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

**Technical Manual** 

Santiago Navonne

 $\mathrm{June}\ 2014$ 

# Contents

1.	Introduction	3
2.	Hardware	3
	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
3	Software	80
٥.	3.1. System Overview	
	3.2. Initialization and Main Loop	
	3.3. User Interface	
	3.4. Trace Processing	
	3.5. Key Hardware Interface	
	3.6. Display Hardware Interface	
	3.7. Analog Hardware Interface	
		. 00
Αp	ppendices	85
Α.	Original Documents	85
В.	Software Code	93

# 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.

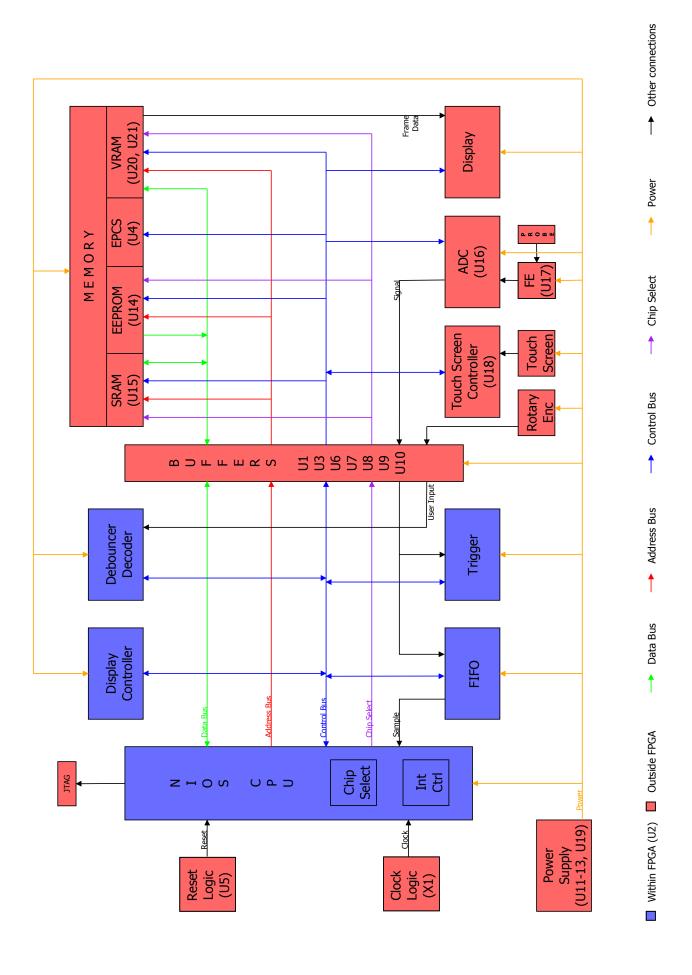


Figure 1: Highest level block diagram of the SoPC Oscilloscope system. The diagram is described in Section 2.1.

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.

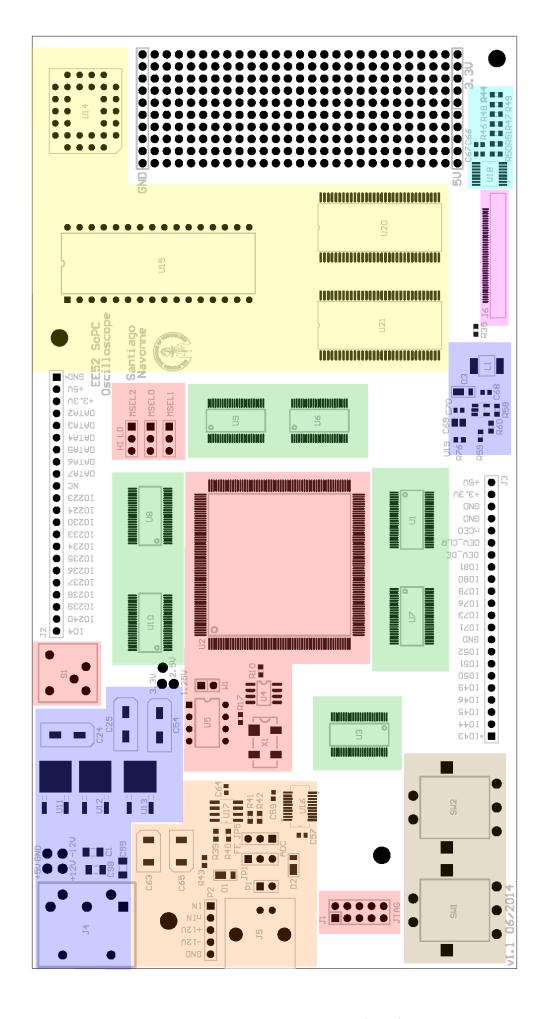


Figure 2: Front side of the system's Printed Circuit Board (PCB). The color each section is highlighted and identifies the corresponding block.

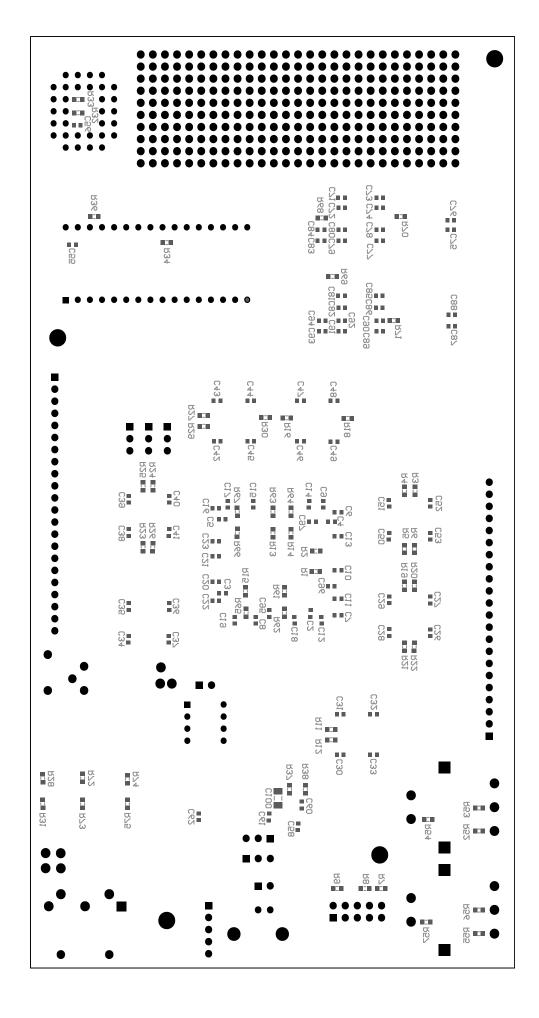


Figure 3: Back side of the system's Printed Circuit Board (PCB). Note that this side is only used for routing and placement of passive parts, such as resistors and capacitors.

Device	Address Range	Size (b	View
fifo data	0x00241140 - 0x0024114F	16	jtag
fifo full	0x00241130 - 0x0024113F	16	+uiu usuisa
jtag	0x00241180 - 0x00241187	8	trig period trig level
pio 0	0x002410A0 - 0x002410BF	32	fifo data
ram	0x00220000 - 0x0023FFFF	131072	fifo full
rom	0x00180000 - 0x001FFFFF	524288	trig delay
trig ctrl	0x00241060 - 0x0024107F	32	crig delay
trig delay	0x00241120 - 0x0024112F	16	pio 0
trig_level	0x00241150 - 0x0024115F	16	p10_0
trig_period	0x00241160 - 0x0024116F	16	trig ctrl
vram	0x00000000 - 0x000FFFFF	1048576	CIIG CCII
			rom CSO
			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 nCSO; 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  $PIO_i$ 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 singleword 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  $I^2C$  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.

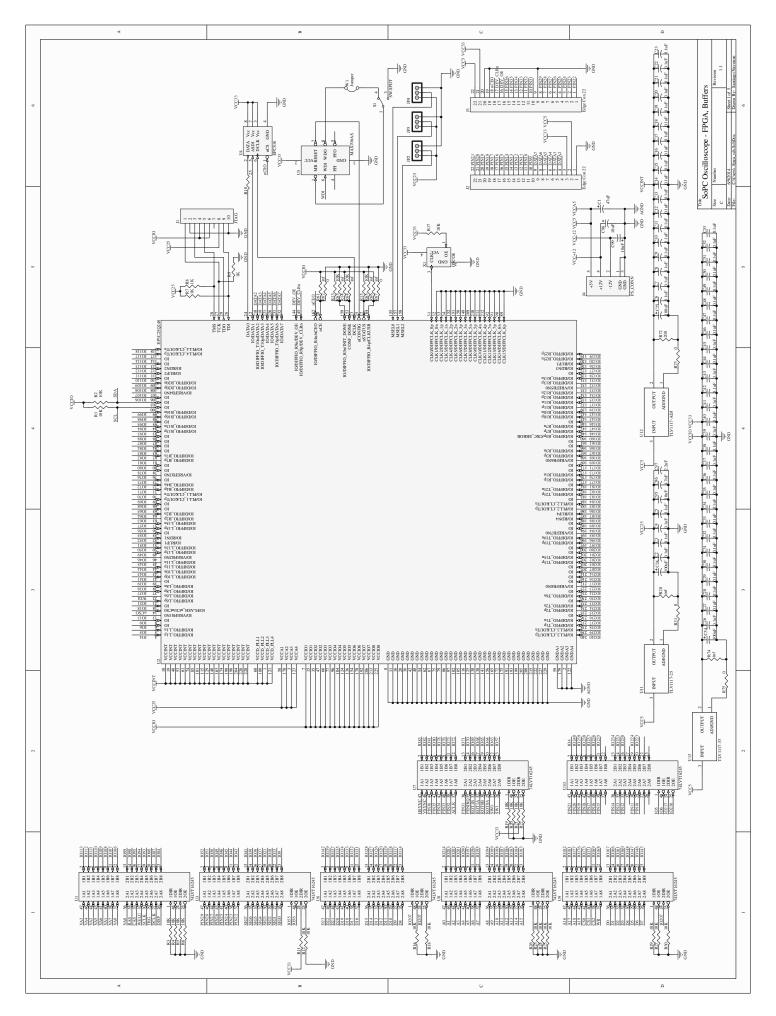


Figure 5: Schematic of the FPGA and related components. Also included are the buffers and power supply. The document is described in Section 2.2.

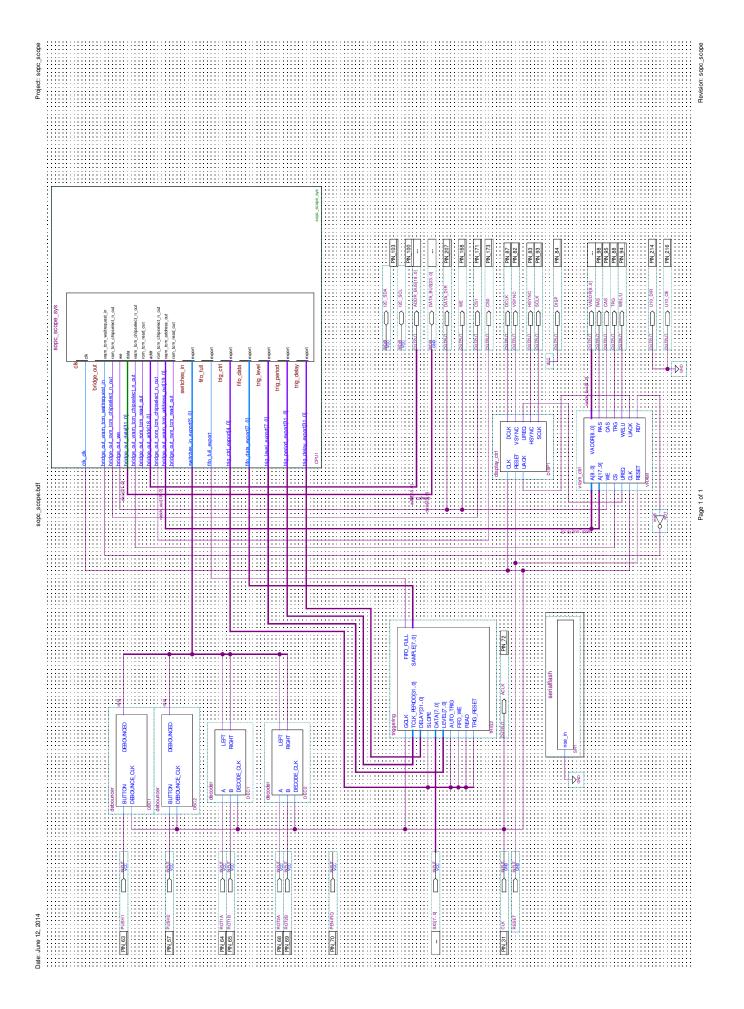


Figure 6: Main block diagram of the FPGA's internal design. The document is described in Section 2.2.

### 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
   --
      This is an implementation of a trigger for a digital oscilloscope in
5
   --
       VHDL. There are three inputs to the system, one selects the trigger
   --
6
       slope and the other two determine the relationship between the trigger
7
       level and the signal level. The only output is a trigger signal which
8
       indicates a trigger event has occurred.
9
   --
10
       The file contains multiple architectures for a Moore state machine
11
   --
       implementation to demonstrate the different ways of building a state
12
       machine.
13
14
   --
15
      Revision History:
16
   ___
   --
        13 Apr 04 Glen George
                                         Initial revision.
17
           4 Nov 05 Glen George
                                         Updated comments.
18
          13 Feb 10 Glen George Updated comments.

14 Santiage Comments.

Updated comments.

Added more evample
         17 Nov 07 Glen George
19
                                         Added more example architectures.
20
          01 Mar 14 Santiago Navonne Removed unnecessary architectures.
21
22
23
24
25
   -- bring in the necessary packages
26
27
   library ieee;
  use ieee.std_logic_1164.all;
28
29
30
31
   -- Oscilloscope Digital Trigger entity declaration
32
33
34
   entity ScopeTrigger is
35
       port (
36
                       : in std_logic;
: in std_logic;
37
                                                -- trigger slope (1 -> negative, 0 -> positive)
                                                -- signal and trigger levels equal
38
           TEQ
                       : in std_logic;
                                                -- signal level < trigger level
           T_{L}T
39
                      : in std logic;
                                                -- clock
40
           Reset
                      : in std logic;
                                                -- reset the system
41
           TrigEvent : out std logic
                                                -- a trigger event has occurred
42
43
       );
   end ScopeTrigger;
44
45
46
47
       Oscilloscope Digital Trigger Moore State Machine
48
          State Assignment Architecture
   __
49
50
       This architecture just shows the basic state machine syntax when the state
51
       assignments are made manually. This is useful for minimizing output
52
       decoding logic and avoiding glitches in the output (due to the decoding
53
       logic).
54
55
56
   architecture assign_statebits of ScopeTrigger is
57
58
       subtype states is std_logic_vector(2 downto 0); -- state type
59
60
       -- define the actual states as constants
61
       constant IDLE
                        : states := "000"; -- waiting for start of trigger event
62
       constant WAIT_POS : states := "001"; -- waiting for positive slope trigger
constant WAIT_NEG : states := "010"; -- waiting for negative slope trigger
63
64
                           : states := "100"; -- got a trigger event
       constant TRIGGER
65
66
67
       signal CurrentState : states; -- current state
68
```

```
signal NextState : states; -- next state
 69
70
71
   begin
72
73
        -- the output is always the high bit of the state encoding
74
        TrigEvent <= CurrentState(2);</pre>
75
 76
77
        -- compute the next state (function of current state and inputs)
78
79
        transition: process (Reset, TS, TEQ, TLT, CurrentState)
80
        begin
81
82
            case CurrentState is
                                            -- do the state transition/output
83
84
                when IDLE =>
                                             -- in idle state, do transition
85
                    if (TS = '0') and TLT = '1' and TEQ = '0') then
86
                        NextState <= WAIT POS;</pre>
                                                   -- below trigger and + slope
87
                    elsif (TS = '1' and TLT = '0' and TEQ = '0') then
88
                        NextState <= WAIT_NEG;</pre>
                                                     -- above trigger and - slope
89
90
                                                      -- trigger not possible yet
91
                        NextState <= IDLE;</pre>
                    end if;
92
93
                when WAIT POS =>
                                             -- waiting for positive slope trigger
94
                    if (T\overline{S} = '0' \text{ and } TLT = '1') then
95
                    NextState <= WAIT_POS; -- no trigger yet elsif (TS = '0' and TLT = '0') then
96
97
98
                        NextState <= TRIGGER;</pre>
                                                    -- got a trigger
99
                        NextState <= IDLE;</pre>
                                                    -- trigger slope changed
100
                    end if;
101
102
                when WAIT NEG =>
                                             -- waiting for negative slope trigger
103
                    if (\overline{TS} = '1') and \overline{TLT} = '0' and \overline{TEQ} = '0') then
104
105
106
107
108
                        NextState <= IDLE;</pre>
                                                     -- trigger slope changed
109
110
                    end if;
111
                when TRIGGER =>
                                             -- in the trigger state
112
                    NextState <= IDLE; -- always go back to idle
113
114
                    when others =>
115
                         NextState <= IDLE;</pre>
116
117
            end case;
118
119
            if Reset = '1' then
                                             -- reset overrides everything
120
                NextState <= IDLE;
                                             -- go to idle on reset
121
            end if;
122
123
        end process transition;
124
125
126
        -- storage of current state (loads the next state on the clock)
127
128
        process (clk)
129
        begin
130
131
            if clk = '1' then
                                             -- only change on rising edge of clock
132
                CurrentState <= NextState; -- save the new state information
133
            end if;
134
135
        end process;
136
```

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

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 (TDFF1) 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 setting are being changed.

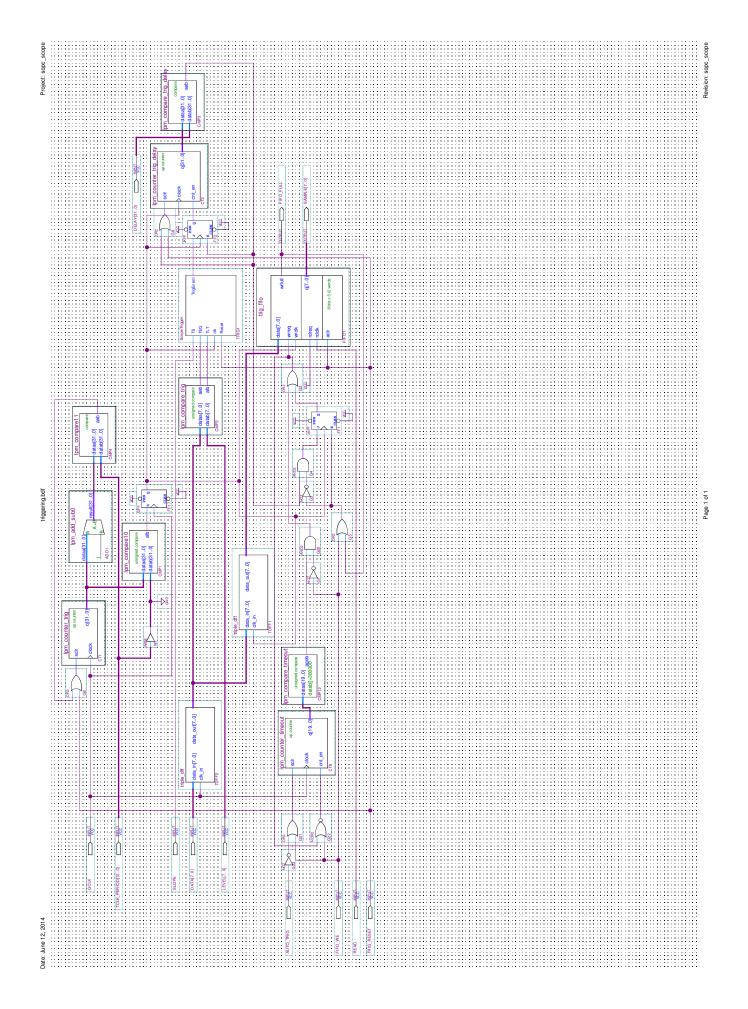


Figure 7: Block diagram of the triggering logic within the FPGA. The document is described in Section 2.2.2.

#### 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 deboucing the signal again) and output on line *DEBOUNCED*.

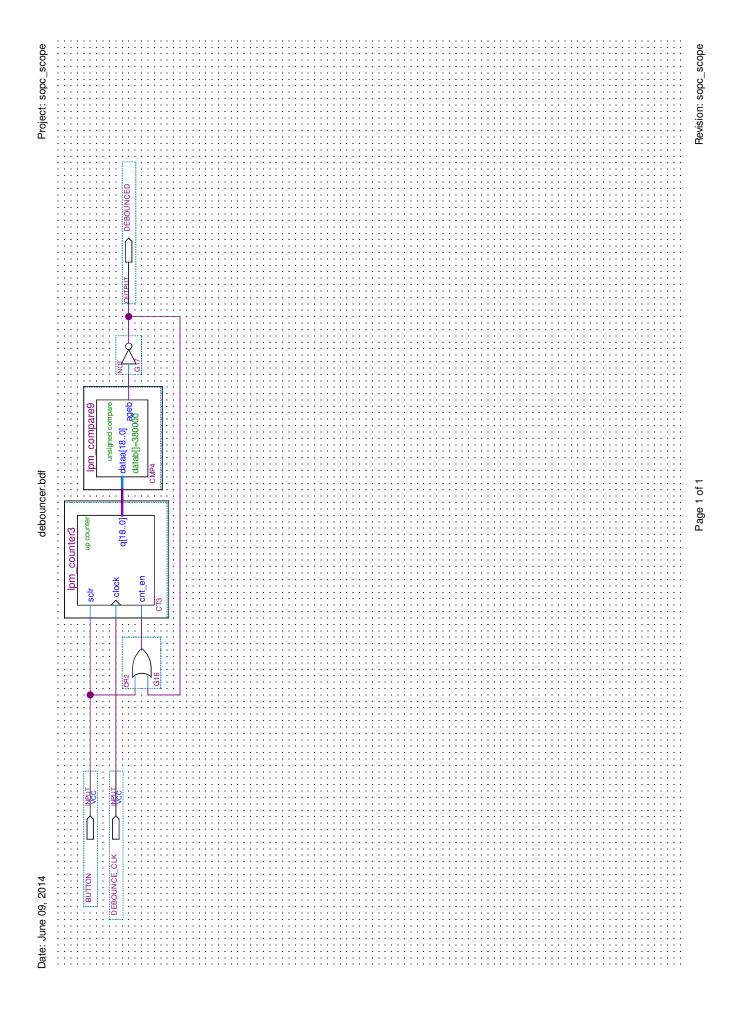


Figure 8: Block diagram of the push-button debouncer component within the FPGA. The document is described in Section 2.2.3.

#### 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.

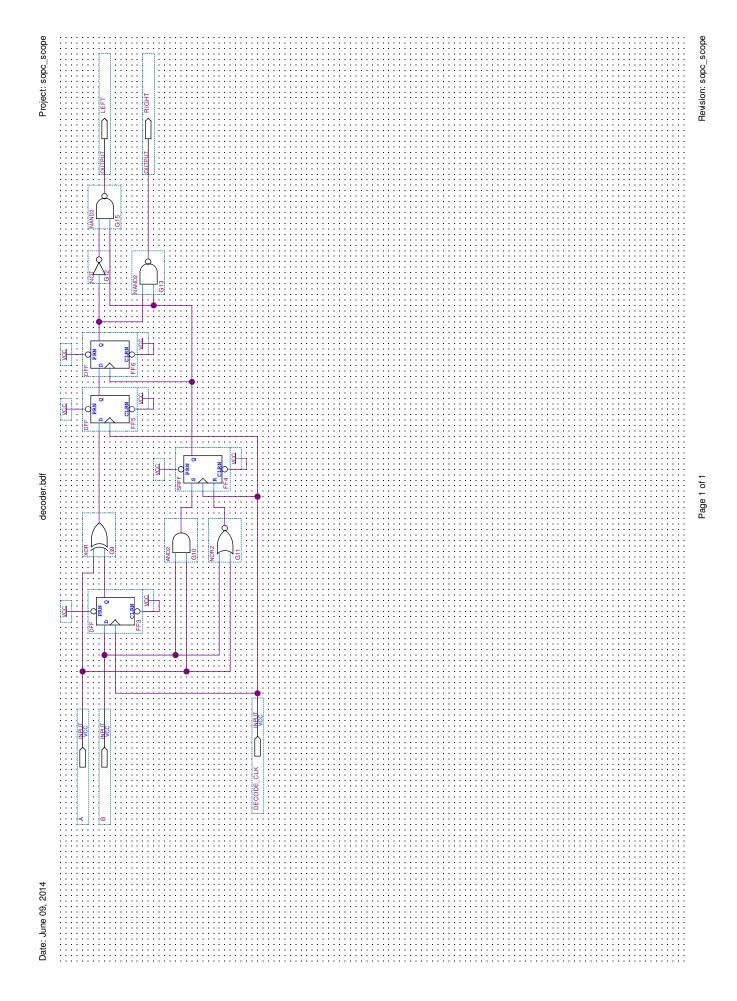


Figure 9: Block diagram of the rotary encoder decoder component within the FPGA. The document is described in Section 2.2.

#### 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
   --
      Implementation of the VRAM Controller for the SoPC Oscilloscope project.
5
   --
      The state machine generates the necessary timing signals for the VRAM
6
      based on the needs of the CPU and display controller. Additionally,
7
      it refreshes the VRAM as necessary whenever no other cycle is being
8
      performed.
9
      The inputs to the system determine what action needs to be performed:
10
   -- cs+we requests a read or a write, while ureq requests a SAM row update.
11
      The system then outputs the necessary timing signals, and a rdy/uack
12
      signal to notify the sender of the end of the cycle.
13
      The state machine is implemented using a Moore state machine and a state
14
   --
       assignment architecture.
15
   __
16
17
   --
      Revision History:
18
        13 Apr 04 Glen George Initial template.
   __
19
         20 Feb 14 Santiago Navonne Initial revision.
20
   __
21
22
   23
24
   -- bring in the necessary packages
25
  library ieee;
26
  use ieee.std_logic_1164.all;
27
28
29
30
   -- Oscilloscope VRAM Controller entity declaration
31
32
33
   entity VRAMCtrl is
34
      port (
35
                   : in std_logic;
: in std_logic;
: in std_logic;
                                            -- read / not write
36
           we
37
           CS
                                            -- chip select
                                            -- serial row update request
           ureq
38
                    : in std_logic;
                                            -- clock
          clk
39
          Reset : in std logic;
                                            -- reset the system
40
          ras
                  : out std logic;
                                            -- RAS timing signal
41
          cas
                   : out std_logic;
42
                                            -- CAS timing signal
                 : out std_logic;
: out std_logic;
: out std_logic;
: out std_logic;
          trg
                                            -- transfer/read signal
43
           welu
                                            -- write signal
44
                                            -- address source selection
45
          asrc
           arow
                                            -- address row/column selection
46
                   : out std logic;
                                            -- serial row update acknowledge
           uack
47
                  : out std_logic
                                            -- read/write acknowledge
48
           rdy
49
  end VRAMCtrl;
50
51
52
53
54
      Oscilloscope VRAM Controller Moore State Machine
55
56
57
   architecture assign_statebits of VRAMCtrl is
58
59
       subtype states is std_logic_vector(10 downto 0); -- state type
60
61
       -- define the actual states as constants
62
63
       -- bits are: RAS CAS TRG WE ASRC AROW UACK RDY ID[2..0]
64
       constant IDLE : states := "11111100000"; -- waiting for events
65
66
      constant READ1 : states := "011111100000"; -- read state 1
constant READ2 : states := "01011000000"; -- read state 2
67
68
```

```
constant READ3
                             : states := "00011000000"; -- read state 3
 69
                             : states := "00011101000"; -- read state 4
        constant READ4
 70
                             : states := "11111100001"; -- read state 5
 71
        constant READ5
                             : states := "11111100010"; -- read state 6
        constant READ6
 72
73
        constant WRITE1
                             : states := "01111100001"; -- write state 1
74
                              : states := "01101000000";
                                                            -- write state 2
75
        constant WRITE2
                              : states := "00101001000";
                                                            -- write state 3
        constant
                  WRITE3
 76
                              : states := "11111100011"; -- write state 4
        constant WRITE4
77
                             : states := "11111100100"; -- write state 5
        constant WRITE5
78
79
                             : states := "11010100000"; -- serial transfer state 1
        constant SERIAL1
 80
                             : states := "010101000000"; -- serial transfer state 2
        constant SERIAL2
 81
                             : states := "01110000000"; -- serial transfer state 3
: states := "00110000000"; -- serial transfer state 4
        constant SERIAL3
constant SERIAL4
 82
 83
        constant SERIAL5
                            : states := "11111110000"; -- serial transfer state 5
84
        constant SERIAL6 : states := "111111100101"; -- serial transfer state 6
85
86
        constant REFRESH1 : states := "101111100000"; -- refresh state 1
 87
        constant REFRESH2 : states := "00111100001"; -- refresh state 2
88
        constant REFRESH3 : states := "00111100010";
constant REFRESH4 : states := "00111100011";
constant REFRESH5 : states := "111111100110";
                                                            -- refresh state 3
89
90
                                                             -- refresh state 4
 91
                                                             -- refresh state 5
        constant REFRESH6 : states := "111111100111"; -- refresh state 6
92
93
        signal CurrentState : states;
                                               -- current state
94
95
        signal NextState : states;
                                              -- next state
96
97
    begin
98
99
        -- the output is always the 8 highest bits of the encoding
100
        ras <= CurrentState(10);</pre>
101
         cas <= CurrentState(9);</pre>
102
         trg <= CurrentState(8);</pre>
103
         welu <= CurrentState(7);</pre>
104
105
         asrc <= CurrentState(6);</pre>
106
         arow <= CurrentState(5);</pre>
         uack <= CurrentState(4);</pre>
107
         rdy <= CurrentState(3);</pre>
108
109
110
        -- compute the next state (function of current state and inputs)
111
112
        transition: process (Reset, ureq, we, cs, CurrentState)
113
        begin
114
115
            case CurrentState is
                                               -- do the state transition/output
116
117
             -- transition from idle
118
119
                 when IDLE =>
                                                -- in idle state, do transition
                          (ureq = '1') then
120
                          NextState <= SERIAL1;</pre>
121
                                                       -- serial update request has priority
                     elsif (cs = '0' and we = '1') then
122
                          NextState <= READ1;</pre>
                                                      -- read request
123
                      elsif (cs = '0' and we = '0') then
124
125
                          NextState <= WRITE1;</pre>
                                                       -- write request
126
                          NextState <= REFRESH1; -- nothing to do; refresh</pre>
127
128
                     end if;
129
            -- read cycle
130
                 when READ1 =>
                                             -- continue read cycle
131
                     NextState <= READ2;</pre>
132
133
                 when READ2 =>
                                             -- continue read cycle
134
                     NextState <= READ3;</pre>
135
136
```

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

```
205
         end case;
206
207
          208
209
          end if;
210
211
212
      end process transition;
213
214
      -- storage of current state (loads the next state on the clock)
215
216
      process (clk)
217
      begin
218
219
          if clk = '1' then
                                      -- only change on rising edge of clock
220
             CurrentState <= NextState; -- save the new state information
221
222
223
      end process;
224
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, 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.

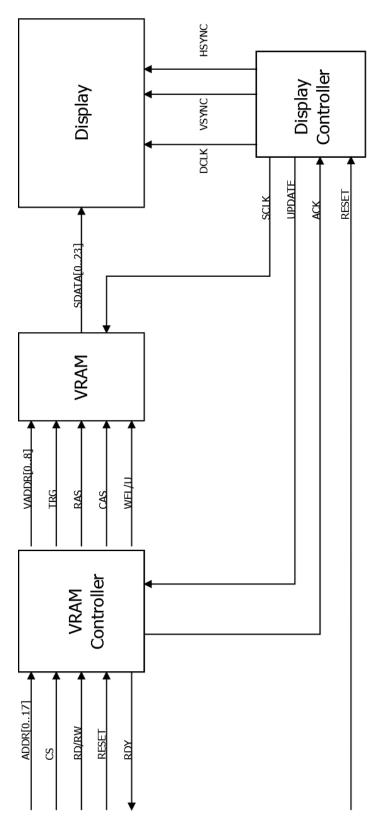


Figure 10: High level, summarizing block diagram of the VRAM and display interface. The VRAM controller listens to the CPU's generic memory controller commands, and generates the necessary timing lines for the VRAM device. The display controller periodically updates the serial row on the VRAM device,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

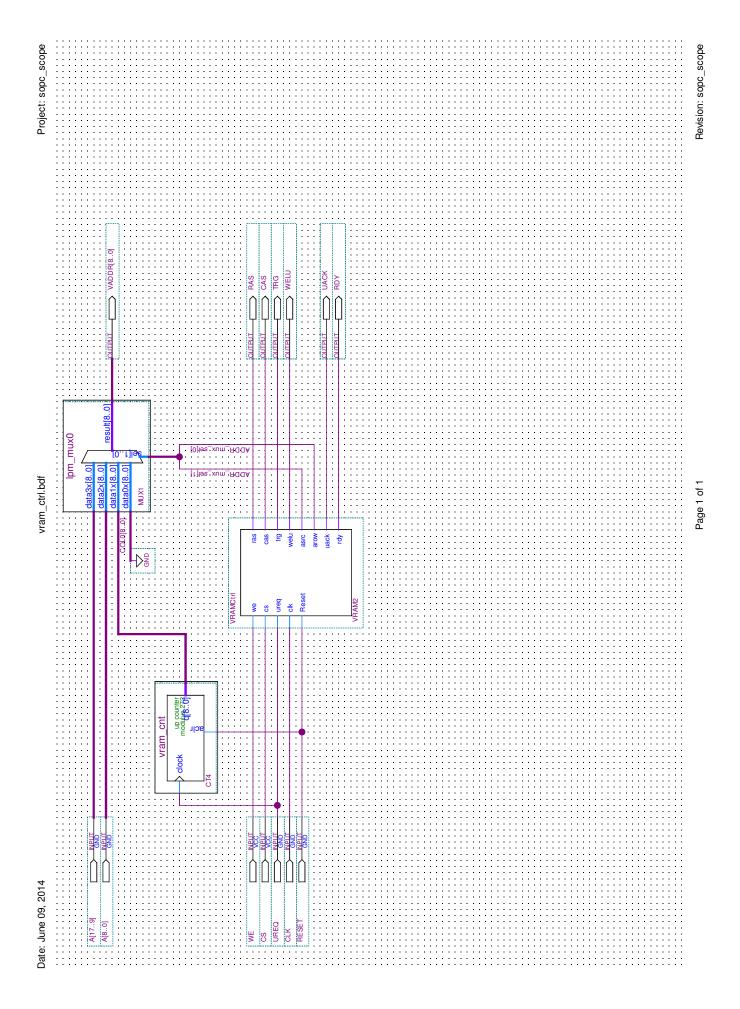


Figure 11: Block diagram of the VRAM controller component within the FPGA. The document is described in Section 2.2.5.

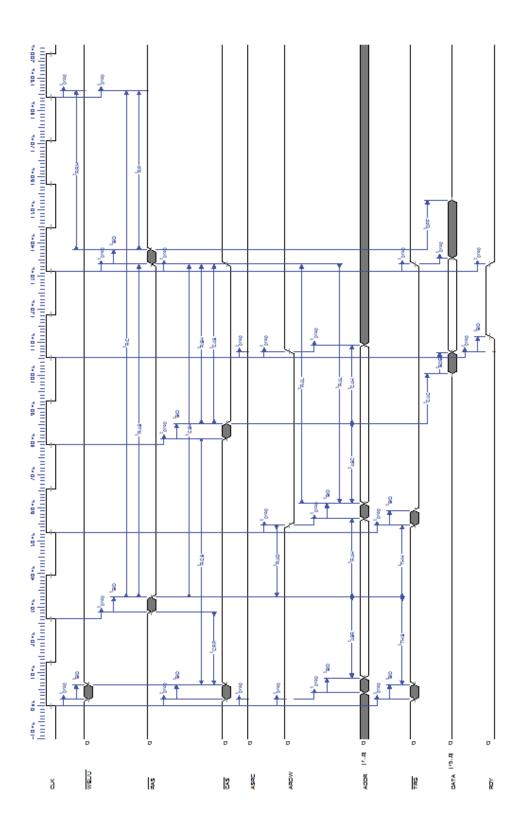


Figure 12: Timing diagram of the read cycle of the VRAM device, described in Section 2.2.5.

General Data					
SYMBOL	DEFINITION	DESCRIPTION	NIE	MAX	NOTES
t prop		Propagation Delay		2ns	
t <sub>BIO</sub>		Buffer I/O Delay		4.5ns	
tboe		Buffer output enable		6.5ns	
t <sub>RAS</sub>		RAS Pulse Width	50ns		
t <sub>CAS</sub>		CAS Pulse Width	10ns		
tcsh		CAS Hold Time	45ns		
t <sub>RC</sub>		Read/Write Cycle Time	104ns		
t <sub>RP</sub>		RAS Precharge Time	40ns		
trsh		RAS Hold Time	15ns		
<sup>t</sup> ван		Row Address Hold Time	10ns		
tash		Row Address Setup	0ns		
Сан		Column Address Hold Time	10ns		
tasc		Column Address Setup Time	Ons		
# <sub>↓</sub>		TRG High Hold Time	10ns		
t⊤HS		TRG High Setup Time	0ns		
tcac		CAS Output Time		15ns	
toff		Data Bus Off Time		15ns	
t <sub>RR</sub> H		Read Command Hold Time (from RAS)	0ns		
<sup>t</sup> 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
<sup>†</sup> ясн		Read Command Hold Time (from CAS)	0ns		
thes		Read Command Setup Time	0ns		
tcrP		CAS to RAS Precharge Time	5ns		
<sup>†</sup> FAL		Column Address to RAS Lead Time	30ns		
<sup>‡</sup> RAD		RAS to Column Address Delay Time	12ns		

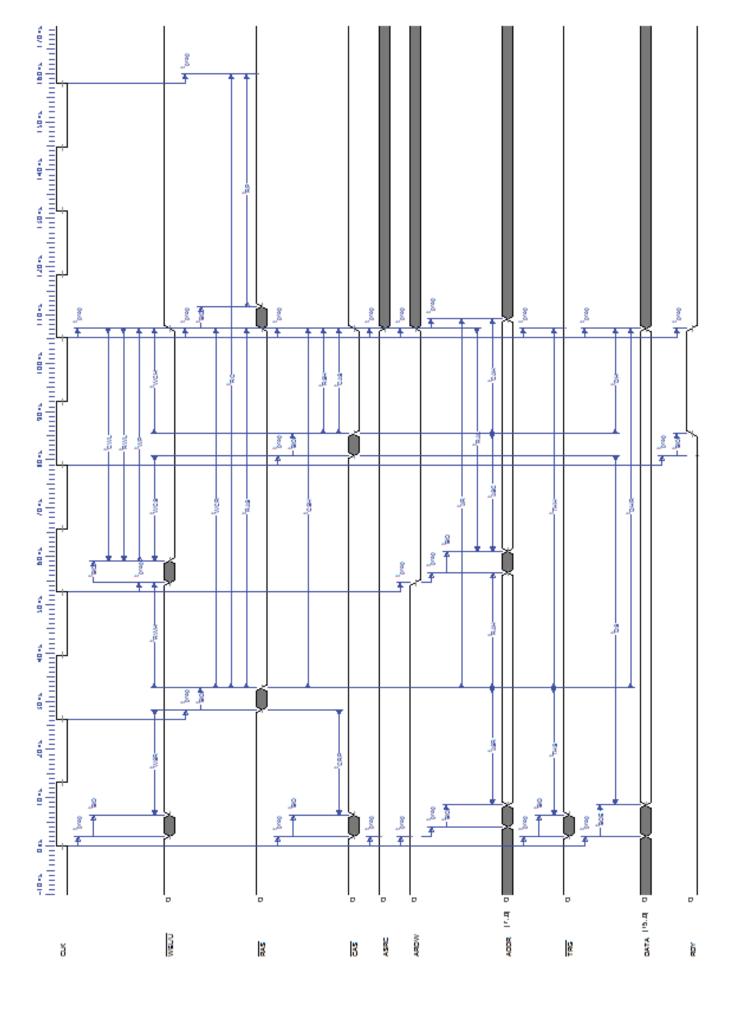


Figure 13: Timing diagram of the write cycle of the VRAM device, described in Section 2.2.5.

General Data					
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOTES
t prop		Propagation Delay		2ns	
t <sub>BIO</sub>		Buffer I/O Delay		4.5ns	
tBOE		Buffer output enable		6.5ns	
t <sub>RAS</sub>		RAS Pulse Width	50ns		
t <sub>CAS</sub>		CAS Pulse Width	10ns		
tcsh		CAS Hold Time	45ns		
t <sub>RC</sub>		Read/Write Cycle Time	104ns		
t <sub>RP</sub>		RAS Precharge Time	40ns		
trsh		RAS Hold Time	15ns		
tван		Row Address Hold Time	10ns		
tash		Row Address Setup	0ns		
tasc		Column Address Setup Time	0ns		
САН		Column Address Hold Time	10ns		
тнs		TRG High Setup Time	0ns		
# <sub>↓</sub>		TRG High Hold Time	10ns		
<sup>†</sup> CRP		CAS to RAS Precharge Time	5ns		
twsR		WE Setup Time	0ns		
tвwн		WE Hold Time	10ns		
twcs		Write Command Setup Time	Ons		

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	MIM	MAX	NOTES
t <sup>†</sup> AR		Column Address Hold Time (from RAS)	50ns		
ф,		Write Command Hold Time	10ns		
tal		Column Address to RAS Lead Time	30ns		
twp		Write Command Pulse Width	10ns		
twcn		Write Command Hold Time (from RAS)	50ns		
t <sub>RWL</sub>		Write Command to RAS Lead Time	15ns		
tcwL		Write Command to CAS Lead Time	15ns		
sa <sub>j</sub>		Data Setup TIme	0ns		
ф		Data Hold Time	10ns		
<sup>†</sup> рнв		Data Hold Time (from RAS)	50ns		

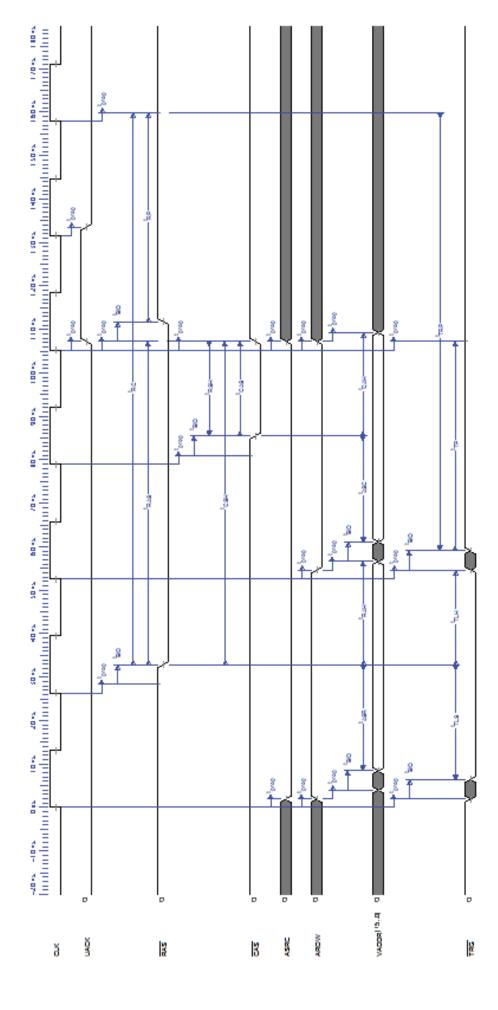


Figure 14: Timing diagram of the serial row transfer cycle of the VRAM device, described in Section 2.2.5.

General Data					
SYMBOL	DEFINITION	DESCRIPTION	NIN	MAX	NOTES
t prop		Propagation Delay		2ns	
t <sub>BIO</sub>		Buffer Input/Output Delay		4.5ns	
t <sub>RAS</sub>		RAS Pulse Width	60ns		
thc		Read/Write Cycle Time	104ns		
t <sub>RP</sub>		RAS Precharge Time	40ns		
t <sub>CSH</sub>		CAS Hold Time	45ns		
t <sub>RSH</sub>		RAS Hold Time	15ns		
t <sub>RCD</sub>		RAS to CAS Delay Time	15ns	42ns	
t <sub>CAS</sub>		CAS Pulse Width	10ns	10000ns	
<sup>t</sup> AR		Column Address Hold Time (from RAS)	50ns		
<sup>t</sup> RAD		RAS to Column Address Delay Time	12ns		
t <sub>RAL</sub>		Column Address to RAS Lead Time	30ns		
tash		Row Address Setup	0ns		
<sup>t</sup> ван		Row Address Hold Time	10ns		
<sup>t</sup> ASC		Column Address Setup Time	0ns		
САН		Column Address Hold Time	10ns		
t⊤LS		TRG Low Setup Time	0ns		
<sup>†</sup> -сн		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	NIM	MAX	NOTES
thsp		RAS to First SC Delay Time	e0ns		
th		TRG to RAS Precharge Time	40ns		
ţтР		TRG Precharge Time	20ns		
tsp		TRG to First SC Delay Time	10ns		

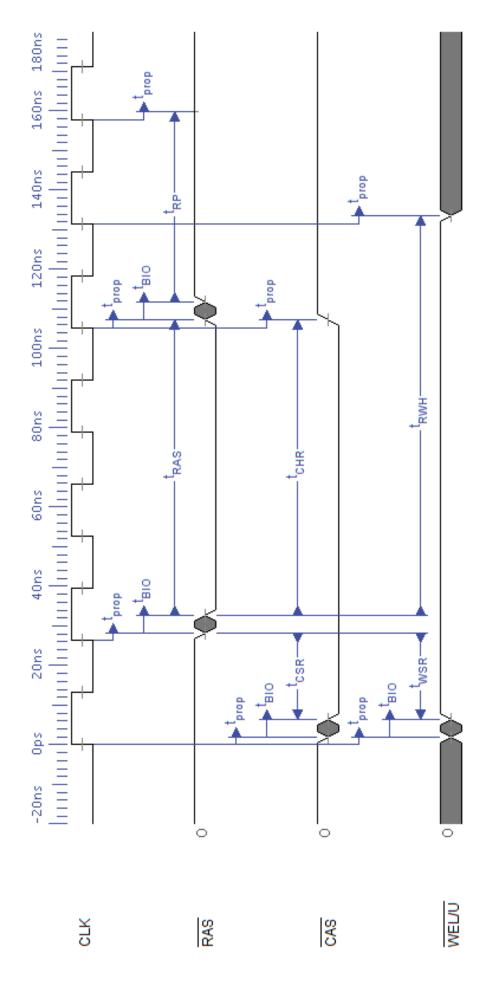


Figure 15: Timing diagram of the refresh cycle of the VRAM device, described in Section 2.2.5.

General Data					
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOTES
t prop		Propagation Delay		2ns	
t <sup>B</sup> IO		Buffer Input/Output delay		4.5ns	
t <sub>RP</sub>		RAS Precharge Time	40ns		
t <sub>RAS</sub>		RAS Pulse Width	50ns	10000ns	
t <sub>RC</sub>		Read/Write Cycle Time	104ns		
t <sub>RPC</sub>		RAS Precharge to CAS Active	Ons		
tosh		CAS Setup Time	5ns		
toth		CAS Hold Time	10ns		
twsR		WE Setup Time	0ns		
t <sup>t</sup> 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 HSYNCs, and then going high and staying high for the remainder of the frame. Each cycle lasts a total of 286 HSYNCs. 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 pipielined 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 DCLKs, and then going high and staying high for the remainder of the row. Each cycle lasts a total of 525 SCLKs. 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 DCLKs and the horizontal back porch of 2 DCLKs). 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 SCLKs and the vertical back porch of 2 SCLKs.

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  $(\mathit{UREQ})$ ; the flip-flop is reset when the VRAM controller confirms the completion of the row update  $(\mathit{UACK})$ .

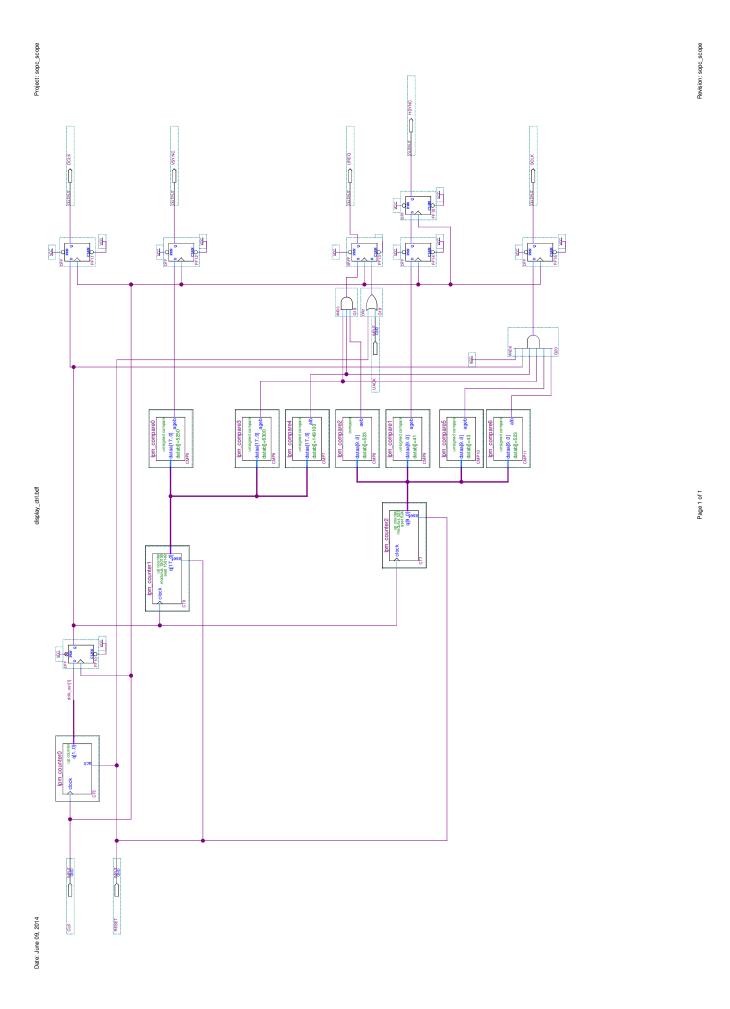
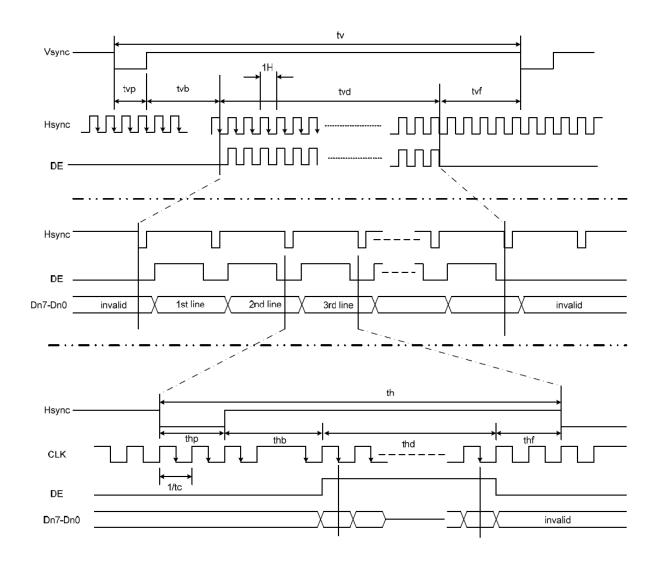


Figure 16: Block diagram of the display controller component within the FPGA. The document is described in Section 2.2.6.



[7]

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

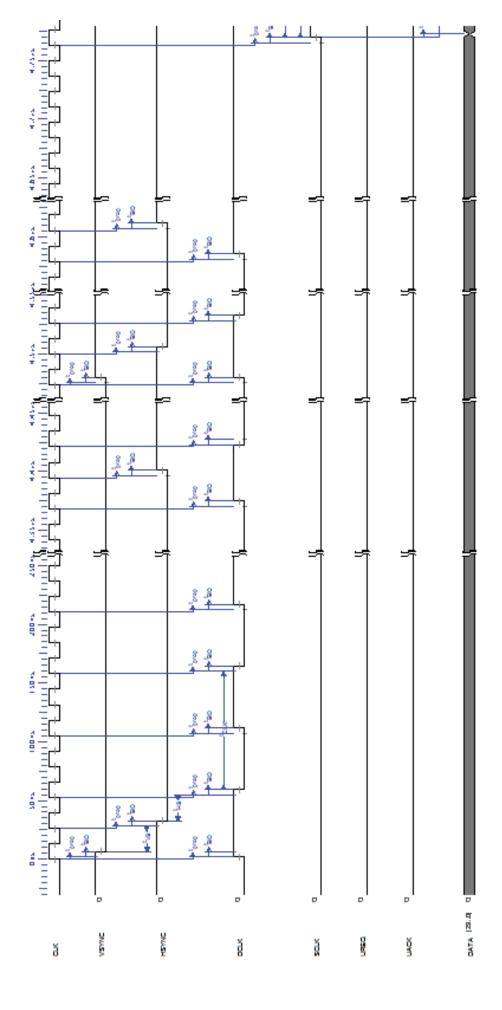
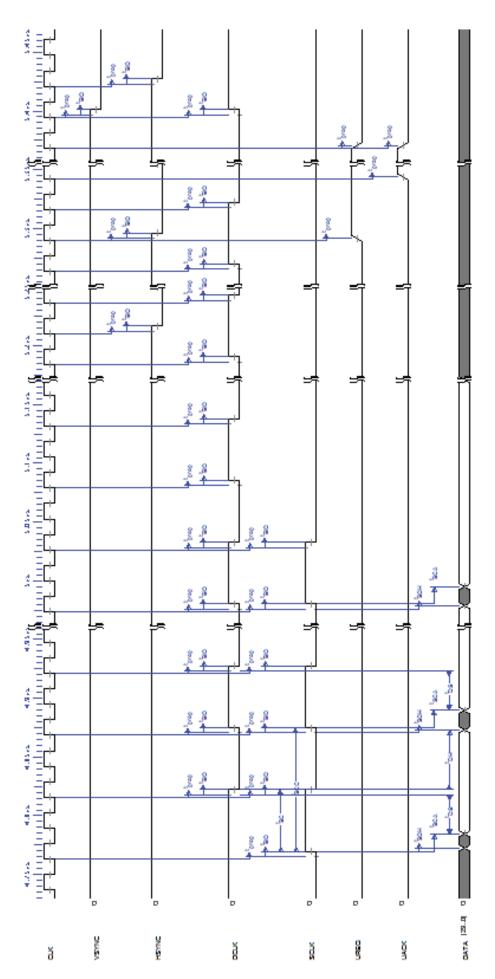


Figure 18: Timing diagram of the display cycle, described in Section 2.2.6.



General Data					
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOTES
t <sub>BIO</sub>		Buffer I/O Delay		4.5ns	
t prop		FPGA Propagation Delay		2ns	
tolk		DCLK Period	66.7ns		
I		HSync Cycle - 2100			
t√P		VSync Pulse Width - 10 H			
tvB		VSync Back Porch - 2 H			
t^D		VSync Display Period - 272 H			
t VF		VSync Front Porch - 2 H			
₽		HSync Pulse Width - 164 CLKS	260ns	4.33us	
t ⊞		HSync Back Porch - 8 CLKS			
무		HSync Display Period - 1920 CLKS			
<b>.</b> ‡		HSync Front Porch - 8 CLKS			
<sup>†</sup> SCA		VRAM Access Time (from SC)		15ns	
ноѕ <sub>1</sub>		VRAM Serial Output Hold Time (rom SC)	3ns		
sat		Data Setup Time	10ns		
но <sub>т</sub>		Data Hold Time	10ns		
tvs		VSync Setup Time	10ns		
s <sub>+</sub> ,		HSync 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...

NOTES		
MAX		
Z	Sns	18ns
DESCRIPTION	Serial Clock Pulse Width	Serial Clock Period
DEFINITION		
SYMBOL	tsc	

### 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-U10that 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  $(B\rightarrow A)$  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  $(B\rightarrow A)$ , 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, 10E and 20E 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 ( $B\rightarrow A$ ), 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  $(A\rightarrow B)$  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  $(B\rightarrow A)$  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.

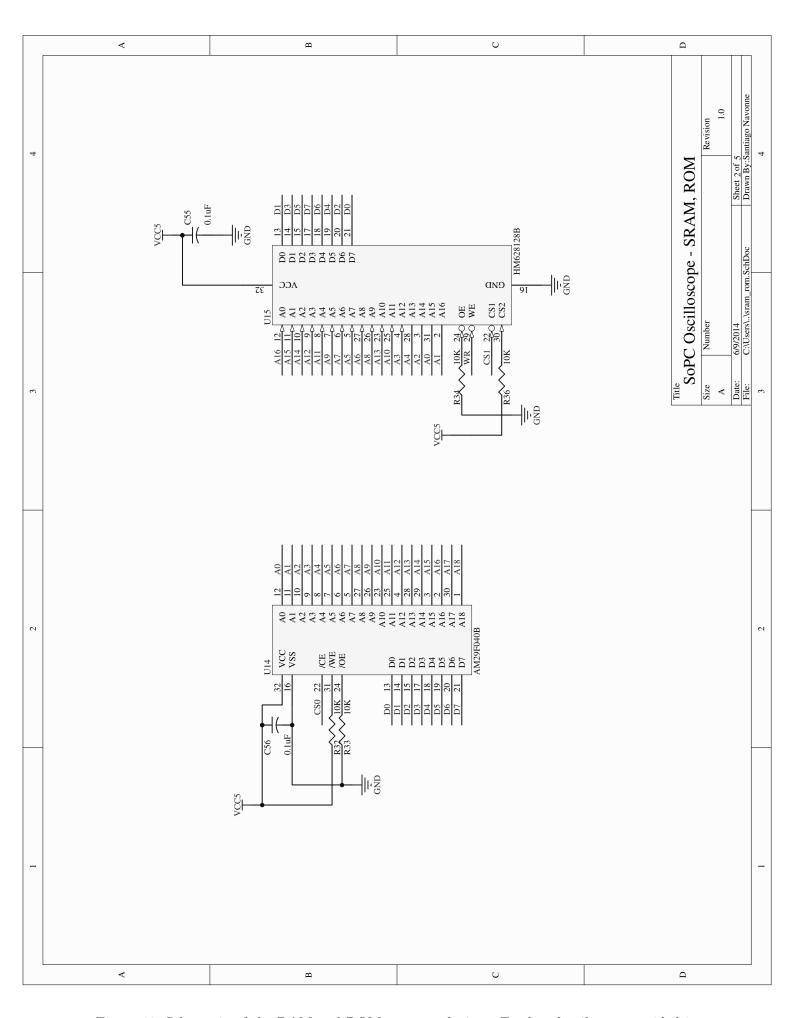


Figure 19: Schematic of the RAM and ROM memory devices. Further details are provided in Sections 2.8.1 and 2.8.2.

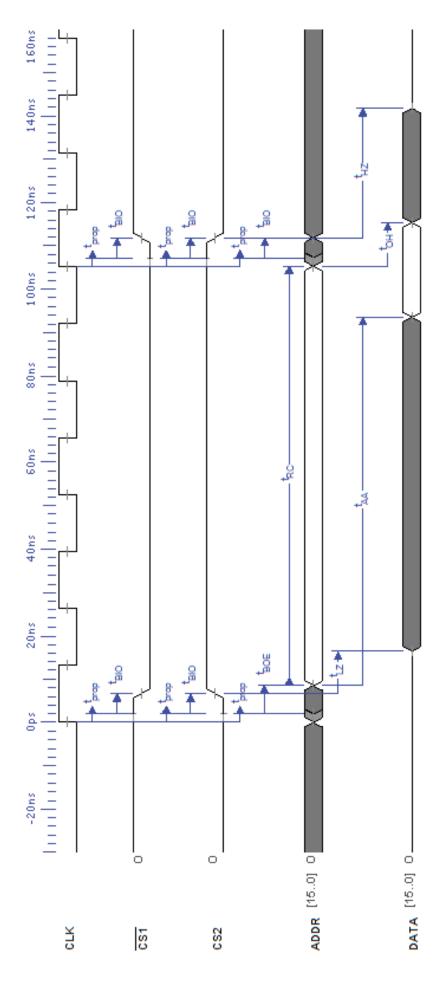


Figure 20: Timing diagram of the read cycle of the RAM device, described in Section 2.8.1.

General Data					
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOTES
t prop		Propagation delay of FPGA	Ons	2ns	
t <sub>BIO</sub>		Buffer input-output delay		4.5ns	
<sup>t</sup> BOE		Buffer output enable		6.5ns	
t <sub>LZ</sub>		Chip selection to output enable		10ns	
taA		Address access time		85ns	
trc		Read cycle time	85ns		
t <sup>†</sup> +z		Chip deselection to output high-Z		30ns	
+O <sub>t</sub>		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.

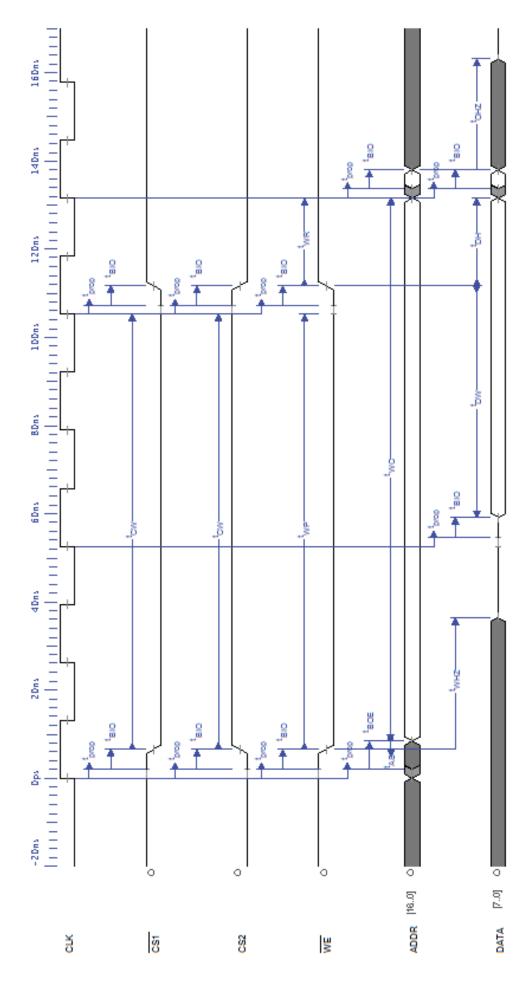


Figure 21: Timing diagram of the write cycle of the RAM device, described in Section 2.8.2.

General Data					
SYMBOL	DEFINITION	DESCRIPTION	NIM	MAX	NOTES
t prop		FPGA Propagation Delay	0	2ns	
t <sub>BIO</sub>		Buffer Input-Output Delay		4.5ns	
<sup>†</sup> BOE		Buffer Output Enable Delay		6.5ns	
tow		Chip selection to end of write (CS1, CS2 hold time)	75ns		
twc		Write cycle time	85ns		
<sup>†</sup> WP		Write pulse width (WE hold time)	55ns		
tas		Address setup time	0ns		
<sup>t</sup> wHZ		Write output to high-Z (don't drive data during this delay)		30ns	
t two		Data to write setup time	30ns		
но <sub>ф</sub>		Data hold from write time	0ns		
twR		Write recovery time	Ons		
zho <sub>t</sub>		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 0x180000-0x1FFFFF, 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.

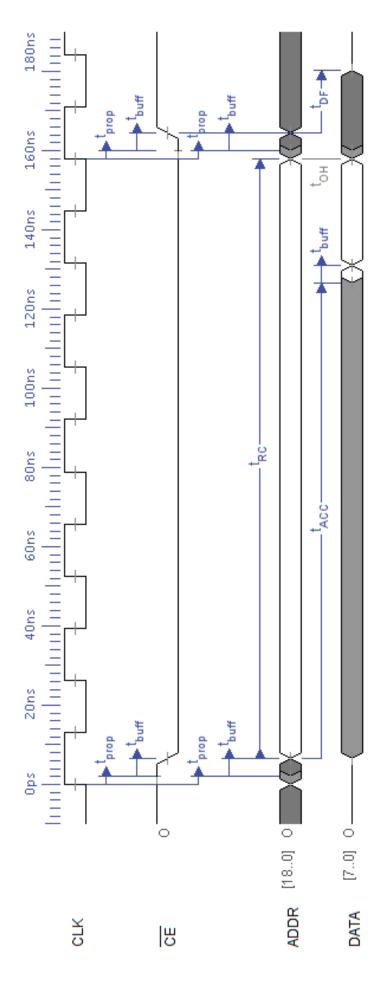


Figure 22: Timing diagram of the read cycle of the ROM device, described in Section 2.8.2.

General Data					
SYMBOL	DEFINITION	DESCRIPTION	NIM	MAX	NOTES
t prop		Propagation delay of FPGA	Ons	2ns	
tbuff		Buffer output delay		4.5ns	
tacc		Address to Data Valid delay		120ns	
toe		Chip Enable to Data Valid delay		120ns	
trc		Address stable hold time	120ns		
<sup>t</sup> DF		Chip enable to output high-Zdelay		16ns	
ф		Outputho <u>ld time from</u> address/CE	Ons	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 0x000000-0xFFFFF, 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.

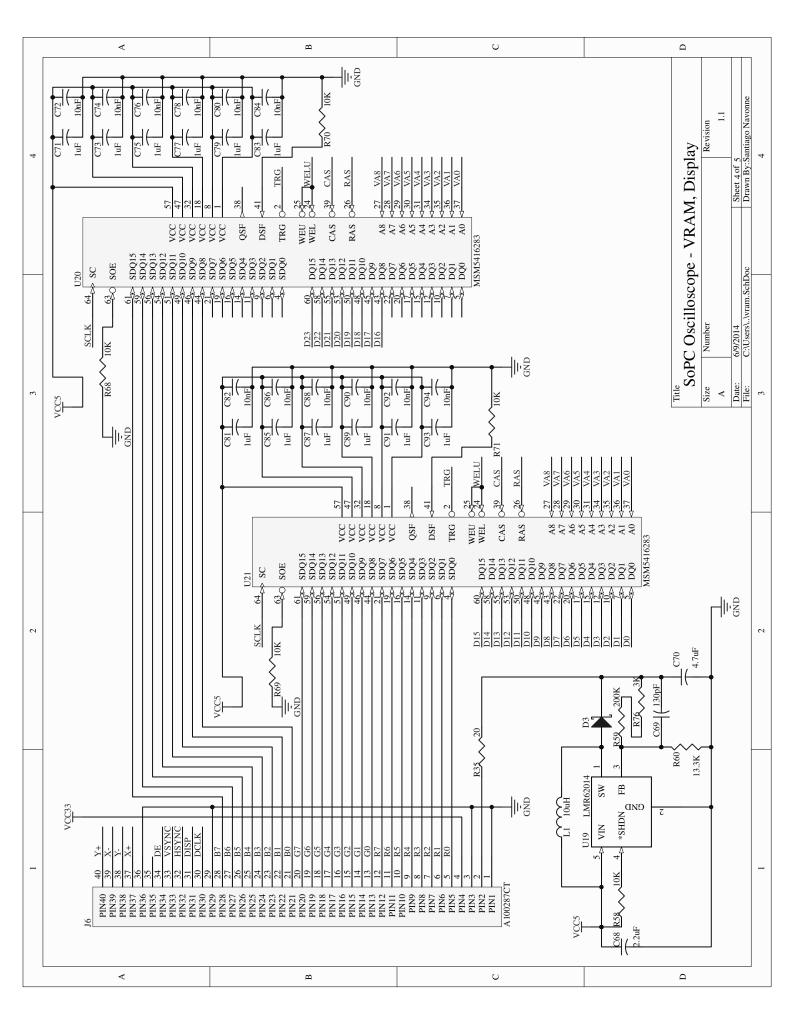


Figure 23: Schematic of the VRAM memory devices and display interface. Also shown is the section of the power supply used to power the display's LEDs. Further details are provided in Section 2.8.3.

### 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 it 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.

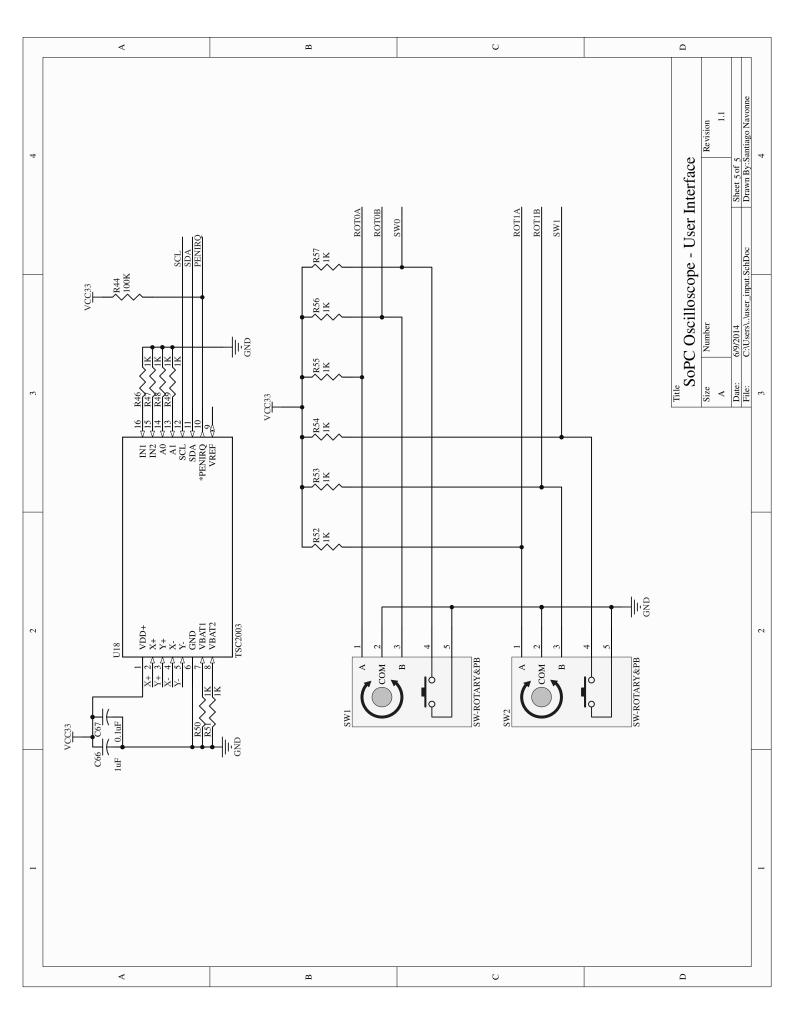


Figure 24: Schematic of the user input section of the system. This includes the rotary encoders, described in Section 2.11, and the touch screen controller, described in Section 2.10.

# 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.

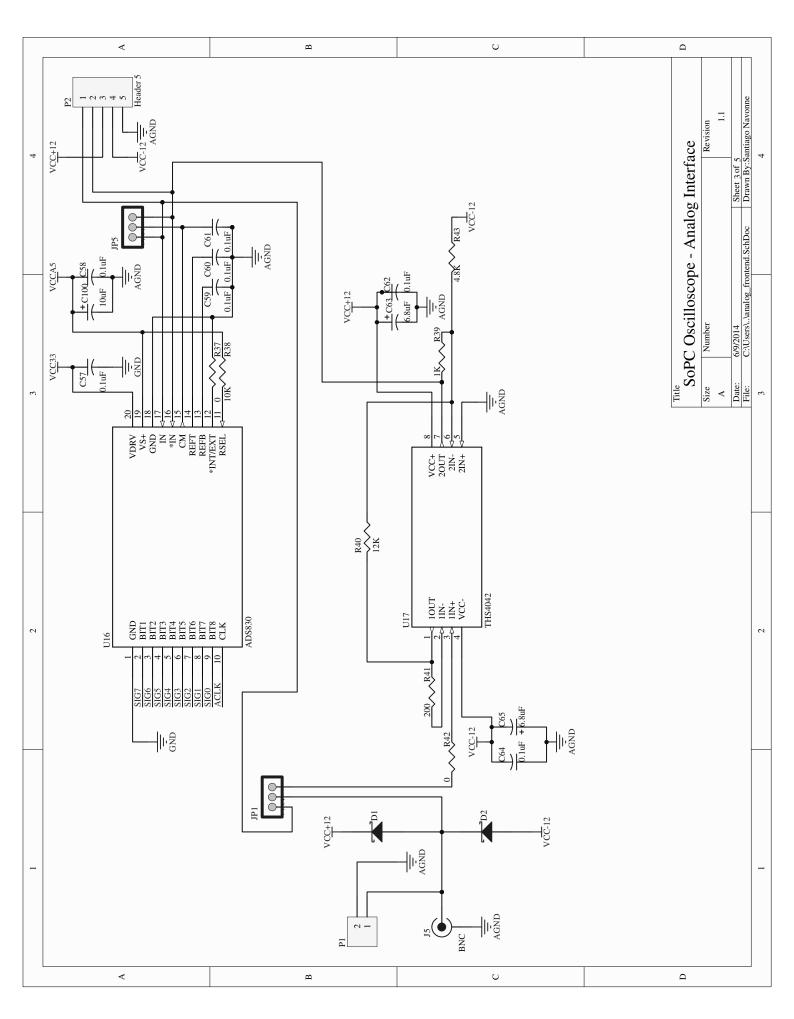


Figure 25: Schematic of the system's analog interface, described in Section 2.12. This includes the analog front-end and the analog-to-digital converter (ADC).

#### 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  $(\pm 1 \text{ V})$ , 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 -12V/4.8 = -2.5V (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.

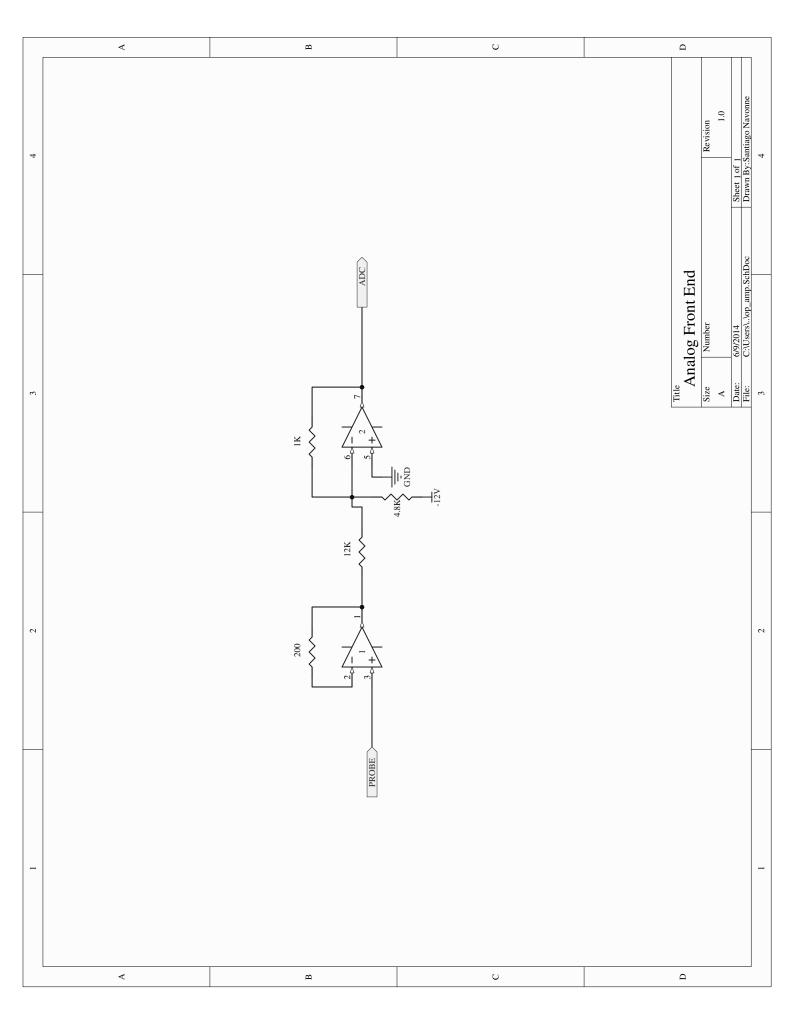


Figure 26: Symbol schematic of the operational amplifier scaling and shifting circuit used in the system's analog front-end, and described in Section 2.12.1.

#### 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.

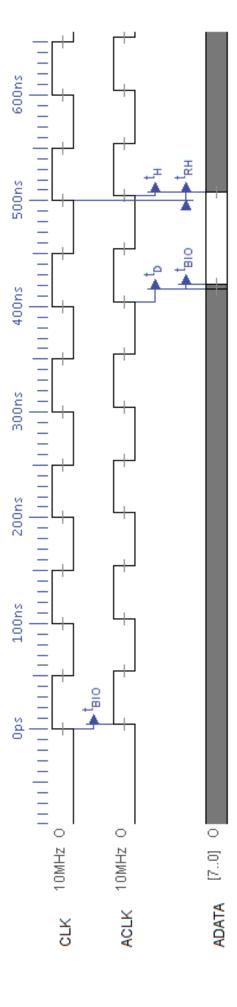


Figure 27: Timing diagram of an ADC sampling clock, described in Section 2.12.2.

General Data					
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOTES
ţ.		New data delay after fourth clock		12ns	
ţ.Ŧ		Data hold after fifth clock 3.9ns	3.9ns		
t <sub>BIO</sub>		Buffer Input/Output delay		4.5ns	
тян		Read cycle hold	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	Changed rotary encoder pull-up resistors to
		Schematic	$1 k\Omega$ .
			Added pull-up resistor on PENIRQ line.
		FPGA	Corrected JTAG connector wiring.
		Schematic	
			Fixed error in routing of A0.
		PCB	Changed voltage regulators' footprints.
		Analog Inter-	Changed value of pull-down resistor at ADC.
		face Schematic	

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 lcdlout.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 constains 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.

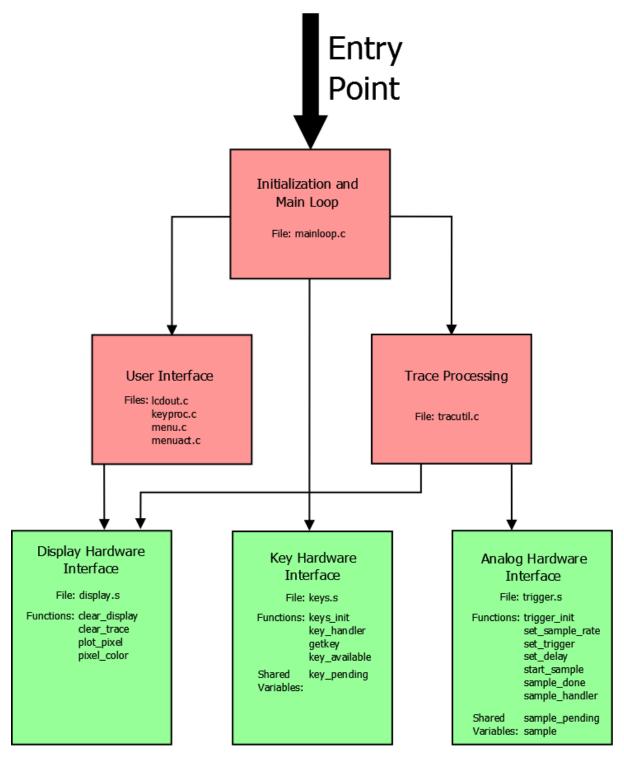


Figure 28: Block diagram of the SoPC Oscilloscope system's software. Blocks in red are part of the EE/CS 52 Oscilloscope library; blocks in green are hardware-specific and implemented for this system. The diagram is described in Section 3.1.

#### 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 *qetkey* 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 the 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 ( $A\theta$ ) 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  $10k\Omega$  to  $1k\Omega$ , 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).

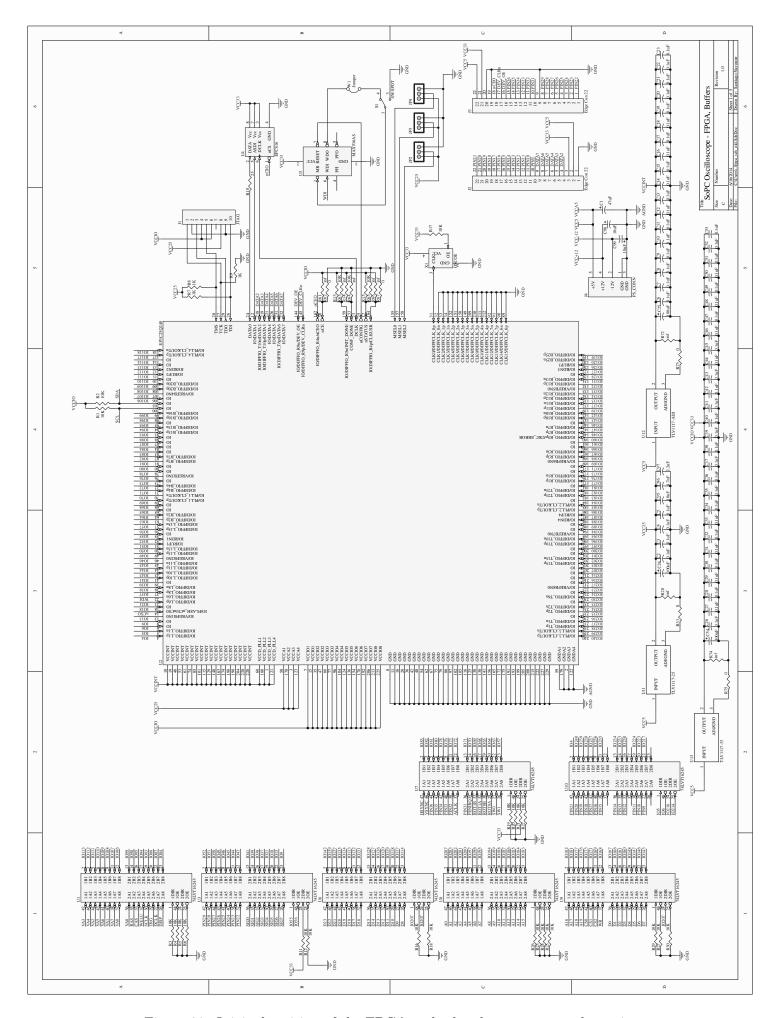


Figure 29: Original revision of the FPGA and related components schematic.

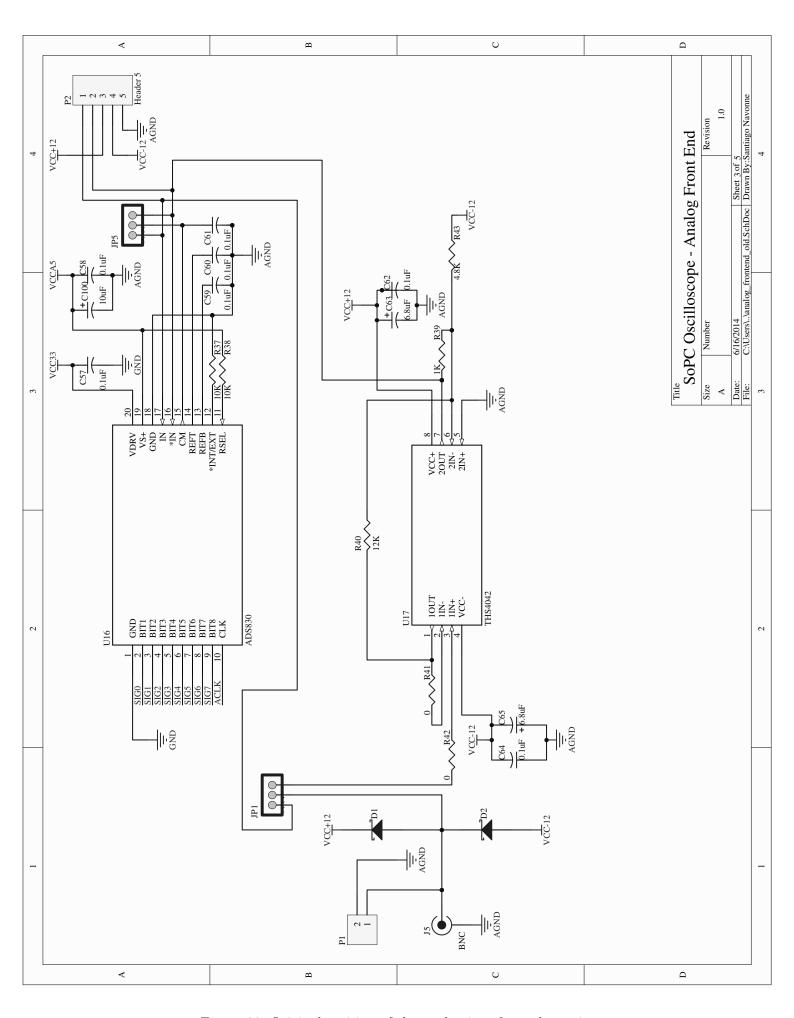


Figure 30: Original revision of the analog interface schematic.

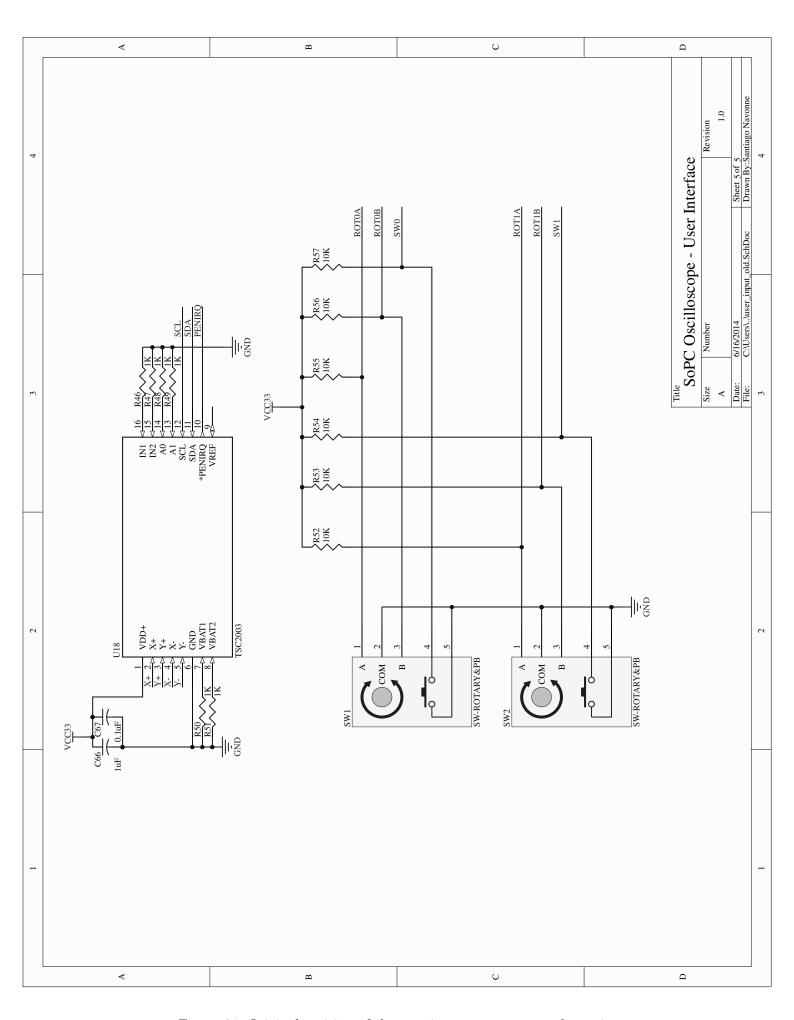


Figure 31: Original revision of the user input components schematic.

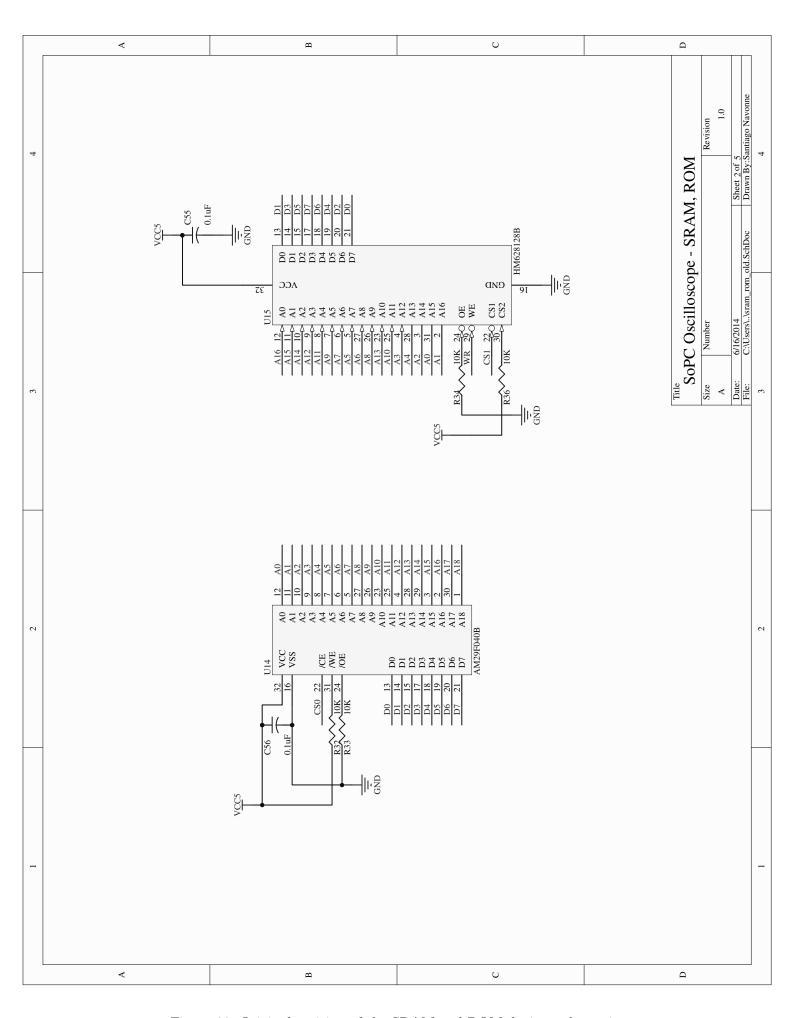


Figure 32: Original revision of the SRAM and ROM devices schematic.

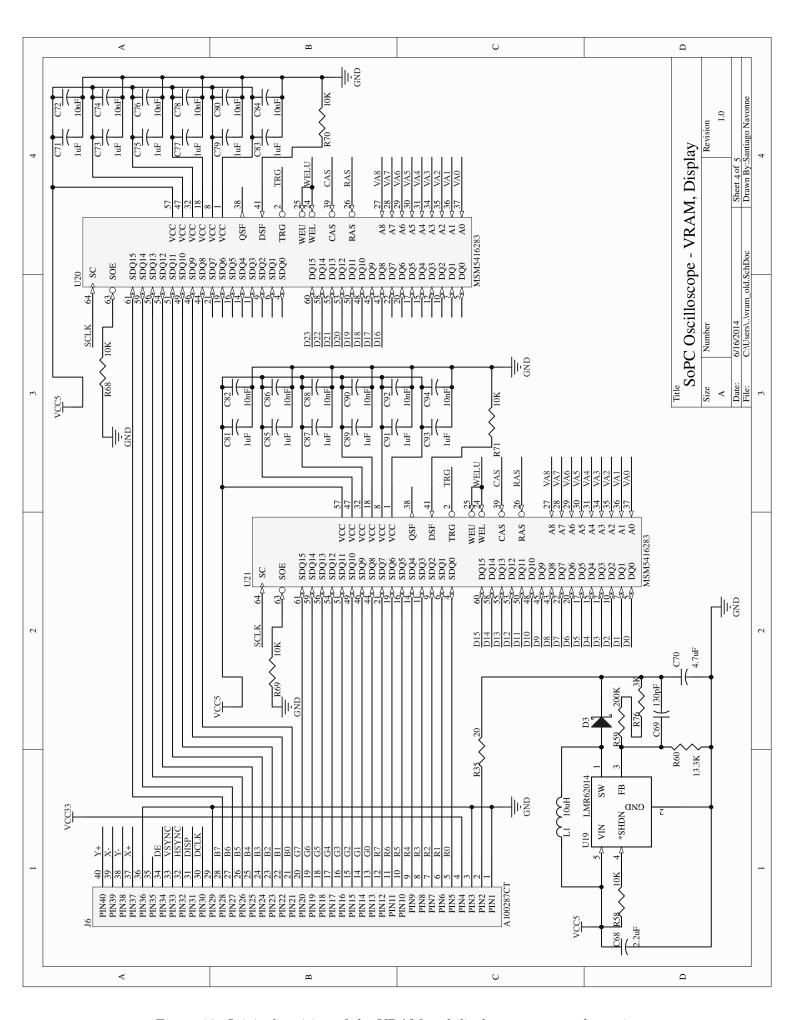


Figure 33: Original revision of the VRAM and display connector schematic.

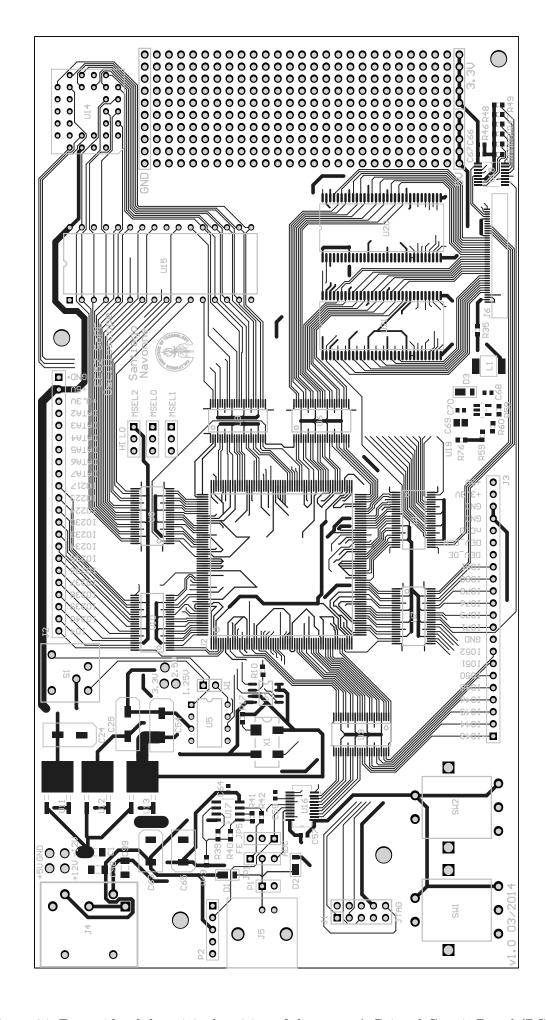


Figure 34: Front side of the original revision of the system's Printed Circuit Board (PCB).

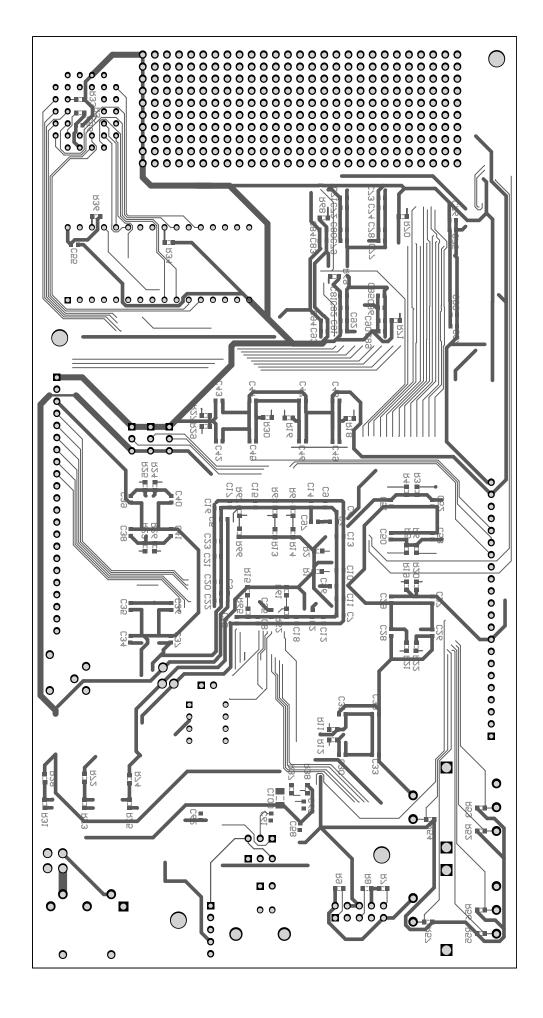


Figure 35: Back side of the original revision of the system's Printed Circuit Board (PCB).

# **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
                               Display Interface Functions
4
5
                                                                                     */
                              Digital Oscilloscope Project
6
   /*
                                        EE/CS 52
7
                                    Santiago Navonne
8
             ***********
9
10
11
      Display interface and control routines for the EE/CS 52 Digital Oscilloscope project. Function definitions are included in this file, and are laid out
12
13
      as follows:
14
       - clear_display: Completely clears the display;
15
16
       - clear_trace: Clears the pixels on the display that are the color of the
17
                       trace;
       - plot_pixel: Changes the color of the pixel at a given location;
18
19
       - pixel_color: Accesses the color of the pixel currently being displayed at
                       a given location.
20
21
22
23
      Revision History:
2.4
         6/3/14 Santiago Navonne Initial revision.
25
26
   #include "general.h"
2.7
28
   #include "system.h"
   #include "interfac.h"
29
   #include "display.h"
30
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
                           pixel to black immediately.
40
41
       Operation:
                           The procedure loops through every pixel in the display-mapped
42
                           region of the VRAM, storing 0 (black; clear pixel) into every
43
44
                           location.
45
                           None.
       Arguments:
46
47
48
       Return Value:
                           None.
49
50
       Local Variables:
                           None.
51
       Shared Variables:
                           None.
52
53
       Global Variables:
                           None.
54
55
56
       Input:
57
                           Clears every pixel on the display (changes color to black).
58
       Output:
59
       Error Handling:
                           None.
60
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
                                    /* clear the whole display */
74
   clear_display:
               r8, %hi(VRAM BASE) /* start at base of VRAM */
75
       MOVHI
```

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

150

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

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

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

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

```
***************
2
   /*
3
                                       INTERFAC.H
4
5
                                  Interface Definitions
                                      Include File
6
   /*
                              Digital Oscilloscope Project
   /*
7
                                        EE/CS 52
8
           ****************
9
10
11
      This file contains the constants for interfacing between the C code and
12
      the assembly code/hardware for the Digital Oscilloscope project.
13
14
15
16
      Revision History:
17
         3/8/94 Glen George
                                      Initial revision.
         3/13/94 Glen George
                                      Updated comments.
18
19
         3/17/97 Glen George
                                      Added constant MAX_SAMPLE_SIZE and removed
                                      KEY UNUSED.
20
         5/14/14 Santiago Navonne Changed keypad codes.
6/01/14 Santiago Navonne Changed scope and sampling parameters.
21
22
         6/03/14 Santiago Navonne Changed and added display parameters.
23
2.4
   * /
25
26
2.7
28
   #ifndef
              INTERFAC H
       #define __INTERFAC_H_
29
30
31
   /* library include files */
32
33
     /* none */
34
   /* local include files */
35
36
     /* none */
37
38
39
40
   /* constants */
41
42
   /* keypad constants */
43
   #define KEY_MENU
                                /* <Menu>
44
   #define KEY UP
                                /* <Up>
45
   #define KEY_DOWN
                            3
                                /* <Down>
46
                                 /* <Left>
47
   #define
            KEY LEFT
                            4
   #define KEY RIGHT
                                /* <Right>
48
                                 /* illegal key */
   #define KEY_ILLEGAL
49
50
   /* display constants */
51
   #define SIZE_X
                            480
                              80 /* size in the x dimension */
272 /* size in the y dimension */
52
            SIZE_Y
53
   #define
   #define PIXEL CLEAR
                            0x00000000 /* pixel off is black */
54
                            0x001B3830 /* lines are gray */
55
   #define PIXEL_LINE
   #define PIXEL_TEXT_H
#define PIXEL TRACE
                            0x00FFFFFF /* highlighted text is white */
56
                            0x0000A000 /* trace is green */
57
                            0x001B3830 /* normal text is gray */
   #define PIXEL_TEXT_N
58
   #define
            PIXEL CURSOR
                            0x00A00000 /* cursor is red */
59
   #define NO_TRACE
                            0xFFFFFFFF /* no trace found */
60
61
62
   /* scope parameters */
   #define MIN DELAY
                                       /* minimum trigger delay */
63
                            0xFFFFFFE/* maximum trigger delay */
64
   #define MAX DELAY
                                    /* minimum trigger level (in mV) */
/* maximum trigger level (in mV) */
                            -12000
65
   #define
            MIN_LEVEL
   #define MAX_LEVEL
                            12000
66
67
   /* sampling parameters */
68
   #define MAX_SAMPLE_SIZE
                                512
                                       /* maximum size of a sample (in samples) */
69
70
71
72
   #endif
73
```

```
******************
2
3
   /*
                                        KEYPROC
   /*
                                                                                    */
                               Key Processing Functions
4
5
   /*
                             Digital Oscilloscope Project
6
   /*
                                        EE/CS 52
7
      *******************
8
9
10
      This file contains the key processing functions for the Digital
11
      Oscilloscope project. These functions are called by the main loop of the system. The functions included are:
12
13
         menu down - process the <Down> key while in a menu
14
                    - process the <Menu> key
- process the <Left> key while in a menu
15
         menu_key
16
         menu left
         menu_right - process the <Right> key while in a menu
17
         menu_up - process the <Up> key while in a menu no_action - nothing to do
18
19
20
      The local functions included are:
21
22
         none
23
24
      The locally global variable definitions included are:
25
26
2.7
28
      Revision History
         3/8/94 Glen George
                                      Initial revision.
29
30
         3/13/94 Glen George
                                      Updated comments.
31
32
33
34
   /* library include files */
35
36
     /* none */
37
   /* local include files */
38
39
   #include "scopedef.h"
   #include
             "keyproc.h"
40
             "menu.h"
   #include
41
42
43
44
45
46
47
      no_action
48
                         This function handles a key when there is nothing to be
49
      Description:
50
                         done. It just returns.
51
                         cur_state (enum status) - the current system state.
      Arguments:
52
53
      Return Value:
                         (enum status) - the new system state (same as current
54
                 state).
55
56
                         None.
      Input:
      Output:
                         None.
57
58
      Error Handling:
59
60
61
      Algorithms:
                         None.
      Data Structures: None.
62
63
64
      Global Variables: None.
65
      Author:
                         Glen George
66
67
      Last Modified:
                         Mar. 8, 1994
68
   * /
69
70
71
   enum status no action(enum status cur state)
72
   {
73
       /* variables */
         /* none */
74
75
```

```
76
77
        /* return the current state */
78
        return cur_state;
79
80
81
    }
82
83
84
85
86
       menu_key
87
88
       Description:
                           This function handles the <Menu> key. If the passed
 89
                           state is MENU_ON, the menu is turned off. If the passed
90
91
                  state is MENU_OFF, the menu is turned on. The returned
                  state is the "opposite" of the passed state.
92
93
94
       Arguments:
                           cur_state (enum status) - the current system state.
                           (enum status) - the new system state ("opposite" of the
95
       Return Value:
                  as current state).
96
97
98
       Input:
99
       Output:
                           The menu is either turned on or off.
100
       Error Handling:
                           None.
101
102
103
       Algorithms:
       Data Structures:
                          None.
104
105
       Global Variables: None.
106
107
108
       Author:
                           Glen George
109
       Last Modified:
                          Mar. 8, 1994
110
111
112
    enum status menu_key(enum status cur_state)
113
114
        /* variables */
115
116
          /* none */
117
118
119
120
        /* check if need to turn the menu on or off */
        if (cur_state == MENU_ON)
121
122
             /* currently the menu is on, turn it off */
123
        clear menu();
124
             /* currently the menu is off, turn it on */
125
        display_menu();
126
127
128
        /* all done, return the "opposite" of the current state */
129
130
        if (cur_state == MENU_ON)
131
             /* state was MENU ON, change it to MENU OFF */
            return MENU OFF;
132
133
        else
             /* state was MENU OFF, change it to MENU ON */
134
            return MENU_ON;
135
136
137
    }
138
139
140
141
142
       menu up
143
144
145
                           This function handles the <Up> key when in a menu. It
       Description:
                           goes to the previous menu entry and leaves the system
146
147
                  state unchanged.
148
                          cur_state (enum status) - the current system state.
       Arguments:
149
150
       Return Value:
                           (enum status) - the new system state (same as current
```

```
151
                  state).
152
       Input:
153
                           None.
                           The menu display is updated.
       Output:
154
155
       Error Handling:
                           None.
156
157
158
       Algorithms:
                           None.
       Data Structures:
                           None.
159
160
       Global Variables: None.
161
162
       Author:
                           Glen George
163
       Last Modified:
                           Mar. 8, 1994
164
165
166
167
    enum status menu_up(enum status cur_state)
168
169
         /* variables */
170
          /* none */
171
172
173
174
175
         /* go to the previous menu entry */
        previous_entry();
176
177
178
        /* return the current state */
179
180
        return cur_state;
181
182
    }
183
184
185
186
187
       menu_down
188
189
       Description:
                           This function handles the <Down> key when in a menu. It
190
                           goes to the next menu entry and leaves the system state
191
192
193
                           cur_state (enum status) - the current system state.
194
       Arguments:
195
       Return Value:
                            (enum status) - the new system state (same as current
                  state).
196
197
198
       Input:
       Output:
                           The menu display is updated.
199
200
201
       Error Handling:
                           None.
202
203
       Algorithms:
                           None.
       Data Structures:
                           None.
204
205
206
       Global Variables: None.
207
       Author:
                           Glen George
208
209
       Last Modified:
                           Mar. 8, 1994
210
211
212
    enum status menu_down(enum status cur_state)
213
214
         /* variables */
215
          /* none */
216
217
218
219
220
         /* go to the next menu entry */
221
        next_entry();
222
223
        /* return the current state */
224
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.
                           invokes the left function for the current menu entry and
236
                  leaves the system state unchanged.
237
238
239
       Arguments:
                           cur state (enum status) - the current system state.
       Return Value:
                           (enum status) - the new system state (same as current
240
241
                  state).
242
       Input:
                           None.
243
244
       Output:
                           The menu display may be updated.
245
       Error Handling:
                           None.
246
247
248
       Algorithms:
                           None.
249
       Data Structures:
                           None.
250
       Global Variables: None.
251
252
253
                           Glen George
       Last Modified:
                           Mar. 8, 1994
254
255
256
257
258
    enum status menu_left(enum status cur_state)
259
         /* variables */
260
261
           /* none */
262
263
264
         /* invoke the <Left> key function for the current menu entry */
265
266
        menu_entry_left();
267
268
         /* return the current state */
269
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.
281
                           invokes the right function for the current menu entry and
                  leaves the system state unchanged.
282
283
       Arguments:
                           cur state (enum status) - the current system state.
284
       Return Value:
                           (enum status) - the new system state (same as current
285
286
                  state).
287
       Input:
                           None.
288
289
       Output:
                           The menu display may be updated.
290
       Error Handling:
                           None.
291
292
       Algorithms:
                           None.
293
       Data Structures:
294
                           None.
295
296
       Global Variables: None.
297
298
                           Glen George
       Last Modified:
                           Mar. 8, 1994
299
300
```

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

```
2
3
   /*
                                       KEYPROC.H
                                                                                     */
                                Key Processing Functions
4
5
   /*
                                      Include File
6
   /*
                              Digital Oscilloscope Project
7
   /*
                                        EE/CS 52
   /*
8
            *************
9
10
11
      This file contains the constants and function prototypes for the key
12
      processing functions (defined in keyproc.c) for the Digital Oscilloscope
13
14
      project.
15
16
      Revision History:
17
         3/8/94
                  Glen George
                                      Initial revision.
18
         3/13/94 Glen George
19
                                      Updated comments.
20
21
22
23
24
   #ifndef
              _KEYPROC_H
       #define __KEYPROC_H_
25
26
2.7
   /* library include files */
  /* none */
28
29
30
31
   /* local include files */
   #include "scopedef.h"
32
33
34
35
36
   /* constants */
37
       /* none */
38
39
40
41
42
   /* structures, unions, and typedefs */
43
       /* none */
44
45
46
47
48
   /* function declarations */
49
50
   enum status no_action(enum status);
                                               /* nothing to do */
51
52
53
   enum status menu_key(enum status);
                                           /* process the <Menu> key */
54
                                           /* < Up > key in a menu */
55
   enum status menu_up(enum status);
                                               /* <Down> key in a menu */
/* <Left> key in a menu */
56
   enum status menu down(enum status);
   enum status menu left(enum status);
57
                                                /* <Right> key in a menu */
   enum status menu_right(enum status);
58
59
60
   #endif
61
62
```

```
**************
2
3
   /*
                                         KEYS.S
                                                                                     */
4
5
                                      Key handlers
   /*
                              Digital Oscilloscope Project
6
   /*
                                        EE/CS 52
   /*
7
                                    Santiago Navonne
8
   /*
             ************
9
10
11
      Key and rotary encoder control routines for the EE/CS 52 Digital Oscilloscope
12
      project. Function definitions are included in this file, and are laid out
13
      as follows:
14
       - keys_init: Initializes the key handler's shared variables, and enables interrupts from the required sources, effectively preparing
15
16
                     the user input section for use;
17
       - keys_handler: Handles key press (and rotary encoder turn) interrupts;
18
19
         getkey: Returns the currently pending user action, blocking if none is
20
                  available.
       - key available: Checks whether a user action is currently pending.
21
22
23
24
      Revision History:
         5/7/14 Santiago Navonne
                                     Initial revision.
25
         5/14/14 Santiago Navonne
                                     Added additional documentation.
26
         6/7/14 Santiago Navonne Changed up/down rotation direction.
2.7
28
29
30
   /* Includes */
   #include "general.h"
                          /* General constants */
31
   #include "system.h"
                          /* Base addresses */
32
   #include "interfac.h" /* Software interface definitions */
33
   #include "keys.h"
                           /* Local constants */
34
35
36
37
   /* Variables */
       .section .data /* No alignment necessary: variables are bytes */
38
39
   curr key: .byte 0
                        /* Current pending key; 0 if no key available */
40
        .section .text /* Code starts here */
41
42
43
       keys_init
44
45
                            This procedure initializes the internal state of the key/
       Description:
46
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
       Operation:
                            This procedure initializes any shared variables to their
51
                            default states:
52
53
                             - curr_key: value of the currently pending key (default: 0).
                            Additionally, the function registers the key press handler
54
55
                            as the default interrupt handler for key presses using the HAL
                           API alt_ic_isr_register, and finally unmasks all interrupts by writing to the corresponding PIO register.
56
57
58
       Arguments:
59
                           None.
60
61
       Return Value:
                           None.
62
       Local Variables:
                           None.
63
64
65
       Shared Variables:
                           - curr key (write only).
66
67
       Global Variables:
                           None.
68
       Input:
69
                            None.
70
71
       Output:
                           None.
72
73
       Error Handling:
                           None.
74
75
       Limitations:
                           None.
```

```
76
77
        Algorithms:
                             None.
        Data Structures:
78
                             None.
79
        Registers Changed: r4, r5, r6, r7, r8, r9.
80
81
82
        Revision History:
83
            5/7/14
                       Santiago Navonne
                                              Initial revision.
                                              Added additional documentation.
            5/14/14
                       Santiago Navonne
84
85
86
        .global keys init
87
88
    keys init:
        ADDI
                 sp, sp, NEG WORD SIZE /* push return address */
89
        STW
90
                 ra, (sp)
91
92
        MOVIA
                 r9, curr_key
                                          /* no key (r0) available at start */
        STB
                 r0, (r9)
                                          /* so store it into variable curr key */
93
94
                                          /* write to the PIO registers */
95
        MOVHI
                 r8, %hi(PIO 0 BASE)
        ORI
                 r8, r8, %lo(PIO_0_BASE)
96
97
        MOVI
                 r9, ENABLE_ALL
                                              the ENABLE ALL value */
        STBIO
                 r9, EDGE CAP OF(r8)
                                          /* sending general EOI to clear ints */
98
99
        MOV
                                          /* argument ic id is ignored */
100
                 r4, r0
                 r5, PIO_0 IRQ
        MOVI
                                          /* second arg is IRQ num */
101
                                          /* third arg is int handler */
        MOVTA
102
                 r6, keys_handler
103
        MOV
                 r7, r0
                                          /* fourth arg is data struct (null) */
                 sp, sp, NEG_WORD SIZE
        ADDI
                                         /* fifth arg goes on stack */
104
                                          /*
105
        STW
                 r0, (sp)
                                             and is ignored (so 0) */
                                          /* finally, call setup function */
106
        CALL
                 alt_ic_isr_register
                                          /* clean up stack after call */
107
        ADDI
                 sp, sp, WORD_SIZE
108
109
        T_1DW
                                          /* pop return address */
                 ra, (sp)
                 sp, sp, WORD_SIZE
        ADDI
110
111
112
        STBIO
                 r9, INTMASK OF(r8)
                                          /* enable (unmask) interrupts */
113
        RET
                                          /* and finally return */
114
115
116
117
118
119
        keys_handler
120
        Description:
                             This procedure handles hardware interrupts generated by
121
122
                             key presses and rotary encoder steps. Every time one of
                             these fires, the shared variable containing the currently
123
124
                             pending key is updated to indicate a key press. Note that
125
                             previously pending key presses are overwritten by this
126
                             function.
                             The function is designed to support only one key press
127
128
                             at a time; its behavior in the event of simultaneous key
                             presses is undefined.
129
130
                             When called, the function first reads the edge capture register of the user input PIO interface to figure out
131
        Operation:
132
133
                             which interrupt fired. It compares the read value to all
                             the known constants, translating it into a key ID. Unknown
134
                             values, which are caused by simultaneous key presses,
135
136
                             are handled in the else case.
137
                             After the key press is decoded, the identification code is
                             saved to the shared variable curr_key.
138
139
                             Note that the procedure uses multiple comparisons and not
140
                             a jump table in order to save space; furthermore, the
                             interrupt register value is not simply used as a key
141
142
                             identifier to prevent simultaneous key presses from
                             breaking the system.
143
144
        Arguments:
145
                             None.
146
147
        Return Value:
                             None.
148
        Local Variables:
149
                             None.
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
        Output:
157
                             None.
158
        Error Handling:
                             If multiple keys are pressed at once, the function's
159
                             behavior is undefined.
160
161
                             Only one simultaneous key press is accepted. Any previously recognized but not yet polled key presses
        Limitations:
162
163
                             are lost (overwritten) when a new event is received.
164
165
166
        Algorithms:
                             None.
167
        Data Structures:
                             None.
168
169
        Registers Changed: et.
170
        Revision History:
171
172
            5/7/14
                       Santiago Navonne
                                              Initial revision.
173
            5/14/14
                       Santiago Navonne
                                              Added additional documentation.
174
175
        .qlobal keys handler
176
177
    keys handler:
                 sp, sp, NEG_WORD_SIZE
178
        ADDI
                                          /* save r8 */
        STW
179
                 r8, (sp)
180
                 et, %hi(PIO_0_BASE) /* fetch PIO edge capture register */
181
        MOVHI
                 et, et, %lo(PIO_0_BASE)
182
        ORT
                 r8, EDGE CAP OF(et)
183
        LDBIO
184
        STBIO
                 r8, EDGE CAP OF(et) /* and write back to send EOI */
185
186
                                        /* figure out what interrupt fired */
                 et, PUSH1 MASK
187
        MOVI
                                        /* check if it was pushbutton 1 */
                 r8, et, keys_handler_push1
188
        BEO
                 et, PUSH2 MASK
        IVOM
                                       /* check if it was pushbutton 2 */
189
                 r8, et, keys_handler_push2
190
        BEO
                                        /* check if it was rotary enc 1 right */
191
        MOVT
                 et, ROT1R MASK
192
        BEO
                 r8, et, keys_handler_rot1r
                 et, ROT1L MASK
                                        /* check if it was rotary enc 1 left */
        MOVI
193
194
        BEO
                 r8, et, keys_handler_rot11
                 et, ROT2R MASK
195
        MOVI
                                       /* check if it was rotary enc 2 right */
                 r8, et, keys_handler_rot2r
        BEO
196
197
        JMPI
                 keys_handler_rot21
                                        /* else it must be rotary enc 2 left */
198
                                         /* handle pushbutton 1 ints */
    keys_handler_push1:
199
200
        MOVI
                 et, KEY MENU
                                         /* translates into menu key */
        JMPI
                 keys_handler_done
201
202
203
    keys_handler_push2:
                                         /* handle pushbutton 2 ints */
                 et, KEY MENU
        MOVI
                                         /* translates into menu key */
204
205
        JMPT
                 keys handler done
206
    keys_handler rot1r:
                                         /* handle rotary enc 1 right ints */
207
                 et, KEY_DOWN
                                         /* translates into down key */
208
        MOVI
                 keys handler done
209
        JMPI
210
211
    keys handler rot11:
                                         /* handle rotary enc 1 left ints */
212
        MOVT
                 et, KEY UP
                                         /* translates into up key */
                 keys_handler_done
213
        JMPI
214
215
    keys handler rot2r:
                                         /* handle rotary enc 2 right ints */
                 et, KEY_RIGHT
                                         /* translates into right key */
216
        MOVT
217
        JMPI
                 keys_handler_done
218
                                         /* handle rotary enc 2 left ints */
219
    keys_handler_rot21:
                 et, KEY_LEFT
                                          /* translates into left key */
220
        MOVI
221
        JMPT
                 keys handler done
222
223
    keys handler done:
                                         /* handling completed */
                                         /* save to curr_key */
        MOVIA
                 r8, curr_key
224
225
        STB
                 et, (r8)
                                         /* the processed key */
```

```
226
227
        LDW
                 r8, (sp)
                                         /* restore r8 */
        ADDI
                 sp, sp, WORD_SIZE
228
                                         /* all done */
        RET
229
230
231
232
233
        getkey
234
235
        Description:
                             This procedure returns the identifier of the last pressed,
236
                             unpolled key, as described in interfac.h.
237
                             If no key press is pending, the function blocks.
238
                             (To ensure non-blocking behavior, getkey calls should be
239
                             preceded by key_available calls.)
240
241
242
        Operation:
                             The function first fetches the value stored in curr key and
                             compares it to 0, which would indicate that there isn't
243
244
                             actually any pending key press. In no key press is pending,
                             the function keeps fetching the value until it is not 0.
245
                             When the value is not 0, the function clears the value of
246
247
                             curr_key (to delete the now reported press) and returns
248
                             the retrieved value.
249
        Arguments:
                             None.
250
251
                             key (r2) - ID code of the pending key, as defined in
252
        Return Value:
253
                                         interfac.h.
254
255
        Local Variables:
                             None.
256
        Shared Variables:
                             - curr key: currently pending key press code (read/write).
257
258
259
        Global Variables:
                             None.
260
261
        Input:
                             None.
262
        Output:
                             None.
263
264
                             If no key is available, the funciton blocks until a key
265
        Error Handling:
266
                             is pressed.
267
        Limitations:
                             None.
268
269
270
        Algorithms:
                             None.
        Data Structures:
                             None.
271
272
273
        Registers Changed: r2, r8.
274
275
        Revision History:
276
             5/7/14
                       Santiago Navonne
                                              Initial revision.
                                              Added additional documentation.
             5/14/14
                       Santiago Navonne
277
278
279
280
        .global getkey
281
    getkey:
        MOVIA
                                     /* return current pending key */
                 r8, curr_key
282
                 r2, (r8)
r0, r2, getkey
283
        \mathtt{LDB}
                                     /* if there is no key (curr key == r0), block */
284
        BEQ
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
                             This procedure checks whether a key has been pressed and
294
        Description:
                             is available for polling. The function returns true
295
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
                             blocking.
299
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
       Return Value:
                            key available (r2) - true (non-zero) if a key press is
307
308
                                                  available, false (zero) otherwise.
309
       Local Variables:
310
                            None.
311
        Shared Variables: - curr key: currently pending key press code (read only).
312
313
314
       Global Variables: None.
315
       Input:
                            Key presses and rotary encoder turns from the user interface.
316
317
       Output:
                            None.
318
319
       Error Handling:
                            None.
320
321
       Limitations:
322
                            None.
323
324
       Algorithms:
                            None.
325
        Data Structures:
                           None.
326
       Registers Changed: r2, r8.
327
328
       Revision History:
329
330
            5/7/14
                      Santiago Navonne
                                           Initial revision.
331
     *
            5/14/14
                      Santiago Navonne
                                            Added additional documentation.
332
333
334
        .globl key_available
    key_available:
335
                                    /* return current pending key */
336
       MOVIA
               r8, curr key
        LDB
                r2, (r8)
                                    /* will be zero (FALSE) if no key is pending */
337
338
339
        RET
                                    /* return with boolean in r2 */
340
```

341 342

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

```
****************
1
   /*
2
3
  /*
                                        LCDOUT
  /*
                                 LCD Output Functions
                                                                                   */
4
   /*
                                                                                  */
5
                             Digital Oscilloscope Project
6
   /*
                                       EE/CS 52
7
       ******************
8
9
10
      This file contains the functions for doing output to the LCD screen for the
11
      Digital Oscilloscope project. The functions included are: clear_region - clear a region of the display
12
13
         plot char
                      - output a character
14
         plot_hline
                       - draw a horizontal line
15
16
         plot string

    output a string

         plot_vline
                       - draw a vertical line
17
         plot_cursor - plot the cursor
18
19
20
      The local functions included are:
         none
21
22
23
      The locally global variable definitions included are:
24
         none
25
26
      Revision History
2.7
28
         3/8/94
                  Glen George
                                     Initial revision.
         3/13/94
                  Glen George
                                     Updated comments.
29
30
         3/13/94
                  Glen George
                                     Simplified code in plot_string function.
31
         3/17/97
                  Glen George
                                     Updated comments.
         3/17/97
                                     Change plot_char() and plot_string() to use
                  Glen George
32
33
                         enum char_style instead of an int value.
         5/27/98
                  Glen George
                                     Change plot char() to explicitly declare the
34
                         size of the external array to avoid linker
35
36
                         errors.
37
         6/3/14
                  Santiago Navonne Changed UI display colors, added support for
                         highlighted characters.
38
39
   */
40
41
42
   /* library include files */
  /* none */
43
44
45
   /* local include files */
46
47
   #include
             "interfac.h"
             "scopedef.h"
48
   #include
             "lcdout.h"
   #include
49
50
51
   extern int pixel_color(int, int);
52
53
54
55
56
      clear region
57
58
      Description:
                         This function clears the passed region of the display.
59
                         The region is described by its upper left corner pixel
60
61
                         coordinate and the size (in pixels) in each dimension.
62
      Arguments:
                                      - x coordinate of upper left corner of the
63
                         x_ul (int)
64
                        region to be cleared.
                y_ul (int) - y coordinate of upper left corner of the
65
                        region to be cleared.
66
67
                 x_size (int) - horizontal size of the region.
                y_size (int) - vertical size of the region.
68
      Return Value:
69
                         None.
70
71
      Input:
                         None.
72
      Output:
                         A portion of the screen is cleared (set to PIXEL CLEAR).
73
                        No error checking is done on the coordinates.
74
      Error Handling:
75
```

```
76
        Algorithms:
                             None.
 77
        Data Structures:
                            None.
 78
        Global Variables: None.
79
 80
                             Glen George
 81
        Last Modified:
                             June 03, 2014
 82
 83
 84
 85
    void clear region(int x ul, int y ul, int x size, int y size)
 86
 87
         /* variables */
 88
         int x;
                      /* x coordinate to clear */
 89
                       /* y coordinate to clear */
         int y;
 90
 91
 92
93
 94
         /* loop, clearing the display */
         for (x = x_ul; x < (x_ul + x_size); x++) {
    for (y = y_ul; y < (y_ul + y_size); y++) {
 95
96
 97
 98
              /* clear this pixel */
99
              plot_pixel(x, y, PIXEL_CLEAR);
100
101
         }
102
103
         /* done clearing the display region - return */
104
105
         return;
106
107
108
109
110
111
112
        plot_hline
113
114
                             This function draws a horizontal line from the passed position for the passed length. The line is always drawn
        Description:
115
116
                             with the color PIXEL LINE. The position (0,0) is the
117
                    upper left corner of the screen.
118
119
120
        Arguments:
                             start x (int) - starting x coordinate of the line.
                   start_y (int) - starting y coordinate of the line.
length (int) - length of the line (positive for a line
121
122
                                 to the "right" and negative for a line to
123
                             the "left").
124
125
        Return Value:
                             None.
126
        Input:
                             None.
127
128
        Output:
                             A horizontal line is drawn at the specified position.
129
130
        Error Handling:
                             No error checking is done on the coordinates.
131
        Algorithms:
                             None.
132
133
        Data Structures:
                             None.
134
        Global Variables: None.
135
136
137
        Author:
                             Glen George
        Last Modified:
                             June 03, 2014
138
139
140
141
142
    void plot_hline(int start_x, int start_y, int length)
143
         /* variables */
144
                      /* x position while plotting */
145
         int x;
146
                            /* starting x position to plot */
147
         int init_x;
         int end \bar{x};
                            /* ending x position to plot */
148
149
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;
        end \bar{x} = start \bar{x} + length;
157
158
        else {
159
160
            /* line to the "left" - start at start x + length, end at start x */
161
        init_x = start_x + length;
162
163
        end x = start x;
        }
165
166
        /* loop, outputting points for the line (always draw to the "right") */
167
        for (x = init_x; x < end_x; x++)
168
            /* plot a point of the line */
169
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
       plot_vline
182
183
184
       Description:
                           This function draws a vertical line from the passed
                          position for the passed length. The line is always drawn
185
186
                           with the color PIXEL LINE. The position (0,0) is the
187
                  upper left corner of the screen.
188
                          start_x (int) - starting x coordinate of the line.
189
       Arguments:
190
                  start_y (int) - starting y coordinate of the line.
                  length (int) - length of the line (positive for a line
191
                               going "down" and negative for a line
192
                           going "up").
193
194
       Return Value:
                           None.
195
       Input:
196
                           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
       Algorithms:
                           None.
201
       Data Structures:
202
                          None.
203
       Global Variables: None.
204
205
206
       Author:
                           Glen George
       Last Modified:
                          June 03, 2014
207
208
209
210
211
    void plot_vline(int start_x, int start_y, int length)
212
        /* variables */
213
214
                    /* y position while plotting */
        int y;
215
                         /* starting y position to plot */
216
        int init_y;
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
             /* line going "down" - start at start_y, end at start_y + length */
224
225
        init_y = start_y;
```

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

```
301
         int color = PIXEL_TEXT_N; /* pixel drawing color */
302
303
304
305
         /* setup the pixel positions for the character */
        x = pos x * HORIZ SIZE;
306
        y = pos y * VERT SIZE;
307
308
309
310
         /* loop outputting the bits to the screen */
         for (row = 0; row < VERT SIZE; row++)</pre>
311
312
             /* get the character bits for this row from the character table */
313
         if (row == (VERT SIZE - 1))
314
             /* last row - blank it */
315
316
             bits = 0;
317
         else
             /st in middle of character, get the row from the bit patterns st/
318
319
                  bits = char_patterns[(c * (VERT_SIZE - 1)) + row];
320
         /* take care of "normal/reverse video" */
321
322
         if (style == REVERSE)
323
             /* invert the bits for "reverse video" */
324
             bits = ~bits;
      if (style == HIGHLIGHTED)
325
           color = PIXEL_TEXT_H;
326
327
328
             /* get the bits "in position" (high bit is output first */
        bits <<= (8 - HORIZ SIZE);
329
330
331
         /* now output the row of the character, pixel by pixel */
332
333
         for (col = 0; col < HORIZ_SIZE; col++)</pre>
334
                  /* output this pixel in the appropriate color */
335
336
             if ((bits & 0x80) == 0)
             /* blank pixel - output in PIXEL_CLEAR */
plot_pixel(x + col, y, PIXEL_CLEAR);
337
338
339
             /* black pixel - output in PIXEL_TEXT */
plot_pixel(x + col, y, color);
340
341
             /* shift the next bit into position */
343
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
                            character position with (0,0) being the upper left corner
366
367
                   of the screen. There is no line wrapping, so the entire
                   string must fit on the passed line (pos_y). The string can be drawn in "normal video" (black on white) or
368
369
                   "reverse video" (white on black).
370
371
372
       Arguments:
                                                        - x coordinate (in character
                            pos_x (int)
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.
                 style (enum char style) - style with which to plot
379
380
                                        characters of the string.
381
       Return Value:
                          None.
382
383
       Input:
                          None.
                          A string is output to the LCD screen.
384
       Output:
385
386
       Error Handling: No checking is done to insure the string is fully on the
                 screen (the x and y coordinates and length of the string
387
                 are not checked).
388
389
       Algorithms:
390
                          None.
       Data Structures: None.
391
392
       Global Variables: None.
393
394
                          Glen George
395
       Last Modified: Mar. 17, 1997
396
397
398
399
400
    void plot string(int pos x, int pos y, const char *s, enum char style style)
401
        /* variables */
402
403
         /* none */
404
405
406
        /* loop, outputting characters from string s */
407
408
        while (*s != '\0')
409
            /* output this character and move to the next character and screen position */
410
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
5
                                LCD Output Functions
  /*
                                                                                */
                                    Include File
6
   /*
                            Digital Oscilloscope Project
                                                                                */
7
   /*
                                     EE/CS 52
                                                                                */
   /*
8
             ************
9
10
11
      This file contains the constants and function prototypes for the LCD output
12
      functions used in the Digital Oscilloscope project and defined in lcdout.c.
13
14
15
16
      Revision History:
         3/8/94
                                    Initial revision.
17
                 Glen George
         3/13/94
                  Glen George
                                    Updated comments.
18
                                    Added enumerated type char_style and updated
19
         3/17/97 Glen George
20
                                       function prototypes.
         6/3/14
                                   Added highlighted character style.
                  Santiago Navonne
21
22
23
24
25
26
             LCDOUT H
2.7
   #ifndef
28
       #define __LCDOUT_H_
29
30
   /* library include files */
31
     /* none */
32
33
   /* local include files */
34
     /* none */
35
36
37
38
39
   /* constants */
40
41
   /* character output styles */
43
   /* size of a character (includes 1 pixel space to the left and below character) */
44
45
   #define VERT SIZE
                        8
                                   /* vertical size (in pixels -> 7+1) */
   #define HORIZ_SIZE
                         6
                                   /* horizontal size (in pixels -> 5+1) */
46
47
48
49
50
   /* structures, unions, and typedefs */
51
52
53
   /* character output styles */
                                    /* "normal video" */
   enum char_style { NORMAL,
54
                                      /* "reverse video" */
55
                        REVERSE.
56
                        HIGHLIGHTED /* highlighted text */
57
                 };
58
59
60
61
   /* function declarations */
62
63
   void clear_region(int, int, int, int);
                                                /* clear part of the display */
64
65
   void plot_hline(int, int, int);
                                             /* draw a horizontal line */
66
67
   void plot_vline(int, int, int);
                                             /* draw a vertical line */
68
   void plot_char(int, int, char, enum char_style); /* output a character */
69
70
        plot string(int, int, const char *, enum char style); /* output a string */
   void
71
                                         /* draws the cursor on the trace */
72
   int
         plot cursor(int, int);
73
74
75
   #endif
```

```
2
3
  /*
                                      MAINLOOP
  /*
                                                                                */
4
5
                                  Main Program Loop
  /*
                             Digital Oscilloscope Project
6
   /*
                                      EE/CS 52
7
      ******************
8
9
10
      This file contains the main processing loop (background) for the Digital
11
      Oscilloscope project. The only global function included is: main - background processing loop
12
13
14
      The local functions included are:
15
16
         key_lookup - get a key and look up its keycode
17
      The locally global variable definitions included are:
18
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
                    (in addition to const).
25
         3/9/94
                  Glen George
                                   Moved the position of the const keyword in
26
                    declarations of arrays of pointers.
2.7
28
         3/13/94
                  Glen George
                                    Updated comments.
         3/13/94
                  Glen George
                                    Removed display menu call after plot trace,
29
30
                    the plot function takes care of the menu.
         3/17/97
31
                  Glen George
                                    Updated comments.
         3/17/97
                  Glen George
                                    Made key_lookup function static to make it
32
33
                    truly local.
         3/17/97
                  Glen George
                                    Removed KEY UNUSED and KEYCODE UNUSED
34
                    references (no longer used).
35
36
         5/27/08 Glen George
                                    Changed code to only check for sample done if
37
                    it is currently sampling.
         6/03/14 Santiago Navonne Added initialization code.
38
39
         6/11/14 Santiago Navonne Added sleep time between draws.
40
41
43
   /* library include files */
44
   #include "unistd.h"
45
46
47
   /* local include files */
  #include "interfac.h"
48
             "scopedef.h"
   #include
49
50
   #include
            "keyproc.h"
            "menu.h"
  #include
51
   #include "tracutil.h"
52
53
54
55
56
   /* local function declarations */
57
   static enum keycode key_lookup(void);
                                             /* translate key values into keycodes */
58
59
60
61
62
63
64
      main
65
      Description:
                        This procedure is the main program loop for the Digital
66
67
                        Oscilloscope. It loops getting keys from the keypad,
                        processing those keys as is appropriate. It also handles
68
                        starting scope sample collection and updating the LCD
69
                        screen. Additionally, it initializes the triggering logic
70
71
                        and key interface.
72
73
      Arguments:
      Return Value:
74
                        (int) - return code, always 0 (never returns).
75
```

```
76
       Input:
                          Keys from the keypad.
77
       Output:
                          Traces and menus to the display.
78
       Error Handling:
                          Invalid input is ignored.
79
80
                          The function is table-driven. The processing routines
81
       Algorithms:
                          for each input are given in tables which are selected
82
83
                          based on the context (state) the program is operating in.
                          Array (process_key) to associate keys with actions
       Data Structures:
84
85
                  (functions to call).
86
       Global Variables: None.
87
88
       Author:
                          Glen George
89
                          June 11, 2014
       Last Modified:
90
91
    */
92
93
94
    int main()
95
    {
        /* initialize keys, triggering */
96
97
          keys_init();
98
          trigger_init();
99
        /* variables */
100
        enum keycode
                                               /* an input key */
101
                              key;
102
103
        enum status
        state = MENU ON;
                              /* current program state */
104
105
                                           /* a captured trace */
106
        unsigned char *sample;
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) =
           /*
                Current System State
110
           /*
111
               MENU ON
                             MENU OFF
                                                    Input Key
                                                /* <Menu>
112
               menu_key,
                             menu key
          { {
                                           },
                                                /* <Up>
113
               menu_up,
                             no_action
                                           },
                                                /* <Down>
               menu down,
114
                              no action
                                           },
                             no\_action
               menu_left,
                                                /* <Left>
115
                                           },
                                                /* <Right>
116
               menu right,
                              no action
                                           } }; /* illegal key */
117
               no action,
                             no action
118
119
120
        /* first initialize everything */
121
122
        clear_display();
                                  /* clear the display */
123
                              /* initialize the trace routines */
124
        init_trace();
125
        init menu();
                              /* initialize the menu system */
126
127
128
        /* infinite loop processing input */
        while(TRUE) {
129
130
131
            /* check if ready to do a trace */
        if (trace rdy())
132
133
            /* ready for a trace - do it */
134
            do trace();
135
136
137
        /* check if have a trace to display */
        if (is_sampling() && ((sample = sample_done()) != NULL)) {
138
139
140
            /* have a trace - output it */
            plot_trace(sample);
141
142
            /* sleep for some time to reduce blinking of display */
143
            /*usleep(DRAW_INTERVAL);
144
145
            /* done processing this trace */
146
147
            trace_done();
148
        }
149
```

150

```
151
        /* now check for keypad input */
152
        if (key_available()) {
153
             /* have keypad input - get the key */
154
155
             key = key_lookup();
156
             /* execute processing routine for that key */
157
158
             state = process_key[key][state](state);
159
160
        }
161
162
        /* done with main (never should get here), return 0 */
163
165
166
    }
167
168
169
170
171
172
       key_lookup
173
174
       Description:
                           This function gets a key from the keypad and translates
                           the raw keycode to an enumerated keycode for the main
175
176
                           loop.
177
178
       Arguments:
       Return Value:
                           (enum keycode) - type of the key input on keypad.
179
180
181
       Input:
                           Keys from the keypad.
       Output:
182
                           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
       Last Modified:
                           Mar. 17, 1997
192
193
194
195
    static enum keycode key_lookup()
196
197
        /* variables */
198
199
200
        const static enum keycode keycodes[] = /* array of keycodes */
                                           /* order must match keys array exactly */
201
                                     /* <Menu>
                                                    */ /* also need an extra element */
                KEYCODE_MENU,
202
                                                     /* for unknown key codes */
203
           KEYCODE UP,
                                 /* <Up>
                                /* <Down>
                                                */
           KEYCODE DOWN,
204
                                 /* <Left>
                                                */
205
           KEYCODE_LEFT,
           KEYCODE_RIGHT,
KEYCODE ILLEGAL
206
                                 /* <Right>
                                /* other keys */
207
208
209
        const static int keys[] = /* array of key values */
210
211
                           /* order must match keycodes array exactly */
             {
                              /* <Menu>
212
                KEY MENU,
                          /* <Up>
           KEY UP,
213
                          /* <Down>
                                         */
214
           KEY DOWN,
                          /* <Left>
215
           KEY_LEFT,
                          /* <Right>
                                         * /
           KEY RIGHT,
216
217
             };
218
                          /* an input key */
219
        int key;
220
221
        int. i:
                              /* general loop index */
222
223
224
225
        /* get a key */
```

```
*************
2
3
  /*
                                         MENU
4
5
  /*
                                                                                  */
                                    Menu Functions
  /*
                             Digital Oscilloscope Project
6
   /*
                                       EE/CS 52
7
       *****************
8
9
10
      This file contains the functions for processing menu entries for the
11
      Digital Oscilloscope project. These functions take care of maintaining the
12
      menus and handling menu updates for the system. The functions included
13
14
         clear menu
                           - remove the menu from the display
15
                           - display the menu
16
         display menu
                           - initialize menus
17
         init_menu
         menu_entry_left - take care of <Left> key for a menu entry menu_entry_right - take care of <Right> key for a menu entry
18
19
                          - next menu entry
20
         next entry
         previous_entry
                          - previous menu entry
21
                          - re-display the menu if currently being displayed
22
         refresh menu
                           - reset the current selection to the top of the menu
23
         reset menu
24
      The local functions included are:
25
         display entry
                          - display a menu entry (including option setting)
26
2.7
28
      The locally global variable definitions included are:
                          - the menu
29
         menu
         menu_display
30
                           - whether or not the menu is currently displayed
                           - the currently selected menu entry
31
         menu_entry
32
33
      Revision History
34
                  Glen George
         3/8/94
                                     Initial revision.
35
36
         3/9/94
                  Glen George
                                     Changed position of const keyword in array
                     declarations involving pointers.
37
         3/13/94 Glen George
                                     Updated comments.
38
39
         3/13/94
                  Glen George
                                     Added display entry function to output a menu
40
                    entry and option setting to the LCD (affects
                    many functions).
41
         3/13/94 Glen George
                                     Changed calls to set status due to changing
42
                         enum scale status definition.
43
         3/13/94 Glen George
                                     No longer clear the menu area before
44
45
                    restoring the trace in clear menu() (not
                     needed).
46
47
         3/17/97
                  Glen George
                                     Updated comments.
         3/17/97
                                     Fixed minor bug in reset menu().
48
                  Glen George
         3/17/97
                  Glen George
                                     When initializing the menu in init_menu(),
49
50
                     set the delay to MIN DELAY instead of 0 and
                     trigger to a middle value instead of
51
                    MIN TRG LEVEL SET.
52
53
         5/3/06
                  Glen George
                                     Changed to a more appropriate constant in
                                     display entry().
54
55
         5/3/06
                  Glen George
                                     Updated comments.
         5/9/06
                                     Changed menus to handle a list for mode and
56
                  Glen George
                                 scale (move up and down list), instead of
57
                         toggling values.
58
59
60
61
62
   /* library include files */
63
64
     /* none */
65
   /* local include files */
66
67
   #include "scopedef.h"
             "lcdout.h"
   #include
68
             "menu.h"
  #include
69
   #include
             "menuact.h"
70
71
   #include
             "tracutil.h"
72
73
74
75
```

```
/* local function declarations */
77
    static void display entry(int, int);
                                               /* display a menu entry and its setting */
78
79
80
81
    /* locally global variables */
82
    static int menu_display;
                                          /* TRUE if menu is currently displayed */
83
84
    const static struct menu_item menu[] =
85
                                                  /* the menu */
        0, 4, display_mode
86
                                                 },
                       0, 5, display_scale 0, 5, display_sweep
87
                                                 },
            "Sweep",
88
                                                 },
            "Trigger", 0, 7, no_display
 89
           "Level",
                       2, 7, display_trg_level },
2, 7, display_trg_slope },
90
            "Slope",
91
            "Delay",
                       2, 7, display_trg_delay },
92
93
94
    static int menu entry; /* currently selected menu entry */
95
96
97
98
99
100
       init menu
101
102
103
       Description:
                          This function initializes the menu routines. It sets
                          the current menu entry to the first entry, indicates the
104
105
                 display is off, and initializes the options (and
106
                 hardware) to normal trigger mode, scale displayed, the
                 fastest sweep rate, a middle trigger level, positive
107
108
                 trigger slope, and minimum delay. Finally, it displays
109
110
111
       Arguments:
                          None.
       Return Value:
112
                          None.
113
       Input:
                          None.
114
                          The menu is displayed.
115
       Output:
116
       Error Handling:
117
118
119
       Algorithms:
                          None.
       Data Structures: None.
120
121
122
       Global Variables: menu_display - reset to FALSE.
                 menu entry - reset to first entry (0).
123
124
125
       Author:
                          Glen George
       Last Modified:
                         Mar. 17, 1997
126
127
128
129
130
    void init menu(void)
131
        /* variables */
132
133
          /* none */
134
135
136
137
        /* set the menu parameters */
        138
139
140
141
142
        /* set the scope (option) parameters */
        set_trigger_mode(NORMAL_TRIGGER); /* normal triggering */
143
                                   /* scale is axes */
144
        set_scale(SCALE_AXES);
                                 /* first sweep rate */
145
        set sweep(0);
        set_trg_level((MIN_TRG_LEVEL_SET + MAX_TRG_LEVEL_SET) / 2); /* middle trigger level */
146
        set_trg_slope(SLOPE_POSITIVE); /* positive slope */
set_trg_delay(MIN_DELAY); /* minimum delay */
147
148
149
150
```

```
151
         /* now display the menu */
152
         display_menu();
153
154
         /* done initializing, return */
155
156
157
158
    }
159
160
161
162
163
        clear menu
164
165
166
        Description:
                             This function removes the menu from the display. The
                             trace under the menu is restored. The flag menu_display,
167
                   is cleared, indicating the menu is no longer being displayed. Note: if the menu is not currently being
168
169
                   displayed this function does nothing.
170
171
172
        Arguments:
                             None.
173
       Return Value:
                             None.
174
        Input:
175
        Output:
                             The menu if displayed, is removed and the trace under it
176
                   is rewritten.
177
178
       Error Handling:
                             None.
179
180
181
        Algorithms:
                             None.
       Data Structures:
182
                             None.
183
184
        Global Variables: menu display - checked and set to FALSE.
185
186
        Author:
                             Glen George
        Last Modified:
                             Mar. 13, 1994
187
188
189
    */
190
    void clear menu(void)
191
192
         /* variables */
193
194
           /* none */
195
196
197
         /* check if the menu is currently being displayed */
198
199
         if (menu_display) {
200
              /* menu is being displayed - turn it off and restore the trace in that area */
201
         restore_menu_trace();
202
203
204
205
206
         /* no longer displaying the menu */
         menu display = FALSE;
207
208
209
         /* all done, return */
210
211
         return;
212
213
    }
214
215
216
217
218
        display_menu
219
220
221
       Description:
                             This function displays the menu. The trace under the
                             menu is overwritten (but it was saved). The flag
222
                   menu_display, is also set, indicating the menu is currently being displayed. Note: if the menu is already
223
224
225
                   being displayed this function does not redisplay it.
```

```
226
227
       Arguments:
                           None.
       Return Value:
                           None.
228
229
230
       Input:
                           None.
       Output:
                           The menu is displayed.
231
232
233
       Error Handling:
                           None.
234
235
       Algorithms:
                           None.
       Data Structures:
                           None.
236
237
       Global Variables: menu_display - set to TRUE.
238
239
                  menu entry - used to highlight currently selected entry.
240
241
       Author:
                           Glen George
       Last Modified:
                           Mar. 13, 1994
242
243
    */
244
245
    void display_menu(void)
246
247
248
         /* variables */
249
        int i;
                     /* loop index */
250
251
252
253
         /* check if the menu is currently being displayed */
        if (!menu display)
254
255
256
             /* menu is not being displayed - turn it on */
         /* display it entry by entry */
257
         for (i = 0; i < NO_MENU_ENTRIES; i++)</pre>
258
259
             /* display this entry - check if it should be highlighted */
260
261
             if (i == menu entry)
                 /* currently selected entry - highlight it */
262
                 display_entry(i, TRUE);
263
264
                  /* not the currently selected entry - "normal video" */
265
                 display_entry(i, FALSE);
266
267
             }
268
        }
269
270
        /* now are displaying the menu */
271
272
        menu_display = TRUE;
273
274
         /* all done, return */
275
276
        return;
277
278
    }
279
280
281
282
283
       refresh menu
284
285
286
       Description:
                           This function displays the menu if it is currently being
                  displayed. The trace under the menu is overwritten (but
287
                  it was already saved).
288
289
290
       Arguments:
                           None.
       Return Value:
                           None.
291
292
293
       Input:
                           None.
                           The menu is displayed.
294
       Output:
295
296
       Error Handling:
                           None.
297
298
       Algorithms:
       Data Structures:
299
                           None.
300
```

```
301
       Global Variables: menu_display - determines if menu should be displayed.
302
303
       Author:
                           Glen George
       Last Modified:
                          Mar. 8, 1994
304
305
306
307
308
    void refresh_menu(void)
309
        /* variables */
310
          /* none */
311
312
313
314
        /* check if the menu is currently being displayed */
315
316
        if (menu_display)
317
             /* menu is currently being displayed - need to refresh it */
318
319
        /* do this by turning off the display, then forcing it back on */
        menu display = FALSE;
320
        display_menu();
321
322
323
324
        /* refreshed the menu if it was displayed, now return */
325
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
                  displayed the display is updated.
338
339
       Arguments:
340
                           None.
341
       Return Value:
                          None.
342
       Input:
343
                           None.
                           The menu display is updated if it is being displayed.
344
       Output:
345
       Error Handling:
346
                          None.
347
348
       Algorithms:
                           None.
349
       Data Structures: None.
350
       Global Variables: menu_display - checked to see if menu is displayed.
351
                               - reset to 0 (first entry).
352
                  menu_entry
353
       Author:
                           Glen George
354
355
       Last Modified:
                          Mar. 17, 1997
356
357
358
    void reset menu(void)
359
360
361
        /* variables */
362
          /* none */
363
364
365
        /* check if the menu is currently being displayed */
366
367
        if (menu_display)
368
             /* menu is being displayed */
369
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
        /* finally, highlight the first entry if the menu is being displayed */
379
380
        if (menu display)
        display_entry(menu_entry, TRUE);
381
382
383
384
        /* all done, return */
385
        return;
386
387
388
389
390
391
392
393
394
       next_entry
395
       Description:
                           This function changes the current menu selection to the
396
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
       Arguments:
                           None.
401
       Return Value:
402
                          None.
403
       Input:
                           None.
404
405
       Output:
                           The menu display is updated if it is being displayed and
406
                  the entry selected changes.
407
408
       Error Handling:
                           None.
409
       Algorithms:
                           None.
410
411
       Data Structures:
                          None.
412
       Global Variables: menu_display - checked to see if menu is displayed.
413
414
                               - updated to a new entry (if not at end).
                  menu entry
415
       Author:
416
                           Glen George
417
       Last Modified:
                          Mar. 13, 1994
418
419
420
    void next_entry(void)
421
422
    {
423
        /* variables */
          /* none */
424
425
426
427
428
        /* only update if not at end of the menu */
        if (menu entry < (NO MENU ENTRIES - 1)) {
429
430
431
             /* not at the end of the menu */
432
        /* turn off current entry if displaying */
433
        if (menu display)
434
                 /* displaying menu - turn off currently selected entry */
435
436
            display_entry(menu_entry, FALSE);
437
        /* update the menu entry to the next one */
438
439
        menu entry++;
440
        /* now highlight this entry if displaying the menu */
441
442
        if (menu_display)
                 \overline{/}* displaying menu - highlight newly selected entry */
443
             display_entry(menu_entry, TRUE);
444
445
        }
446
447
448
        /* all done, return */
        return;
449
450
```

```
451
   }
452
453
454
455
456
       previous entry
457
458
       Description:
                           This function changes the current menu selection to the
459
                           previous menu entry. If the current selection is the
460
                  first entry in the menu, it is not changed. If the menu
461
                  is currently being displayed, the display is updated.
462
463
       Arguments:
                           None.
464
       Return Value:
                           None.
465
466
467
       Input:
                           None.
                           The menu display is updated if it is being displayed and
       Output:
468
469
                  the currently selected entry changes.
470
       Error Handling:
                           None.
471
472
473
       Algorithms:
                           None.
474
       Data Structures:
                           None.
475
476
       Global Variables: menu display - checked to see if menu is displayed.
                                - updated to a new entry (if not at start).
477
                  menu entry
478
       Author:
479
                           Glen George
480
       Last Modified:
                           Mar. 13, 1994
481
482
483
    void previous entry(void)
484
485
    {
486
        /* variables */
487
          /* none */
488
489
490
        /* only update if not at the start of the menu */
491
        if (menu entry > 0) {
492
493
             /* not at the start of the menu */
494
495
        /* turn off current entry if displaying */
496
497
        if (menu_display)
                 \overline{/}* displaying menu - turn off currently selected entry */
498
499
            display_entry(menu_entry, FALSE);
500
501
        /* update the menu entry to the previous one */
        menu_entry--;
502
503
        /* now highlight this entry if displaying the menu */
504
505
        if (menu_display)
506
                 \overline{/}* displaying menu - highlight newly selected entry */
            display entry(menu entry, TRUE);
507
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
                           selection. It does this by doing a table lookup on the
524
525
                  current menu selection.
```

```
526
527
       Arguments:
                          None.
528
       Return Value:
                          None.
529
530
       Input:
                          The menu display is updated if it is being displayed and
531
       Output:
                  the <Left> key causes a change to the display.
532
533
       Error Handling:
                          None.
534
535
                          Table lookup is used to determine what to do for the
536
       Algorithms:
537
                  input key.
                          An array holds the table of key processing routines.
538
       Data Structures:
539
       Global Variables: menu_entry - used to select the processing function.
540
541
542
       Author:
                          Glen George
       Last Modified:
                          May 9, 2006
543
544
545
546
    void menu_entry_left(void)
547
548
549
        /* variables */
550
        /* key processing functions */
551
552
        static void
                     (* const process[])(void) =
553
               Mode
                                 Scale
                                                    Sweep
                                                                      Trigger
               mode down,
                                 scale down,
                                                    sweep down,
554
                                                                      trace rearm,
            {
555
                trg_level_down, trg_slope_toggle, trg_delay_down
                                                                                    };
556
               Level
                                 Slope
                                                    Delay
557
558
559
        /* invoke the appropriate <Left> key function */
560
561
        process[menu entry]();
562
        /st if displaying menu entries, display the new value st/
563
564
        /* note: since it is being changed - know this option is selected */
565
        if (menu display)
            menu[menu entry].display((MENU X + menu[menu entry].opt off),
566
                               (MENU Y + menu entry), OPTION SELECTED);
567
568
        }
569
570
        /* all done, return */
571
572
        return;
573
574
    }
575
576
577
578
579
580
       menu entry right
581
                          This function handles the <Right> key for the current
582
       Description:
583
                          menu selection. It does this by doing a table lookup on
                  the current menu selection.
584
585
586
       Arguments:
                          None.
587
       Return Value:
                          None.
588
589
       Input:
590
       Output:
                          The menu display is updated if it is being displayed and
                  the <Right> key causes a change to the display.
591
592
       Error Handling:
                          None.
593
594
                          Table lookup is used to determine what to do for the
595
       Algorithms:
596
                  input key.
597
       Data Structures: An array holds the table of key processing routines.
598
                                      - used to display the new menu value.
599
       Global Variables: menu
600
                  menu_entry - used to select the processing function.
```

```
601
602
       Author:
                          Glen George
       Last Modified:
                          May 9, 2006
603
604
605
    * /
606
    void menu entry right(void)
607
608
    {
609
        /* variables */
610
        /* key processing functions */
611
        static void (* const process[])(void) =
612
                                                                                  */
613
           /* Mode
                              Scale
                                                  Sweep
                                                                   Trigger
               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
        /* invoke the appropriate <Right> key function */
620
        process[menu_entry]();
621
622
623
        /* if displaying menu entries, display the new value */
624
        /* note: since it is being changed - know this option is selected */
        if (menu display)
625
            menu[menu_entry].display((MENU_X + menu[menu_entry].opt_off),
626
                               (MENU Y + menu entry), OPTION SELECTED);
627
628
        }
629
630
        /* all done, return */
631
632
        return:
633
634
    }
635
636
637
638
639
       display_entry
640
641
                          This function displays the passed menu entry and its
642
       Description:
                  current option setting. If the second argument is TRUE
643
                  it displays them with color SELECTED and OPTION_SELECTED
644
645
                  respectively. If the second argument is FALSE it
                  displays the menu entry with color NORMAL and the option
646
647
                  setting with color OPTION_NORMAL.
648
       Arguments:
649
                          entry (int)
                                          - menu entry to be displayed.
650
                  selected (int) - whether or not the menu entry is
                                currently selected (determines the color
651
                           with which the entry is output).
652
653
       Return Value:
                          None.
654
655
       Input:
                          None.
656
       Output:
                          The menu entry is output to the LCD.
657
658
       Error Handling:
                          None.
659
       Algorithms:
                          None.
660
661
       Data Structures:
                          None.
662
       Global Variables: menu - used to display the menu entry.
663
664
665
       Author:
                          Glen George
       Last Modified:
                          Aug. 13, 2004
666
667
668
669
    static void display entry(int entry, int selected)
670
671
        /* variables */
672
673
          /* none */
674
675
```

```
676
    677
678
679
680
681
682
683
684
    /* all done outputting this menu entry - return */
685
686
687
688
689
```

```
*****************
   /*
2
  /*
3
                                      MENU.H
  /*
                                  Menu Functions
                                                                               */
4
5
  /*
                                                                               */
                                   Include File
6
   /*
                            Digital Oscilloscope Project
  /*
7
                                     EE/CS 52
                                                                               */
   /*
8
          ****************
9
10
11
      This file contains the constants and function prototypes for the functions
12
     which deal with menus (defined in menu.c) for the Digital Oscilloscope
13
14
     project.
15
16
17
     Revision History:
         3/8/94
                 Glen George
                                   Initial revision.
18
19
         3/13/94
                Glen George
                                   Updated comments.
                                   Added definitions for SELECTED,
20
        3/13/94 Glen George
                       OPTION_NORMAL, and OPTION_SELECTED.
21
22
         6/03/14 Santiago Navonne Changed selected menu and option style to HIGHLIGHTED.
23
2.4
25
26
   #ifndef
             MENU H
2.7
28
      #define MENU_H_
29
30
   /* library include files */
31
     /* none */
32
33
   /* local include files */
34
  #include "interfac.h"
35
            "scopedef.h"
  #include
36
   #include "lcdout.h"
37
38
39
40
41
   /* constants */
42
43
   /* menu size */
44
45
   #define MENU WIDTH
                                  /* menu width (in characters) */
                        7
  #define MENU_HEIGHT
                                  /* menu height (in characters) */
46
  #define MENU_SIZE_X (MENU_WIDTH * HORIZ_SIZE) /* menu width (in pixels) */
#define MENU_SIZE_Y (MENU_HEIGHT * VERT_SIZE) /* menu height (in pixels) */
47
49
50
   /* menu position */
                     (LCD_WIDTH - MENU_WIDTH - 1) /* x position (in characters) */
  #define MENU X
51
  #define MENU_Y
                                                         /* y position (in characters) */
52
                      0
                                                   /* x position (in pixels) */
53
   #define MENU_UL_X (MENU_X * HORIZ_SIZE)
   #define MENU_UL_Y (MENU_Y * VERT_SIZE)
                                                   /* y position (in pixels) */
54
55
56
   /* menu colors */
  #define SELECTED
                            HIGHLIGHTED
                                            /* color for a selected menu entry */
57
   #define OPTION_SELECTED HIGHLIGHTED /* color for a selected menu entry option */
58
   #define OPTION NORMAL
                            NORMAL
                                          /* color for an unselected menu entry option */
59
60
   /* number of menu entries */
61
   #define NO MENU ENTRIES (sizeof(menu) / sizeof(struct menu item))
62
63
64
65
66
67
   /* structures, unions, and typedefs */
68
69
   /* data for an item in a menu */
  70
71
72
73
                void
                          (*display)(int, int, int); /* option display function */
74
            };
75
```

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

```
***********
3
   /*
                                           MENUACT
   /*
4
5
                                   Menu Action Functions
   /*
                                Digital Oscilloscope Project
6
   /*
                                            EE/CS 52
7
        *****************
8
9
10
       This file contains the functions for carrying out menu actions for the
11
      Digital Oscilloscope project. These functions are invoked when the <Left> or <Right> key is pressed for a menu item. Also included are the functions
12
13
       for displaying the current menu option selection. The functions included
14
15
16
          display mode
                               - display trigger mode
                               - display the scale type
17
          display_scale
          display_sweep - display the sweep rate
display_trg_delay - display the tigger delay
display_trg_level - display the trigger level
18
19
20
          display_trg_slope - display the trigger slope
21
          get_trigger_mode - get the current trigger mode
mode down - go to the "next" trigger mode
22
23
                              - go to the "previous" trigger mode
24
          mode_up
                              - nothing to display for option setting
          no display
25
          no_display - nothing to display for option setting
no menu action - no action to perform for <Left> or <Right> key
26
          scale_down
                              - go to the "next" scale type
2.7
                              - go to the "previous" scale type
28
          scale up
                              - set the scale type
          set scale
29
30
          set_sweep
                              - set the sweep rate
          set_trg_delay
set_trg_level
                              - set the tigger delay
31
                              - set the trigger level
32
          set_trg_slope - set the trigger slope
set_trigger_mode - set the trigger mode
33
34
                              - decrease the sweep rate
          sweep_down
35
                              - increase the sweep rate
36
          sweep up
          trg_delay_down
                              - decrease the trigger delay
37
          trg_delay_up
                              - increase the trigger delay
38
          trg level down - decrease the trigger level
39
          trg_level_up - increase the trigger level trg_slope_toggle - toggle the trigger slope between "+" and "-"
40
41
42
      The local functions included are:
43
          adjust_trg_delay - adjust the trigger delay for a new sweep rate cvt_num_field - converts a numeric field value to a string
44
45
46
47
       The locally global variable definitions included are:
          delay

    current trigger delay

48
                          - current trigger level
49
          level
                          - current display scale type
50
          scale
                          - current trigger slope
          slope
51
                          - current sweep rate
52
          sweep
          sweep_rates
53
                          - table of information on possible sweep rates
          trigger mode - current triggering mode
54
55
56
      Revision History
57
                    Glen George
                                         Initial revision.
58
          3/8/94
          3/13/94
                    Glen George
                                         Updated comments.
59
          3/13/94 Glen George
                                         Changed all arrays of constant strings to be
60
61
                            static so compiler generates correct code.
                           George Changed scale to type enum scale_type and output the selection as "None" or "Axes".
62
          3/13/94 Glen George
63
                       This will allow for easier future expansion.
64
                                      Changed name of set axes function (in
65
          3/13/94 Glen George
                            tracutil.c) to set_display_scale.
66
67
          3/10/95
                    Glen George
                                    Changed calculation of displayed trigger
                            level to use constants MIN_TRG_LEVEL_SET and
68
                       MAX_TRG_LEVEL_SET to get the trigger level
69
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
                            1 us and updated functions to handle these
74
75
                       new rates.
```

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

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

```
226
227
    }
228
229
230
231
232
233
       set_trigger_mode
234
235
       Description:
                            This function sets the triggering mode to the passed
236
237
                           m (enum trigger_type) - mode to which to set the
       Arguments:
238
239
                                triggering mode.
       Return Value:
240
                            None.
241
                            None.
242
       Input:
       Output:
                            None.
243
244
       Error Handling:
                            None.
245
246
247
       Algorithms:
                            None.
248
       Data Structures:
                           None.
249
250
       Global Variables: trigger mode - initialized to the passed value.
251
       Author:
                            Glen George
252
253
       Last Modified:
                           Mar. 8, 1994
254
255
256
    void set_trigger_mode(enum trigger_type m)
257
258
    {
259
         /* variables */
           /* none */
260
261
262
263
264
         /* set the trigger mode */
        trigger_mode = m;
265
266
267
         /* set the new mode */
        set mode(trigger mode);
268
269
270
         /* all done setting the trigger mode - return */
271
272
        return;
273
274
    }
275
276
277
278
279
280
       get_trigger_mode
281
       Description:
                           This function returns the current triggering mode.
282
283
284
       Arguments:
                            (enum trigger_type) - current triggering mode.
285
       Return Value:
286
287
       Input:
                            None.
       Output:
288
                            None.
289
290
       Error Handling:
                            None.
291
292
       Algorithms:
                            None.
       Data Structures:
                           None.
293
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
    {
        /* variables */
304
305
           /* none */
306
307
308
309
        /* return the current trigger mode */
        return trigger_mode;
310
311
312
    }
313
314
315
316
317
       mode_down
318
319
                           This function handles moving down the list of trigger
320
       Description:
                           modes. It changes to the "next" triggering mode and
321
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
       Algorithms:
332
                           None.
333
       Data Structures:
                           None.
334
       Global Variables: trigger_mode - changed to "next" trigger mode.
335
336
337
       Author:
                           Glen George
       Last Modified:
                           May 9, 2006
338
339
    */
340
341
          mode down()
342
    void
343
344
        /* variables */
345
           /* none */
346
347
348
        /* move to the "next" triggering mode */
349
350
        if (trigger mode == NORMAL TRIGGER)
             trigger_mode = AUTO_TRIGGER;
351
        else if (trigger_mode == AUTO_TRIGGER)
352
             trigger_mode = ONESHOT_TRIGGER;
353
        else
354
             trigger_mode = NORMAL_TRIGGER;
355
356
        /* set the new mode */
357
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
                           sets that as the current mode.
374
375
```

```
376
       Arguments:
                           None.
377
       Return Value:
                           None.
378
                           None.
379
       Input:
380
       Output:
                           None.
381
       Error Handling:
                           None.
382
383
384
       Algorithms:
                           None.
385
       Data Structures:
                           None.
386
       Global Variables: trigger mode - changed to "previous" trigger mode.
387
388
389
                           Glen George
                           May 9, 2006
       Last Modified:
390
391
392
393
394
    void mode_up()
395
        /* variables */
396
397
          /* none */
398
399
400
        /* move to the "previous" triggering mode */
401
        if (trigger_mode == NORMAL_TRIGGER)
402
             trigger_mode = ONESHOT_TRIGGER;
403
        else if (trigger_mode == AUTO TRIGGER)
404
405
            trigger_mode = NORMAL_TRIGGER;
406
        else
            trigger_mode = AUTO_TRIGGER;
407
408
409
        /* set the new mode */
        set_mode(trigger_mode);
410
411
412
        /* all done with the trigger mode - return */
413
414
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
                           x_pos (int) - x position (in character cells) at which to
       Arguments:
427
428
                         display the trigger mode.
                  y_pos (int) - y position (in character cells) at which to
429
430
                         display the trigger mode.
                  style (int) - style with which to display the trigger
431
                             mode.
432
433
       Return Value:
                           None.
434
       Input:
                           None.
435
436
       Output:
                           The trigger mode is displayed at the passed position on
437
                  the screen.
438
439
       Error Handling:
                           None.
440
       Algorithms:
441
                           None.
442
       Data Structures:
                           None.
443
       Global Variables: trigger_mode - determines which string is displayed.
444
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) */
        const static char * const modes[] = {
456
                                                       Normal
                                                     " Automatic"
457
                                                     " One-Shot " };
458
459
460
461
        /* display the trigger mode */
462
        plot_string(x_pos, y_pos, modes[trigger_mode], style);
463
464
465
        /* all done displaying the trigger mode - return */
466
467
        return;
468
469
    }
470
471
472
473
474
475
       set scale
476
       Description:
                           This function sets the scale type to the passed value.
477
478
       Arguments:
                           s (enum scale type) - scale type to which to initialize
479
480
                             the scale status.
       Return Value:
                           None.
481
482
483
       Input:
                           None.
484
       Output:
                           The new trace display is updated with the new scale.
485
486
       Error Handling:
                          None.
487
       Algorithms:
                          None.
488
489
       Data Structures:
                          None.
490
       Global Variables: scale - initialized to the passed value.
491
492
493
       Author:
                           Glen George
       Last Modified:
                          Mar. 13, 1994
494
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
        /* output the scale appropriately */
508
        set display scale(scale);
509
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
                           types. It changes to the "next" type of scale and sets
524
525
                  this as the current scale type.
```

```
526
527
       Arguments:
                            None.
       Return Value:
                             None.
528
529
530
       Input:
                             None.
       Output:
                             The new scale is output to the trace display.
531
532
533
       Error Handling:
                            None.
534
535
       Algorithms:
                             None.
536
       Data Structures:
                            None.
537
       Global Variables: scale - changed to the "next" scale type.
538
539
       Author:
                             Glen George
540
                            May 9, 2006
541
       Last Modified:
542
543
544
    void scale down()
545
546
547
         /* variables */
548
           /* none */
549
550
551
         /* change to the "next" scale type */
552
553
         if (scale == SCALE NONE)
             scale = SCALE_AXES;
554
555
         else if (scale == SCALE_AXES)
             scale = SCALE_GRID;
556
557
         else
558
             scale = SCALE_NONE;
559
         /* set the scale type */
560
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
                            This function handles moving up the list of scale types. It changes to the "previous" type of scale and sets this
575
       Description:
576
                   as the current scale type.
577
578
579
       Arguments:
                            None.
580
       Return Value:
                            None.
581
       Input:
582
                            The new scale is output to the trace display.
583
       Output:
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
       Last Modified:
                            May 9, 2006
593
594
595
596
597
    void scale_up()
598
         /* variables */
599
600
           /* none */
```

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

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

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

```
826
827
         /* increase the sweep rate, if not already the maximum */
828
        if (sweep < (NO_SWEEP_RATES - 1)) {
    /* not at maximum, adjust delay for new sweep */</pre>
829
830
         adjust_trg_delay(sweep, (sweep + 1));
831
         /* now set new sweep rate */
832
833
             sweep++;
834
835
         /* set the sweep rate for the hardware */
836
         sample_size = set_sample_rate(sweep_rates[sweep].sample_rate);
837
         /* also set the sample size for the trace capture */
838
         set trace size(sample size);
839
840
841
         /* all done with raising the sweep rate - return */
842
         return:
843
844
845
846
847
848
849
850
       display sweep
851
852
853
       Description:
                            This function displays the current sweep rate at the
                           passed position, in the passed style.
854
855
                            x_pos (int) - x position (in character cells) at which to
856
       Arguments:
                         display the sweep rate.
857
858
                         (int) - y position (in character cells) at which to
859
                         display the sweep rate.
                   style (int) - style with which to display the sweep rate.
860
861
       Return Value:
                           None.
862
       Input:
                           None.
863
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
          display sweep(int x pos, int y pos, int style)
879
880
881
         /* variables */
           /* none */
882
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 */
         return:
891
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
                           1 (int) - value to which to set the trigger level.
903
       Arguments:
       Return Value:
904
                           None.
905
906
       Input:
                           None.
       Output:
                           None.
907
908
909
       Error Handling:
                           The passed value is not checked for validity.
910
       Algorithms:
911
       Data Structures:
                           None.
912
913
914
       Global Variables: level - initialized to the passed value.
915
916
       Author:
                           Glen George
       Last Modified:
                           Mar. 8, 1994
917
918
    * /
919
920
    void set_trg_level(int l)
921
922
    {
923
        /* variables */
924
          /* none */
925
926
927
928
        /* set the trigger level */
        level = 1;
929
930
931
        /* set the trigger level in hardware too */
        set_trigger(level, slope);
932
933
934
        /* all done initializing the trigger level - return */
935
936
        return;
937
938
939
940
941
942
943
       trg_level_down
944
945
                           This function handles decreasing the current trigger
946
       Description:
947
                           The new trigger level is sent to the hardware.
                  If an attempt is made to lower the trigger level below
948
                  the minimum value it is not changed.
949
950
951
       Arguments:
                           None.
       Return Value:
                           None.
952
953
       Input:
                           None.
954
955
       Output:
                           None.
956
       Error Handling:
957
                           None.
958
       Algorithms:
959
       Data Structures:
960
                           None.
961
       Global Variables: level - decremented if not already at the minimum value.
962
963
964
       Author:
                           Glen George
       Last Modified:
                           Mar. 8, 1994
965
966
967
    */
968
    void trg_level_down()
969
970
    {
971
        /* variables */
          /* none */
972
973
974
975
```

```
976
         /st decrease the trigger level, if not already the minimum st/
977
         if (level > MIN_TRG_LEVEL_SET)
             level--;
978
979
980
         /* set the trigger level for the hardware */
         set trigger(level, slope);
981
982
983
984
         /* all done with lowering the trigger level - return */
985
         return;
986
987
     }
988
989
990
991
992
        trg_level_up
993
994
                            This function handles increasing the current trigger
995
        Description:
                            The new trigger level is sent to the hardware.
                   level.
996
997
                   If an attempt is made to raise the trigger level above
998
                   the maximum value it is not changed.
999
1000
                            None.
        Arguments:
        Return Value:
1001
                            None.
1002
1003
        Input:
                            None.
        Output:
                            None.
1004
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
        Author:
                            Glen George
1013
1014
        Last Modified:
                            Mar. 8, 1994
1015
1016
1017
     void trg level up()
1018
1019
     {
1020
         /* variables */
           /* none */
1021
1022
1023
1024
1025
         /* increase the trigger level, if not already the maximum */
         if (level < MAX_TRG_LEVEL_SET)</pre>
1026
             level++;
1027
1028
         /* tell the hardware the new trigger level */
1029
1030
         set trigger(level, slope);
1031
1032
         /* all done raising the trigger level - return */
1033
1034
         return;
1035
1036
     }
1037
1038
1039
1040
1041
1042
        display_trg_level
1043
        Description:
                            This function displays the current trigger level at the
1044
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.
                   y_pos (int) - y position (in character cells) at which to
1049
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.
                            The trigger level is displayed at the passed position on
1056
        Output:
                   the display.
1057
1058
        Error Handling:
                           None.
1059
1060
        Algorithms:
1061
                           None.
        Data Structures:
1062
                           None.
1063
        Global Variables: level - determines the value displayed.
1064
1065
1066
        Author:
                            Glen George
1067
        Last Modified:
                           Mar. 10, 1995
1068
1069
    * /
1070
    void display_trg_level(int x_pos, int y_pos, int style)
1071
1072
1073
         /* variables */
1074
         char
                    level_str[] = "
                                             "; /* string containing the trigger level */
         long int 1;
                                   /* trigger level in mV */
1075
1076
1077
1078
         /* compute the trigger level in millivolts */
1079
1080
         1 = ((long int) MAX_LEVEL - MIN_LEVEL) * level / (MAX_TRG_LEVEL_SET - MIN_TRG_LEVEL_SET) + MIN_LEV
1081
         /* convert the level to the string (leave first character blank) */
1082
1083
         cvt_num_field(1, &level_str[1]);
1084
         /* add in the units */
1085
1086
         level str[7] = 'V';
1087
1088
1089
         /* now finally display the trigger level */
1090
         plot_string(x_pos, y_pos, level_str, style);
1091
1092
         /* all done displaying the trigger level - return */
1093
1094
         return;
1095
1096
    }
1097
1098
1099
1100
1101
        set_trg_slope
1102
1103
        Description:
                           This function sets the trigger slope to the passed value.
1104
1105
                            s (enum slope_type) - trigger slope type to which to set
1106
        Arguments:
                              the locally global slope.
1107
1108
        Return Value:
                           None.
1109
        Input:
                           None.
1110
1111
        Output:
                           None.
1112
        Error Handling:
1113
                           None.
1114
1115
        Algorithms:
                            None.
        Data Structures: None.
1116
1117
        Global Variables: slope - set to the passed value.
1118
1119
1120
                            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
         /* set the slope type */
1132
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
        trg_slope_toggle
1148
1149
1150
                            This function handles toggling (and setting) the current
        Description:
1151
                   trigger slope.
1152
1153
        Arguments:
                            None.
        Return Value:
                            None.
1154
1155
1156
        Input:
                            None.
1157
        Output:
                            None.
1158
1159
        Error Handling:
                            None.
1160
1161
        Algorithms:
                            None.
        Data Structures: None.
1162
1163
1164
        Global Variables: slope - toggled.
1165
        Author:
                            Glen George
1166
1167
        Last Modified:
                            Mar. 8, 1994
1168
1169
1170
     void trg_slope_toggle()
1171
1172
     {
1173
         /* variables */
           /* none */
1174
1175
1176
1177
1178
         /* toggle the trigger slope */
         if (slope == SLOPE POSITIVE)
1179
1180
              slope = SLOPE_NEGATIVE;
1181
         else
              slope = SLOPE POSITIVE;
1182
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
        Description:
                            This function displays the current trigger slope at the
1199
1200
                            passed position, in the passed style.
```

```
1201
                            x_pos (int) - x position (in character cells) at which to
1202
        Arguments:
1203
                         display the trigger slope.
                   y_pos (int) - y position (in character cells) at which to
1204
1205
                         display the trigger slope.
                   style (int) - style with which to display the trigger
1206
1207
                              slope.
1208
        Return Value:
                            None.
1209
1210
        Input:
                            None.
                            The trigger slope is displayed at the passed position on
1211
        Output:
1212
                   the screen.
1213
        Error Handling:
1214
1215
1216
        Algorithms:
                            None.
        Data Structures:
1217
                           None.
1218
1219
        Global Variables: slope - determines which string is displayed.
1220
        Author:
                            Glen George
1221
1222
        Last Modified:
                            Mar. 13, 1994
1223
1224
     * /
1225
     void display_trg_slope(int x_pos, int y_pos, int style)
1226
1227
     {
1228
         /* variables */
1229
1230
         /* the trigger slope strings (must match enumerated type) */
1231
         const static char * const slopes[] = { " +", " -"
                                                                    };
1232
1233
1234
         /* display the trigger slope */
1235
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
                            d (long int) - value to which to set the trigger delay.
        Arguments:
1252
1253
        Return Value:
                            None.
1254
1255
        Input:
                            None.
1256
        Output:
1257
                           The passed value is not checked for validity.
1258
        Error Handling:
1259
        Algorithms:
                            None.
1260
1261
        Data Structures:
                           None.
1262
        Global Variables: delay - initialized to the passed value.
1263
1264
1265
        Author:
                            Glen George
        Last Modified:
                           Mar. 8, 1994
1266
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 */
         set delay(delay);
1281
1282
1283
1284
         /* all done initializing the trigger delay - return */
1285
         return;
1286
1287
     }
1288
1289
1290
1291
1292
        trg_delay_down
1293
1294
                            This function handles decreasing the current trigger
1295
        Description:
                            The new trigger delay is sent to the hardware.
                   delay.
1296
1297
                   If an attempt is made to lower the trigger delay below
1298
                   the minimum value it is not changed.
1299
                            None.
1300
        Arguments:
        Return Value:
                            None.
1301
1302
1303
        Input:
                            None.
                            None.
        Output:
1304
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
        Author:
                            Glen George
1313
1314
        Last Modified:
                            Mar. 8, 1994
1315
1316
1317
     void trg delay down()
1318
1319
     {
1320
         /* variables */
           /* none */
1321
1322
1323
1324
         /* decrease the trigger delay, if not already the minimum */
1325
         if (delay > MIN_DELAY)
1326
             delay--;
1327
1328
         /* set the trigger delay for the hardware */
1329
1330
         set delay(delay);
1331
1332
         /* all done with lowering the trigger delay - return */
1333
         return;
1334
1335
1336
1337
1338
1339
1340
1341
1342
        trg_delay_up
1343
                            This function handles increasing the current trigger
1344
        Description:
1345
                            The new trigger delay is sent to the hardware.
                   If an attempt is made to raise the trigger delay above
1346
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
        Algorithms:
                           None.
1357
1358
        Data Structures:
                           None.
1359
1360
        Global Variables: delay - incremented if not already the maximum value.
1361
1362
        Author:
                            Glen George
                           Mar. 8, 1994
        Last Modified:
1363
1364
1365
1366
    void trg_delay_up()
1367
1368
1369
         /* variables */
1370
           /* none */
1371
1372
1373
1374
         /* increase the trigger delay, if not already the maximum */
         if (delay < MAX DELAY)
1375
             delay++;
1376
1377
1378
         /* tell the hardware the new trigger delay */
         set delay(delay);
1379
1380
1381
         /* all done raising the trigger delay - return */
1382
1383
         return;
1384
1385
1386
1387
1388
1389
1390
        adjust trg delay
1391
1392
                           This function adjusts the trigger delay for a new sweep
        Description:
1393
1394
                   rate.
                          The factor to adjust the delay by is determined
1395
                   by looking up the sample rates in the sweep rates array.
                   If the delay goes out of range, due to the adjustment it
1396
1397
                   is reset to the maximum or minimum valid value.
1398
1399
        Arguments:
                            old_sweep (int) - old sweep rate (index into sweep_rates
1400
1401
                             (int) - new sweep rate (index into sweep_rates
                   new sweep
1402
                              array).
1403
        Return Value:
                            None.
1404
1405
        Input:
                            None.
1406
        Output:
                            None.
1407
1408
        Error Handling:
                           None.
1409
                           The delay is multiplied by 10 times the ratio of the
1410
        Algorithms:
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
        Global Variables: delay - adjusted based on passed sweep rates.
1416
1417
                            The updated delay time is not displayed. Since the time
1418
        Known Bugs:
                   is typically only rounded to the new sample rate, this is
1419
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
         /* variables */
1429
1430
           /* none */
1431
1432
1433
         /* multiply by 10 times the ratio of sweep rates */
1434
1435
         delay *= (10 * sweep_rates[new_sweep].sample_rate) / sweep_rates[old_sweep].sample_rate;
         /* now divide the factor of 10 back out */
1436
         delay /= 10;
1437
1438
         /* make sure delay is not out of range */
1439
1440
         if (delay > MAX_DELAY)
1441
              /* delay is too large - set to maximum */
             delay = MAX DELAY;
1442
         if (delay < MIN_DELAY)</pre>
1443
1444
              /* delay is too small - set to minimum */
         delay = MIN DELAY;
1445
1446
1447
1448
         /* tell the hardware the new trigger delay */
1449
         set_delay(delay);
1450
1451
         /* all done adjusting the trigger delay - return */
1452
1453
         return:
1454
1455
1456
1457
1458
1459
1460
1461
        display trg delay
1462
        Description:
                            This function displays the current trigger delay at the
1463
1464
                           passed position, in the passed style.
1465
                            x pos (int) - x position (in character cells) at which to
1466
        Arguments:
                         display the trigger delay.
1467
                   y_pos (int) - y position (in character cells) at which to
1468
                         display the trigger delay.
1469
1470
                   style (int) - style with which to display the trigger
                              delay.
1471
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 */
                                              "; /* string containing the trigger delay */
1493
         char
                    delay_str[] = "
                                         /* adjustment to get to microseconds */
1494
         long int
                    units_adj;
1495
         long int
                    d:
                                                 /* delay in appropriate units */
1496
1497
         float
                    temp_d;
                                                 /* delay in float to avoid overflows */
1498
         /st compute the delay in the appropriate units st/
1499
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 */
             /* first compute with float to avoid overflow */
1503
             temp_d = delay * (1000000000L / sweep_rates[sweep].sample_rate);
1504
1505
1506
         /* now convert to int */
         d = (int) temp_d;
1507
1508
         /* need to divide by 1000 to get to microseconds */
1509
         units_adj = 1000;
1510
1511
             e {
/* slow sweep rate, don't have to worry about overflow */
1512
             d = delay * (1000000L / sweep rates[sweep].sample rate);
1513
         /* already in microseconds, so adjustment is 1 */
1514
1515
         units_adj = 1;
1516
1517
         /* convert it to the string (leave first character blank) */
1518
1519
         cvt_num_field(d, &delay_str[1]);
1520
         /* add in the units */
1521
         if (((d / units_adj) < 1000) && ((d / units_adj) > -1000) && (units_adj == 1000)) {
1522
1523
             /* delay is in microseconds */
         delay_str[7] = '\004';
1524
         delay_str[8] = 's';
1525
1526
         else if (((d / units adj) < 1000000) && ((d / units adj) > -1000000)) {
1527
1528
            /* delay is in milliseconds */
         delay_str[7] = 'm';
1529
         delay_str[8] = 's';
1530
1531
         else if (((d / units_adj) < 1000000000) && ((d / units_adj) > -1000000000)) {
1532
1533
             /* delay is in seconds */
1534
         delay_str[7] = 's';
         delay_str[8] = ' ';
1535
1536
1537

   {
   /* delay is in kiloseconds */

         else
1538
1539
         delay str[7] = 'k';
         delay_str[8] = 's';
1540
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
        cvt num field
1557
1558
                           This function converts the passed number (numeric field
1559
        Description:
                           value) to a string and returns that in the passed string
1560
1561
                  reference. The number may be signed, and a sign (+ or -)
1562
                  is always generated. The number is assumed to have three
                  digits to the right of the decimal point. Only the four
1563
1564
                  most significant digits of the number are displayed and
1565
                  the decimal point is shifted appropriately. (Four digits
                  are always generated by the function).
1566
1567
                           n (long int) - numeric field value to convert.
1568
        Arguments:
                  s (char *) - pointer to string in which to return the
1569
                              converted field value.
1570
1571
        Return Value:
                           None.
1572
1573
        Input:
                           None.
        Output:
1574
                           None.
1575
```

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

1651 | } 1652 | }

```
****************
  /*
2
3
  /*
                                     MENUACT.H
4
5
  /*
                                                                                */
                               Menu Action Functions
  /*
                                    Include File
6
   /*
                            Digital Oscilloscope Project
7
   /*
                                      EE/CS 52
   /*
8
           ****************
9
10
11
      This file contains the constants and function prototypes for the functions
12
      which carry out menu actions and display and initialize menu settings for
13
      the Digital Oscilloscope project (the functions are defined in menuact.c).
14
15
16
17
      Revision History:
         3/8/94
                  Glen George
                                    Initial revision.
18
         3/13/94
19
                 Glen George
                                    Updated comments.
                                    Changed definition of enum scale type (was
20
         3/13/94 Glen George
                        enum scale_status).
21
                                    Changed MAX TRG LEVEL SET (maximum trigger
22
         3/10/95 Glen George
23
                        level) to 127 to match specification.
24
         3/17/97 Glen George
                                    Updated comments.
         5/3/06
                  Glen George
                                    Updated comments.
25
         5/9/06
                  Glen George
                                    Added a new mode (AUTO TRIGGER) and a new
26
                                    scale (SCALE GRID).
2.7
28
         5/9/06
                  Glen George
                                    Added menu functions for mode and scale to
                                    move up and down a list instead of just
29
30
                    toggling the selection.
         5/9/06
31
                  Glen George
                                    Added declaration for the accessor to the
                                    current trigger mode (get_trigger_mode).
32
33
   */
34
35
36
37
   #ifndef
             MENUACT H
       #define __MENUACT H
38
39
40
   /* library include files */
41
     /* none */
42
43
   /* local include files */
44
45
   #include "interfac.h"
   #include "lcdout.h"
46
47
48
49
50
   /* constants */
51
52
53
   /* min and max trigger level settings */
   #define MIN TRG LEVEL SET
54
55
   #define MAX TRG LEVEL SET
56
   /* number of different sweep rates */
57
   #define NO_SWEEP_RATES
                               (sizeof(sweep_rates) / sizeof(struct sweep_info))
58
59
60
61
62
   /* structures, unions, and typedefs */
63
64
65
   /* types of triggering modes */
                                               /* normal triggering */
   enum trigger_type { NORMAL_TRIGGER,
66
67
                 AUTO TRIGGER,
                                 /* automatic triggering */
                 ONESHOT TRIGGER
                                      /* one-shot triggering */
68
69
              };
70
   /* types of displayed scales */
71
   enum scale_type { SCALE_NONE, SCALE_AXES, /
                                          /* no scale is displayed */
72
                               /* scale is a set of axes */
73
                                   /* scale is a grid */
                 SCALE_GRID
74
75
              };
```

```
76
    /* types of trigger slopes */
 77
    78
                   SLOPE_NEGATIVE
79
 80
 81
    /* sweep rate information */
 82
    struct sweep_info { long int
                                           sample_rate; /* sample rate */
 83
                  const char *s;
                                               /* sweep rate string */
 84
 85
 86
 87
 88
    /* function declarations */
90
91
 92
    /* menu option actions */
    void no_menu_action(void);
                                       /* no action to perform */
93
                                      /* change to the "next" trigger mode */
/* change to the "previous" trigger mode */
 94
    void
          mode down(void);
95
    void mode up(void);
                                      /* change to the "next" scale type */
          scale_down(void);
    void
96
                                      /* change to the "previous" scale type */
97
    void
          scale_up(void);
    void sweep_down(void);
                                      /* decrease the sweep rate */
98
                                      /* increase the sweep rate */
99
    void sweep_up(void);
          trg level down(void);
                                      /* decrease the trigger level */
100
          trg_level_up(void);
                                      /* increase the trigger level */
    void
101
    void trg_slope_toggle(void); /* toggle the trigger slope */
102
103
          trg_delay_down(void);
                                      /* decrease the trigger delay */
    void trg_delay_up(void);
                                      /* increase the trigger delay */
104
105
106
    /* option accessor routines */
    enum trigger_type get_trigger_mode(void); /* get the current trigger mode */
107
108
109
    /* option initialization routines */
    void set_trigger_mode(enum trigger_type); /* set the trigger mode */
110
    void
           set scale(enum scale type);
                                                     /* set the scale type */
111
    void set_sweep(int);
void set_trg_level(int);
                                                 /* set the sweep rate */
112
                                                /* set the trigger level */
113
    void set trg slope(enum slope type);
                                                   /* set the trigger slope */
114
    void set_trg_delay(long int);
                                                 /* set the tigger delay */
115
116
    /* option display routines */
117
    void no_display(int, int, int);
                                             /* no option setting to display */
118
          display_mode(int, int, int);
                                                 /* display trigger mode */
119
    void
                                                 /* display the scale type */
          display_scale(int, int, int);
120
    void display_sweep(int, int, int); /* display the sweep rate */
void display_trg_level(int, int, int); /* display the trigger level */
void display_trg_slope(int, int, int); /* display the trigger slope */
void display_trg_delay(int, int, int); /* display the tigger delay */
121
122
124
125
126
    #endif
127
128
```

```
/*
2
   /*
3
                                      SCOPEDEF.H
                                 General Definitions
                                                                                   */
4
5
   /*
                                     Include File
6
   /*
                             Digital Oscilloscope Project
   /*
7
                                        EE/CS 52
8
   /*
           *****************
9
10
11
      This file contains the general definitions for the Digital Oscilloscope
12
13
      project. This includes constant and structure definitions along with the
      function declarations for the assembly language functions.
14
15
16
17
      Revision History:
         3/8/94
                   Glen George
                                      Initial revision.
18
19
         3/13/94
                   Glen George
                                      Updated comments.
         3/17/97
                                      Removed KEYCODE UNUSED (no longer used).
20
                   Glen George
         5/3/06
                                      Added conditional definitions for handling
                  Glen George
21
22
                                      different architectures.
                                      Updated declaration of start_sample() to
         5/9/06
                   Glen George
23
2.4
                                      match the new specification.
                                      Added check for __nios__ definition to also indicate the compilation is for an Altera
         5/27/08 Glen George
25
26
                         NIOS CPU.
2.7
28
         6/03/14 Santiago Navonne Added cursor text area, and NO TRACE value.
   */
29
30
31
32
33
   #ifndef
             SCOPEDEF H
       #define __SCOPEDEF_H_
34
35
36
   /* library include files */
  /* none */
37
38
39
   /* local include files */
40
   #include "interfac.h"
41
   #include "lcdout.h"
43
44
45
46
47
   /* constants */
48
   /* general constants */
49
   #define FALSE
#define TRUE
50
                         0
                         !FALSE
51
   #define NULL
                         (void *) 0
52
53
   /* display size (in characters) */
54
   #define LCD_WIDTH (SIZE_X / HORIZ_SIZE)
#define LCD_HEIGHT (SIZE_Y / VERT_SIZE)
55
56
57
   /* cursor area */
58
   #define CURSOR STR X
59
   #define CURSOR_STR_Y
                               5
60
61
   #define CURSOR_STR_W
                               100
62
   #define CURSOR STR H
63
64
65
   /* macros */
66
67
   /* let __nios__ also mean a NIOS compilation */
#ifdef __nios__
68
69
           __nios_
     #define NIOS
                            /* use the standard NIOS defintion */
70
71
   #endif
72
73
  /* add the definitions necessary for the Altera NIOS chip */
  #ifdef NIOS
74
     #define FLAT MEMORY /* use the flat memory model */
75
```

```
76 #endif
77
78
   /* if a flat memory model don't need far pointers */
79
   #ifdef FLAT MEMORY
80
    #define far
81
   #endif
82
83
84
85
86
   /* structures, unions, and typedefs */
87
88
   /* program states */
89
   enum status { MENU_ON, /* menu is displayed with the cursor in it */
90
         MENU_OFF, /* menu is not displayed - no cursor */
NUM_STATES /* number of states */
91
92
93
94
   95
                                              */
96
97
              KEYCODE DOWN,
                              /* <Down>
98
                                          */
          KEYCODE_LEFT, /* <Left> */
KEYCODE_RIGHT, /* <Right> */
KEYCODE_ILLEGAL, /* other keys */
NUM_KEYCODES /* number of key codes */
99
100
101
102
103
               };
104
105
106
107
   /* function declarations */
108
109
   /* keypad functions */
110
   111
112
113
114
   /* display functions */
   115
116
117
   /* sampling parameter functions */
118
   119
120
   void set_delay(long int);
                              /* set the trigger delay time */
121
122
   /* sampling functions */
123
   124
125
126
127
128
   #endif
129
```

```
*****************
3
   /*
                                         TRACUTIL
4
5
  /*
                                                                                      */
                                Trace Utility Functions
   /*
                              Digital Oscilloscope Project
6
   /*
                                         EE/CS 52
7
       *****************
8
9
10
      This file contains the utility functions for handling traces (capturing
11
      and displaying data) for the Digital Oscilloscope project. The functions
12
13
      included are:
         clear saved areas - clear all the save areas
14
         do trace
                              - start a trace
15
16
         init trace
                              - initialize the trace routines
17
         plot trace

    plot a trace (sampled data)

         restore_menu_trace - restore the saved area under the menus
18
         restore_trace - restore the saved area of a trace set_display_scale - set the type of displayed scale (and display it)
19
20
                             - set the triggering mode
          set_mode
21
                             - determine an area of a trace to save
22
          set save area
                             - set the number of samples in a trace
         set trace size
23
24
         trace_done
                             - inform this module that a trace has been completed
                              - determine if system is ready to start another trace
25
          trace rdy
         trace_rearm
                              - re-enable tracing (in one-shot triggering mode)
26
2.7
28
      The local functions included are:
29
30
31
      The locally global variable definitions included are:
         cur scale

    current scale type

32
          sample_size - the size of the sample for the trace
33
          sampling
                       - currently doing a sample
34
                      - saved trace under a specified area
          saved_area
35
36
          saved_axis_x - saved trace under the x lines (axes or grid)
         saved_axis_y - saved trace under the y lines (axes or grid)
saved_menu - saved trace under the menu
37
38
          saved_pos_x - starting position (x coorindate) of area to save
39
         saved_pos_y - starting position (y coorindate) of area to save
saved_end_x - ending position (x coorindate) of area to save
40
41
          saved end y - ending position (y coorindate) of area to save
42
         trace status - whether or not ready to start another trace
43
44
45
      Revision History
46
47
          3/8/94
                   Glen George
                                       Initial revision.
                                      Updated comments.
48
          3/13/94
                   Glen George
                                       Fixed inversion of signal in plot_trace.
          3/13/94 Glen George
49
          3/13/94 Glen George
                                      Added sampling flag and changed the functions
50
                              init_trace, do_trace and trace_done to update
51
                     the flag. Also the function trace_rdy now uses it. The function set_mode was updated
52
53
                     to always say a trace is ready for normal
54
55
                     triggering.
                                       Fixed bug in trace restoring due to operator
56
                   Glen George
                         misuse (&& instead of &) in the functions
57
58
                     set_axes, restore_menu_trace, and
                     restore trace.
59
         3/13/94
                   Glen George
                                       Fixed bug in trace restoring due to the clear
60
61
                          function (clear saved areas) not clearing all
62
                     of the menu area.
          3/13/94
                   Glen George
                                       Fixed comparison bug when saving traces in
63
                         plot trace.
64
                                      Changed name of set_axes to set_display_scale
65
          3/13/94 Glen George
                         and the name of axes_state to cur_scale to
66
67
                     more accurately reflect the function/variable
                     use (especially if add scale display types).
68
          3/17/97
                   Glen George
                                       Updated comments.
69
                                       Changed set_display_scale to use plot_hline
70
          3/17/97
                   Glen George
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
                                      definition of start_sample().
Removed normal_trg variable, its use is now
74
75
         5/9/06
                   Glen George
```

```
76
                                         handled by the get_trigger_mode() accessor.
77
           5/9/06
                    Glen George
                                         Added tick marks to the axes display.
           5/9/06
                    Glen George
 78
                                         Added ability to display a grid.
                                         Added is_sampling() function to be able to
           5/27/08
                    Glen George
79
80
                                    tell if the system is currently taking a
81
                       sample.
          5/27/08 Glen George
                                         Changed set mode() to always turn off the
82
                                    sampling flag so samples with the old mode
 83
                                         setting are ignored.
84
85
           6/3/08
                    Glen George
                                         Fixed problems with non-power of 2 display
 86
                       sizes not working.
           6/3/14
                                        Changed UI display colors; changed plot_trace
87
                     Santiago Navonne
                                         to clear just trace instead of whole display.
88
    */
 89
90
91
92
    /* library include files */
  /* none */
93
94
95
    /* local include files */
96
97
    #include
              "scopedef.h"
               "lcdout.h"
   #include
98
               "menu.h"
99
    #include
    #include
               "menuact.h"
100
              "tracutil.h"
    #include
101
102
103
104
105
    /* locally global variables */
106
107
    static int
                trace status;
                                   /* ready to start another trace */
108
109
                                        /* currently sampling data */
    static int
                 sampling;
110
111
                 sample size;
                                   /* number of data points in a sample */
112
    static int
113
    static int old sample[SIZE X]; /* sample currently being displayed */
114
115
    static enum scale type cur scale; /* current display scale type */
116
117
    /* traces (sampled data) saved under the axes */
118
    static unsigned char saved_axis_x[2 * Y_TICK_CNT + 1][PLOT_SIZE_X/8]; /* saved trace under x lines */static unsigned char saved_axis_y[2 * X_TICK_CNT + 1][PLOT_SIZE_Y/8]; /* saved trace under y lines */
119
120
121
122
    /* traces (sampled data) saved under the menu */
    static unsigned char saved menu[MENU SIZE Y][(MENU SIZE X + 7)/8];
123
124
    /* traces (sampled data) saved under any area */
125
    static unsigned char saved_area[SAVE_SIZE_Y][SAVE_SIZE_X/8]; /* saved trace under any area */
126
    static int
                                         /* starting x position of saved area */
127
                        saved_pos_x;
                        saved_pos_y;
                                          /* starting y position of saved area */
128
    static int
    static int
                        saved end x;
                                         /* ending x position of saved area */
129
                                          /* ending y position of saved area */
130
    static int
                        saved end y;
131
132
133
134
135
136
       init_trace
137
                           This function initializes all of the locally global
138
       Description:
139
                           variables used by these routines. The saved areas are
140
                  set to non-existant with cleared saved data. Normal
                  normal triggering is set, the system is ready for a
141
142
                  trace, the scale is turned off and the sample size is set
                  to the screen size.
143
144
                           None.
145
       Arguments:
       Return Value:
                           None.
146
147
148
       Input:
                           None.
       Output:
                           None.
149
150
```

```
151
       Error Handling:
                           None.
152
153
       Algorithms:
                           None.
       Data Structures: None.
154
155
       Global Variables: trace status - set to TRUE.
156
                                - set to FALSE.
157
                  sampling
                                - set to SCALE_NONE (no displayed scale).
158
                  cur_scale
                  sample_size - set to screen size (SIZE_X).
159
                  saved_axis_x - cleared.
160
                  saved_axis_y - cleared.
161
                  saved_menu
162
                                - cleared.
                                - cleared.
                  saved area
163
                  saved pos x
                                - set to off-screen.
164
                  saved_pos_y
                                - set to off-screen.
165
166
                  saved end x
                                - set to off-screen.
                               - set to off-screen.
167
                  saved_end_y
168
169
       Author:
                           Glen George
       Last Modified:
                          May 9, 2006
170
171
172
173
174
    void init_trace()
175
        /* variables */
176
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 */
        cur_scale = SCALE_NONE;
190
191
        /* sample size is the screen size */
192
        sample size = SIZE X;
193
194
195
        /* clear save areas */
196
197
        clear_saved_areas();
198
        /* also clear the general saved area location variables (off-screen) */
199
200
        saved pos x = SIZE X + 1;
        saved_pos_y = SIZE_Y + 1;
201
        saved_end_x = SIZE_X + 1;
202
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
                           based on the passed value (one of the possible enumerated
218
219
                            The triggering mode is used to determine when
                  the system is ready for another trace. The sampling flag
220
221
                           is also reset so a new sample will be started (if that is
222
                           appropriate).
223
                           trigger_mode (enum trigger_type) - the mode with which to
       Arguments:
224
225
                                  set the triggering.
```

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

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

```
376
377
    void trace_done()
378
         /* variables */
379
380
           /* none */
381
382
383
         /* done with a trace - if retriggering, ready for another one */
if (get_trigger_mode() != ONESHOT_TRIGGER)
384
385
386
              /* in a retriggering mode - set trace status to TRUE (ready) */
387
         trace_status = TRUE;
388
389
         /* no longer sampling data */
         sampling = FALSE;
390
391
392
         /* done so return */
393
394
         return;
395
396
397
398
399
400
401
       trace_rearm
402
403
                            This function is called to rearm the trace. It sets the
       Description:
404
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.
       Output:
412
                            None.
413
414
       Error Handling:
                            None.
415
       Algorithms:
416
                            None.
417
       Data Structures:
                            None.
418
       Global Variables: trace_status - set to TRUE.
419
420
       Author:
                            Glen George
421
422
       Last Modified:
                            Mar. 8, 1994
423
    */
424
425
    void trace_rearm()
426
427
    {
428
         /* variables */
           /* none */
429
430
431
432
         /* rearm the trace - set status to ready (TRUE) */
433
         trace status = TRUE;
434
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
                            passed value. This is used to scale the data when
449
450
                   plotting a trace.
```

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

```
527
528
        /* whenever change scale type, need to clear out previous scale */
529
        /* unnecessary if going to SCALE_GRID or from SCALE_NONE or not changing the scale */
530
531
        if ((scale != SCALE GRID) && (cur scale != SCALE NONE) && (scale != cur scale)) {
532
            /* need to restore the trace under the lines (tick, grid, or axis) */
533
534
535
        /* go through all points on horizontal lines */
        for (j = -Y TICK CNT; j \le Y TICK CNT; j++)
536
537
            /* get y position of the line */
538
            p = X AXIS POS + j * Y TICK SIZE;
            /* make sure it is in range */
540
541
            if (p >= PLOT_SIZE_Y)
542
                p = PLOT_SIZE_Y - 1;
            if (p < 0)
543
                p = 0;
544
545
            /* look at entire horizontal line */
546
547
            for (i = 0; i < PLOT_SIZE_X; i++) {</pre>
                 /* check if this point is on or off (need to look at bits) */
548
549
            if ((saved_axis_x[j + Y_TICK_CNT][i / 8] & (0x80 >> (i % 8))) == 0)
                 /* saved pixel is off */
550
                plot_pixel(i, p, PIXEL_CLEAR);
551
552
            else
553
                 /* saved pixel is on */
                plot_pixel(i, p, PIXEL_TRACE);
554
555
556
        }
557
558
        /* go through all points on vertical lines */
        for (j = -X TICK CNT; j \le X TICK CNT; j++) {
559
560
561
            /* get x position of the line */
562
            p = Y_AXIS_POS + j * X_TICK_SIZE;
            /* make sure it is in range */
563
            if (p >= PLOT SIZE X)
564
                p = PLOT_SIZE_X - 1;
565
566
            if (p < 0)
                p = 0;
567
568
569
            /* look at entire vertical line */
            for (i = 0; i < PLOT_SIZE_Y; i++) {</pre>
570
                 /* check if this point is on or off (need to look at bits) */
571
572
               ((saved_axis_y[j + X_TICK_CNT][i / 8] \& (0x80 >> (i % 8))) == 0)
                 /* saved pixel is off */
573
574
                plot_pixel(p, i, PIXEL_CLEAR);
575
            else
                /* saved pixel is on */
576
                 plot_pixel(p, i, PIXEL_TRACE);
577
578
            }
579
        }
580
        }
581
582
583
        /* now handle the scale type appropriately */
584
        switch (scale)
585
586
            case SCALE AXES:
                                  /* axes for the scale */
                                  /* grid for the scale */
587
            case SCALE GRID:
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 */
                     p = X_AXIS_POS + i * Y_TICK_SIZE;
593
                     /* make sure it is in range */
594
                     if (p >= PLOT_SIZE Y)
                         p = PLOT_SIZE_Y - 1;
596
597
                        (p < 0)
598
                         p = 0;
599
                     /* should we draw a grid, an axis, or a tick mark */
600
```

526

```
601
                      if (scale == SCALE GRID)
602
                          /* drawing a grid line */
                              plot_hline(X_GRID_START, p, (X_GRID_END - X_GRID_START));
603
                      else if (i == 0)
604
605
                          /* drawing the x axis */
                               plot hline(X AXIS START, p, (X AXIS END - X AXIS START));
606
607
                      else
608
                          /* must be drawing a tick mark */
609
                               plot hline((Y AXIS POS - (TICK LEN / 2)), p, TICK LEN);
610
611
                      /* draw y lines (grid or tick marks) */
for (i = -X_TICK_CNT; i <= X_TICK_CNT; i++) {</pre>
612
613
615
                      /* get x position of the line */
                      p = Y_AXIS_POS + i * X_TICK SIZE;
616
                      /* make sure it is in range */
617
                      if (p >= PLOT_SIZE_X)
618
                          p = PLOT_SIZE_X - 1;
619
                          if (p < \overline{0})
620
                          p = 0;
621
622
623
                      /* should we draw a grid, an axis, or a tick mark */
624
                      if (scale == SCALE_GRID)
                          /* drawing a grid line */
625
                              plot_vline(p, Y_GRID_START, (Y_GRID_END - Y_GRID_START));
626
                      else if (i == 0)
627
628
                          /* drawing the y axis */
                              plot_vline(p, Y_AXIS_START, (Y_AXIS_END - Y AXIS START));
629
630
                      else
631
                          /* must be drawing a tick mark */
                               plot_vline(p, (X_AXIS_POS - (TICK_LEN / 2)), TICK_LEN);
632
633
                      }
634
                      /* done with the axes */
635
636
                      break;
637
             case SCALE NONE:
                                   /* there is no scale */
638
639
                      /* already restored plot so nothing to do */
                      break;
640
641
642
        }
643
644
645
        /* now remember the new (now current) scale type */
        cur scale = scale;
646
647
648
        /* scale is taken care of, return */
649
650
        return;
651
652
653
654
655
656
657
658
       clear_saved_areas
659
                           This function clears all the saved areas (for saving the
       Description:
660
661
                           trace under the axes, menus, and general areas).
662
       Arguments:
663
                           None.
664
       Return Value:
                           None.
665
                           None.
666
       Input:
667
       Output:
                           None.
668
       Error Handling:
669
                           None.
670
671
       Algorithms:
                           None.
672
       Data Structures:
                           None.
673
       Global Variables: saved_axis_x - cleared.
674
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
         /* variables */
686
         int i;
int j;
                       /* loop indices */
687
688
689
690
691
         /* clear x-axis and y-axis save areas */
for (j = 0; j <= (2 * Y_TICK_CNT); j++)
    for (i = 0; i < (SIZE_X / 8); i++)</pre>
692
693
694
         saved_axis_x[j][i] = 0;
for (j = 0; j <= (2 * X_TICK_CNT); j++)
for (i = 0; i < (SIZE_Y / 8); i++)</pre>
695
696
697
                   saved_axis_y[j][i] = 0;
698
699
700
          /* clear the menu save ares */
         for (i = 0; i < MENU_SIZE_Y; i++)</pre>
701
              for (j = 0; j < ((MENU_SIZE_X + 7) / 8); j++)
702
703
              saved menu[i][j] = 0;
704
705
         /* clear general save area */
         for (i = 0; i < SAVE_SIZE_Y; i++)
    for (j = 0; j < (SAVE_SIZE_X / 8); j++)</pre>
706
707
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.
        Output:
                              The trace under the menu is restored to the LCD screen.
729
730
731
        Error Handling:
732
733
        Algorithms:
                               None.
        Data Structures:
734
735
736
        Global Variables: saved menu - used to restore trace data under the menu.
737
        Author:
                               Glen George
738
739
        Last Modified:
                              June 03, 2014
740
741
742
    void restore menu trace()
743
744
          /* variables */
745
746
         int bit_position;
                                 /* position of bit to restore (in saved data) */
                                  /* offset (in bytes) of bit within saved row */
747
         int bit_offset;
748
                        /* loop indices */
749
         int x;
         int y;
750
```

```
751
752
753
        /* loop, restoring the trace under the menu */
754
755
        for (y = MENU_UL_Y; y < (MENU_UL_Y + MENU_SIZE_Y); y++) {
756
757
            /* starting a row - initialize bit position */
                               /* start at high-order bit in the byte */
758
        bit_position = 0x80;
                             //* first byte of the row */
759
        bit offset = 0;
760
            for (x = MENU UL X; x < (MENU UL X + MENU SIZE X); x++)
761
762
            /* check if this point is on or off (need to look at bits) */
763
            if ((saved_menu[y - MENU_UL_Y][bit_offset] & bit_position) == 0)
764
                 /* saved pixel is off */
765
766
            plot_pixel(x, y, PIXEL_CLEAR);
767
            else
                 /* saved pixel is on */
768
769
            plot_pixel(x, y, PIXEL_TRACE);
770
            /* move to the next bit position */
771
772
            bit_position >>= 1;
773
            /* check if moving to next byte */
774
            if (bit_position == 0) {
                 /* now on high bit of next byte */
775
            bit_position = 0x80;
776
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
                 currently saved.
796
797
798
       Arguments:
                          pos x (int) - x position of upper left corner of the
799
                         saved area.
800
                               - y position of upper left corner of the
                  pos y (int)
                             saved area.
801
                  size_x (int) - horizontal size of the saved area.
802
803
                  size_y (int) - vertical size of the saved area.
       Return Value:
804
                          None.
805
806
       Input:
                          None.
807
       Output:
                          None.
808
       Error Handling:
809
                          None.
810
811
       Algorithms:
                          None.
812
       Data Structures:
                          None.
813
814
       Global Variables: saved area - cleared.
815
                  saved pos x - set to passed value.
                  saved_pos_y - set to passed value.
816
817
                  saved_end_x - computed from passed values.
                  saved end y - computed from passed values.
818
819
820
                          Glen George
821
       Last Modified:
                          Mar. 8, 1994
822
823
824
   void set_save_area(int pos_x, int pos_y, int size_x, int size_y)
825
```

```
826
        /* variables */
827
                 /* loop indices */
828
        int x;
829
        int y;
830
831
832
833
        /st just setup all the locally global variables from the passed values st/
834
        saved_pos_x = pos_x;
835
        saved_pos_y = pos_y;
        saved end x = pos x + size x;
836
        saved_end_y = pos_y + size_y;
837
838
839
        /* clear the save area */
840
        for (y = 0; y < SAVE_SIZE_Y; y++)
841
            for (x = 0; x < (SAVE\_SIZE_X / 8); x++) {
842
            saved_area[y][x] = 0;
843
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
                          area. (The area was previously set and the trace was
860
861
                  previously saved.)
862
       Arguments:
                          None.
863
864
       Return Value:
                          None.
865
866
       Input:
                          None.
                          The trace under the saved ares is restored to the LCD.
867
       Output:
868
869
       Error Handling:
                          None.
870
       Algorithms:
871
                          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.
                  saved_pos_y - gives starting y position of saved area.
876
                  saved_end_x - gives ending x position of saved area.
877
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
    void restore_trace()
885
886
887
        /* variables */
        int bit_position; /* position of bit to restore (in saved data) */
888
                              /* offset (in bytes) of bit within saved row */
889
        int bit offset;
890
                     /* loop indices */
891
        int x;
892
        int y;
893
894
895
896
        /* loop, restoring the saved trace */
897
        for (y = saved_pos_y; y < saved_end_y; y++) {</pre>
898
             /* starting a row - initialize bit position */
899
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++)</pre>
904
905
             /* check if this point is on or off (need to look at bits) */
             if ((saved_area[y - saved_pos_y][bit_offset] & bit_position) == 0)
   /* saved pixel is off */
906
907
908
             plot_pixel(x, y, PIXEL_CLEAR);
             else
909
                 /* saved pixel is on */
910
             plot pixel(x, y, PIXEL TRACE);
911
912
             /* move to the next bit position */
913
             bit position >>= 1;
             /* check if moving to next byte */
915
916
             if (bit_position == 0)
                 /* now on high bit of next byte */
917
             bit_position = 0x80;
918
919
             bit_offset++;
920
921
             }
922
        }
923
924
        /* restored the saved area - return */
925
926
        return:
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
                  ready flag (trace_status) to FALSE and the sampling flag
938
939
                  (sampling) to TRUE.
940
941
       Arguments:
                           None.
       Return Value:
942
                           None.
943
944
       Input:
                           None.
945
       Output:
                           None.
946
947
       Error Handling:
                           None.
948
949
       Algorithms:
                           None.
950
       Data Structures:
                           None.
951
       Global Variables: trace_status - set to FALSE (not ready for another trace).
952
953
                                 - set to TRUE (doing a sample now).
954
955
       Author:
                           Glen George
956
       Last Modified:
                           Mar. 13, 1994
957
    */
958
959
    void do_trace()
960
961
962
        /* variables */
          /* none */
963
964
965
966
967
        /* start up the trace */
        /* indicate whether using automatic triggering or not */
968
        start_sample(get_trigger_mode() == AUTO_TRIGGER);
969
970
        /* now not ready for another trace (currently doing one) */
971
972
        trace_status = FALSE;
973
        /* and are currently sampling data */
974
975
        sampling = TRUE;
```

```
976
977
978
         /* trace is going, return */
979
         return:
980
981
     }
982
983
984
985
        plot_trace
986
                            This function plots the passed trace. The trace is
987
        Description:
                            assumed to contain sample size points of sampled data.
988
                            Any points falling within any of the save areas are also
989
                            saved by this routine. The data is also scaled to be
990
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
                            The sample is plotted on the screen.
        Output:
999
        Error Handling:
1000
1001
                            If there are more sample points than screen width the
1002
        Algorithms:
1003
                   sample is plotted with multiple points per horizontal
                   position.
1004
1005
        Data Structures:
                           None.
1006
        Global Variables: cur_scale
1007
                                          - determines type of scale to plot.
1008
                   sample_size - determines size of passed sample.
1009
                   saved axis x - stores trace under x-axis.
                   saved_axis_y - stores trace under y-axis.
saved_menu - stores trace under the menu.
1010
1011
1012
                   saved_area
                                 - stores trace under the saved area.
                   saved_pos_x - determines location of saved area.
1013
1014
                                - determines location of saved area.
                   saved pos y
1015
                   saved end x
                                - determines location of saved area.
                                - determines location of saved area.
1016
                   saved end y
1017
                            Glen George
1018
        Author:
        Last Modified:
                            June 03, 2014
1019
1020
1021
1022
1023
     void plot trace(unsigned char *sample)
1024
1025
         /* variables */
         int x = 0;
                                    /* current x position to plot */
1026
         int x_pos = (PLOT_SIZE_X / 2); /* "fine" x position for multiple point plotting */
1027
1028
         int y;
                               /* y position of point to plot */
1029
1030
1031
                                                 /* an x or y coordinate */
         int
              p;
1032
1033
         int
              i;
                               /* loop indices */
1034
         int
              j;
1035
1036
         /* clear the saved areas too */
1037
         clear_saved_areas();
1038
1039
1040
         /* re-display the menu (if it was on) */
         refresh_menu();
1041
1042
1043
         /* plot the sample */
1044
         for (i = 0; i < sample_size; i++)</pre>
1045
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 */
         plot_pixel(x, y, PIXEL_TRACE);
1054
1055
1056
         /* and save new value */
1057
         old sample[i] = y;
1058
1059
1060
         /* check if the point is in a save area */
1061
         /* check if in the menu area */
1062
         if ((x \ge MENU UL X) \&\& (x < (MENU UL X + MENU SIZE X)) \&\&
1063
              (y \ge MENU UL Y) \&\& (y < (MENU UL Y + MENU SIZE Y)))
1064
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
         /* check if in the saved area */
1068
1069
         if ((x \ge saved_pos_x) \& (x \le saved_end_x) \& (y \ge saved_pos_y) \& (y \le saved_end_y))
1070
              /* point is in the save area - save it */
              saved\_area[y - saved\_pos\_y][(x - saved\_pos\_x)/8] = (0x80 >> ((x - saved\_pos\_x) % 8));
1071
1072
1073
         /* check if on a grid line */
1074
         /* go through all the horizontal lines */
         for (j = -Y TICK CNT; j \le Y TICK CNT; j++) {
1075
1076
              /* get y position of the line */
1077
1078
              p = X_AXIS_POS + j * Y_TICK_SIZE;
              \frac{1}{1} make sure it is in range */
1079
1080
              if (p >= PLOT_SIZE_Y)
                  p = PLOT_{\overline{S}}IZE_{\overline{Y}} - 1;
1081
              if (p < 0)
1082
                  p = 0;
1083
1084
              /* if the point is on this line, save it */
1085
1086
              if (y == p)
1087
              saved_axis_x[j + Y_TICK_CNT][x / 8] |= (0x80 >> (x % 8));
1088
         }
1089
         /* go through all the vertical lines */
for (j = -X_TICK_CNT; j <= X_TICK_CNT; j++) {</pre>
1090
1091
1092
              /* get x position of the line */
1093
              p = Y_AXIS_POS + j * X_TICK_SIZE;
1094
1095
              /* make sure it is in range */
              if (p >= PLOT_SIZE_X)
1096
1097
                  p = PLOT_SIZE_X - 1;
1098
              if (p < 0)
                  p = 0;
1099
1100
              /* if the point is on this line, save it */
1101
              if (x == p)
1102
1103
              saved_axis_y[j + X_TICK_CNT][y / 8] = (0x80 >> (y % 8));
1104
         }
1105
1106
         /* update x position */
1107
         x_pos += PLOT_SIZE_X;
1108
1109
            check if at next horizontal position */
         if (x_pos >= sample_size) {
1110
1111
              /\overline{*} at next position - update positions */
1112
              x++;
1113
              x_pos -= sample_size;
1114
         }
1115
1116
1117
         /* finally, output the scale if need be */
1118
1119
         set_display_scale(cur_scale);
1120
1121
         /* done with plot, return */
1122
1123
         return;
1124
1125 | }
```

```
*************
   /*
3
                                        TRACUTIL.H
                                                                                       */
                                 Trace Utility Functions
4
   /*
                                       Include File
   /*
                              Digital Oscilloscope Project
6
   /*
                                         EE/CS 52
7
8
   /*
            ***************
9
10
11
      This file contains the constants and function prototypes for the trace
12
      utility functions (defined in tracutil.c) for the Digital Oscilloscope
13
14
      project.
15
16
17
      Revision History:
          3/8/94
                   Glen George
                                       Initial revision.
18
19
          3/13/94
                   Glen George
                                       Updated comments.
20
          3/13/94 Glen George
                                       Changed name of set axes function to
                          set_display_scale.
21
22
          5/9/06
                   Glen George
                                       Added the constants for grids and tick marks.
         5/27/08 Glen George
                                       Added is_sampling() function to be able to
23
2.4
                                   tell if the system is currently taking a
25
                      sample.
         6/3/08
                   Glen George
                                       Removed Y SCALE FACTOR - no longer used to
26
                                   fix problems with non-power of 2 display
2.7
28
                      sizes.
29
30
31
32
33
   #ifndef
              TRACUTIL H
       #define __TRACUTIL_H_
34
35
36
   /* library include files */
  /* none */
37
38
39
   /* local include files */
40
   #include "interfac.h"
41
   #include "menuact.h"
43
44
45
46
47
   /* constants */
48
   /* plot size */
49
   #define PLOT_SIZE_X
#define PLOT_SIZE_Y
                             SIZE X
                                         /* plot takes entire screen width */
50
                                          /* plot takes entire screen height */
                             SIZE Y
51
52
   /* axes position and size */
53
   #define X AXIS START 0
                                         /* starting x position of x-axis */
54
  #define X_AXIS_END (PLOT_SIZE_X - 1) /* ending x position of x-axis */
#define X_AXIS_POS (PLOT_SIZE_Y / 2) /* y position of x-axis */
#define Y_AXIS_START 0 /* starting y position of y-axis */
55
56
57
   #define Y_AXIS_END (PLOT_SIZE_Y - 1) /* ending y position of y-axis */
#define Y_AXIS_POS (PLOT_SIZE_X / 2) /* x position of y-axis */
58
59
60
61
   /* tick mark and grid constants */
62
   #define TICK LEN 5
                                          /* length of axis tick mark */
   /* tick mark counts are for a single quadrant, thus total number of tick */
63
   /* marks or grids is twice this number */
64
   #define X_TICK_CNT
#define X_TICK_SIZE
65
                                   /* always 5 tick marks on x axis */
                             (PLOT_SIZE_X / (2 * X_TICK_CNT)) /* distance between tick marks */
66
67
   #define Y_TICK_SIZE
                             X_TICK_SIZE /* same size as x */
                             (PLOT_SIZE_Y / (2 * Y_TICK_SIZE)) /* number of y tick marks */
   #define
            Y_TICK_CNT
68
   #define X_GRID_START
                                         /* starting x position of x grid */
69
                             0
                             (PLOT_SIZE_X - 1) /* ending x position of x grid */
   #define X GRID END
70
71
   #define
            Y_GRID_START
                                         /* starting y position of y-axis */
                             0
   #define Y GRID END
72
                             (PLOT_SIZE_Y - 1) /* ending y position of y-axis */
73
  /st maximum size of the save area (in pixels) st/
74
75
   #define SAVE SIZE X
                             120 /* maximum width */
```

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

```
*************
   /*
2
   /*
3
                                        TRIGGER.S
4
5
   /*
                              Data sampling and triggering
                                                                                     */
   /*
                              Digital Oscilloscope Project
6
   /*
                                        EE/CS 52
7
   /*
                                    Santiago Navonne
   /*
8
              *************
9
10
11
      Data sampling and triggering control routines for the EE/CS 52 Digital
12
      Oscilloscope project. Function definitions are included in this file, and
13
      are laid out as follows:
14
       - set sample rate: Configures the sampling rate;
15
       - set trigger: Configures the manual trigger level and slope;
16
       - set delay: Configures the manual trigger delay;
17
       - start_sample: Starts a new data sample with the previously configured
18
                        settings and passed auto-trigger configuration;
19
       - sample done: Checks whether a new data sample set is available, returning
20
                       a pointer to a buffer containing it if there is, or a NULL
21
22
                       pointer if there isn't;
       - sample handler: Handles sampling FIFO full interrupts;
23
24
       - trigger_init: Initializes the environment's shared variables and the
                        triggering logic circuit (resetting it), effectively
25
                        preparing the sampling/triggering interface for use.
26
2.7
28
      Revision History:
29
30
         5/29/14 Santiago Navonne Initial revision.
         6/01/14 Santiago Navonne Minor fixes; updated documentation. 6/11/14 Santiago Navonne Changed division algorithm in set_sample_rate.
31
32
33
   * /
34
   /* Includes */
35
   #include "general.h" /* General assembly constants */
#include "system.h" /* Base addresses */
#include "interfac.h" /* Software interface definitions */
36
37
38
   #include "trigger.h" /* Local constants */
39
40
41
   /* Variables */
42
       .section .data
                               /* No alignment necessary: variables are bytes */
43
                              /* Logical value: whether a sample is pending */
   sample_pending: .byte 0
44
45
   sample: .skip FIFO SIZE
                              /* Sample buffer */
46
47
       .section .text
                               /* Code starts here */
48
49
50
       set sample rate
51
                           This procedure configures the sampling rate of the sampling
       Description:
52
53
                            interface. After execution, the interface will start sampling
                            at the requested rate, rounded up to a multiple of the system
54
                            clock. The return value is how many samples will be acquired,
55
                            which is always the size of the FIFO.
56
                            If an argument of 0 is passed, the function has no effect, and
57
58
                            returns 0. The argument must however by less than or equal to
    *
                            the system clock divided by two; no error checking is performed
59
                           on this.
60
61
62
       Operation:
                           The procedure starts by error checking the value of the argument,
                            simply returning 0 if it is invalid. Then, it computes the
63
64
                            required clock period in system clock periods by dividing the
65
                            system clock frequency by the requested sample rate.
                           Finally, it saves the computed value to the trigger period
66
67
                            register, and pulses the reset bit in the control register to
                            reset the triggering logic. SIZE_X is ultimately moved into
    *
68
                           r2 as constant return value.
69
70
                           samples_per_sec - positive integer indicating the sample rate
71
       Arguments:
                                               in samples per second (r4). The value must
72
73
                                               be less than or equal to the system clock
                                               divided by two.
74
75
```

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

150

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

```
226
        ADDI
                 r8, r8, WORD_SIZE
                                          /* and then move to bit clr reg */
                                          /* to send it low */
227
        STWIO
                 r9, (r8)
228
        RET
                                          /* all done, so return */
229
230
231
232
233
        set_delay
234
235
        Description:
                             This procedure configures the sampling delay on manual triggers.
                             After execution, triggering will occur <delay> samples after the
236
                             configured level and slope settings are satisfied. Note that this
237
238
                             setting is only used when manual triggering is enabled.
                             Also note that delay must be less than MAX DELAY.
239
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
                             delay register.
243
244
                             Finally, the reset bit within the trigger control register is
                             pulsed to reset the triggering logic.
245
246
247
        Arguments:
                             delay - unsigned integer <= MAX DELAY; trigger delay from
248
                                     trigger event in number of samples (r4).
249
        Return Value:
                             None.
250
251
        Local Variables:
252
                             None
253
        Shared Variables:
                             None.
254
255
256
        Global Variables:
                            None.
257
258
        Input:
                             None.
259
        Output:
                             None.
260
261
262
        Error Handling:
                             None.
263
        Limitations:
                             Only positive delays less than or equal to MAX DELAY are valid.
264
265
266
        Algorithms:
                             None.
        Data Structures:
267
                             None.
268
269
        Registers Changed: r4, r10.
270
271
        Revision History:
272
             5/29/14
                       Santiago Navonne
                                              Initial revision.
                                              Expanded documentation.
273
             6/01/14
                       Santiago Navonne
274
275
        .global set_delay
276
277
        delay:
278
        MOVHI
                 r10, %hi(TRIG_DELAY_BASE) /* load trigger delay register address to update */
                 r10, r10, %lo(TRIG DELAY BASE) /* the desired delay time */
        ORI
279
280
        ADDT
                 r4, r4, DELAY_CONSTANT
                                             /* add delay constant to correct argument */
                                             /* and output to delay register, effectively */
/* configuring delay */
        STWIO
281
                 r4, (r10)
282
283
        MOVHI
                 r8, %hi(TRIG CTRL SET)
                                             /* load trigger control bit set reg address */
284
                 r8, r8, %lo(TRIG_CTRL_SET)
                                                 /* to reset trigger logic */
        ORI
285
                                             /* by sending reset bit high */
                 r9, FIFO_RESET_BIT
286
        MOVT
                r9, (r8)
r8, r8, WORD_SIZE
287
        STWIO
        ADDT
                                             /* and then move to bit clr reg */
288
289
        STWIO
                 r9, (r8)
                                             /* to send it low */
290
        RET
                                             /* all done, so return */
291
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
                             is TRUE, sampling starts immediately.
299
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
                            control register to enable or disable auto triggering.
304
305
                            Finally, it starts the sample by enabling writing to the FIFO
                            through the write enable bit in the control register, and resets
306
307
                            the triggering logic.
308
                            auto_trigger - TRUE if sampling should be started
309
        Arguments:
310
                                             automatically (i.e. as soon as possible),
                                             FALSE if it should be started on a trigger
311
                                            event (r4).
312
313
        Return Value:
                            None.
314
315
316
        Local Variables:
                            None.
317
        Shared Variables:
                            None.
318
319
320
        Global Variables:
                            None.
321
322
        Input:
                            None.
323
324
        Output:
                            None.
325
        Error Handling:
326
                            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
        .global start_sample
340
341
   start sample:
342
                 r8, %hi(TRIG CTRL CLR) /* load trigger control bit clear reg address */
        MOVHI
343
                r8, r8, %lo(TRIG_CTRL_CLR) /* assuming we'll clear auto trigger bit */
344
        ORI
345
        IVOM
                r9, 2
                                         /* subtract argument multiplied by word size */
        SLL
                                         /* effectively moving to set bit register if enabling */
                r4, r4, r9
346
347
        SUB
                r8, r8, r4
                                             auto trigger*/
348
                                         /* store auto trigger bit in configured register */
349
        MOVT
                r9, AUTO_TRIG_BIT
350
        STWIO
                r9, (r8)
                                         /* enabling or disabling it as needed */
351
        MOVHT
                r8, %hi(TRIG_CTRL_SET)
                                             /* load trigger control bit set reg address */
352
353
        ORI
                 r8, r8, %lo(TRIG_CTRL_SET)
                                                 /* to reset trigger logic */
                r9, FIFO_RESET BIT
                                             /* by sending reset bit high */
        MOVT
354
                r9, (r8)
r8, r8, WORD_SIZE
355
        STWIO
                                             /* and then move to bit clr reg */
356
        ADDI
                                             /* to send it low */
357
        STWIO
                r9, (r8)
358
                 r8, %hi(TRIG CTRL CLR) /* load trigger control bit clear reg address */
359
        MOVHI
                r8, r8, %lo(TRIG_CTRL_CLR) /* to clear fifo write enable (make active) */
        ORI
360
                 r9, FIFO_WE_BIT
361
        MOVT
                                         /* 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.
                            If the sample was completed, a pointer to the buffer containing the
372
373
     *
                            sampled data is provided. If the sample was not completed, a NULL
                            pointer is returned.
374
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
                            ensure that a sample is ready. If no sample is ready, it simply
379
380
                            returns with NULL in r2.
                            Then, it resets the values of the shared variable to indicate that
381
                            a sample was completed.
382
383
                            Finally, the function clocks the FIFO twice to account for its
                            latency, and then reads FIFO_SIZE bytes in a loop, storing them in
384
385
                            array <samples>. Note that at each iteration, reading is performed
                            by bit-banging the FIFO's read clock. Also note that a calibration
386
                            constant is added to each sample to account for the front end's DC
387
388
                            offset.
389
        Arguments:
390
                            None.
391
392
        Return Value:
                            *samples - pointer to bytes acquired in sample if any; NULL
                                       otherwise (r2).
393
394
395
       Local Variables:
                            r13 - pointer to current place in samples array.
                            r10 - number of sample currently being copied.
396
397
398
        Shared Variables: - sample pending: logical value; zero if no sample is pending,
399
                                               non-zero otherwise. Read/Write.
400
       Global Variables: None.
401
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
                                      whose pointer is returned.
413
414
       Registers Changed: r2, r8, r9, r10, r11, r12, r13, r14.
415
416
417
        Revision History:
     *
            5/29/14
                                             Initial revision.
                      Santiago Navonne
418
419
            6/01/14
                      Santiago Navonne
                                             Expanded documentation.
420
421
        .global sample_done
422
423
    sample done:
                                        /* assume no sample ready: null pointer return val */
424
       MOV
                r2, r0
425
       MOVIA
                r8, sample pending
                                         /* fetch current pending value to see if this call */
       LDB
                r9, (r8)
                                        /* should be ignored */
426
                                                which is when value is zero */
                r0, r9, sample_done_done
                                             /*
427
        BEO
428
       MOVTA
                                        /* reset sample_pending to indicate */
429
                r8, sample pending
430
       STB
                r0, (r8)
                                         /* no sample is ready for processing */
431
       MOVHI
                r12, %hi(FIFO DATA BASE) /* load fifo data register address */
432
                r12, r12, %lo(FIFO_DATA_BASE) /* to actually read data from fifo */
433
        ORI
        MOVHI
                r8, %hi(TRIG CTRL SET) /* load ctrl reg set bit addr for */
434
                r8, r8, %lo(TRIG_CTRL_SET)
                                              /* for bit banging */
435
       ORI
436
       MOVTA
                r13, sample
                                        /* load array address to store samples */
437
       MOV
                r2, r13
                                        /* and also use it as return value (pointer) */
                                        /* and start a counter at 0 for looping */
       MOV
                r10, r0
438
                                         /* which will stop at FIFO_SIZE */
439
       IVOM
                r11, FIFO SIZE
440
       MOVI
                r9, FIFO READ BIT
                                        /* finally load read clk bit for big banging */
441
442
                                         /* FIFO has 2 clocks latency */
                r9, (r8)
        STWIO
                                         /* send read clock high to output sample */
443
                r8, r8, WORD_SIZE
                                         /* and move to clear register: will send low next time */
444
        ADDI
                                        /* wait for sample to actually come through */
445
        NOP
446
        STWIO
                                        /* send read clock low to prepare for next sample */
                r9, (r8)
                r8, r8, NEG WORD SIZE
447
        ADDI
                                        /* and move to set register: will send high next time */
448
        NOP
                                        /* wait for sample to actually come through */
449
        STWIO
                r9, (r8)
                                        /* send read clock high to output sample */
450
```

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

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

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

641

```
*******************
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
      This file contains the constants for the data sampling and triggering
13
      routines. The file includes hardware constants used to interact with the
14
     triggering logic; masks used to access hardware registers; PIO register
15
     offsets; PIO register addresses; and default configuration values.
16
17
18
19
     Revision History:
        5/30/14 Santiago Navonne Initial revision.
2.0
   */
21
22
  /* Hardware constants */
23
2.4
  #define
             CLK_FREQ
                           38000000
                                       /* System clock frequency in Hz */
   #define
             FIFO SIZE
                                       /* Size of sample FIFO in words */
25
                           512
             TRIG LEVEL SHIFT 1
                                       /* Shift trig level left once to convert [0, 127] -> [0, 255] */
  #define
26
             CALIBRATION 13
                                       /* DC offset of front end */
  #define
2.7
28
  #define
             DELAY CONSTANT 1
                                       /* Hardware delay offset */
                          0xffffffff - 1 - DELAY CONSTANT
  #define
             MAXDELAY
29
30
                                      /* Maximum delay must take hardware delay offset into account */
31
  /* Masks */
32
                                      /* FIFO interrupt bit */
33
  #define
             FIFO_INT
                           1
  #define
             AUTO TRIG BIT 1<<0
                                       /* Auto trigger bit is bit 0 in trigger control register */
34
                                       /* Slope control bit is bit 1 in trigger control register */
             SLOPE BIT 1<<1
  #define
35
                                      /* FIFO write enable bit is bit 2 in trigger control register */
36
  #define
            FIFO WE BIT
                          1<<2
             FIFO READ BIT 1<<3
                                       /* FIFO read clock bit is bit 3 in trigger control register */
37
  #define
  #define
             FIFO RESET BIT 1<<4
                                      /* FIFO reset bit is bit 4 in trigger control register */
38
39
40
   /* PIO register offsets */
             EDGE CAP OF 3*WORD SIZE /* Offset of edge capture PIO register */
  #define
41
  #define
             INTMASK_OF
                           2*WORD_SIZE /* Offset of interrupt mask PIO register */
42
                           4*WORD_SIZE /* Offset of bit set PIO register */
  #define
             SET OF
43
                           5*WORD_SIZE /* Offset of bit clear PIO register */
44
  #define
             CLR_OF
45
   /* PIO offset locations */
46
             TRIG_CTRL_SET TRIG_CTRL_BASE+SET_OF /* Location of trigger control set bit register */
47
  #define
             TRIG CTRL CLR TRIG CTRL BASE+CLR OF /* Location of trigger control clear bit register */
48
  #define
49
   /* Default values */
50
             TRIG CTRL DEF 0b00000111 /* Initialize control register to: low read clock, inactive */
  #define
51
                                      ^{\prime *} (high) write enable, negative slope, auto trigger ^{*}/
52
53
  #define
             TRIG_DELAY_DEF 0+DELAY_CONSTANT /* Default trigger delay (desired delay + DELAY_CONSTANT) *
             TRIG LEVEL DEF 128 /* Default trigger level */
  #define
54
             DEFAULT_SAMPLE RATE 19000000
   #define
                                            /* Default sample rate */
55
   #define
             TRIG PERIOD DEF CLK FREQ/DEFAULT SAMPLE RATE /* Translates into this trigger period */
56
57
```

58

```
* system.h - SOPC Builder system and BSP software package information
2
3
      Machine generated for CPU 'nios' in SOPC Builder design 'sopc scope sys'
4
      SOPC Builder design path: C:/Users/tago/Dropbox/OUT/EE52/quartus/sopc scope sys.sopcinfo
5
6
      Generated: Wed Jun 11 15:26:36 PDT 2014
7
8
9
10
    * DO NOT MODIFY THIS FILE
11
12
    * Changing this file will have subtle consequences
13
      which will almost certainly lead to a nonfunctioning
14
      system. If you do modify this file, be aware that your
15
      changes will be overwritten and lost when this file
16
    * is generated again.
17
18
19
    * DO NOT MODIFY THIS FILE
20
21
22
    * License Agreement
23
24
      Copyright (c) 2008
25
      Altera Corporation, San Jose, California, USA.
26
2.7
      All rights reserved.
28
    * Permission is hereby granted, free of charge, to any person obtaining a
29
30
      copy of this software and associated documentation files (the "Software"),
      to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense,
31
32
    * and/or sell copies of the Software, and to permit persons to whom the
33
      Software is furnished to do so, subject to the following conditions:
34
35
      The above copyright notice and this permission notice shall be included in
36
      all copies or substantial portions of the Software.
37
38
39
    * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
      IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
40
    * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
41
    * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
42
      LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
43
    * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
44
45
    * DEALINGS IN THE SOFTWARE.
46
47
    * This agreement shall be governed in all respects by the laws of the State
    * of California and by the laws of the United States of America.
48
49
50
   #ifndef __SYSTEM_H_
51
   #define __SYSTEM_H_
52
53
   /* Include definitions from linker script generator */
54
   #include "linker.h"
55
56
57
58
    * CPU configuration
59
60
61
62
   #define ALT_CPU_ARCHITECTURE "altera_nios2_qsys"
63
   #define ALT CPU BIG ENDIAN 0
64
   #define ALT_CPU_BREAK_ADDR 0x00240820
#define ALT_CPU_CPU_FREQ 50000000u
65
66
   #define ALT_CPU_CPU_ID_SIZE 1
  #define ALT_CPU_CPU_ID_VALUE 0x00000000
#define ALT_CPU_CPU_IMPLEMENTATION "tiny"
68
69
   #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
```

```
#define ALT CPU FREQ 50000000
     #define ALT_CPU_HARDWARE_DIVIDE PRESENT 0
    #define ALT_CPU_HARDWARE_MULTIPLY_PRESENT 0 #define ALT_CPU_HARDWARE_MULX_PRESENT 0
 78
 79
 80
    #define ALT_CPU_HAS_DEBUG_CORE 1
    #define ALT_CPU_HAS_DEBUG_STUB
#define ALT_CPU_HAS_JMPI_INSTRUCTION
 81
 82
    #define ALT_CPU_ICACHE_LINE_SIZE 0
 83
    #define ALT_CPU_ICACHE_LINE_SIZE_LOG2 0
#define ALT_CPU_ICACHE_SIZE 0
 84
 85
    #define ALT_CPU_INST_ADDR_WIDTH 0x16
    #define ALT_CPU_NAME "nios"
 87
    #define ALT CPU RESET ADDR 0x00180000
 88
 89
 90
 91
      * CPU configuration (with legacy prefix - don't use these anymore)
 92
 93
 94
 95
    #define NIOS2_BIG_ENDIAN 0
#define NIOS2_BREAK_ADDR 0x00240820
 96
 97
    #define NIOS2_CPU_FREQ 50000000u
 98
99
    #define NIOS2_CPU_ID_SIZE 1
    #define NIOS2_CPU_ID_VALUE 0x00000000
#define NIOS2_CPU_IMPLEMENTATION "tiny"
101
    #define NIOS2_DATA_ADDR_WIDTH 0x16
102
    #define NIOS2_DCACHE_LINE_SIZE 0
#define NIOS2_DCACHE_LINE_SIZE_LOG2_0
104
105
    #define NIOS2_DCACHE_SIZE 0
    #define NIOS2_EXCEPTION_ADDR 0x00180020
#define NIOS2_FLUSHDA_SUPPORTED
107
108
    #define NIOS2_HARDWARE_DIVIDE_PRESENT 0
    #define NIOS2_HARDWARE_MULTIPLY_PRESENT 0
#define NIOS2_HARDWARE_MULX_PRESENT 0
109
110
    #define NIOS2 HAS DEBUG CORE 1
111
    #define NIOS2_HAS_DEBUG_STUB
#define NIOS2_HAS_JMPI_INSTRUCTION
112
113
    #define NIOS2 ICACHE LINE SIZE 0
    #define NIOS2_ICACHE_LINE_SIZE_LOG2 0 #define NIOS2_ICACHE_SIZE 0
115
116
    #define NIOS2 INST ADDR WIDTH 0x16
    #define NIOS2 RESET ADDR 0x00180000
118
119
120
121
122
     * Define for each module class mastered by the CPU
123
124
125
    #define __ALTERA_AVALON_JTAG_UART
126
     #define __ALTERA_AVALON_PIO
127
    #define __ALTERA_GENERIC_TRISTATE_CONTROLLER
#define __ALTERA_NIOS2_QSYS
128
129
130
131
132
     * System configuration
133
134
135
136
     #define ALT DEVICE FAMILY "Cyclone III"
137
    #define ALT_ENHANCED_INTERRUPT_API_PRESENT
138
139
     #define ALT IRQ BASE NULL
    #define ALT_LOG_PORT "/dev/null"
#define ALT_LOG_PORT_BASE 0x0
140
141
    #define ALT_LOG_PORT_DEV null
    #define ALT_LOG_PORT_TYPE ""
143
    #define ALT_NUM_EXTERNAL_INTERRUPT_CONTROLLERS 0
144
    #define ALT NUM INTERNAL INTERRUPT CONTROLLERS 1
    #define ALT_NUM_INTERRUPT_CONTROLLERS 1
146
    #define ALT_STDERR "/dev/jtag"
147
148 #define ALT STDERR BASE 0x241180
149 #define ALT_STDERR_DEV jtag
150 #define ALT_STDERR_IS_JTAG_UART
```

```
#define ALT_STDERR_PRESENT
    #define ALT_STDERR_TYPE "altera_avalon_jtag_uart"
   #define ALT_STDIN "/dev/jtag"
#define ALT STDIN BASE 0x241180
153
154
155
    #define ALT_STDIN_DEV jtag
    #define ALT_STDIN_IS_JTAG_UART
#define ALT STDIN PRESENT
156
157
    #define ALT_STDIN_TYPE "altera_avalon_jtag_uart"
158
    #define ALT_STDOUT "/dev/jtag"
159
    #define ALT_STDOUT_BASE 0x241180
160
    #define ALT STDOUT DEV jtag
    #define ALT_STDOUT_IS_JTAG_UART
#define ALT_STDOUT_PRESENT
162
163
    #define ALT STDOUT TYPE "altera avalon jtag uart"
    #define ALT_SYSTEM_NAME "sopc_scope_sys"
165
166
167
168
169
     * fifo_data configuration
170
     * /
171
172
    #define ALT MODULE CLASS fifo data altera avalon pio
173
174
    #define FIFO_DATA_BASE 0x241140
    #define FIFO DATA BIT CLEARING EDGE REGISTER 0
175
    #define FIFO_DATA_BIT_MODIFYING_OUTPUT_REGISTER 0
176
177
    #define FIFO_DATA_CAPTURE 0
    #define FIFO DATA DATA WIDTH 8
    #define FIFO DATA DO TEST BENCH WIRING 0
179
180
    #define FIFO_DATA_DRIVEN_SIM_VALUE 0
   #define FIFO_DATA_EDGE_TYPE "NONE"
#define FIFO_DATA_FREQ 50000000
182
183
    #define FIFO_DATA_HAS_IN 1
#define FIFO_DATA_HAS_OUT 0
#define FIFO_DATA_HAS_TRI 0
    #define FIFO DATA IRQ -1
186
    #define FIFO_DATA_IRQ_INTERRUPT_CONTROLLER_ID -1 #define FIFO_DATA_IRQ_TYPE "NONE"
187
188
    #define FIFO DATA NAME "/dev/fifo data"
189
190 #define FIFO_DATA_RESET_VALUE 0
191
    #define FIFO DATA SPAN 16
    #define FIFO DATA TYPE "altera avalon pio"
192
193
194
195
     * fifo_full configuration
196
197
198
199
    #define ALT MODULE CLASS fifo full altera avalon pio
200
    #define FIFO_FULL_BASE 0x241130
201
    #define FIFO_FULL_BIT_CLEARING_EDGE_REGISTER 0
202
203
    #define FIFO_FULL_BIT_MODIFYING_OUTPUT_REGISTER 0
    #define FIFO FULL CAPTURE 1
204
205
    #define FIFO_FULL_DATA_WIDTH 1
    #define FIFO FULL DO TEST BENCH WIRING 0
206
    #define FIFO FULL DRIVEN SIM VALUE 0
207
208
    #define FIFO_FULL_EDGE_TYPE "RISING"
    #define FIFO FULL FREQ 50000000
#define FIFO_FULL_HAS_IN 1
211
    #define FIFO_FULL_HAS_OUT 0
    #define FIFO_FULL_HAS_TRI 0
    #define FIFO_FULL_IRQ 4
213
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
    #define FIFO_FULL_SPAN 16
218
    #define FIFO_FULL_TYPE "altera_avalon_pio"
219
220
221
222
223
     * hal configuration
224
225
```

```
226
227
    #define ALT_MAX_FD 32
    #define ALT_SYS_CLK none
#define ALT_TIMESTAMP_CLK none
228
229
230
231
232
233
     * jtag configuration
234
235
     * /
236
    #define ALT_MODULE_CLASS_jtag altera_avalon_jtag_uart
237
    #define JTAG BASE 0x241180
238
    #define JTAG IRQ 0
    #define JTAG_IRQ_INTERRUPT_CONTROLLER_ID 0
240
    #define JTAG NAME "/dev/jtag"
241
    #define JTAG READ DEPTH 64
    #define JTAG_READ_THRESHOLD 8
243
244
    #define JTAG_SPAN 8
    #define JTAG TYPE "altera_avalon_jtag_uart"
245
    #define JTAG_WRITE_DEPTH \overline{64}
246
247
    #define JTAG_WRITE_THRESHOLD 8
248
249
250
     * pio_0 configuration
251
252
253
254
255
    #define ALT_MODULE_CLASS_pio_0 altera_avalon_pio
    #define PIO_0_BASE 0x2410a0
#define PIO_0_BIT_CLEARING_EDGE_REGISTER 1
256
257
    #define PIO_0_BIT_MODIFYING_OUTPUT_REGISTER 1
258
    #define PIO_0_CAPTURE 1
#define PIO_0_DATA_WIDTH 6
259
260
    #define PIO 0 DO TEST BENCH WIRING 0
261
    #define PIO_0_DRIVEN_SIM_VALUE 0
#define PIO_0_EDGE_TYPE "FALLING"
262
263
    #define PIO 0 FREQ 50000000
   #define PIO_0_HAS_IN 1
#define PIO_0_HAS_OUT 0
265
266
    #define PIO 0 HAS TRI 0
    #define PIO_0_IRQ_I
#define PIO_0_IRQ_INTERRUPT_CONTROLLER_ID 0
268
269
    #define PIO 0 IRQ TYPE "EDGE"
    #define PIO_0_NAME "/dev/pio 0"
271
272
    #define PIO_0_RESET_VALUE 0
    #define PIO 0 SPAN 32
    #define PIO_0_TYPE "altera_avalon_pio"
274
275
276
277
278
     * ram configuration
279
280
281
    #define ALT MODULE CLASS ram altera generic tristate controller
282
283
    #define RAM_BASE 0x220000
284
    #define RAM IRQ -1
    #define RAM_IRQ_INTERRUPT_CONTROLLER_ID -1
285
286
    #define RAM NAME "/dev/ram"
287
    #define RAM SPAN 131072
    #define RAM_TYPE "altera_generic_tristate_controller"
288
289
290
291
292
     * rom configuration
293
294
    #define ALT MODULE CLASS_rom altera_generic_tristate_controller
296
297
    #define ROM BASE 0x180000
    #define ROM IRQ -1
    #define ROM_IRQ_INTERRUPT_CONTROLLER ID -1
299
300
    #define ROM_NAME "/dev/rom"
```

```
#define ROM SPAN 524288
302
    #define ROM_TYPE "altera_generic_tristate_controller"
303
304
305
     * trig ctrl configuration
306
307
308
309
310
    #define ALT_MODULE_CLASS_trig_ctrl altera_avalon_pio
    #define TRIG CTRL BASE 0x241060
    #define TRIG_CTRL_BIT_CLEARING_EDGE_REGISTER 0
#define TRIG_CTRL_BIT_MODIFYING_OUTPUT_REGISTER 1
312
313
    #define TRIG CTRL CAPTURE 0
    #define TRIG_CTRL_DATA_WIDTH 5
315
    #define TRIG CTRL DO TEST BENCH WIRING 0
316
    #define TRIG_CTRL_DRIVEN_SIM_VALUE 0 #define TRIG_CTRL_EDGE_TYPE "NONE"
318
    #define TRIG CTRL FREQ 50000000
319
    #define TRIG CTRL HAS IN 0
    #define TRIG_CTRL_HAS_OUT 1
321
322
    #define TRIG CTRL HAS TRI 0
    #define TRIG_CTRL_IRQ -1
324
    #define TRIG_CTRL_IRQ_INTERRUPT_CONTROLLER_ID -1
    #define TRIG CTRL IRQ TYPE "NONE"
    #define TRIG_CTRL_NAME "/dev/trig_ctrl"
    #define TRIG_CTRL_RESET_VALUE 3
327
    #define TRIG_CTRL_SPAN 32
#define TRIG_CTRL_TYPE "altera_avalon_pio"
328
329
330
331
332
     * trig_delay configuration
333
334
335
336
337
    #define ALT_MODULE_CLASS_trig_delay altera_avalon_pio
    #define TRIG_DELAY_BASE 0x241120
338
    #define TRIG DELAY BIT CLEARING EDGE REGISTER 0
    #define TRIG_DELAY_BIT_MODIFYING_OUTPUT_REGISTER 0
#define TRIG_DELAY_CAPTURE 0
340
341
    #define TRIG DELAY DATA WIDTH 32
    #define TRIG_DELAY_DO_TEST_BENCH_WIRING 0 #define TRIG_DELAY_DRIVEN_SIM_VALUE 0
343
344
    #define TRIG DELAY EDGE TYPE "NONE"
    #define TRIG_DELAY_FREQ 50000000
346
347
    #define TRIG DELAY HAS IN 0
    #define TRIG DELAY HAS OUT 1
    #define TRIG_DELAY_HAS_TRI 0
349
    #define TRIG DELAY IRQ -1
350
    #define TRIG_DELAY_IRQ_INTERRUPT_CONTROLLER_ID -1
    #define TRIG_DELAY_IRQ_TYPE "NONE"
352
353
    #define TRIG_DELAY_NAME "/dev/trig_delay"
    #define TRIG DELAY RESET VALUE 1
354
355
    #define TRIG_DELAY_SPAN 16
    #define TRIG DELAY TYPE "altera avalon pio"
356
357
358
359
    * trig_level configuration
360
361
362
363
    #define ALT MODULE CLASS trig level altera avalon pio
    #define TRIG_LEVEL_BASE 0x241150
365
    #define TRIG_LEVEL_BIT_CLEARING_EDGE_REGISTER 0
366
    #define TRIG_LEVEL_BIT_MODIFYING_OUTPUT_REGISTER 0
    #define TRIG_LEVEL_CAPTURE 0
#define TRIG_LEVEL_DATA_WIDTH 8
368
369
    #define TRIG LEVEL DO TEST BENCH WIRING 0
    #define TRIG_LEVEL_DRIVEN_SIM_VALUE 0
371
    #define TRIG_LEVEL_EDGE_TYPE "NONE"
372
    #define TRIG LEVEL FREQ 50000000
    #define TRIG_LEVEL_HAS_IN 0
374
375
    #define TRIG LEVEL HAS OUT 1
```

```
376 #define TRIG_LEVEL_HAS_TRI 0
    #define TRIG_LEVEL_IRQ -1
    #define TRIG_LEVEL_IRQ_INTERRUPT_CONTROLLER_ID -1
#define TRIG_LEVEL_IRQ_TYPE "NONE"
378
379
380
    #define TRIG_LEVEL_NAME "/dev/trig_level"
    #define TRIG_LEVEL_RESET_VALUE 0
381
    #define TRIG_LEVEL_SPAN 16
382
383
    #define TRIG_LEVEL_TYPE "altera_avalon_pio"
384
385
386
     * trig_period configuration
387
388
389
390
    #define ALT_MODULE_CLASS_trig_period altera_avalon_pio
391
    #define TRIG_PERIOD_BASE 0x241160
    #define TRIG_PERIOD_BIT_CLEARING_EDGE_REGISTER 0
393
394
    #define TRIG_PERIOD_BIT_MODIFYING_OUTPUT_REGISTER 0
    #define TRIG PERIOD CAPTURE 0
    #define TRIG_PERIOD_DATA_WIDTH 32
396
397
    #define TRIG_PERIOD_DO_TEST_BENCH_WIRING 0
    #define TRIG PERIOD_DRIVEN_SIM_VALUE 0
399
    #define TRIG_PERIOD_EDGE_TYPE "NONE"
    #define TRIG PERIOD FREQ 50000000
    #define TRIG_PERIOD_HAS_IN 0
401
    #define TRIG_PERIOD_HAS_OUT 1
402
    #define TRIG_PERIOD_HAS_TRI 0
#define TRIG_PERIOD_IRQ -1
404
405
    #define TRIG_PERIOD_IRQ_INTERRUPT_CONTROLLER_ID -1
    #define TRIG_PERIOD_IRQ_TYPE "NONE"
#define TRIG_PERIOD_NAME "/dev/trig_period"
407
    #define TRIG_PERIOD_RESET_VALUE 1
408
    #define TRIG_PERIOD_SPAN 16
#define TRIG_PERIOD_TYPE "altera_avalon_pio"
409
410
411
412
413
414
     * vram configuration
415
416
417
    #define ALT MODULE CLASS vram altera generic tristate controller
418
    #define VRAM_BASE 0x0
419
    #define VRAM IRQ -1
    #define VRAM_IRQ_INTERRUPT_CONTROLLER_ID -1
421
422
    #define VRAM_NAME "/dev/vram"
    #define VRAM SPAN 1048576
    #define VRAM_TYPE "altera_generic_tristate_controller"
424
425
426
    #endif /* __SYSTEM_H_ */
427
```