm the
406                  trigger in one-shot mode.
407
408 Arguments:      None.
409 Return Value:   None.
410
411 Input:          None.
412 Output:         None.
413
414 Error Handling:  None.
415
416 Algorithms:     None.
417 Data Structures: None.
418
419 Global Variables: trace_status - set to TRUE.
420
421 Author:         Glen George
422 Last Modified:  Mar. 8, 1994
423
424 */
425
426 void trace_rearm()
427 {
428     /* variables */
429     /* none */
430
431
432
433     /* rearm the trace - set status to ready (TRUE) */
434     trace_status = TRUE;
435
436
437     /* all done - return */
438     return;
439
440 }
441
442
443
444
445 /*
446 set_trace_size
447
448 Description:      This function sets the locally global sample size to the
449                  passed value. This is used to scale the data when
450                  plotting a trace.

```

```

451
452 Arguments:      size (int) - the trace sample size.
453 Return Value:   None.
454
455 Input:          None.
456 Output:         None.
457
458 Error Handling:  None.
459
460 Algorithms:      None.
461 Data Structures: None.
462
463 Global Variables: sample_size - set to the passed value.
464
465 Author:          Glen George
466 Last Modified:   Mar. 8, 1994
467
468 */
469
470 void set_trace_size(int size)
471 {
472     /* variables */
473     /* none */
474
475
476
477     /* set the locally global sample size */
478     sample_size = size;
479
480
481     /* all done, return */
482     return;
483 }
484
485
486
487
488
489 /*
490 set_display_scale
491
492 Description:      This function sets the displayed scale type to the passed
493                   argument.  If the scale is turned on, it draws it.  If it
494                   is turned off (SCALE_NONE), it restores the saved trace
495                   under the scale.  Scales can be axes with tick marks
496                   (SCALE_AXES) or a grid (SCALE_GRID).
497
498 Arguments:        scale (scale_type) - new scale type.
499 Return Value:     None.
500
501 Input:            None.
502 Output:           Either a scale is output or the trace under the old scale
503                   is restored.
504
505 Error Handling:    None.
506
507 Algorithms:        None.
508 Data Structures:   None.
509
510 Global Variables:  cur_scale - set to the passed value.
511                   saved_axis_x - used to restore trace data under x-axis.
512                   saved_axis_y - used to restore trace data under y-axis.
513
514 Author:           Glen George
515 Last Modified:    June 03, 2014
516
517 */
518
519 void set_display_scale(enum scale_type scale)
520 {
521     /* variables */
522     int p;          /* x or y coordinate */
523
524     int i;          /* loop indices */
525     int j;

```

```

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

```

```

601         if (scale == SCALE_GRID)
602             /* drawing a grid line */
603             plot_hline(X_GRID_START, p, (X_GRID_END - X_GRID_START));
604         else if (i == 0)
605             /* drawing the x axis */
606             plot_hline(X_AXIS_START, p, (X_AXIS_END - X_AXIS_START));
607         else
608             /* must be drawing a tick mark */
609             plot_hline((Y_AXIS_POS - (TICK_LEN / 2)), p, TICK_LEN);
610     }
611
612     /* draw y lines (grid or tick marks) */
613     for (i = -X_TICK_CNT; i <= X_TICK_CNT; i++) {
614
615         /* get x position of the line */
616         p = Y_AXIS_POS + i * X_TICK_SIZE;
617         /* make sure it is in range */
618         if (p >= PLOT_SIZE_X)
619             p = PLOT_SIZE_X - 1;
620         if (p < 0)
621             p = 0;
622
623         /* should we draw a grid, an axis, or a tick mark */
624         if (scale == SCALE_GRID)
625             /* drawing a grid line */
626             plot_vline(p, Y_GRID_START, (Y_GRID_END - Y_GRID_START));
627         else if (i == 0)
628             /* drawing the y axis */
629             plot_vline(p, Y_AXIS_START, (Y_AXIS_END - Y_AXIS_START));
630         else
631             /* must be drawing a tick mark */
632             plot_vline(p, (X_AXIS_POS - (TICK_LEN / 2)), TICK_LEN);
633     }
634
635     /* done with the axes */
636     break;
637
638     case SCALE_NONE:    /* there is no scale */
639         /* already restored plot so nothing to do */
640         break;
641
642 }
643
644 /* now remember the new (now current) scale type */
645 cur_scale = scale;
646
647
648 /* scale is taken care of, return */
649 return;
650
651 }
652
653
654
655
656
657 /*
658 clear_saved_areas
659
660 Description:      This function clears all the saved areas (for saving the
661                   trace under the axes, menus, and general areas).
662
663 Arguments:       None.
664 Return Value:    None.
665
666 Input:          None.
667 Output:         None.
668
669 Error Handling:  None.
670
671 Algorithms:      None.
672 Data Structures: None.
673
674 Global Variables: saved_axis_x - cleared.
675                   saved_axis_y - cleared.

```

```

676         saved_menu    - cleared.
677         saved_area    - cleared.
678
679     Author:            Glen George
680     Last Modified:     May 9, 2006
681
682 */
683
684 void clear_saved_areas()
685 {
686     /* variables */
687     int i;             /* loop indices */
688     int j;
689
690
691
692     /* clear x-axis and y-axis save areas */
693     for (j = 0; j <= (2 * Y_TICK_CNT); j++)
694         for (i = 0; i < (SIZE_X / 8); i++)
695             saved_axis_x[j][i] = 0;
696     for (j = 0; j <= (2 * X_TICK_CNT); j++)
697         for (i = 0; i < (SIZE_Y / 8); i++)
698             saved_axis_y[j][i] = 0;
699
700     /* clear the menu save ares */
701     for (i = 0; i < MENU_SIZE_Y; i++)
702         for (j = 0; j < ((MENU_SIZE_X + 7) / 8); j++)
703             saved_menu[i][j] = 0;
704
705     /* clear general save area */
706     for (i = 0; i < SAVE_SIZE_Y; i++)
707         for (j = 0; j < (SAVE_SIZE_X / 8); j++)
708             saved_area[i][j] = 0;
709
710
711     /* done clearing the saved areas - return */
712     return;
713 }
714
715
716
717
718
719 /*
720 restore_menu_trace
721
722 Description:            This function restores the trace under the menu when the
723                        menus are turned off. (The trace was previously saved.)
724
725 Arguments:             None.
726 Return Value:          None.
727
728 Input:                 None.
729 Output:                The trace under the menu is restored to the LCD screen.
730
731 Error Handling:        None.
732
733 Algorithms:            None.
734 Data Structures:       None.
735
736 Global Variables:      saved_menu - used to restore trace data under the menu.
737
738 Author:                Glen George
739 Last Modified:         June 03, 2014
740
741 */
742
743 void restore_menu_trace()
744 {
745     /* variables */
746     int bit_position;   /* position of bit to restore (in saved data) */
747     int bit_offset;     /* offset (in bytes) of bit within saved row */
748
749     int x;              /* loop indices */
750     int y;

```

```

751
752
753
754 /* loop, restoring the trace under the menu */
755 for (y = MENU_UL_Y; y < (MENU_UL_Y + MENU_SIZE_Y); y++) {
756
757     /* starting a row - initialize bit position */
758     bit_position = 0x80; /* start at high-order bit in the byte */
759     bit_offset = 0; /* first byte of the row */
760
761     for (x = MENU_UL_X; x < (MENU_UL_X + MENU_SIZE_X); x++) {
762
763         /* check if this point is on or off (need to look at bits) */
764         if ((saved_menu[y - MENU_UL_Y][bit_offset] & bit_position) == 0)
765             /* saved pixel is off */
766             plot_pixel(x, y, PIXEL_CLEAR);
767         else
768             /* saved pixel is on */
769             plot_pixel(x, y, PIXEL_TRACE);
770
771         /* move to the next bit position */
772         bit_position >>= 1;
773         /* check if moving to next byte */
774         if (bit_position == 0) {
775             /* now on high bit of next byte */
776             bit_position = 0x80;
777             bit_offset++;
778         }
779     }
780 }
781
782
783 /* restored menu area - return */
784 return;
785
786 }
787
788
789
790
791 /*
792 set_save_area
793
794 Description:      This function sets the position and size of the area to
795                  be saved when traces are drawn. It also clears any data
796                  currently saved.
797
798 Arguments:      pos_x (int) - x position of upper left corner of the
799                  saved area.
800                  pos_y (int) - y position of upper left corner of the
801                  saved area.
802                  size_x (int) - horizontal size of the saved area.
803                  size_y (int) - vertical size of the saved area.
804 Return Value:   None.
805
806 Input:         None.
807 Output:        None.
808
809 Error Handling: None.
810
811 Algorithms:    None.
812 Data Structures: None.
813
814 Global Variables: saved_area - cleared.
815                  saved_pos_x - set to passed value.
816                  saved_pos_y - set to passed value.
817                  saved_end_x - computed from passed values.
818                  saved_end_y - computed from passed values.
819
820 Author:        Glen George
821 Last Modified: Mar. 8, 1994
822
823 */
824
825 void set_save_area(int pos_x, int pos_y, int size_x, int size_y)

```

```

826 {
827     /* variables */
828     int x;      /* loop indices */
829     int y;
830
831
832
833     /* just setup all the locally global variables from the passed values */
834     saved_pos_x = pos_x;
835     saved_pos_y = pos_y;
836     saved_end_x = pos_x + size_x;
837     saved_end_y = pos_y + size_y;
838
839
840     /* clear the save area */
841     for (y = 0; y < SAVE_SIZE_Y; y++) {
842         for (x = 0; x < (SAVE_SIZE_X / 8); x++) {
843             saved_area[y][x] = 0;
844         }
845     }
846
847
848     /* setup the saved area - return */
849     return;
850
851 }
852
853
854
855
856 /*
857     restore_trace
858
859     Description:      This function restores the trace under the set saved
860                      area. (The area was previously set and the trace was
861                      previously saved.)
862
863     Arguments:       None.
864     Return Value:    None.
865
866     Input:           None.
867     Output:          The trace under the saved ares is restored to the LCD.
868
869     Error Handling:  None.
870
871     Algorithms:      None.
872     Data Structures: None.
873
874     Global Variables: saved_area - used to restore trace data.
875                      saved_pos_x - gives starting x position of saved area.
876                      saved_pos_y - gives starting y position of saved area.
877                      saved_end_x - gives ending x position of saved area.
878                      saved_end_y - gives ending y position of saved area.
879
880     Author:          Glen George
881     Last Modified:   June 03, 2014
882
883 */
884
885 void restore_trace()
886 {
887     /* variables */
888     int bit_position; /* position of bit to restore (in saved data) */
889     int bit_offset;   /* offset (in bytes) of bit within saved row */
890
891     int x;      /* loop indices */
892     int y;
893
894
895
896     /* loop, restoring the saved trace */
897     for (y = saved_pos_y; y < saved_end_y; y++) {
898
899         /* starting a row - initialize bit position */
900         bit_position = 0x80; /* start at high-order bit in the byte */

```

```

901 bit_offset = 0;      /* first byte of the row */
902
903     for (x = saved_pos_x; x < saved_end_x; x++) {
904
905         /* check if this point is on or off (need to look at bits) */
906         if ((saved_area[y - saved_pos_y][bit_offset] & bit_position) == 0)
907             /* saved pixel is off */
908             plot_pixel(x, y, PIXEL_CLEAR);
909         else
910             /* saved pixel is on */
911             plot_pixel(x, y, PIXEL_TRACE);
912
913         /* move to the next bit position */
914         bit_position >>= 1;
915         /* check if moving to next byte */
916         if (bit_position == 0) {
917             /* now on high bit of next byte */
918             bit_position = 0x80;
919             bit_offset++;
920         }
921     }
922 }
923
924 /* restored the saved area - return */
925 return;
926
927 }
928
929
930
931
932
933 /*
934 do_trace
935
936 Description:      This function starts a trace. It starts the hardware
937                   sampling data (via a function call) and sets the trace
938                   ready flag (trace_status) to FALSE and the sampling flag
939                   (sampling) to TRUE.
940
941 Arguments:        None.
942 Return Value:     None.
943
944 Input:            None.
945 Output:           None.
946
947 Error Handling:   None.
948
949 Algorithms:       None.
950 Data Structures:  None.
951
952 Global Variables: trace_status - set to FALSE (not ready for another trace).
953                   sampling      - set to TRUE (doing a sample now).
954
955 Author:           Glen George
956 Last Modified:    Mar. 13, 1994
957
958 */
959
960 void do_trace()
961 {
962     /* variables */
963     /* none */
964
965
966
967     /* start up the trace */
968     /* indicate whether using automatic triggering or not */
969     start_sample(get_trigger_mode() == AUTO_TRIGGER);
970
971     /* now not ready for another trace (currently doing one) */
972     trace_status = FALSE;
973
974     /* and are currently sampling data */
975     sampling = TRUE;

```



```

976
977
978     /* trace is going, return */
979     return;
980
981 }
982
983
984 /*
985 plot_trace
986
987 Description:      This function plots the passed trace.  The trace is
988                   assumed to contain sample_size points of sampled data.
989                   Any points falling within any of the save areas are also
990                   saved by this routine.  The data is also scaled to be
991                   within the range of the entire screen.
992
993
994 Arguments:        sample (unsigned char far *) - sample to plot.
995 Return Value:      None.
996
997 Input:            None.
998 Output:           The sample is plotted on the screen.
999
1000 Error Handling:    None.
1001
1002 Algorithms:        If there are more sample points than screen width the
1003                   sample is plotted with multiple points per horizontal
1004                   position.
1005 Data Structures:   None.
1006
1007 Global Variables:  cur_scale      - determines type of scale to plot.
1008                   sample_size    - determines size of passed sample.
1009                   saved_axis_x    - stores trace under x-axis.
1010                   saved_axis_y    - stores trace under y-axis.
1011                   saved_menu      - stores trace under the menu.
1012                   saved_area      - stores trace under the saved area.
1013                   saved_pos_x     - determines location of saved area.
1014                   saved_pos_y     - determines location of saved area.
1015                   saved_end_x     - determines location of saved area.
1016                   saved_end_y     - determines location of saved area.
1017
1018 Author:           Glen George
1019 Last Modified:    June 03, 2014
1020
1021 */
1022
1023 void plot_trace(unsigned char *sample)
1024 {
1025     /* variables */
1026     int x = 0;          /* current x position to plot */
1027     int x_pos = (PLOT_SIZE_X / 2); /* "fine" x position for multiple point plotting */
1028
1029     int y;              /* y position of point to plot */
1030
1031     int p;              /* an x or y coordinate */
1032
1033     int i;              /* loop indices */
1034     int j;
1035
1036
1037     /* clear the saved areas too */
1038     clear_saved_areas();
1039
1040     /* re-display the menu (if it was on) */
1041     refresh_menu();
1042
1043
1044     /* plot the sample */
1045     for (i = 0; i < sample_size; i++) {
1046
1047         /* determine y position of point (note: screen coordinates invert) */
1048         y = (PLOT_SIZE_Y - 1) - ((sample[i] * (PLOT_SIZE_Y - 1)) / 255);
1049
1050         /* clear previous point on trace */

```

```

1051 plot_pixel(i, old_sample[i], PIXEL_CLEAR);
1052
1053 /* plot this point */
1054 plot_pixel(x, y, PIXEL_TRACE);
1055
1056 /* and save new value */
1057 old_sample[i] = y;
1058
1059
1060 /* check if the point is in a save area */
1061
1062 /* check if in the menu area */
1063 if ((x >= MENU_UL_X) && (x < (MENU_UL_X + MENU_SIZE_X)) &&
1064     (y >= MENU_UL_Y) && (y < (MENU_UL_Y + MENU_SIZE_Y)))
1065     /* point is in the menu area - save it */
1066     saved_menu[y - MENU_UL_Y][(x - MENU_UL_X)/8] |= (0x80 >> ((x - MENU_UL_X) % 8));
1067
1068 /* check if in the saved area */
1069 if ((x >= saved_pos_x) && (x <= saved_end_x) && (y >= saved_pos_y) && (y <= saved_end_y))
1070     /* point is in the save area - save it */
1071     saved_area[y - saved_pos_y][(x - saved_pos_x)/8] |= (0x80 >> ((x - saved_pos_x) % 8));
1072
1073 /* check if on a grid line */
1074 /* go through all the horizontal lines */
1075 for (j = -Y_TICK_CNT; j <= Y_TICK_CNT; j++) {
1076
1077     /* get y position of the line */
1078     p = X_AXIS_POS + j * Y_TICK_SIZE;
1079     /* make sure it is in range */
1080     if (p >= PLOT_SIZE_Y)
1081         p = PLOT_SIZE_Y - 1;
1082     if (p < 0)
1083         p = 0;
1084
1085     /* if the point is on this line, save it */
1086     if (y == p)
1087         saved_axis_x[j + Y_TICK_CNT][x / 8] |= (0x80 >> (x % 8));
1088 }
1089
1090 /* go through all the vertical lines */
1091 for (j = -X_TICK_CNT; j <= X_TICK_CNT; j++) {
1092
1093     /* get x position of the line */
1094     p = Y_AXIS_POS + j * X_TICK_SIZE;
1095     /* make sure it is in range */
1096     if (p >= PLOT_SIZE_X)
1097         p = PLOT_SIZE_X - 1;
1098     if (p < 0)
1099         p = 0;
1100
1101     /* if the point is on this line, save it */
1102     if (x == p)
1103         saved_axis_y[j + X_TICK_CNT][y / 8] |= (0x80 >> (y % 8));
1104 }
1105
1106
1107 /* update x position */
1108 x_pos += PLOT_SIZE_X;
1109 /* check if at next horizontal position */
1110 if (x_pos >= sample_size) {
1111     /* at next position - update positions */
1112     x++;
1113     x_pos -= sample_size;
1114 }
1115 }
1116
1117
1118 /* finally, output the scale if need be */
1119 set_display_scale(cur_scale);
1120
1121
1122 /* done with plot, return */
1123 return;
1124
1125 }

```

```

1  /*****
2  /*
3  /*          TRACUTIL.H
4  /*          Trace Utility 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 trace
13  utility functions (defined in tracutil.c) for the Digital Oscilloscope
14  project.
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 name of set_axes function to
21                               set_display_scale.
22      5/9/06   Glen George      Added the constants for grids and tick marks.
23      5/27/08  Glen George      Added is_sampling() function to be able to
24                               tell if the system is currently taking a
25                               sample.
26      6/3/08   Glen George      Removed Y_SCALE_FACTOR - no longer used to
27                               fix problems with non-power of 2 display
28                               sizes.
29  */
30
31
32
33 #ifndef __TRACUTIL_H__
34 #define __TRACUTIL_H__
35
36
37 /* library include files */
38 /* none */
39
40 /* local include files */
41 #include "interfac.h"
42 #include "menuact.h"
43
44
45
46
47 /* constants */
48
49 /* plot size */
50 #define PLOT_SIZE_X    SIZE_X      /* plot takes entire screen width */
51 #define PLOT_SIZE_Y    SIZE_Y      /* plot takes entire screen height */
52
53 /* axes position and size */
54 #define X_AXIS_START    0           /* starting x position of x-axis */
55 #define X_AXIS_END      (PLOT_SIZE_X - 1) /* ending x position of x-axis */
56 #define X_AXIS_POS      (PLOT_SIZE_X / 2) /* y position of x-axis */
57 #define Y_AXIS_START    0           /* starting y position of y-axis */
58 #define Y_AXIS_END      (PLOT_SIZE_Y - 1) /* ending y position of y-axis */
59 #define Y_AXIS_POS      (PLOT_SIZE_Y / 2) /* x position of y-axis */
60
61 /* tick mark and grid constants */
62 #define TICK_LEN        5           /* length of axis tick mark */
63 /* tick mark counts are for a single quadrant, thus total number of tick */
64 /* marks or grids is twice this number */
65 #define X_TICK_CNT      5           /* always 5 tick marks on x axis */
66 #define X_TICK_SIZE     (PLOT_SIZE_X / (2 * X_TICK_CNT)) /* distance between tick marks */
67 #define Y_TICK_SIZE     X_TICK_SIZE /* same size as x */
68 #define Y_TICK_CNT      (PLOT_SIZE_Y / (2 * Y_TICK_SIZE)) /* number of y tick marks */
69 #define X_GRID_START    0           /* starting x position of x grid */
70 #define X_GRID_END      (PLOT_SIZE_X - 1) /* ending x position of x grid */
71 #define Y_GRID_START    0           /* starting y position of y-axis */
72 #define Y_GRID_END      (PLOT_SIZE_Y - 1) /* ending y position of y-axis */
73
74 /* maximum size of the save area (in pixels) */
75 #define SAVE_SIZE_X      120 /* maximum width */

```

```

76 #define SAVE_SIZE_Y    16  /* maximum height */
77
78 /* sleep time between samples, designed to reduce blinking */
79 #define DRAW_INTERVAL  50000
80
81
82
83
84 /* structures, unions, and typedefs */
85     /* none */
86
87
88
89
90 /* function declarations */
91
92 /* initialize the trace utility routines */
93 void  init_trace(void);
94
95 /* trace status functions */
96 void  set_mode(enum trigger_type); /* set the triggering mode */
97 int   is_sampling(void);          /* currently trying to take a sample */
98 int   trace_rdy(void);            /* determine if ready to start a trace */
99 void  trace_done(void);           /* signal a trace has been completed */
100 void  trace_rearm(void);          /* re-enable tracing */
101
102 /* trace save area functions */
103 void  clear_saved_areas(void);     /* clears all saved areas */
104 void  restore_menu_trace(void);    /* restore the trace under menus */
105 void  set_save_area(int, int, int, int); /* set an area of a trace to save */
106 void  restore_trace(void);        /* restore saved area of a trace */
107
108 /* set the scale type */
109 void  set_display_scale(enum scale_type);
110
111 /* setup and plot a trace */
112 void  set_trace_size(int);         /* set the number of samples in a trace */
113 void  do_trace(void);              /* start a trace */
114 void  plot_trace(unsigned char *); /* plot a trace (sampled data) */
115
116
117 #endif
118

```

```

1  /*****
2  /*
3  /*          TRIGGER.S
4  /*          Data sampling and triggering
5  /*          Digital Oscilloscope Project
6  /*          EE/CS 52
7  /*          Santiago Navonne
8  /*
9  /*****/
10
11 /*
12 Data sampling and triggering control routines for the EE/CS 52 Digital
13 Oscilloscope project. Function definitions are included in this file, and
14 are laid out as follows:
15 - set_sample_rate: Configures the sampling rate;
16 - set_trigger: Configures the manual trigger level and slope;
17 - set_delay: Configures the manual trigger delay;
18 - start_sample: Starts a new data sample with the previously configured
19 settings and passed auto-trigger configuration;
20 - sample_done: Checks whether a new data sample set is available, returning
21 a pointer to a buffer containing it if there is, or a NULL
22 pointer if there isn't;
23 - sample_handler: Handles sampling FIFO full interrupts;
24 - trigger_init: Initializes the environment's shared variables and the
25 triggering logic circuit (resetting it), effectively
26 preparing the sampling/triggering interface for use.
27
28
29 Revision History:
30 5/29/14 Santiago Navonne Initial revision.
31 6/01/14 Santiago Navonne Minor fixes; updated documentation.
32 6/11/14 Santiago Navonne Changed division algorithm in set_sample_rate.
33 */
34
35 /* Includes */
36 #include "general.h" /* General assembly constants */
37 #include "system.h" /* Base addresses */
38 #include "interfac.h" /* Software interface definitions */
39 #include "trigger.h" /* Local constants */
40
41
42 /* Variables */
43 .section .data /* No alignment necessary: variables are bytes */
44 sample_pending: .byte 0 /* Logical value: whether a sample is pending */
45 sample: .skip FIFO_SIZE /* Sample buffer */
46
47 .section .text /* Code starts here */
48
49 /*
50 * set_sample_rate
51 *
52 * Description: This procedure configures the sampling rate of the sampling
53 * interface. After execution, the interface will start sampling
54 * at the requested rate, rounded up to a multiple of the system
55 * clock. The return value is how many samples will be acquired,
56 * which is always the size of the FIFO.
57 * If an argument of 0 is passed, the function has no effect, and
58 * returns 0. The argument must however be less than or equal to
59 * the system clock divided by two; no error checking is performed
60 * on this.
61 *
62 * Operation: The procedure starts by error checking the value of the argument,
63 * simply returning 0 if it is invalid. Then, it computes the
64 * required clock period in system clock periods by dividing the
65 * system clock frequency by the requested sample rate.
66 * Finally, it saves the computed value to the trigger period
67 * register, and pulses the reset bit in the control register to
68 * reset the triggering logic. SIZE_X is ultimately moved into
69 * r2 as constant return value.
70 *
71 * Arguments: samples_per_sec - positive integer indicating the sample rate
72 * in samples per second (r4). The value must
73 * be less than or equal to the system clock
74 * divided by two.
75 *

```

```

76 * Return Value:      sample_num - positive integer, number of samples that will be
77 *                    acquired at the desired rate (r2).
78 *
79 * Local Variables:   None.
80 *
81 * Shared Variables:  None.
82 *
83 * Global Variables:  None.
84 *
85 * Input:             None.
86 *
87 * Output:            None.
88 *
89 * Error Handling:     If the argument is zero, the function has no effect, and returns 0.
90 *                    No error checking is performed on the upper bound of the sampling
91 *                    rate.
92 *
93 * Limitations:       Resulting sample clock is an integer multiple of the system clock;
94 *                    corresponding rate will be greater than or equal to the requested
95 *                    rate, with a difference in period less than the system clock's.
96 *                    Number of samples acquired must be <= FIFO_SIZE per hardware
97 *                    limitations (size of FIFO).
98 *
99 * Algorithms:        Division is performed using a repeated subtraction algorithm since
100 *                    hardware division cannot be assumed to be available. This algorithm
101 *                    is acceptable because generally very few iterations will be needed
102 *                    to reach the result.
103 * Data Structures:    None.
104 *
105 * Registers Changed: r2, r4, r8, r9.
106 *
107 * Revision History:
108 *     5/29/14   Santiago Navonne   Initial revision.
109 *     6/01/14   Santiago Navonne   Added error checking, expanded documentation.
110 *     6/11/14   Santiago Navonne   Changed hardware divide instruction to division
111 *                                   by repeated subtraction.
112 *
113 */
114 .global set_sample_rate
115 set_sample_rate:
116     MOV     r2, r0                /* load return value of 0 in case of error */
117     BEQ     r4, r0, set_sample_rate_done /* error if argument is 0 */
118
119     MOVHI   r8, %hi(CLK_FREQ)    /* load system clock frequency to */
120     ORI     r8, r8, %lo(CLK_FREQ) /* find number of system clocks that takes */
121     /*DIVU   r9, r8, r4          /* by dividing the sys clk by the requested rate */
122     XOR     r9, r9, r9           /* prepare register for division: r9 is quotient */
123
124 div_check:
125     BLT     r8, r4, div_done      /* check if the divisor fits in the dividend */
126     /* we're done when it doesn't any more */
127
128 div_loop:
129     SUB     r8, r8, r4            /* need to keep subtracting: */
130     /* subtract divisor from dividend */
131     ADDI    r9, r9, 1             /* and increment quotient */
132     JMPL    div_check            /* thus repeat as needed */
133
134 div_done:
135     MOVHI   r8, %hi(TRIG_PERIOD_BASE) /* load period data register address to */
136     ORI     r8, r8, %lo(TRIG_PERIOD_BASE) /* finally save result to trigger period */
137     STWIO   r9, (r8)              /* data, effectively setting the sample rate */
138
139     MOVHI   r8, %hi(TRIG_CTRL_SET) /* load trigger control bit set reg address */
140     ORI     r8, r8, %lo(TRIG_CTRL_SET) /* to reset trigger logic */
141     MOVI    r9, FIFO_RESET_BIT     /* by sending reset bit high */
142     STWIO   r9, (r8)
143     ADDI    r8, r8, WORD_SIZE       /* and then move to bit clr reg */
144     STWIO   r9, (r8)              /* to send it low */
145
146     MOVI    r2, SIZE_X             /* number of samples acquired is always size of display */
147
148 set_sample_rate_done:
149     RET                            /* all done */
150     /* return value is in r2 */

```

```

151  /*
152  *   set_trigger
153  *
154  *   Description:       This function configures the triggering settings on the sampling
155  *                       interface. After execution, triggering will occur as soon as the
156  *                       input passes the value of <level>, in the direction indicated by
157  *                       <slope>. Note that these settings are only used when a sample is
158  *                       started with manual triggering enabled.
159  *
160  *   Operation:         The procedure first "corrects" the level, mapping it to the
161  *                       right range ([0, 255]) and adding any necessary calibration
162  *                       constants.
163  *                       Then, it writes the slope bit to either the trigger control set
164  *                       or clear register, depending on what action needs to be performed,
165  *                       followed by the corrected level argument to the trigger level
166  *                       register.
167  *                       Finally, the reset bit within the trigger control register is
168  *                       pulsed to reset the triggering logic.
169  *
170  *   Arguments:         level - trigger level to be configured, as a value between 0 and
171  *                       127, where 0 is the most negative level, and 127 is the
172  *                       most positive level (r4).
173  *                       slope - desired trigger slope; 1 for positive slope, 0 for
174  *                       negative slope (r5).
175  *
176  *   Return Value:      None.
177  *
178  *   Local Variables:   None.
179  *
180  *   Shared Variables:  None.
181  *
182  *   Global Variables:  None.
183  *
184  *   Input:             None.
185  *
186  *   Output:            None.
187  *
188  *   Error Handling:    None.
189  *
190  *   Limitations:       None.
191  *
192  *   Algorithms:        None.
193  *   Data Structures:   None.
194  *
195  *   Registers Changed: r4, r8, r9, r10.
196  *
197  *   Revision History:
198  *       5/29/14   Santiago Navonne   Initial revision.
199  *       6/01/14   Santiago Navonne   Expanded documentation.
200  *
201  */
202  .global set_trigger
203  set_trigger:
204      MOVHI    r10, %hi(TRIG_LEVEL_BASE) /* load trigger level register address to update */
205      ORI      r10, r10, %lo(TRIG_LEVEL_BASE) /* the desired trigger level */
206      MOVI     r9, TRIG_LEVEL_SHIFT /* shift the passed argument left as needed to */
207      SLL      r4, r4, r9 /* make sure we output a full byte */
208      SUBI     r4, r4, CALIBRATION /* and correct value with calibration data */
209
210      MOVHI    r8, %hi(TRIG_CTRL_CLR) /* load control register bit clear address to */
211      ORI      r8, r8, %lo(TRIG_CTRL_CLR) /* initially assume that we want to set */
212      MOVI     r9, 2 /* slope to negative (clear the bit) */
213      SLL      r5, r5, r9 /* subtract argument multiplied by word size */
214      SUB      r8, r8, r5 /* effectively moving to set bit register if enabling */
215                        /* positive slope */
216
217      MOVI     r9, SLOPE_BIT /* finally write the appropriate bit to the register */
218      STWIO    r9, (r8) /* enabling or disabling the bit as needed */
219
220      STWIO    r4, (r10) /* and output desired trigger level */
221
222      MOVHI    r8, %hi(TRIG_CTRL_SET) /* load trigger control bit set reg address */
223      ORI      r8, r8, %lo(TRIG_CTRL_SET) /* to reset trigger logic */
224      MOVI     r9, FIFO_RESET_BIT /* by sending reset bit high */
225      STWIO    r9, (r8)

```

```

226     ADDI    r8, r8, WORD_SIZE      /* and then move to bit clr reg */
227     STWIO   r9, (r8)              /* to send it low */
228
229     RET                                /* all done, so return */
230
231
232 /*
233 * set_delay
234 *
235 * Description:      This procedure configures the sampling delay on manual triggers.
236 *                  After execution, triggering will occur <delay> samples after the
237 *                  configured level and slope settings are satisfied. Note that this
238 *                  setting is only used when manual triggering is enabled.
239 *                  Also note that delay must be less than MAX_DELAY.
240 *
241 * Operation:        The function first corrects the argument by adding the necessary
242 *                  hardware constant to it, and then outputs it to the trigger
243 *                  delay register.
244 *                  Finally, the reset bit within the trigger control register is
245 *                  pulsed to reset the triggering logic.
246 *
247 * Arguments:        delay - unsigned integer <= MAX_DELAY; trigger delay from
248 *                  trigger event in number of samples (r4).
249 *
250 * Return Value:      None.
251 *
252 * Local Variables:   None.
253 *
254 * Shared Variables:  None.
255 *
256 * Global Variables:  None.
257 *
258 * Input:             None.
259 *
260 * Output:            None.
261 *
262 * Error Handling:     None.
263 *
264 * Limitations:        Only positive delays less than or equal to MAX_DELAY are valid.
265 *
266 * Algorithms:         None.
267 * Data Structures:    None.
268 *
269 * Registers Changed: r4, r10.
270 *
271 * Revision History:
272 *     5/29/14    Santiago Navonne    Initial revision.
273 *     6/01/14    Santiago Navonne    Expanded documentation.
274 *
275 */
276 .global set_delay
277 set_delay:
278     MOVHI    r10, %hi(TRIG_DELAY_BASE) /* load trigger delay register address to update */
279     ORI      r10, r10, %lo(TRIG_DELAY_BASE) /* the desired delay time */
280     ADDI     r4, r4, DELAY_CONSTANT      /* add delay constant to correct argument */
281     STWIO    r4, (r10)                  /* and output to delay register, effectively */
282                                         /* configuring delay */
283
284     MOVHI    r8, %hi(TRIG_CTRL_SET)     /* load trigger control bit set reg address */
285     ORI      r8, r8, %lo(TRIG_CTRL_SET) /* to reset trigger logic */
286     MOVI     r9, FIFO_RESET_BIT         /* by sending reset bit high */
287     STWIO    r9, (r8)
288     ADDI     r8, r8, WORD_SIZE           /* and then move to bit clr reg */
289     STWIO    r9, (r8)                  /* to send it low */
290
291     RET                                /* all done, so return */
292
293
294 /*
295 * start_sample
296 *
297 * Description:      This procedure immediately starts sampling data. If the argument
298 *                  is FALSE, sampling starts upon a trigger event. If the argument
299 *                  is TRUE, sampling starts immediately.
300 *                  Any previously started but incomplete samples are cancelled and

```



```

301 * replaced.
302 *
303 * Operation: The procedure sets or clears the auto trigger bit in the trigger
304 * control register to enable or disable auto triggering.
305 * Finally, it starts the sample by enabling writing to the FIFO
306 * through the write enable bit in the control register, and resets
307 * the triggering logic.
308 *
309 * Arguments: auto_trigger - TRUE if sampling should be started
310 * automatically (i.e. as soon as possible),
311 * FALSE if it should be started on a trigger
312 * event (r4).
313 *
314 * Return Value: None.
315 *
316 * Local Variables: None.
317 *
318 * Shared Variables: None.
319 *
320 * Global Variables: None.
321 *
322 * Input: None.
323 *
324 * Output: None.
325 *
326 * Error Handling: None.
327 *
328 * Limitations: None.
329 *
330 * Algorithms: None.
331 * Data Structures: None.
332 *
333 * Registers Changed: r8, r9.
334 *
335 * Revision History:
336 * 5/29/14 Santiago Navonne Initial revision.
337 * 6/01/14 Santiago Navonne Expanded documentation.
338 *
339 */
340 .global start_sample
341 start_sample:
342
343     MOVHI    r8, %hi(TRIG_CTRL_CLR) /* load trigger control bit clear reg address */
344     ORI      r8, r8, %lo(TRIG_CTRL_CLR) /* assuming we'll clear auto trigger bit */
345     MOVI     r9, 2 /* subtract argument multiplied by word size */
346     SLL      r4, r4, r9 /* effectively moving to set bit register if enabling */
347     SUB      r8, r8, r4 /* auto trigger*/
348
349     MOVI     r9, AUTO_TRIG_BIT /* store auto trigger bit in configured register */
350     STWIO    r9, (r8) /* enabling or disabling it as needed */
351
352     MOVHI    r8, %hi(TRIG_CTRL_SET) /* load trigger control bit set reg address */
353     ORI      r8, r8, %lo(TRIG_CTRL_SET) /* to reset trigger logic */
354     MOVI     r9, FIFO_RESET_BIT /* by sending reset bit high */
355     STWIO    r9, (r8)
356     ADDI     r8, r8, WORD_SIZE /* and then move to bit clr reg */
357     STWIO    r9, (r8) /* to send it low */
358
359     MOVHI    r8, %hi(TRIG_CTRL_CLR) /* load trigger control bit clear reg address */
360     ORI      r8, r8, %lo(TRIG_CTRL_CLR) /* to clear fifo write enable (make active) */
361     MOVI     r9, FIFO_WE_BIT /* which allows the fifo to be filled with samples */
362     STWIO    r9, (r8) /* effectively starting a sample */
363
364 start_sample_done:
365     RET /* all done, so return */
366
367
368 /*
369 * sample_done
370 *
371 * Description: This function checks whether the started sample was completed.
372 * If the sample was completed, a pointer to the buffer containing the
373 * sampled data is provided. If the sample was not completed, a NULL
374 * pointer is returned.
375 * Note that this function returns a non-NULL pointer once per call to

```

```

376 *          start_sample.
377 *
378 * Operation:      The function first checks the value of sample_pending to
379 *                  ensure that a sample is ready. If no sample is ready, it simply
380 *                  returns with NULL in r2.
381 *                  Then, it resets the values of the shared variable to indicate that
382 *                  a sample was completed.
383 *                  Finally, the function clocks the FIFO twice to account for its
384 *                  latency, and then reads FIFO_SIZE bytes in a loop, storing them in
385 *                  array <samples>. Note that at each iteration, reading is performed
386 *                  by bit-banging the FIFO's read clock. Also note that a calibration
387 *                  constant is added to each sample to account for the front end's DC
388 *                  offset.
389 *
390 * Arguments:      None.
391 *
392 * Return Value:   *samples - pointer to bytes acquired in sample if any; NULL
393 *                  otherwise (r2).
394 *
395 * Local Variables: r13 - pointer to current place in samples array.
396 *                  r10 - number of sample currently being copied.
397 *
398 * Shared Variables: - sample_pending: logical value; zero if no sample is pending,
399 *                    non-zero otherwise. Read/Write.
400 *
401 * Global Variables: None.
402 *
403 * Input:          Data samples from the FIFO.
404 *
405 * Output:         None.
406 *
407 * Error Handling:  None.
408 *
409 * Limitations:    None.
410 *
411 * Algorithms:     None.
412 * Data Structures: samples - array of size FIFO_SIZE where samples are stored and
413 *                    whose pointer is returned.
414 *
415 * Registers Changed: r2, r8, r9, r10, r11, r12, r13, r14.
416 *
417 * Revision History:
418 *      5/29/14    Santiago Navonne    Initial revision.
419 *      6/01/14    Santiago Navonne    Expanded documentation.
420 *
421 */
422
423 .global sample_done
424 sample_done:
425     MOV     r2, r0                /* assume no sample ready: null pointer return val */
426     MOVIA   r8, sample_pending    /* fetch current pending value to see if this call */
427     LDB     r9, (r8)              /* should be ignored */
428     BEQ     r0, r9, sample_done_done /* which is when value is zero */
429
430     MOVIA   r8, sample_pending    /* reset sample_pending to indicate */
431     STB     r0, (r8)              /* no sample is ready for processing */
432
433     MOVHI   r12, %hi(FIFO_DATA_BASE) /* load fifo data register address */
434     ORI     r12, r12, %lo(FIFO_DATA_BASE) /* to actually read data from fifo */
435     MOVHI   r8, %hi(TRIG_CTRL_SET) /* load ctrl reg set bit addr for */
436     ORI     r8, r8, %lo(TRIG_CTRL_SET) /* for bit banging */
437     MOVIA   r13, sample           /* load array address to store samples */
438     MOV     r2, r13              /* and also use it as return value (pointer) */
439     MOV     r10, r0              /* and start a counter at 0 for looping */
440     MOVI    r11, FIFO_SIZE       /* which will stop at FIFO_SIZE */
441     MOVI    r9, FIFO_READ_BIT    /* finally load read clk bit for big banging */
442
443     /* FIFO has 2 clocks latency */
444     STWIO   r9, (r8)             /* send read clock high to output sample */
445     ADDI    r8, r8, WORD_SIZE    /* and move to clear register: will send low next time */
446     NOP     /* wait for sample to actually come through */
447     STWIO   r9, (r8)             /* send read clock low to prepare for next sample */
448     ADDI    r8, r8, NEG_WORD_SIZE /* and move to set register: will send high next time */
449     NOP     /* wait for sample to actually come through */
450
451     STWIO   r9, (r8)             /* send read clock high to output sample */

```

```

451     ADDI    r8, r8, WORD_SIZE      /* and move to clear register: will send low next time */
452     NOP                                /* wait for sample to actually come through */
453     STWIO   r9, (r8)                /* send read clock low to prepare for next sample */
454     ADDI    r8, r8, NEG_WORD_SIZE  /* and move to set register: will send high next time */
455     NOP                                /* wait for sample to actually come through */
456
457 get_data:
458     STWIO   r9, (r8)                /* send read clock high to output sample */
459     ADDI    r8, r8, WORD_SIZE      /* and move to clear register: will send low next time */
460     NOP                                /* wait for sample to actually come through */
461
462     LDBIO   r14, (r12)              /* read sample from fifo */
463     ADDI    r14, r14, CALIBRATION  /* add calibration constant */
464     STBIO   r14, (r13)              /* and store it in the sample array */
465
466     STWIO   r9, (r8)                /* send read clock low to prepare for next sample */
467     ADDI    r8, r8, NEG_WORD_SIZE  /* and move to set register: will send high next time */
468
469     ADDI    r10, r10, 1              /* increment counter */
470     ADDI    r13, r13, 1              /* and sample pointer */
471     BNE     r10, r11, get_data      /* and keep getting data until we reach end */
472
473 sample_done_done:                  /* all done */
474     RET                                /* so return with pointer (or NULL) in r2 */
475
476
477 /*
478 *   sample_handler
479 *
480 *   Description:      This function handles FIFO full hardware interrupts, notifying
481 *                     the interface that a sample is ready to be read.
482 *
483 *   Operation:       The function changes the value of shared variable sample_pending
484 *                     to indicate that a sample is now ready.
485 *                     Then, it disables writing to the FIFO to make sure no data is
486 *                     written as the FIFO is emptied.
487 *                     Finally, it sends an EOI to reset the interrupt interface.
488 *
489 *   Arguments:       None.
490 *
491 *   Return Value:    None.
492 *
493 *   Local Variables: None.
494 *
495 *   Shared Variables: - sample_pending: logical value; zero if no sample is pending,
496 *                     non-zero otherwise. Write only.
497 *
498 *   Global Variables: None.
499 *
500 *   Input:           None.
501 *
502 *   Output:          None.
503 *
504 *   Error Handling:  None.
505 *
506 *   Limitations:     None.
507 *
508 *   Algorithms:      None.
509 *   Data Structures: None.
510 *
511 *   Registers Changed: r8, r9.
512 *
513 *   Revision History:
514 *       5/29/14   Santiago Navonne   Initial revision.
515 *       6/01/14   Santiago Navonne   Expanded documentation.
516 *
517 */
518 .global sample_handler
519 sample_handler:
520     MOVIA   r8, sample_pending      /* mark sample_pending as true to indicate */
521     MOVI    r9, TRUE                 /* a sample is ready for processing */
522     STB     r9, (r8)
523
524     MOVHI   r8, %hi(TRIG_CTRL_SET) /* load trigger control bit set reg address */
525     ORI     r8, r8, %lo(TRIG_CTRL_SET) /* to set fifo write enable (make inactive) */

```

```

526     MOVI    r9, FIFO_WE_BIT          /* which prevents the fifo from being filled again */
527     STWIO   r9, (r8)                 /* effectively stopping a sample */
528
529     MOVHI    r8, %hi(FIFO_FULL_BASE) /* write to edge capture register */
530     ORI      r8, r8, %lo(FIFO_FULL_BASE) /* to send EOI */
531     MOVI     r9, FIFO_INT
532     STWIO    r9, EDGE_CAP_OF(r8)
533
534     RET                                           /* all done, so return */
535
536
537 /*
538 * trigger_init
539 *
540 * Description:      This function performs all the necessary initialization of the
541 *                  sampling and triggering interface, preparing shared variables
542 *                  for use and configuring the triggering logic. It must be called
543 *                  before using any of the other provided functions.
544 *
545 * Operation:        The procedure first sets the shared variable sample_pending to
546 *                  0, indicating that no sample is pending and no sample has been
547 *                  started.
548 *                  Then, it resets the triggering logic using the reset bit in the
549 *                  control register, and configures the default triggering level,
550 *                  delay, rate, and other settings.
551 *                  Finally, it installs the interrupt handler by sending an EOI,
552 *                  using the HAL API alt_ic_isr_register, and enabling interrupts
553 *                  in the interrupt mask register.
554 *
555 * Arguments:         None.
556 *
557 * Return Value:      None.
558 *
559 * Local Variables:   None.
560 *
561 * Shared Variables:  - sample_pending: logical value; zero if no sample is pending,
562 *                  non-zero otherwise. Write only.
563 *
564 * Global Variables:  None.
565 *
566 * Input:             None.
567 *
568 * Output:            None.
569 *
570 * Error Handling:     None.
571 *
572 * Limitations:        None.
573 *
574 * Algorithms:         None.
575 * Data Structures:    None.
576 *
577 * Registers Changed: r4, r5, r6, r7, r8, r9.
578 *
579 * Revision History:
580 *      5/29/14    Santiago Navonne    Initial revision.
581 *      6/01/14    Santiago Navonne    Expanded documentation.
582 *
583 */
584 .global trigger_init
585 trigger_init:
586     MOVIA    r8, sample_pending        /* mark sample_pending as false to indicate */
587     STB       r0, (r8)                 /* no sample is ready for processing */
588
589     MOVHI    r8, %hi(TRIG_LEVEL_BASE)  /* load trigger level reg address */
590     ORI      r8, r8, %lo(TRIG_LEVEL_BASE) /* to set default value */
591     MOVI     r9, TRIG_LEVEL_DEF
592     STWIO    r9, (r8)
593
594     MOVHI    r8, %hi(TRIG_DELAY_BASE)  /* load trigger delay reg address */
595     ORI      r8, r8, %lo(TRIG_DELAY_BASE) /* to set default value */
596     MOVI     r9, TRIG_DELAY_DEF
597     STWIO    r9, (r8)
598
599     MOVHI    r8, %hi(TRIG_PERIOD_BASE) /* load trigger period reg address */
600     ORI      r8, r8, %lo(TRIG_PERIOD_BASE) /* to set default value for rate */

```

```

601     MOVI     r9, TRIG_PERIOD_DEF
602     STWIO    r9, (r8)
603
604     MOVHI    r8, %hi(TRIG_CTRL_SET) /* load trigger control bit set reg address */
605     ORI      r8, r8, %lo(TRIG_CTRL_SET) /* to reset trigger logic */
606     MOVI     r9, FIFO_RESET_BIT /* by sending reset bit high */
607     STWIO    r9, (r8)
608
609     MOVI     r9, TRIG_CTRL_DEF /* load default WE, read clock, auto */
610     STWIO    r9, (r8) /* trigger, and slope values */
611     ADDI     r8, r8, WORD_SIZE /* and move to clear register */
612     MOVI     r9, FIFO_RESET_BIT /* to send reset bit low */
613     STWIO    r9, (r8)
614
615     MOVHI    r8, %hi(FIFO_FULL_BASE) /* write to edge capture register to send */
616     ORI      r8, r8, %lo(FIFO_FULL_BASE) /* EOI to pending interrupts */
617     MOVI     r9, FIFO_INT /* and to edge capture register to send */
618     STWIO    r9, EDGE_CAP_OF(r8) /* EOI to pending interrupts */
619
620
621     ADDI     sp, sp, NEG_WORD_SIZE /* register interrupt handler */
622     STW      ra, 0(sp) /* push return address */
623     MOV      r4, r0 /* argument ic_id is ignored */
624     MOVI     r5, FIFO_FULL_IRQ /* second arg is IRQ num */
625     MOVIA    r6, sample_handler /* third arg is int handler */
626     MOV      r7, r0 /* fourth arg is data struct (null) */
627     ADDI     sp, sp, NEG_WORD_SIZE /* fifth arg goes on stack */
628     STW      r0, 0(sp) /* and is ignored (so 0) */
629     CALL     alt_ic_isr_register /* finally, call setup function */
630     ADDI     sp, sp, WORD_SIZE /* clean up stack after call */
631     LDW      ra, 0(sp) /* pop return address */
632     ADDI     sp, sp, WORD_SIZE
633
634     MOVHI    r8, %hi(FIFO_FULL_BASE) /* write to interrupt mask register */
635     ORI      r8, r8, %lo(FIFO_FULL_BASE) /* to enable interrupts */
636     MOVI     r9, FIFO_INT
637     STWIO    r9, INTMASK_OF(r8)
638
639
640     RET /* all done, so return */
641

```

```

1  /*****
2  /*
3  /*          TRIGGER.H
4  /*          Data Sampling and Triggering Definitions
5  /*          Include File
6  /*          Digital Oscilloscope Project
7  /*          EE/CS 52
8  /*          Santiago Navonne
9  /*
10 /*****
11
12 /*
13 This file contains the constants for the data sampling and triggering
14 routines. The file includes hardware constants used to interact with the
15 triggering logic; masks used to access hardware registers; PIO register
16 offsets; PIO register addresses; and default configuration values.
17
18
19 Revision History:
20 5/30/14 Santiago Navonne Initial revision.
21 */
22
23 /* Hardware constants */
24 #define CLK_FREQ      38000000 /* System clock frequency in Hz */
25 #define FIFO_SIZE     512 /* Size of sample FIFO in words */
26 #define TRIG_LEVEL_SHIFT 1 /* Shift trig level left once to convert [0, 127] -> [0, 255] */
27 #define CALIBRATION   13 /* DC offset of front end */
28 #define DELAY_CONSTANT 1 /* Hardware delay offset */
29 #define MAXDELAY      0xFFFFFFF - 1 - DELAY_CONSTANT
30 /* Maximum delay must take hardware delay offset into account */
31
32 /* Masks */
33 #define FIFO_INT      1 /* FIFO interrupt bit */
34 #define AUTO_TRIG_BIT 1<<0 /* Auto trigger bit is bit 0 in trigger control register */
35 #define SLOPE_BIT     1<<1 /* Slope control bit is bit 1 in trigger control register */
36 #define FIFO_WE_BIT   1<<2 /* FIFO write enable bit is bit 2 in trigger control register */
37 #define FIFO_READ_BIT 1<<3 /* FIFO read clock bit is bit 3 in trigger control register */
38 #define FIFO_RESET_BIT 1<<4 /* FIFO reset bit is bit 4 in trigger control register */
39
40 /* PIO register offsets */
41 #define EDGE_CAP_OF   3*WORD_SIZE /* Offset of edge capture PIO register */
42 #define INTMASK_OF    2*WORD_SIZE /* Offset of interrupt mask PIO register */
43 #define SET_OF        4*WORD_SIZE /* Offset of bit set PIO register */
44 #define CLR_OF        5*WORD_SIZE /* Offset of bit clear PIO register */
45
46 /* PIO offset locations */
47 #define TRIG_CTRL_SET TRIG_CTRL_BASE+SET_OF /* Location of trigger control set bit register */
48 #define TRIG_CTRL_CLR TRIG_CTRL_BASE+CLR_OF /* Location of trigger control clear bit register */
49
50 /* Default values */
51 #define TRIG_CTRL_DEF 0b00000111 /* Initialize control register to: low read clock, inactive */
52 /* (high) write enable, negative slope, auto trigger */
53 #define TRIG_DELAY_DEF 0+DELAY_CONSTANT /* Default trigger delay (desired delay + DELAY_CONSTANT) */
54 #define TRIG_LEVEL_DEF 128 /* Default trigger level */
55 #define DEFAULT_SAMPLE_RATE 19000000 /* Default sample rate */
56 #define TRIG_PERIOD_DEF CLK_FREQ/DEFAULT_SAMPLE_RATE /* Translates into this trigger period */
57
58

```

```

1  /*
2  * system.h - SOPC Builder system and BSP software package information
3  *
4  * Machine generated for CPU 'nios' in SOPC Builder design 'sopc_scope_sys'
5  * SOPC Builder design path: C:/Users/tago/Dropbox/OUT/EE52/quartus/sopc_scope_sys.sopcinfo
6  *
7  * Generated: Wed Jun 11 15:26:36 PDT 2014
8  */
9
10 /*
11 * DO NOT MODIFY THIS FILE
12 *
13 * Changing this file will have subtle consequences
14 * which will almost certainly lead to a nonfunctioning
15 * system. If you do modify this file, be aware that your
16 * changes will be overwritten and lost when this file
17 * is generated again.
18 *
19 * DO NOT MODIFY THIS FILE
20 */
21
22 /*
23 * License Agreement
24 *
25 * Copyright (c) 2008
26 * Altera Corporation, San Jose, California, USA.
27 * All rights reserved.
28 *
29 * Permission is hereby granted, free of charge, to any person obtaining a
30 * copy of this software and associated documentation files (the "Software"),
31 * to deal in the Software without restriction, including without limitation
32 * the rights to use, copy, modify, merge, publish, distribute, sublicense,
33 * and/or sell copies of the Software, and to permit persons to whom the
34 * Software is furnished to do so, subject to the following conditions:
35 *
36 * The above copyright notice and this permission notice shall be included in
37 * all copies or substantial portions of the Software.
38 *
39 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
40 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
41 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
42 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
43 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
44 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
45 * DEALINGS IN THE SOFTWARE.
46 *
47 * This agreement shall be governed in all respects by the laws of the State
48 * of California and by the laws of the United States of America.
49 */
50
51 #ifndef __SYSTEM_H_
52 #define __SYSTEM_H_
53
54 /* Include definitions from linker script generator */
55 #include "linker.h"
56
57
58 /*
59 * CPU configuration
60 *
61 */
62
63 #define ALT_CPU_ARCHITECTURE "altera_nios2_qsys"
64 #define ALT_CPU_BIG_ENDIAN 0
65 #define ALT_CPU_BREAK_ADDR 0x00240820
66 #define ALT_CPU_CPU_FREQ 50000000u
67 #define ALT_CPU_CPU_ID_SIZE 1
68 #define ALT_CPU_CPU_ID_VALUE 0x00000000
69 #define ALT_CPU_CPU_IMPLEMENTATION "tiny"
70 #define ALT_CPU_DATA_ADDR_WIDTH 0x16
71 #define ALT_CPU_DCACHE_LINE_SIZE 0
72 #define ALT_CPU_DCACHE_LINE_SIZE_LOG2 0
73 #define ALT_CPU_DCACHE_SIZE 0
74 #define ALT_CPU_EXCEPTION_ADDR 0x00180020
75 #define ALT_CPU_FLUSHDA_SUPPORTED

```

```

76 #define ALT_CPU_FREQ 50000000
77 #define ALT_CPU_HARDWARE_DIVIDE_PRESENT 0
78 #define ALT_CPU_HARDWARE_MULTIPLY_PRESENT 0
79 #define ALT_CPU_HARDWARE_MULX_PRESENT 0
80 #define ALT_CPU_HAS_DEBUG_CORE 1
81 #define ALT_CPU_HAS_DEBUG_STUB
82 #define ALT_CPU_HAS_JMPI_INSTRUCTION
83 #define ALT_CPU_ICACHE_LINE_SIZE 0
84 #define ALT_CPU_ICACHE_LINE_SIZE_LOG2 0
85 #define ALT_CPU_ICACHE_SIZE 0
86 #define ALT_CPU_INST_ADDR_WIDTH 0x16
87 #define ALT_CPU_NAME "nios"
88 #define ALT_CPU_RESET_ADDR 0x00180000
89
90
91 /*
92  * CPU configuration (with legacy prefix - don't use these anymore)
93  *
94  */
95
96 #define NIOS2_BIG_ENDIAN 0
97 #define NIOS2_BREAK_ADDR 0x00240820
98 #define NIOS2_CPU_FREQ 50000000u
99 #define NIOS2_CPU_ID_SIZE 1
100 #define NIOS2_CPU_ID_VALUE 0x00000000
101 #define NIOS2_CPU_IMPLEMENTATION "tiny"
102 #define NIOS2_DATA_ADDR_WIDTH 0x16
103 #define NIOS2_DCACHE_LINE_SIZE 0
104 #define NIOS2_DCACHE_LINE_SIZE_LOG2 0
105 #define NIOS2_DCACHE_SIZE 0
106 #define NIOS2_EXCEPTION_ADDR 0x00180020
107 #define NIOS2_FLUSHDA_SUPPORTED
108 #define NIOS2_HARDWARE_DIVIDE_PRESENT 0
109 #define NIOS2_HARDWARE_MULTIPLY_PRESENT 0
110 #define NIOS2_HARDWARE_MULX_PRESENT 0
111 #define NIOS2_HAS_DEBUG_CORE 1
112 #define NIOS2_HAS_DEBUG_STUB
113 #define NIOS2_HAS_JMPI_INSTRUCTION
114 #define NIOS2_ICACHE_LINE_SIZE 0
115 #define NIOS2_ICACHE_LINE_SIZE_LOG2 0
116 #define NIOS2_ICACHE_SIZE 0
117 #define NIOS2_INST_ADDR_WIDTH 0x16
118 #define NIOS2_RESET_ADDR 0x00180000
119
120
121 /*
122  * Define for each module class mastered by the CPU
123  *
124  */
125
126 #define __ALTERA_AVALON_JTAG_UART
127 #define __ALTERA_AVALON_PIO
128 #define __ALTERA_GENERIC_TRISTATE_CONTROLLER
129 #define __ALTERA_NIOS2_QSYS
130
131
132 /*
133  * System configuration
134  *
135  */
136
137 #define ALT_DEVICE_FAMILY "Cyclone III"
138 #define ALT_ENHANCED_INTERRUPT_API_PRESENT
139 #define ALT_IRQ_BASE NULL
140 #define ALT_LOG_PORT "/dev/null"
141 #define ALT_LOG_PORT_BASE 0x0
142 #define ALT_LOG_PORT_DEV null
143 #define ALT_LOG_PORT_TYPE ""
144 #define ALT_NUM_EXTERNAL_INTERRUPT_CONTROLLERS 0
145 #define ALT_NUM_INTERNAL_INTERRUPT_CONTROLLERS 1
146 #define ALT_NUM_INTERRUPT_CONTROLLERS 1
147 #define ALT_STDERR "/dev/jtag"
148 #define ALT_STDERR_BASE 0x241180
149 #define ALT_STDERR_DEV jtag
150 #define ALT_STDERR_IS_JTAG_UART

```



```

151 #define ALT_STDERR_PRESENT
152 #define ALT_STDERR_TYPE "altera_avalon_jtag_uart"
153 #define ALT_STDIN "/dev/jtag"
154 #define ALT_STDIN_BASE 0x241180
155 #define ALT_STDIN_DEV jtag
156 #define ALT_STDIN_IS_JTAG_UART
157 #define ALT_STDIN_PRESENT
158 #define ALT_STDIN_TYPE "altera_avalon_jtag_uart"
159 #define ALT_STDOUT "/dev/jtag"
160 #define ALT_STDOUT_BASE 0x241180
161 #define ALT_STDOUT_DEV jtag
162 #define ALT_STDOUT_IS_JTAG_UART
163 #define ALT_STDOUT_PRESENT
164 #define ALT_STDOUT_TYPE "altera_avalon_jtag_uart"
165 #define ALT_SYSTEM_NAME "sopc_scope_sys"
166
167
168 /*
169  * fifo_data configuration
170  *
171  */
172
173 #define ALT_MODULE_CLASS_fifo_data altera_avalon_pio
174 #define FIFO_DATA_BASE 0x241140
175 #define FIFO_DATA_BIT_CLEARING_EDGE_REGISTER 0
176 #define FIFO_DATA_BIT_MODIFYING_OUTPUT_REGISTER 0
177 #define FIFO_DATA_CAPTURE 0
178 #define FIFO_DATA_DATA_WIDTH 8
179 #define FIFO_DATA_DO_TEST_BENCH_WIRING 0
180 #define FIFO_DATA_DRIVEN_SIM_VALUE 0
181 #define FIFO_DATA_EDGE_TYPE "NONE"
182 #define FIFO_DATA_FREQ 50000000
183 #define FIFO_DATA_HAS_IN 1
184 #define FIFO_DATA_HAS_OUT 0
185 #define FIFO_DATA_HAS_TRI 0
186 #define FIFO_DATA_IRQ -1
187 #define FIFO_DATA_IRQ_INTERRUPT_CONTROLLER_ID -1
188 #define FIFO_DATA_IRQ_TYPE "NONE"
189 #define FIFO_DATA_NAME "/dev/fifo_data"
190 #define FIFO_DATA_RESET_VALUE 0
191 #define FIFO_DATA_SPAN 16
192 #define FIFO_DATA_TYPE "altera_avalon_pio"
193
194
195 /*
196  * fifo_full configuration
197  *
198  */
199
200 #define ALT_MODULE_CLASS_fifo_full altera_avalon_pio
201 #define FIFO_FULL_BASE 0x241130
202 #define FIFO_FULL_BIT_CLEARING_EDGE_REGISTER 0
203 #define FIFO_FULL_BIT_MODIFYING_OUTPUT_REGISTER 0
204 #define FIFO_FULL_CAPTURE 1
205 #define FIFO_FULL_DATA_WIDTH 1
206 #define FIFO_FULL_DO_TEST_BENCH_WIRING 0
207 #define FIFO_FULL_DRIVEN_SIM_VALUE 0
208 #define FIFO_FULL_EDGE_TYPE "RISING"
209 #define FIFO_FULL_FREQ 50000000
210 #define FIFO_FULL_HAS_IN 1
211 #define FIFO_FULL_HAS_OUT 0
212 #define FIFO_FULL_HAS_TRI 0
213 #define FIFO_FULL_IRQ 4
214 #define FIFO_FULL_IRQ_INTERRUPT_CONTROLLER_ID 0
215 #define FIFO_FULL_IRQ_TYPE "EDGE"
216 #define FIFO_FULL_NAME "/dev/fifo_full"
217 #define FIFO_FULL_RESET_VALUE 0
218 #define FIFO_FULL_SPAN 16
219 #define FIFO_FULL_TYPE "altera_avalon_pio"
220
221
222 /*
223  * hal configuration
224  *
225  */

```

```

226
227 #define ALT_MAX_FD 32
228 #define ALT_SYS_CLK none
229 #define ALT_TIMESTAMP_CLK none
230
231
232 /*
233  * jtag configuration
234  *
235  */
236
237 #define ALT_MODULE_CLASS_jtag altera_avalon_jtag_uart
238 #define JTAG_BASE 0x241180
239 #define JTAG_IRQ 0
240 #define JTAG_IRQ_INTERRUPT_CONTROLLER_ID 0
241 #define JTAG_NAME "/dev/jtag"
242 #define JTAG_READ_DEPTH 64
243 #define JTAG_READ_THRESHOLD 8
244 #define JTAG_SPAN 8
245 #define JTAG_TYPE "altera_avalon_jtag_uart"
246 #define JTAG_WRITE_DEPTH 64
247 #define JTAG_WRITE_THRESHOLD 8
248
249
250 /*
251  * pio_0 configuration
252  *
253  */
254
255 #define ALT_MODULE_CLASS_pio_0 altera_avalon_pio
256 #define PIO_0_BASE 0x2410a0
257 #define PIO_0_BIT_CLEARING_EDGE_REGISTER 1
258 #define PIO_0_BIT_MODIFYING_OUTPUT_REGISTER 1
259 #define PIO_0_CAPTURE 1
260 #define PIO_0_DATA_WIDTH 6
261 #define PIO_0_DO_TEST_BENCH_WIRING 0
262 #define PIO_0_DRIVEN_SIM_VALUE 0
263 #define PIO_0_EDGE_TYPE "FALLING"
264 #define PIO_0_FREQ 50000000
265 #define PIO_0_HAS_IN 1
266 #define PIO_0_HAS_OUT 0
267 #define PIO_0_HAS_TRI 0
268 #define PIO_0_IRQ 1
269 #define PIO_0_IRQ_INTERRUPT_CONTROLLER_ID 0
270 #define PIO_0_IRQ_TYPE "EDGE"
271 #define PIO_0_NAME "/dev/pio_0"
272 #define PIO_0_RESET_VALUE 0
273 #define PIO_0_SPAN 32
274 #define PIO_0_TYPE "altera_avalon_pio"
275
276
277 /*
278  * ram configuration
279  *
280  */
281
282 #define ALT_MODULE_CLASS_ram altera_generic_tristate_controller
283 #define RAM_BASE 0x220000
284 #define RAM_IRQ -1
285 #define RAM_IRQ_INTERRUPT_CONTROLLER_ID -1
286 #define RAM_NAME "/dev/ram"
287 #define RAM_SPAN 131072
288 #define RAM_TYPE "altera_generic_tristate_controller"
289
290
291 /*
292  * rom configuration
293  *
294  */
295
296 #define ALT_MODULE_CLASS_rom altera_generic_tristate_controller
297 #define ROM_BASE 0x180000
298 #define ROM_IRQ -1
299 #define ROM_IRQ_INTERRUPT_CONTROLLER_ID -1
300 #define ROM_NAME "/dev/rom"

```

```

301 #define ROM_SPAN 524288
302 #define ROM_TYPE "altera_generic_tristate_controller"
303
304
305 /*
306  * trig_ctrl configuration
307  *
308  */
309
310 #define ALT_MODULE_CLASS_trig_ctrl altera_avalon_pio
311 #define TRIG_CTRL_BASE 0x241060
312 #define TRIG_CTRL_BIT_CLEARING_EDGE_REGISTER 0
313 #define TRIG_CTRL_BIT_MODIFYING_OUTPUT_REGISTER 1
314 #define TRIG_CTRL_CAPTURE 0
315 #define TRIG_CTRL_DATA_WIDTH 5
316 #define TRIG_CTRL_DO_TEST_BENCH_WIRING 0
317 #define TRIG_CTRL_DRIVEN_SIM_VALUE 0
318 #define TRIG_CTRL_EDGE_TYPE "NONE"
319 #define TRIG_CTRL_FREQ 50000000
320 #define TRIG_CTRL_HAS_IN 0
321 #define TRIG_CTRL_HAS_OUT 1
322 #define TRIG_CTRL_HAS_TRI 0
323 #define TRIG_CTRL_IRQ -1
324 #define TRIG_CTRL_IRQ_INTERRUPT_CONTROLLER_ID -1
325 #define TRIG_CTRL_IRQ_TYPE "NONE"
326 #define TRIG_CTRL_NAME "/dev/trig_ctrl"
327 #define TRIG_CTRL_RESET_VALUE 3
328 #define TRIG_CTRL_SPAN 32
329 #define TRIG_CTRL_TYPE "altera_avalon_pio"
330
331
332 /*
333  * trig_delay configuration
334  *
335  */
336
337 #define ALT_MODULE_CLASS_trig_delay altera_avalon_pio
338 #define TRIG_DELAY_BASE 0x241120
339 #define TRIG_DELAY_BIT_CLEARING_EDGE_REGISTER 0
340 #define TRIG_DELAY_BIT_MODIFYING_OUTPUT_REGISTER 0
341 #define TRIG_DELAY_CAPTURE 0
342 #define TRIG_DELAY_DATA_WIDTH 32
343 #define TRIG_DELAY_DO_TEST_BENCH_WIRING 0
344 #define TRIG_DELAY_DRIVEN_SIM_VALUE 0
345 #define TRIG_DELAY_EDGE_TYPE "NONE"
346 #define TRIG_DELAY_FREQ 50000000
347 #define TRIG_DELAY_HAS_IN 0
348 #define TRIG_DELAY_HAS_OUT 1
349 #define TRIG_DELAY_HAS_TRI 0
350 #define TRIG_DELAY_IRQ -1
351 #define TRIG_DELAY_IRQ_INTERRUPT_CONTROLLER_ID -1
352 #define TRIG_DELAY_IRQ_TYPE "NONE"
353 #define TRIG_DELAY_NAME "/dev/trig_delay"
354 #define TRIG_DELAY_RESET_VALUE 1
355 #define TRIG_DELAY_SPAN 16
356 #define TRIG_DELAY_TYPE "altera_avalon_pio"
357
358
359 /*
360  * trig_level configuration
361  *
362  */
363
364 #define ALT_MODULE_CLASS_trig_level altera_avalon_pio
365 #define TRIG_LEVEL_BASE 0x241150
366 #define TRIG_LEVEL_BIT_CLEARING_EDGE_REGISTER 0
367 #define TRIG_LEVEL_BIT_MODIFYING_OUTPUT_REGISTER 0
368 #define TRIG_LEVEL_CAPTURE 0
369 #define TRIG_LEVEL_DATA_WIDTH 8
370 #define TRIG_LEVEL_DO_TEST_BENCH_WIRING 0
371 #define TRIG_LEVEL_DRIVEN_SIM_VALUE 0
372 #define TRIG_LEVEL_EDGE_TYPE "NONE"
373 #define TRIG_LEVEL_FREQ 50000000
374 #define TRIG_LEVEL_HAS_IN 0
375 #define TRIG_LEVEL_HAS_OUT 1

```

```

376 #define TRIG_LEVEL_HAS_TRI 0
377 #define TRIG_LEVEL_IRQ -1
378 #define TRIG_LEVEL_IRQ_INTERRUPT_CONTROLLER_ID -1
379 #define TRIG_LEVEL_IRQ_TYPE "NONE"
380 #define TRIG_LEVEL_NAME "/dev/trig_level"
381 #define TRIG_LEVEL_RESET_VALUE 0
382 #define TRIG_LEVEL_SPAN 16
383 #define TRIG_LEVEL_TYPE "altera_avalon_pio"
384
385
386 /*
387  * trig_period configuration
388  *
389  */
390
391 #define ALT_MODULE_CLASS_trig_period altera_avalon_pio
392 #define TRIG_PERIOD_BASE 0x241160
393 #define TRIG_PERIOD_BIT_CLEARING_EDGE_REGISTER 0
394 #define TRIG_PERIOD_BIT_MODIFYING_OUTPUT_REGISTER 0
395 #define TRIG_PERIOD_CAPTURE 0
396 #define TRIG_PERIOD_DATA_WIDTH 32
397 #define TRIG_PERIOD_DO_TEST_BENCH_WIRING 0
398 #define TRIG_PERIOD_DRIVEN_SIM_VALUE 0
399 #define TRIG_PERIOD_EDGE_TYPE "NONE"
400 #define TRIG_PERIOD_FREQ 50000000
401 #define TRIG_PERIOD_HAS_IN 0
402 #define TRIG_PERIOD_HAS_OUT 1
403 #define TRIG_PERIOD_HAS_TRI 0
404 #define TRIG_PERIOD_IRQ -1
405 #define TRIG_PERIOD_IRQ_INTERRUPT_CONTROLLER_ID -1
406 #define TRIG_PERIOD_IRQ_TYPE "NONE"
407 #define TRIG_PERIOD_NAME "/dev/trig_period"
408 #define TRIG_PERIOD_RESET_VALUE 1
409 #define TRIG_PERIOD_SPAN 16
410 #define TRIG_PERIOD_TYPE "altera_avalon_pio"
411
412
413 /*
414  * vram configuration
415  *
416  */
417
418 #define ALT_MODULE_CLASS_vram altera_generic_tristate_controller
419 #define VRAM_BASE 0x0
420 #define VRAM_IRQ -1
421 #define VRAM_IRQ_INTERRUPT_CONTROLLER_ID -1
422 #define VRAM_NAME "/dev/vram"
423 #define VRAM_SPAN 1048576
424 #define VRAM_TYPE "altera_generic_tristate_controller"
425
426 #endif /* __SYSTEM_H */
427

```