EE/CS 52 SoPC Digital Oscilloscope

Technical Manual

Santiago Navonne

Spring 2014

Contents

1 Introduction

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

The guide describes first the hardware, and then the software, giving for both a system overview, followed by a detailed description of every part of the system. In Appendix 3.4, the schematics and printed circuit board used in the original prototype can be found.

2 Hardware

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

2.1 System Overview

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

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

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

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

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

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

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

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

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

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

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

purposes the; *trig_period*, *trig_level*, *fifo_data*, *fifo_full*, *fifo_delay*, and *trig_ctrl* are parallel IO devices used to interface with the triggering logic. Each of these locations in memory contains several registers necessary for the interface to function. *pio_0* is another parallel IO device, this time used to interface with the rotary encoders and push buttons. Finally, *ram*, *rom*, and *vram* are the memory devices described above. Note that only memory devices have assigned chip select signals, since the other components do not require them; furthermore, note that the VRAM chip select signal does not exist in hardware external to the FPGA.

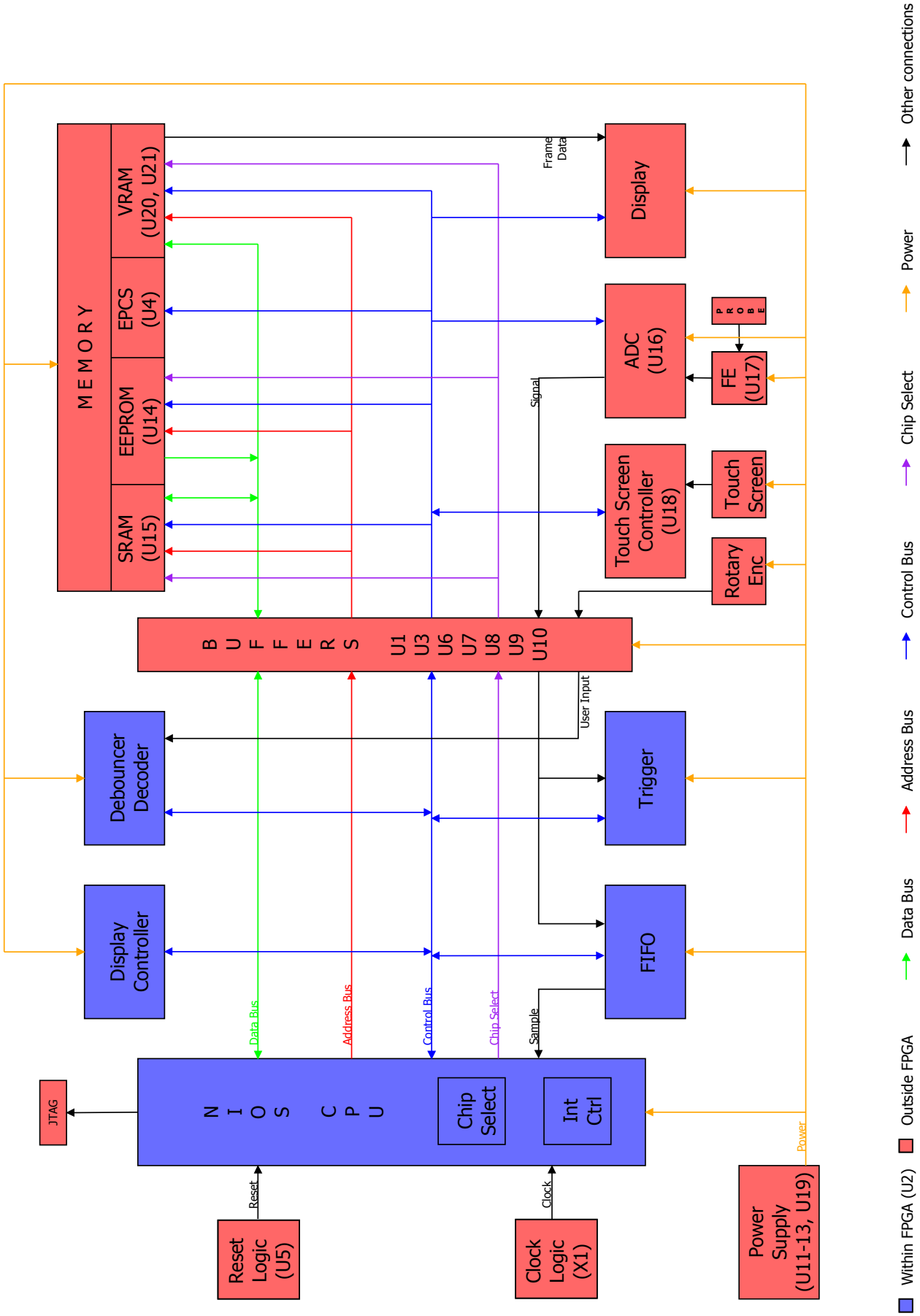


Figure 1: Highest level block diagram of the SoPC Oscilloscope system. The diagram is described in Section 2.1.

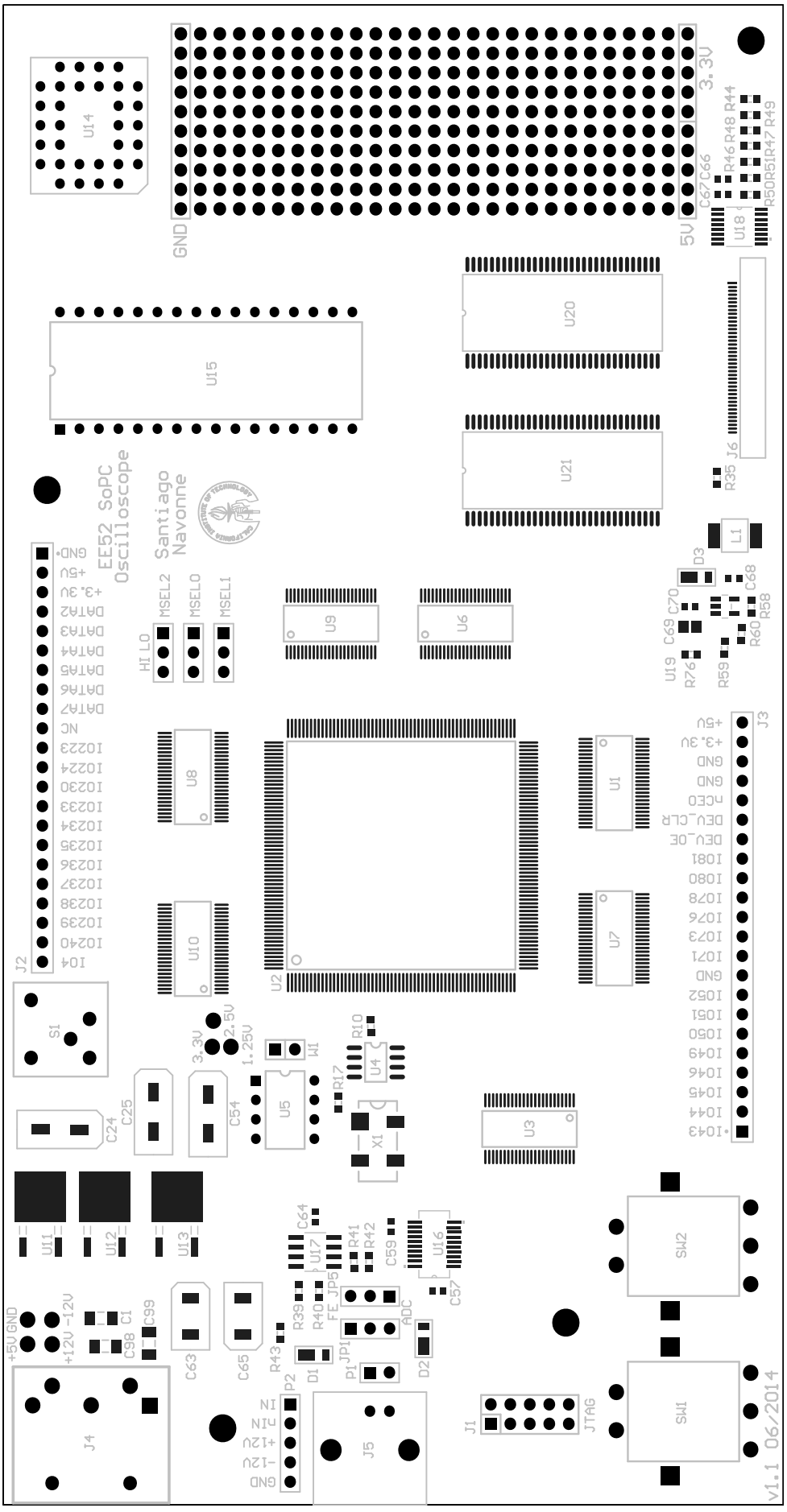


Figure 2: Front side of the system's Printed Circuit Board (PCB). The color each section is highlighted in identifies the corresponding block.

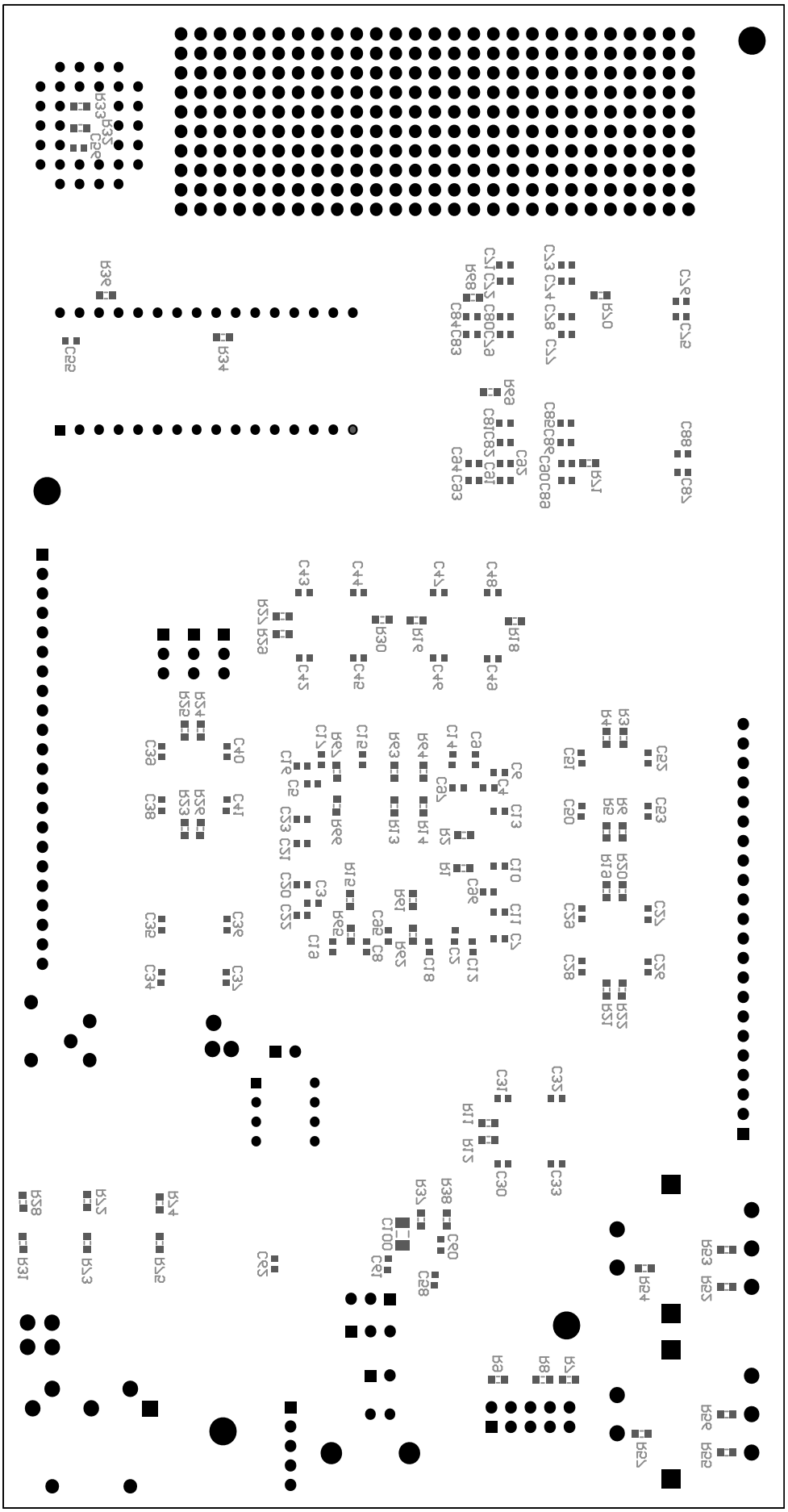


Figure 3: Back side of the system's Printed Circuit Board (PCB). Note that this side is only used for routing and placement of passive parts, such as resistors and capacitors.

Device	Address Range	Size (b...	...
fifo_data	0x00241140 - 0x0024114F	16	
fifo_full	0x00241130 - 0x0024113F	16	
jtag	0x00241180 - 0x00241187	8	..
pio_0	0x002410A0 - 0x002410BF	32	
ram	0x00220000 - 0x0023FFFF	131072	..
rom	0x00180000 - 0x001FFFFF	524288	..
trig_ctrl	0x00241060 - 0x0024107F	32	
trig_delay	0x00241120 - 0x0024112F	16	
trig_level	0x00241150 - 0x0024115F	16	
trig_period	0x00241160 - 0x0024116F	16	
vram	0x00000000 - 0x000FFFFF	1048576	..

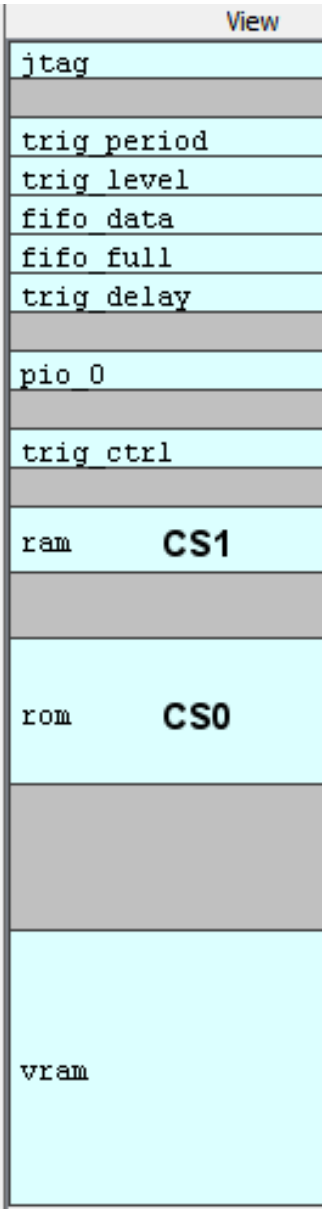


Figure 4: Memory map of the system within the NIOS processor, with associated chip select lines and addresses.

2.2 FPGA

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

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

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

The top left section of the diagram constitutes the user input section. Rotary encoder channels A (*ROT1A*, *ROT2A*) and B (*ROT1B*, *ROT2B*), as well as the push-button lines (*PUSH1*, *PUSH2*), for both devices, are input on pins 57, 63-65, 68, 69. The inputs are processed through debouncers (*DBC1*, *DBC2*) and decoders (*DEC1*, *DEC2*), which filter the signals, generating events as appropriate at their outputs. These events are collected into a bus, that is input to the NIOS processor (*CPU1*) at the *switches_in* port of *PIO_0*.

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

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

In the bottom right, we recognize the VRAM (*VRAM1*) and display (*DISP1*) controllers. These components' inputs are connected to the processor (*CPU1*), with the exception of the interconnected serial row update request (*UREQ*) and acknowledge (*UACK*) signals. The VRAM receives its own dedicated address bus shifted right twice to turn 4-byte addressing into single-word addressing (*vaddr_out19..2* to *A17..9* and *A8..0*). The dedicated chip select (*CS*) and

shared write enable (*WE*) signals are operated as for any other generic controller. All clocks are connected to the system clock. Finally, these two components output all the necessary timing signals for the display and VRAM devices on pins 82, 83, 87, 88, 93-95, 98. Additionally, the VRAM controller output a *RDY* signal, converted to an active-high *WAIT* signal for the processor through inverter *NOT*, to regulate the VRAM access cycles, which use a variable number of wait states.

In the top right, we see the system's NIOS processor (*CPU1*). Apart from the previously described connections, the devices shared address (*ADDR_BUS18..0*) and data (*DATA_BUS23..0*) buses for the ROM and RAM devices are output on pins 114, 117-120, 126-128, 131-135, 137, 139, 142, 143, 146-148, 160, 161, 166, 167, 181-189, 194-197, 200-203, 217. The write enable signal (*WE*) is output to the RAM on pin 168, and is also used as the direction signal for the data bus buffers (*U6*, *U9*) on pin 207 (*DATA_DIR*). The RAM and ROM's dedicated chip select signals (*CS1*, *CS0*) are output on pins 171 and 173, respectively.

Finally note that I^2C bus lines *I2C_SDA* and *I2C_SCL* on pins 103 and 100, as well as the *PENIRQ* input on pin 70, are left unconnected since the touch screen interface remains unimplemented. Also note that the display enable line (*DISP*) is tied high to permanently enable the device, and that buffer *U10* is configured as always enabled, output by pulling *U10_DIR* on pin 214 and *U10_OE* on pin 216 low.

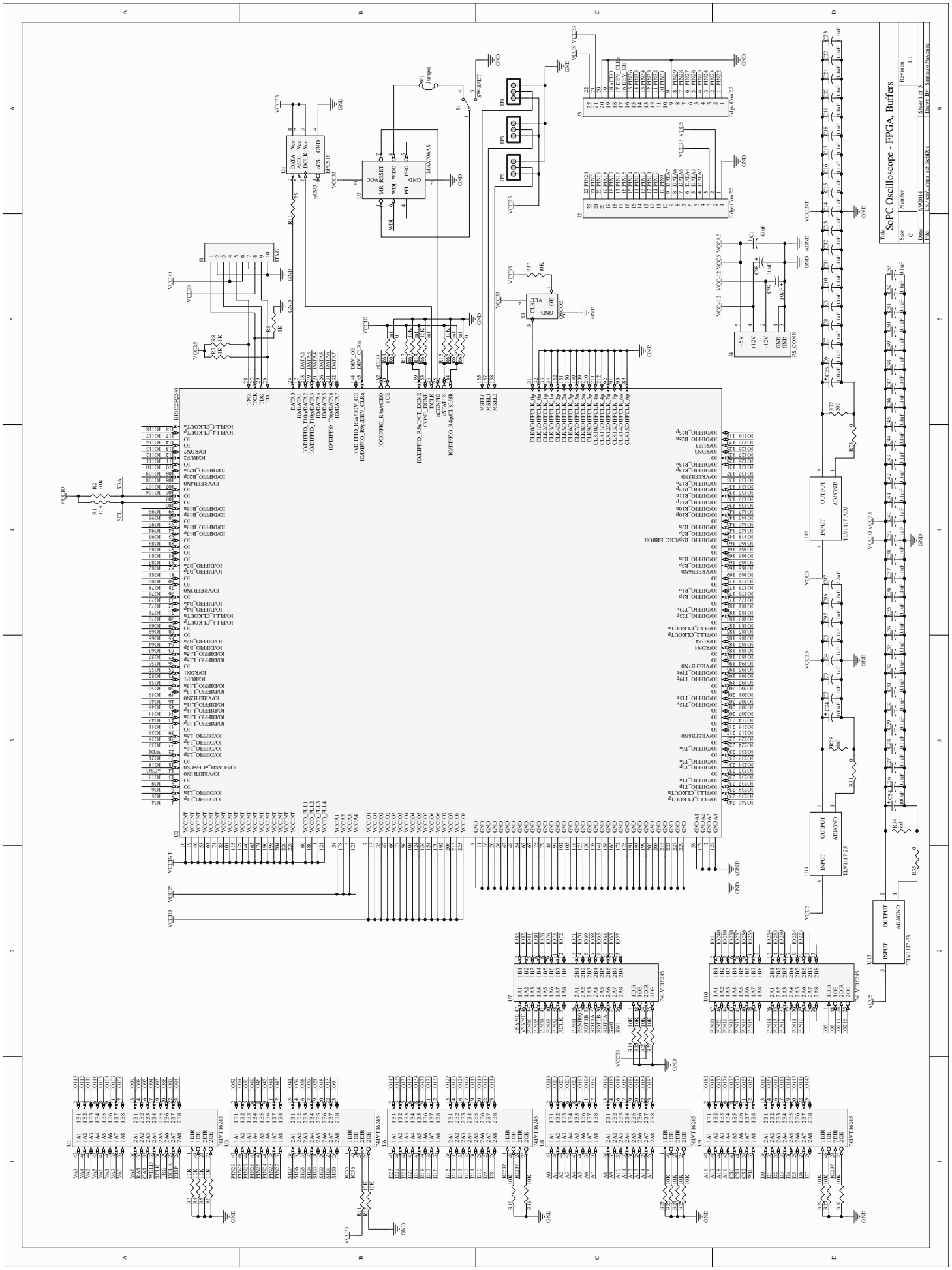


Figure 5: Schematic of the FPGA and related components. Also included are the buffers and power supply. The document is described in Section 2.2.

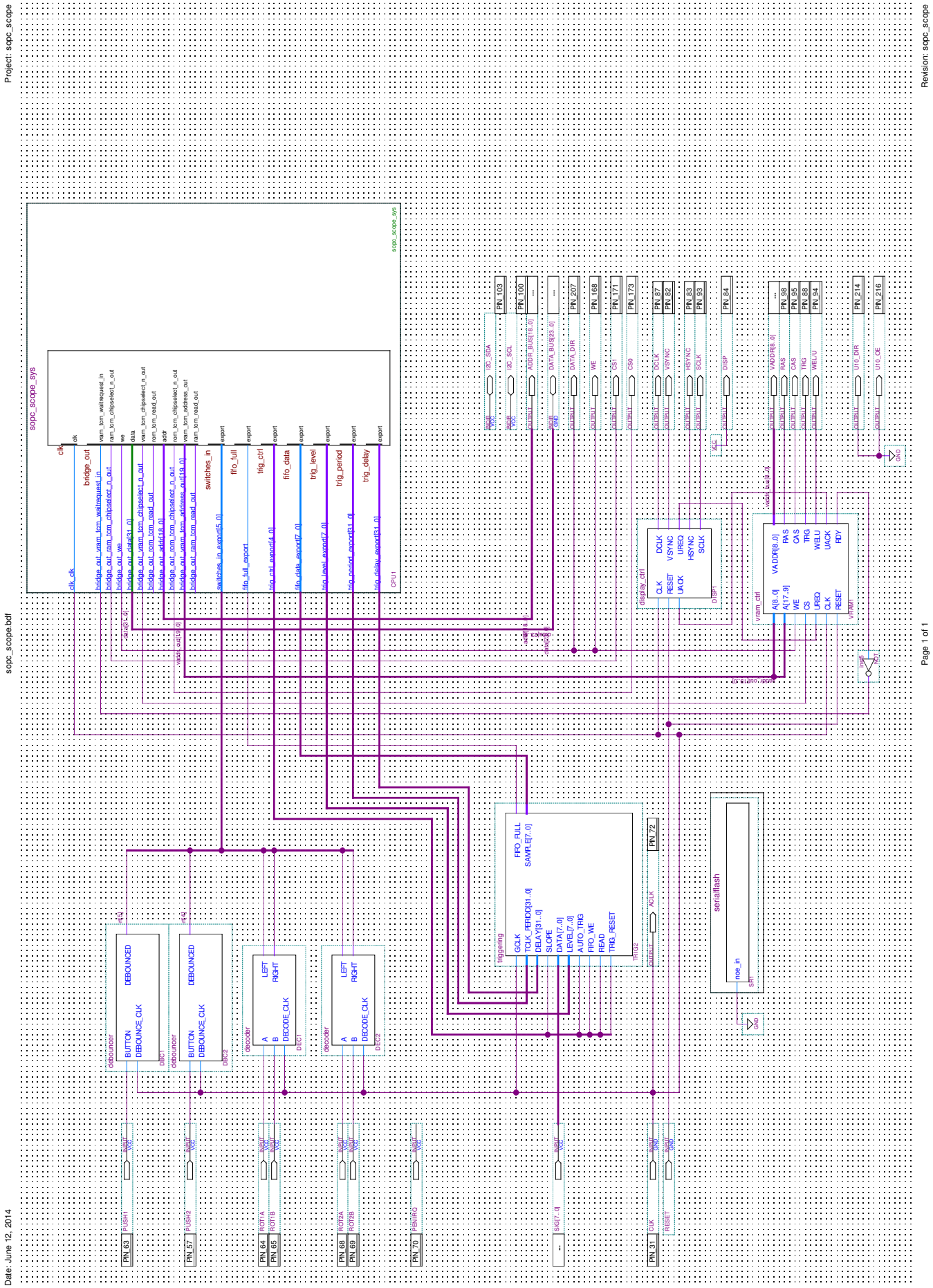


Figure 6: Main block diagram of the FPGA's internal design. The document is described in Section 2.2.

2.2.1 NIOS Processor

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

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

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

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

2.2.2 Triggering and FIFO

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

The very top section of the diagram divides the system clock (*GCLK*) down to the sample clock requested by the CPU. The requested value is input as the 32-bit duration of the sample clock in number of system clock cycles (*CLK_PERIOD31..0*). A counter (*CT1*) counts system clocks, and the output count is then compared in *CMP1* to the required period divided by two (shifted right twice in *W1*): when the count is less than the compare value, the output of *CMP1* is high; when the count is greater, the output is low. The counter is cleared by *CMP4* when the count reaches the clock period or when the triggering logic is reset (*G8*), effectively generating a 50

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

```
--------------------------------------------------------------------------------
--
--  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.
--     01 Mar 14  Santiago Navonne  Removed unnecessary 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
```

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

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

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

The bottom section of the diagram uses the so far generated delayed trigger events and other settings from the CPU to correctly fill the FIFO (*FIFO1*). When automatic triggering is disabled (*AUTO_TRIG* low), counter *CT8* gets constantly cleared and is therefore "bypassed." If writing to the FIFO is disabled because no sample has been started (*FIFO_WE* high), J/K flip-flop *FF1* will be cleared, and its output, *FIFO1*'s *wrreq*, will be disabled; no data will thus be written to the FIFO. If writing to the FIFO is enabled (*FIFO_WE* low), *FF1* will be set through *G4* whenever a delayed trigger event (output of *CMP3*) is received. Once *FF1* is set, *wrreq* becomes enabled, and samples from the ADC (*DATA7..0*), sent through three additional delayed flip-flops (*TDFF1*) for pipelining, are written to the FIFO. When the FIFO becomes full, *G5* clears *FF1*, disabling writing to *FIFO1*. The FIFO full signal also acts as an interrupt for the processor through output *FIFO_FULL*, prompting it to read the completed sample. The sample is read by first disabling writing (sending *FIFO_WE* low), and then bit-banging the read clock (*READ*). Since the read enable line (*rdreq*) is permanently enabled, every time the *READ* line transitions from low to high a new sample is output from *FIFO1* onto *SAMPLE7..0*. *RESET* signals clear the *FIFO1*.

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

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

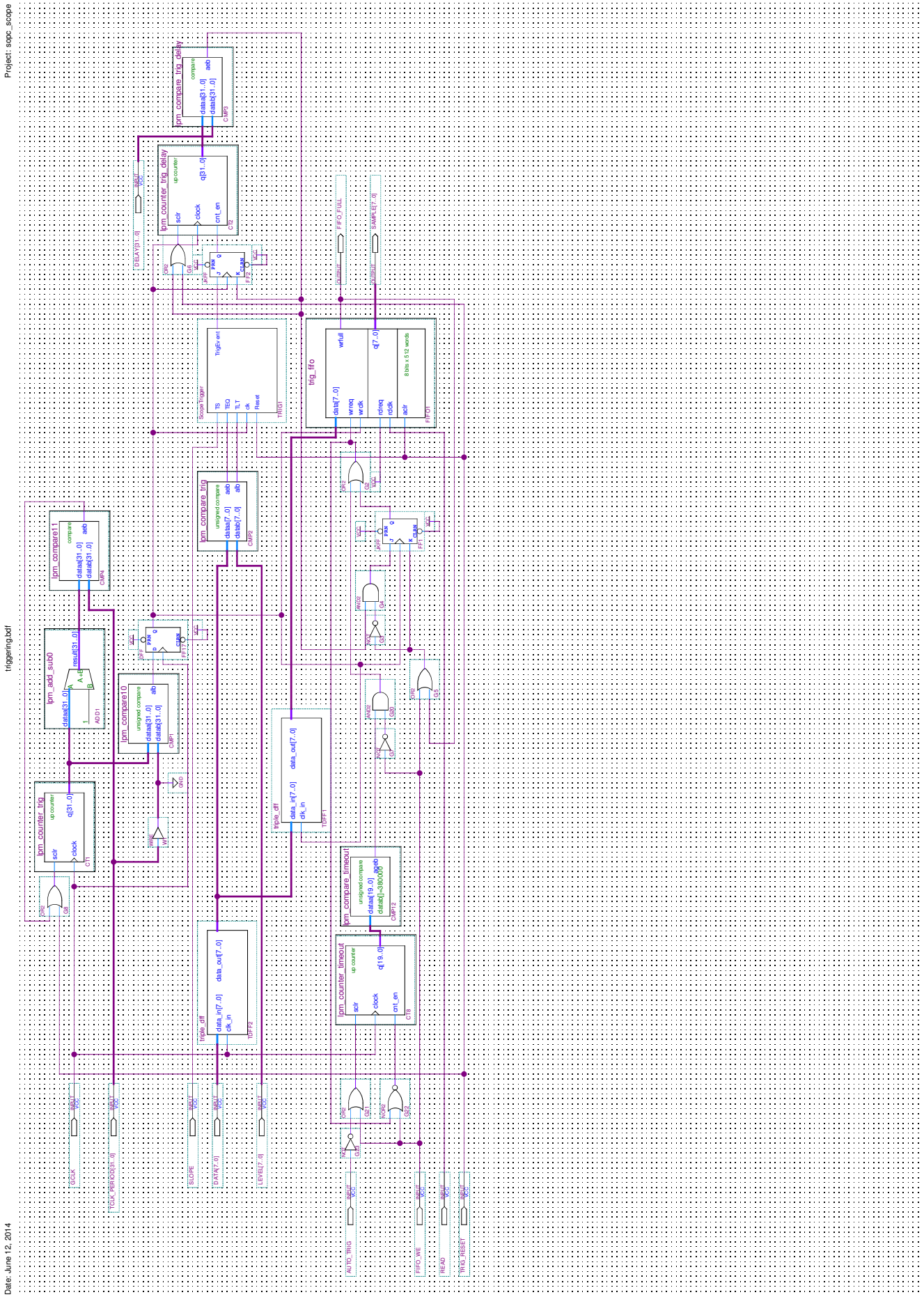


Figure 7: Block diagram of the triggering logic within the FPGA. The document is described in Section 2.2.2.

2.2.3 Debouncer

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

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

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

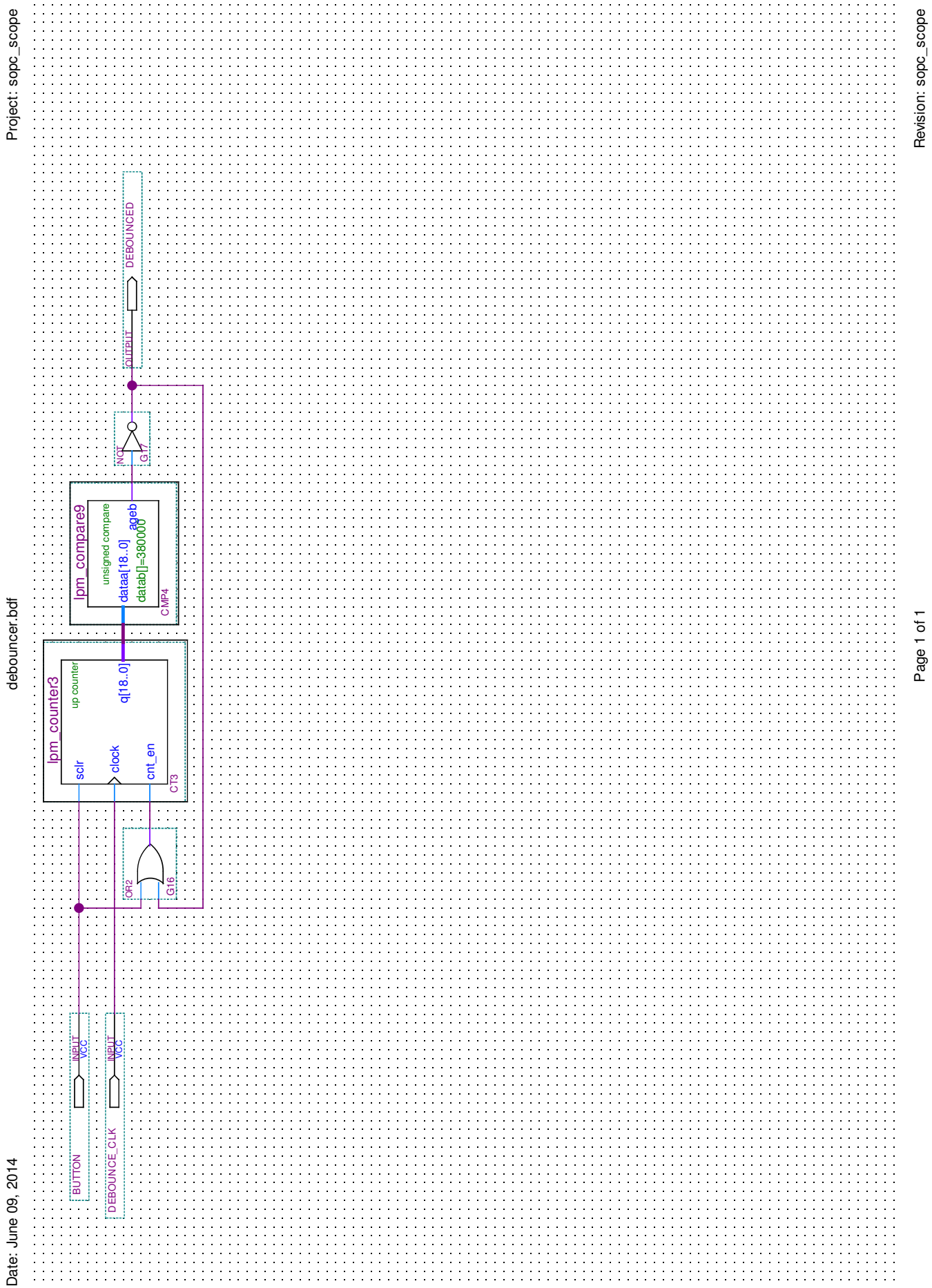


Figure 8: Block diagram of the push-button debouncer component within the FPGA. The document is described in Section 2.2.3.

2.2.4 Decoder

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

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

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

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

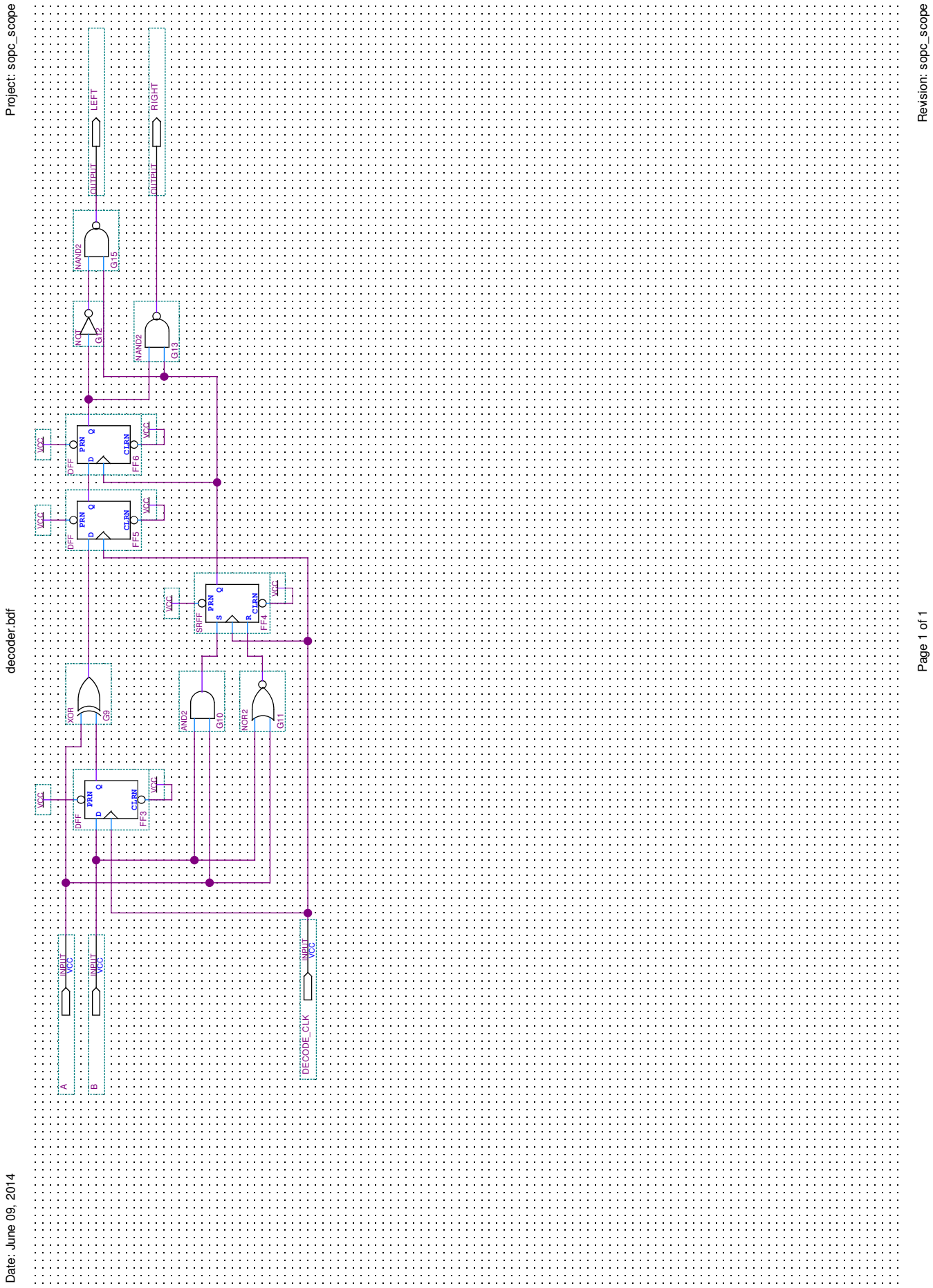


Figure 9: Block diagram of the rotary encoder decoder component within the FPGA. The document is described in Section 2.2.

2.2.5 VRAM Controller

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

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

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

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

```
--------------------------------------------------------------------------------
--
--  SoPC Oscilloscope VRAM Controller
--
--  Implementation of the VRAM Controller for the SoPC Oscilloscope project.
--  The state machine generates the necessary timing signals for the VRAM
--  based on the needs of the CPU and display controller. Additionally,
--  it refreshes the VRAM as necessary whenever no other cycle is being
--  performed.
--  The inputs to the system determine what action needs to be performed:
--  cs+we requests a read or a write, while ureq requests a SAM row update.
--  The system then outputs the necessary timing signals, and a rdy/uack
--  signal to notify the sender of the end of the cycle.
--  The state machine is implemented using a Moore state machine and a state
--  assignment architecture.
--
--
--  Revision History:
--     13 Apr 04  Glen George       Initial template.
--     20 Feb 14  Santiago Navonne  Initial revision.
--
--------------------------------------------------------------------------------


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


--
--  Oscilloscope VRAM Controller entity declaration
--

entity  VRAMCtrl  is
    port (
        we       :  in  std_logic;       -- read / not write
        cs       :  in  std_logic;       -- chip select
        ureq     :  in  std_logic;       -- serial row update request
        clk      :  in  std_logic;       -- clock
        Reset    :  in  std_logic;       -- reset the system
        ras      :  out  std_logic;      -- RAS timing signal
        cas      :  out  std_logic;      -- CAS timing signal
        trg      :  out  std_logic;      -- transfer/read signal
        welu     :  out  std_logic;      -- write signal
        asrc     :  out  std_logic;      -- address source selection
        arow     :  out  std_logic;      -- address row/column selection
        uack     :  out  std_logic;      -- serial row update acknowledge
        rdy      :  out  std_logic       -- read/write acknowledge
    );
end  VRAMCtrl;



--
--  Oscilloscope VRAM Controller Moore State Machine
--

architecture  assign_statebits  of  VRAMCtrl  is

    subtype  states  is  std_logic_vector(10 downto 0);     -- state type

    -- define the actual states as constants

    -- bits are: RAS CAS TRG WE ASRC AROW UACK RDY ID[2..0]
    constant  IDLE      : states := "11111100000";  -- waiting for events

    constant  READ1     : states := "01111100000";  -- read state 1
    constant  READ2     : states := "01011000000";  -- read state 2
```

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

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

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

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

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

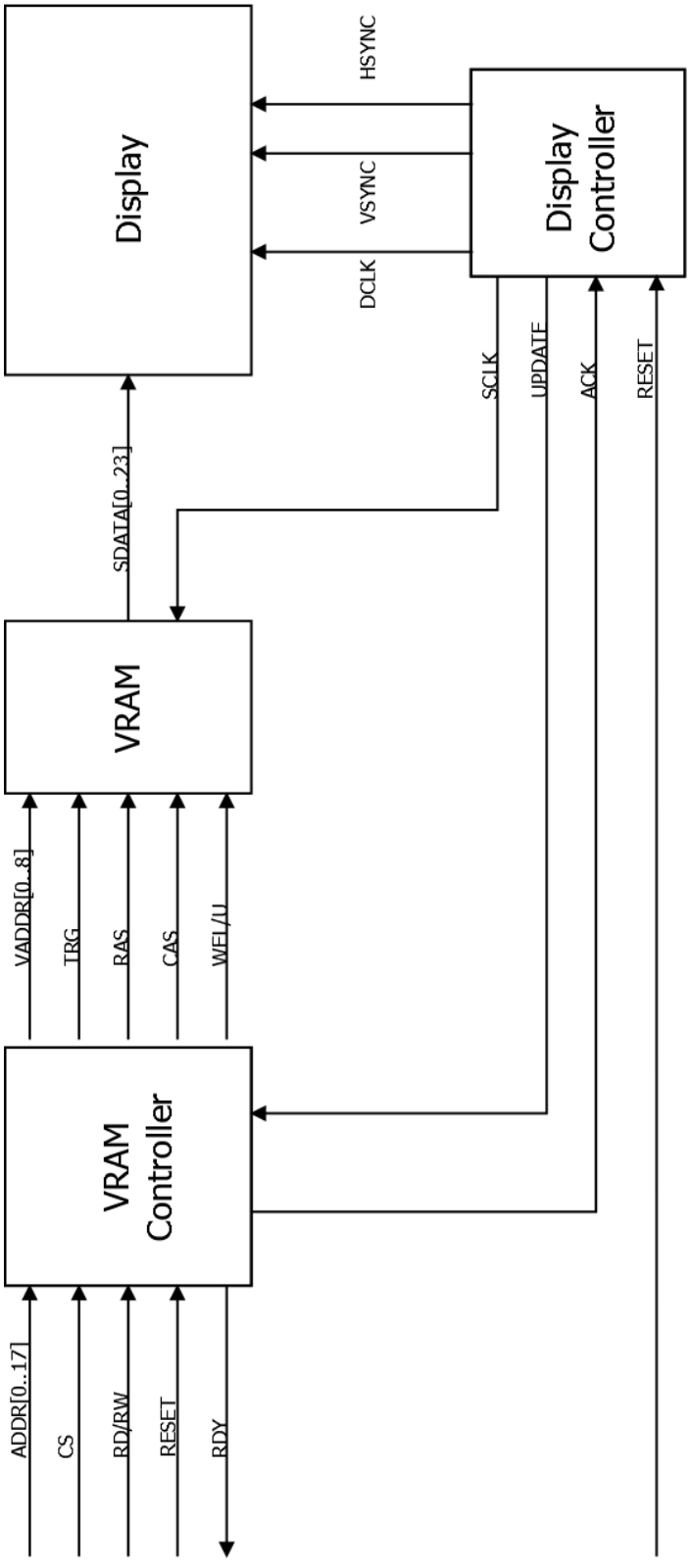


Figure 10: High level, summarizing block diagram of the VRAM and display interface. The VRAM controller listens to the CPU's generic memory controller commands, and generates the necessary timing lines for the VRAM device. The display controller periodically updates the serial row on the VRAM device,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

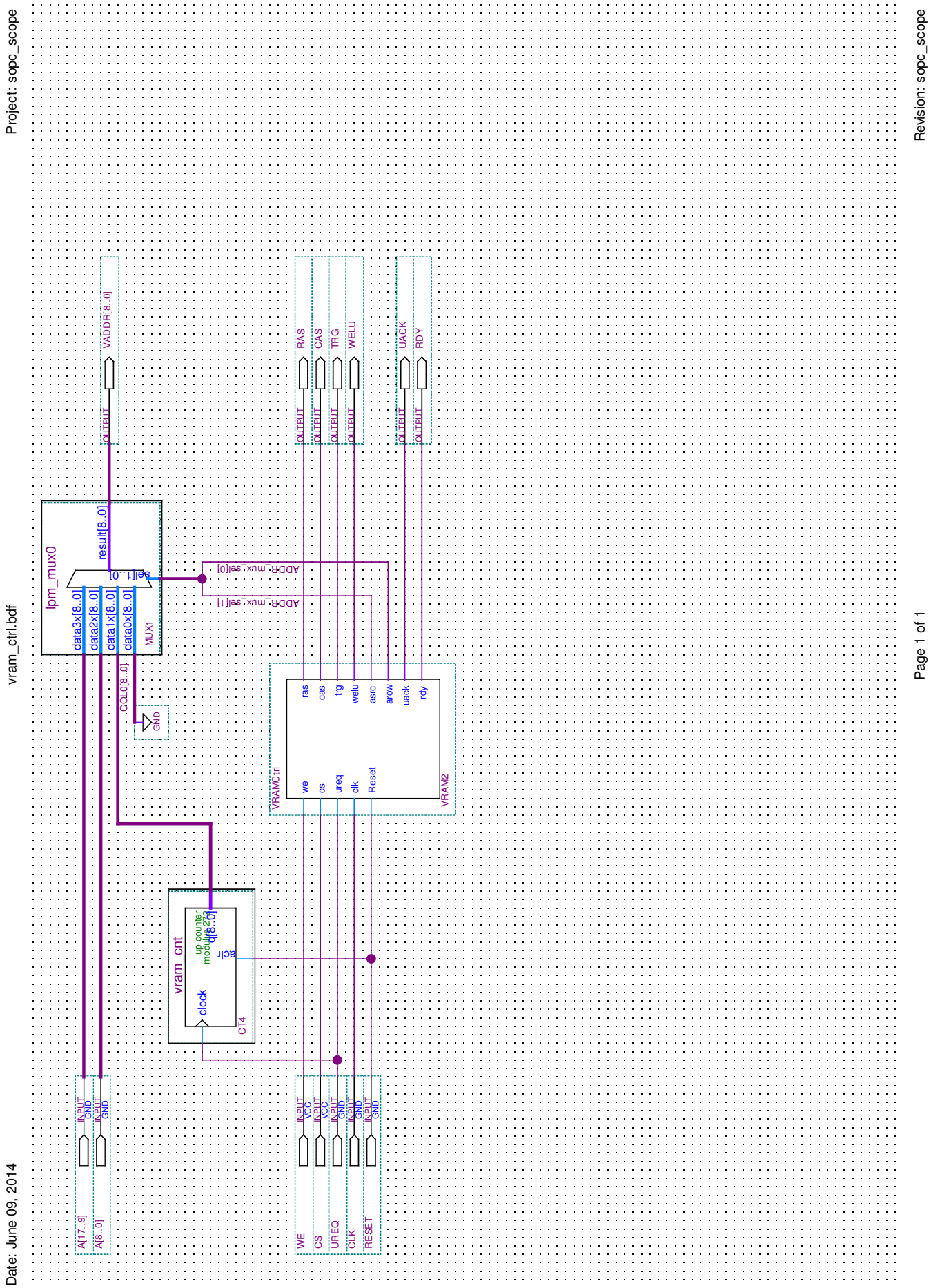


Figure 11: Block diagram of the VRAM controller component within the FPGA. The document is described in Section 2.2.5.

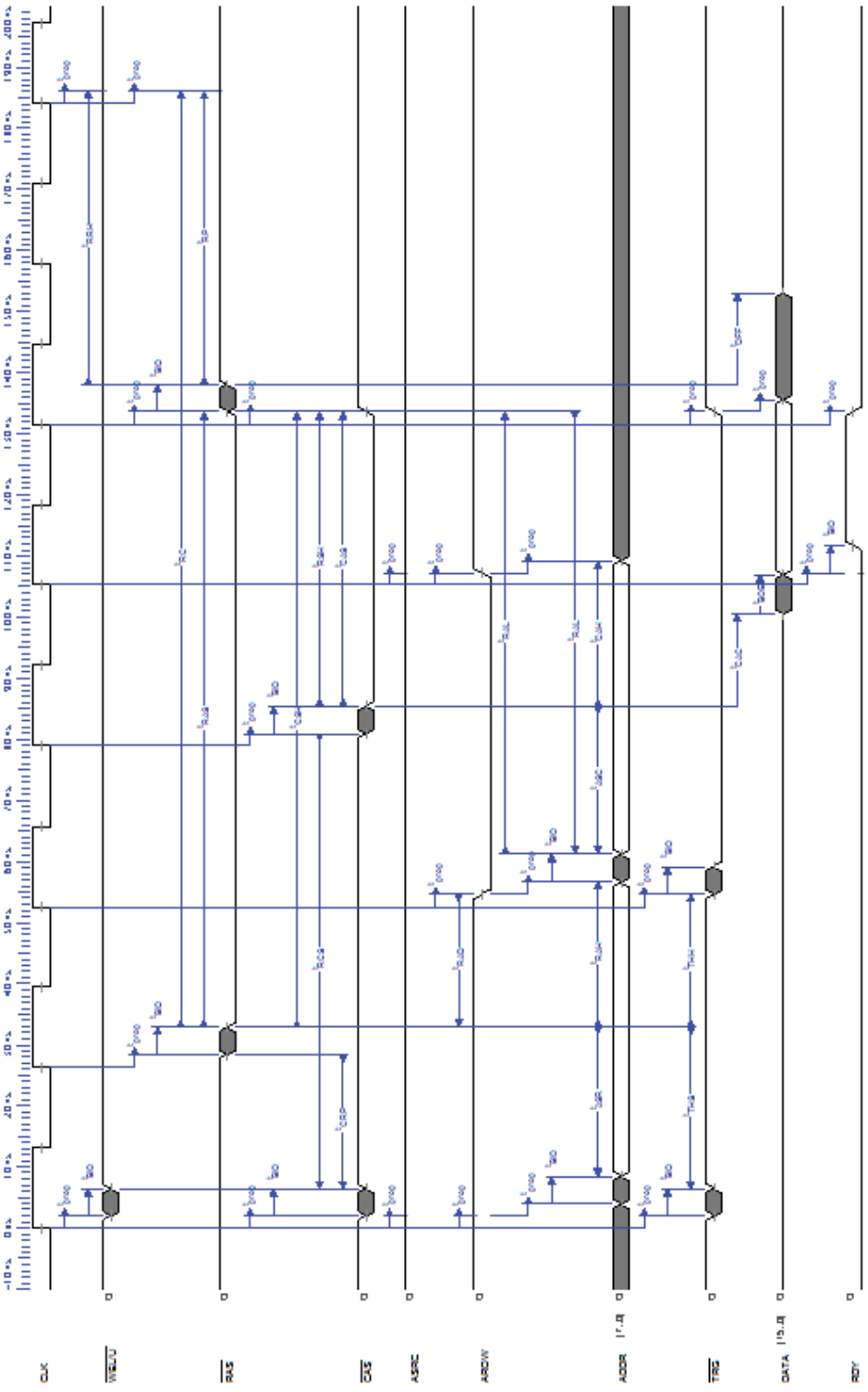


Figure 12: Timing diagram of the read cycle of the VRAM device, described in Section **??**.

General Data

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

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

General Data continued...

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

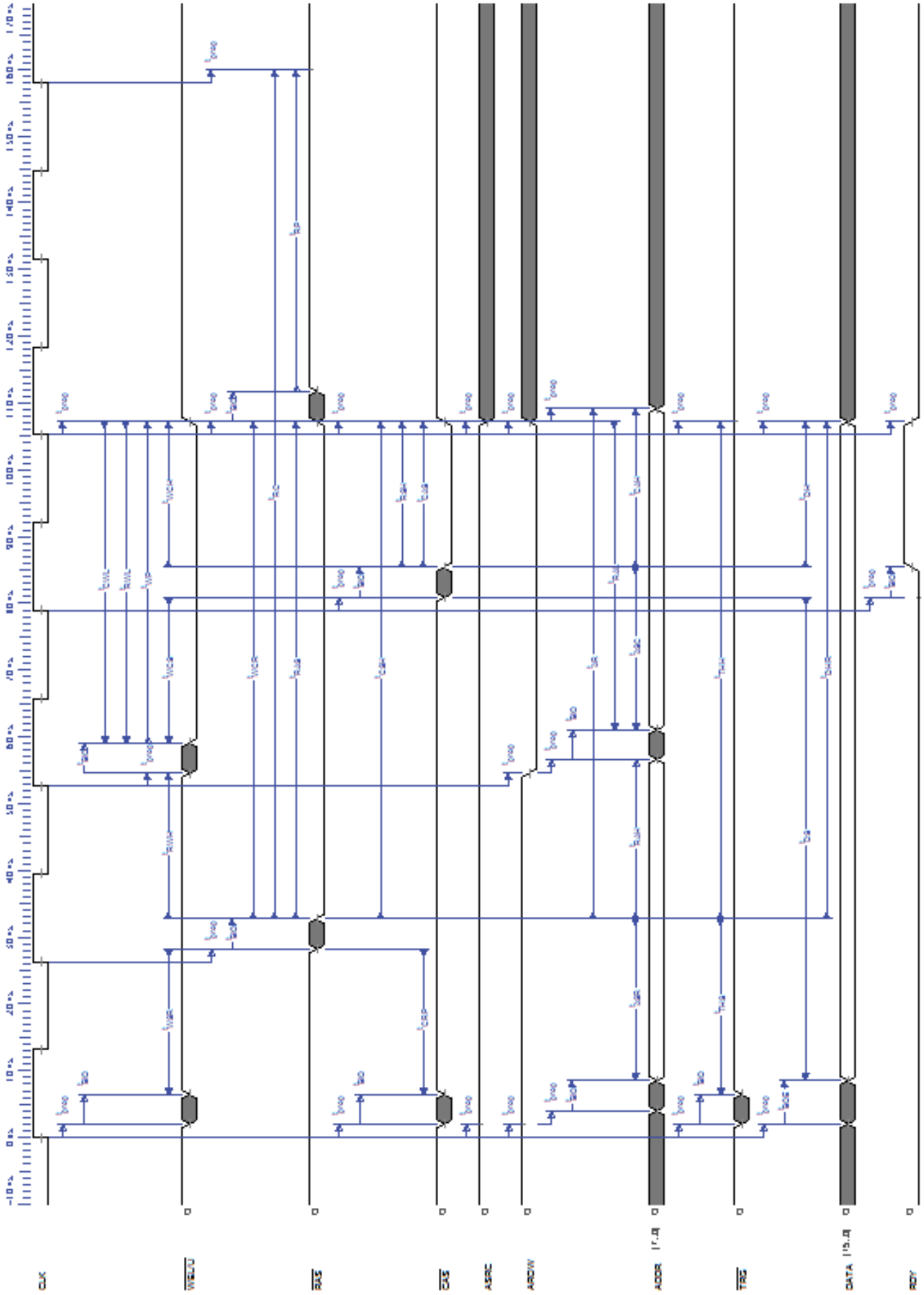


Figure 13: Timing diagram of the write cycle of the VRAM device, described in Section 2.2.5.

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

General Data

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

General Data continued...

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

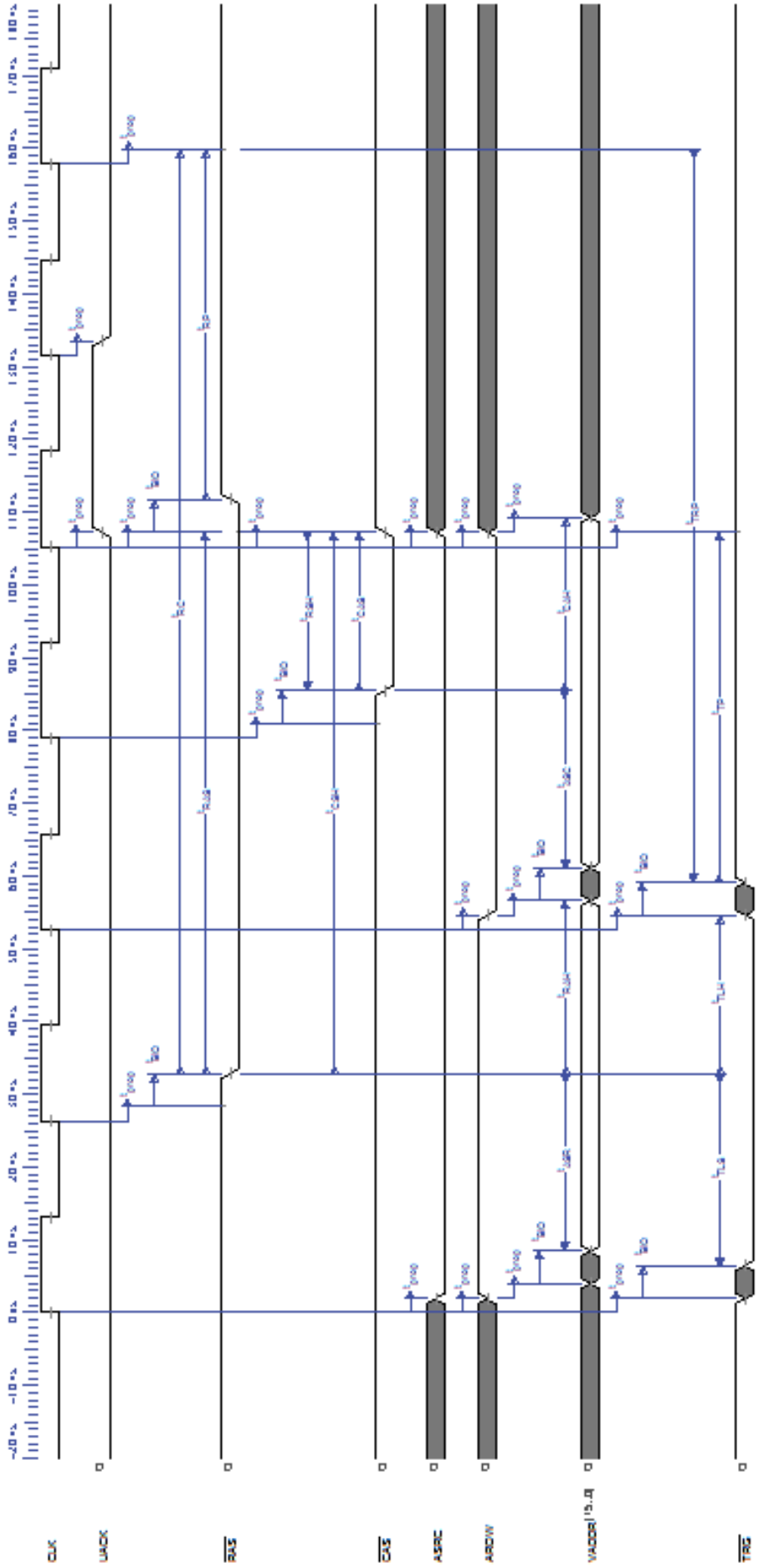


Figure 14: Timing diagram of the serial row transfer cycle of the VRAM device, described in Section 2.2.5.

General Data

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

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

General Data continued...

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

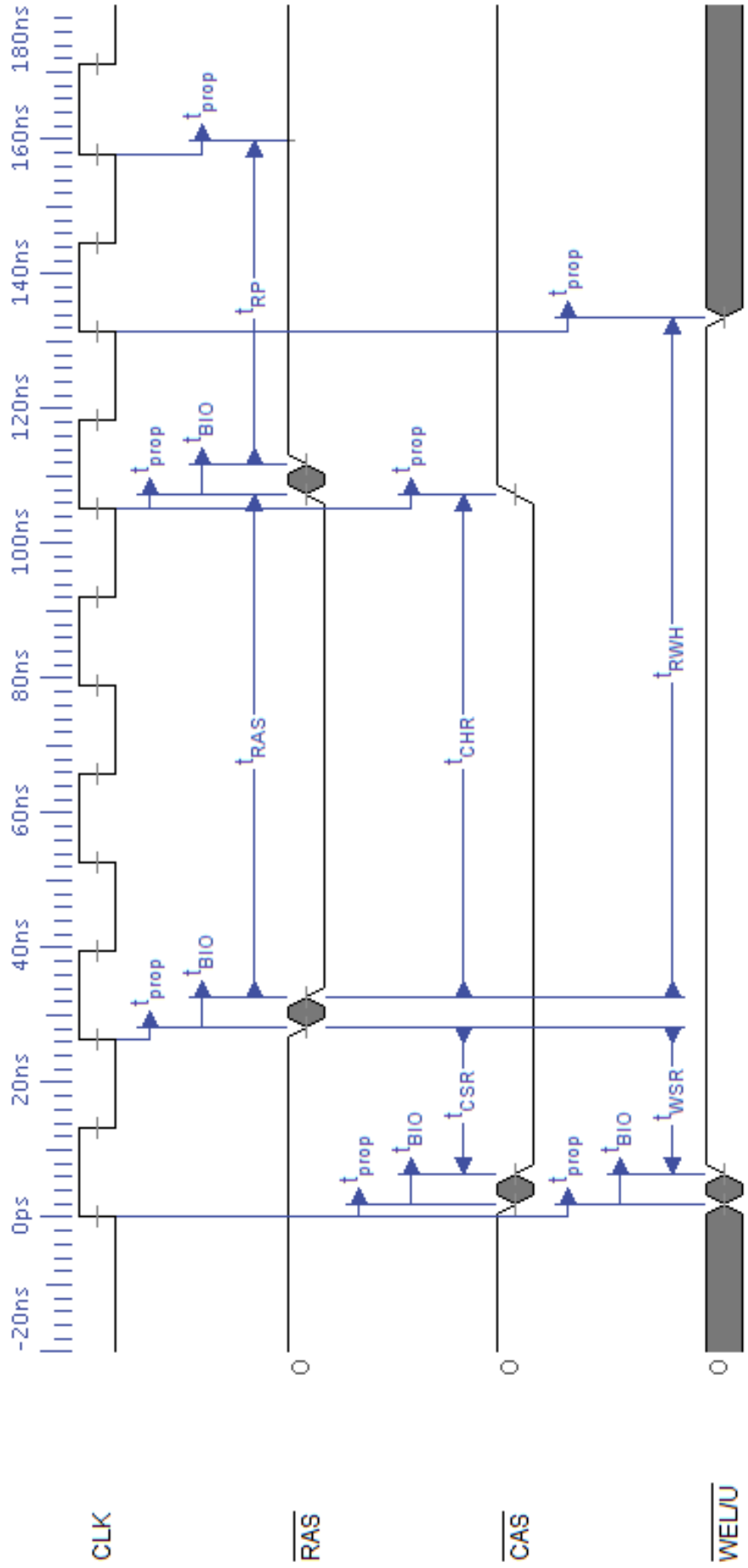


Figure 15: Timing diagram of the refresh cycle of the VRAM device, described in Section 2.2.5.

General Data

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

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

2.2.6 Display Controller

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

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

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

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

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

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

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

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

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

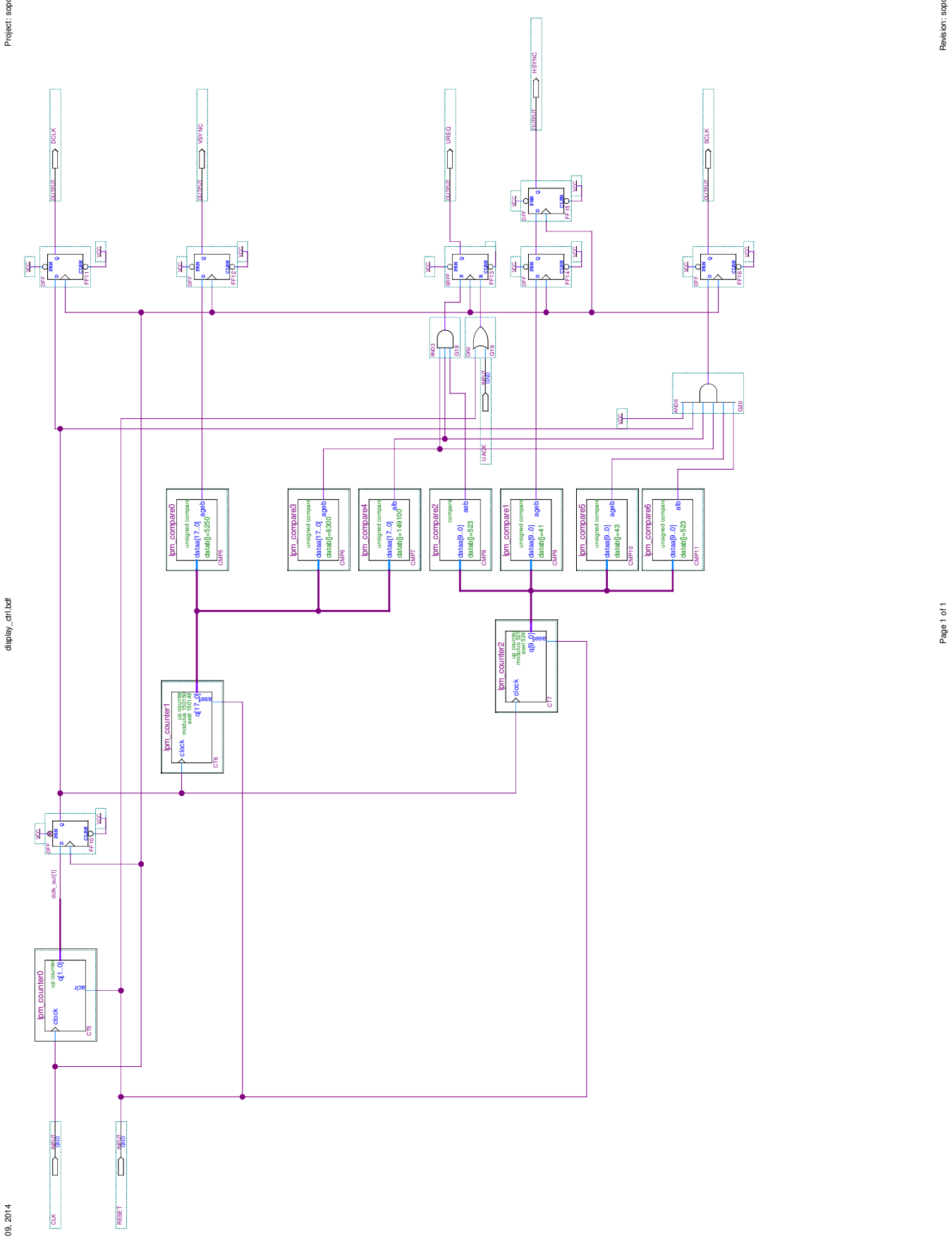
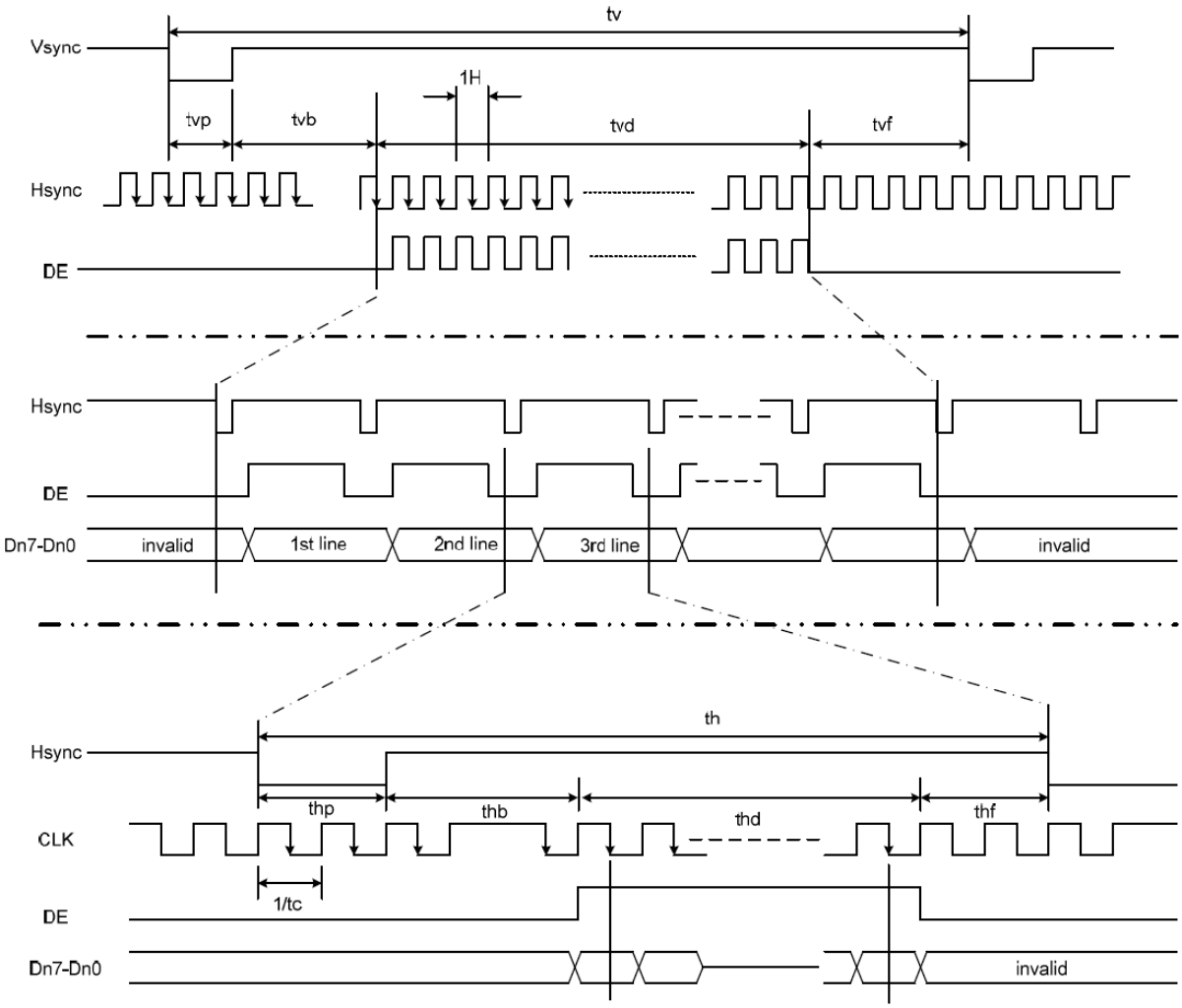


Figure 16: Block diagram of the display controller component within the FPGA. The document is described in Section 2.2.6.



[7]

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

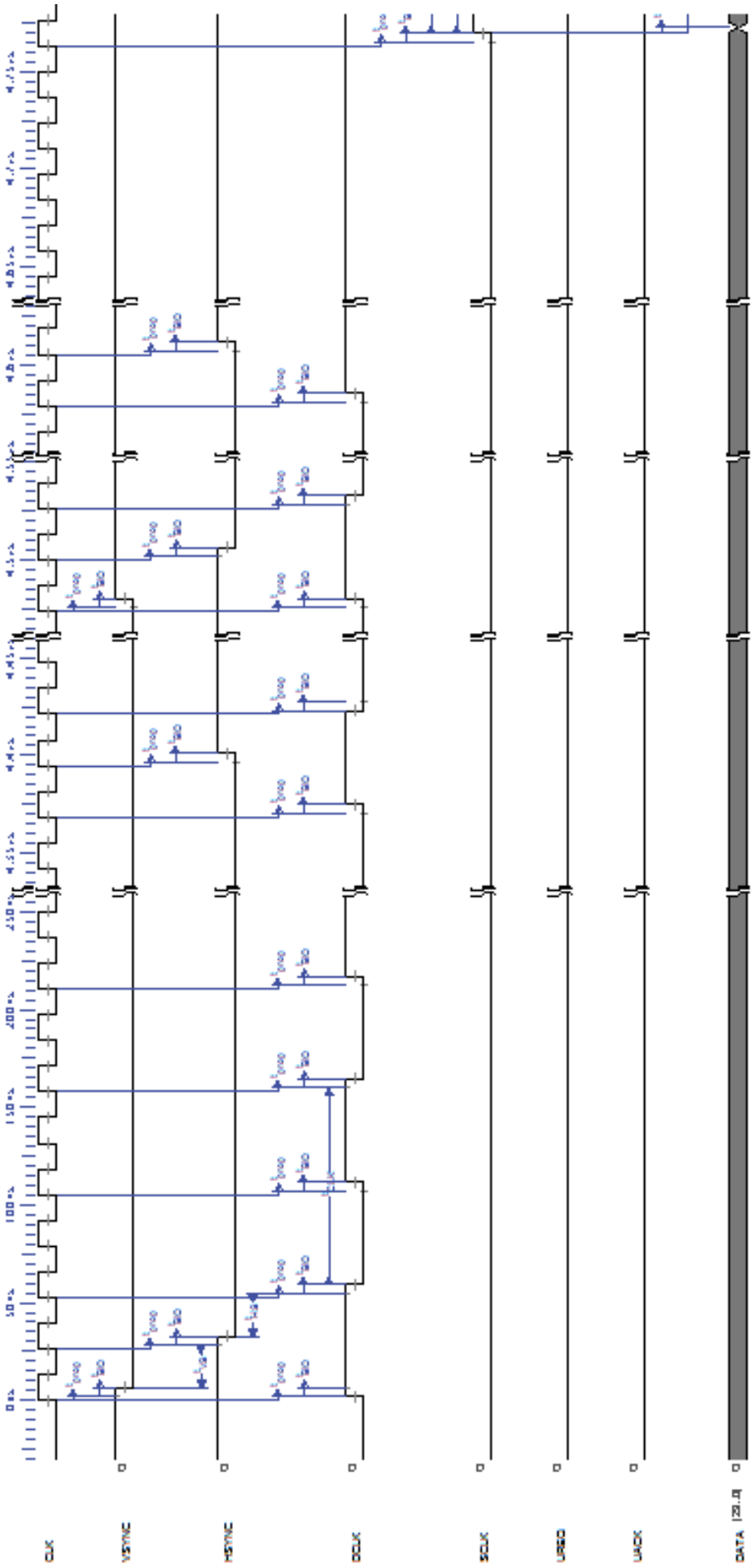
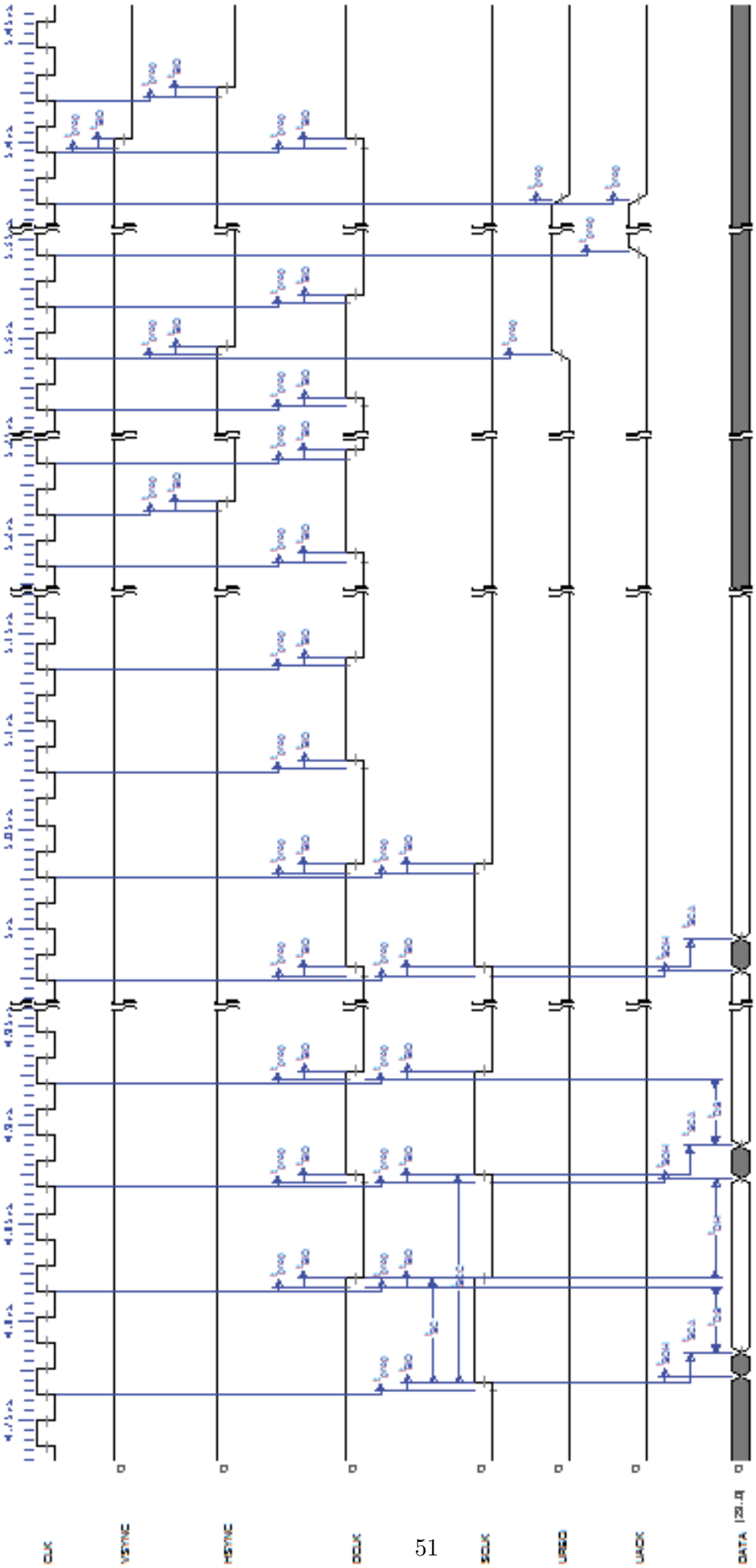


Figure 18: Timing diagram of the display cycle, described in Section 2.2.6.


CLK
VSYNC
HSYNC
DCLK
SCLK
USED
UACK
DATA [23..0]

General Data

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

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

General Data continued...

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

2.3 Reset Logic

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

2.4 Clock Logic

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

2.5 JTAG Interface

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

2.6 Power Supply

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

Additionally, the +5 V line is also regulated to +20 V to power the display's backlight LEDs, as illustrated in Figure 23. This is performed using an LMR62014 boost regulator. Resistors *R59* and *R76* are chosen to select the correct output voltage, while inductor *L1* and capacitor *C69* are selected to obtain the best output waveform characteristics.

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

2.7 Buffers

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

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

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

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

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

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

Buffer *U7* bridges the user input lines from the rotary encoders (*SW0,1*, *ROT0A*, *ROT0B*, *ROT1A*, *ROT1B*) and the interrupt line from the touch screen controller (*PENIRQ*), as well as an unused pin made available on *J3*, on port 2. These lines are configured as always-enabled, input-only (A-¿B) by tying port 2 direction line *2DIR* high through *R21* and output enable line *2OE* low through *R22*. Port 1 connects display timing lines *HSYNC* and *VSYNC*, ADC clock *ACLK*, and five unused pins; all of these lines are configured always-enabled, output-only (B-¿A) by tying both port 1 control lines low through *R19,20*.

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

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

on the backside of the board, drawn in Figure 3.

2.8 Memory

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

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

2.8.1 RAM

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

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

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

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

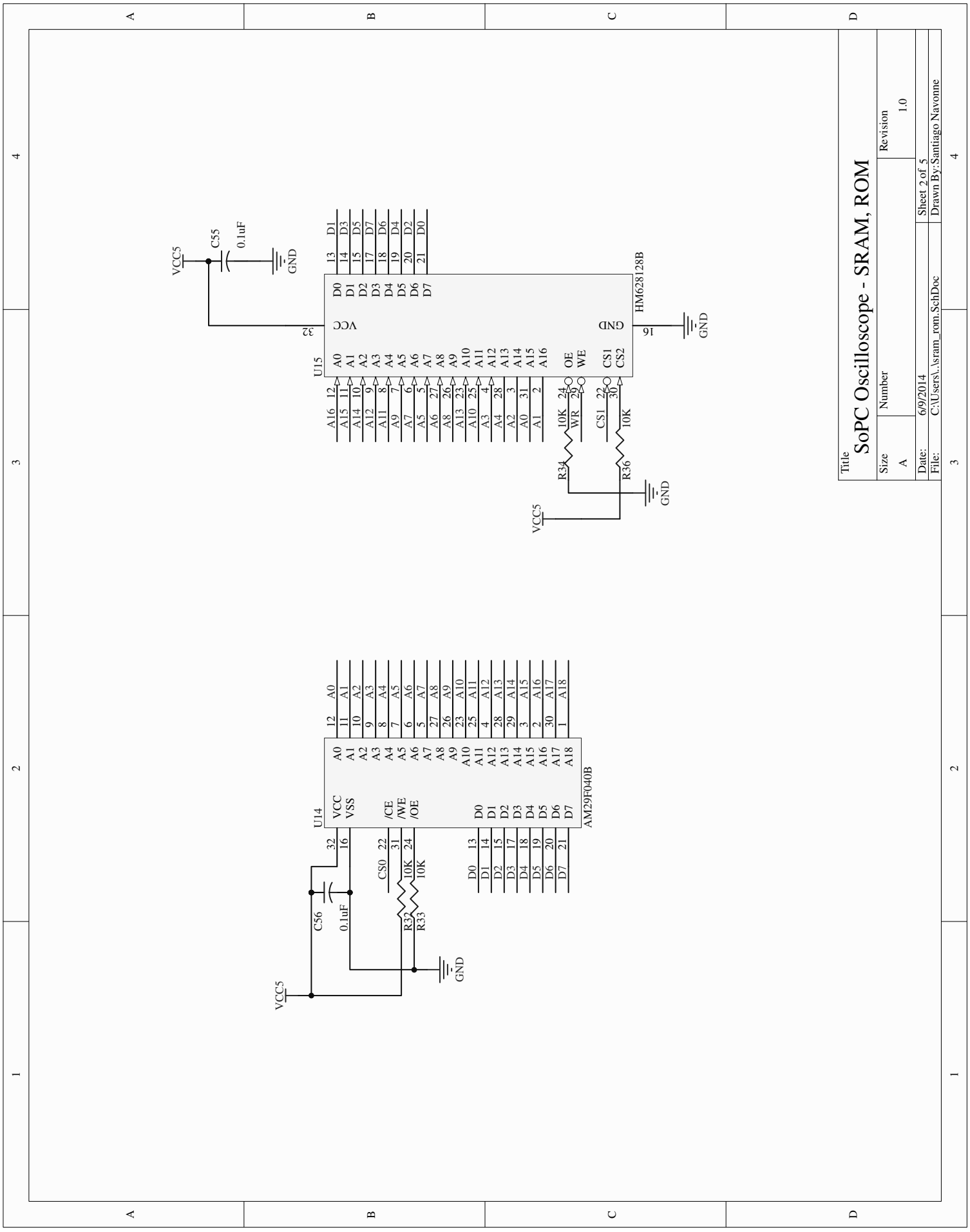


Figure 19: Schematic of the RAM and ROM memory devices. Further details are provided in Sections 2.8.1 and 2.8.2.

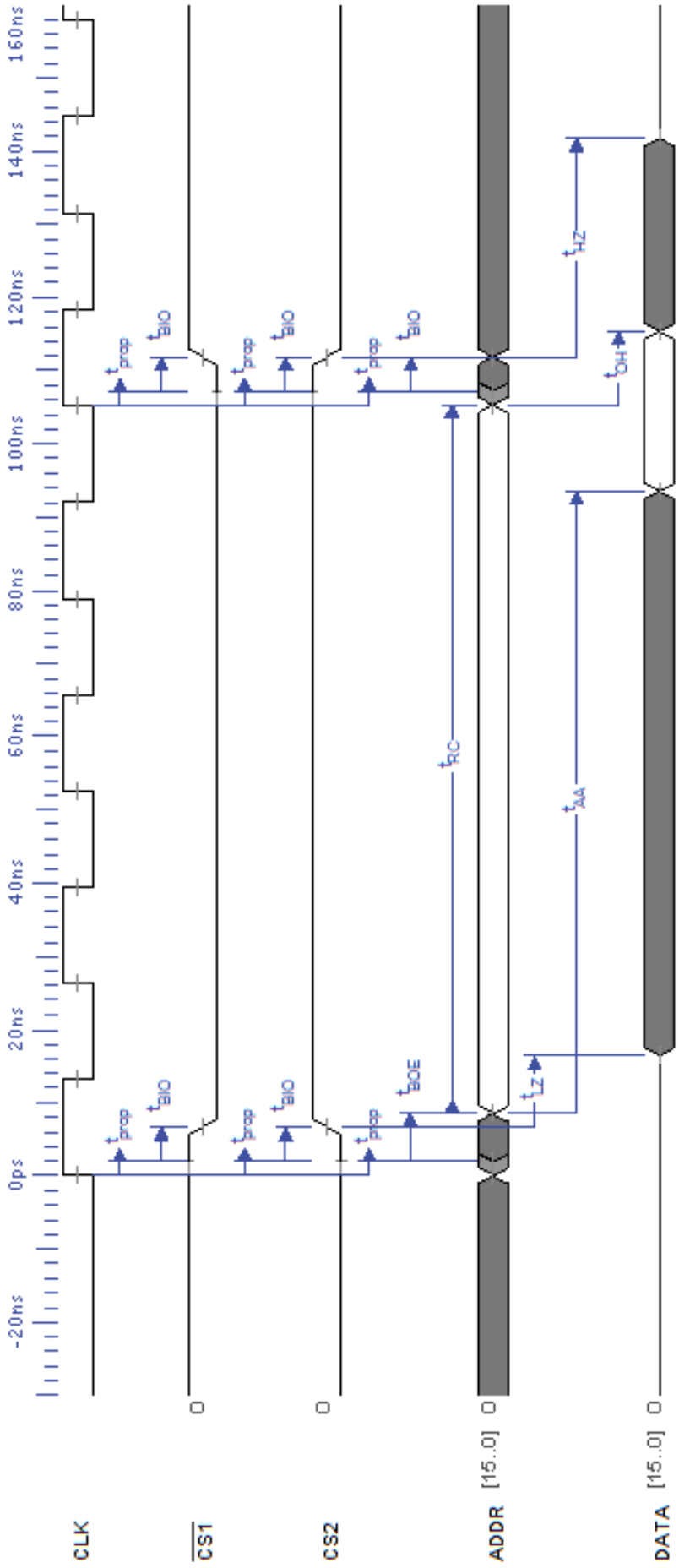


Figure 20: Timing diagram of the read cycle of the RAM device, described in Section 2.8.1.

General Data

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

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

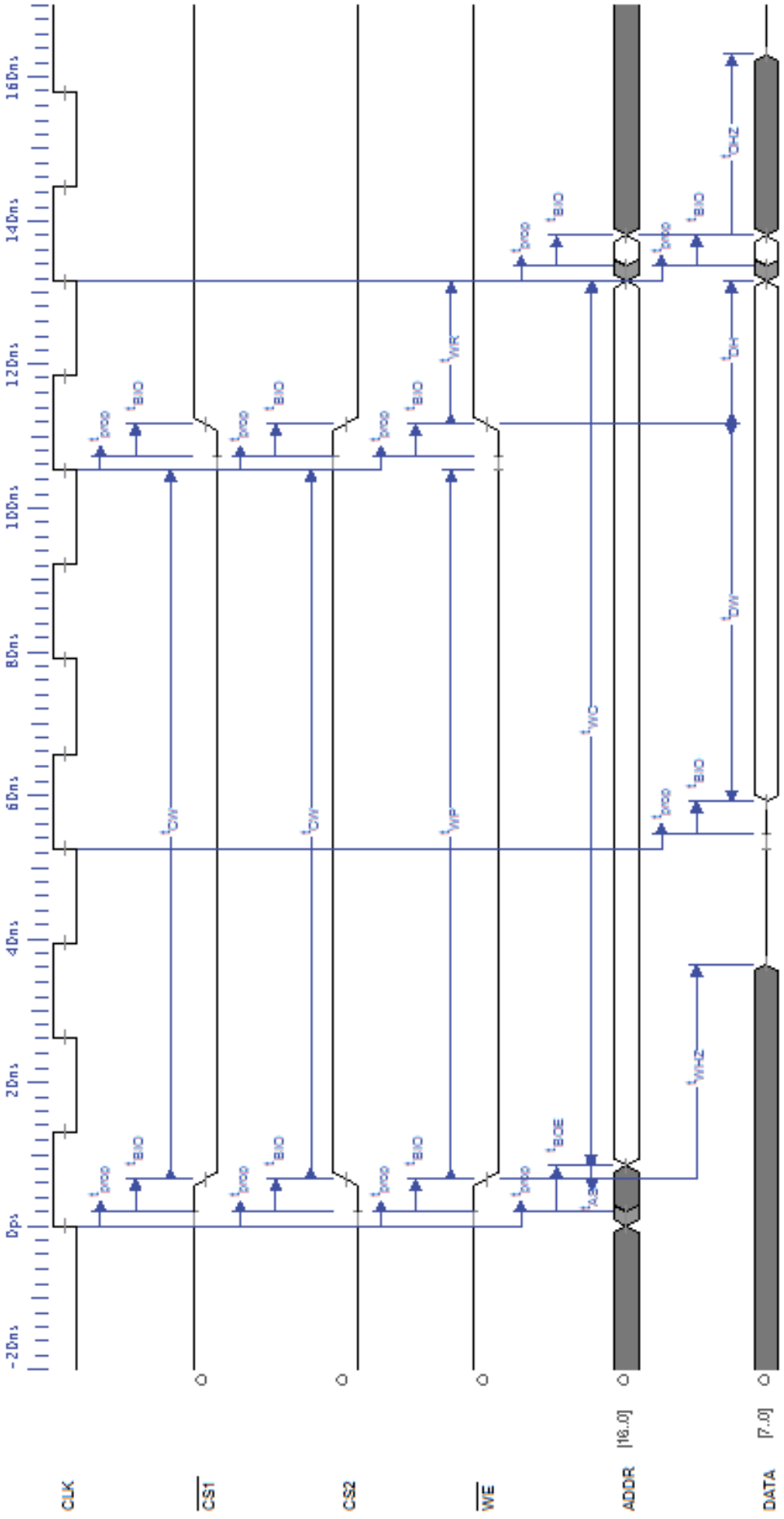


Figure 21: Timing diagram of the write cycle of the RAM device, described in Section 2.8.2.

General Data

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

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

2.8.2 ROM

A AM29F040B 512 Kword x 8-bit ROM chip, *U14*, constitutes the system's non-volatile storage. The device's connections are shown in Figure 19. The device is accessible by the processor at addresses 0x180000-0x1FFFFF, as shown in Figure 4.

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

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

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

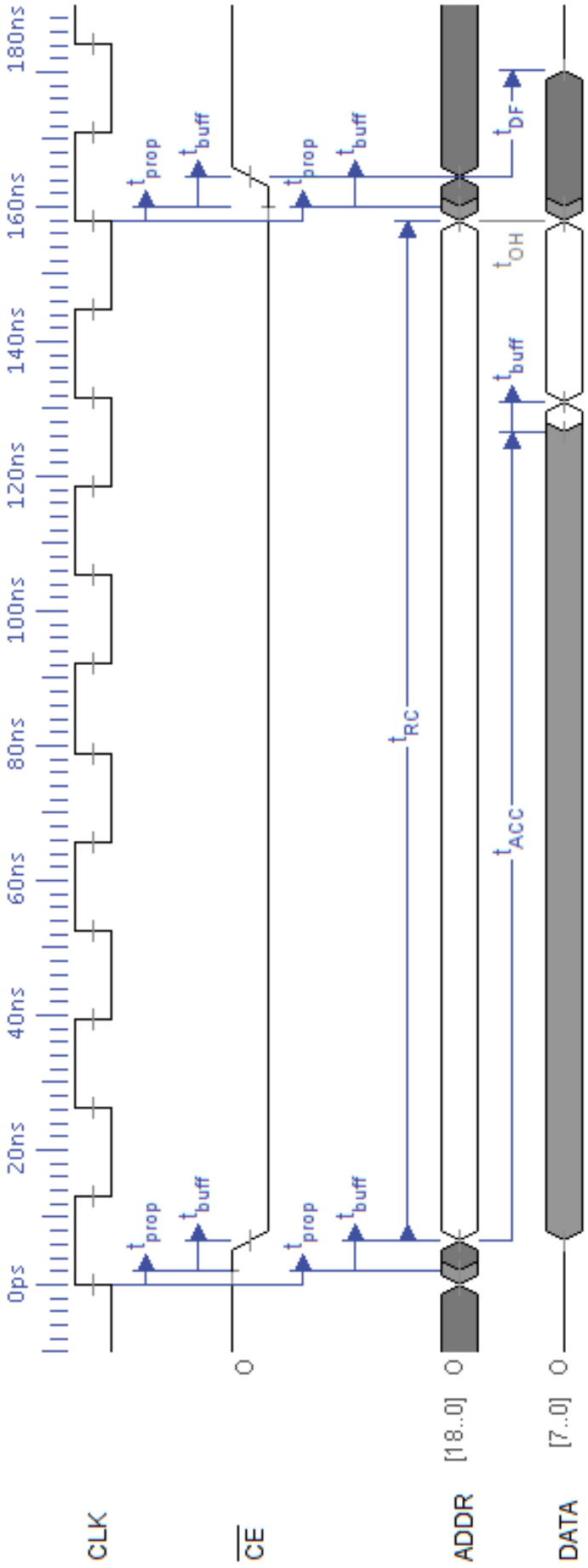


Figure 22: Timing diagram of the read cycle of the ROM device, described in Section 2.8.2.

General Data

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

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

2.8.3 VRAM

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

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

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

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

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

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

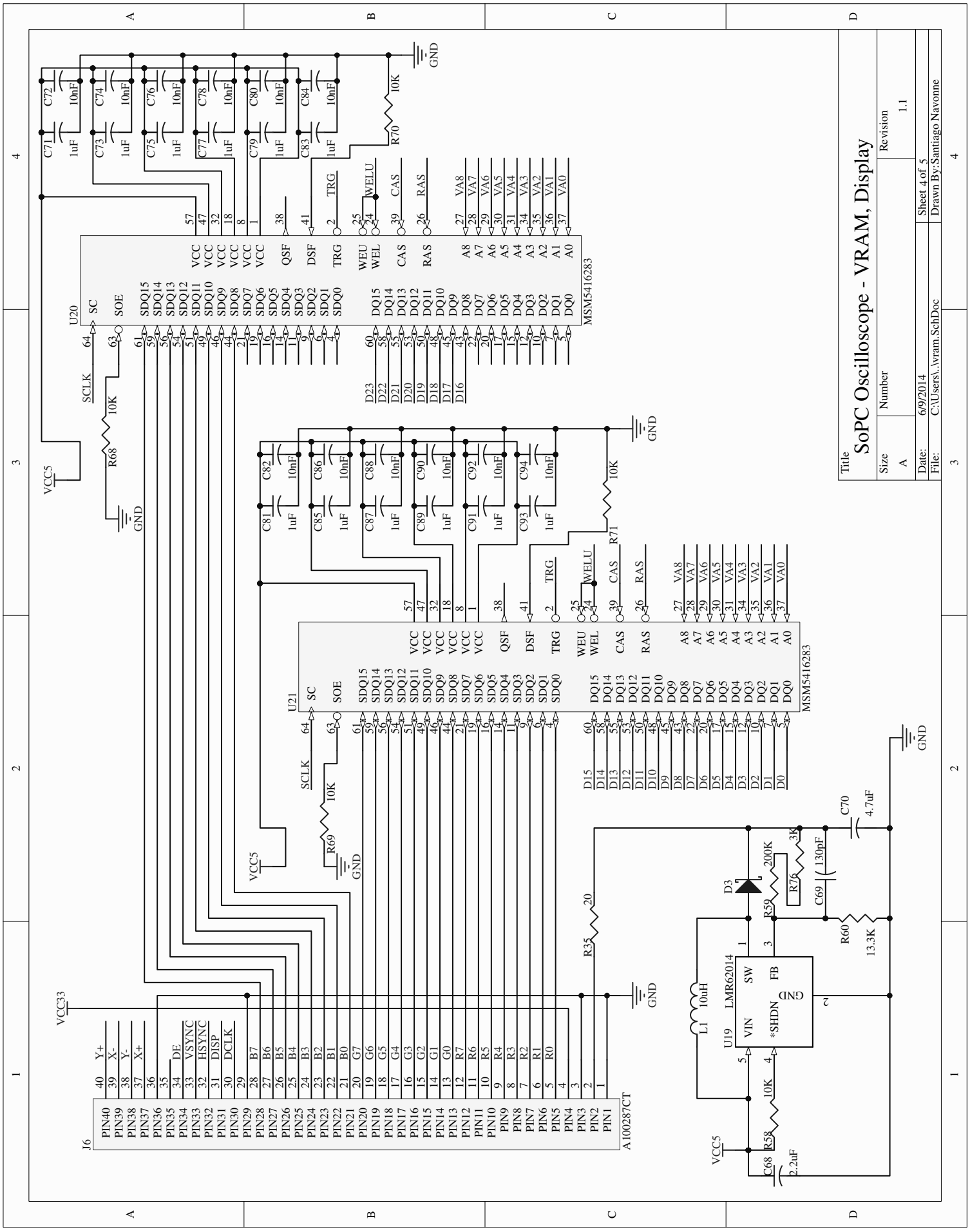


Figure 23: Schematic of the VRAM memory devices and display interface. Also shown is the section of the power supply used to power the display's LEDs. Further details are provided in Section 2.8.3.

2.9 Display

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

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

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

The display connector is highlighted in pink in Figure 2.

2.10 Touch Screen Controller

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

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

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

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

2.11 Rotary Encoders

Two rotary encoders with temporary push-buttons are used to provide the main user input interface for the system. The devices' connections are illustrated in Figure 24. Once debounced and decoded within the FPGA, their signals are made available to the NIOS processor on the *PIO_0* interface, at addresses 0x2410A0-0x2410BF.

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

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

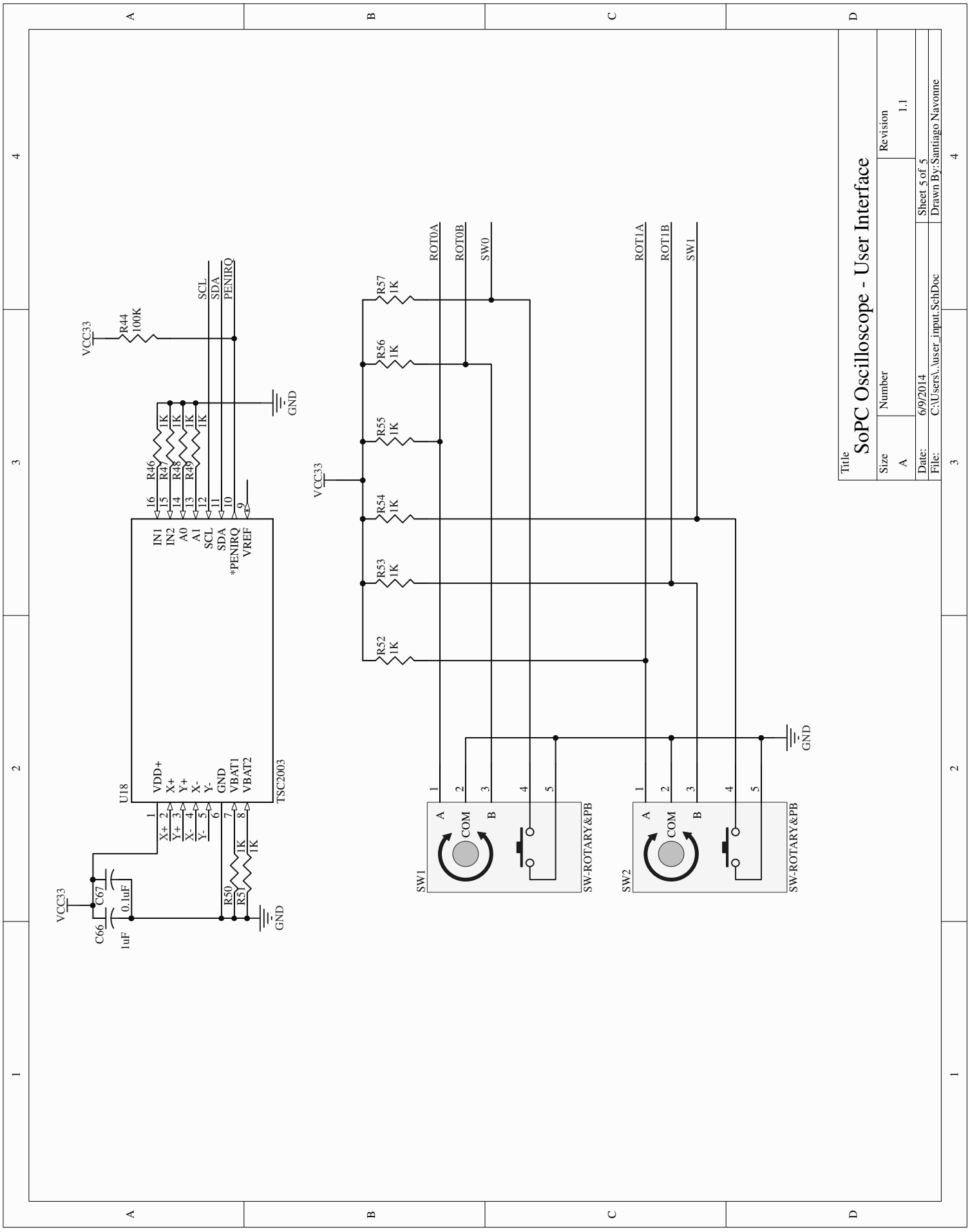


Figure 24: Schematic of the user input section of the system. This includes the rotary encoders, described in Section 2.11, and the touch screen controller, described in Section 2.10.

2.12 Analog Interface

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

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

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

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

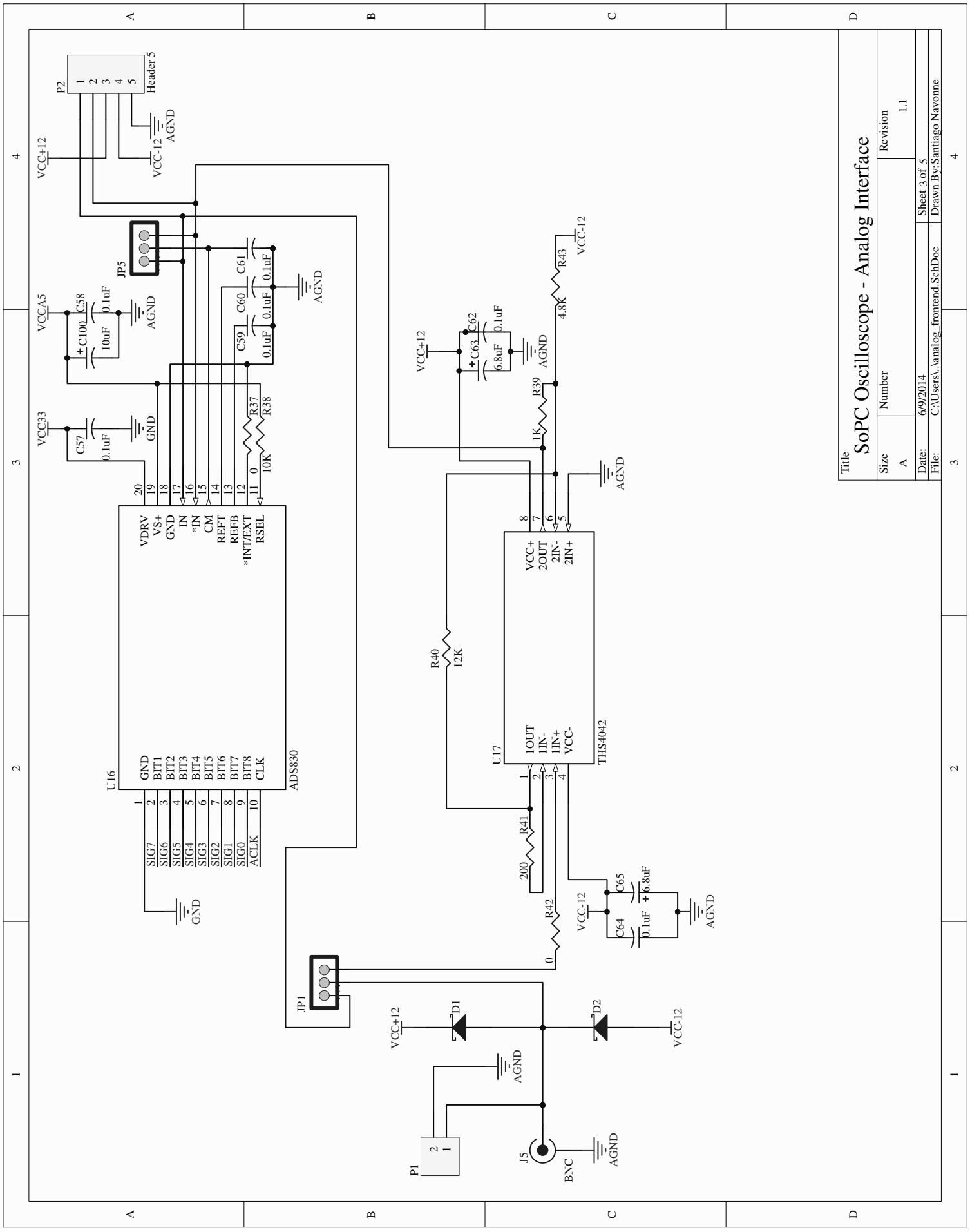


Figure 25: Schematic of the system's analog interface, described in Section 2.12. This includes the analog front-end and the analog-to-digital converter (ADC).

2.12.1 Analog Front-End

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

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

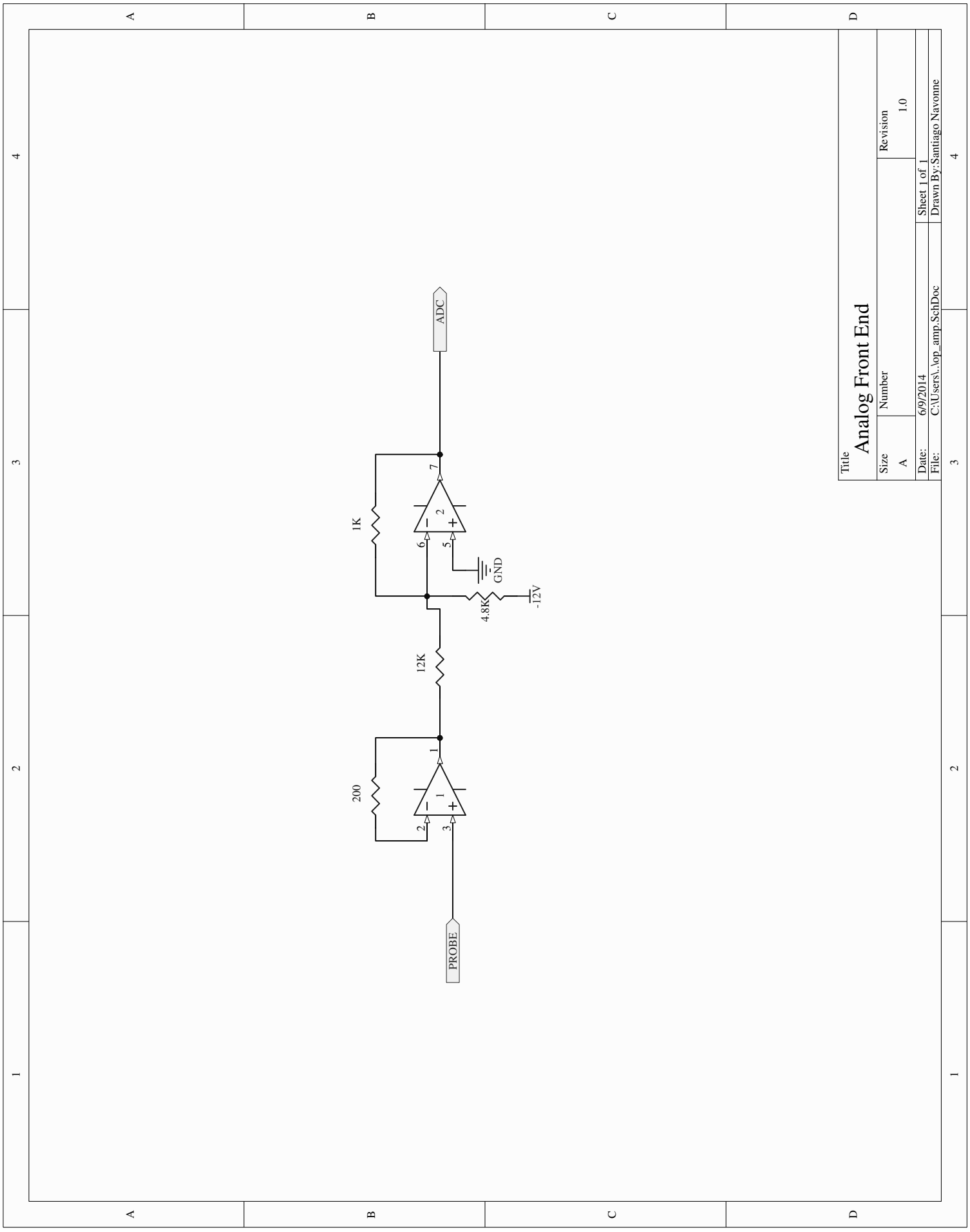


Figure 26: Symbol schematic of the operational amplifier scaling and shifting circuit used in the system's analog front-end, and described in Section 2.12.1.

2.12.2 Analog to Digital Converter

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

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

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

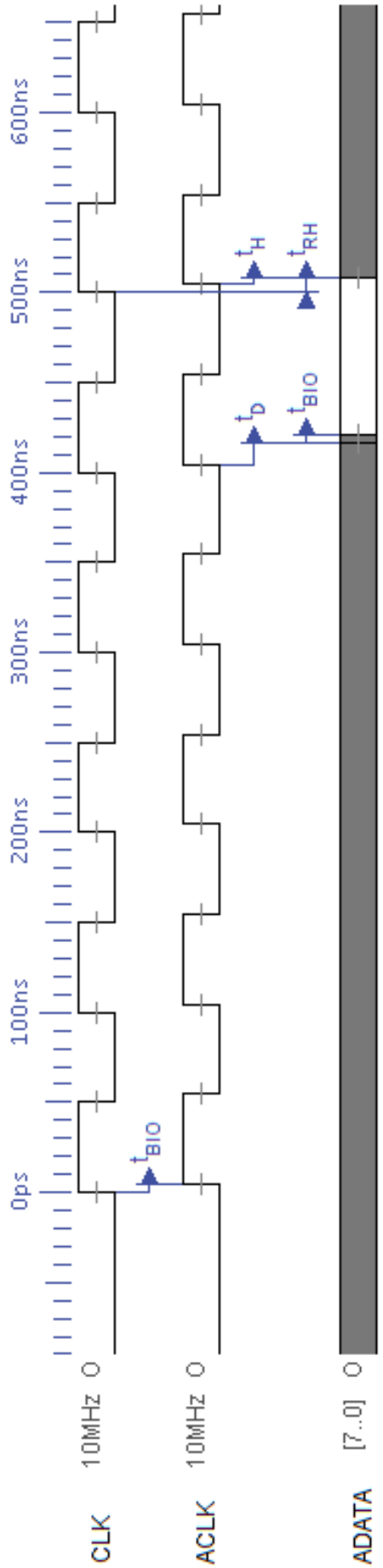


Figure 27: Timing diagram of an ADC sampling clock, described in Section 2.12.2.

General Data

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

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

2.13 Revision History

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

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

Table 11: Revision history.

3 Software

This section describes how the system's software works, from the system overview to the detailed description of each element. The roles and interactions of the various part of the program are described. The code for the program is provided in Appendix 3.4.

3.1 System Overview

TODO

3.2 Block 1 TODO

TODO

3.3 Block 2 TODO

TODO

3.4 ...

TODO

Original Documents Figures **??**, **??**, **??**, **??**, **??** show the schematic of the original design; Figures **??**, **??** show the resulting PCB used in the prototype. Note that the RAM and ROM section is identical to the one described in Section **??**. The changes made and their reasons are summarized below.

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

Additionally, the values of some resistors had to be changed after noticing too big voltage drops across them: the pull-up resistors at the rotary encoders had to be decreased from 10k to 1k, and the pull-down resistor at the ADC had to be shorted.

Finally, a missing pull-up resistor had to be added to the PENIRQ line in order to keep it from floating when not active.

TODO schematic figures (fig:orig_schem_1..5)
TODO PCB figures (fig:orig_pcb_1, fig:orig_pcb_2)

Software Code In this appendix, all the code contained within the program's software is provided. Table **??** shows a quick overview of the various files and their contents for quick reference.

TODO TOC (tab:code_toc)
TODO code @ end