

EE 52

SPRING 2017

Digital Oscilloscope Documentation

SOPHIA LIU

Contents

1 User Manual	3
1.1 System Description	3
1.2 How to Use the System	3
1.3 System Settings	3
1.3.1 Menu	3
1.3.2 Rotary Encoder Functionality	5
2 Technical Documentation	6
2.1 Hardware	6
2.1.1 Hardware System Overview	6
2.1.2 FPGA	9
2.1.3 Power	21
2.1.4 Memory	23
2.1.5 Analog System	27
2.1.6 Rotary Encoders	32
2.1.7 VRAM and LCD	37
2.1.8 JTAG, Reset, and Clock	44
2.1.9 Fixes	47
2.2 Software	48
2.2.1 Software System Overview	48
2.2.2 Analog Software	49
2.2.3 Display Software	59
2.2.4 Keypad Software	64
2.2.5 Watchdog Reset	72
2.2.6 Header files	75

A Timing Diagrams	78
A.1 ADC	78
A.2 LCD	78
A.3 ROM and RAM	79
A.4 VRAM	82
B Device Images	86
C Schematics and PCB	88
D Software	96
D.1 Hardware descriptions	96
D.2 Main code	107

1 User Manual

1.1 System Description

The System on Programmable Chip (SoPC) Digital Oscilloscope is an FPGA/microprocessor-based system capable of capturing and displaying up to 5 MHz signals. The analog input can range from 0 V to 3.8 V. The system has all the features of a standard oscilloscope with the exception of input signal scaling. Two rotary encoders are used to control the settings, and a LCD display is used to display the captured signal.

1.2 How to Use the System

The system will begin immediately upon powering up.¹ The system starts in one-shot trace mode with a sampling rate of 100 ns, a mid-level trigger (halfway between the minimum and maximum trigger levels, at 1.9 V) with no delay and positive slope, and with the menu displayed and scale set to axes. The probe can be attached to the desired input source. The oscilloscope settings are described in section 1.3. The reset button can be used to restart the system.

1.3 System Settings

1.3.1 Menu

The scope parameter menu is located in the upper right corner of the LCD and contains the following entries in order:

- Mode
- Scale
- Sweep
- Trigger
 - Level
 - Slope
 - Delay

The menu display can be toggled on and off by pressing the menu rotary encoder. The user can cursor to any of the entries by turning the menu rotary encoder and can change a setting by turning the secondary rotary encoder. Changes take effect

¹ The serial configuration device does not work, so the FPGA must first be programmed through the JTAG interface.

immediately. More rotary details can be found in section 1.3.2. The menu entries are described in more detail below.

- Mode : The Mode menu entry can be set to Mode Normal, Mode Automatic, or Mode One-Shot. In Mode Normal the scope waits for another trigger after every retrace. In this mode, new traces are captured as fast as the scope can redraw the screen. Mode Automatic works the same as Mode Normal if there are trigger events. But if no trigger event occurs after a specified delay (typically significantly longer than the time represented by a screen of data) the scope triggers automatically without a trigger event occurring. In Mode One-Shot the scope triggers only once and then holds that trace on the screen. It does not look for another trigger event until the Trigger menu item is selected and secondary rotary encoder is turned.
- Scale : The Scale menu entry can be set to Scale Axes, Scale Grid, or Scale Off. If the scale is set to Scale Axes, the x and y axes are displayed along with the trace. If the scale is set to Scale Grid, an x-y grid is displayed along with the trace. If the scale is set to Scale Off, no axes or grid are displayed.
- Sweep : The Sweep menu entry sets the sweep rate (in time per sample) for the scope. Possible settings are: 100, 200, and 500 nanoseconds, and 1, 2, 5, 10, 20, 50, 100, 200, and 500 microseconds, and 1, 2, 5, 10, and 20 milliseconds (per sample).
- Trigger : The Trigger menu entry re-arms the trigger for the scope in one-shot mode. Any time it is selected and the secondary rotary encoder is turned the scope trigger is re-armed and a new trace will then be captured once the trigger conditions (level and slope) are met.
- Level : The Level menu entry sets the trigger level. It can be set to any value from the most negative input voltage to the most positive in 128 steps. Additionally, the trigger level is displayed as a line on the screen when the trigger level is being changed.
- Slope : The Slope menu entry is either Slope + or Slope - and determines whether the scope is triggered on a positive or negative slope respectively.

Delay : The Delay menu entry determines the trigger delay. It sets the time after the trigger event at which the trace will start. It may be set to any value from the minimum delay to the maximum delay times the sample rate and it is displayed as a time.

1.3.2 Rotary Encoder Functionality

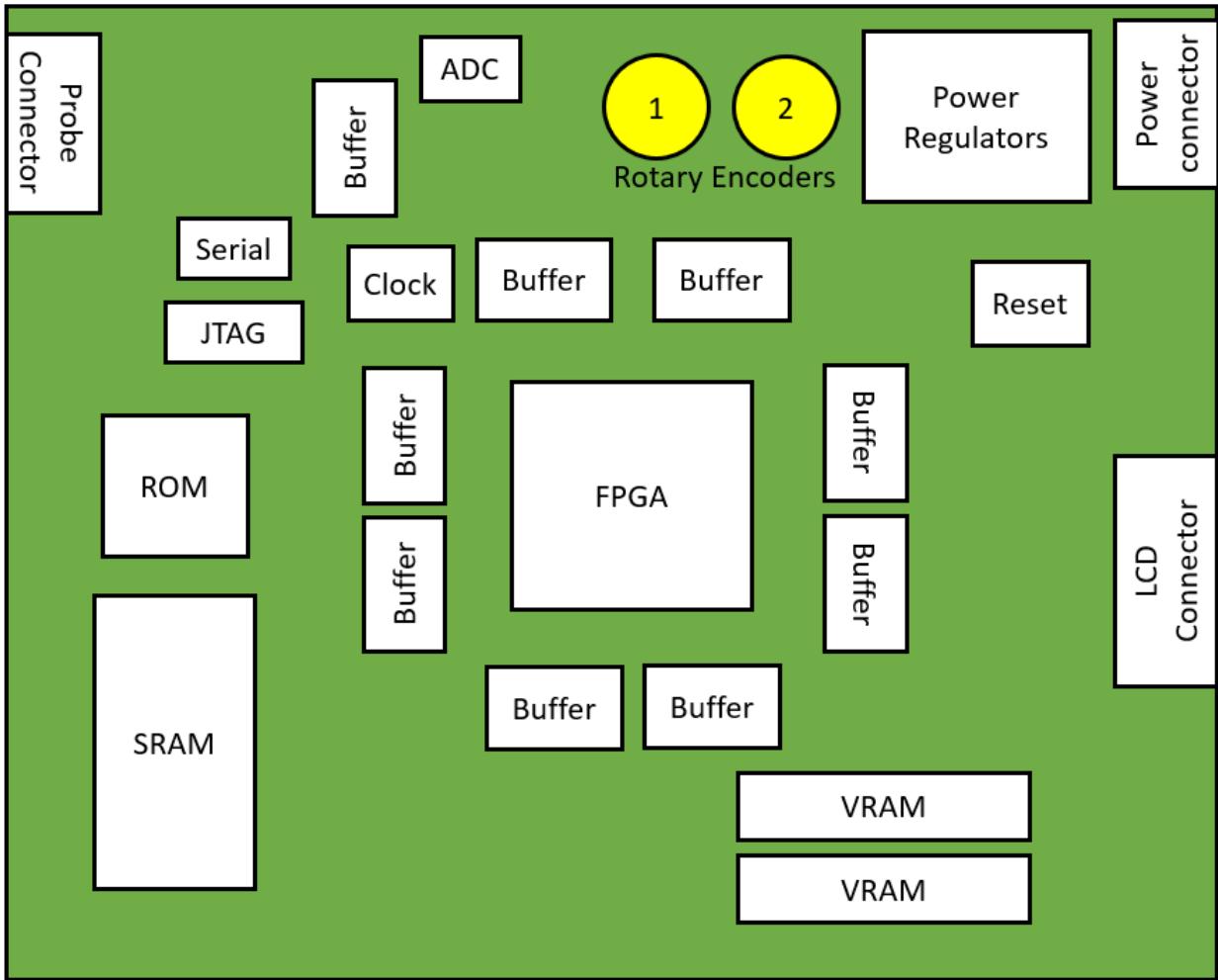


Figure 1: Board Layout: Labeled Rotary Encoders.

The user can change the scope configuration via two rotary encoders (seen in Figure 1). All of the scope parameters are set via an on-screen menu. The rotary actions are detailed below:

Press encoder 1 : Turns the menu on/off. If the menu is off it is not displayed and turning the rotary encoders have no effect on

the settings.

Turn encoder 1 CW : Moves the cursor down, if not already at the bottom menu item. If at the bottom, the cursor does not move.

Turn encoder 1 CCW : Moves the cursor up, if not already at the top menu item. If at the top, the cursor does not move.

Turn encoder 2 CW : Changes the currently selected (with cursor) menu item. Goes "forward" through the list of possible settings. If at the "end" of the list, doesn't change the current selection.

Turn encoder 2 CCW : Changes the currently selected (with cursor) menu item. Goes "backward" through the list of possible settings. If at the "beginning" of the list, doesn't change the current selection.

2 Technical Documentation

2.1 Hardware

2.1.1 Hardware System Overview

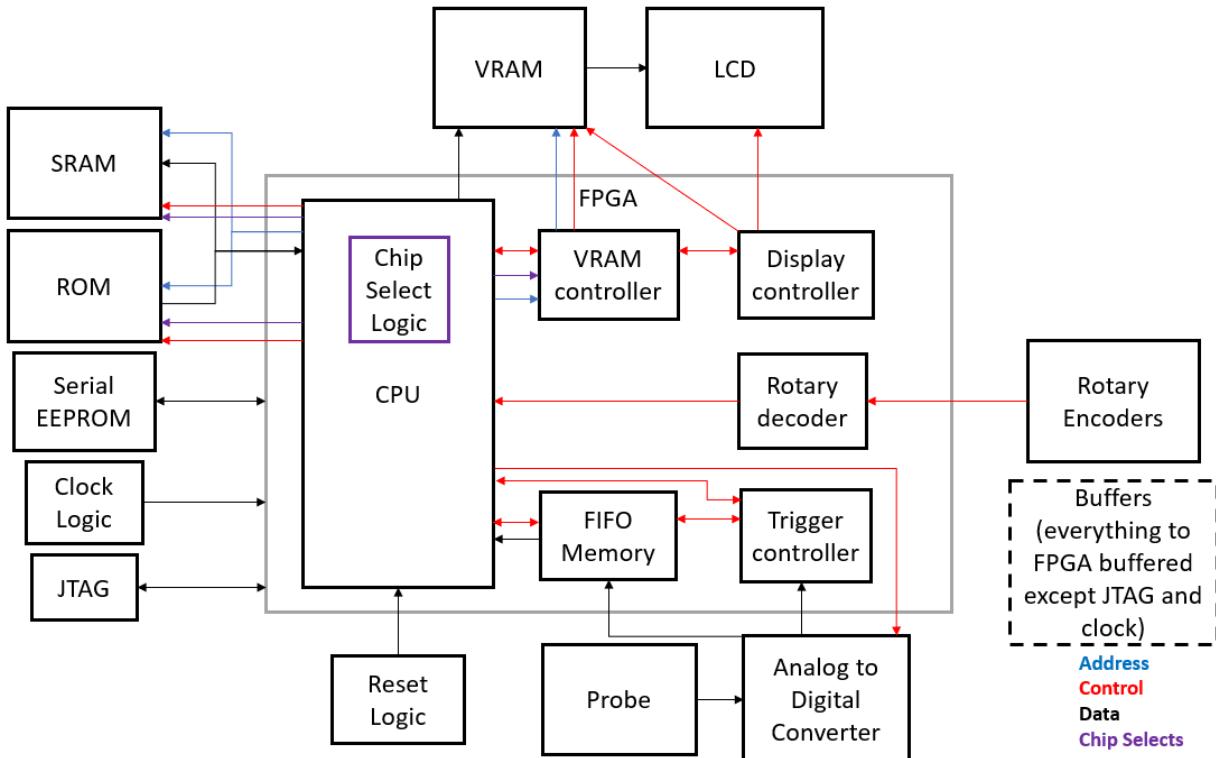


Figure 2: Block diagram overview of FPGA and peripheral chips.

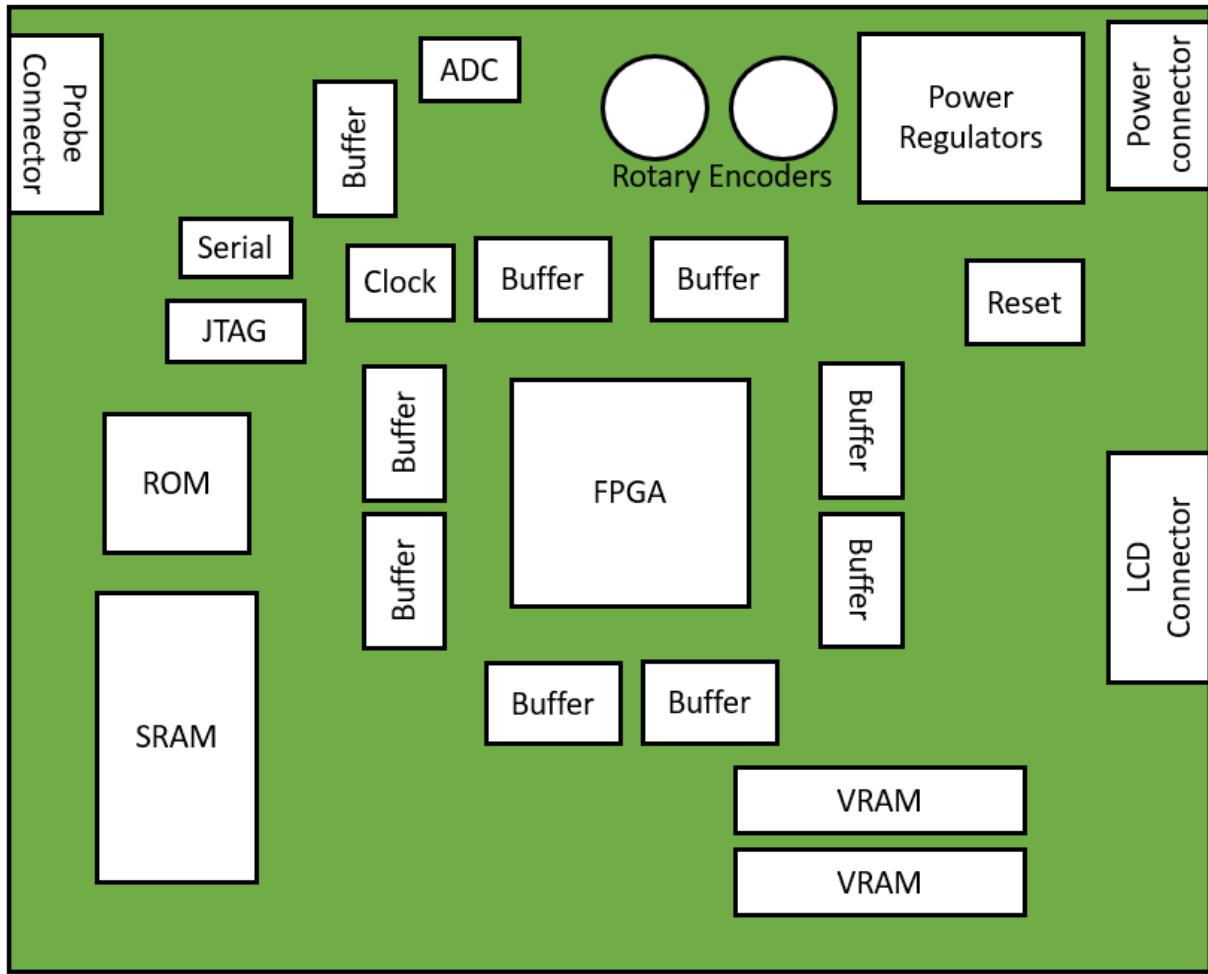


Figure 3: Board Layout.

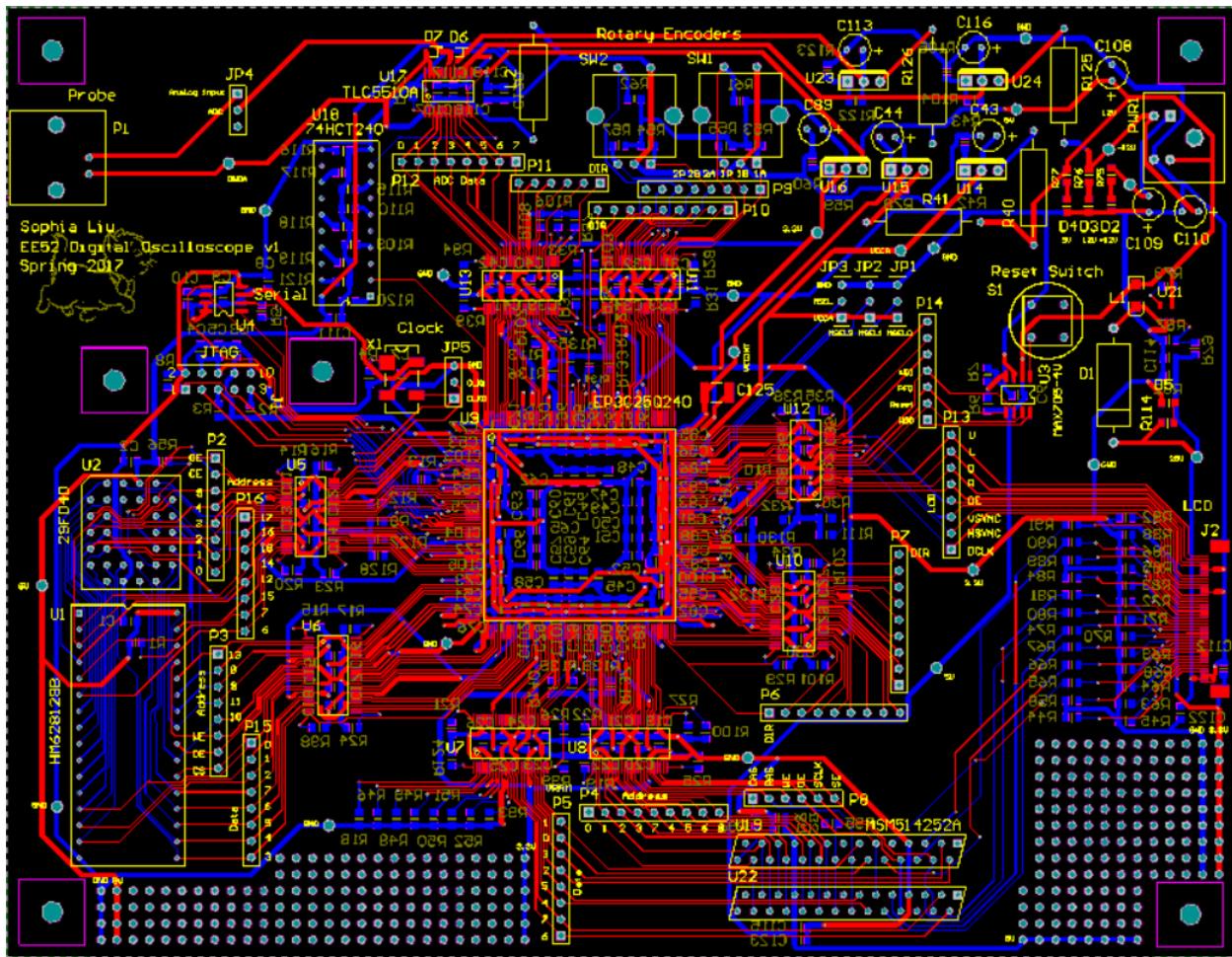


Figure 4: Image of PCB.

The oscilloscope system is centered around a Cyclone III FPGA, which includes a Nios II CPU and controllers for the VRAM, LCD, rotary encoders, and analog input and triggering. Upon startup, the bitmap is loaded from the serial EEPROM. The is then loaded from the ROM.

Receiving a Signal: When the probe is connected to a desired input, the analog signal is sent to an Analog to Digital Converter chip. 8 bits of data are then sent in parallel to the FPGA to an analog controller. The various trigger setting inputs (auto triggering, trigger enable, trigger slope, trigger level) are used along with the 8 bit signal to determine if the system should begin sampling. Once triggering has occurred, the signal is written to a FIFO buffer at the sampling rate. Once the FIFO is full, the CPU clocks out and stores the buffer data to be displayed.

Displaying a Signal: In order to display the signal on the LCD, data is written to the VRAM, which is then sent out to the LCD. Several control signals (write enable, chip select, address, wait) are sent from the CPU to the VRAM controller to perform read and write operations on the VRAM. An LCD controller sends out the necessary control signals to display the signal.

2.1.2 FPGA

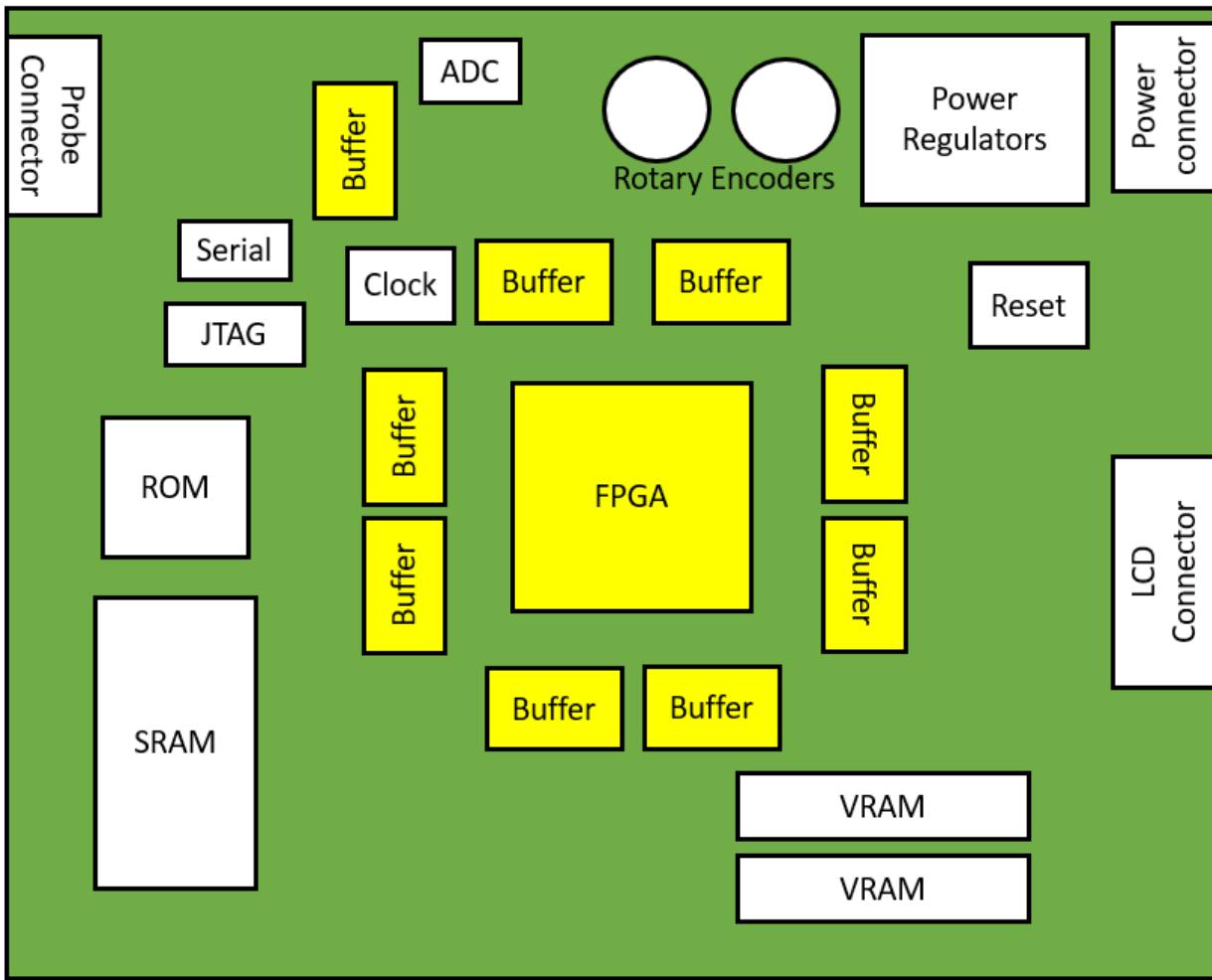


Figure 5: Board Layout: FPGA and buffers.

An Altera Cyclone III FPGA was used for all the logic and processing required (U9, EP3C25Q240). The three required voltage levels were generated with regulators, and each power pin was locally bypassed. The full schematics with the FPGA can be seen in Appendix C.

Buffers, or level shifters, were used to connect the buses and control signals from the peripheral chips to the FPGA to go between the low FPGA voltage and mainly 5V environment (U5, U6, U7, U8, U10, U12; 74LVT16245). A separate buffer was used to interface with the ADC (U17, TLC5510A), which had a larger input voltage range requirement (U18, SN74HCT240).

Pin Assignments: The pins were connected on the PCB as listed below.

Cyclone III Pins

Pin	Active	Direction	Connection	Use
VCCINT	N/A	N/A	1.2 V	Power
VCCIO1	N/A	N/A	3.3 V	I/O supply voltage pin
VCCIO2	N/A	N/A	3.3 V	I/O supply voltage pin
VCCIO3	N/A	N/A	3.3 V	I/O supply voltage pin
VCCIO4	N/A	N/A	3.3 V	I/O supply voltage pin
VCCIO5	N/A	N/A	3.3 V	I/O supply voltage pin
VCCIO6	N/A	N/A	3.3 V	I/O supply voltage pin
VCCIO7	N/A	N/A	3.3 V	I/O supply voltage pin
VCCIO8	N/A	N/A	3.3 V	I/O supply voltage pin
VCCA1	N/A	N/A	2.5 V	Analog power for PLL1
VCCA2	N/A	N/A	2.5 V	Analog power for PLL2
VCCA3	N/A	N/A	2.5 V	Analog power for PLL3
VCCA4	N/A	N/A	2.5 V	Analog power for PLL4
VCCD_PLL1	N/A	N/A	1.2 V	Digital power for PLL1
VCCD_PLL2	N/A	N/A	1.2 V	Digital power for PLL2
VCCD_PLL3	N/A	N/A	1.2 V	Digital power for PLL3
VCCD_PLL4	N/A	N/A	1.2 V	Digital power for PLL4
GND	N/A	N/A	Ground	Device ground pins
GNDA1	N/A	N/A	Analog Ground	PLL1, analog circuits ground
GNDA2	N/A	N/A	Analog Ground	PLL2, analog circuits ground

Pin	Active	Direction	Connection	Use
GNDA3	N/A	N/A	Analog Ground	PLL3, analog circuits ground
GNDA4	N/A	N/A	Analog Ground	PLL4, analog circuits ground
CLK0/DIFFCLK_0p	N/A	Input	Clock	Clock input
CLK1/DIFFCLK_0n	N/A	Input	Clock	Backup clock input
CLK2/DIFFCLK_1p	N/A	Input	Ground	Unused, connect to ground
CLK3/DIFFCLK_1n	N/A	Input	Ground	Unused, connect to ground
CLK4/DIFFCLK_2p	N/A	Input	Ground	Unused, connect to ground

CLK5/DIFFCLK_2n	N/A	Input	Ground	Unused, connect to ground
CLK6/DIFFCLK_3p	N/A	Input	Ground	Unused, connect to ground
CLK7/DIFFCLK_3n	N/A	Input	Ground	Unused, connect to ground
CLK8/DIFFCLK_5n	N/A	Input	Ground	Unused, connect to ground
CLK9/DIFFCLK_5p	N/A	Input	Ground	Unused, connect to ground
CLK10/DIFFCLK_4n	N/A	Input	Ground	Unused, connect to ground
CLK11/DIFFCLK_4p	N/A	Input	Ground	Unused, connect to ground
CLK12/DIFFCLK_7n	N/A	Input	Ground	Unused, connect to ground
CLK13/DIFFCLK_7p	N/A	Input	Ground	Unused, connect to ground
CLK14/DIFFCLK_6n	N/A	Input	Ground	Unused, connect to ground
CLK15/DIFFCLK_6p	N/A	Input	Ground	Unused, connect to ground
nCE	Low	Input	Ground	Active-low chip enable, tie to ground to enable
nCONFIG	Low	Input	VCCIO	Configuration control input, connect to VCCIO (resets at low)
nSTATUS	N/A	Bi	Pull high	Configuration status pin, pull high

Pin	Active	Direction	Connection	Use
CONF_DONE	N/A	Bi	Pull high	Configuration status pin, pull high
DATA0	N/A	Input	Serial	Serial config device DATA pin
DCLK	N/A	Output	Serial	Serial config device DCLK pin
MSEL0	N/A	Input	Mode select	Configuration input pin
MSEL1	N/A	Input	Mode select	Configuration input pin
MSEL2	N/A	Input	Mode select	Configuration input pin
TCK	High	Input	JTAG	JTAG input pin
TDI	High	Input	JTAG	JTAG input pin
TDO	High	Output	JTAG	JTAG output pin
TMS	High	Input	JTAG	JTAG input pin
IO4/DIFFIO_L1p	High	Output	ROM	ROM output enable
IO5/DIFFIO_L1n	High	Output	ROM	ROM chip enable
IO6	N/A	Output	ROM/SRAM	ROM/SRAM address bus
IO9	N/A	Output	ROM/SRAM	ROM/SRAM address bus
IO12/DATA1	N/A	Output	Serial	Serial config device ASDO, used to read out configuration data
IO13/VREFB1N0	N/A	Output	ROM/SRAM	ROM/SRAM address bus
IO14/FLASH_nCE/nCS0	N/A	Output	Serial	Serial config device nCSO, output control signal to enable config device
IO18	N/A	Output	ROM/SRAM	ROM/SRAM address bus

IO21/DIFFIO_L6p	N/A	Output	ROM/SRAM	ROM/SRAM address bus
IO22/DIFFIO_L6n	N/A	Output	ROM/SRAM	ROM/SRAM address bus
IO37/DIFFIO_L8p	N/A	Output	ROM/SRAM	ROM/SRAM address bus

Pin	Active	Direction	Connection	Use
IO38/DIFFIO_L8n	N/A	Output	ROM/SRAM	ROM/SRAM address bus
IO39	N/A	Output	ROM/SRAM	ROM/SRAM address bus
IO41/DIFFIO_L10p	N/A	Output	ROM/SRAM	ROM/SRAM address bus
IO43/DIFFIO_L10n	N/A	Output	ROM/SRAM	ROM/SRAM address bus
IO44/DIFFIO_L11p	N/A	Output	ROM/SRAM	ROM/SRAM address bus
IO45/DIFFIO_L11n	N/A	Output	ROM/SRAM	ROM/SRAM address bus
IO46/VREFB2N0	N/A	Output	ROM/SRAM	ROM/SRAM address bus
IO49/DIFFIO_L13p	N/A	Output	ROM/SRAM	ROM/SRAM address bus
IO50/DIFFIO_L13n	N/A	Output	ROM/SRAM	ROM/SRAM address bus
IO51/RUP1	N/A	Output	ROM/SRAM	ROM/SRAM address bus
IO52/RDN1	N/A	Output	ROM/SRAM	ROM/SRAM address bus
IO55	N/A	Output	ROM/SRAM	ROM/SRAM address bus
IO56/DIFFIO_L15p	Low	Output	SRAM	SRAM write enable input
IO57/DIFFIO_L15n	Low	Output	SRAM	SRAM output enable input
IO63/DIFFIO_B2p	Low	Output	SRAM	SRAM chip select input
IO64/DIFFIO_B2n	N/A	I/O	ROM/SRAM	ROM/SRAM data bus
IO65	N/A	I/O	ROM/SRAM	ROM/SRAM data bus
IO68	N/A	I/O	ROM/SRAM	ROM/SRAM data bus
IO69/PLL1_CLKOUTp	N/A	I/O	ROM/SRAM	ROM/SRAM data bus
IO70/PLL1_CLKOUTn	N/A	I/O	ROM/SRAM	ROM/SRAM data bus
IO71/DIFFIO_B4p	N/A	I/O	ROM/SRAM	ROM/SRAM data bus

Pin	Active	Direction	Connection	Use
IO72/DIFFIO_B4n	N/A	I/O	ROM/SRAM	ROM/SRAM data bus
IO73	N/A	I/O	ROM/SRAM	ROM/SRAM data bus
IO76/VREFB3N0	High	Output	Buffer	ROM/SRAM data bus direction
IO78	N/A	I/O	DRAM	DRAM data bus
IO80	N/A	I/O	DRAM	DRAM data bus
IO81/DIFFIO_B7p	N/A	I/O	DRAM	DRAM data bus
IO82/DIFFIO_B7n	N/A	I/O	DRAM	DRAM data bus
IO83	N/A	I/O	DRAM	DRAM data bus

IO84	N/A	I/O	DRAM	DRAM data bus
IO87	N/A	I/O	DRAM	DRAM data bus
IO88	N/A	I/O	DRAM	DRAM data bus
IO93/DIFFIO_B13p	High	Output	Buffer	DRAM data bus direction
IO94/DIFFIO_B13n	N/A	Output	DRAM	DRAM address bus 2
IO95	N/A	Output	DRAM	DRAM address bus 3
IO98/DIFFIO_B16p	N/A	Output	DRAM	DRAM address bus 4
IO99/DIFFIO_B16n	N/A	Output	DRAM	DRAM address bus 5
IO100	N/A	Output	DRAM	DRAM address bus 6
IO103	N/A	Output	DRAM	DRAM address bus 7
IO106	N/A	Output	DRAM	DRAM address bus 8
IO107/VREFB4N0	N/A	Output	DRAM	DRAM address bus 9
IO108/DIFFIO_B20p	N/A	Output	DRAM	DRAM address bus 9

Pin	Active	Direction	Connection	Use
IO109/DIFFIO_B20n	Low	Output	DRAM	DRAM CAS
IO110	Low	Output	DRAM	DRAM RAS
IO111/RUP2	Low	Output	DRAM	DRAM mask enable/write enable
IO112/RDN2	Low	Output	DRAM	DRAM data transfer/output enable
IO113	N/A	Output	DRAM	DRAM serial clock input
IO114	Low	Output	DRAM	DRAM serial port enable
IO117/PLL4_CLKOUTp	High	Output	Buffer	Buffer direction
IO118/PLL4_CLKOUTn	N/A	I/O	Buffer	Extra I/O
IO119/DIFFIO_B25p	N/A	I/O	Buffer	Extra I/O
IO120/DIFFIO_B25n	N/A	I/O	Buffer	Extra I/O
IO126/RUP3	N/A	I/O	Buffer	Extra I/O
IO127/RDN3	N/A	I/O	Buffer	Extra I/O
IO128	N/A	I/O	Buffer	Extra I/O
IO131/DIFFIO_R13n	N/A	I/O	Buffer	Extra I/O
IO132/DIFFIO_R13p	N/A	I/O	Buffer	Extra I/O
IO133/VREFB5N0	N/A	I/O	Buffer	Extra I/O
IO134/DIFFIO_R12n	N/A	I/O	Buffer	Extra I/O
IO135/DIFFIO_R12p	N/A	I/O	Buffer	Extra I/O
IO137/DIFFIO_R11n	N/A	I/O	Buffer	Extra I/O
IO139/DIFFIO_R11p	N/A	I/O	Buffer	Extra I/O
IO142/DIFFIO_R10n	N/A	I/O	Buffer	Extra I/O

Pin	Active	Direction	Connection	Use
IO143/DIFFIO_R10p	N/A	I/O	Buffer	Extra I/O
IO144/DIFFIO_R9n/ DEV_OE	N/A	I/O	Buffer	Extra I/O
IO145/DIFFIO_R9p/ DEV_CLRn	High	Output	Buffer	Buffer direction
IO146	High	Input	Display	Display touch panel right
IO147/DIFFIO_R7n	High	Input	Display	Display touch panel down
IO148/DIFFIO_R7p	High	Input	Display	Display touch panel left
IO159/DIFFIO_R5n/ INIT_DONE	N/A	Output	VCCIO	Status pin
IO160/DIFFIO_R5p/ CRC_ERROR	High	Input	Display	Display touch panel up
IO161	Low	Input	Reset chip	Reset watchdog output
IO162/DIFFIO_R4n/nCEO	Low	Input	Reset chip	Reset signal
IO164/DIFFIO_R4p/ CLKUSR	Low	Input	Reset chip	Reset power-fail output
IO166	N/A	Output	Reset chip	Reset watchdog input
IO167/DIFFIO_R3n	N/A	Output	Display	Display DCLK
IO168/DIFFIO_R3p	N/A	Output	Display	Display HSYNC signal
IO169/VREFB6N0	N/A	Output	Display	Display VSYNC signal
IO171	High	Output	Display	Display data enable
IO173	N/A	Output	Buffer	Extra I/O
IO176/DIFFIO_R1n	N/A	Output	Buffer	Extra I/O
IO177/DIFFIO_R1p	N/A	Output	Buffer	Extra I/O
IO181	N/A	Input	Rotary encoder 1	Rotary A
IO182/DIFFIO_T23n	N/A	Input	Rotary encoder 1	Rotary B

Pin	Active	Direction	Connection	Use
IO183/DIFFIO_T23p	High	Input	Rotary encoder 1	Rotary Press
IO184	N/A	Input	Rotary encoder 2	Rotary A
IO185/PLL2_CLKOUTn	N/A	Input	Rotary encoder 2	Rotary B
IO186/PLL2_CLKOUTp	High	Input	Rotary encoder 2	Rotary Press
IO187/RUP4	N/A	Input	Buffer	Extra I/O
IO188/RDN4	N/A	Input	Buffer	Extra I/O
IO189	N/A	I/O	Buffer	Extra I/O
IO194	N/A	I/O	Buffer	Extra I/O
IO195/VREFB7N0	N/A	I/O	Buffer	Extra I/O
IO196/DIFFIO_T19n	N/A	I/O	Buffer	Extra I/O

IO197/DIFFIO_T19p	N/A	I/O	Buffer	Extra I/O
IO200	N/A	I/O	Buffer	Extra I/O
IO201	N/A	I/O	Buffer	Extra I/O
IO202/DIFFIO_T15n	N/A	I/O	Buffer	Extra I/O
IO203/DIFFIO_T15p	High	Output	Buffer	Buffer direction
IO207	High	Output	Buffer	Buffer direction
IO214	N/A	I/O	Buffer	Extra I/O
IO216	N/A	I/O	Buffer	Extra I/O
IO217	N/A	I/O	Buffer	Extra I/O
IO218/DIFFIO_T10n/ DATA2	N/A	I/O	None	Unused, leave unconnected
IO219/DIFFIO_T10p/ DATA3	N/A	I/O	None	Unused, leave unconnected

Pin	Active	Direction	Connection	Use
IO221/DATA4	N/A	I/O	None	Unused, leave unconnected
IO223/VREFB8N0	N/A	I/O	Buffer	Extra I/O
IO224	N/A	I/O	Buffer	Extra I/O
IO226/DATA5	N/A	I/O	None	Unused, leave unconnected
IO230/DIFFIO_T6n	N/A	I/O	DRAM	ADC data bus
IO231/DIFFIO_T6p/ DATA6	N/A	I/O	None	Unused, leave unconnected
IO232/DATA7	N/A	I/O	None	Unused, leave unconnected
IO233	N/A	I/O	DRAM	ADC data bus
IO234/DIFFIO_T2n	N/A	I/O	DRAM	ADC data bus
IO235/DIFFIO_T2p	N/A	I/O	DRAM	ADC data bus
IO236	N/A	I/O	DRAM	ADC data bus
IO237/DIFFIO_T1n	N/A	I/O	DRAM	ADC data bus
IO238/DIFFIO_T1p	N/A	I/O	DRAM	ADC data bus
IO239/PLL3_CLKOUTn	N/A	I/O	DRAM	ADC data bus
IO240/PLL3_CLKOUTp	High	Output	ADC	ADC clock input

The final product included several changes. Pin 162, an output for device configuration, was left floating, and the reset input was changed to Pin 187. One buffer was also left unusable, resulting in rewiring to extra I/O pins. The pin assignments are shown below.

To	Direction	Location	I/O Bank	VREF Group	Fitter Location
ADC_clk	Output	PIN_240	8	B8_N0	PIN_240
ADC_db[7]	Input	PIN_230	8	B8_N0	PIN_230
ADC_db[6]	Input	PIN_233	8	B8_N0	PIN_233
ADC_db[5]	Input	PIN_234	8	B8_N0	PIN_234
ADC_db[4]	Input	PIN_235	8	B8_N0	PIN_235
ADC_db[3]	Input	PIN_236	8	B8_N0	PIN_236
ADC_db[2]	Input	PIN_237	8	B8_N0	PIN_237
ADC_db[1]	Input	PIN_238	8	B8_N0	PIN_238
ADC_db[0]	Input	PIN_239	8	B8_N0	PIN_239
altera_reserved_tck	Input				PIN_27
altera_reserved_tdi	Input				PIN_26
altera_reserved_tdo	Output				PIN_29
altera_reserved_tms	Input				PIN_28
Clock	Input	PIN_31	1	B1_N0	PIN_31
LCD_dclk	Output	PIN_167	6	B6_N0	PIN_167
LCD_DE	Output	PIN_171	6	B6_N0	PIN_171
LCD_hsync	Output	PIN_168	6	B6_N0	PIN_168
LCD_vsync	Output	PIN_169	6	B6_N0	PIN_169
PFO	Input	PIN_164	6	B6_N0	PIN_164
RAMROM_ab[18]	Output	PIN_39	2	B2_N0	PIN_39
RAMROM_ab[17]	Output	PIN_37	2	B2_N0	PIN_37
RAMROM_ab[16]	Output	PIN_38	2	B2_N0	PIN_38
RAMROM_ab[15]	Output	PIN_44	2	B2_N0	PIN_44
RAMROM_ab[14]	Output	PIN_41	2	B2_N0	PIN_41
RAMROM_ab[13]	Output	PIN_49	2	B2_N0	PIN_49
RAMROM_ab[12]	Output	PIN_43	2	B2_N0	PIN_43
RAMROM_ab[11]	Output	PIN_52	2	B2_N0	PIN_52
RAMROM_ab[10]	Output	PIN_55	2	B2_N0	PIN_55
RAMROM_ab[9]	Output	PIN_51	2	B2_N0	PIN_51
RAMROM_ab[8]	Output	PIN_50	2	B2_N0	PIN_50
RAMROM_ab[7]	Output	PIN_45	2	B2_N0	PIN_45
RAMROM_ab[6]	Output	PIN_46	2	B2_N0	PIN_46
RAMROM_ab[5]	Output	PIN_6	1	B1_N0	PIN_6
RAMROM_ab[4]	Output	PIN_9	1	B1_N0	PIN_9
RAMROM_ab[3]	Output	PIN_13	1	B1_N0	PIN_13
RAMROM_ab[2]	Output	PIN_18	1	B1_N0	PIN_18
RAMROM_ab[1]	Output	PIN_21	1	B1_N0	PIN_21
RAMROM_ab[0]	Output	PIN_22	1	B1_N0	PIN_22
RAMROM_db[7]	Bidir	PIN_69	3	B3_N0	PIN_69
RAMROM_db[6]	Bidir	PIN_70	3	B3_N0	PIN_70
RAMROM_db[5]	Bidir	PIN_71	3	B3_N0	PIN_71
RAMROM_db[4]	Bidir	PIN_72	3	B3_N0	PIN_72
RAMROM_db[3]	Bidir	PIN_73	3	B3_N0	PIN_73

RAMROM_db[2]	Bidir	PIN_68	3	B3_N0	PIN_68
RAMROM_db[1]	Bidir	PIN_65	3	B3_N0	PIN_65
RAMROM_db[0]	Bidir	PIN_64	3	B3_N0	PIN_64
Reset	Input	PIN_187	7	B7_N0	PIN_187
ROM_CS	Output	PIN_5	1	B1_N0	PIN_5
ROM_OE	Output	PIN_4	1	B1_N0	PIN_4
romram_db_dir	Output	PIN_76	3	B3_N0	PIN_76
rotary_1_A	Input	PIN_181	7	B7_N0	PIN_181
rotary_1_B	Input	PIN_182	7	B7_N0	PIN_182
rotary_1_P	Input	PIN_183	7	B7_N0	PIN_183
rotary_2_A	Input	PIN_184	7	B7_N0	PIN_184
rotary_2_B	Input	PIN_185	7	B7_N0	PIN_185
rotary_2_P	Input	PIN_186	7	B7_N0	PIN_186
SRAM_CS	Output	PIN_63	3	B3_N0	PIN_63
SRAM_OE	Output	PIN_57	2	B2_N0	PIN_57
SRAM_WE	Output	PIN_56	2	B2_N0	PIN_56
VRAM_ab[8]	Output	PIN_132	5	B5_N0	PIN_132
VRAM_ab[7]	Output	PIN_126	5	B5_N0	PIN_126
VRAM_ab[6]	Output	PIN_131	5	B5_N0	PIN_131
VRAM_ab[5]	Output	PIN_128	5	B5_N0	PIN_128
VRAM_ab[4]	Output	PIN_127	5	B5_N0	PIN_127
VRAM_ab[3]	Output	PIN_120	4	B4_N0	PIN_120
VRAM_ab[2]	Output	PIN_119	4	B4_N0	PIN_119
VRAM_ab[1]	Output	PIN_118	4	B4_N0	PIN_118
VRAM_ab[0]	Output	PIN_133	5	B5_N0	PIN_133
VRAM_ab_dir	Output	PIN_117	4	B4_N0	PIN_117
VRAM_CAS	Output	PIN_134	5	B5_N0	PIN_134
VRAM_control_dir	Output	PIN_145	5	B5_N0	PIN_145
VRAM_db[7]	Bidir	PIN_80	3	B3_N0	PIN_80
VRAM_db[6]	Bidir	PIN_78	3	B3_N0	PIN_78
VRAM_db[5]	Bidir	PIN_82	3	B3_N0	PIN_82
VRAM_db[4]	Bidir	PIN_81	3	B3_N0	PIN_81
VRAM_db[3]	Bidir	PIN_84	3	B3_N0	PIN_84
VRAM_db[2]	Bidir	PIN_83	3	B3_N0	PIN_83
VRAM_db[1]	Bidir	PIN_88	3	B3_N0	PIN_88
VRAM_db[0]	Bidir	PIN_87	3	B3_N0	PIN_87
VRAM_db_dir	Output	PIN_93	4	B4_N0	PIN_93
VRAM_DSF	Output	PIN_173	6	B6_N0	PIN_173
VRAM_OE	Output	PIN_139	5	B5_N0	PIN_139
VRAM_RAS	Output	PIN_135	5	B5_N0	PIN_135
VRAM_SC	Output	PIN_142	5	B5_N0	PIN_142
VRAM_SE	Output	PIN_143	5	B5_N0	PIN_143
VRAM_WE	Output	PIN_137	5	B5_N0	PIN_137
WDI	Output	PIN_166	6	B6_N0	PIN_166
WDO	Input	PIN_161	6	B6_N0	PIN_161

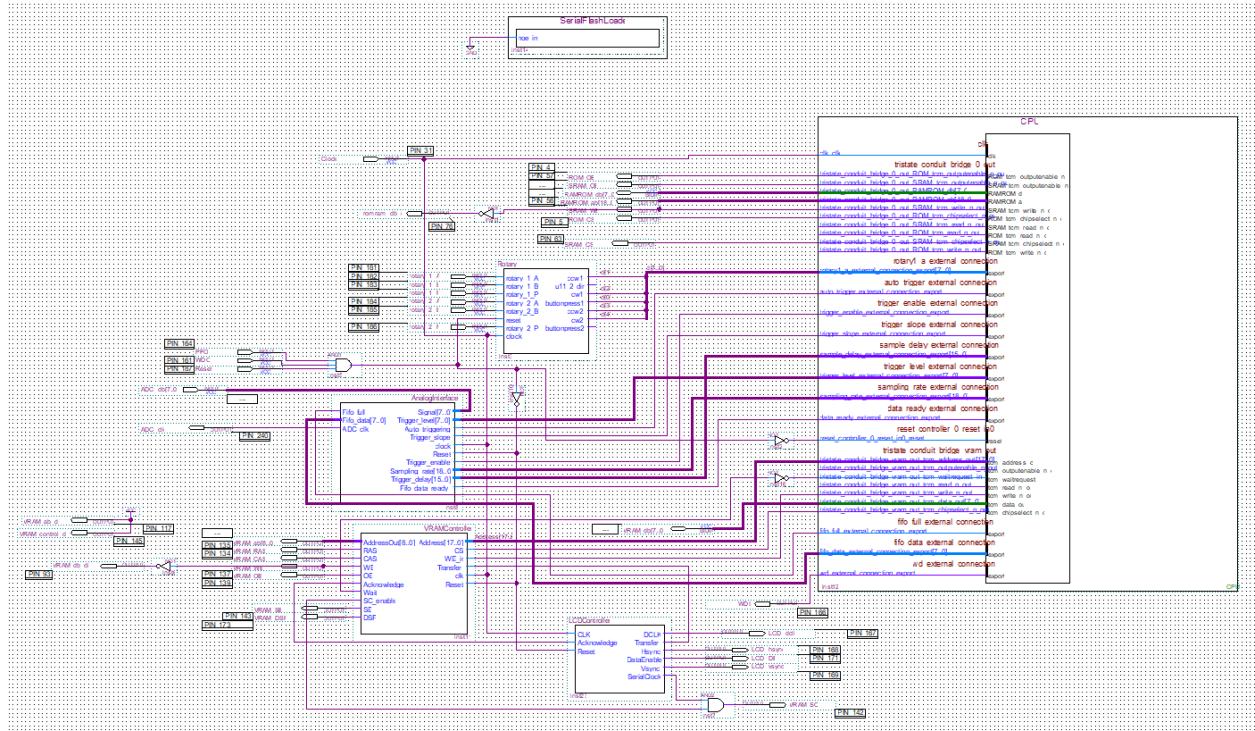


Figure 6: The main FPGA block diagram with the CPU, inputs and outputs, and controllers.

Logic: A CPU and several controllers were programmed into the FPGA, as seen in Figure 2. These will be discussed in greater detail in their respective subsections. As seen in the main block diagram, the peripheral memory, user interface elements, and other elements all interact with the FPGA and its various controllers. The input signal and settings are sent to the CPU and the appropriate controllers, and the output data is sent from the CPU to be displayed on the LCD.

2.1.3 Power

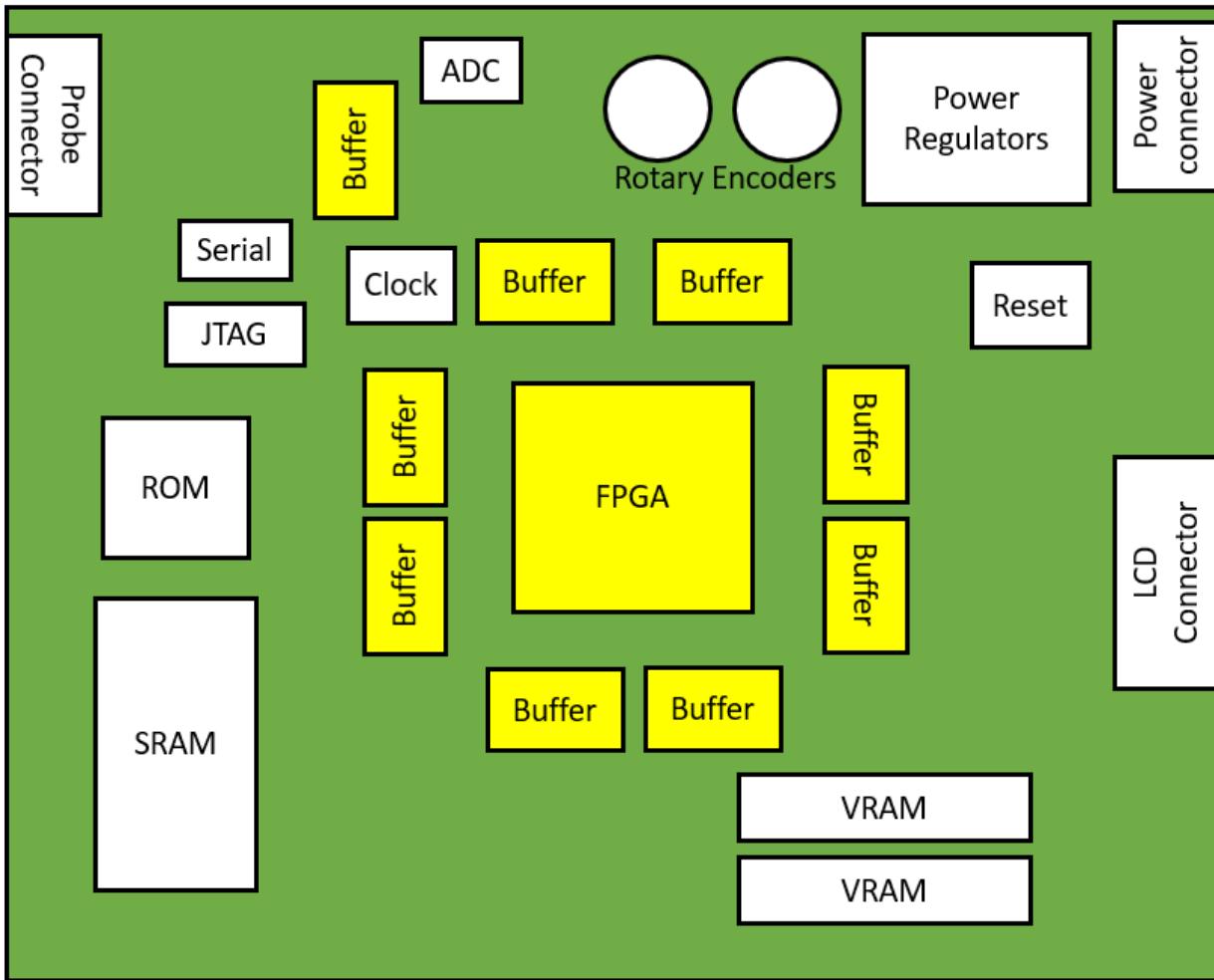


Figure 7: Board Layout: Power circuitry.

The full power schematics can be found at Appendix C.

A din4 connector (PWR1) is used to supply the board with +/- 12 V and 5 V. All chips are locally bypassed.

5 V is used to power the reset, VRAM, SRAM, ROM, and ADC chips. Several LM1086 regulators are used to generate the other required voltages. Specifically, U14 uses the 5 V supply to generate 2.5 V for analog power to the FPGA PLLs, and U15 uses the 5 V supply to generate 1.2 V for FPGA power supply and digital power to the FPGA PLLs. U16 generates 3.3 V from 5 V, which is used to power the FPGA I/O supply pins, buffers,

JTAG, serial configuration device, clock, and for pulling high the rotary encoder outputs.

Regulators also generate 5 V from 12 V (U24) and 4 V from 5 V (U23) for ADC reference voltages. The main power schematics can be seen in Figure ??.

A boost converter is used to generate the 25 V necessary for the LCD backlight. U21 (AP3012) steps up from 5 V and outputs 25 V (Figure 8).

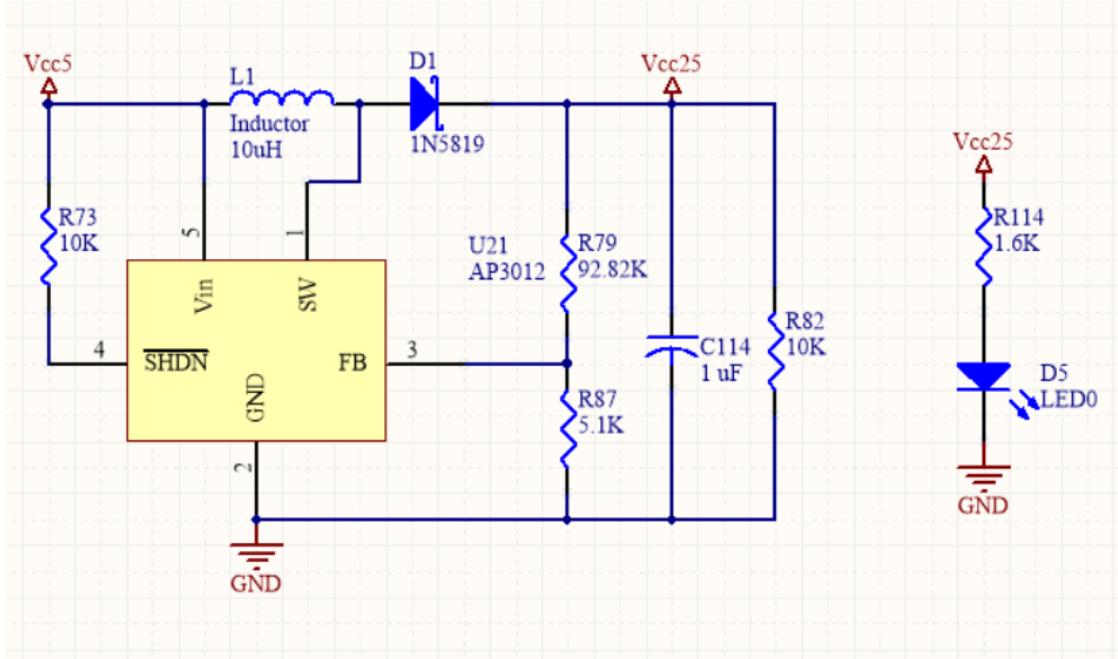


Figure 8: Boost converter circuit for LCD backlight power supply.

2.1.4 Memory

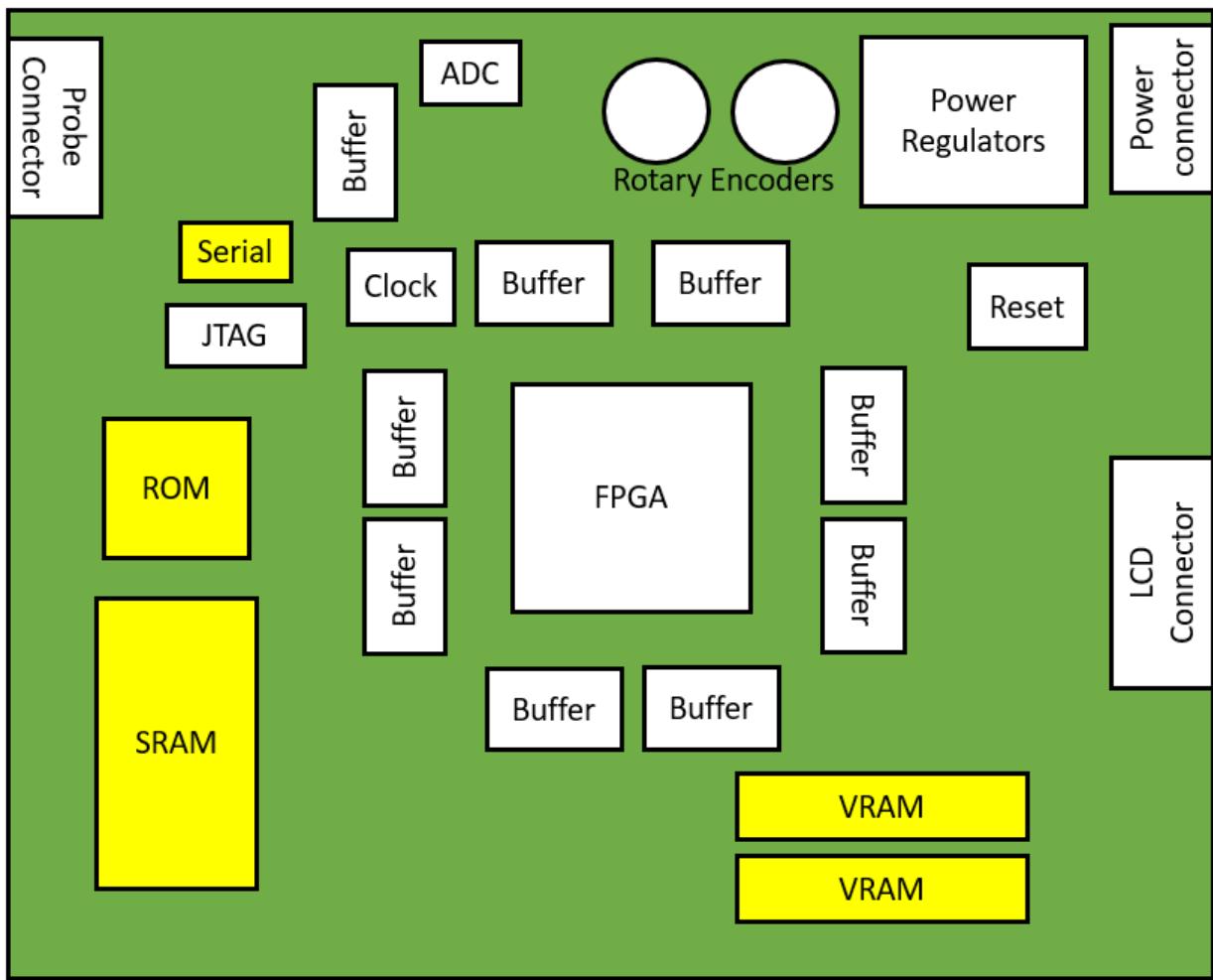


Figure 9: Board Layout: Memory.

Memory Map:

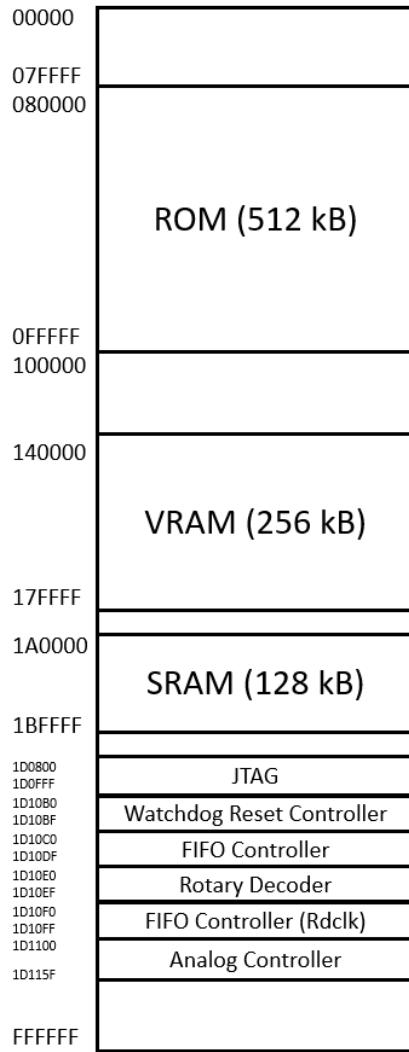


Figure 10: Memory map diagram

Several types of storage were used, including ROM (U2) for permanent program storage, and a smaller amount of static RAM (U1) for storage including the data segment, stack, and heap. Video RAM (U19, U22), with dynamic RAM, which is cheaper and smaller than static RAM, and serial memory, was used for the display memory. A serial device (U4) was also used to store configuration data for the FPGA after powering on.

ROM: 1 512 K x 8 bit EEROM (U2, Am29F040) was used for program storage. 8 data bits are connected in parallel through the buffers to the CPU, along with 19 address bits, an active-low chip enable (CE), and output enable (OE) signal. The write enable (WE)

signal was pulled up, as the ROM was written to with a dedicated ROM programmer.

The ROM timing diagram for the read cycle can be seen at Appendix A, Figure 43. 3 read wait states were used.

The executable file from the Nios II IDE was loaded into the ROM using a ROM programmer.

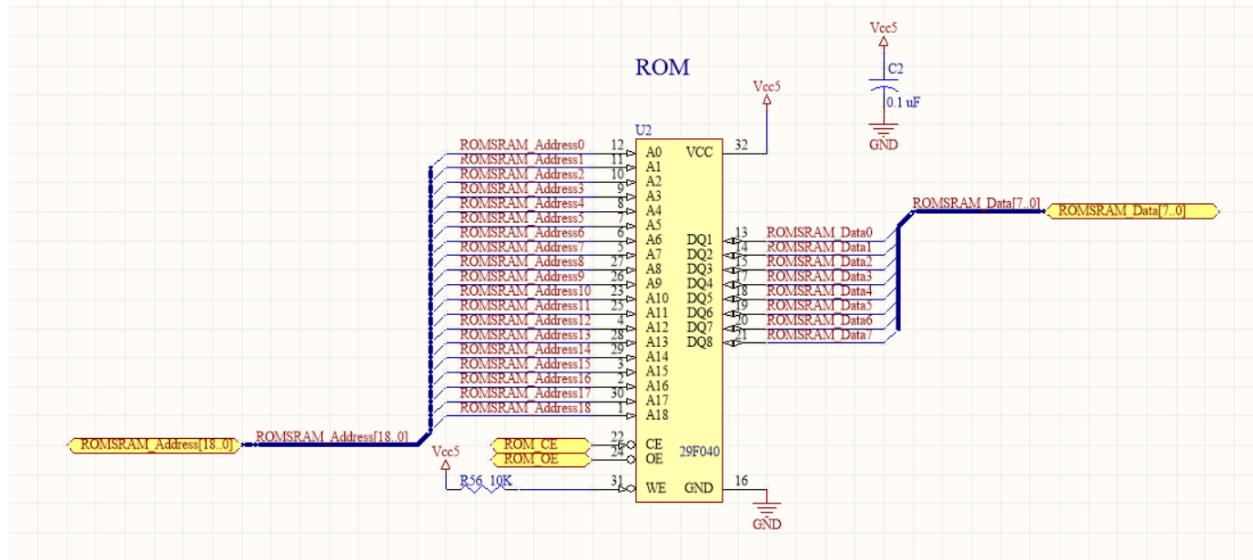


Figure 11: ROM Schematic.

SRAM: 1 128 K x 8 bit CMOS static RAM was used (U1, HM628128B)

The address line is shared between the ROM and SRAM, with only the 17 least significant bits used for RAM addressing. The 8-bit data bus is also shared between the ROM and SRAM. The active-low output enable (OE), write enable (WE), and chip select (CS1) are also connected between the SRAM and CPU, with the second active high chip select (CS2) pulled high as required by the read and write cycles.

The SRAM timing diagrams can be seen at Appendix A, Figures 44 and 45. 2 read wait states and 1 write wait state was used.

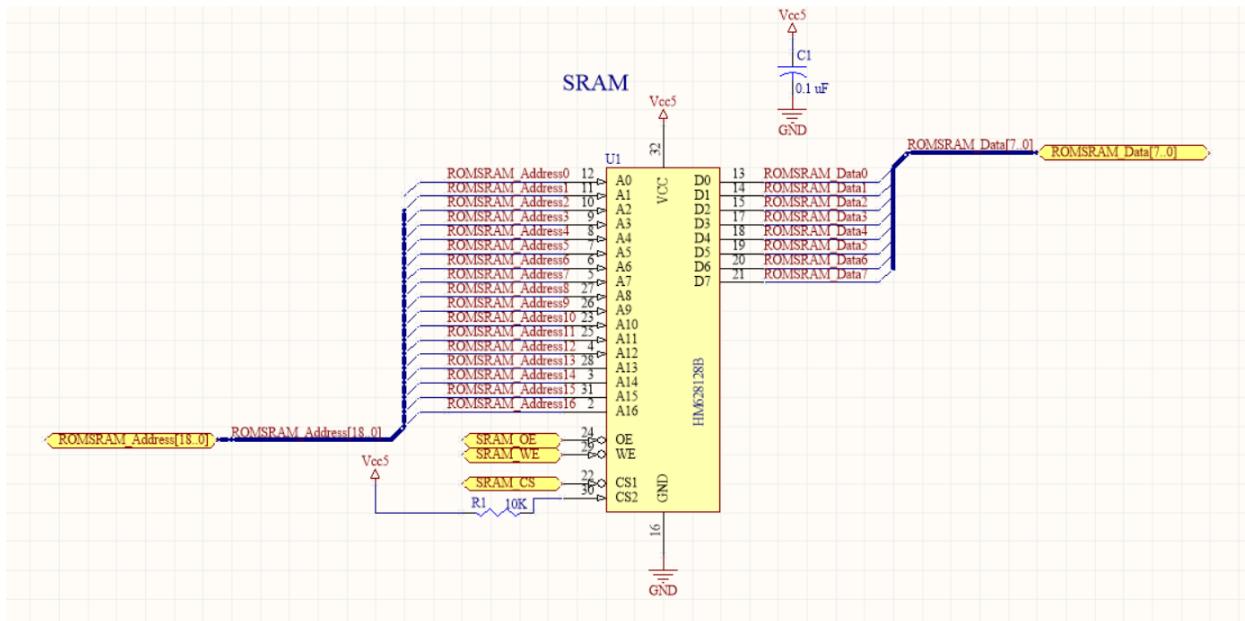


Figure 12: SRAM Schematic.

VRAM: See Section 2.1.7 for details on the VRAM.

Serial Device: A serial configuration device (U4, EPSCS16), a 2 MB flash memory device, was used to store FPGA configuration data ².

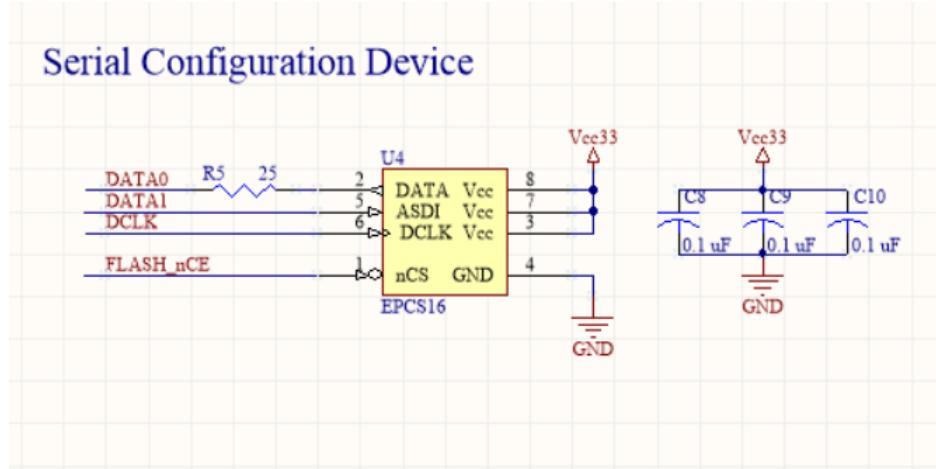


Figure 13: Serial Device Schematic.

² Was unable to successfully use serial device for unknown reasons, possibly due to incorrect FPGA configurations.

2.1.5 Analog System

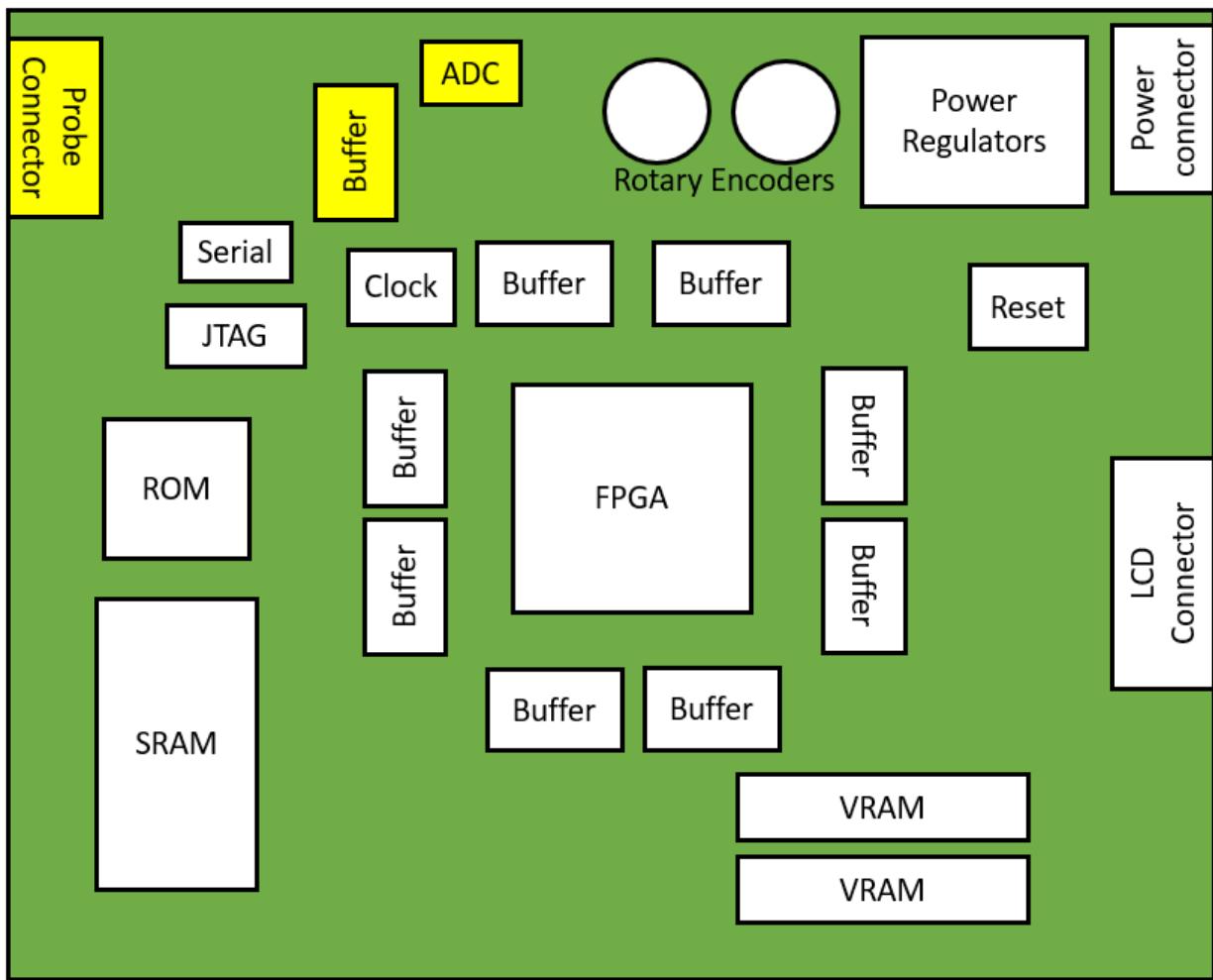


Figure 14: Board Layout: Analog circuitry.

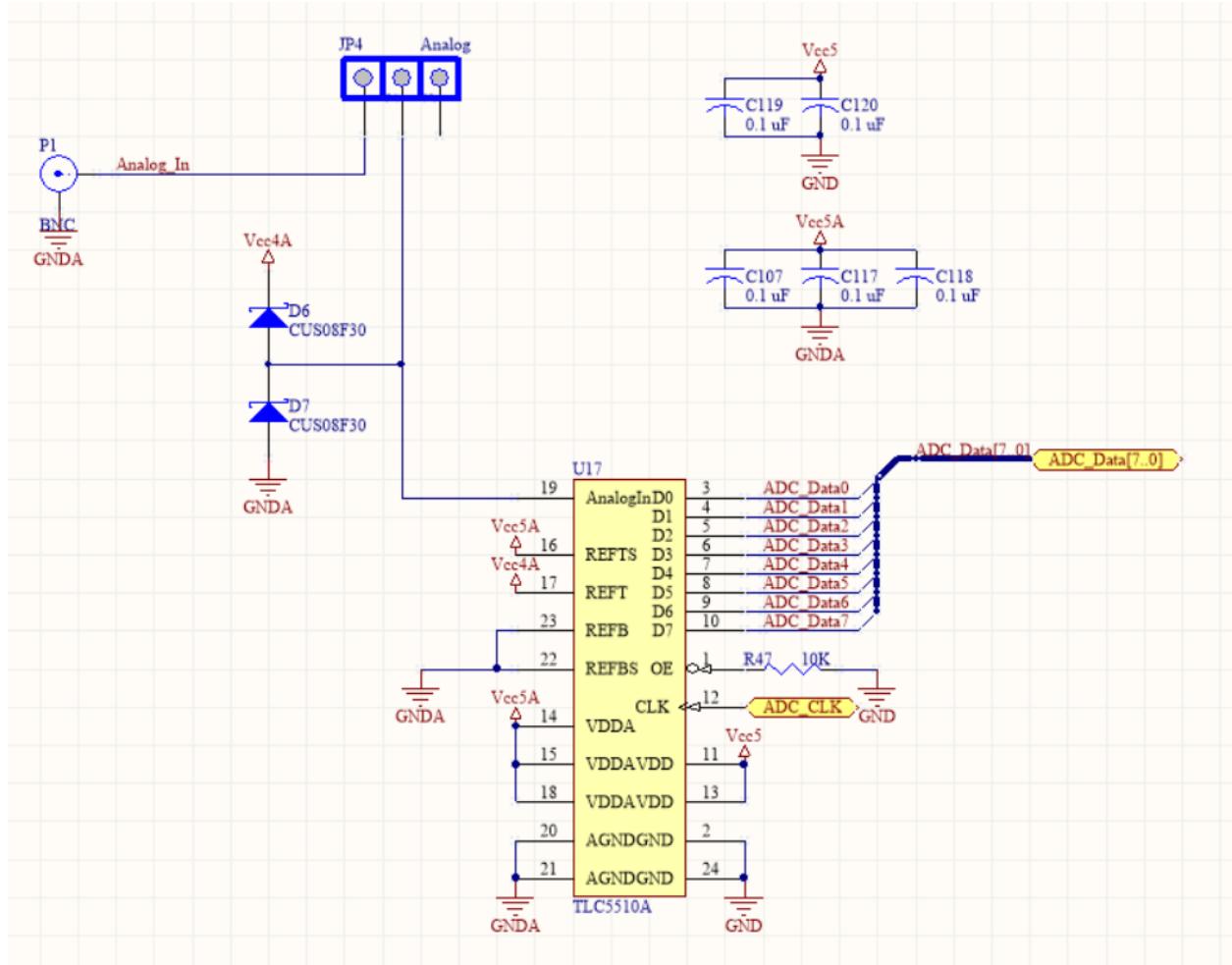


Figure 15: ADC circuit schematic.

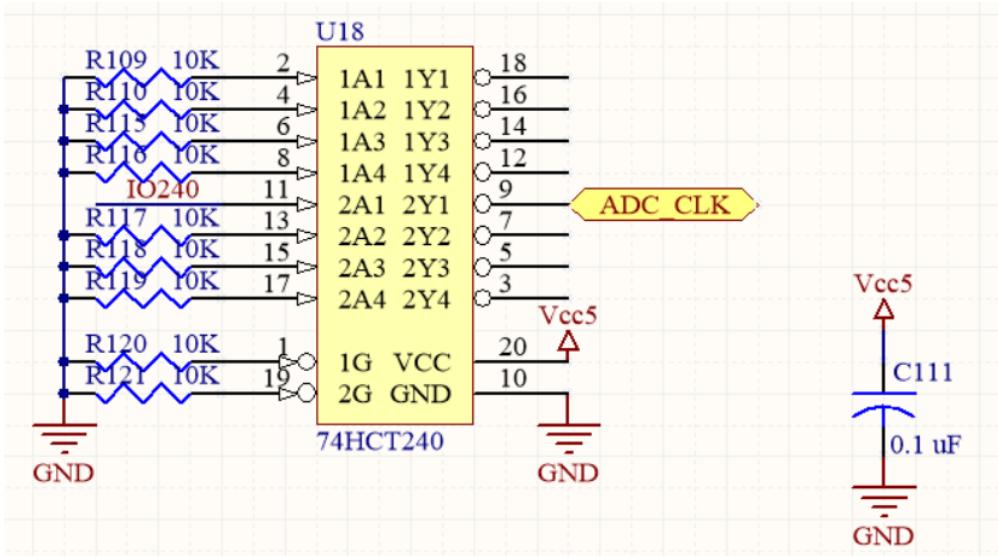


Figure 16: Buffer for FPGA to ADC signals.

Summary: An analog signal is inputted through the BNC connector (P1). A jumper connector was used in case additional analog front end circuitry was used. The signal then passes through protection diodes and is sent to the ADC. The data is then sent through the buffers to the FPGA and analog controller, which stores the signal at the sampling rate once it has been triggered.

ADC: The analog input can range from 0 to 4 V. An analog-to-digital converter (U17, TLC5510A) was used to digitize the input signal to 8 bits. This specific ADC was chosen because of the simplicity of the circuit needed in order to meet the 0-3 V requirement.

The 8 bit data bus was connected to the CPU in parallel, and a clock input was generated from the analog controller. A separate buffer was required to meet the ADC input voltage requirements (U18 in Figure 16). Protection diodes (D6, D7 in Figure 15) with a low forward voltage (about 0.2 V) were used to keep the analog input within the allowed input range.

Several analog supply voltages were used to isolate the analog and digital circuitry. An external 4 V analog reference was generated and used for a 0-4 V input signal range. A 5 V analog power supply was also required, along with an analog ground reference connected to digital ground through a power inductor (Part L2, Figure ??).

Analog logic: The analog controller consists of a trigger controller to determine

when a trigger has occurred, along with logic for storing the signal at the specified sampling rate. If a trigger occurs, or if the auto triggering times out, the signal is stored in a FIFO buffer to be clocked out and displayed.

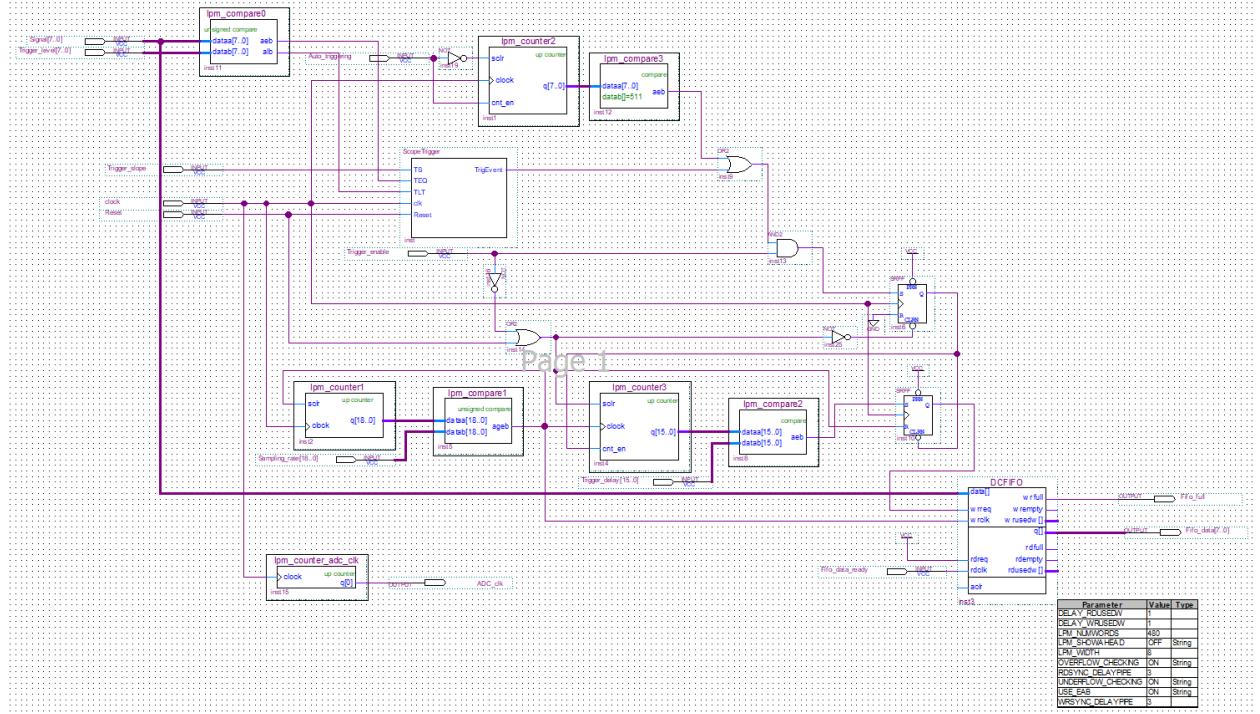


Figure 17: Analog controller block diagram.

Analog input settings: Several values are taken as outputs from the CPU and are used in the hardware logic. Specifically, an 8 bit trigger level, 1 bit auto triggering, 1 bit trigger slope, 1 bit trigger enable, 19 bit sampling rate, and 16 bit trigger delay signal are used as set by the user.

Trigger logic: First, the system determines whether or not a trigger has occurred. A Moore state machine (the ScopeTrigger block in Figure 17) is used to determine whether or not a trigger has occurred. The trigger event output is set high when the trigger slope is positive and the signal has transitioned from below to above the trigger level, or if the trigger slope is negative and the signal has transitioned from above to below the trigger level. The logic description can be found at Appendix ??.

In order to do this, the 8-bit signal is compared to the trigger level in lpm_compare0 in Figure 17, and the outputs are sent to

the trigger state machine.

Autotriggering: The system also generates a trigger event after 512 clocks if auto trigger is enabled, which was arbitrarily chosen as a short amount of time. Future designs may be improved by correlating the auto trigger countdown to the number of samples and the sampling rate.

An SRFF is set once a trigger event has occurred, and the controller is reset to wait for another trigger event when the trigger enable signal is disabled and re-enabled (set low then high).

FIFO buffer storage: A FIFO buffer is used to store the signal once a trigger event has occurred (DCFIFO in Figure 17). The FIFO has a size of 480 (the width of the LCD) x 8 bits of data. Once a trigger event has occurred, a counter and comparator are used to create the trigger delay, and an SRFF is set once the delay has been reached to enable writing to the FIFO. A counter and comparator are used to create a clock at the given sampling rate, which is sent to the write clock of the FIFO. The signal is then written into a FIFO buffer at the sampling rate once the trigger delay has finished, and is clocked out by the CPU when it is full.

Analog outputs: As seen in Figure 17, the Fifo_full signal indicates when the FIFO is full and the sample has been stored, the Fifo_data bus contains the stored signal data. The Fifo_data_ready input is controlled from the CPU, which sends the clock necessary to read out the signal from the data bus.

The ADC clock (ADC_clk) is also outputted from the controller to the ADC. It is half the 24 MHz system clock (or a 12 MHz clock) because of the maximum ADC conversion rate, which is 20 M samples per second.

2.1.6 Rotary Encoders

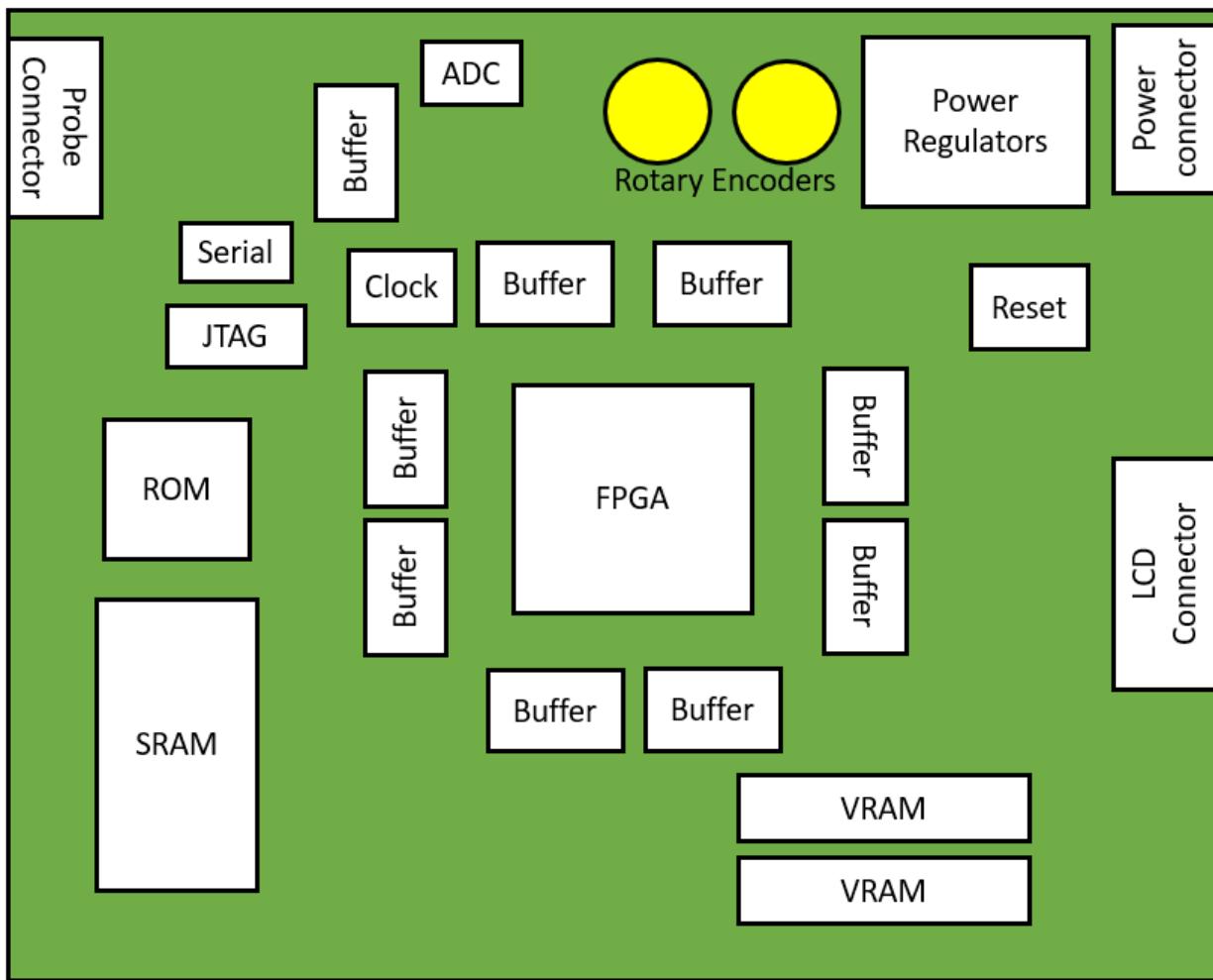


Figure 18: Board Layout: Rotary Encoders.

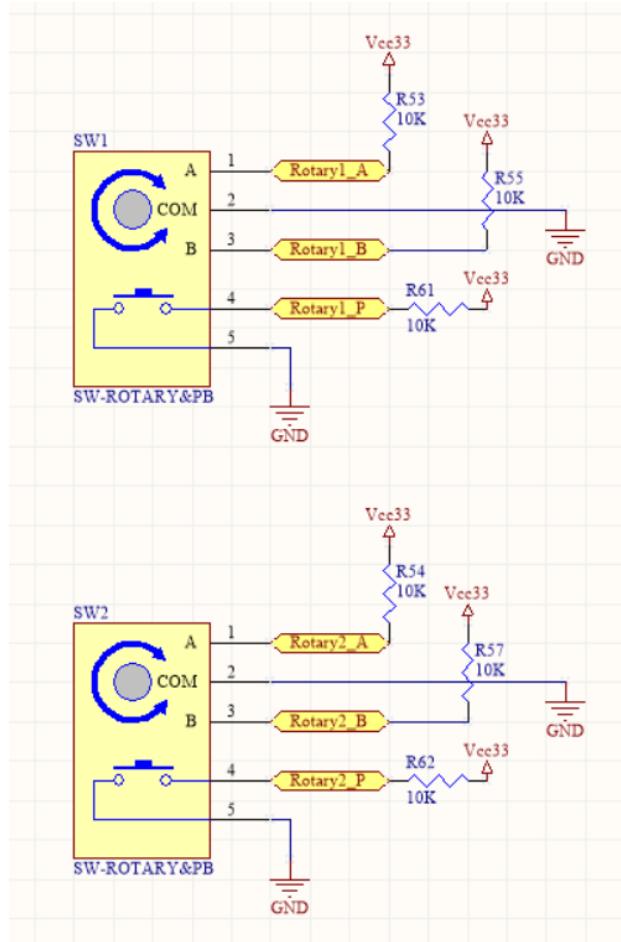


Figure 19: Rotary encoders schematics.

Two rotary encoders are used for the user input (SW1, SW2). These include and A and B output for the encoder and a single pull single throw push button switch.

Pins A and B are pulled high³, while the common pin was grounded to create out-of-phase pulses from outputs A and B.

Rotary decoder logic:

³ 1 K resistors were used in place of the 10 K resistors on the schematic because the voltage high output was too low for the buffers.

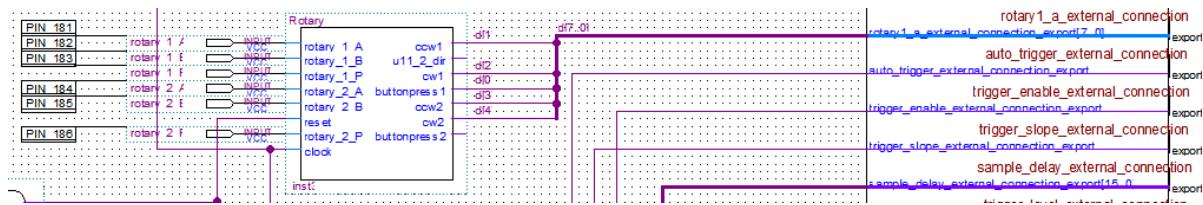


Figure 20: Connections between CPU and rotary controller.

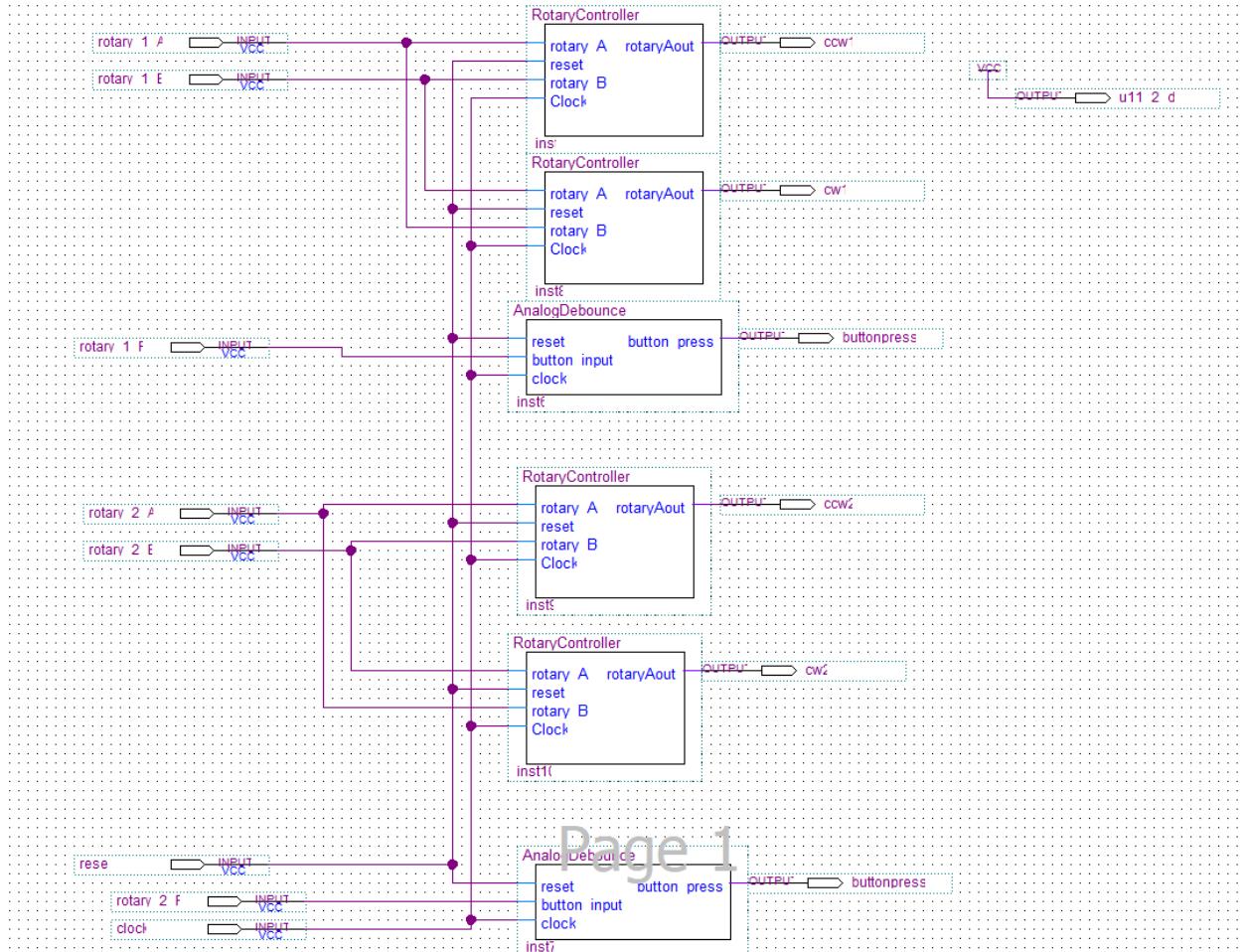


Figure 21: Rotary controller.

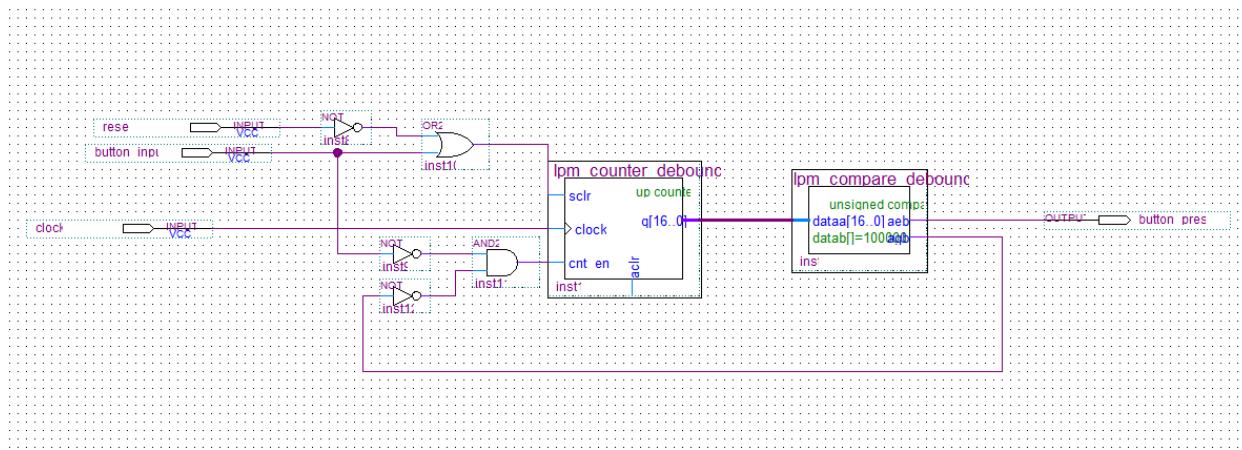


Figure 22: Debouncer for push button.

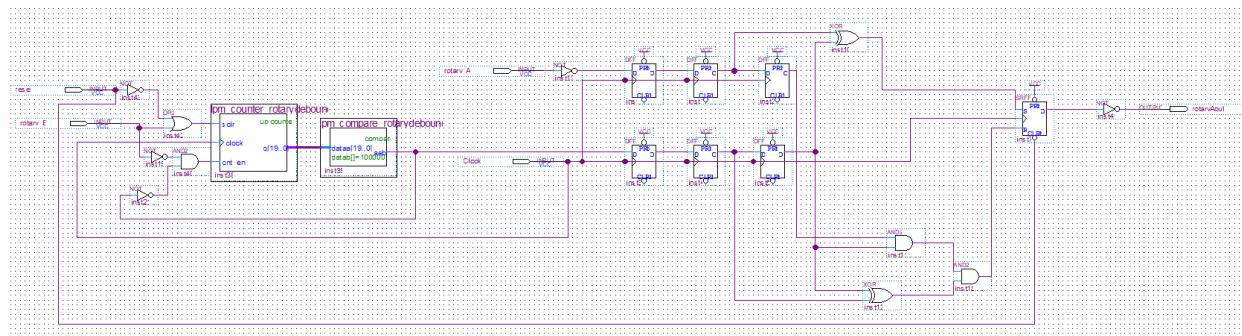


Figure 23: Rotary decoder.

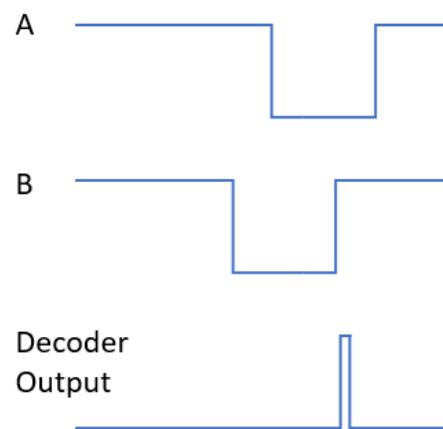


Figure 24: Output of rotary decoder, counterclockwise turn.

The switch input and the two out-of-phase signals from each rotary encoder are decoded with the rotary controller. The rotary logic block consists of a debouncer for the push button and a decoder for the rotary encoder.

The debouncer (Figure 22) takes as inputs the active-low button input (Rotary1_P and Rotary2_P on the schematic), clock, and active-low reset signal. It outputs a single active-high clock pulse whenever the button input signal is low for longer than 100000 clocks, which was chosen through testing the switches.

The decoder (Figure 23) takes as inputs the two active-low outputs from the encoder (Rotary1_A and Rotary1_B, and Rotary2_A and Rotary2_B on the schematic), and the reset and clock signals, and outputs a single active-high clock pulse for each counter-clockwise turn of the rotary encoder, where output A leads B (Figure 24).

This is accomplished by first debouncing the input using a counter and comparator to eliminate glitches. Several DFFs are used in series in order to read inputs A and B from two consecutive clocks to determine when edges have occurred. An SRFF is set low when input B goes from low to high while input A is still low, and is set high otherwise. The inverse of this is returned as the output, resulting in a decoder for one direction. Two decoders are used for each rotary encoder to distinguish clockwise and counterclockwise turns (Figure 21).

2.1.7 VRAM and LCD

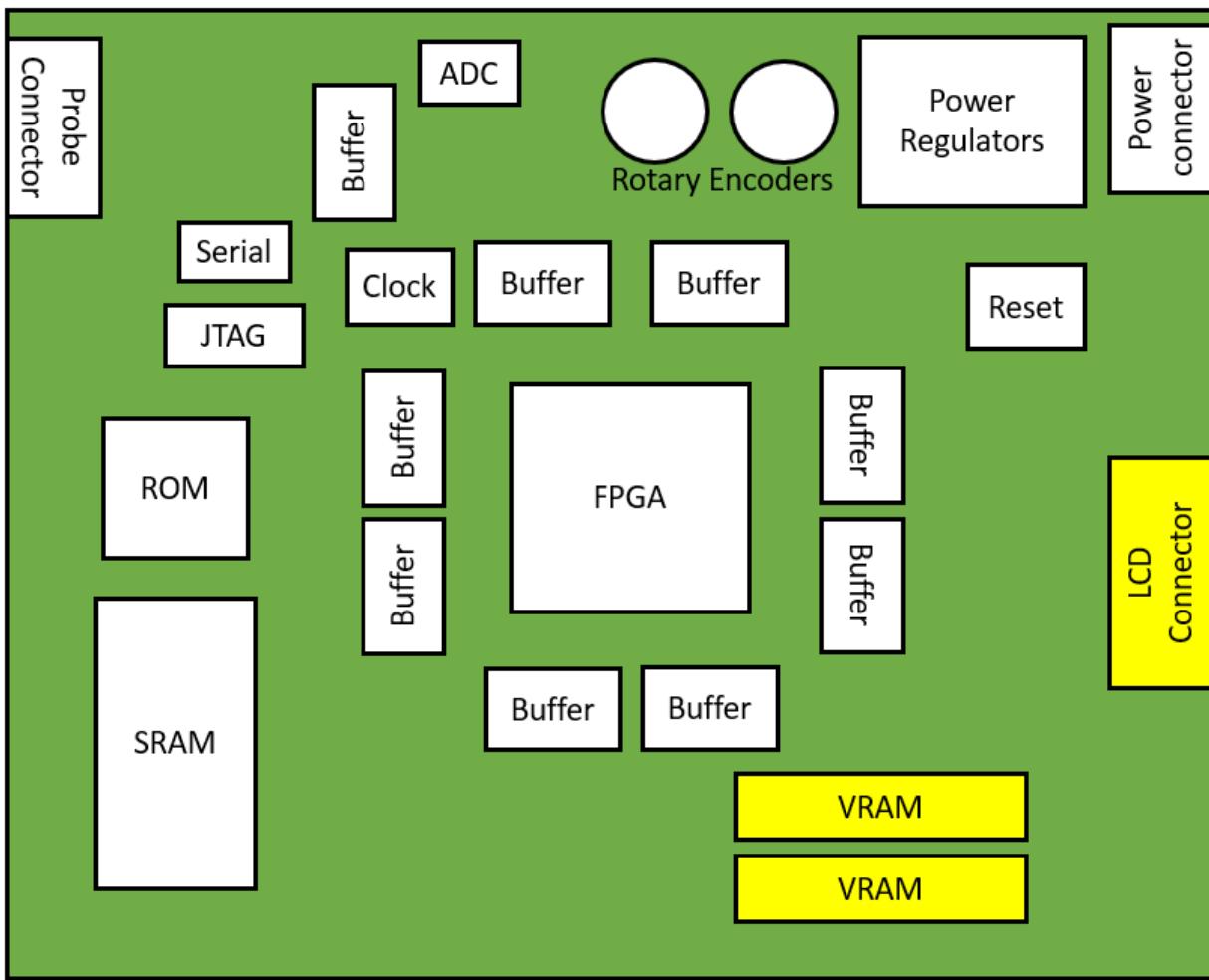


Figure 25: Board Layout: Display.

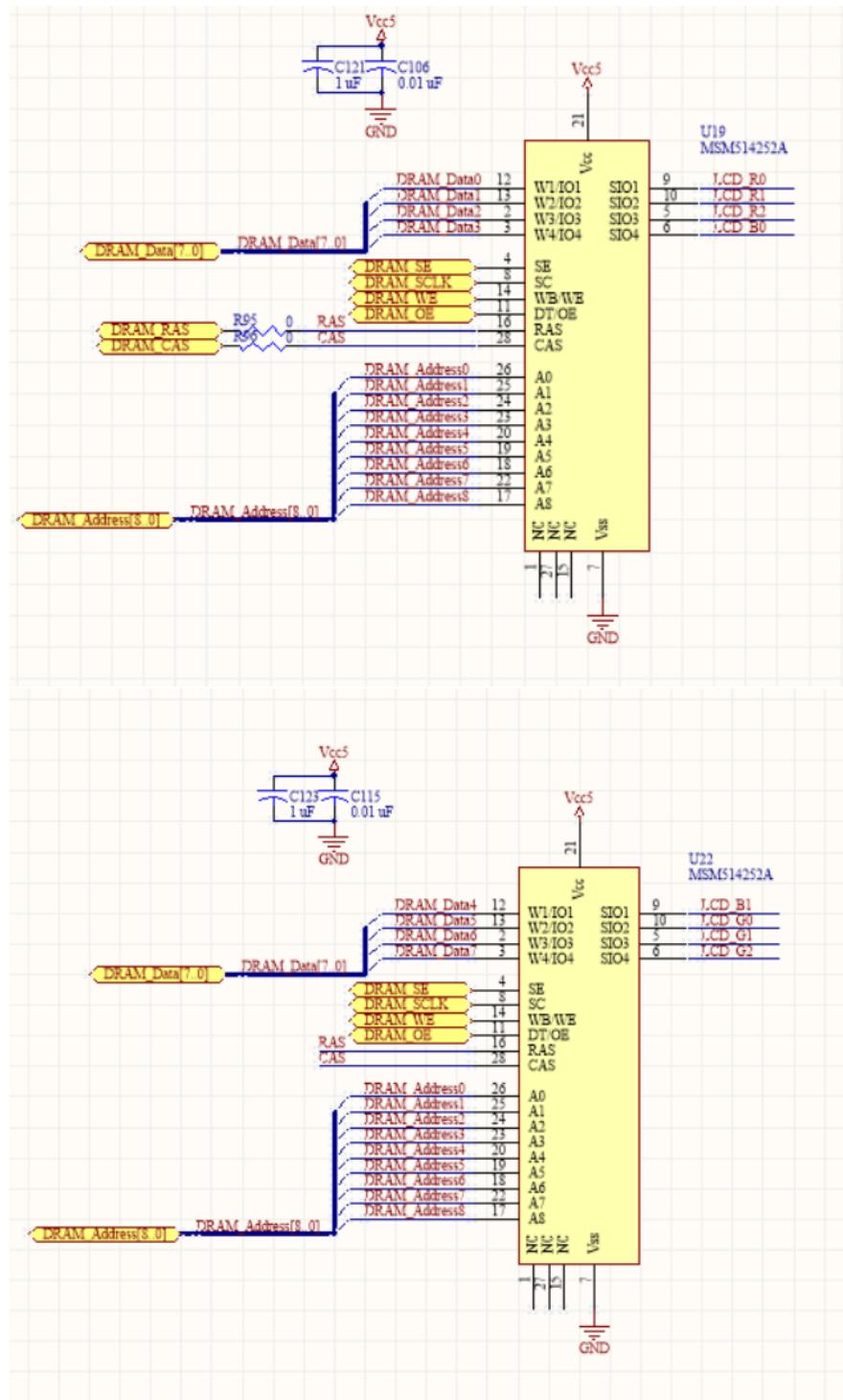


Figure 26: VRAM schematics.

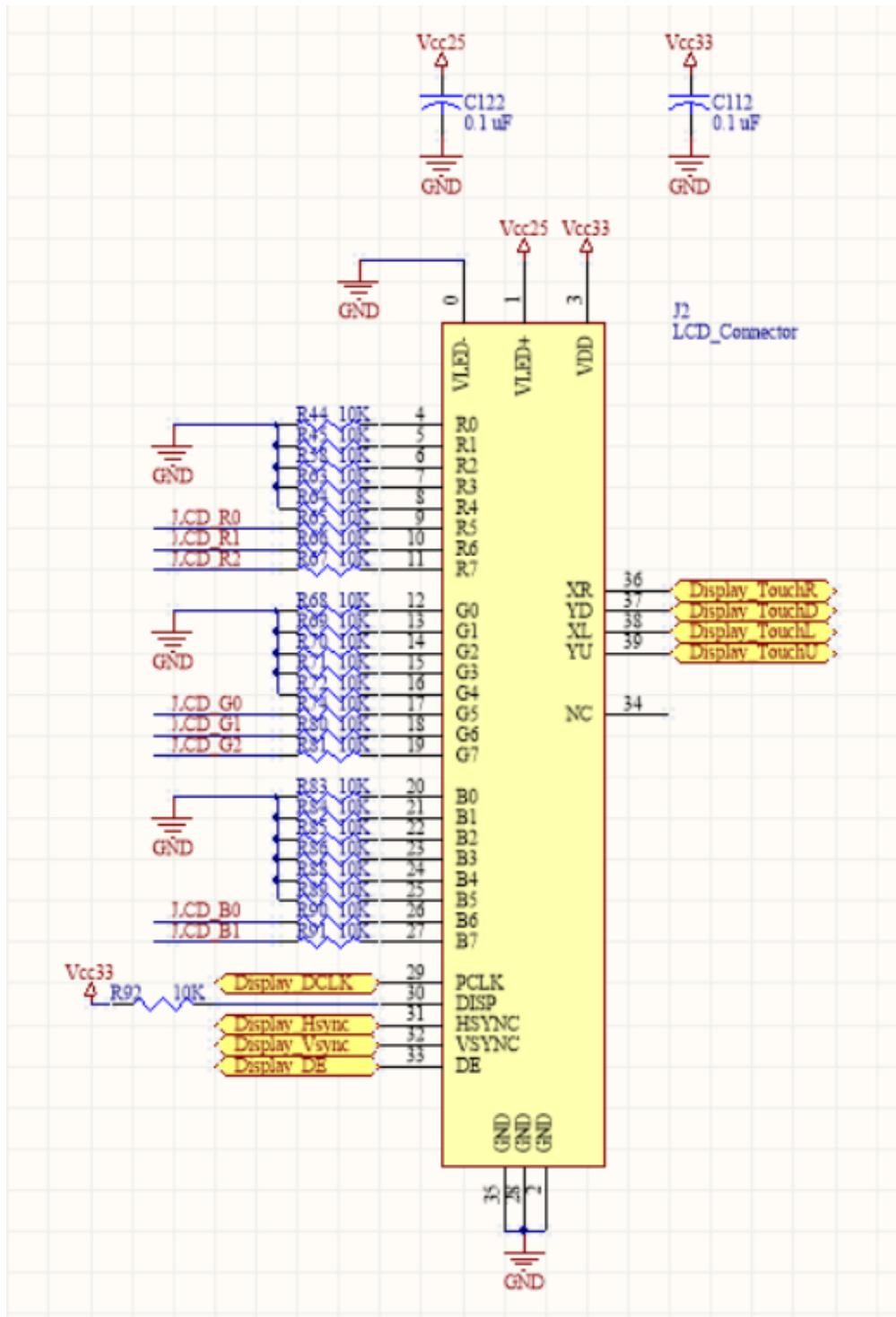


Figure 27: LCD connector schematics.

Two 256Kx4 DRAM with 512x4 SAM (Serial Access Memory) chips (U19, U22, MT42C4225) were used for a total of 8 bits of data, with a 480x272 RGB TFT LCD (Connector J2, ER-TFT043-3).

VRAM: Each VRAM sends 4 bits of serial data to the LCD, and receives 4 bits of data from the CPU, along with a 9 bit address bus, row address strobe (RAS), column address strobe (CAS), write enable (WE), and output enable (OE) signals from the VRAM controller. A serial clock signal is also sent to the VRAM. The LCD controller generates the various clock signals that are sent to synchronize the VRAM and LCD. The same address bus and control signals are used between the two VRAM, and each sends and receives 4 different data bits.

Resistors R95 and R96 are included in series in the RAS and CAS lines for termination.

The VRAM consists of the DRAM, transfer circuitry, and SAM. Read and write cycles are performed to read and write to the DRAM, and read transfer cycles are performed to transfer a row from the DRAM to SAM. Serial output cycles are performed to output the current row data from the SAM to the LCD. The VRAM is refreshed when no other operations are being performed in order to retain data.

LCD: The LCD receives 8 bits of data in parallel from the VRAM: 3 bits red, 3 bits green, and 2 bits blue, with the unused color bits pulled down. It also receives a horizontal sync, vertical sync, clock, and data enable signal from the LCD controller. The touch inputs were unused.

The LCD also required a 25V backlight supply (VLED on the schematic). A step-up converter (U21) was used, which is discussed in more detail in Section 2.1.3.

Logic overview: The VRAM-LCD system consists of a VRAM controller and LCD controller that interact with each other, the CPU, and the VRAM and LCD.

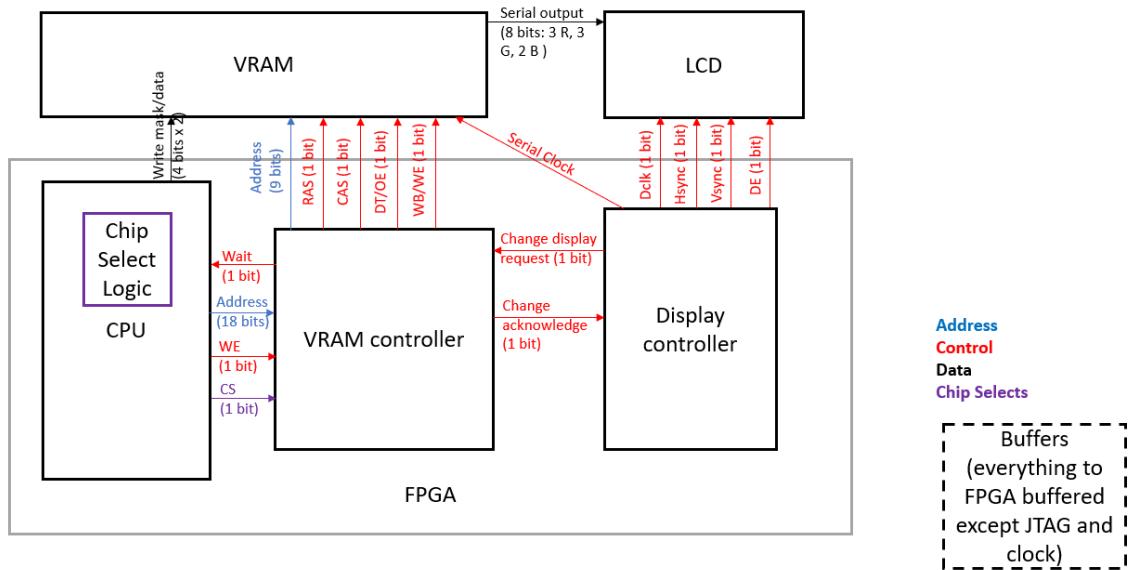


Figure 28: Block Diagram for VRAM and LCD.

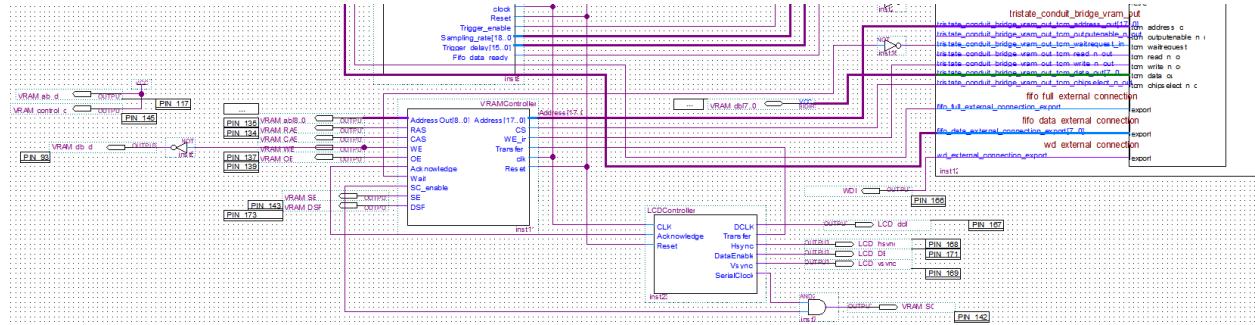


Figure 29: VRAM and LCD Controllers.

VRAM Controller: The VRAM controller takes several inputs from the CPU: 1-bit chip select and write enable signals, and an 18-bit address. It also takes as an input a transfer request signal from the display controller. A state machine is used to generate the output signals, the RAS, CAS, address selector, WE, OE, serial clock, CPU wait output, and transfer acknowledge signals.

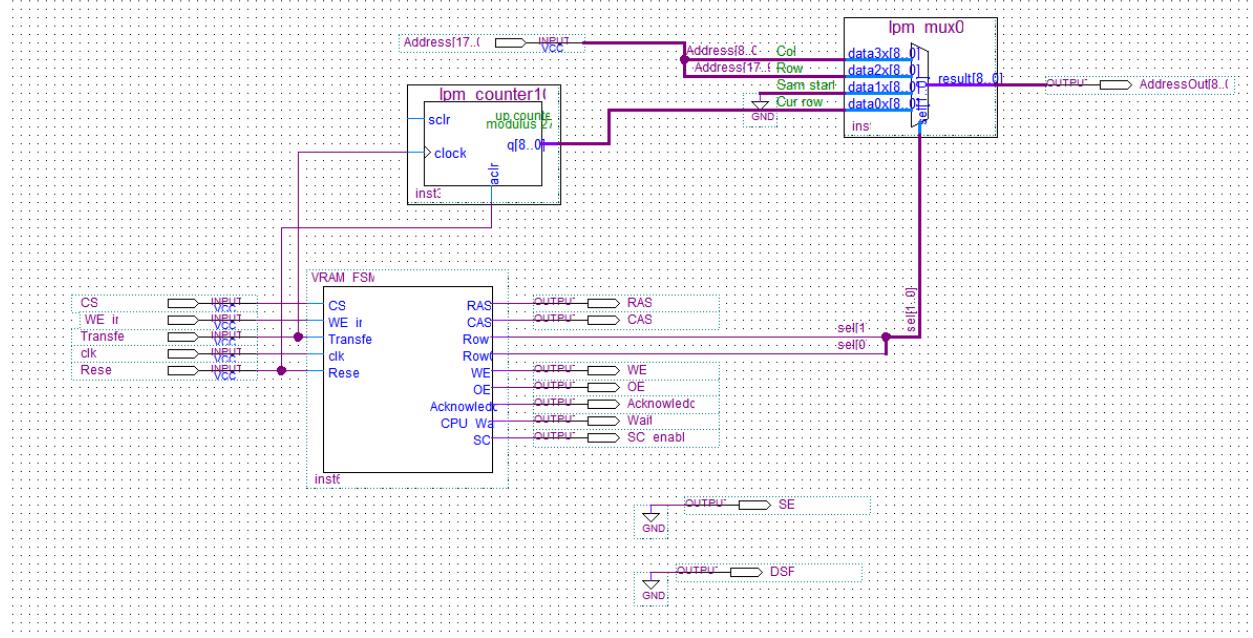


Figure 30: VRAM Controller.

Address: The output address is selected and outputted using a multiplexer. For either the column or row address, the first or second half of the input address is returned. In the case of a row transfer, the SAM start address, 0 for the beginning of the memory, and the current row, are outputted at the appropriate times. A counter is used to increment the current row for after each row transfer cycle, from row 0 to 272, or the total height of the LCD.

VRAM State Machine: The read, write, row transfer, and refresh cycles are performed with a Moore state machine, Appendix D.1.

The state machine begins at an idle state, and goes back to the idle state after each cycle is completed. If there is a row transfer request, when the transfer flag is active, the row transfer cycle is performed. If there is a write request, when the chip select and write enable inputs are active, then the write cycle is performed. If there is a read select, when the chip select is enabled but write enable is not, then the read cycle is performed. If none of these cycles are to be completed when in the idle state, a refresh cycle is performed.

Timing diagrams were used to create the state machine with the correct outputs, found in Appendix A.

LCD Controller: The LCD controller is used to generate the clock signals for the

LCD.

The Vertical Sync (VSYNC) signal is used for changing rows, and Horizontal Sync (HSYNC) for changing columns. The Data Enable (DE) signal is also generated for when input data is valid within the VSYNC AND HSYNC signals. A 12 MHz clock signal is also sent to the LCD, while a serial clock signal is generated for the VRAM clock input to the serial address counter for the SAM registers.

The row transfer request (Transfer signal in Figure 31) is set when the end of the row has been reached, and the flag is cleared once the VRAM controller has completed the row transfer and set the transfer acknowledge flag.

The timing diagrams can be found in Appendix A, Figures 41 and 42.

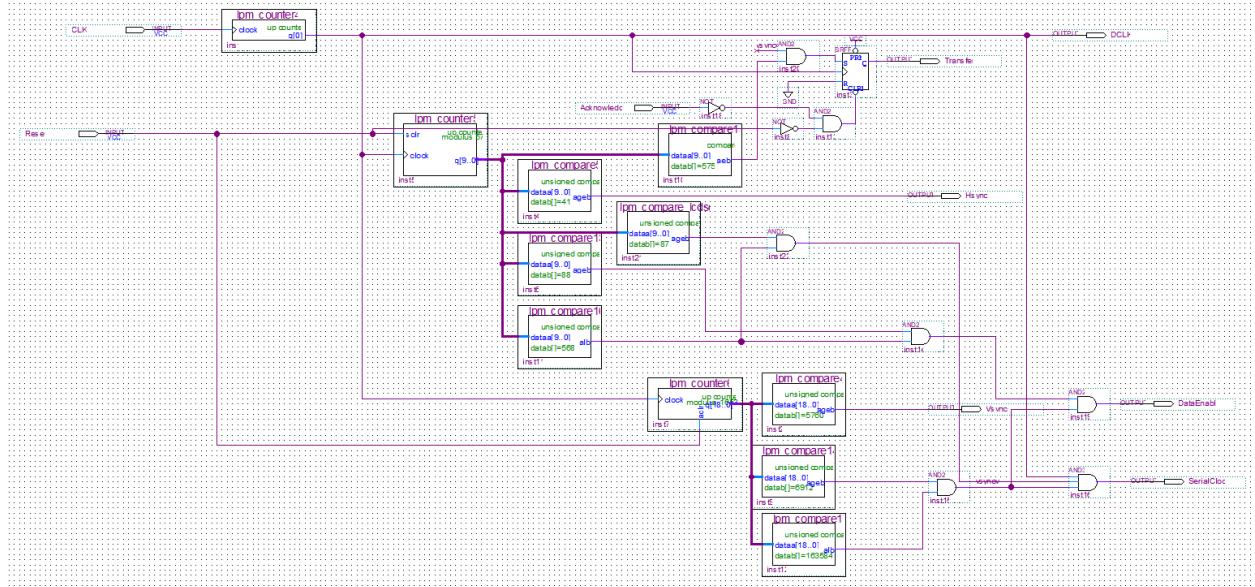


Figure 31: LCD Controller, with comparators for defining valid regions in the signals.

2.1.8 JTAG, Reset, and Clock

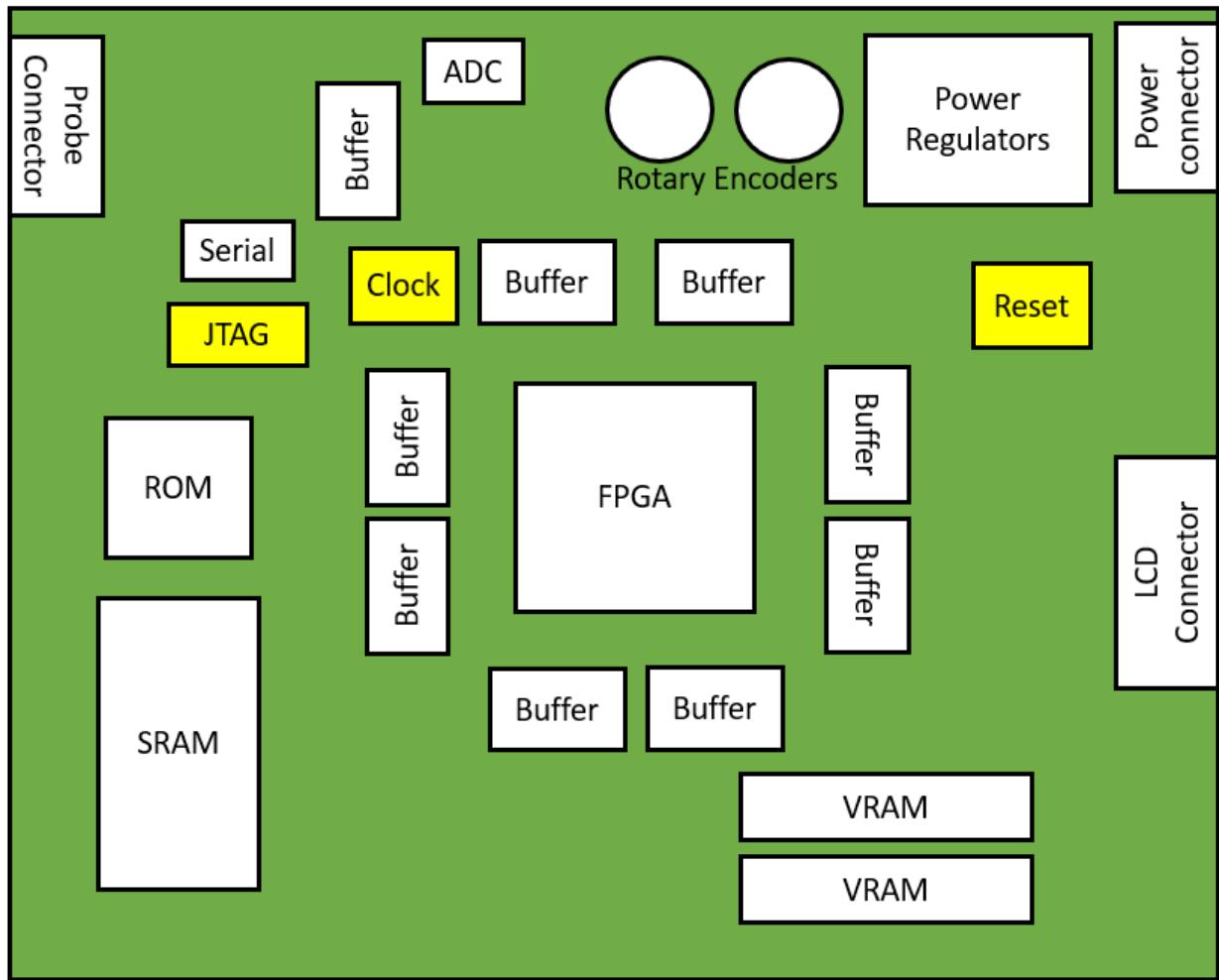


Figure 32: Board Layout: Clock, JTAG, Reset.

JTAG:

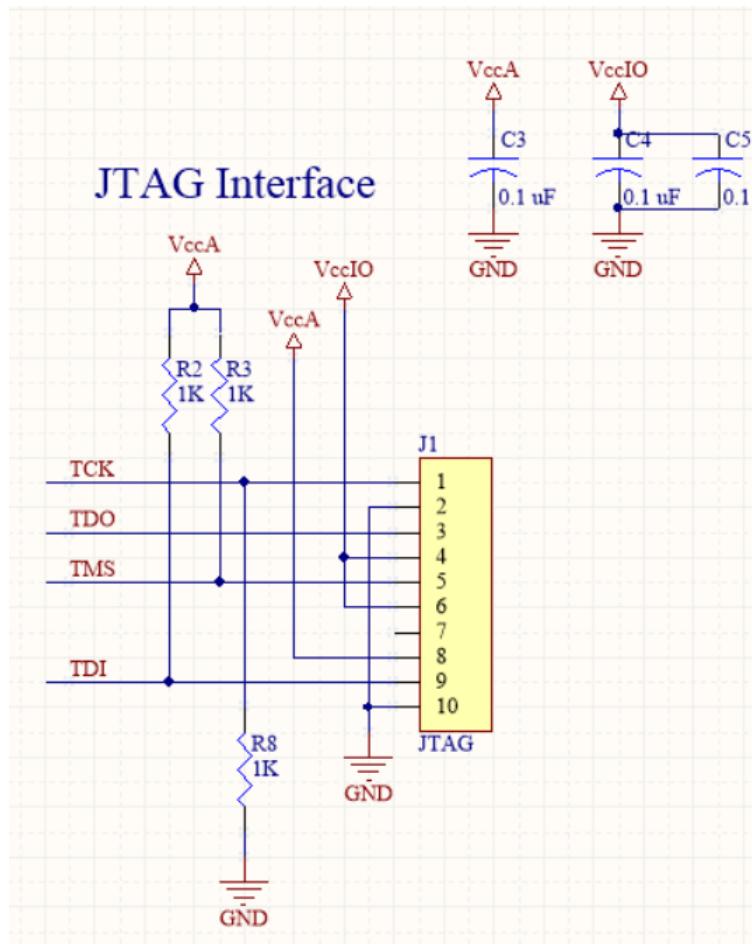


Figure 33: JTAG interface Schematic.

A JTAG interface was used for debugging the system, seen in Figure 33 and is connected directly to the appropriate FPGA pins.

Reset:

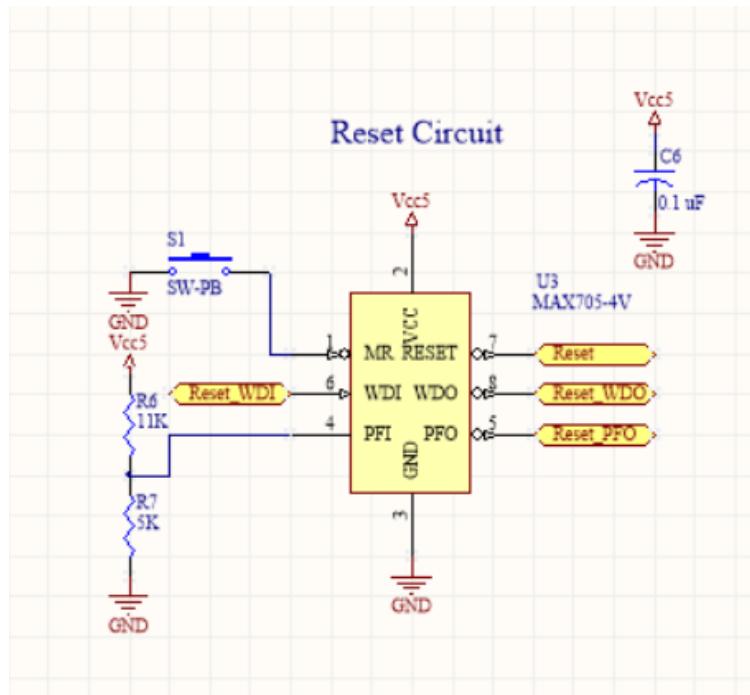


Figure 34: Reset Circuit Schematic.

A reset circuit is used, which includes a microprocessor supervisory device (U3, MAX705). The watchdog input (WDI) is implemented in the main loop, and a voltage divider is used for a power-fail warning at 4 V from the 5 V supply.

The active-low watchdog output, power-fail output, and reset output, and manual reset output are combined for an overall reset signal that is sent to the CPU and various controllers.

Clock:

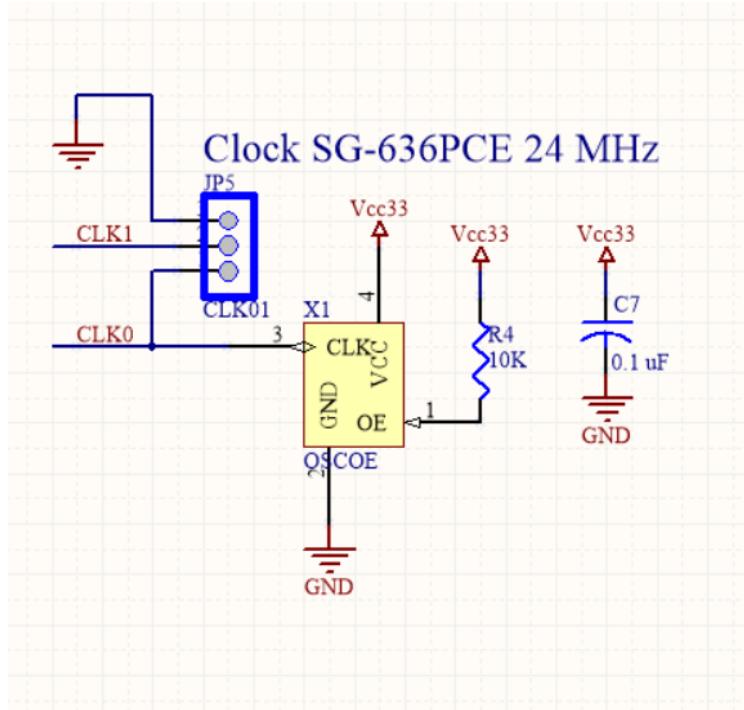


Figure 35: Clock Circuit Schematic.

A 24 MHz oscillator (X1) was used for the clock input for the device, with the CLK0 signal connected directly to the appropriate FPGA pin.

2.1.9 Fixes

wrong adc footprint, buffer soldering error, backwards LCD connector footprint

2.2 Software

2.2.1 Software System Overview

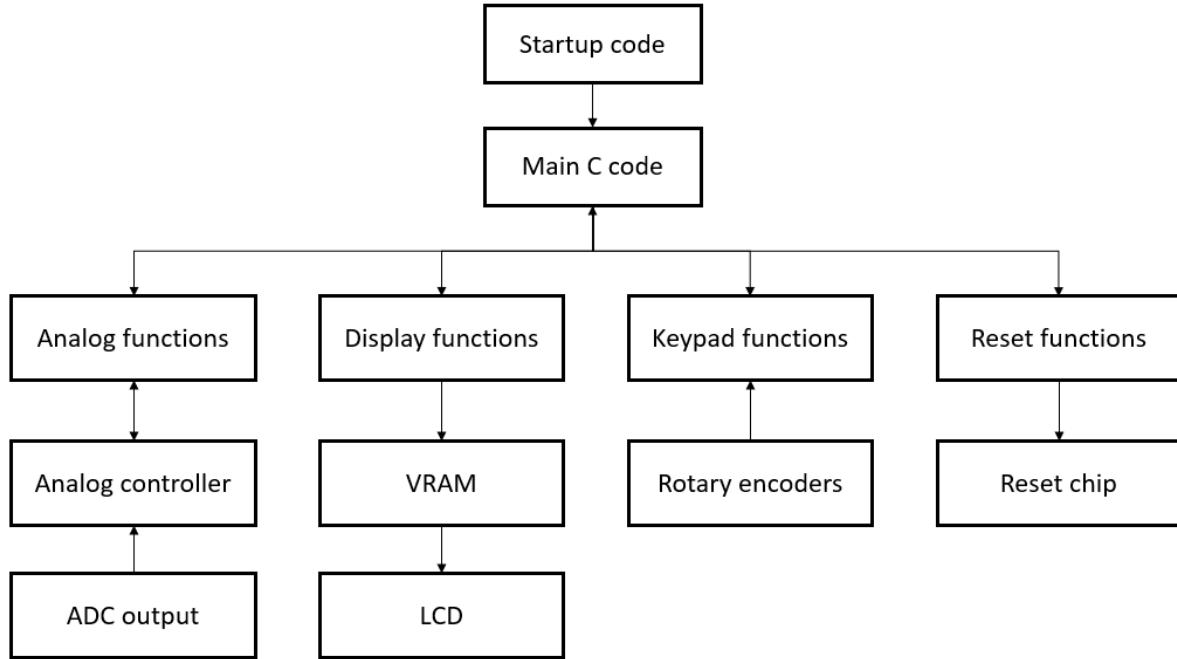


Figure 36: Software block diagram overview.

The main components that were written were routines for the analog settings, display, and keypad⁴. As shown in Figure 36, the main code handles the main loop and calls the various utility functions. The analog functions set the trigger settings used by the analog controller, and also clock out the signal data from the FIFO buffer into a buffer in the data section to be displayed. Display functions clear the display and plot a pixel by writing to the VRAM, whose data is outputted to the LCD. Keypad routines include an event handler and getting the key pressed. A reset function that toggles the watchdog input on the reset chip was also written. These will be discussed in more details below.

⁴ rotary encoders were implemented instead of a keypad, so references to the keypad refer to the rotary encoders

2.2.2 Analog Software

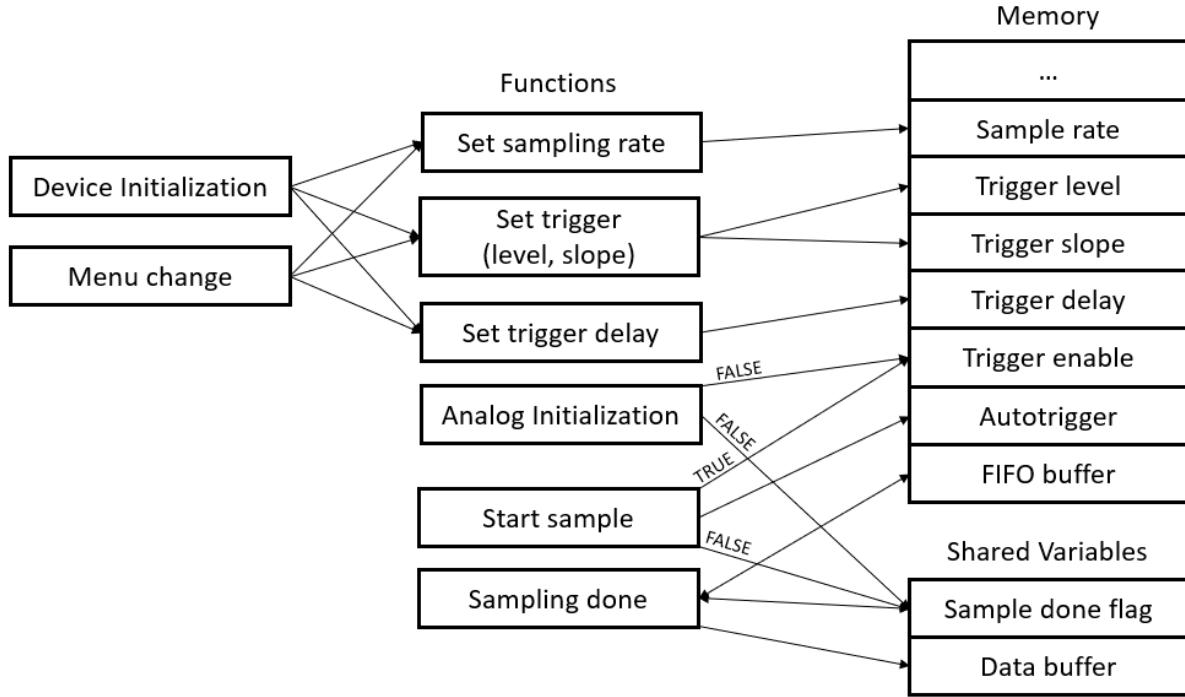


Figure 37: Analog software block diagram.

The analog software includes analog initialization, which sets the trigger enable and sample done flags to false, as well as a start_sample initialization for to start the sample. It enables triggering, sets autotriggering, and sets the sample done flag to false. The sample done flag is used to ensure that only one sample is taken per enabled trigger.

Sampling_done checks to see if the buffer is full and a sample has been taken. If it is, and no sample has been taken yet, the data from the FIFO is clocked out and stored in the data buffer shared variable to be displayed. These functions are called by the main loop and the various trace and sampling utilities.

There are also functions for setting the sampling rate, trigger level, trigger slope, and trigger delay. These are called upon initialization and when their corresponding settings have been changed from the menu. These directly change the outputs that are sent to the analog controller and are used to take the sample.

Listing 1: ..//osc134/analog.S

```
# ######
#                               #
#           Analog functions for digital oscilloscope      #
#                           EE 52                                #
#                               #
#                               #
# ######
# Name of file: analog.s
# Description: Analog functions for digital oscilloscope
# Public functions:
#     int set_sample_rate(long int samples_per_sec): Sets the sample rate
#             to the passed value (in samples per second). The number of samples
#             that will be taken at that sample rate is returned.
#     void set_trigger(int level, int slope): Sets the trigger level to the
#             first argument and the slope to the second argument
#             (1 for negative slope, 0 for positive slope). The trigger level is
#             passed as a value between 0 and 127, with 0 indicating the lowest
#             (most negative for bipolar input) trigger input level and 127
#             indicating the highest (most positive) trigger input level.
#     void set_delay(long int delay): Sets the trigger delay to the passed value.
#     void start_sample(int auto_trigger): Immediately starts sampling data.
#             If the argument is FALSE, then the sampling should start when there
#             is a trigger event. If the argument is TRUE, then the sampling should
#             start when there is a trigger event or when the automatic trigger
#             timeout occurs, whichever is first.
#     unsigned char far *sample_done(void): Returns NULL if not done with the
#             current sample and a pointer to the sampled data otherwise. The
#             sampled data should contain the number of data points previously
#             returned by the set_sample_rate function. The function should only
#             return a non-NUL pointer once for each call to the start_sample
#             function.
# Local functions: None.
# Input:  None.
# Output: None.
#
# Revision History: 06/24/17 Sophia Liu initial revision
#                   09/10/17 Sophia Liu edited comments
#
# inc header files for constants
#include "general.h"
#
# set sample rate
#
#
```

```

# Description: Sets the sample rate to the passed argument, in samples per
#      second. Returns the number of samples that will be taken at that sample
#      rate.
#
# Operation: Divides the clock frequency by the rate to get the number of
#      clocks between samples. Writes the sample number to the sample rate pio.
#      Returns the sample number.
#
# Arguments:      Long int samples_per_sec (r4) - 19-bit value for the
#                  number of samples per second.
#
# Return Values:   int num_samples: number of samples that will be taken at
#                  the clock rate constant.
#
# Local Variables: None.
# Shared Variables: None.
# Global Variables: None.
#
# Input:          None.
# Output:         None.
#
# Error Handling: None.
# Algorithms:     None.
# Data Structures: None.
#
# Known Bugs:     None.
# Limitations:    None.
# Registers changed: r8. r9. r2.
# Stack depth:    0 words.
#
# Revision History: 06/24/17 Sophia Liu    initial revision
.section .text
.align 4
.global set_sample_rate
.type set_sample_rate, @function
set_sample_rate:
    movhi r8, %hi(CLOCK_FREQ) # store 32 bit clock frequency constant into register
    ori r8, r8, %lo(CLOCK_FREQ)
    divu r9, r8, r4           # divide clock frequency by sample rate for sample number

    movia r8, SAMPLE_RATE_PIO # get address of sample rate pio
    stwio r9, 0(r8)           # drive sample number on sample rate pio output

    movi r2, NUM_SAMPLES      # return sample size constant (have a set fifo size)
    ret

```

```

# set trigger
#
#
# Description: Takes two arguments for trigger level and slope. The trigger
#   level is between 0 (lowest trigger input level) and 127 (highest trigger
#   input level), and the slope is 1 (negative slope) or 0 (positive slope).
#   Sets the trigger level and slope pios to the passed arguments.
#
# Operation: Scales the trigger level argument to match the analog input
#   signal and writes the trigger level and slope arguments to the
#   corresponding pio cores.
#
# Arguments: int level (r4) - 7-bit value for the trigger level,
#             with 0 being the lowest trigger input level and 127 being the
#             highest trigger input level.
#             int slope (r5) - 1-bit value for the trigger slope, with
#             1 being a negative slope and 0 being the positive slope.
#
# Return Values: None.
#
# Local Variables: None.
# Shared Variables: None.
# Global Variables: None.
#
# Input:        None.
# Output:       None.
#
# Error Handling: None.
# Algorithms:    None.
# Data Structures: None.
#
# Known Bugs:    None.
# Limitations:   None.
# Registers changed: r4, r8.
# Stack depth:    0 words.
#
# Revision History: 06/24/17 Sophia Liu    initial revision
.section .text
.align 4
.global set_trigger
.type set_trigger, @function
set_trigger:
    movia r8, TRIGGER_LEVEL_PIO # get trigger level pio address
    muli r4, r4, 2              # multiply by 2 to get a range of 0 to 254,

```

```

        #      to match 8 bit ADC data/signal range
stbio r4, 0(r8)          # write level argument to trigger level pio

movia r8, TRIGGER_SLOPE_PIO # get trigger slope pio address
stbio r5, 0(r8)          # write slope argument to trigger slope pio

ret

# set delay
#
#
# Description: Takes one argument, the delay in samples. Sets the trigger
#      delay to the passed value (in samples)
#
# Operation: Writes the sample delay argument to the corresponding pio.
#
# Arguments:      long int delay- 16-bit value for the delay (in samples).
# Return Values:  None.
#
# Local Variables: None.
# Shared Variables: None.
# Global Variables: None.
#
# Input:          None.
# Output:         None.
#
# Error Handling: None.
# Algorithms:    None.
# Data Structures: None.
#
# Known Bugs:     None.
# Limitations:   None.
# Registers changed: r8.
# Stack depth:   0 words.
#
# Revision History: 06/24/17 Sophia Liu    initial revision
.section .text
.align 4
.global set_delay
.type set_delay, @function
set_delay:
movia r8, SAMPLE_DELAY_PIO # get the sample delay pio address
stwio r4, 0(r8)          # write delay argument to delay pio core

ret

```

```

# start sample
#
#
# Description: Starts sampling data. Takes one argument for auto triggering.
#   If the argument is false, sampling starts when there is a trigger event.
#   If the argument is true, the sampling starts when there is a trigger event
#   or when the automatic trigger timeout occurs (whichever is first)
#
# Operation: Writes the auto trigger argument to the auto trigger pio, and
#   sets the trigger enable pio high to start sampling data. Sets the sample
#   done shared variable to false to indicate that no trigger has occurred yet.
#
# Arguments:      int auto_trigger - 1-bit value for auto-triggering.
#                  FALSE if auto-triggering is not enabled, TRUE
#                  if it is enabled.
# Return Values:  None.
#
# Local Variables: None.
# Shared Variables: sample_done_flag(W) - 1 byte boolean flag. TRUE if the
#                   sample has been taken, FALSE otherwise.
# Global Variables: None.
#
# Input:          None.
# Output:         None.
#
# Error Handling: None.
# Algorithms:    None.
# Data Structures: None.
#
# Known Bugs:     None.
# Limitations:   None.
# Registers changed: r8, r9.
# Stack depth:    0 words.
#
# Revision History: 06/24/17 Sophia Liu    initial revision
.section .text
.align 4
.global start_sample
.type start_sample, @function
start_sample:
    movia r8, AUTO_TRIGGER_PIO    # get auto trigger pio address
    stbio r4, 0(r8)              # write auto triggering argument to pio core

    # enable triggering- pulse trigger enable pio and set high

```

```

movia r8, TRIGGER_ENABLE_PIO # get trigger enable pio address
movi r9, FALSE
stbio r9, 0(r8)           # set trigger enable pio low

movi r9, TRUE
stbio r9, 0(r8)           # set trigger enable pio high

# reset sample done flag - only after each start_sample call
movia r8, sample_done_flag # get sample done flag shared variable address
movi r9, FALSE
stbio r9, 0(r8)           # set the sample done flag to false
ret

# sample done
#
#
# Description: Returns NULL if not done with the current sample and a pointer
#               to the sampled data otherwise. The sampled data contains the number of
#               data points previously returned by the set_sample_rate functions. Only
#               returns a non-NULL pointer once for each call to the start_sample function.
#
# Operation: Checks the sample done flag and returns null if the sample is
#             already completed. Checks the fifo full pio and returns null if the fifo
#             is not full. Otherwise, the fifo is clocked out into the data buffer
#             shared variable, the sample done flag is set to TRUE, trigger enabling is
#             set to FALSE, and a pointer to the data buffer is returned.
#
# Arguments:      None.
# Return Values:   sampled data - 32-bit pointer to the sampled data. Returns
#                  NULL if not done with the current sample. Only
#                  returns a non-NULL pointer once for each call to the
#                  start_sample function.
#
# Local Variables: None.
# Shared Variables: sample_done_flag(R/W) - 1 byte boolean flag. TRUE if the
#                   sample has been taken, FALSE otherwise.
#                   data_buffer(W) - NUM_SAMPLES (number of samples taken)
#                   bytes, buffer for sampled data from the fifo buffer.
#
# Global Variables: None.
#
# Input:          None.
# Output:         None.
#
# Error Handling: None.

```

```

# Algorithms:      None.
# Data Structures: None.
#
# Known Bugs:      None.
# Limitations:     None.
# Registers changed: r2, r8, r9, r10, r11, r12, r13.
# Stack depth:     0 words.
#
# Revision History: 06/24/17 Sophia Liu    initial revision
.section .text
.align 4
.global sample_done
.type sample_done, @function
sample_done:
movi r8, TRUE          # store true to compare to sample done flag

movia r9, sample_done_flag # get sample done flag shared variable address
ldb r10, 0(r9)           # load sample done flag value from shared variable
beq r10, r8, return_null_ # If the flag is true, the sample is already done,
                         #      return null.
# bne r10, r8, check_fifo # Else flag is false, check if fifo is full

check_fifo:
movia r9, FIFO_FULL_PIO # get address of the fifo full pio
ldbio r10, 0(r9)         # check to see if the fifo is full
bne r10, r8, return_null_ # If the flag is not true, fifo is not full,
                         #      return null.
# beq r10, r8, return_pointer # Else flag is true, fifo is full, return pointer

return_pointer:          # copy fifo sample buffer into data buffer
movia r8, DATA_READY_PIO # get address for fifo read clock input pio
movia r10, FIFO_DATA_PIO # get address to read fifo data from
movia r11, data_buffer   # get address of data buffer shared variable
addi r13, r11, NUM_SAMPLES # get address of last sample in data buffer

# loop to clock out fifo buffer
read_fifo_loop:
ldbio r12, 0(r10)        # load current sample from fifo
stb r12, 0(r11)          # store sample in data buffer
addi r11, r11, 1          # go to next address in data buffer

bge r11, r13, got_sample # if at address of last sample in data buffer,
                         #      finished reading samples from fifo
# blt r11, r13, read_fifo_loop_next # otherwise, continue loop

```

```

read_fifo_loop_next:      # clock out next sample from fifo
movi r9, 0
stbio r9, 0(r8)
movi r9, 1
stbio r9, 0(r8)
movi r9, 0
stbio r9, 0(r8)          # pulse read clock for fifo
jmpi read_fifo_loop      # go back to top of loop to read the next sample

got_sample:
movia r8, sample_done_flag # get address of sample done flag shared variable
movi r9, TRUE
stb r9, 0(r8)             # set sample done flag to true - completed sample

movia r8, TRIGGER_ENABLE_PIO # get address of trigger enable pio
movi r9, FALSE
stbio r9, 0(r8)            # stop sampling, set trigger enable to false

movia r2, data_buffer      # return pointer to sampled data
ret

return_null_:
movi r2, PTRNL             # return null
ret

# init analog
#
#
# Description: Initializes variables used for analog functions.
#
# Operation: Initializes the sample done flag to false (sample has not been
# completed) and sets the trigger enable pio to false (triggering not
# enabled yet).
#
# Arguments:      None.
# Return Values:   None.
#
# Local Variables: None.
# Shared Variables: sample_done_flag(W) - 1 byte boolean flag. TRUE if the
#                   sample has been taken, FALSE otherwise.
# Global Variables: None.
#
# Input:           None.
# Output:          None.
#

```

```

# Error Handling: None.
# Algorithms: None.
# Data Structures: None.
#
# Known Bugs: None.
# Limitations: None.
# Registers changed: r8, r9.
# Stack depth: 2 words.
#
# Revision History: 06/24/17 Sophia Liu initial revision

.section .text
.align 4
.global init_analog
.type init_analog, @function
init_analog:
    addi sp, sp, -8      # adjust stack pointer
    stw ra, 0(sp)        # store return address
    stw r28, 4(sp)        # store frame pointer

    movia r8, sample_done_flag # get address of sample done flag shared variable
    movi r9, FALSE
    stb r9, 0(r8)          # set sample done flag to false

    init_pios:
    # disable triggering
    movia r8, TRIGGER_ENABLE_PIO # get address of the trigger enable pio
    movi r9, FALSE
    stbio r9, 0(r8)          # set trigger enable pio to false

    ldw ra, 0(sp)          # load return address
    ldw r28, 4(sp)          # load frame pointer
    addi sp, sp, 8           # adjust stack pointer
    ret

    # data section
.section .data
.align 4
sample_done_flag: .byte 0x0 # 1 byte boolean flag. TRUE if the sample
                           # has been taken, FALSE otherwise.

data_buffer: .skip NUM_SAMPLES # NUM_SAMPLES (number of samples taken) bytes,
                           # buffer for sampled data from the fifo buffer.

```

2.2.3 Display Software

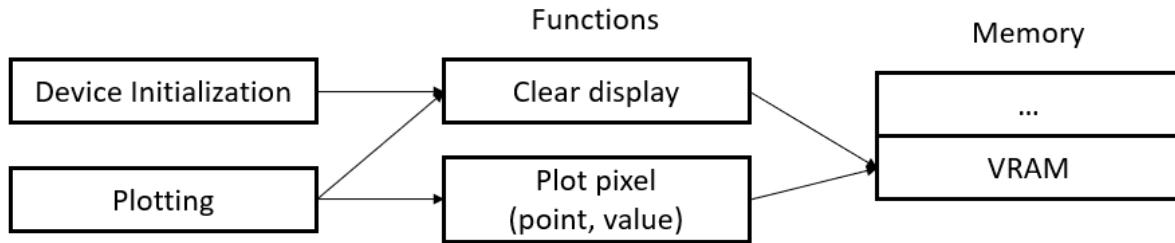


Figure 38: Display software block diagram.

The display functions perform the basic requirements of clearing the LCD, or turning all the pixels black, and plotting a specified pixel on the display. These modify the data at the appropriate addresses in the VRAM. These functions are used upon initialization and when plotting the menu and sample, as the display is cleared between each sample.

Listing 2: ..//osc134/displayFunc.S

```
# ######
#                               #
#           Display functions for digital oscilloscope      #
#                           EE 52                                #
#                               #
#                               #
# ######
# Name of file: display.s
# Description: Display functions for digital oscilloscope
# Public functions:
#     void clear_display(void): Clears the display (turns black) by storing
#           the pixel off value (PIXEL_WHITE) in all VRAM locations.
#     void plot_pixel(unsigned int x, unsigned int y, int p): Sets the pixel at
#           the passed position (x, y) to the passed value p. The position is
#           passed in binary with (0,0) being the upper left corner.
#
# Local functions: None.
#
# Input:  None.
# Output: None.
#
# Revision History: 06/06/17 Sophia Liu Initial revision
#                   09/04/17 Sophia Liu Edited comments
#                   12/2017 Sophia Liu Edited comments
#
# inc files for display constants
#include "interfac.h"
#include "general.h"
#
# Clear display
#
# Description: Clears the display (turns black) by storing the pixel off value
#           (PIXEL_WHITE) in all VRAM locations.
#
# Operation: Goes through each VRAM address and stores PIXEL_WHITE (pixel off)
#           at each location.
#
# Arguments:      None.
# Return Values:   None.
#
# Local Variables: None.
# Shared Variables: None.
# Global Variables: None.
```

```

#
# Input:          None.
# Output:         None.
#
# Error Handling: None.
# Algorithms:    None.
# Data Structures: None.
#
# Known Bugs:     None.
# Limitations:   None.
# Registers changed: r6, r9, r11, r12, r13
# Stack depth: 0 words.
#
# Revision History: 06/22/17 Sophia Liu    initial revision
.section .text
.align 4
.global clear_display
.type clear_display, @function
clear_display:
movia r6, VRAM_ADDRESS    # store vram beginning address
movia r13, VRAM_ADDRESS_END # store vram ending address
movi r9, PIXEL_WHITE      # store value for pixel off
movi r12, TRUE            # store value for true, for comparisons

clear_display_start:
stb    r9, 0(r6)           # clear next vram byte
addi  r6, r6, 1             # move to next address

cmpeq r11, r6, r13         # check if reached end of vram
beq    r11, r12, clear_display_end # if reached end address, end function
jmpi  clear_display_start    # else move to next address

clear_display_end:
ret

# Plot_pixel
#
# Description: Sets the pixel at the passed position (x, y) to the passed
#               value p. The position is passed in binary with (0,0) being the
#               upper left corner.
#
# Operation: Checks to make sure the position (x, y) is within the bounds of the
#            display screen. If it is, the corresponding vram address is
#            calculated and the pixel value p is stored.
#

```

```

# Arguments:      unsigned int x (r4) - x value of pixel to plot, with (0,0)
#                  being the upper left corner.
#      unsigned int y (r5)- y value of pixel to plot, with (0,0)
#                  being the upper left corner.
#      int p (r6) - 8-bit value of the pixel to plot.
# Return Values:  None.

#
# Local Variables: None.
# Shared Variables: None.
# Global Variables: None.

#
# Input:          None.
# Output:         None.

#
# Error Handling: None.
# Algorithms:    None.
# Data Structures: None.

#
# Known Bugs:     None.
# Limitations:   None.

# Registers changed: r8, r9, r10, r11.
# Stack depth:    0 words.

#
# Revision History: 06/22/17 Sophia Liu    initial revision

.section .text
.align 4
.global plot_pixel
.type plot_pixel, @function
plot_pixel:
check_pixel:
movi r9, TRUE
cmpgei r8, r4, SIZE_X    # check if x point is too large for LCD
beq r8, r9, invalid_point # if it is invalid, end

cmpgei r8, r5, SIZE_Y    # check if y point is too large for LCD
beq r8, r9, invalid_point # if it is invalid, end

movia r10, VRAM_ADDRESS  # get vram address
add r10, r4, r10          # add x offset to vram address

muli r11, r5, VRAM_WIDTH # multiply y by length of vram row for y offset
add r10, r11, r10          # add y offset to vram address

stb r6, 0(r10)            # store pixel byte value in vram address
jmpi end_plot

```

```
invalid_point:          # invalid point, return
ret

end_plot:              # plotted point, return
ret
```

2.2.4 Keypad Software

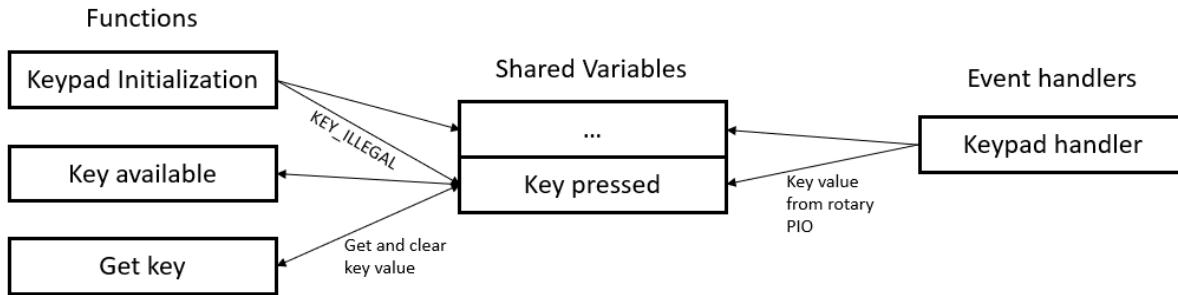


Figure 39: Keypad software block diagram.

The keypad software has an event handler for whenever a key is pressed and a keypad interrupt occurs. The key value is stored in the key_pressed shared variable. This is accessed by the key_available and getKey functions. Key_available returns whether or not there is a valid key, and if there is, getKey is called and will return the valid key. These are called from the main loop when checking if there is keypad input.

Listing 3: ..//osc134/keypad.S

```
# ##### Keypad functions for digital oscilloscope #####
#          EE 52
#
# Name of file: keypad.s
# Description: Keypad functions for digital oscilloscope. Used to get the
#               current key value and handle keypad interrupts. Rotary encoders
#               were implemented, so "keypad" often refers to the rotary encoders.
# Public functions:
#     unsigned char key_available(void): Whether or not there is a valid key
#               value. Returns TRUE (non-zero) if there is a valid key ready to be
#               processed and FALSE (zero) otherwise.
#     int getkey(void): Returns the keycode (found in interfac.h) for the
#               debounced key. This value is never KEY_ILLEGAL. The function does
#               not return until it has a valid key.
#     void initKeypad(void): Initializes keypad interrupts and variables.
#               Registers keypad event handler, initializes key_pressed with
#               KEY_ILLEGAL, initializes rotary PIO and enables interrupts.
#               Must be called before using the keypad.
#
# Local functions:
#     keypadInterruptHandler: Event handler for the keypad. Stores the current
#               valid key value in key_pressed.
#
# Input:  None
# Output: None
#
# Revision History: 06/06/17 Sophia Liu initial revision
#                   12/2017 Sophia Liu   edited comments

#include "interfac.h"
#include "general.h"

.section .text

# Key_available
#
#
# Description: Whether or not there is a valid key value. Returns TRUE
#               (non-zero) if there is a valid key ready to be processed and
#               FALSE (zero) otherwise.
```

```

#
# Operation: Reads current key_pressed value. Returns TRUE (non-zero) if there
#           is a valid key ready to be processed and FALSE (zero) otherwise.
#
# Arguments:      None.
# Return Values:   key is available - unsigned char, TRUE (non-zero) if
#                   current key value is valid or FALSE (zero) if it
#                   is not; whether or not there is a valid key ready
#                   to be processed.
#
# Local Variables: None.
# Shared Variables: key_pressed (R) - 1 word unsigned int, contains current
#                   debounced key.
# Global Variables: None.
#
# Input:          None.
# Output:         None.
#
# Error Handling: None.
# Algorithms:     None.
# Data Structures: None.
#
# Known Bugs:     None.
# Limitations:    None.
# Registers changed: r8, r9, r2.
# Stack depth:    2 words.
#
# Revision History: 06/06/17 Sophia Liu    initial revision
.align 4
.global key_available
.type key_available, @function
key_available:
    addi sp, sp, -8      # adjust stack pointer
    stw ra, 0(sp)        # store return address
    stw r28, 4(sp)        # store frame pointer

    movia r8, key_pressed  # get address of key_pressed
    ldbu r9, 0(r8)        # load value in key_pressed
    cmpnei r2, r9, KEY_ILLEGAL # check if current key value is illegal key
                                # r2 = 0 (FALSE) if is illegal key, 1 (TRUE) if not

    ldw ra, 0(sp)        # load return address
    ldw r28, 4(sp)        # load frame pointer
    addi sp, sp, 8        # adjust stack pointer

```

```

ret

# GetKey
#
#
# Description:      Returns the keycode (defined in interfac.h) for the debounced
#                   key, and does not return until there is a valid key.
#
# Operation:        Waits until there is a valid key from key_pressed, and
#                   returns the key value. Never returns KEY_ILLEGAL.
#
# Arguments:        None.
# Return Values:    key - int, the keycode for the debounced key.
#                   Always a valid key, never KEY_ILLEGAL.
#
# Local Variables: None.
# Shared Variables: key_pressed (R/W) - 1 word unsigned int, contains current
#                   debounced key.
# Global Variables: None.
#
# Input:            None.
# Output:           None.
#
# Error Handling:   None.
# Algorithms:       None.
# Data Structures:  None.
#
# Known Bugs:       None.
# Limitations:      None.
# Registers changed: r2, r9, r10.
# Stack depth:      2 words.
#
# Revision History: 06/06/17 Sophia Liu    initial revision
.align 4
.global getkey
.type getkey, @function
getkey:
    addi sp, sp, -8      # adjust stack pointer
    stw ra, 0(sp)        # store return address
    stw r28, 4(sp)        # store frame pointer

    movia r9, key_pressed # get address of key_pressed

get_key_loop:
    ldw r2, 0(r9)          # load value in key_pressed

```

```

cmpeqi r10, r2, KEY_ILLEGAL # check if current key value is illegal key
bne r0, r10, get_key_loop # if current key is illegal key, loop back and
                           #     check again
# beq r0, r10, have_key # else key is not illegal key, can return

have_key:
movi r10, KEY_ILLEGAL
stw r10, 0(r9)           # clear key_pressed

ldw ra, 0(sp)            # load return address
ldw r28, 4(sp)            # load frame pointer
addi sp, sp, 8             # adjust stack pointer

ret                      # have key, return

# keypadInterruptHandler
#
#
# Description:      Event handler for the keypad. Stores the current
#                   valid key value in key_pressed.
#
# Operation:        Reads in the keycode value and saves it in key_pressed if
#                   it is valid. Clears edge capture register and re-enables
#                   interrupts.
#
# Arguments:        None.
# Return Values:    None.
#
# Local Variables: None.
# Shared Variables: key_pressed (W) - 1 word unsigned int, contains current
#                   debounced key.
# Global Variables: None.
#
# Input:            None.
# Output:           None.
#
# Error Handling:   None.
# Algorithms:       None.
# Data Structures:  None.
#
# Known Bugs:       None.
# Limitations:      None.
# Registers changed: r8, r9, r10.
# Stack depth:      2 words.
#

```

```

# Revision History: 06/06/17 Sophia Liu    initial revision
.align 4
.global keypadInterruptHandler
.type keypadInterruptHandler, @function
keypadInterruptHandler:
    addi sp, sp, -8      # adjust stack pointer
    stw ra, 4(sp)        # store return address
    stw r28, 0(sp)        # store frame pointer

load_key:
    movia r9, ROT_ADDRESS    # read in rotary pio core address
    ldbio r10, 12(r9)

    cmpeqi r8, r10, KEY_ILLEGAL    # check if current key value is illegal
    bne r0, r8, endKeypadInterruptHandler # if current key is illegal key, end
                                            #      (all rotary actions are valid)

    movia r8, key_pressed # get address of key_pressed
    stw r10, 0(r8)        # store pio data value from edge register in key_pressed

    movui r8, 0
    stbio r8, 12(r9) # edge capture register, clear all

    movi r8, 0x1f
    stbio r8, 8(r9)  # interrupt mask register, enable interrupts

endKeypadInterruptHandler:
    ldw r28, 0(sp)    # load return address
    ldw ra, 4(sp)    # load frame pointer
    addi sp, sp, 8    # adjust stack pointer

ret

# InitKeypad
#
#
# Description: Initializes keypad interrupts and variables. Registers keypad
#               event handler, initializes key_pressed with
#               KEY_ILLEGAL, initializes rotary PIO and enables interrupts.
#               Must be called before using the keypad.
#
# Operation: Registers keypad event handler, initializes key_pressed with
#             KEY_ILLEGAL, initializes rotary PIO and enables interrupts.
#
# Arguments:      None.

```

```

# Return Values: None.
#
# Local Variables: None.
# Shared Variables: key_pressed (W) - 1 word unsigned int, contains current
#                   debounced key.
# Global Variables: None.
#
# Input:           None.
# Output:          None.
#
# Error Handling: None.
# Algorithms:     None.
# Data Structures: None.
#
# Known Bugs:      None.
# Limitations:    None.
# Registers changed: r4, r5, r6, r8, r9.
# Stack depth:    2 words.
#
# Revision History: 06/06/17 Sophia Liu    initial revision
.align 4
.global initKeypad
.type initKeypad, @function
initKeypad:
    addi sp, sp, -8      # adjust stack pointer
    stw ra, 0(sp)        # store return address
    stw r28, 4(sp)        # store frame pointer

    movia r8, ROT_ADDRESS # get rotary pio core address
    movi r9, 0
    stbio r9, 8(r8)       # interrupt mask register, disable interrupts

    movui r4, 0            # arguments for alt_irq_register
    movui r5, 0
    movia r6, keypadInterruptHandler
    call alt_irq_register # register interrupt handler for keypad (rotary) event

    movia r8, key_pressed # get key_pressed address
    movi r9, KEY_ILLEGAL
    stw r9, 0(r8)         # initialize key_pressed with illegal key

initInterrupts:
    movia r8, ROT_ADDRESS # get rotary pio core address
    movi r9, 0x0
    stbio r9, 4(r8)       # direction register for pio, set to input

```

```
movi r9, 0x0
stbio r9, 12(r8)      # edge capture register for pio, clear all

movi r9, 0x1f
stbio r9, 8(r8)       # interrupt mask register for pio, enable interrupts

ldw ra, 0(sp)          # load return address
ldw r28, 4(sp)          # load frame pointer
addi sp, sp, 8           # adjust stack pointer
ret

# data section
.section .data
.align 4
key_pressed: .word 0x0 # 1 word unsigned int, contains current debounced key
```

2.2.5 Watchdog Reset

A reset function to toggle the watchdog input on the reset circuit (Reset_WDI, Figure 34) was also written. The toggled output is sent out to the reset device. This function is called in the main loop (Listing D.2), which should never stall or take greater than the time required for the reset device to send out a watchdog reset signal. If it does, some error has occurred and the system will reset.

Listing 4:/osc134/reset.S

```
# ######
#          Reset functions for digital oscilloscope      #
#          EE 52                                         #
#          #
#          #
# #####
# Name of file: reset.s
# Description: Reset functions for digital oscilloscope
# Public functions:
#     void pulse_wd(void): Pulses the watchdog reset output that is sent to the
#                         reset chip. The reset input on the MAX705 used must be toggled
#                         at least every 1.6 secs. If this function is not called within
#                         that time, the system will reset.
# Local functions:
# Input:  None
# Output: None
#
# Revision History:
#     06/24/17 Sophia Liu    initial revision
#     12/17    Sophia Liu    edited comments

# inc header files for constants
#include "general.h"

# pulse wd
#
#
# Description: Pulses the watchdog reset output that is sent to the reset
#               chip. The reset input on the MAX705 used must be toggled
#               at least every 1.6 secs. If this function is not called within
#               that time, the system will reset.
#
# Operation: Gets the reset PIO address and sets the output high then low.
#
# Arguments:      None.
# Return Values:   None.
#
# Local Variables: None.
# Shared Variables: None.
# Global Variables: None.
#
# Input:          None.
```

```
# Output:      None.
#
# Error Handling: None.
# Algorithms:   None.
# Data Structures: None.
#
# Known Bugs:    None.
# Limitations:   None.
# Registers changed: r8, r9
# Stack depth:    0 words.
#
# Revision History: 06/24/17 Sophia Liu    initial revision
.section .text
.align 4
.global pulse_wd
.type pulse_wd, @function
pulse_wd:
    movia r8, WD_PIO # get watchdog input pio address
    movi r9, 1
    stbio r9, 0(r8) # set watchdog input high

    movi r9, 0
    stbio r9, 0(r8) # set watchdog input low

ret
```

2.2.6 Header files

Listing 5: ..//osc134/general.h

```
*****  
/* */  
/* GENERAL.H */  
/* General Definitions */  
/* Include File */  
/* Digital Oscilloscope Project */  
/* EE/CS 52 */  
/* */  
*****  
  
/*  
   This file contains definitions for the Digital Oscilloscope  
   project. This includes addresses and other constants specific to this scope.  
  
Revision History:  
 07/16/17 Sophia Liu Initial revision  
 09/03/17 Sophia Liu Edited comments  
*/  
  
#ifndef __GENERAL_H__  
#define __GENERAL_H__  
  
// addresses  
#define VRAM_ADDRESS      0x140000 // vram start address  
#define VRAM_ADDRESS_END 0x17ffff // vram end address  
  
// PIO addresses  
#define ROT_ADDRESS       0x1d10e0 // rotary pio address  
  
// analog PIO addresses  
#define AUTO_TRIGGER_PIO 0x1d1150 // 1 bit data, 1 for auto triggering,  
                                // 0 for no auto triggering  
#define TRIGGER_ENABLE_PIO 0x1d1140 // 1 bit data, 1 for trigger enable,  
                                // 0 if not enabled  
#define TRIGGER_SLOPE_PIO 0x1d1130 // 1 bit data, 1 for negative slope,  
                                // 0 for positive slope  
#define SAMPLE_DELAY_PIO 0x1d1120 // 16 bit data, sample delay (in # samples)  
#define TRIGGER_LEVEL_PIO 0x1d1110 // 8 bit data for trigger level  
#define SAMPLE_RATE_PIO 0x1d1100 // 19 bit data for sample rate (samples/sec)  
  
// PIO addresses - FIFO  
#define DATA_READY_PIO 0x1d10f0 // data ready signal, read clock input for fifo  
#define FIFO_DATA_PIO 0x1d10c0 // 8 bit sample currently read from fifo
```

```
#define FIFO_FULL_PIO    0x1d10d0 // 1 bit, 1 if fifo full, 0 if not full

#define WD_PIO           0x1d10b0 // 1 bit output for watchdog timer

// other constants
#define VRAM_WIDTH      512          // length of VRAM row
#define NUM_SAMPLES     512          // number of samples in fifo
#define CLOCK_FREQ      24000000 // clock frequency, 24 MHz

#define ROT_IRQ         0           // rotary interrupt number

#define PTRNL          0           // null pointer reference
#define FALSE          0
#define TRUE           !FALSE

#endif
```

The rest of the code used can be found at Appendix D.2.

A Timing Diagrams

A.1 ADC

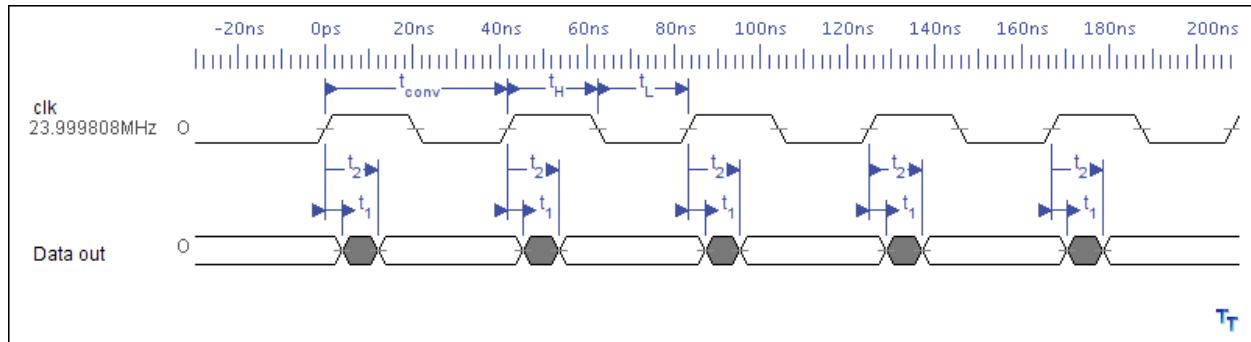


Figure 40: ADC Timing Diagram

A.2 LCD

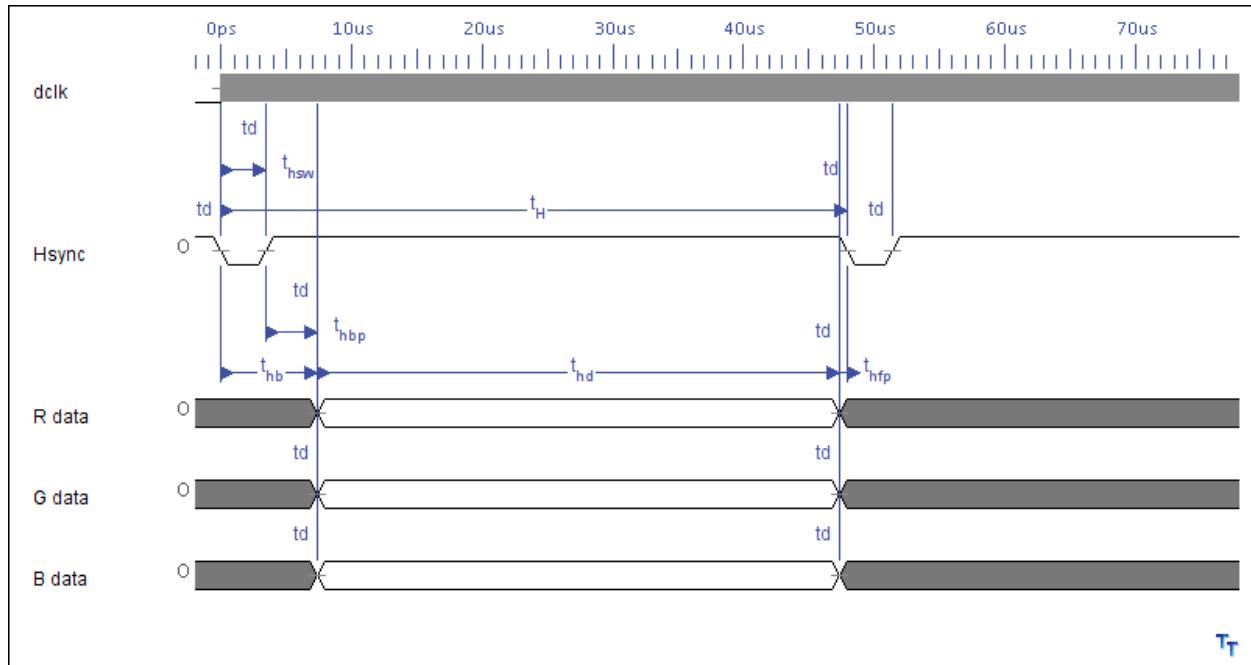


Figure 41: LCD Horizontal Timing Diagram

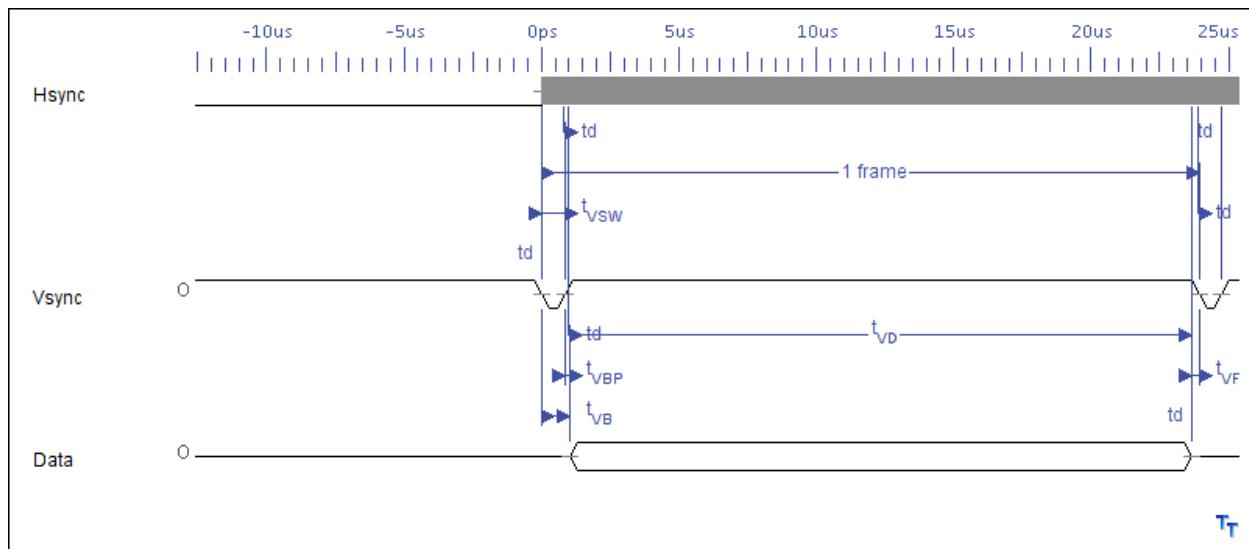


Figure 42: LCD Vertical Timing Diagram

A.3 ROM and RAM

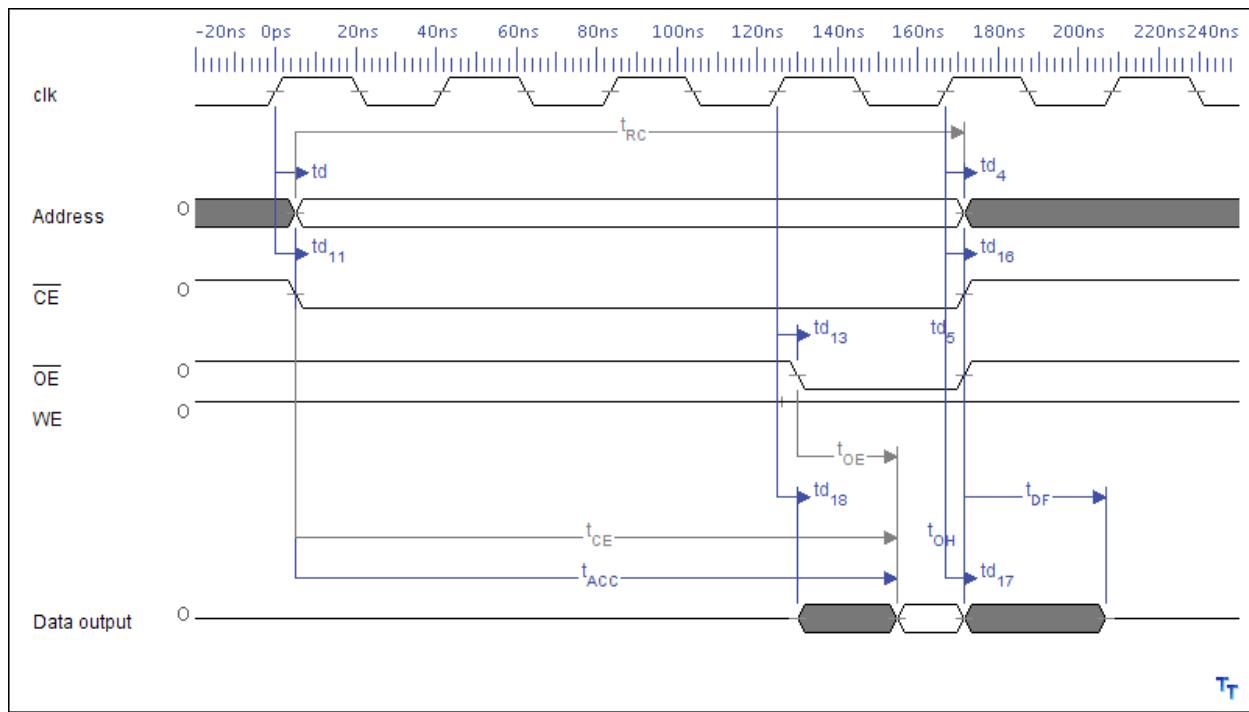


Figure 43: ROM Read Cycle Diagram

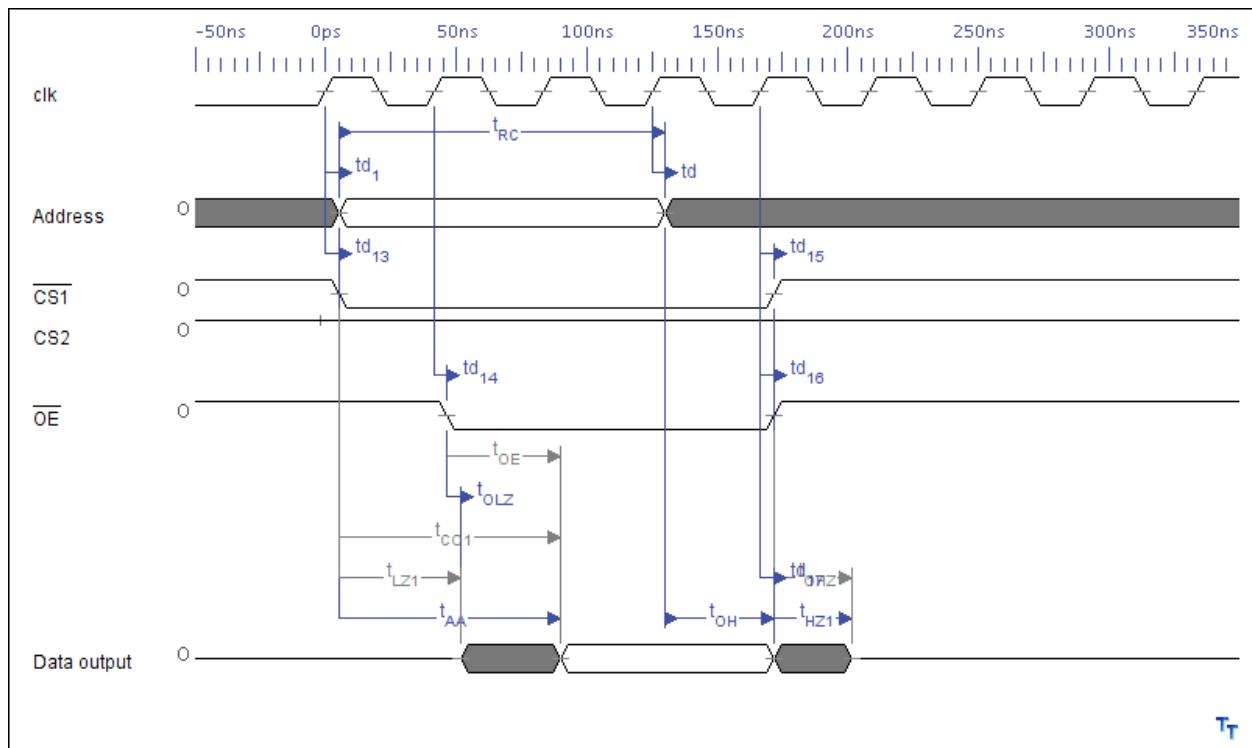


Figure 44: RAM Read Cycle Diagram

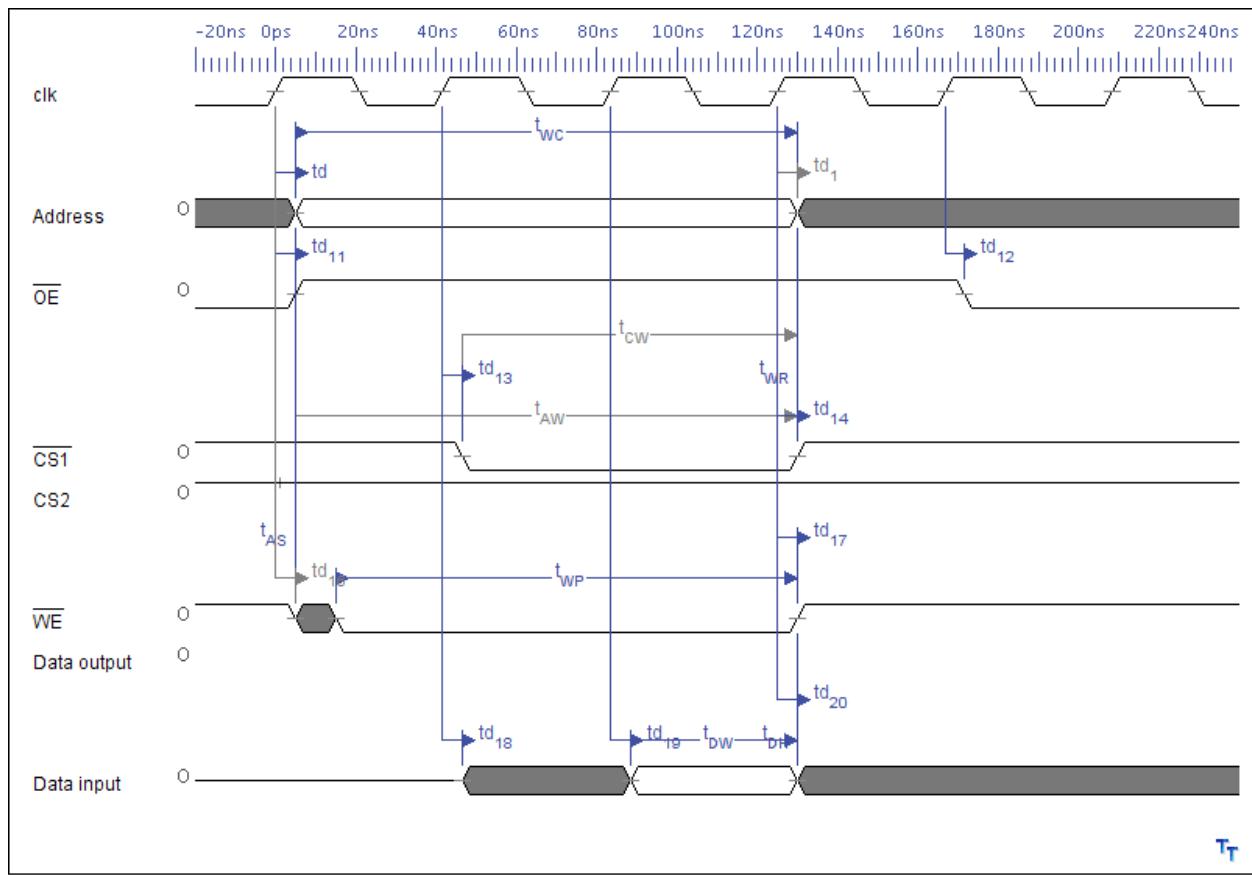


Figure 45: RAM Write Cycle Diagram

A.4 VRAM

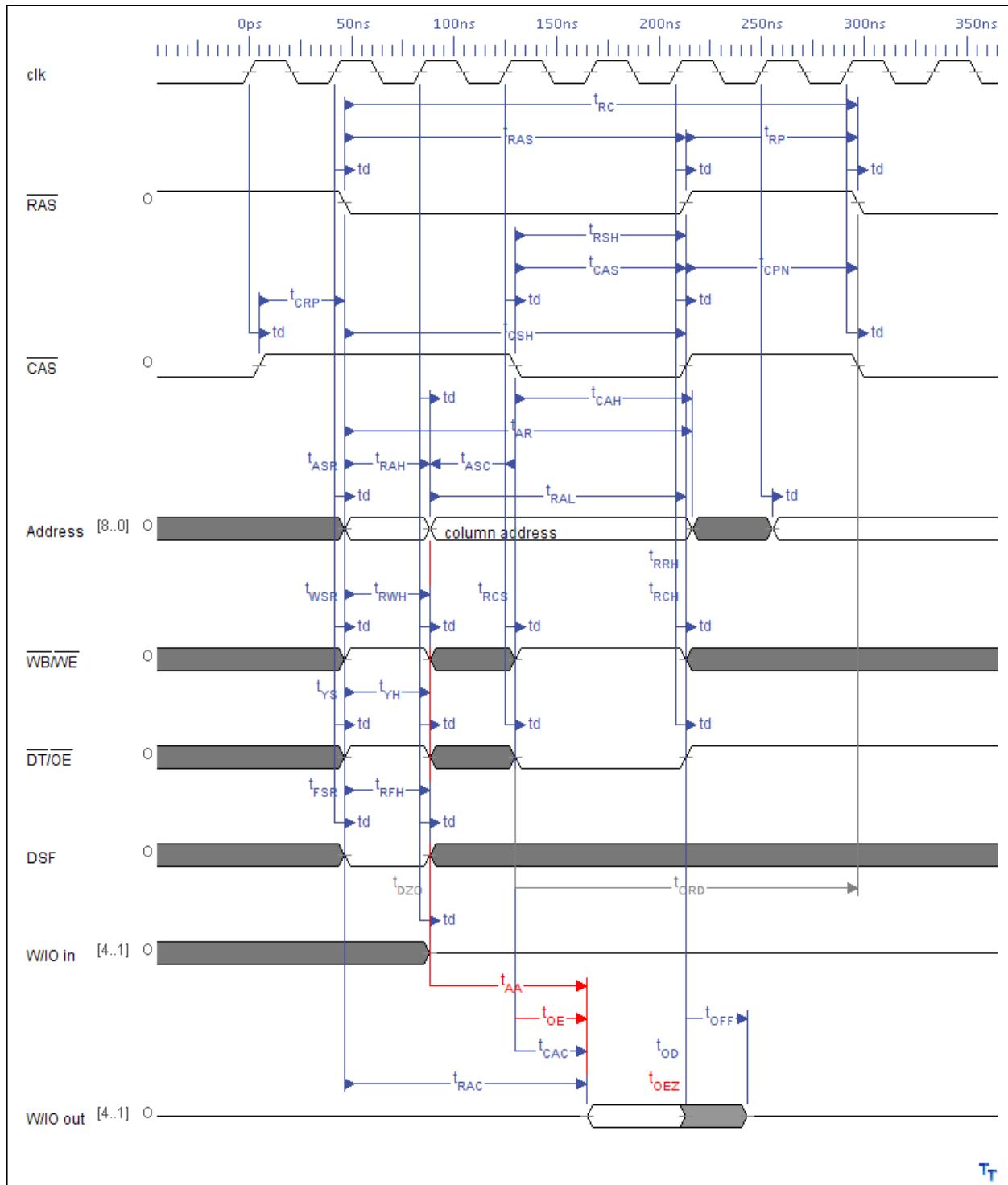


Figure 46: VRAM Read Cycle Diagram

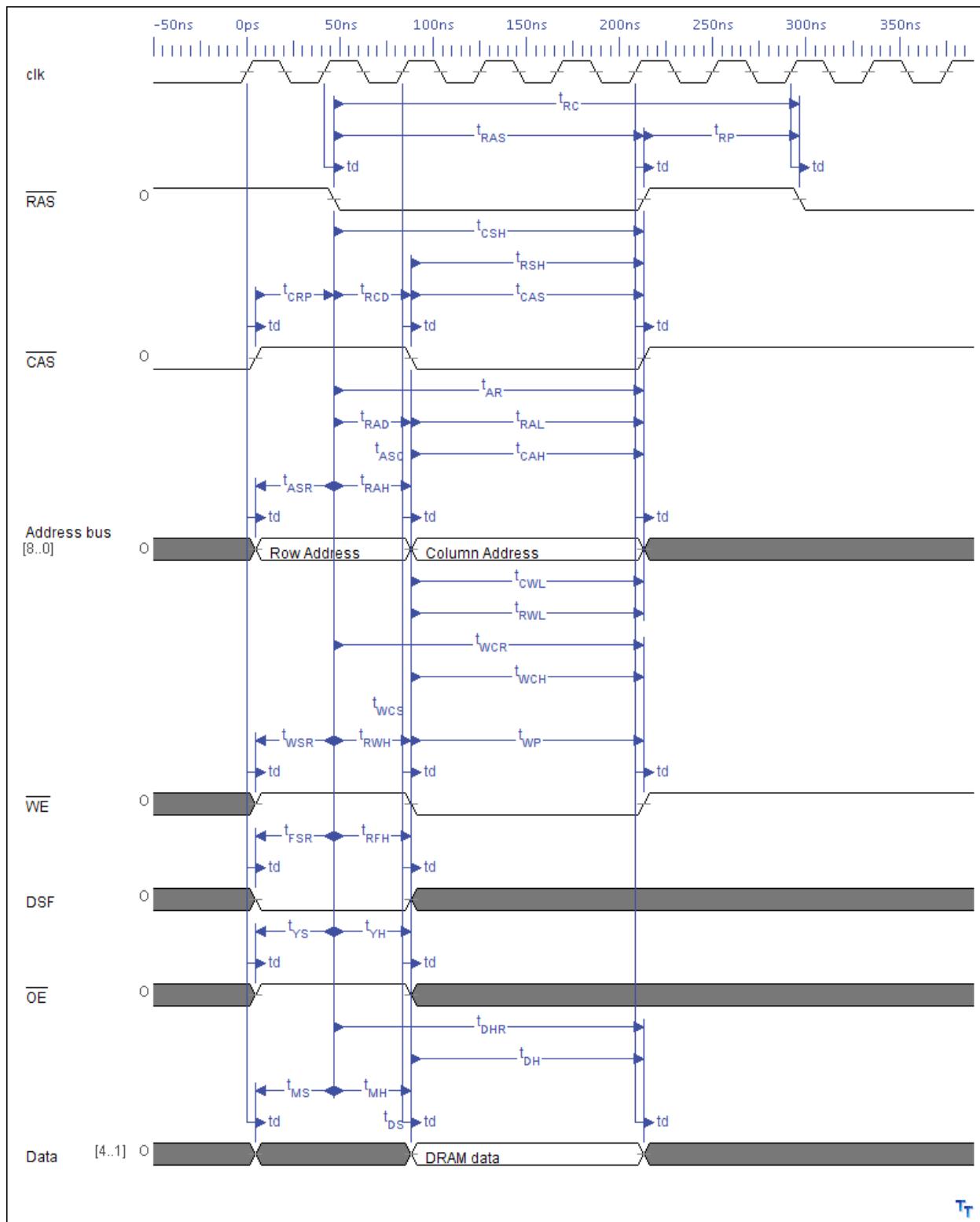


Figure 47: VRAM Write Cycle Diagram

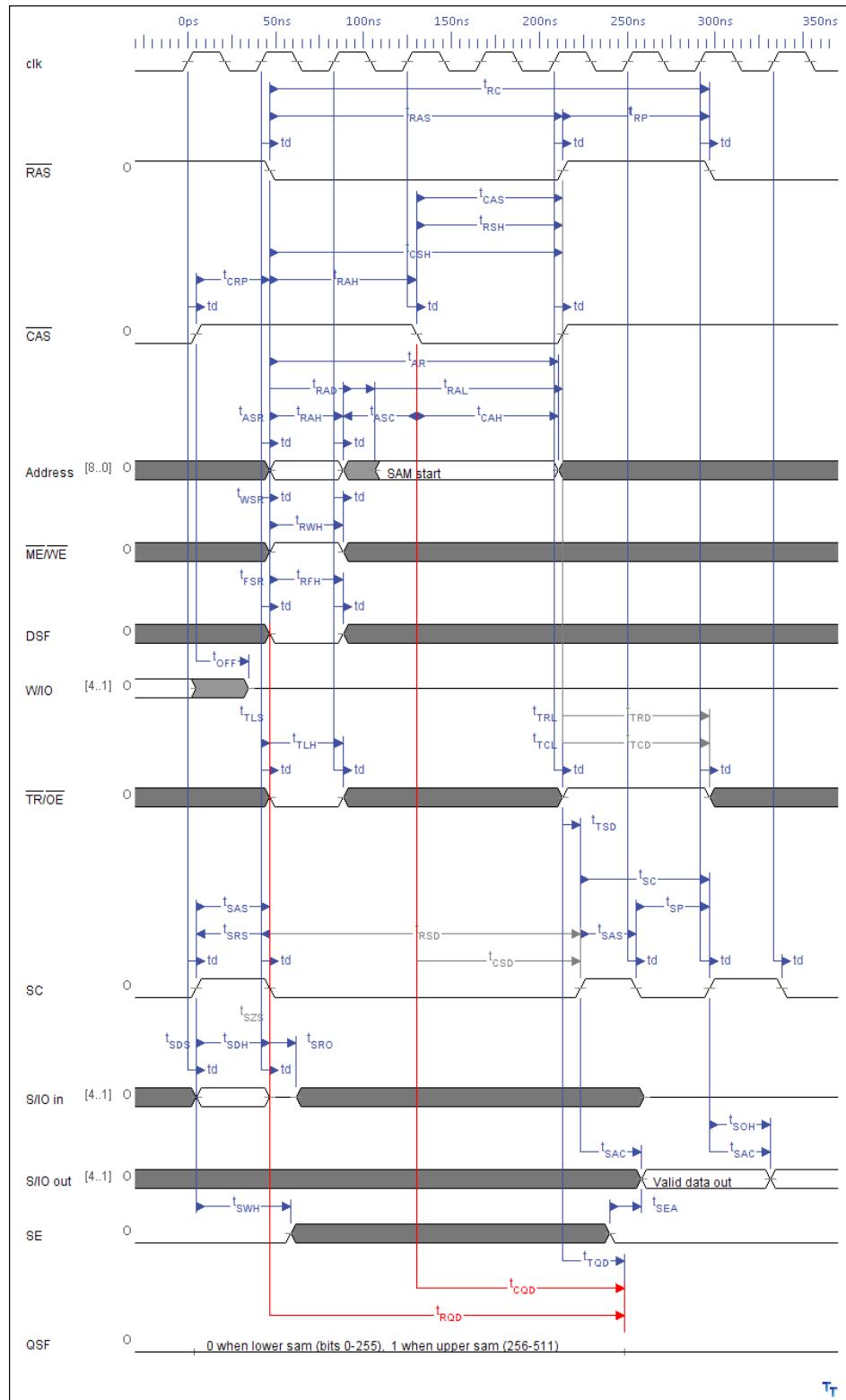


Figure 48: VRAM Row Transfer Cycle Diagram

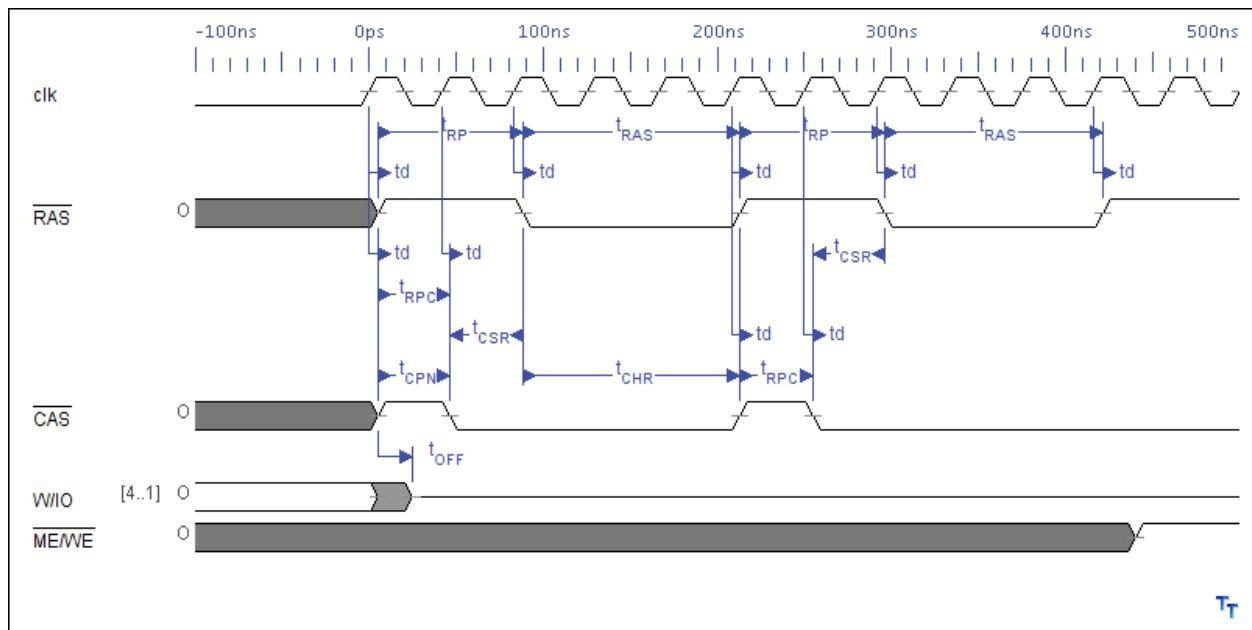


Figure 49: VRAM Refresh Cycle Diagram (CAS before RAS)

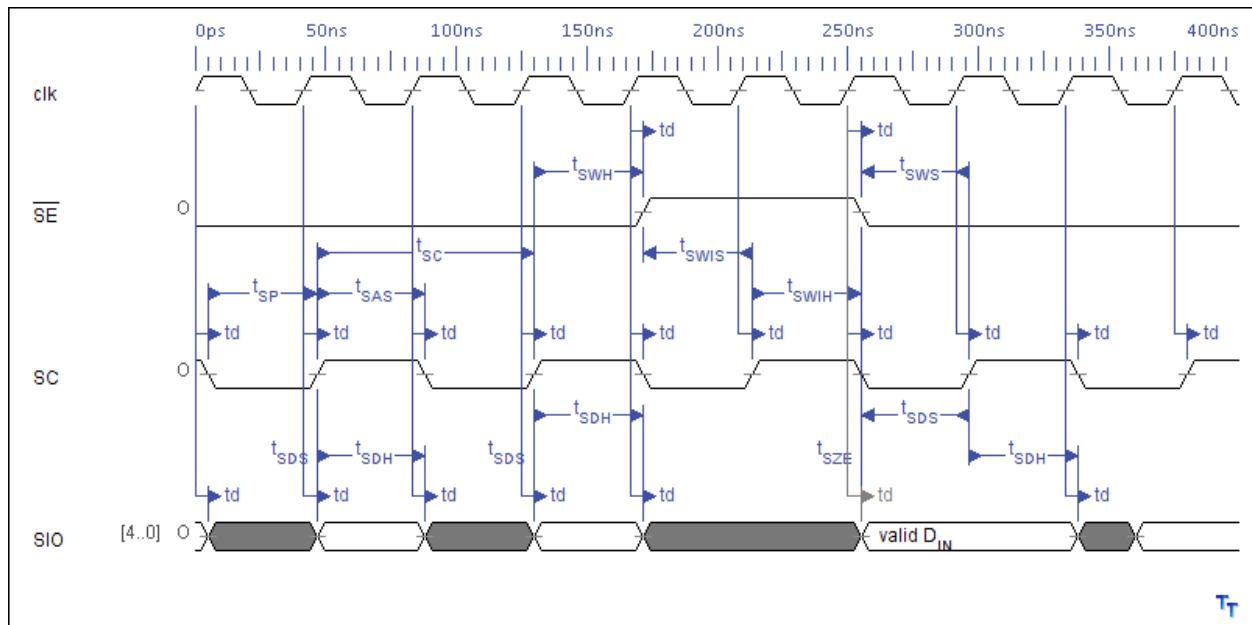


Figure 50: VRAM Serial Write Cycle Diagram

B Device Images

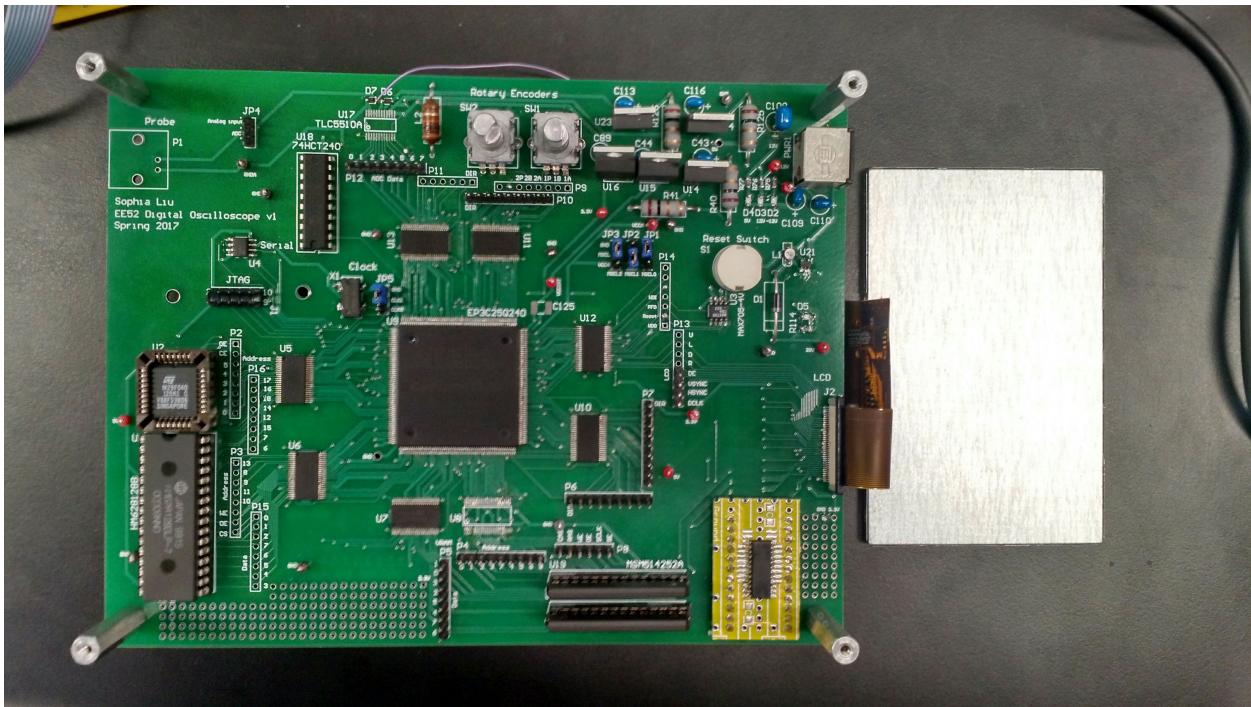


Figure 51: Front of device

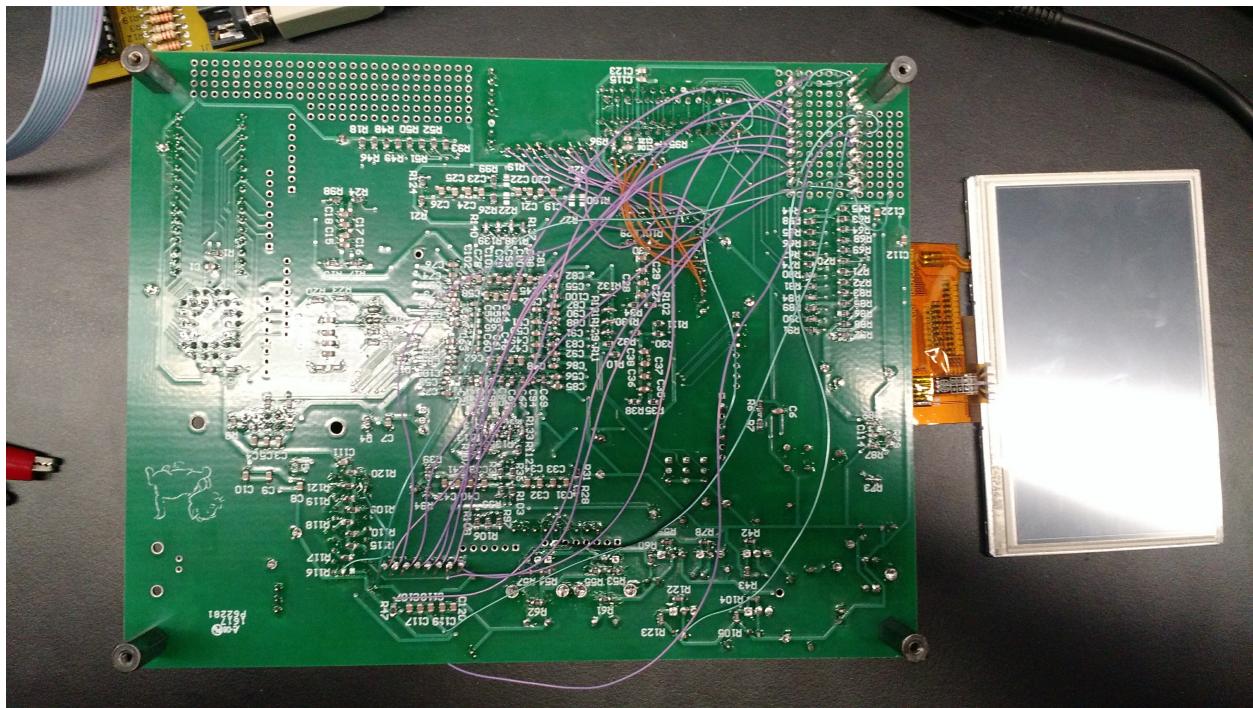


Figure 52: Back of device

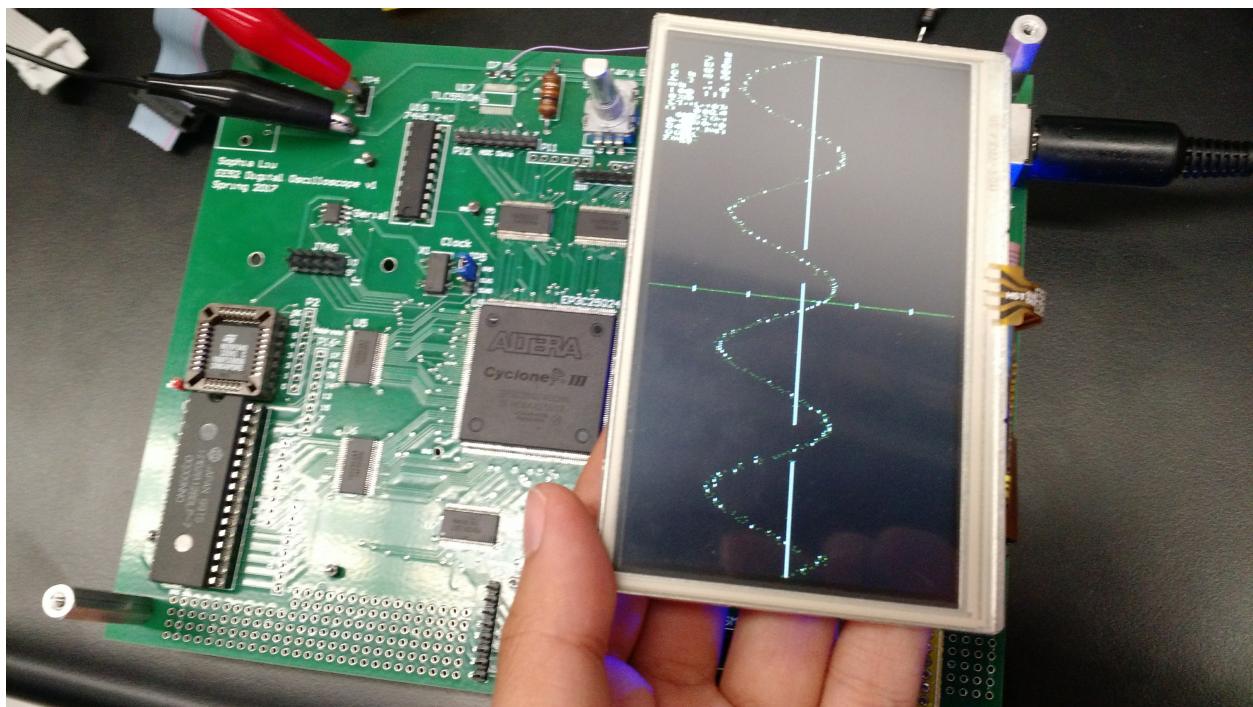
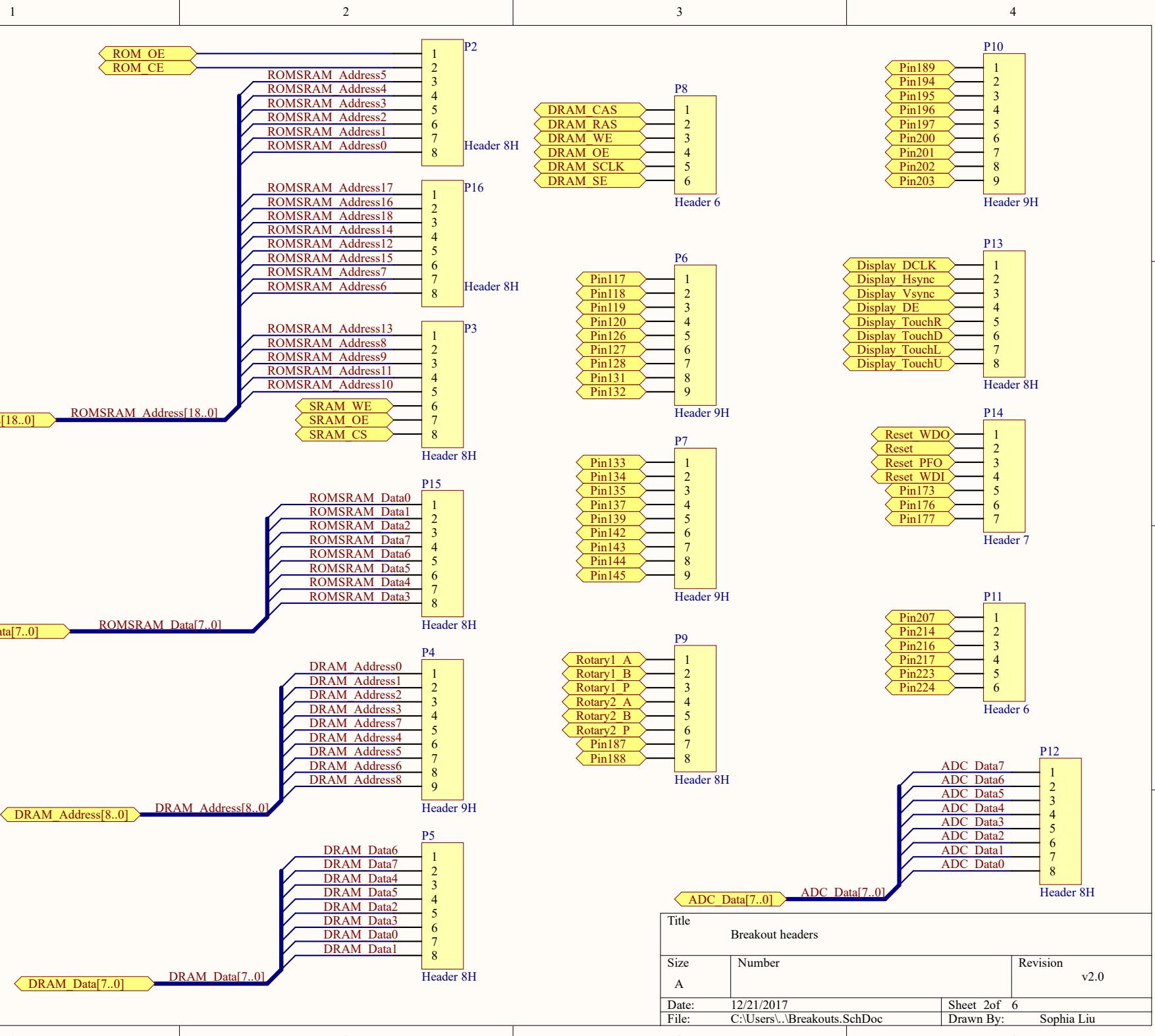
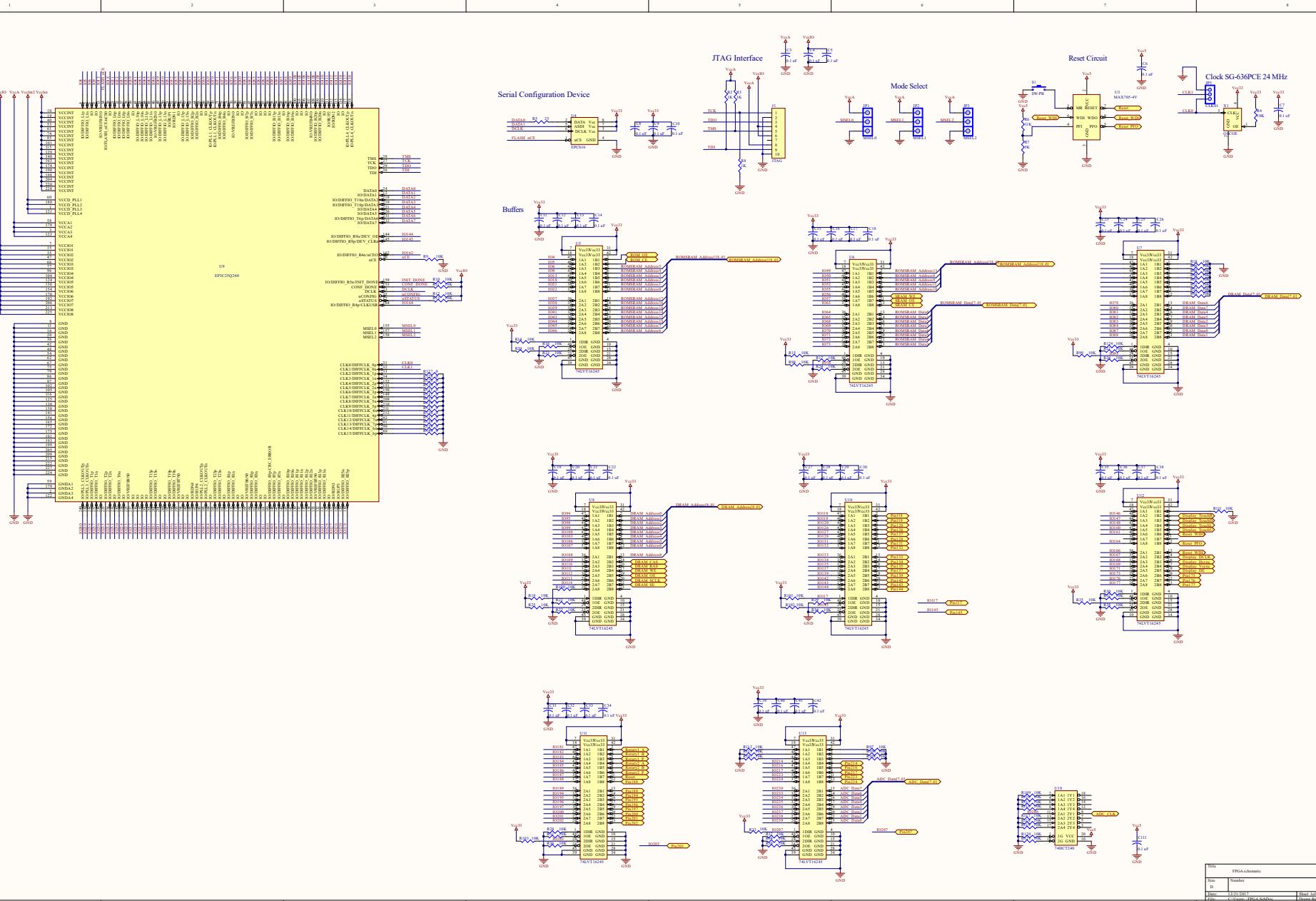


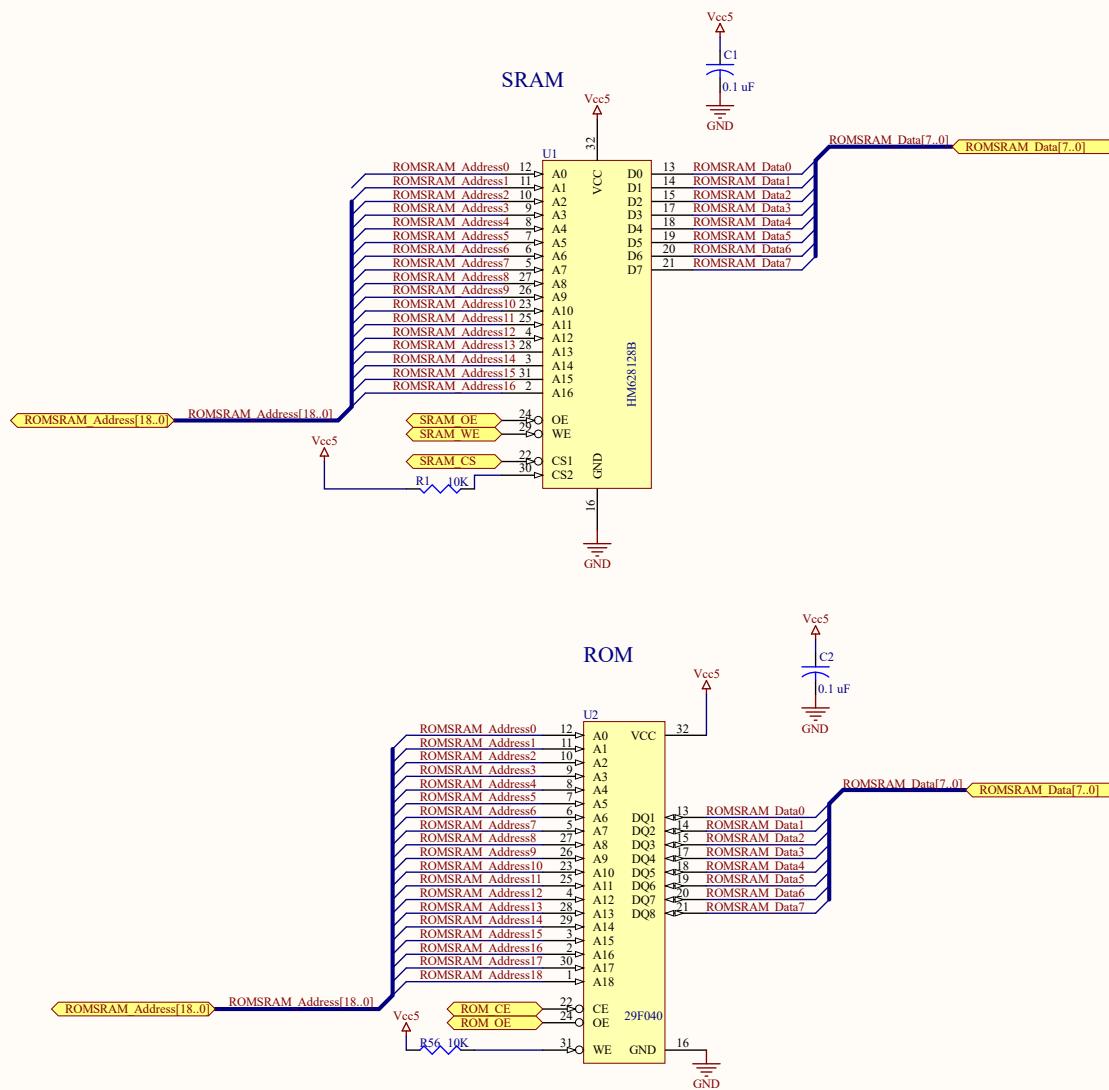
Figure 53: Device running with sinusoidal input

C Schematics and PCB

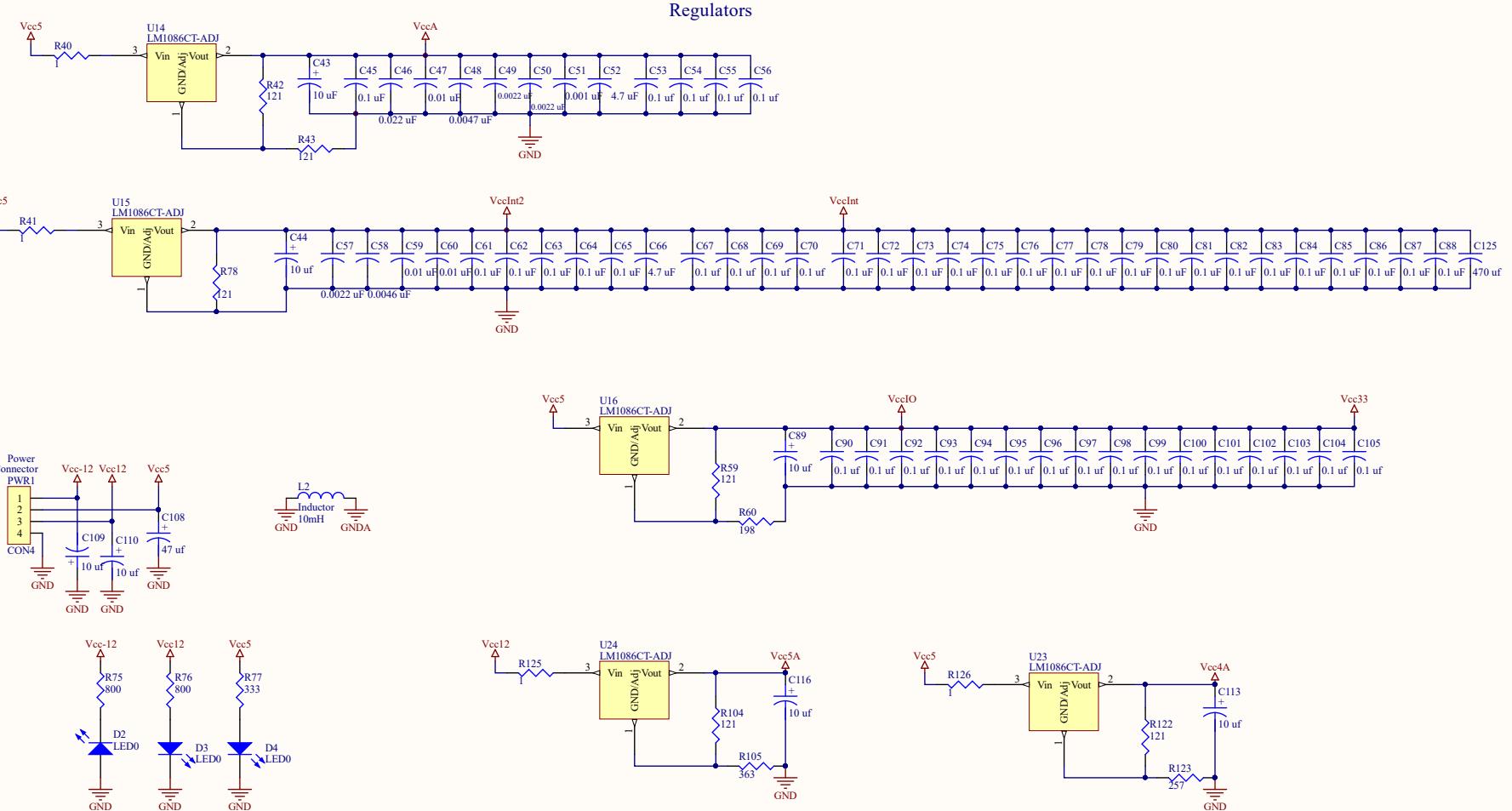




FPGA schematic		
Size	Number	Revision
D		v2.0
Date:	12/21/2017	Sheet Inf 6
Title:	C:\Users\JINGA\ScalDoc	Draw By: Sogolishan

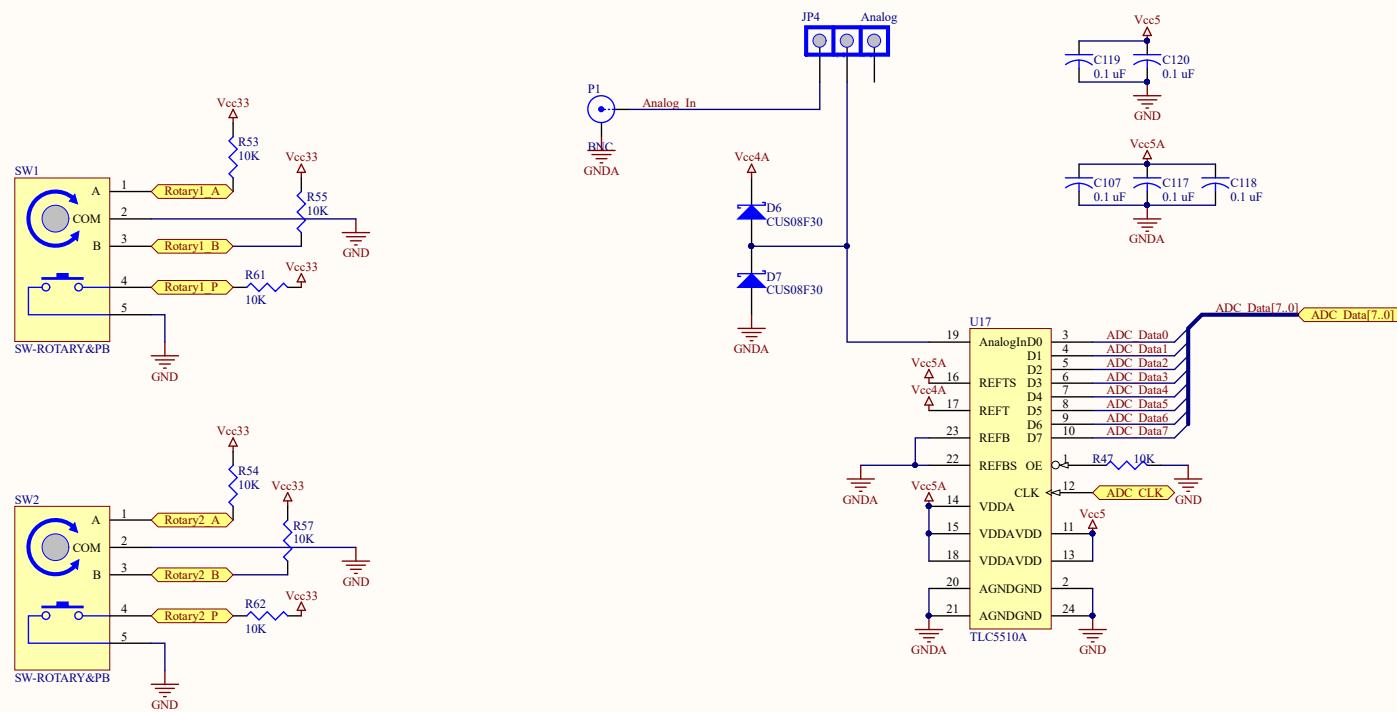


Title		
Size		Number
B		Revision v2.0
Date:	12/21/2017	Sheet 4 of 6
File:	C:\Users\Memory.SchDoc	Drawn By: Sophia Liu



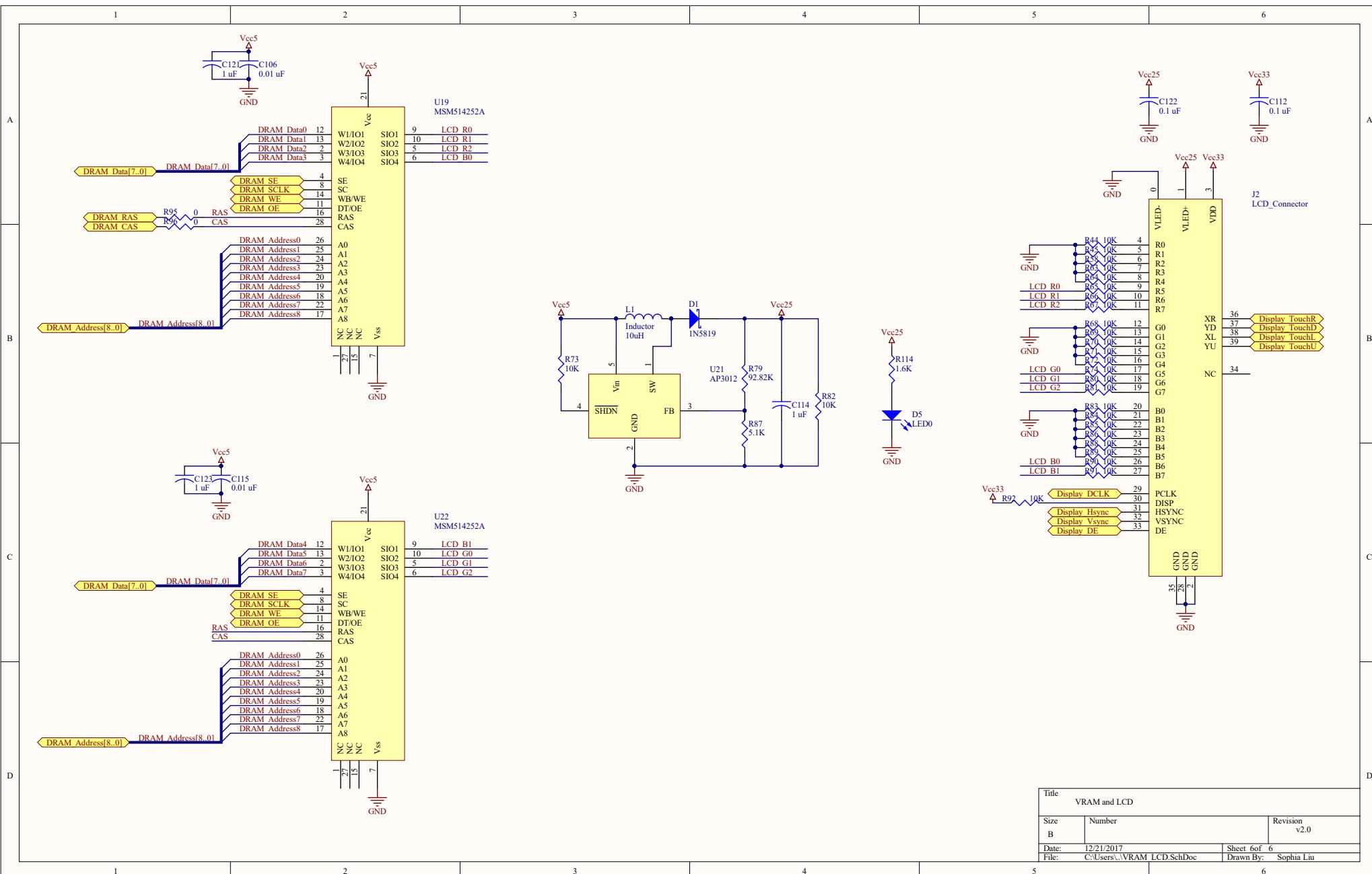
Title		Power circuitry		
Size	Number	Revision		v2.0
B				
Date:	12/21/2017	Sheet	3 of 6	
File:	C:\Users\l\PowerschDoc	Drawn By:		Sophia Liu

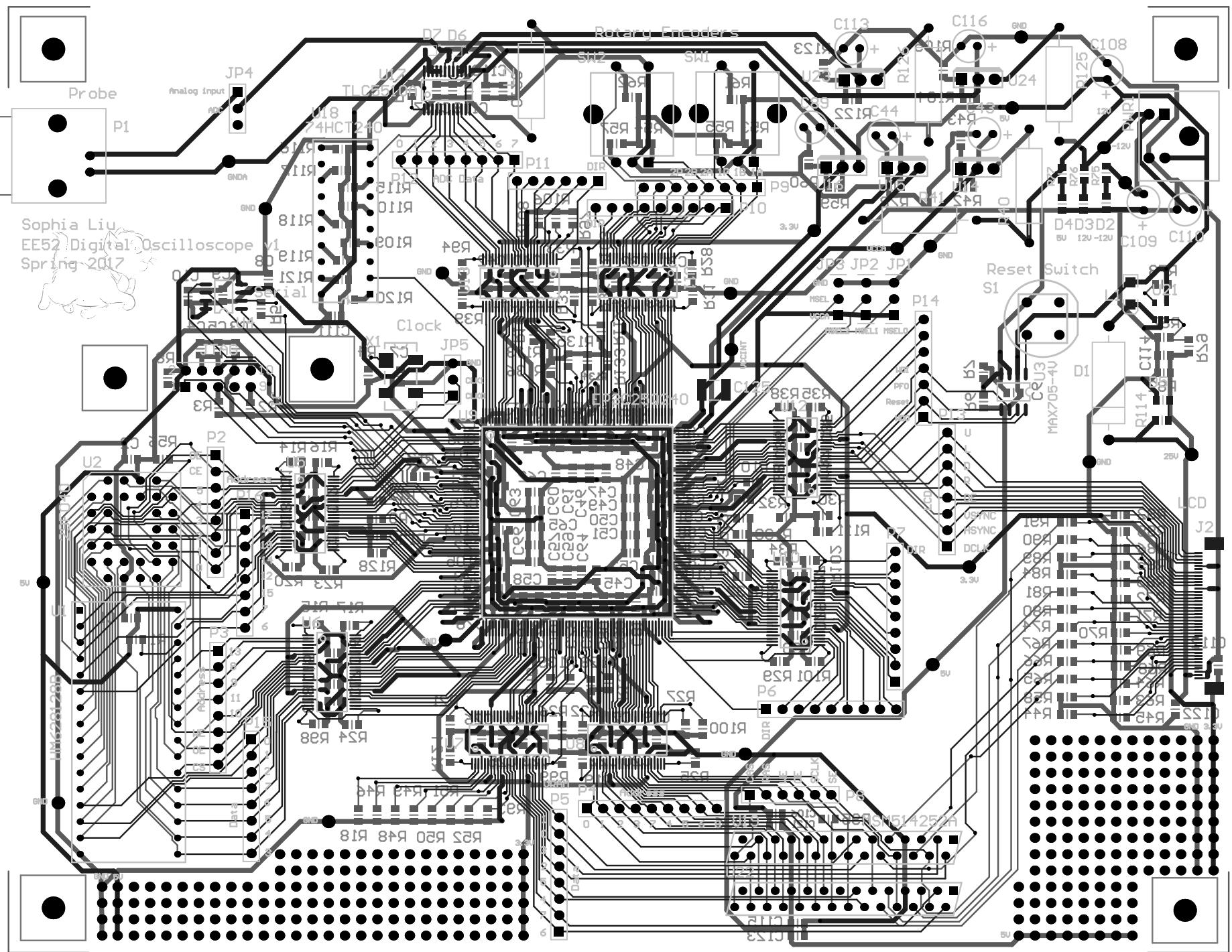
1 2 3 4 5 6



Title User and Analog Input		
Size B	Number	Revision v2.0
Date: 12/21/2017	Sheet 5 of 6	
File: C:\Users\...\User_Analog_Input.SchDoc		Drawn By: Sophia Liu

1 2 3 4 5 6





D Software

D.1 Hardware descriptions

Listing 6: data/scoptrig.vhd

```
--  
-- Oscilloscope Digital Trigger  
--  
-- This is an implementation of a trigger for a digital oscilloscope in  
-- VHDL. There are three inputs to the system, one selects the trigger  
-- slope and the other two determine the relationship between the trigger  
-- level and the signal level. The only output is a trigger signal which  
-- indicates a trigger event has occurred.  
--  
-- The file contains multiple architectures for a Moore state machine  
-- implementation to demonstrate the different ways of building a state  
-- machine.  
--  
--  
-- Revision History:  
-- 13 Apr 04 Glen George Initial revision.  
-- 4 Nov 05 Glen George Updated comments.  
-- 17 Nov 07 Glen George Updated comments.  
-- 13 Feb 10 Glen George Added more example architectures.  
--  
-- bring in the necessary packages  
library ieee;  
use ieee.std_logic_1164.all;  
  
--  
-- Oscilloscope Digital Trigger entity declaration  
--  
  
entity ScopeTrigger is  
    port (  
        TS      : in std_logic;      -- trigger slope (1 -> negative, 0 ->  
                                     -- positive)  
        TEQ     : in std_logic;      -- signal and trigger levels equal  
        TLT     : in std_logic;      -- signal level < trigger level  
        clk     : in std_logic;      -- clock  
        Reset   : in std_logic;      -- reset the system  
        TrigEvent : out std_logic   -- a trigger event has occurred  
    );  
end ScopeTrigger;
```

```

-- 
-- Oscilloscope Digital Trigger Moore State Machine
-- State Assignment Architecture
-- 

-- This architecture just shows the basic state machine syntax when the state
-- assignments are made manually. This is useful for minimizing output
-- decoding logic and avoiding glitches in the output (due to the decoding
-- logic).
-- 

architecture assign_statebits of ScopeTrigger is

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

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

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

begin

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

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

    transition: process (Reset, TS, TEQ, TLT, CurrentState)
    begin

        case CurrentState is          -- do the state transition/output

            when IDLE =>           -- in idle state, do transition
                if (TS = '0' and TLT = '1' and TEQ = '0') then
                    NextState <= WAIT_POS;    -- below trigger and + slope
                elsif (TS = '1' and TLT = '0' and TEQ = '0') then

```

```

        NextState <= WAIT_NEG;      -- above trigger and - slope
    else
        NextState <= IDLE;       -- trigger not possible yet
    end if;

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

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

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

when others =>
    NextState <= IDLE;

end case;

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

end process transition;

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

process (clk)
begin

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

```

```

    end if;

end process;

end assign_statebits;

```

Listing 7: data/VRAM_FSM.vhd

```

--  

-- VRAM state machine  

--  

-- This is an implementation of a Moore state machine for the VRAM controller  

-- in VHDL. The inputs to the system are the chip select, write enable,  

-- transfer, clock, and reset enable signals. The outputs are the RAS, CAS,  

-- address selecting, write enable, output enable, transfer acknowledge,  

-- CPU wait, and serial clock enable signals for the VRAM.  

--  

-- Revision History:  

-- 02/17/2017 Sophia Liu Initial Revision  

-- 09/24/2017 Sophia Liu Edited comments  

-- 12/2017 Sophia Liu Edited comments

```

```

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

--  

-- VRAM controller FSM entity declaration
--  

entity VRAM_FSM is
  port (
    CS      : in std_logic;      -- chip select
    WE_in   : in std_logic;      -- write enable
    Transfer : in std_logic;     -- row transfer
    clk     : in std_logic;      -- clock
    Reset   : in std_logic;      -- reset the system
    RAS     : out std_logic;     -- row address strobe
    CAS     : out std_logic;     -- column address strobe
    Row1    : out std_logic;     -- 10 = row address, 11 = column address

```

```

Row0      : out std_logic;    -- 01 = sam start, 00 = current row
WE        : out std_logic;    -- write enable
OE        : out std_logic;    -- output enable/data transfer
Acknowledge : out std_logic; -- finished row transfer
CPU_Wait   : out std_logic;  -- wait signal to cpu, 1 on last state
                             -- of read/write/transfer/refresh
SC        : out std_logic;    -- serial clock enabled if 1, off if 0
);
end VRAM_FSM;

```

```

--
-- Oscilloscope VRAM Moore State Machine
--

architecture assign_statebits of VRAM_FSM is

subtype states is std_logic_vector(8 downto 0); -- state type

-- define the actual states as constants
constant IDLE      : states := "110011000"; -- Waiting

constant ROW_1      : states := "110011001"; -- Row transfer states
constant ROW_2      : states := "010010000"; --
constant ROW_3      : states := "010111000"; --
constant ROW_4      : states := "000111000"; --
constant ROW_5      : states := "000111001"; --
constant ROW_6      : states := "110111000"; --
constant ROW_7      : states := "110111001"; --

constant READ_1     : states := "111011010"; -- Read cycle states
constant READ_2     : states := "011011000"; --
constant READ_3     : states := "011111000"; --
constant READ_4     : states := "001110100"; -- wait state
constant READ_5     : states := "001110001"; --
constant READ_6     : states := "111111000"; --
constant READ_7     : states := "111111010"; --

constant WRITE_1    : states := "111011011"; -- write cycle states
constant WRITE_2    : states := "011011101"; -- wait state
constant WRITE_3    : states := "001101000"; --
constant WRITE_4    : states := "001101001"; --
constant WRITE_5    : states := "001101010"; --
constant WRITE_6    : states := "111111001"; --

```

```

constant WRITE_7    : states := "111111011"; --
constant REFRESH_1  : states := "110011010"; -- refresh cycle states
constant REFRESH_2  : states := "100011000"; --
constant REFRESH_3  : states := "000011000"; --
constant REFRESH_4  : states := "000011001"; --
constant REFRESH_5  : states := "001011011"; --
constant REFRESH_6  : states := "111011000"; --
constant REFRESH_7  : states := "101011000"; --
constant REFRESH_8  : states := "001011000"; --
constant REFRESH_9  : states := "001011001"; --
constant REFRESH_10 : states := "001111000"; --
constant REFRESH_11 : states := "100011011"; --
constant REFRESH_12 : states := "100011001"; --
constant REFRESH_13 : states := "000011011"; --
constant REFRESH_14 : states := "110011011"; --

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

begin

    -- output is state bits
    RAS <= CurrentState(8);
    CAS <= CurrentState(7);
    ROW1 <= CurrentState(6);
    ROW0 <= CurrentState(5);
    WE  <= CurrentState(4);
    OE  <= CurrentState(3);
    CPU_Wait <= CurrentState(2);
    Acknowledge <= '1' when (CurrentState = ROW_7)
                           else '0';
    SC <= '0' when (CurrentState = ROW_2 or CurrentState = Row_3 or CurrentState
                     = Row_4
                     or CurrentState = Row_5)
                     else '1';

transition: process (CS, WE_in, Transfer, Reset, CurrentState)
begin

    case CurrentState is           -- do the state transition/output

        when IDLE =>             -- in idle state, do transition
            if (Transfer = '1') then

```

```

        NextState <= ROW_1;      -- prioritize row transfer
    elsif (CS = '0' and WE_in = '0') then
        NextState <= WRITE_1;  -- begin write
    elsif (CS = '0' and WE_in = '1') then
        NextState <= READ_1;   -- begin read
    else
        NextState <= REFRESH_1; -- if not doing anything else
                                -- then refresh
    end if;

when ROW_1 =>           -- Row cycle
    NextState <= ROW_2;

when ROW_2 =>           -- Row cycle
    NextState <= ROW_3;

when ROW_3 =>           -- Row cycle
    NextState <= ROW_4;

when ROW_4 =>           -- Row cycle
    NextState <= ROW_5;

when ROW_5 =>           -- Row cycle
    NextState <= ROW_6;

when ROW_6 =>           -- Row cycle
    NextState <= ROW_7;

when ROW_7 =>           -- End row cycle
    NextState <= IDLE;

when READ_1 =>          -- Read cycle
    NextState <= READ_2;

when READ_2 =>          -- Read cycle
    NextState <= READ_3;

when READ_3 =>          -- Read cycle
    NextState <= READ_4;

when READ_4 =>          -- Read cycle
    NextState <= READ_5;

when READ_5 =>          -- Read cycle
    NextState <= READ_6;

```

```

when READ_6 =>           -- Read cycle
    NextState <= READ_7;

when READ_7 =>           -- End read cycle
    NextState <= IDLE;

when WRITE_1 =>          -- Write cycle
    NextState <= WRITE_2;

when WRITE_2 =>          -- Write cycle
    NextState <= WRITE_3;

when WRITE_3 =>          -- Write cycle
    NextState <= WRITE_4;

when WRITE_4 =>          -- Write cycle
    NextState <= WRITE_5;

when WRITE_5 =>          -- Write cycle
    NextState <= WRITE_6;

when WRITE_6 =>          -- Write cycle
    NextState <= WRITE_7;

when WRITE_7 =>          -- End write cycle
    NextState <= IDLE;

when REFRESH_1 =>         -- Refresh cycle
    NextState <= REFRESH_2;

when REFRESH_2 =>         -- Refresh cycle
    NextState <= REFRESH_3;

when REFRESH_3 =>         -- Refresh cycle
    NextState <= REFRESH_4;

when REFRESH_4 =>         -- Refresh cycle
    NextState <= REFRESH_5;

when REFRESH_5 =>         -- Refresh cycle
    NextState <= REFRESH_6;

when REFRESH_6 =>         -- Refresh cycle
    NextState <= REFRESH_7;

```

```

when REFRESH_7 =>          -- Refresh cycle
    NextState <= REFRESH_8;

when REFRESH_8 =>          -- Refresh cycle
    NextState <= REFRESH_9;

when REFRESH_9 =>          -- Refresh cycle
    NextState <= REFRESH_10;

when REFRESH_10 =>          -- Refresh cycle
    NextState <= REFRESH_11;

when REFRESH_11 =>          -- Refresh cycle
    NextState <= REFRESH_12;

when REFRESH_12 =>          -- Refresh cycle
    NextState <= REFRESH_13;

when REFRESH_13 =>          -- Refresh cycle
    NextState <= REFRESH_14;

when REFRESH_14 =>          -- End refresh cycle
    NextState <= IDLE;

when others =>
    NextState <= IDLE;

end case;

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

end process transition;

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

process (clk)
begin

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

```

```
end process;  
  
end assign_statebits;
```

D.2 Main code

Listing 8: ..//osc134/mainloop.c

```
*****  
/* */  
/* MAINLOOP */  
/* Main Program Loop */  
/* Digital Oscilloscope Project */  
/* EE/CS 52 */  
/* */  
*****  
  
/*  
This file contains the main processing loop (background) for the Digital  
Oscilloscope project. The only global function included is:  
    main - background processing loop  
  
The local functions included are:  
    key_lookup - get a key and look up its keycode  
  
The locally global variable definitions included are:  
    none  
  
Revision History  
3/8/94 Glen George    Initial revision.  
3/9/94 Glen George    Changed initialized const arrays to static  
                      (in addition to const).  
3/9/94 Glen George    Moved the position of the const keyword in  
                      declarations of arrays of pointers.  
3/13/94 Glen George   Updated comments.  
3/13/94 Glen George   Removed display_menu call after plot_trace,  
                      the plot function takes care of the menu.  
3/17/97 Glen George   Updated comments.  
3/17/97 Glen George   Made key_lookup function static to make it  
                      truly local.  
3/17/97 Glen George   Removed KEY_UNUSED and KEYCODE_UNUSED  
                      references (no longer used).  
5/27/08 Glen George   Changed code to only check for sample done if  
                      it is currently sampling.  
08/17 Sophia Liu      Added watchdog reset pulsing  
*/  
  
/* library include files */  
/* none */
```

```

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

/* local function declarations */
static enum keycode key_lookup(void); /* translate key values into keycodes */

/*
main

Description: This procedure is the main program loop for the Digital
Oscilloscope. It loops getting keys from the keypad,
processing those keys as is appropriate. It also handles
starting scope sample collection and updating the LCD
screen.

Arguments: None.
Return Value: (int) - return code, always 0 (never returns).

Input: Keys from the keypad.
Output: Traces and menus to the display.

Error Handling: Invalid input is ignored.

Algorithms: The function is table-driven. The processing routines
for each input are given in tables which are selected
based on the context (state) the program is operating in.

Data Structures: Array (process_key) to associate keys with actions
(functions to call).

Global Variables: None.

Author: Glen George
Last Modified: May 27, 2008

```

```

*/
int main()
{
    /* variables */
    enum keycode      key;          /* an input key */

    enum status       state = MENU_ON; /* current program state */

    unsigned char /*far*/ *sample;    /* a captured trace */

    /* key processing functions (one for each system state type and key) */
    static enum status (* const process_key[NUM_KEYCODES][NUM_STATES])(enum
        status) =
    {
        /* Current System State                      */
        /* MENU_ON      MENU_OFF           Input Key */
        { { menu_key,   menu_key },   /* <Menu>      */
          { menu_up,     no_action }, /* <Up>         */
          { menu_down,   no_action }, /* <Down>        */
          { menu_left,   no_action }, /* <Left>        */
          { menu_right,  no_action }, /* <Right>       */
          { no_action,   no_action } }; /* illegal key */
    };

    pulse_wd(); /* pulse watchdog reset input */

    /* first initialize everything */
    clear_display(); /* clear the display */

    plot_pixel(428, 10, PIXEL_BLACK);

    init_trace(); /* initialize the trace routines */
    init_menu(); /* initialize the menu system */
    initKeypad(); /* initialize keypad */
    init_analog(); /* initialize analog system */

    /* infinite loop processing input */
    while(TRUE) {
        pulse_wd(); /* pulse watchdog reset input */

        /* check if ready to do a trace */
        if (trace_rdy())
            /* ready for a trace - do it */
            do_trace();
    }
}

```

```

/* check if have a trace to display */
if (is_sampling() && ((sample = sample_done()) != NULL)) {

    /* have a trace - output it */
    plot_trace(sample);
    /* done processing this trace */
    trace_done();
}

/* now check for keypad input */
if (key_available()) {

    /* have keypad input - get the key */
    key = key_lookup();

    /* execute processing routine for that key */
    state = process_key[key][state](state);
}
}

/* done with main (never should get here), return 0 */
return 0;
}

/*
key_lookup

Description: This function gets a key from the keypad and translates
the raw keycode to an enumerated keycode for the main
loop.

Arguments: None.
Return Value: (enum keycode) - type of the key input on keypad.

Input: Keys from the keypad.
Output: None.

Error Handling: Invalid keys are returned as KEYCODE_ILLEGAL.

Algorithms: The function uses an array to lookup the key types.

```

Data Structures: Array of key types versus key codes.

Global Variables: None.

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

```
*/  
  
static enum keycode key_lookup()  
{  
    /* variables */  
  
    const static enum keycode keycodes[] = /* array of keycodes */  
    {  
        /* order must match keys array exactly */  
        KEYCODE_MENU,      /* <Menu>      */ /* also need an extra element */  
        KEYCODE_UP,        /* <Up>        */ /* for unknown key codes */  
        KEYCODE_DOWN,      /* <Down>      */  
        KEYCODE_LEFT,      /* <Left>      */  
        KEYCODE_RIGHT,     /* <Right>     */  
        KEYCODE_ILLEGAL   /* other keys */  
    };  
  
    const static int keys[] = /* array of key values */  
    {  
        /* order must match keycodes array exactly */  
        KEY_MENU,        /* <Menu>      */  
        KEY_UP,          /* <Up>        */  
        KEY_DOWN,         /* <Down>      */  
        KEY_LEFT,         /* <Left>      */  
        KEY_RIGHT,        /* <Right>     */  
    };  
  
    int key;      /* an input key */  
  
    int i;        /* general loop index */  
  
    /* get a key */  
    key = getkey();  
  
    /* lookup key in keys array */  
    for (i = 0; ((i < (sizeof(keys)/sizeof(int))) && (key != keys[i])); i++);
```

```
    /* return the appropriate key type */
    return keycodes[i];

}
```

Listing 9: ..//osc134/scopedef.h

```
/******************************************************************************/
/*
 *                      SCOPEDEF.H
 *          General Definitions
 *          Include File
 *          Digital Oscilloscope Project
 *          EE/CS 52
 */
/******************************************************************************/
/*
 This file contains the general definitions for the Digital Oscilloscope
 project. This includes constant and structure definitions along with the
 function declarations for the assembly language functions.

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

```

```
#ifndef __SCOPEDEF_H__
#define __SCOPEDEF_H__


/* library include files */
```

```

/* none */

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

/* constants */

/* general constants */
#define FALSE      0
#define TRUE       !FALSE
#define NULL       (void *) 0

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

/* macros */

/* let __nios__ also mean a NIOS compilation */
#ifndef __nios__
#define NIOS        /* use the standard NIOS defintion */
#endif

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

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

/* structures, unions, and typedefs */

```

```

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

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

/* function declarations */

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

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

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

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

/* watchdog input functions */
void pulse_wd(void); /* pulse watchdog input on reset chip */

#endif

```

Listing 10: ..//osc134/interfac.h

```

/*********************  

/*  

/*          INTERFAC.H  

/*          Interface Definitions  

/*          Include File  

/*          Digital Oscilloscope Project  

/*          EE/CS 52  

/*  

/*  

/*********************  

/*  

This file contains the constants for interfacing between the C code and  

the assembly code/hardware for the Digital Oscilloscope project. This is  

a sample interface file to allow compilation of the .c files.  

Revision History:  

  3/8/94  Glen George      Initial revision.  

  3/13/94 Glen George     Updated comments.  

  3/17/97 Glen George     Added constant MAX_SAMPLE_SIZE and removed  

                           KEY_UNUSED.  

  06/22/17 Sophia Liu      changed keypad and display constants  

*/

```

```

#ifndef __INTERFAC_H__
#define __INTERFAC_H__


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


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


/* constants */

/* keypad constants */
#define KEY_MENU      1 /* <Menu>      */
#define KEY_UP        2 /* <Up>        */
#define KEY_DOWN      4 /* <Down>      */

```

```

#define KEY_LEFT      8 /* <Left>      */
#define KEY_RIGHT     16 /* <Right>     */
#define KEY_ILLEGAL   0 /* illegal key */

/* display constants */
#define SIZE_X        480 /* size in the x dimension */
#define SIZE_Y        272 /* size in the y dimension */
#define PIXEL_WHITE   0 /* pixel off */
#define PIXEL_BLACK   0xff /* pixel on */

/* scope parameters */
#define MIN_DELAY     0 /* minimum trigger delay */
#define MAX_DELAY    50000 /* maximum trigger delay */
#define MIN_LEVEL     0 /* minimum trigger level (in mV) */
#define MAX_LEVEL    3800 /* maximum trigger level (in mV) */

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

#endif

```

Listing 11: ..//osc134/menu.c

```

*****
*/
/*
         MENU
Menu Functions
Digital Oscilloscope Project
EE/CS 52
*/
*****
*/

/*
This file contains the functions for processing menu entries for the
Digital Oscilloscope project. These functions take care of maintaining the
menus and handling menu updates for the system. The functions included
are:
    clear_menu      - remove the menu from the display
    display_menu    - display the menu
    init_menu       - initialize menus
    menu_entry_left - take care of <Left> key for a menu entry
    menu_entry_right - take care of <Right> key for a menu entry
    next_entry      - next menu entry
    previous_entry  - previous menu entry

```

```
refresh_menu      - re-display the menu if currently being displayed  
reset_menu        - reset the current selection to the top of the menu
```

The local functions included are:

```
display_entry     - display a menu entry (including option setting)
```

The locally global variable definitions included are:

```
menu              - the menu  
menu_display     - whether or not the menu is currently displayed  
menu_entry       - the currently selected menu entry
```

Revision History

```
3/8/94  Glen George    Initial revision.  
3/9/94  Glen George    Changed position of const keyword in array  
                        declarations involving pointers.  
3/13/94 Glen George    Updated comments.  
3/13/94 Glen George    Added display_entry function to output a menu  
                        entry and option setting to the LCD (affects  
                        many functions).  
3/13/94 Glen George    Changed calls to set_status due to changing  
                        enum scale_status definition.  
3/13/94 Glen George    No longer clear the menu area before  
                        restoring the trace in clear_menu() (not  
                        needed).  
3/17/97 Glen George    Updated comments.  
3/17/97 Glen George    Fixed minor bug in reset_menu().  
3/17/97 Glen George    When initializing the menu in init_menu(),  
                        set the delay to MIN_DELAY instead of 0 and  
                        trigger to a middle value instead of  
                        MIN_TRG_LEVEL_SET.  
5/3/06  Glen George    Changed to a more appropriate constant in  
                        display_entry().  
5/3/06  Glen George    Updated comments.  
5/9/06  Glen George    Changed menus to handle a list for mode and  
                        scale (move up and down list), instead of  
                        toggling values.
```

*/

```
/* library include files */
```

```
/* none */
```

```
/* local include files */
```

```

#include "scopedef.h"
#include "lcdout.h"
#include "menu.h"
#include "menuact.h"
#include "tracutil.h"

/* local function declarations */
static void display_entry(int, int); /* display a menu entry and its setting */

/* locally global variables */
static int menu_display; /* TRUE if menu is currently displayed */

const static struct menu_item menu[] = /* the menu */
{ { "Mode", 0, 4, display_mode },,
{ "Scale", 0, 5, display_scale },
{ "Sweep", 0, 5, display_sweep },
{ "Trigger", 0, 7, no_display },
{ "Level", 2, 7, display_trg_level },
{ "Slope", 2, 7, display_trg_slope },
{ "Delay", 2, 7, display_trg_delay },
};

static int menu_entry; /* currently selected menu entry */

/*
 * init_menu
 *
 * Description: This function initializes the menu routines. It sets
 *              the current menu entry to the first entry, indicates the
 *              display is off, and initializes the options (and
 *              hardware) to normal trigger mode, scale displayed, the
 *              fastest sweep rate, a middle trigger level, positive
 *              trigger slope, and minimum delay. Finally, it displays
 *              the menu.
 *
 * Arguments:    None.

```

```

Return Value:    None.

Input:          None.
Output:         The menu is displayed.

Error Handling: None.

Algorithms:    None.
Data Structures: None.

Global Variables: menu_display - reset to FALSE.
                  menu_entry - reset to first entry (0).

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

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

    /* set the menu parameters */
menu_entry = 0;    /* first menu entry */
menu_display = FALSE; /* menu is not currently displayed (but it will be
shortly) */

    /* set the scope (option) parameters */
set_trigger_mode(NORMAL_TRIGGER); /* normal triggering */
set_scale(SCALE_AXES); /* scale is axes */
set_sweep(0); /* first sweep rate */
set_trg_level((MIN_TRG_LEVEL_SET + MAX_TRG_LEVEL_SET) / 2); /* middle
trigger level */
set_trg_slope(SLOPE_POSITIVE); /* positive slope */
set_trg_delay(MIN_DELAY); /* minimum delay */

    /* now display the menu */
display_menu();

```

```

/* done initializing, return */
return;

}

/*
  clear_menu

Description: This function removes the menu from the display. The
             trace under the menu is restored. The flag menu_display,
             is cleared, indicating the menu is no longer being
             displayed. Note: if the menu is not currently being
             displayed this function does nothing.

Arguments:    None.
Return Value: None.

Input:        None.
Output:       The menu if displayed, is removed and the trace under it
             is rewritten.

Error Handling: None.

Algorithms:   None.
Data Structures: None.

Global Variables: menu_display - checked and set to FALSE.

Author:        Glen George
Last Modified: Mar. 13, 1994

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

  /* check if the menu is currently being displayed */
  if (menu_display) {

```

```

    /* menu is being displayed - turn it off and restore the trace in that
       area */
restore_menu_trace();
}

/* no longer displaying the menu */
menu_display = FALSE;

/* all done, return */
return;
}

```

*/**

display_menu

Description: This function displays the menu. The trace under the menu is overwritten (but it was saved). The flag *menu_display*, is also set, indicating the menu is currently being displayed. Note: if the menu is already being displayed this function does not redisplay it.

Arguments: None.

Return Value: None.

Input: None.

Output: The menu is displayed.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Global Variables: *menu_display* - set to TRUE.

menu_entry - used to highlight currently selected entry.

Author: Glen George

Last Modified: Mar. 13, 1994

```

*/
void display_menu(void)
{
    /* variables */
    int i;    /* loop index */

    /* check if the menu is currently being displayed */
    if (!menu_display) {

        /* menu is not being displayed - turn it on */
        /* display it entry by entry */
        for (i = 0; i < NO_MENU_ENTRIES; i++) {

            /* display this entry - check if it should be highlighted */
            if (i == menu_entry)
                /* currently selected entry - highlight it */
                display_entry(i, TRUE);
            else
                /* not the currently selected entry - "normal video" */
                display_entry(i, FALSE);
        }
    }

    /* now are displaying the menu */
    menu_display = TRUE;

    /* all done, return */
    return;
}

/*
refresh_menu

Description: This function displays the menu if it is currently being
displayed. The trace under the menu is overwritten (but
it was already saved).

```

```

Arguments:      None.
Return Value:   None.

Input:          None.
Output:         The menu is displayed.

Error Handling: None.

Algorithms:     None.
Data Structures: None.

Global Variables: menu_display - determines if menu should be displayed.

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

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

    /* check if the menu is currently being displayed */
    if (menu_display) {

        /* menu is currently being displayed - need to refresh it */
        /* do this by turning off the display, then forcing it back on */
        menu_display = FALSE;
        display_menu();
    }

    /* refreshed the menu if it was displayed, now return */
    return;
}

/*

```

```

reset_menu

Description: This function resets the current menu selection to the
             first menu entry. If the menu is currently being
             displayed the display is updated.

Arguments:    None.
Return Value: None.

Input:        None.
Output:       The menu display is updated if it is being displayed.

Error Handling: None.

Algorithms:   None.
Data Structures: None.

Global Variables: menu_display - checked to see if menu is displayed.
                  menu_entry - reset to 0 (first entry).

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

*/

```

```

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

    /* check if the menu is currently being displayed */
    if (menu_display) {

        /* menu is being displayed */
        /* remove highlight from currently selected entry */
        display_entry(menu_entry, FALSE);
    }

    /* reset the currently selected entry */
    menu_entry = 0;
}

```

```

/* finally, highlight the first entry if the menu is being displayed */
if (menu_display)
display_entry(menu_entry, TRUE);

/* all done, return */
return;

}

/*
next_entry

Description: This function changes the current menu selection to the
             next menu entry. If the current selection is the last
             entry in the menu, it is not changed. If the menu is
             currently being displayed, the display is updated.

Arguments:    None.
Return Value: None.

Input:        None.
Output:       The menu display is updated if it is being displayed and
             the entry selected changes.

Error Handling: None.

Algorithms:   None.
Data Structures: None.

Global Variables: menu_display - checked to see if menu is displayed.
                  menu_entry - updated to a new entry (if not at end).

Author:        Glen George
Last Modified: Mar. 13, 1994

*/
void next_entry(void)
{
    /* variables */

```

```

/* none */

/* only update if not at end of the menu */
if (menu_entry < (NO_MENU_ENTRIES - 1)) {

    /* not at the end of the menu */

    /* turn off current entry if displaying */
    if (menu_display)
        /* displaying menu - turn off currently selected entry */
        display_entry(menu_entry, FALSE);

    /* update the menu entry to the next one */
    menu_entry++;

    /* now highlight this entry if displaying the menu */
    if (menu_display)
        /* displaying menu - highlight newly selected entry */
        display_entry(menu_entry, TRUE);
}

/* all done, return */
return;
}

```

*/**

previous_entry

Description: This function changes the current menu selection to the previous menu entry. If the current selection is the first entry in the menu, it is not changed. If the menu is currently being displayed, the display is updated.

Arguments: None.

Return Value: None.

Input: None.

Output: The menu display is updated if it is being displayed and

the currently selected entry changes.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Global Variables: menu_display - checked to see if menu is displayed.
menu_entry - updated to a new entry (if not at start).

Author: Glen George
Last Modified: Mar. 13, 1994

*/

```
void previous_entry(void)
{
    /* variables */
    /* none */

    /* only update if not at the start of the menu */
    if (menu_entry > 0) {

        /* not at the start of the menu */

        /* turn off current entry if displaying */
        if (menu_display)
            /* displaying menu - turn off currently selected entry */
            display_entry(menu_entry, FALSE);

        /* update the menu entry to the previous one */
        menu_entry--;

        /* now highlight this entry if displaying the menu */
        if (menu_display)
            /* displaying menu - highlight newly selected entry */
            display_entry(menu_entry, TRUE);
    }

    /* all done, return */
    return;
}
```

```
}
```

```
/*
```

```
menu_entry_left
```

```
Description: This function handles the <Left> key for the current menu  
selection. It does this by doing a table lookup on the  
current menu selection.
```

```
Arguments: None.
```

```
Return Value: None.
```

```
Input: None.
```

```
Output: The menu display is updated if it is being displayed and  
the <Left> key causes a change to the display.
```

```
Error Handling: None.
```

```
Algorithms: Table lookup is used to determine what to do for the  
input key.
```

```
Data Structures: An array holds the table of key processing routines.
```

```
Global Variables: menu_entry - used to select the processing function.
```

```
Author: Glen George
```

```
Last Modified: May 9, 2006
```

```
*/
```

```
void menu_entry_left(void)
```

```
{
```

```
/* variables */
```

```
/* key processing functions */
```

```
static void (* const process[])(void) =
```

```
/* Mode Scale Sweep Trigger */
```

```
{ mode_down, scale_down, sweep_down, trace_rearm,  
trg_level_down, trg_slope_toggle, trg_delay_down };
```

```
/* Level Slope Delay */
```

```

/* invoke the appropriate <Left> key function */
process[menu_entry]();

/* if displaying menu entries, display the new value */
/* note: since it is being changed - know this option is selected */
if (menu_display) {
    menu[menu_entry].display((MENU_X + menu[menu_entry].opt_off),
                           (MENU_Y + menu_entry), OPTION_SELECTED);
}

/* all done, return */
return;
}

```

*/**

menu_entry_right

Description: This function handles the <Right> key for the current menu selection. It does this by doing a table lookup on the current menu selection.

Arguments: None.

Return Value: None.

Input: None.

Output: The menu display is updated if it is being displayed and the <Right> key causes a change to the display.

Error Handling: None.

Algorithms: Table lookup is used to determine what to do for the input key.

Data Structures: An array holds the table of key processing routines.

Global Variables: menu - used to display the new menu value.
menu_entry - used to select the processing function.

Author: Glen George

Last Modified: May 9, 2006

```

*/
void menu_entry_right(void)
{
    /* variables */

    /* key processing functions */
    static void (* const process[])(void) =
    /* Mode      Scale      Sweep      Trigger      */
    { mode_up , scale_up , sweep_up , trace_rearm,
      trg_level_up, trg_slope_toggle, trg_delay_up
    }; /* Level      Slope      Delay      */

    /* invoke the appropriate <Right> key function */
    process[menu_entry]();

    /* if displaying menu entries, display the new value */
    /* note: since it is being changed - know this option is selected */
    if (menu_display) {
        menu[menu_entry].display((MENU_X + menu[menu_entry].opt_off),
                               (MENU_Y + menu_entry), OPTION_SELECTED);
    }

    /* all done, return */
    return;
}

/*
display_entry

Description: This function displays the passed menu entry and its
            current option setting. If the second argument is TRUE
            it displays them with color SELECTED and OPTION_SELECTED
            respectively. If the second argument is FALSE it
            displays the menu entry with color NORMAL and the option
            setting with color OPTION_NORMAL.

```

```

Arguments:      entry (int) - menu entry to be displayed.
               selected (int) - whether or not the menu entry is
                               currently selected (determines the color
                               with which the entry is output).
Return Value:   None.

Input:          None.
Output:         The menu entry is output to the LCD.

Error Handling: None.

Algorithms:    None.
Data Structures: None.

Global Variables: menu - used to display the menu entry.

Author:         Glen George
Last Modified: Aug. 13, 2004

*/
static void display_entry(int entry, int selected)
{
    /* variables */
    /* none */

    /* output the menu entry with the appropriate color */
    plot_string((MENU_X + menu[entry].h_off), (MENU_Y + entry), menu[entry].s,
                (selected ? SELECTED : NORMAL));
    /* also output the menu option with the appropriate color */
    menu[entry].display((MENU_X + menu[entry].opt_off), (MENU_Y + entry),
                        (selected ? OPTION_SELECTED : OPTION_NORMAL));

    /* all done outputting this menu entry - return */
    return;
}

```

Listing 12: ..\osc134\menu.h

```
*****
*/

```

```

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



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

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

```

```

#ifndef __MENU_H__
#define __MENU_H__


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

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

/* constants */

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

```

```

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

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

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

/* structures, unions, and typedefs */

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

/* function declarations */

/* menu initialization function */
void init_menu(void);

/* menu display functions */
void clear_menu(void); /* clear the menu display */
void display_menu(void); /* display the menu */
void refresh_menu(void); /* refresh the menu */

/* menu update functions */
void reset_menu(void); /* reset the menu to first entry */
void next_entry(void); /* go to the next menu entry */
void previous_entry(void); /* go to the previous menu entry */

```

```

/* menu entry functions */
void menu_entry_left(void); /* do the <Left> key for the menu entry */
void menu_entry_right(void); /* do the <Right> key for the menu entry */

#endif

```

Listing 13: ..//osc134/menuact.c

```

*****
*/
/*
 * MENUACT
 * Menu Action Functions
 * Digital Oscilloscope Project
 * EE/CS 52
 */
*****


/*
This file contains the functions for carrying out menu actions for the
Digital Oscilloscope project. These functions are invoked when the <Left>
or <Right> key is pressed for a menu item. Also included are the functions
for displaying the current menu option selection. The functions included
are:
    display_mode      - display trigger mode
    display_scale     - display the scale type
    display_sweep     - display the sweep rate
    display_trg_delay - display the trigger delay
    display_trg_level - display the trigger level
    display_trg_slope - display the trigger slope
    get_trigger_mode - get the current trigger mode
    mode_down         - go to the "next" trigger mode
    mode_up          - go to the "previous" trigger mode
    no_display        - nothing to display for option setting
    no_menu_action   - no action to perform for <Left> or <Right> key
    scale_down        - go to the "next" scale type
    scale_up         - go to the "previous" scale type
    set_scale         - set the scale type
    set_sweep         - set the sweep rate
    set_trg_delay    - set the trigger delay
    set_trg_level    - set the trigger level
    set_trg_slope    - set the trigger slope
    set_trigger_mode - set the trigger mode
    sweep_down       - decrease the sweep rate

```

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

The local functions included are:

```
adjust_trg_delay - adjust the trigger delay for a new sweep rate
cvt_num_field   - converts a numeric field value to a string
```

The locally global variable definitions included are:

```
delay      - current trigger delay
level      - current trigger level
scale      - current display scale type
slope      - current trigger slope
sweep      - current sweep rate
sweep_rates - table of information on possible sweep rates
trigger_mode - current triggering mode
```

Revision History

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

```
        of possibilities instead of just toggling
        between one of two possibilities (since there
        are more than two now).
```

```
5/9/06 Glen George    Added accessor function (get_trigger_mode)
                      to be able to get the current trigger mode.
```

```
*/
```

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

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

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

```
/* locally global variables
```

```
/* trace parameters */
static enum trigger_type trigger_mode; /* current triggering mode */
static enum scale_type scale; /* current scale type */
static int sweep; /* sweep rate index */
static int level; /* current trigger level */
static enum slope_type slope; /* current trigger slope */
static long int delay; /* current trigger delay */
```

```
/* sweep rate information */
static const struct sweep_info sweep_rates[] =
{ { 10000000L, " 100 ns" },
  { 5000000L, " 200 ns" },
  { 2000000L, " 500 ns" },
  { 1000000L, " 1 \004s " },
```

```

{ 500000L, " 2 \004s " },
{ 200000L, " 5 \004s " },
{ 100000L, " 10 \004s " },
{ 50000L, " 20 \004s " },
{ 20000L, " 50 \004s " },
{ 10000L, " 100 \004s" },
{ 5000L, " 200 \004s" },
{ 2000L, " 500 \004s" },
{ 1000L, " 1 ms "   },
{ 500L, " 2 ms "   },
{ 200L, " 5 ms "   },
{ 100L, " 10 ms "  },
{ 50L, " 20 ms "  } };

/*
no_menu_action

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

Arguments: None.
Return Value: None.

Input: None.
Output: None.

Error Handling: None.

Algorithms: None.
Data Structures: None.

Global Variables: None.

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

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

```

```

/* nothing to do - return */
return;

}

/*
no_display

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

Arguments:      x_pos (int) - x position (in character cells) at which to
                display the menu option (not used).
                y_pos (int) - y position (in character cells) at which to
                display the menu option (not used).
                style (int) - style with which to display the menu option
                (not used).
Return Value:   None.

Input:          None.
Output:         None.

Error Handling: None.

Algorithms:    None.
Data Structures: None.

Global Variables: None.

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

*/

```

void no_display(int x_pos, int y_pos, int style)

{

 /* variables */
 /* none */

```

/* nothing to do - return */
return;

}

/*
set_trigger_mode

Description: This function sets the triggering mode to the passed
             value.

Arguments:   m (enum trigger_type) - mode to which to set the
             triggering mode.

Return Value: None.

Input:        None.
Output:       None.

Error Handling: None.

Algorithms:   None.
Data Structures: None.

Global Variables: trigger_mode - initialized to the passed value.

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

*/
void set_trigger_mode(enum trigger_type m)
{
    /* variables */
    /* none */

    /* set the trigger mode */
    trigger_mode = m;
}

```

```

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

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

}

/*
get_trigger_mode

Description: This function returns the current triggering mode.

Arguments: None.
Return Value: (enum trigger_type) - current triggering mode.

Input: None.
Output: None.

Error Handling: None.

Algorithms: None.
Data Structures: None.

Global Variables: trigger_mode - value is returned (not changed).

Author: Glen George
Last Modified: May 9, 2006

*/
enum trigger_type get_trigger_mode()
{
    /* variables */
    /* none */

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

```

```
}
```

```
/*
```

```
mode_down
```

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

```
Arguments: None.
```

```
Return Value: None.
```

```
Input: None.
```

```
Output: None.
```

```
Error Handling: None.
```

```
Algorithms: None.
```

```
Data Structures: None.
```

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

```
Author: Glen George
```

```
Last Modified: May 9, 2006
```

```
*/
```

```
void mode_down()
```

```
{
```

```
/* variables */
```

```
/* none */
```

```
/* move to the "next" triggering mode */
```

```
if (trigger_mode == NORMAL_TRIGGER)
```

```
    trigger_mode = AUTO_TRIGGER;
```

```
else if (trigger_mode == AUTO_TRIGGER)
```

```
    trigger_mode = ONESHOT_TRIGGER;
```

```
else
```

```
    trigger_mode = NORMAL_TRIGGER;
```

```

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

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

}

/*
mode_up

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

Arguments: None.
Return Value: None.

Input: None.
Output: None.

Error Handling: None.

Algorithms: None.
Data Structures: None.

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

Author: Glen George
Last Modified: May 9, 2006

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

```

```

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

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

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

/*
display_mode

Description: This function displays the current triggering mode at the
            passed position, in the passed style.

Arguments:      x_pos (int) - x position (in character cells) at which to
                  display the trigger mode.
                  y_pos (int) - y position (in character cells) at which to
                  display the trigger mode.
                  style (int) - style with which to display the trigger
                  mode.
Return Value:   None.

Input:          None.
Output:         The trigger mode is displayed at the passed position on
                  the screen.

Error Handling: None.

Algorithms:     None.
Data Structures: None.

Global Variables: trigger_mode - determines which string is displayed.

```

Author: Glen George
 Last Modified: May 9, 2006

```

/*
void display_mode(int x_pos, int y_pos, int style)
{
  /* variables */

  /* the mode strings (must match enumerated type) */
  const static char * const modes[] = { " Normal ",
                                         " Automatic",
                                         " One-Shot " };

  /* display the trigger mode */
  plot_string(x_pos, y_pos, modes[trigger_mode], style);

  /* all done displaying the trigger mode - return */
  return;
}

/*
set_scale

Description: This function sets the scale type to the passed value.

Arguments: s (enum scale_type) - scale type to which to initialize
           the scale status.

Return Value: None.

Input: None.

Output: The new trace display is updated with the new scale.

Error Handling: None.

Algorithms: None.
Data Structures: None.

```

```

Global Variables: scale - initialized to the passed value.

Author:          Glen George
Last Modified:  Mar. 13, 1994

*/
void set_scale(enum scale_type s)
{
    /* variables */
    /* none */

    /* set the scale type */
    scale = s;

    /* output the scale appropriately */
    set_display_scale(scale);

    /* all done setting the scale type - return */
    return;
}

/*
scale_down

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

Arguments:    None.
Return Value: None.

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

Error Handling: None.

Algorithms:   None.

```

Description: None.

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

Author: Glen George
Last Modified: May 9, 2006

*/

```
void scale_down()
{
    /* variables */
    /* none */

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

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

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

```
/*
scale_up
```

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

Arguments: None.
Return Value: None.

```

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

Error Handling: None.

Algorithms:    None.
Data Structures: None.

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

Author:         Glen George
Last Modified: May 9, 2006

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

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

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

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

/*
display_scale

```

Description: This function displays the current scale type at the passed position, in the passed style.

Arguments: x_pos (int) - x position (in character cells) at which to display the scale type.
y_pos (int) - y position (in character cells) at which to display the scale type.
style (int) - style with which to display the scale type.

Return Value: None.

Input: None.

Output: The scale type is displayed at the passed position on the display.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Global Variables: scale - determines which string is displayed.

Author: Glen George
Last Modified: Mar. 13, 1994

*/

```
void display_scale(int x_pos, int y_pos, int style)
{
    /* variables */

    /* the scale type strings (must match enumerated type) */
    const static char * const scale_stat[] = { "None",
                                                "Axes",
                                                "Grid" };

    /* display the scale status */
    plot_string(x_pos, y_pos, scale_stat[scale], style);

    /* all done displaying the scale status - return */
    return;
}
```

```

}

/*
set_sweep

Description: This function sets the sweep rate to the passed value.
             The passed value gives the sweep rate to choose from the
             list of sweep rates (it gives the list index).

Arguments:    s (int) - index into the list of sweep rates to which to
              set the current sweep rate.
Return Value: None.

Input:        None.
Output:       None.

Error Handling: The passed index is not checked for validity.

Algorithms:   None.
Data Structures: None.

Global Variables: sweep - initialized to the passed value.

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

*/
void set_sweep(int s)
{
    /* variables */
    int sample_size;      /* sample size for this sweep rate */

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

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

```

```

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

}

/*
sweep_down

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

Arguments: None.
Return Value: None.

Input: None.
Output: None.

Error Handling: None.

Algorithms: None.
Data Structures: None.

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

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

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

*/
void sweep_down()
{

```

```

/* variables */
int sample_size;      /* sample size for the new sweep rate */

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

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

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

```

/*

sweep_up

Description: This function handles increasing the current sweep rate.

The new sweep rate (and sample size) is sent to the hardware (and trace routines). If an attempt is made to raise the sweep rate above the maximum value it is not changed. This routine also updates the sweep delay based on the new sweep rate (to keep the delay time constant).

Arguments: None.

Return Value: None.

Input: None.

Output: None.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Global Variables: sweep - incremented if not already the maximum value.
delay - decreased to keep delay time constant.

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

Author: Glen George

Last Modified: Mar. 8, 1994

*/

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

    /* increase the sweep rate, if not already the maximum */
    if (sweep < (NO_SWEEP_RATES - 1)) {
        /* not at maximum, adjust delay for new sweep */
        adjust_trg_delay(sweep, (sweep + 1));
        /* now set new sweep rate */
        sweep++;
    }

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

    /* all done with raising the sweep rate - return */
    return;
}

/*
```

```

display_sweep

Description: This function displays the current sweep rate at the
passed position, in the passed style.

Arguments: x_pos (int) - x position (in character cells) at which to
display the sweep rate.
y_pos (int) - y position (in character cells) at which to
display the sweep rate.
style (int) - style with which to display the sweep rate.

Return Value: None.

Input: None.
Output: The sweep rate is displayed at the passed position on the
display.

Error Handling: None.

Algorithms: None.
Data Structures: None.

Global Variables: sweep - determines which string is displayed.

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

*/
void display_sweep(int x_pos, int y_pos, int style)
{
    /* variables */
    /* none */

    /* display the sweep rate */
    plot_string(x_pos, y_pos, sweep_rates[sweep].s, style);

    /* all done displaying the sweep rate - return */
    return;
}

```

```

/*
  set_trg_level

  Description: This function sets the trigger level to the passed value.

  Arguments: l (int) - value to which to set the trigger level.
  Return Value: None.

  Input: None.
  Output: None.

  Error Handling: The passed value is not checked for validity.

  Algorithms: None.
  Data Structures: None.

  Global Variables: level - initialized to the passed value.

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

*/

```

```

void set_trg_level(int l)
{
    /* variables */
    /* none */

    /* set the trigger level */
    level = l;

    /* set the trigger level in hardware too */
    set_trigger(level, slope);

    /* all done initializing the trigger level - return */
    return;
}

```

```

/*
trg_level_down

Description: This function handles decreasing the current trigger
            level. The new trigger level is sent to the hardware.
            If an attempt is made to lower the trigger level below
            the minimum value it is not changed.

Arguments:    None.
Return Value: None.

Input:        None.
Output:       None.

Error Handling: None.

Algorithms:   None.
Data Structures: None.

Global Variables: level - decremented if not already at the minimum value.

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

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

    /* decrease the trigger level, if not already the minimum */
    if (level > MIN_TRG_LEVEL_SET)
        level--;

    /* set the trigger level for the hardware */
    set_trigger(level, slope);

    /* all done with lowering the trigger level - return */
    return;
}

```

```
}
```

```
/*
```

```
trg_level_up
```

Description: This function handles increasing the current trigger level. The new trigger level is sent to the hardware. If an attempt is made to raise the trigger level above the maximum value it is not changed.

Arguments: None.

Return Value: None.

Input: None.

Output: None.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Global Variables: level - incremented if not already the maximum value.

Author: Glen George

Last Modified: Mar. 8, 1994

```
*/
```

```
void trg_level_up()
```

```
{
```

```
/* variables */
```

```
/* none */
```

```
/* increase the trigger level, if not already the maximum */
```

```
if (level < MAX_TRG_LEVEL_SET)
```

```
    level++;
```

```
/* tell the hardware the new trigger level */
```

```
set_trigger(level, slope);
```

```

/* all done raising the trigger level - return */
return;

}

/*
display_trg_level

Description: This function displays the current trigger level at the
passed position, in the passed style.

Arguments:    x_pos (int) - x position (in character cells) at which to
              display the trigger level.
              y_pos (int) - y position (in character cells) at which to
              display the trigger level.
              style (int) - style with which to display the trigger
              level.

Return Value: None.

Input:        None.

Output:       The trigger level is displayed at the passed position on
             the display.

Error Handling: None.

Algorithms:   None.

Data Structures: None.

Global Variables: level - determines the value displayed.

Author:        Glen George
Last Modified: Mar. 10, 1995

*/

```

```

void display_trg_level(int x_pos, int y_pos, int style)
{
    /* variables */
    char    level_str[] = "      "; /* string containing the trigger level */
    long int l;           /* trigger level in mV */

```

```

/* compute the trigger level in millivolts */
l = ((long int) MAX_LEVEL - MIN_LEVEL) * level / (MAX_TRG_LEVEL_SET -
    MIN_TRG_LEVEL_SET) + MIN_LEVEL;

/* convert the level to the string (leave first character blank) */
cvt_num_field(l, &level_str[1]);

/* add in the units */
level_str[7] = 'V';

/* now finally display the trigger level */
plot_string(x_pos, y_pos, level_str, style);

/* all done displaying the trigger level - return */
return;
}

```

*/**

set_trg_slope

Description: This function sets the trigger slope to the passed value.

Arguments: s (enum slope_type) – trigger slope type to which to set the locally global slope.

Return Value: None.

Input: None.

Output: None.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Global Variables: slope – set to the passed value.

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

*/

```
void set_trg_slope(enum slope_type s)
{
    /* variables */
    /* none */

    /* set the slope type */
    slope = s;

    /* also tell the hardware what the slope is */
    set_trigger(level, slope);

    /* all done setting the trigger slope - return */
    return;
}
```

/*

trg_slope_toggle

Description: This function handles toggling (and setting) the current trigger slope.

Arguments: None.

Return Value: None.

Input: None.

Output: None.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Global Variables: slope - toggled.

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

*/

```
void trg_slope_toggle()
{
    /* variables */
    /* none */

    /* toggle the trigger slope */
    if (slope == SLOPE_POSITIVE)
        slope = SLOPE_NEGATIVE;
    else
        slope = SLOPE_POSITIVE;

    /* set the new trigger slope */
    set_trigger(level, slope);

    /* all done with the trigger slope - return */
    return;
}
```

/*

display_trg_slope

Description: This function displays the current trigger slope at the passed position, in the passed style.

Arguments: x_pos (int) - x position (in character cells) at which to display the trigger slope.

y_pos (int) - y position (in character cells) at which to display the trigger slope.

style (int) - style with which to display the trigger slope.

Return Value: None.

```

Input:      None.
Output:     The trigger slope is displayed at the passed position on
           the screen.

Error Handling: None.

Algorithms: None.
Data Structures: None.

Global Variables: slope - determines which string is displayed.

Author:      Glen George
Last Modified: Mar. 13, 1994

*/
void display_trg_slope(int x_pos, int y_pos, int style)
{
    /* variables */

    /* the trigger slope strings (must match enumerated type) */
    const static char * const slopes[] = { "+", "-" };

    /* display the trigger slope */
    plot_string(x_pos, y_pos, slopes[slope], style);

    /* all done displaying the trigger slope - return */
    return;
}

/*
set_trg_delay

Description: This function sets the trigger delay to the passed value.

Arguments: d (long int) - value to which to set the trigger delay.
Return Value: None.

```

Input: None.
Output: None.

Error Handling: The passed value is not checked for validity.

Algorithms: None.
Data Structures: None.

Global Variables: delay - initialized to the passed value.

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

*/

```
void set_trg_delay(long int d)
{
    /* variables */
    /* none */

    /* set the trigger delay */
    delay = d;

    /* set the trigger delay in hardware too */
    set_delay(delay);

    /* all done initializing the trigger delay - return */
    return;
}
```

/*
trg_delay_down

Description: This function handles decreasing the current trigger
delay. The new trigger delay is sent to the hardware.
If an attempt is made to lower the trigger delay below
the minimum value it is not changed.

```

Arguments:      None.
Return Value:   None.

Input:          None.
Output:         None.

Error Handling: None.

Algorithms:    None.
Data Structures: None.

Global Variables: delay - decremented if not already at the minimum value.

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

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

    /* decrease the trigger delay, if not already the minimum */
    if (delay > MIN_DELAY)
        delay--;

    /* set the trigger delay for the hardware */
    set_delay(delay);

    /* all done with lowering the trigger delay - return */
    return;
}

/*
trg_delay_up

Description:   This function handles increasing the current trigger

```

delay. The new trigger delay is sent to the hardware.
If an attempt is made to raise the trigger delay above
the maximum value it is not changed.

Arguments: None.
Return Value: None.

Input: None.
Output: None.

Error Handling: None.

Algorithms: None.
Data Structures: None.

Global Variables: delay - incremented if not already the maximum value.

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

*/

```
void trg_delay_up()
{
    /* variables */
    /* none */

    /* increase the trigger delay, if not already the maximum */
    if (delay < MAX_DELAY)
        delay++;

    /* tell the hardware the new trigger delay */
    set_delay(delay);

    /* all done raising the trigger delay - return */
    return;
}
```

```

/*
adjust_trg_delay

Description: This function adjusts the trigger delay for a new sweep
rate. The factor to adjust the delay by is determined
by looking up the sample rates in the sweep_rates array.
If the delay goes out of range, due to the adjustment it
is reset to the maximum or minimum valid value.

Arguments:    old_sweep (int) - old sweep rate (index into sweep_rates
            array).
            new_sweep (int) - new sweep rate (index into sweep_rates
            array).
Return Value: None.

Input:        None.
Output:       None.

Error Handling: None.

Algorithms:   The delay is multiplied by 10 times the ratio of the
              sweep sample rates then divided by 10. This is done to
              avoid floating point arithmetic and integer truncation
              problems.

Data Structures: None.

Global Variables: delay - adjusted based on passed sweep rates.

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

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

*/
void adjust_trg_delay(int old_sweep, int new_sweep)
{
    /* variables */
    /* none */

    /* multiply by 10 times the ratio of sweep rates */

```

```

    delay *= (10 * sweep_rates[new_sweep].sample_rate) /
        sweep_rates[old_sweep].sample_rate;
    /* now divide the factor of 10 back out */
    delay /= 10;

    /* make sure delay is not out of range */
    if (delay > MAX_DELAY)
        /* delay is too large - set to maximum */
        delay = MAX_DELAY;
    if (delay < MIN_DELAY)
        /* delay is too small - set to minimum */
    delay = MIN_DELAY;

    /* tell the hardware the new trigger delay */
    set_delay(delay);

    /* all done adjusting the trigger delay - return */
    return;
}

```

*/**

display_trg_delay

Description: This function displays the current trigger delay at the passed position, in the passed style.

Arguments: x_pos (int) - x position (in character cells) at which to display the trigger delay.
y_pos (int) - y position (in character cells) at which to display the trigger delay.
style (int) - style with which to display the trigger delay.

Return Value: None.

Input: None.

Output: The trigger delay is displayed at the passed position on the display.

Error Handling: None.

Algorithms: None.
Data Structures: None.

Global Variables: delay - determines the value displayed.

Author: Glen George
Last Modified: May 3, 2006

*/

```
void display_trg_delay(int x_pos, int y_pos, int style)
{
    /* variables */
    char    delay_str[] = "      "; /* string containing the trigger delay */
    long int units_adj;        /* adjustment to get to microseconds */

    long int d;                /* delay in appropriate units */

    /* compute the delay in the appropriate units */
    /* have to watch out for overflow, so be careful */
    if (sweep_rates[sweep].sample_rate > 1000000L) {
        /* have a fast sweep rate, could overflow */
        /* first compute in units of 100 ns */
        d = delay * (10000000L / sweep_rates[sweep].sample_rate);
        /* now convert to nanoseconds */
        d *= 100L;
        /* need to divide by 1000 to get to microseconds */
        units_adj = 1000;
    }
    else {
        /* slow sweep rate, don't have to worry about overflow */
        d = delay * (1000000L / sweep_rates[sweep].sample_rate);
        /* already in microseconds, so adjustment is 1 */
        units_adj = 1;
    }

    /* convert it to the string (leave first character blank) */
    cvt_num_field(d, &delay_str[1]);

    /* add in the units */
    if (((d / units_adj) < 1000) && ((d / units_adj) > -1000) && (units_adj == 1000)) {
        /* delay is in microseconds */
```

```

delay_str[7] = '\004';
delay_str[8] = 's';
}
else if (((d / units_adj) < 1000000) && ((d / units_adj) > -1000000)) {
    /* delay is in milliseconds */
delay_str[7] = 'm';
delay_str[8] = 's';
}
else if (((d / units_adj) < 1000000000) && ((d / units_adj) > -1000000000)) {
    /* delay is in seconds */
delay_str[7] = 's';
delay_str[8] = ' ';
}
else {
    /* delay is in kiloseconds */
delay_str[7] = 'k';
delay_str[8] = 's';
}

/* now actually display the trigger delay */
plot_string(x_pos, y_pos, delay_str, style);

/* all done displaying the trigger delay - return */
return;
}

```

/*

cvt_num_field

Description: This function converts the passed number (numeric field value) to a string and returns that in the passed string reference. The number may be signed, and a sign (+ or -) is always generated. The number is assumed to have three digits to the right of the decimal point. Only the four most significant digits of the number are displayed and the decimal point is shifted appropriately. (Four digits are always generated by the function).

Arguments: n (long int) - numeric field value to convert.

```

    s (char *) - pointer to string in which to return the
                  converted field value.

Return Value: None.

Input:        None.
Output:       None.

Error Handling: None.

Algorithms:   The algorithm used assumes four (4) digits are being
              converted.

Data Structures: None.

Global Variables: None.

Known Bugs:    If the passed long int is the largest negative long int,
               the function will display garbage.

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

*/

```

```

void cvt_num_field(long int n, char *s)
{
    /* variables */
    int dp = 3;      /* digits to right of decimal point */
    int d;          /* digit weight (power of 10) */

    int i = 0;        /* string index */

    /* first get the sign (and make n positive for conversion) */
    if (n < 0) {
        /* n is negative, set sign and convert to positive */
        s[i++] = '-';
        n = -n;
    }
    else {
        /* n is positive, set sign only */
        s[i++] = '+';
    }
}

```

```

/* make sure there are no more than 4 significant digits */
while (n > 9999) {
    /* have more than 4 digits - get rid of one */
n /= 10;
/* adjust the decimal point */
dp--;
}

/* if decimal point is non-positive, make positive */
/* (assume will take care of adjustment with output units in this case) */
while (dp <= 0)
    dp += 3;

/* adjust dp to be digits to the right of the decimal point */
/* (assuming 4 digits) */
dp = 4 - dp;

/* finally, loop getting and converting digits */
for (d = 1000; d > 0; d /= 10) {

    /* check if need decimal the decimal point now */
if (dp-- == 0)
    /* time for decimal point */
    s[i++] = '.';

/* get and convert this digit */
s[i++] = (n / d) + '0';
/* remove this digit from n */
n %= d;
}

/* all done converting the number, return */
return;
}

```

Listing 14: ..//osc134/menuact.h

```

*****
/*
 *          MENUACT.H
 *      Menu Action Functions
 */

```

```

/*
           Include File
/*
           Digital Oscilloscope Project
           EE/CS 52
/*
*****



/*
This file contains the constants and function prototypes for the functions
which carry out menu actions and display and initialize menu settings for
the Digital Oscilloscope project (the functions are defined in menuact.c).

Revision History:
 3/8/94  Glen George      Initial revision.
 3/13/94 Glen George      Updated comments.
 3/13/94 Glen George      Changed definition of enum scale_type (was
                           enum scale_status).
 3/10/95 Glen George      Changed MAX_TRG_LEVEL_SET (maximum trigger
                           level) to 127 to match specification.
 3/17/97 Glen George      Updated comments.
 5/3/06  Glen George      Updated comments.
 5/9/06  Glen George      Added a new mode (AUTO_TRIGGER) and a new
                           scale (SCALE_GRID).
 5/9/06  Glen George      Added menu functions for mode and scale to
                           move up and down a list instead of just
                           toggling the selection.
 5/9/06  Glen George      Added declaration for the accessor to the
                           current trigger mode (get_trigger_mode).
*/

```

```

#ifndef __MENUACT_H__
#define __MENUACT_H__


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

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

```

```

/* constants */

/* min and max trigger level settings */
#define MIN_TRG_LEVEL_SET 0
#define MAX_TRG_LEVEL_SET 127

/* number of different sweep rates */
#define NO_SWEEP_RATES  (sizeof(sweep_rates) / sizeof(struct sweep_info))

/* structures, unions, and typedefs */

/* types of triggering modes */
enum trigger_type { NORMAL_TRIGGER, /* normal triggering */
                    AUTO_TRIGGER, /* automatic triggering */
                    ONESHOT_TRIGGER /* one-shot triggering */
};

/* types of displayed scales */
enum scale_type { SCALE_NONE, /* no scale is displayed */
                  SCALE_AXES, /* scale is a set of axes */
                  SCALE_GRID /* scale is a grid */
};

/* types of trigger slopes */
enum slope_type { SLOPE_POSITIVE, /* positive trigger slope */
                  SLOPE_NEGATIVE /* negative trigger slope */
};

/* sweep rate information */
struct sweep_info { long int sample_rate; /* sample rate */
                     const char *s; /* sweep rate string */
};

/* function declarations */

/* menu option actions */
void no_menu_action(void); /* no action to perform */
void mode_down(void); /* change to the "next" trigger mode */

```

```

void mode_up(void);           /* change to the "previous" trigger mode */
void scale_down(void);        /* change to the "next" scale type */
void scale_up(void);          /* change to the "previous" scale type */
void sweep_down(void);        /* decrease the sweep rate */
void sweep_up(void);          /* increase the sweep rate */
void trg_level_down(void);   /* decrease the trigger level */
void trg_level_up(void);     /* increase the trigger level */
void trg_slope_toggle(void); /* toggle the trigger slope */
void trg_delay_down(void);   /* decrease the trigger delay */
void trg_delay_up(void);     /* increase the trigger delay */

/* option accessor routines */
enum trigger_type get_trigger_mode(void); /* get the current trigger mode */

/* option initialization routines */
void set_trigger_mode(enum trigger_type); /* set the trigger mode */
void set_scale(enum scale_type);          /* set the scale type */
void set_sweep(int);                     /* set the sweep rate */
void set_trg_level(int);                 /* set the trigger level */
void set_trg_slope(enum slope_type);    /* set the trigger slope */
void set_trg_delay(long int);           /* set the trigger delay */

/* option display routines */
void no_display(int, int, int); /* no option setting to display */
void display_mode(int, int, int); /* display trigger mode */
void display_scale(int, int, int); /* display the scale type */
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 trigger delay */

#endif

```

Listing 15: ..//osc134/keyproc.c

```

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

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

The local functions included are:
none

The locally global variable definitions included are:
none

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

*/

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

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

/*

no_action

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

Arguments: cur_state (enum status) - the current system state.

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

```

Input:          None.
Output:         None.

Error Handling: None.

Algorithms:    None.
Data Structures: None.

Global Variables: None.

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

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

    /* return the current state */
    return cur_state;
}

/*
menu_key

Description: This function handles the <Menu> key. If the passed
             state is MENU_ON, the menu is turned off. If the passed
             state is MENU_OFF, the menu is turned on. The returned
             state is the "opposite" of the passed state.

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

Input:          None.
Output:         The menu is either turned on or off.

```

```

Error Handling: None.

Algorithms:      None.
Data Structures: None.

Global Variables: None.

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

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

    /* check if need to turn the menu on or off */
    if (cur_state == MENU_ON)
        /* currently the menu is on, turn it off */
        clear_menu();
    else
        /* currently the menu is off, turn it on */
        display_menu();

    /* all done, return the "opposite" of the current state */
    if (cur_state == MENU_ON)
        /* state was MENU_ON, change it to MENU_OFF */
        return MENU_OFF;
    else
        /* state was MENU_OFF, change it to MENU_ON */
        return MENU_ON;
}

/*
menu_up

```

Description: This function handles the <Up> key when in a menu. It goes to the previous menu entry and leaves the system state unchanged.

Arguments: cur_state (enum status) - the current system state.

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

Input: None.

Output: The menu display is updated.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Global Variables: None.

Author: Glen George

Last Modified: Mar. 8, 1994

*/

```
enum status menu_up(enum status cur_state)
{
    /* variables */
    /* none */

    /* go to the previous menu entry */
    previous_entry();

    /* return the current state */
    return cur_state;
}

/*
menu_down
```

Description: This function handles the <Down> key when in a menu. It goes to the next menu entry and leaves the system state unchanged.

Arguments: cur_state (enum status) - the current system state.

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

Input: None.

Output: The menu display is updated.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Global Variables: None.

Author: Glen George

Last Modified: Mar. 8, 1994

*/

```
enum status menu_down(enum status cur_state)
{
    /* variables */
    /* none */

    /* go to the next menu entry */
    next_entry();

    /* return the current state */
    return cur_state;
}

/*
menu_left
```

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

Arguments: cur_state (enum status) - the current system state.

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

Input: None.

Output: The menu display may be updated.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Global Variables: None.

Author: Glen George

Last Modified: Mar. 8, 1994

*/

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

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

    /* return the current state */
    return cur_state;
}

/*
menu_right
```

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

Arguments: cur_state (enum status) - the current system state.

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

Input: None.

Output: The menu display may be updated.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Global Variables: None.

Author: Glen George

Last Modified: Mar. 8, 1994

*/

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

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

    /* return the current state */
    return cur_state;
}
```

Listing 16: ..//osc134/keyproc.h

```
*****
/*
 * KEYPROC.H
 * Key Processing Functions
 */
```

```

/*
           Include File
/*          Digital Oscilloscope Project
/*
           EE/CS 52
/*
*****



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



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



#ifndef __KEYPROC_H__
#define __KEYPROC_H__


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

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



/* constants */
/* none */



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



/* function declarations */

```

```

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

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

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

#endif

```

Listing 17: ..//osc134/lcdout.c

```

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

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

The local functions included are:
    none

The locally global variable definitions included are:
    none

```

Revision History 3/8/94 Glen George Initial revision. 3/13/94 Glen George Updated comments. 3/13/94 Glen George Simplified code in plot_string function. 3/17/97 Glen George Updated comments.
--

```
3/17/97 Glen George      Change plot_char() and plot_string() to use
                           enum char_style instead of an int value.
5/27/98 Glen George      Change plot_char() to explicitly declare the
                           size of the external array to avoid linker
                           errors.
```

```
*/
```

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

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

```
/*
  clear_region
```

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

Arguments: x_ul (int) - x coordinate of upper left corner of the region to be cleared.

y_ul (int) - y coordinate of upper left corner of the region to be cleared.

x_size (int) - horizontal size of the region.

y_size (int) - vertical size of the region.

Return Value: None.

Input: None.

Output: A portion of the screen is cleared (set to PIXEL_WHITE).

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

Algorithms: None.

Data Structures: None.

Global Variables: None.

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

*/

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

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

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

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

/*

plot_hline

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

Arguments: start_x (int) - starting x coordinate of the line.
 start_y (int) - starting y coordinate of the line.
 length (int) - length of the line (positive for a line to the "right" and negative for a line to the "left").

Return Value: None.

```

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

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

Algorithms:     None.
Data Structures: None.

Global Variables: None.

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

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

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

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

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

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

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

```

```

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

}

/*
plot_vline

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

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

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

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

Algorithms:   None.
Data Structures: None.

Global Variables: None.

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

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

```

```

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

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

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

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

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

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

```

/*

plot_char

Description: This function outputs the passed character to the LCD screen at passed location. The passed location is given as a character position with (0,0) being the upper left corner of the screen. The character can be drawn in "normal video" (black on white) or "reverse video" (white on black).

Arguments: pos_x (int) - x coordinate (in character

```

                cells) of the character.
pos_y (int)      - y coordinate (in character
                   cells) of the character.
c (char)          - the character to plot.
style (enum char_style) - style with which to plot the
                           character (NORMAL or REVERSE).

Return Value: None.

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

Error Handling: No error checking is done on the coordinates or the
                 character (to ensure there is a bit pattern for it).

Algorithms:     None.
Data Structures: The character bit patterns are stored in an external
                  array.

Global Variables: None.

Author:          Glen George
Last Modified:   May 27, 2008

*/

```

```

void plot_char(int pos_x, int pos_y, char c, enum char_style style)
{
    /* variables */

    /* pointer to array of character bit patterns */
extern const unsigned char char_patterns[(VERT_SIZE - 1) * 128];

    int bits;          /* a character bit pattern */

    int col;           /* column loop index */
    int row;           /* character row loop index */

    int x;             /* x pixel position for the character */
    int y;             /* y pixel position for the character */

    /* setup the pixel positions for the character */
    x = pos_x * HORIZ_SIZE;
    y = pos_y * VERT_SIZE;

```

```

/* loop outputting the bits to the screen */
for (row = 0; row < VERT_SIZE; row++) {

    /* get the character bits for this row from the character table */
    if (row == (VERT_SIZE - 1))
        /* last row - blank it */
        bits = 0;
    else
        /* in middle of character, get the row from the bit patterns */
        bits = char_patterns[(c * (VERT_SIZE - 1)) + row];

    /* take care of "normal/reverse video" */
    if (style == REVERSE)
        /* invert the bits for "reverse video" */
        bits = ~bits;

    /* get the bits "in position" (high bit is output first */
    bits <<= (8 - HORIZ_SIZE);

    /* now output the row of the character, pixel by pixel */
    for (col = 0; col < HORIZ_SIZE; col++) {

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

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

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

/* all done, return */
return;

```

```
}
```

```
/*
```

```
plot_string
```

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

Arguments: pos_x (int) - x coordinate (in character cells) of the start of the string.

pos_y (int) - y coordinate (in character cells) of the start of the string.

s (const char *) - the string to output.

style (enum char style) - style with which to plot characters of the string.

Return Value: None.

Input: None.

Output: A string is output to the LCD screen.

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

Algorithms: None.

Data Structures: None.

Global Variables: None.

Author: Glen George

Last Modified: Mar. 17, 1997

```
*/
```

```

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

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

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

    /* all done, return */
    return;
}

```

Listing 18: .../osc134/lcdout.h

```

/****************************************************************************
 *                         LCDOUT.H
 *                         LCD Output Functions
 *                         Include File
 *                         Digital Oscilloscope Project
 *                         EE/CS 52
 *
 * This file contains the constants and function prototypes for the LCD output
 * functions used in the Digital Oscilloscope project and defined in lcdout.c.

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

```

```

#ifndef __LCDOUT_H__
#define __LCDOUT_H__


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


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


/* constants */

/* character output styles */

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

/* structures, unions, and typedefs */

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

/* function declarations */

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

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

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

```

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

```
#endif
```

Listing 19: ..//osc134/tracutil.c

```
*****  
/* */  
/* TRACUTIL */  
/* Trace Utility Functions */  
/* Digital Oscilloscope Project */  
/* EE/CS 52 */  
/* */  
*****  
  
/*  
This file contains the utility functions for handling traces (capturing  
and displaying data) for the Digital Oscilloscope project. The functions  
included are:  
    clear_saved_areas - clear all the save areas  
    do_trace        - start a trace  
    init_trace      - initialize the trace routines  
    plot_trace      - plot a trace (sampled data)  
    restore_menu_trace - restore the saved area under the menus  
    restore_trace    - restore the saved area of a trace  
    set_display_scale - set the type of displayed scale (and display it)  
    set_mode         - set the triggering mode  
    set_save_area    - determine an area of a trace to save  
    set_trace_size   - set the number of samples in a trace  
    trace_done       - inform this module that a trace has been completed  
    trace_rdy        - determine if system is ready to start another trace  
    trace_rearm     - re-enable tracing (in one-shot triggering mode)
```

The local functions included are:

none

The locally global variable definitions included are:

cur_scale - current scale type
sample_size - the size of the sample for the trace
sampling - currently doing a sample
saved_area - saved trace under a specified area
saved_axis_x - saved trace under the x lines (axes or grid)
saved_axis_y - saved trace under the y lines (axes or grid)
saved_menu - saved trace under the menu

saved_pos_x - starting position (x coordinate) of area to save
saved_pos_y - starting position (y coordinate) of area to save
saved_end_x - ending position (x coordinate) of area to save
saved_end_y - ending position (y coordinate) of area to save
trace_status - whether or not ready to start another trace

Revision History

3/8/94 Glen George Initial revision.
3/13/94 Glen George Updated comments.
3/13/94 Glen George Fixed inversion of signal in plot_trace.
3/13/94 Glen George Added sampling flag and changed the functions init_trace, do_trace and trace_done to update the flag. Also the function trace_rdy now uses it. The function set_mode was updated to always say a trace is ready for normal triggering.
3/13/94 Glen George Fixed bug in trace restoring due to operator misuse (&& instead of &) in the functions set_axes, restore_menu_trace, and restore_trace.
3/13/94 Glen George Fixed bug in trace restoring due to the clear function (clear_saved_areas) not clearing all of the menu area.
3/13/94 Glen George Fixed comparison bug when saving traces in plot_trace.
3/13/94 Glen George Changed name of set_axes to set_display_scale and the name of axes_state to cur_scale to more accurately reflect the function/variable use (especially if add scale display types).
3/17/97 Glen George Updated comments.
3/17/97 Glen George Changed set_display_scale to use plot_hline and plot_vline functions to output axes.
5/3/06 Glen George Updated formatting.
5/9/06 Glen George Updated do_trace function to match the new definition of start_sample().
5/9/06 Glen George Removed normal_trg variable, its use is now handled by the get_trigger_mode() accessor.
5/9/06 Glen George Added tick marks to the axes display.
5/9/06 Glen George Added ability to display a grid.
5/27/08 Glen George Added is_sampling() function to be able to tell if the system is currently taking a sample.
5/27/08 Glen George Changed set_mode() to always turn off the sampling flag so samples with the old mode

```

                setting are ignored.
6/3/08 Glen George      Fixed problems with non-power of 2 display
                           sizes not working.
*/
/* library include files */
/* none */

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

/* locally global variables */

static int trace_status; /* ready to start another trace */

static int sampling;      /* currently sampling data */

static int sample_size;   /* number of data points in a sample */

static enum scale_type cur_scale; /* current display scale type */

/* traces (sampled data) saved under the axes */
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 */

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

/* traces (sampled data) saved under any area */
static unsigned char saved_area[SAVE_SIZE_Y][SAVE_SIZE_X/8]; /* saved trace
   under any area */

static int      saved_pos_x; /* starting x position of saved area */
static int      saved_pos_y; /* starting y position of saved area */
static int      saved_end_x; /* ending x position of saved area */

```

```

static int      saved_end_y; /* ending y position of saved area */

/*
 * init_trace

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

Arguments:    None.
Return Value: None.

Input:        None.
Output:       None.

Error Handling: None.

Algorithms:   None.
Data Structures: None.

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

Author:        Glen George
Last Modified: May 9, 2006

*/
void init_trace()

```

```

{
    /* variables */
    /* none */

    /* initialize system status variables */

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

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

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

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

    /* clear save areas */
    clear_saved_areas();

    /* also clear the general saved area location variables (off-screen) */
    saved_pos_x = SIZE_X + 1;
    saved_pos_y = SIZE_Y + 1;
    saved_end_x = SIZE_X + 1;
    saved_end_y = SIZE_Y + 1;

    /* done initializing, return */
    return;
}

/*
set_mode

```

Description: This function sets the locally global triggering mode based on the passed value (one of the possible enumerated values). The triggering mode is used to determine when

```
the system is ready for another trace. The sampling flag  
is also reset so a new sample will be started (if that is  
appropriate).
```

Arguments: trigger_mode (enum trigger_type) - the mode with which to
set the triggering.

Return Value: None.

Input: None.

Output: None.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Global Variables: sampling - set to FALSE to turn off sampling
trace_status - set to TRUE if not one-shot triggering.

Author: Glen George

Last Modified: May 27, 2008

*/

```
void set_mode(enum trigger_type trigger_mode)  
{  
    /* variables */  
    /* none */  
  
    /* if not one-shot triggering - ready for trace too */  
    trace_status = (trigger_mode != ONESHOT_TRIGGER);  
  
    /* turn off the sampling flag so will start a new sample */  
    sampling = FALSE;  
  
    /* all done, return */  
    return;  
}
```

```

/*
  is_sampling

  Description: This function determines whether the system is currently
               taking a sample or not. This is just the value of the
               sampling flag.

  Arguments:   None.
  Return Value: (int) - the current sampling status (TRUE if currently
                      trying to take a sample, FALSE otherwise).

  Input:        None.
  Output:       None.

  Error Handling: None.

  Algorithms:   None.
  Data Structures: None.

  Global Variables: sampling - determines if taking a sample or not.

  Author:        Glen George
  Last Modified: May 27, 2008

*/
int is_sampling()
{
    /* variables */
    /* none */

    /* currently sampling if sampling flag is set */
    return sampling;
}

/*
  trace_rdy

```

Description: This function determines whether the system is ready to start another trace. This is determined by whether or not the system is still sampling (sampling flag) and if it is ready for another trace (trace_status flag).

Arguments: None.

Return Value: (int) - the current trace status (TRUE if ready to do another trace, FALSE otherwise).

Input: None.

Output: None.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Global Variables: sampling - determines if ready for another trace.
trace_status - determines if ready for another trace.

Author: Glen George

Last Modified: Mar. 13, 1994

*/

```
int trace_rdy()
{
    /* variables */
    /* none */

    /* ready for another trace if not sampling and trace is ready */
    return (!sampling && trace_status);
}
```

```
/*
trace_done
```

Description: This function is called to indicate a trace has been

completed. If in normal triggering mode this means the system is ready for another trace.

Arguments: None.
Return Value: None.

Input: None.
Output: None.

Error Handling: None.

Algorithms: None.
Data Structures: None.

Global Variables: trace_status - may be set to TRUE.
sampling - set to FALSE.

Author: Glen George
Last Modified: May 9, 2006

*/

```
void trace_done()
{
    /* variables */
    /* none */

    /* done with a trace - if retriggering, ready for another one */
    if (get_trigger_mode() != ONESHOT_TRIGGER)
        /* in a retriggering mode - set trace_status to TRUE (ready) */
    trace_status = TRUE;

    /* no longer sampling data */
    sampling = FALSE;

    /* done so return */
    return;
}
```

```

/*
 trace_rearm

Description: This function is called to rearm the trace. It sets the
trace status to ready (TRUE). It is used to rearm the
trigger in one-shot mode.

Arguments: None.
Return Value: None.

Input: None.
Output: None.

Error Handling: None.

Algorithms: None.
Data Structures: None.

Global Variables: trace_status - set to TRUE.

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

*/

```

```

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

    /* rearm the trace - set status to ready (TRUE) */
    trace_status = TRUE;

    /* all done - return */
    return;
}

```

```

/*
set_trace_size

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

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

Input:        None.
Output:       None.

Error Handling: None.

Algorithms:   None.
Data Structures: None.

Global Variables: sample_size - set to the passed value.

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

*/

```

```

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

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

    /* all done, return */
    return;
}

```

```

/*

```

```

set_display_scale

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

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

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

Error Handling: None.

Algorithms:   None.
Data Structures: None.

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

Author:        Glen George
Last Modified: May 9, 2006

*/
void set_display_scale(enum scale_type scale)
{
    /* variables */
    int p;           /* x or y coordinate */

    int i;          /* loop indices */
    int j;

    /* whenever change scale type, need to clear out previous scale */
    /* unnecessary if going to SCALE_GRID or from SCALE_NONE or not changing the
       scale */
    if ((scale != SCALE_GRID) && (cur_scale != SCALE_NONE) && (scale !=
        cur_scale)) {

```

```

/* need to restore the trace under the lines (tick, grid, or axis) */

/* go through all points on horizontal lines */
for (j = -Y_TICK_CNT; j <= Y_TICK_CNT; j++) {

    /* get y position of the line */
    p = X_AXIS_POS + j * Y_TICK_SIZE;
    /* make sure it is in range */
    if (p >= PLOT_SIZE_Y)
        p = PLOT_SIZE_Y - 1;
    if (p < 0)
        p = 0;

    /* look at entire horizontal line */
    for (i = 0; i < PLOT_SIZE_X; i++) {
        /* check if this point is on or off (need to look at bits) */
        if ((saved_axis_x[j + Y_TICK_CNT][i / 8] & (0x80 >> (i % 8))) == 0)
            /* saved pixel is off */
            plot_pixel(i, p, PIXEL_WHITE);
        else
            /* saved pixel is on */
            plot_pixel(i, p, PIXEL_BLACK);
    }
}

/* go through all points on vertical lines */
for (j = -X_TICK_CNT; j <= X_TICK_CNT; j++) {

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

    /* look at entire vertical line */
    for (i = 0; i < PLOT_SIZE_Y; i++) {
        /* check if this point is on or off (need to look at bits) */
        if ((saved_axis_y[j + X_TICK_CNT][i / 8] & (0x80 >> (i % 8))) == 0)
            /* saved pixel is off */
            plot_pixel(p, i, PIXEL_WHITE);
        else
            /* saved pixel is on */
            plot_pixel(p, i, PIXEL_BLACK);
    }
}

```

```

        }
    }
}

/* now handle the scale type appropriately */
switch (scale) {

    case SCALE_AXES: /* axes for the scale */
    case SCALE_GRID: /* grid for the scale */

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

            /* get y position of the line */
            p = X_AXIS_POS + i * Y_TICK_SIZE;
            /* make sure it is in range */
            if (p >= PLOT_SIZE_Y)
                p = PLOT_SIZE_Y - 1;
            if (p < 0)
                p = 0;

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

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

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

```

```

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

/* done with the axes */
break;

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

}

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

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

```

/*

clear_saved_areas

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

Arguments: None.

Return Value: None.

Input: None.

Output: None.

```

Error Handling: None.

Algorithms:      None.
Data Structures: None.

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

Author:          Glen George
Last Modified:   May 9, 2006

*/
void clear_saved_areas()
{
    /* variables */
    int i;    /* loop indices */
    int j;

    /* 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++)
            saved_axis_x[j][i] = 0;
    for (j = 0; j <= (2 * X_TICK_CNT); j++)
        for (i = 0; i < (SIZE_Y / 8); i++)
            saved_axis_y[j][i] = 0;

    /* clear the menu save ares */
    for (i = 0; i < MENU_SIZE_Y; i++)
        for (j = 0; j < ((MENU_SIZE_X + 7) / 8); j++)
            saved_menu[i][j] = 0;

    /* clear general save area */
    for (i = 0; i < SAVE_SIZE_Y; i++)
        for (j = 0; j < (SAVE_SIZE_X / 8); j++)
            saved_area[i][j] = 0;

    /* done clearing the saved areas - return */
    return;
}

```

```
}
```

```
/*
```

```
restore_menu_trace
```

Description: This function restores the trace under the menu when the menus are turned off. (The trace was previously saved.)

Arguments: None.

Return Value: None.

Input: None.

Output: The trace under the menu is restored to the LCD screen.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Global Variables: saved_menu - used to restore trace data under the menu.

Author: Glen George

Last Modified: Mar. 13, 1994

```
*/
```

```
void restore_menu_trace()
```

```
{
```

```
/* variables */
```

```
int bit_position; /* position of bit to restore (in saved data) */  
int bit_offset; /* offset (in bytes) of bit within saved row */
```

```
int x; /* loop indices */
```

```
int y;
```

```
/* loop, restoring the trace under the menu */
```

```
for (y = MENU_UL_Y; y < (MENU_UL_Y + MENU_SIZE_Y); y++) {
```

```
/* starting a row - initialize bit position */
```

```

bit_position = 0x80; /* start at high-order bit in the byte */
bit_offset = 0;      /* first byte of the row */

    for (x = MENU_UL_X; x < (MENU_UL_X + MENU_SIZE_X); x++) {

        /* check if this point is on or off (need to look at bits) */
        if ((saved_menu[y - MENU_UL_Y][bit_offset] & bit_position) == 0)
            /* saved pixel is off */
        plot_pixel(x, y, PIXEL_WHITE);
        else
            /* saved pixel is on */
        plot_pixel(x, y, PIXEL_BLACK);

        /* move to the next bit position */
        bit_position >>= 1;
        /* check if moving to next byte */
        if (bit_position == 0) {
            /* now on high bit of next byte */
            bit_position = 0x80;
            bit_offset++;
        }
    }
}

/* restored menu area - return */
return;
}

```

/*

set_save_area

Description: This function sets the position and size of the area to be saved when traces are drawn. It also clears any data currently saved.

Arguments: pos_x (int) - x position of upper left corner of the saved area.
pos_y (int) - y position of upper left corner of the saved area.
size_x (int) - horizontal size of the saved area.

size_y (int) - vertical size of the saved area.
Return Value: None.

Input: None.
Output: None.

Error Handling: None.

Algorithms: None.
Data Structures: None.

Global Variables: saved_area - cleared.
 saved_pos_x - set to passed value.
 saved_pos_y - set to passed value.
 saved_end_x - computed from passed values.
 saved_end_y - computed from passed values.

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

*/

```
void set_save_area(int pos_x, int pos_y, int size_x, int size_y)
{
    /* variables */
    int x; /* loop indices */
    int y;

    /* just setup all the locally global variables from the passed values */
    saved_pos_x = pos_x;
    saved_pos_y = pos_y;
    saved_end_x = pos_x + size_x;
    saved_end_y = pos_y + size_y;

    /* clear the save area */
    for (y = 0; y < SAVE_SIZE_Y; y++) {
        for (x = 0; x < (SAVE_SIZE_X / 8); x++) {
            saved_area[y][x] = 0;
        }
    }
}
```

```

/* setup the saved area - return */
return;

}

/*
restore_trace

Description: This function restores the trace under the set saved
             area. (The area was previously set and the trace was
             previously saved.)

Arguments:    None.
Return Value: None.

Input:        None.
Output:       The trace under the saved area is restored to the LCD.

Error Handling: None.

Algorithms:   None.
Data Structures: None.

Global Variables: saved_area - used to restore trace data.
                  saved_pos_x - gives starting x position of saved area.
                  saved_pos_y - gives starting y position of saved area.
                  saved_end_x - gives ending x position of saved area.
                  saved_end_y - gives ending y position of saved area.

Author:        Glen George
Last Modified: Mar. 13, 1994

*/
void restore_trace()
{
    /* variables */
    int bit_position; /* position of bit to restore (in saved data) */
    int bit_offset;   /* offset (in bytes) of bit within saved row */

    int x;      /* loop indices */
    int y;
}

```

```

/* loop, restoring the saved trace */
for (y = saved_pos_y; y < saved_end_y; y++) {

    /* starting a row - initialize bit position */
    bit_position = 0x80; /* start at high-order bit in the byte */
    bit_offset = 0;      /* first byte of the row */

    for (x = saved_pos_x; x < saved_end_x; x++) {

        /* 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 */
            plot_pixel(x, y, PIXEL_WHITE);
        else
            /* saved pixel is on */
            plot_pixel(x, y, PIXEL_BLACK);

        /* move to the next bit position */
        bit_position >>= 1;
        /* check if moving to next byte */
        if (bit_position == 0) {
            /* now on high bit of next byte */
            bit_position = 0x80;
            bit_offset++;
        }
    }
}

/* restored the saved area - return */
return;
}

/*
do_trace

Description: This function starts a trace. It starts the hardware
sampling data (via a function call) and sets the trace

```

ready flag (trace_status) to FALSE and the sampling flag (sampling) to TRUE.

Arguments: None.
 Return Value: None.

Input: None.
 Output: None.

Error Handling: None.

Algorithms: None.
 Data Structures: None.

Global Variables: trace_status - set to FALSE (not ready for another trace).
 sampling - set to TRUE (doing a sample now).

Author: Glen George
 Last Modified: Mar. 13, 1994

*/

```

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

  /* start up the trace */
  /* indicate whether using automatic triggering or not */
  start_sample(get_trigger_mode() == AUTO_TRIGGER);

  /* now not ready for another trace (currently doing one) */
  trace_status = FALSE;

  /* and are currently sampling data */
  sampling = TRUE;

  /* trace is going, return */
  return;
}

```

```

/*
plot_trace

Description: This function plots the passed trace. The trace is
assumed to contain sample_size points of sampled data.
Any points falling within any of the save areas are also
saved by this routine. The data is also scaled to be
within the range of the entire screen.

Arguments:    sample (unsigned char far *) - sample to plot.
Return Value: None.

Input:        None.
Output:       The sample is plotted on the screen.

Error Handling: None.

Algorithms:   If there are more sample points than screen width the
              sample is plotted with multiple points per horizontal
              position.
Data Structures: None.

Global Variables: cur_scale - determines type of scale to plot.
                  sample_size - determines size of passed sample.
                  saved_axis_x - stores trace under x-axis.
                  saved_axis_y - stores trace under y-axis.
                  saved_menu - stores trace under the menu.
                  saved_area - stores trace under the saved area.
                  saved_pos_x - determines location of saved area.
                  saved_pos_y - determines location of saved area.
                  saved_end_x - determines location of saved area.
                  saved_end_y - determines location of saved area.

Author:        Glen George
Last Modified: May 9, 2006

*/

```

```

void plot_trace(unsigned char /*far*/ *sample)
{
    /* variables */
    int x = 0;           /* current x position to plot */

```

```

int x_pos = (PLOT_SIZE_X / 2); /* "fine" x position for multiple point
plotting */

int y;                      /* y position of point to plot */

int p;                      /* an x or y coordinate */

int i;                      /* loop indices */
int j;

/* first, clear the display to get rid of old plots */
clear_display();

/* clear the saved areas too */
clear_saved_areas();

/* re-display the menu (if it was on) */
refresh_menu();

/* plot the sample */
for (i = 0; i < sample_size; i++) {

    /* determine y position of point (note: screen coordinates invert) */
    y = (PLOT_SIZE_Y - 1) - ((sample[i] * (PLOT_SIZE_Y - 1)) / 255);

    /* plot this point */
    plot_pixel(x, y, PIXEL_BLACK);

    /* check if the point is in a save area */

    /* check if in the menu area */
    if ((x >= MENU_UL_X) && (x < (MENU_UL_X + MENU_SIZE_X)) &&
        (y >= MENU_UL_Y) && (y < (MENU_UL_Y + MENU_SIZE_Y)))
        /* point is in the menu area - save it */
        saved_menu[y - MENU_UL_Y][(x - MENU_UL_X)/8] |= (0x80 >> ((x - MENU_UL_X)
            % 8));

    /* check if in the saved area */
    if ((x >= saved_pos_x) && (x <= saved_end_x) && (y >= saved_pos_y) && (y <=
        saved_end_y))
        /* 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));

/* check if on a grid line */
/* go through all the horizontal lines */
for (j = -Y_TICK_CNT; j <= Y_TICK_CNT; j++) {

    /* get y position of the line */
    p = X_AXIS_POS + j * Y_TICK_SIZE;
    /* make sure it is in range */
    if (p >= PLOT_SIZE_Y)
        p = PLOT_SIZE_Y - 1;
    if (p < 0)
        p = 0;

    /* if the point is on this line, save it */
    if (y == p)
        saved_axis_x[j + Y_TICK_CNT][x / 8] |= (0x80 >> (x % 8));
}

/* go through all the vertical lines */
for (j = -X_TICK_CNT; j <= X_TICK_CNT; j++) {

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

    /* if the point is on this line, save it */
    if (x == p)
        saved_axis_y[j + X_TICK_CNT][y / 8] |= (0x80 >> (y % 8));
}

/* update x position */
x_pos += PLOT_SIZE_X;
/* check if at next horizontal position */
if (x_pos >= sample_size) {
    /* at next position - update positions */
    x++;
    x_pos -= sample_size;
}

```

```

    }

/* finally, output the scale if need be */
set_display_scale(cur_scale);

/* done with plot, return */
return;

}

```

Listing 20: ..//osc134/tracutil.h

```

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

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

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

```

```

#ifndef __TRACUTIL_H__
#define __TRACUTIL_H__


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

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


/* constants */

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

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

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

/* maximum size of the save area (in pixels) */

```

```

#define SAVE_SIZE_X 120 /* maximum width */
#define SAVE_SIZE_Y 16 /* maximum height */

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

/* function declarations */

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

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

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

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

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

#endif

```

Listing 21:/osc134/char57.c

```
*****
*/

```

```

/*
          CHAR57
*/
      5x7 Dot Matrix Codes
Digital Oscilloscope Project
EE/CS 52
*/
*****
*/
This file contains a table of dot matrix patterns for vertically scanned
5x7 characters. The table entries are in ASCII order with 7 bytes per
character. The table starts with 32 special characters (mostly blank
characters) then space, the start of the printable ASCII character set.
The table is called char_patterns. In each byte (horizontal row) the
leftmost pixel is given by bit 4 and the rightmost by bit 0.

Revision History
5/27/08 Glen George    Initial revision (from 3/10/95 version of
                           char57.asm).
*/

```

```

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

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

/* the character pattern table */
const unsigned char char_patterns[] = {

  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x00)      */
  0x04, 0x0E, 0x15, 0x04, 0x04, 0x04, 0x04, /* up arrow (0x01)    */
  0x04, 0x04, 0x04, 0x04, 0x15, 0x0E, 0x04, /* down arrow (0x02)   */
  0x00, 0x04, 0x08, 0x1F, 0x08, 0x04, 0x00, /* left arrow (0x03)   */
  0x00, 0x11, 0x11, 0x11, 0x1B, 0x14, 0x10, /* greek u (mu) (0x04) */
  0x00, 0x04, 0x02, 0x1F, 0x02, 0x04, 0x00, /* right arrow (0x05)  */
  0x00, 0x11, 0x0A, 0x04, 0x0A, 0x11, 0x00, /* multiply symbol (0x06) */
  0x00, 0x04, 0x00, 0x1F, 0x00, 0x04, 0x00, /* divide symbol (0x07) */
  0x04, 0x04, 0x1F, 0x04, 0x04, 0x00, 0x1F, /* plus/minus symbol (0x08) */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x09)      */

```

```

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x0A) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x0B) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x0C) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x0D) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x0E) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x0F) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x10) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x11) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x12) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x13) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x14) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x15) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x16) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x17) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x18) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x19) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x1A) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x1B) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x1C) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x1D) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x1E) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* UNUSED (0x1F) */  

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* space (0x20) */  

0x04, 0x04, 0x04, 0x04, 0x00, 0x04, /* ! */  

0x0A, 0x0A, 0x0A, 0x00, 0x00, 0x00, /* " */  

0x0A, 0x0A, 0x1F, 0x0A, 0x1F, 0x0A, 0x0A, /* # */  

0x04, 0x0F, 0x14, 0x0E, 0x05, 0x1E, 0x04, /* $ */  

0x18, 0x19, 0x02, 0x04, 0x08, 0x13, 0x03, /* % */  

0x08, 0x14, 0x14, 0x08, 0x15, 0x12, 0x0D, /* & */  

0x0C, 0x0C, 0x08, 0x10, 0x00, 0x00, 0x00, /* , */  

0x02, 0x04, 0x08, 0x08, 0x04, 0x02, /* ( */  

0x08, 0x04, 0x02, 0x02, 0x04, 0x08, /* ) */  

0x04, 0x15, 0x0E, 0x1F, 0x0E, 0x15, 0x04, /* * */  

0x00, 0x04, 0x04, 0x1F, 0x04, 0x04, 0x00, /* + */  

0x00, 0x00, 0x00, 0x0C, 0x0C, 0x08, 0x10, /* , */  

0x00, 0x00, 0x00, 0x1F, 0x00, 0x00, 0x00, /* - */  

0x00, 0x00, 0x00, 0x00, 0x0C, 0x0C, /* . */  

0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x00, /* / */  

0x0E, 0x11, 0x13, 0x15, 0x19, 0x11, 0x0E, /* 0 */  

0x04, 0x0C, 0x04, 0x04, 0x04, 0x0E, /* 1 */  

0x0E, 0x11, 0x01, 0x0E, 0x10, 0x10, 0x1F, /* 2 */  

0x0E, 0x11, 0x01, 0x06, 0x01, 0x11, 0x0E, /* 3 */  

0x02, 0x06, 0x0A, 0x12, 0x1F, 0x02, 0x02, /* 4 */  

0x1F, 0x10, 0x1E, 0x01, 0x01, 0x11, 0x0E, /* 5 */  

0x06, 0x08, 0x10, 0x1E, 0x11, 0x11, 0x0E, /* 6 */

```

```

0x1F, 0x01, 0x02, 0x04, 0x08, 0x10, 0x10, /* 7 */ */
0x0E, 0x11, 0x11, 0x0E, 0x11, 0x11, 0x0E, /* 8 */ */
0x0E, 0x11, 0x11, 0x0F, 0x01, 0x02, 0x0C, /* 9 */ */
0x00, 0x0C, 0x0C, 0x00, 0x0C, 0x0C, 0x00, /* : */ */
0x0C, 0x0C, 0x00, 0x0C, 0x0C, 0x08, 0x10, /* ; */ */
0x02, 0x04, 0x08, 0x10, 0x08, 0x04, 0x02, /* < */ */
0x00, 0x00, 0x1F, 0x00, 0x1F, 0x00, 0x00, /* = */ */
0x08, 0x04, 0x02, 0x01, 0x02, 0x04, 0x08, /* > */ */
0x0E, 0x11, 0x01, 0x02, 0x04, 0x00, 0x04, /* ? */ */
0x0E, 0x11, 0x01, 0x0D, 0x15, 0x15, 0x0E, /* @ */ */
0x04, 0x0A, 0x11, 0x11, 0x1F, 0x11, 0x11, /* A */ */
0x1E, 0x09, 0x09, 0x0E, 0x09, 0x09, 0x1E, /* B */ */
0x0E, 0x11, 0x10, 0x10, 0x10, 0x11, 0x0E, /* C */ */
0x1E, 0x09, 0x09, 0x09, 0x09, 0x1E, /* D */ */
0x1F, 0x10, 0x10, 0x1C, 0x10, 0x10, 0x1F, /* E */ */
0x1F, 0x10, 0x10, 0x1C, 0x10, 0x10, 0x10, /* F */ */
0x0F, 0x10, 0x10, 0x13, 0x11, 0x11, 0x0F, /* G */ */
0x11, 0x11, 0x11, 0x1F, 0x11, 0x11, 0x11, /* H */ */
0x0E, 0x04, 0x04, 0x04, 0x04, 0x0E, /* I */ */
0x01, 0x01, 0x01, 0x01, 0x01, 0x11, 0x0E, /* J */ */
0x11, 0x12, 0x14, 0x18, 0x14, 0x12, 0x11, /* K */ */
0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x1F, /* L */ */
0x11, 0x1B, 0x15, 0x15, 0x11, 0x11, 0x11, /* M */ */
0x11, 0x19, 0x15, 0x13, 0x11, 0x11, 0x11, /* N */ */
0x0E, 0x11, 0x11, 0x11, 0x11, 0x11, 0x0E, /* O */ */
0x1E, 0x11, 0x11, 0x1E, 0x10, 0x10, 0x10, /* P */ */
0x0E, 0x11, 0x11, 0x11, 0x15, 0x12, 0x0D, /* Q */ */
0x1E, 0x11, 0x11, 0x1E, 0x14, 0x12, 0x11, /* R */ */
0x0E, 0x11, 0x10, 0x0E, 0x01, 0x11, 0x0E, /* S */ */
0x1F, 0x04, 0x04, 0x04, 0x04, 0x04, /* T */ */
0x11, 0x11, 0x11, 0x11, 0x11, 0x0E, /* U */ */
0x11, 0x11, 0x11, 0x0A, 0x0A, 0x04, 0x04, /* V */ */
0x11, 0x11, 0x11, 0x11, 0x15, 0x1B, 0x11, /* W */ */
0x11, 0x11, 0x0A, 0x04, 0x0A, 0x11, 0x11, /* X */ */
0x11, 0x11, 0x0A, 0x04, 0x04, 0x04, /* Y */ */
0x1F, 0x01, 0x02, 0x04, 0x08, 0x10, 0x1F, /* Z */ */
0x0E, 0x08, 0x08, 0x08, 0x08, 0x08, 0x0E, /* [ */ */
0x00, 0x10, 0x08, 0x04, 0x02, 0x01, 0x00, /* \ */ */
0x0E, 0x02, 0x02, 0x02, 0x02, 0x0E, /* ] */ */
0x04, 0x0A, 0x11, 0x00, 0x00, 0x00, 0x00, /* ^ */ */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1F, /* _ */ */
0x06, 0x06, 0x04, 0x02, 0x00, 0x00, 0x00, /* ' */ */
0x00, 0x00, 0x0E, 0x01, 0x0F, 0x11, 0x0F, /* a */ */
0x10, 0x10, 0x16, 0x19, 0x11, 0x19, 0x16, /* b */ */
0x00, 0x00, 0x0E, 0x11, 0x10, 0x11, 0x0E, /* c */ */

```

```
0x01, 0x01, 0x0D, 0x13, 0x11, 0x13, 0x0D, /* d */  
0x00, 0x00, 0x0E, 0x11, 0x1F, 0x10, 0x0E, /* e */  
0x02, 0x05, 0x04, 0x0E, 0x04, 0x04, 0x04, /* f */  
0x0D, 0x13, 0x13, 0x0D, 0x01, 0x11, 0x0E, /* g */  
0x10, 0x10, 0x16, 0x19, 0x11, 0x11, 0x11, /* h */  
0x04, 0x00, 0x0C, 0x04, 0x04, 0x04, 0x0E, /* i */  
0x01, 0x00, 0x01, 0x01, 0x01, 0x11, 0x0E, /* j */  
0x10, 0x10, 0x12, 0x14, 0x18, 0x14, 0x12, /* k */  
0x0C, 0x04, 0x04, 0x04, 0x04, 0x04, 0x0E, /* l */  
0x00, 0x00, 0x1A, 0x15, 0x15, 0x15, 0x15, /* m */  
0x00, 0x00, 0x16, 0x19, 0x11, 0x11, 0x11, /* n */  
0x00, 0x00, 0x0E, 0x11, 0x11, 0x11, 0x0E, /* o */  
0x16, 0x19, 0x11, 0x19, 0x16, 0x10, 0x10, /* p */  
0x0D, 0x13, 0x11, 0x13, 0x0D, 0x01, 0x01, /* q */  
0x00, 0x00, 0x16, 0x19, 0x10, 0x10, 0x10, /* r */  
0x00, 0x00, 0x0F, 0x10, 0x0E, 0x01, 0x1E, /* s */  
0x04, 0x04, 0x1F, 0x04, 0x04, 0x05, 0x02, /* t */  
0x00, 0x00, 0x11, 0x11, 0x11, 0x13, 0x0D, /* u */  
0x00, 0x00, 0x11, 0x11, 0x11, 0x0A, 0x04, /* v */  
0x00, 0x00, 0x11, 0x11, 0x15, 0x15, 0x0A, /* w */  
0x00, 0x00, 0x11, 0x0A, 0x04, 0x0A, 0x11, /* x */  
0x11, 0x11, 0x11, 0x0F, 0x01, 0x11, 0x0E, /* y */  
0x00, 0x00, 0x1F, 0x02, 0x04, 0x08, 0x1F, /* z */  
0x02, 0x04, 0x04, 0x08, 0x04, 0x04, 0x02, /* { */  
0x04, 0x04, 0x04, 0x00, 0x04, 0x04, 0x04, /* | */  
0x08, 0x04, 0x04, 0x02, 0x04, 0x04, 0x08, /* } */  
0x08, 0x15, 0x02, 0x00, 0x00, 0x00, 0x00, /* ~ */  
0x0A, 0x15, 0x0A, 0x15, 0x15, 0x0A /* DEL (0x7F) */
```

};

Index

- Analog
 - Software, 49
 - Hardware, 27
 - Logic, 29
- Appendix, 78
- Clock, 46
- Code, 96, 107
- Display
 - Software, 59
- FIFO, 31
- FPGA, 9
- JTAG, 44
- Keypad
 - Software, 64
- LCD, 37, 40
 - Controller, 42
- memory, 23
- Menu, 3
 - Delay, 4
 - Level, 4
- Mode, 4
- Scale, 4
- Slope, 4
- Sweep, 4
- Trigger, 4
- PCB, 88
- Power, 21
- Reset, 45
- ROM, 24
- Rotary Encoder
 - Functionality, 5
- Rotary Encoders, 32
- Schematics, 88
- Serial Device, 26
- Software, 48
- SRAM, 25
- Timing, 78
- Trigger, 30
 - Autotrigger, 31
- VRAM, 37, 40
 - Controller, 41