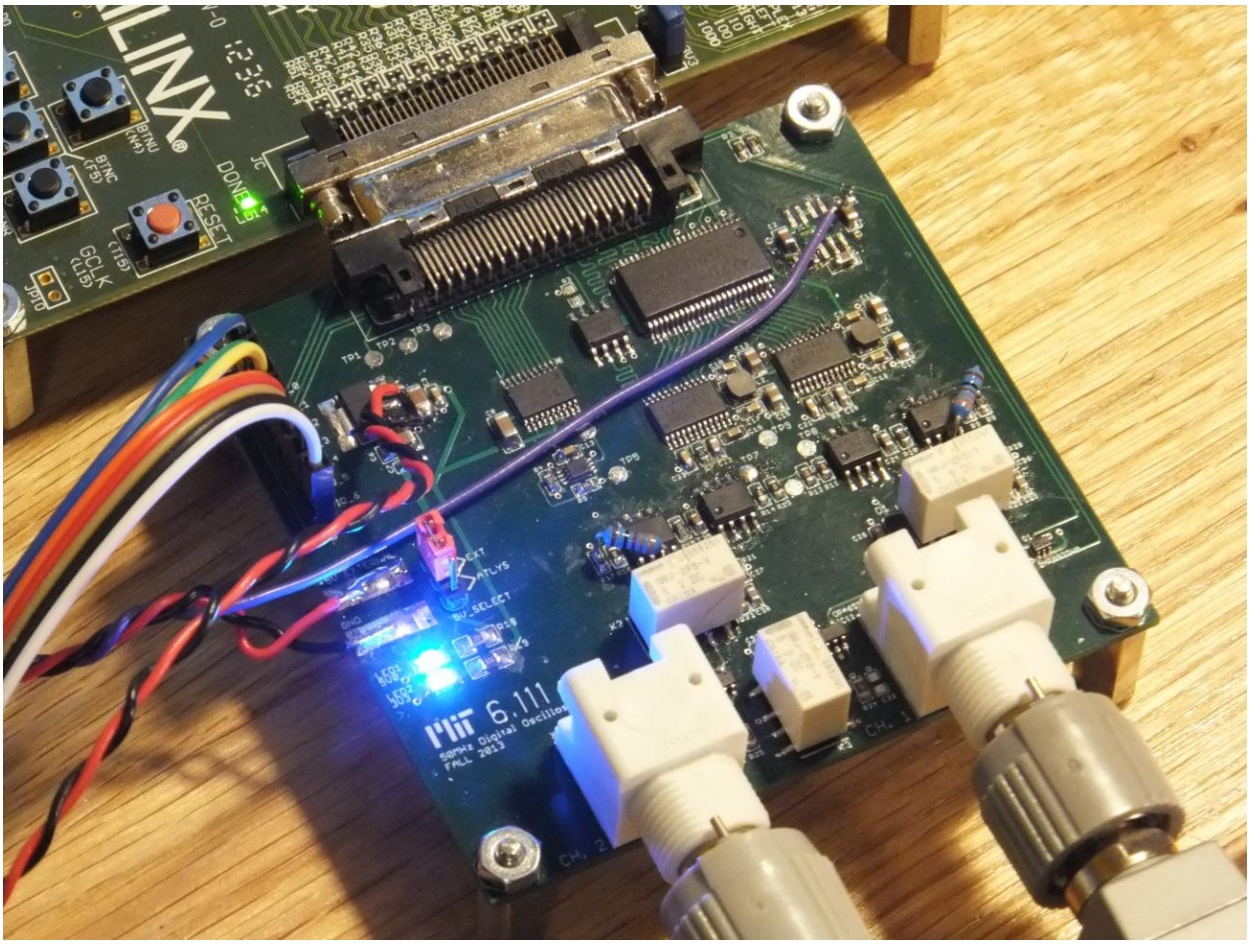
HDMI Digital Oscilloscope

Daniel Kramnik
Fall 2013



Abstract

For my 6.111 final project, I designed and built a two-channel 100$MSPS$ digital oscilloscope with a 25MHz analog bandwidth and a high definition HDMI output that interfaces with a 13.3" WXGA laptop screen. The oscilloscope's analog front end connects to standard x1 and x10 oscilloscope probes and supports multiple vertical scales. The goal of the project was to build a digital oscilloscope with a better display than is commercially available on most test equipment, at a price that rivals that of entry-level oscilloscopes. Future extensions to my work include adding conventional oscilloscope front panel controls using rotary encoders and extending the analog bandwidth to 50MHz by operating the ADCs at 200MSPS and adjusting the anti-aliasing filter.

Table of Contents

1. Introduction

The goal of this project was to implement a digital oscilloscope with an HDMI output that can be connected to a large, high resolution screen superior to what is found in most digital oscilloscopes, or used as a classroom demonstration when connected to a projector. In order to make it a useful instrument, the oscilloscope analog front end connects to standard oscilloscope probes via BNC connectors, is able to support several full-scale input ranges from 100mV full-scale to 50V full-scale, has two channels, and operates at 100MSPS (million samples per second) for a Nyquist rate of 50MHz. The anti-aliasing filter is set conservatively, yielding an analog bandwidth of 25MHz.

Overall, the oscilloscope is comprised of four main stages of signal conditioning and processing, an analog front end, a capture and storage module, a data processing module, and a display module (Fig. 1). The analog front end buffers and scales the analog input signals, converts them to digital signals, and sends them to the FPGA logic board. Within the FPGA, the capture and storage module detects trigger events such as rising edges and stores the inputs from the analog front end into the FPGA's onboard BRAM once the appropriate conditions are met. A data processing module sets the trigger conditions and timing based on the user input buttons and calculates the locations of the pixels in the trace that need to be colored in based on the samples stored in the BRAM. This data is sent to the display module that uses a double buffer also implemented in BRAM to store the trace drawing data and write it to the screen, in addition to the channel and vertical offset text readouts.

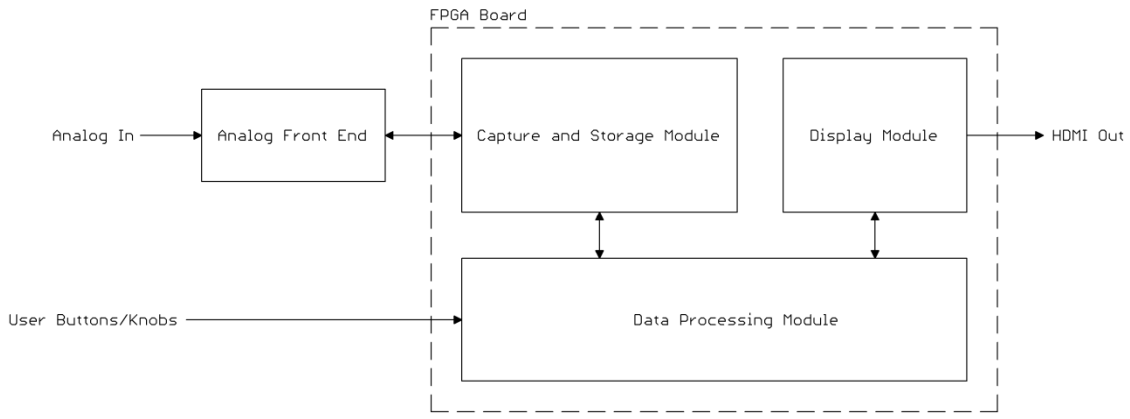


Figure 1: **Block Diagram Showing the Processing Components in the Overall Signal Path of the Oscilloscope.** Analog inputs are converted into digital signals by the analog front end, which then pass through 3 stages of digital processing inside the FPGA that produce an output to the screen. User settings from buttons and knobs are registered in the data processing step.

2. Hardware Implementation

2.1 Hardware Overview

The analog front end digitizes two channels of analog inputs that are probed with oscilloscope probes. In order to do this, the analog front end must buffer the high impedance outputs from the oscilloscope probes, offset and scale the signals so that they fall into the input range of the analog to digital converters, and apply a lowpass filter to prevent aliasing.

Figure 2 is a block diagram of the signal path in a single channel of the oscilloscope. The oscilloscope probe forms a compensated 10:1 (or 1:1, in the case of an x1 probe) voltage divider with the input filter and an AC/DC coupling relay switches in a series capacitor that can be used to AC couple the input signal. Next, an input buffer comprised of a unity gain JFET-input opamp acts as an impedance converter between the high impedance output of the divider and the low impedance required to drive the bipolar-input opamps later in the signal processing chain. A switchable attenuator and preamplifier form a variable amplifier that can be used to set the range of voltages that the oscilloscope is sensitive to; we want to display anywhere from 100mV full-scale to 50V full-scale on the screen. Next, an anti-aliasing filter blocks frequencies below half the Nyquist rate of the ADC. A first order filter is chosen due to its good passband flatness and transient response[1]. It is necessary to put the anti-aliasing filter as close as possible in the signal processing chain to the ADC in order to reduce high frequency noise from any high-bandwidth stages that would otherwise be additively aliased into the digitized signal. Finally, an offset generator adds a DC offset to the signal that puts it into the ADC's input range. A precision DAC with an output current buffer and two voltage amplifiers is used to set the reference voltage of the ADC and the DC offset of the signal. Appendix A contains a complete schematic of the analog front end and Appendix B contains the 4-layer PCB board layout.

[1] http://cp.literature.agilent.com/litweb/pdf/5989-5733EN.pdf

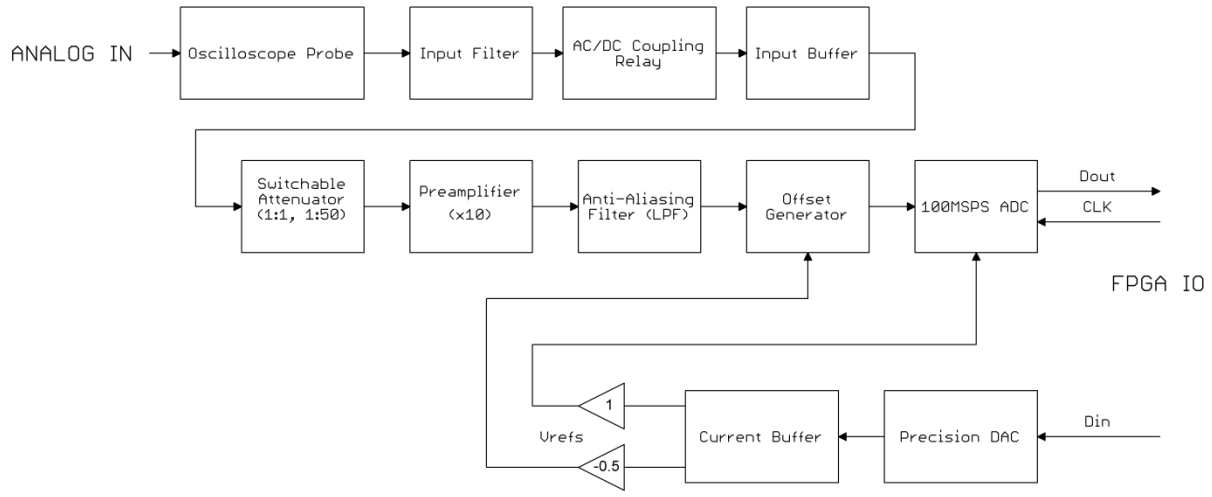


Figure 2: **Block Diagram of the Signal Path in One Channel of the Analog Front End.** An analog input is captured by an oscilloscope probe and passes through a signal conditioning chain that ends with a 100MSPS analog to digital converter that digitizes the signal and sends it to the FPGA logic board.

2.2 Printed Circuit Board Design

In order to establish good signal integrity in the sensitive analog and high speed digital signals, a continuous ground plane that provides the shortest path for return currents is necessary. Although a split ground plane that segregates analog and digital signals can be used, a continuous ground plane with component placement selected to eliminate crosstalk between analog and digital signal domains is preferred because it does not introduce the risk of increasing crosstalk due to long return current paths[2]. In order to accommodate all of the power and signal traces on the board, a 4-layer stackup is necessary. Furthermore, a 4-layer stackup is superior to a 2-layer stackup for routing high speed digital signals that require controlled impedance traces because the outer 2 layers are separated by much thinner material, allowing the microstrip traces to achieve the same characteristic impedance with a much lower trace thickness.

Figure 3 shows the stackup that was used for the ADC sampling board. The top layer is used primarily for analog and high speed digital signal trace, layer 2 is a continuous ground reference plane, layer 3 is used primarily for power supply routes, and the bottom layer is used for low speed and non-critical signal traces, as well as some high speed digital traces that did not fit on the top layer. To achieve a characteristic impedance of 50 ohms between a trace routed on the top or bottom layer and a ground plane on the adjacent inner layer, 0.009”-thick traces are required, assuming the prepreg thickness given in the stackup in Figure 3 and the dielectric parameters given for FR402 in the following table: http://www.4pcb.com/media/laminate-vendors-2.pdf.

[2] http://www.hottconsultants.com/techtips/split-gnd-plane.html

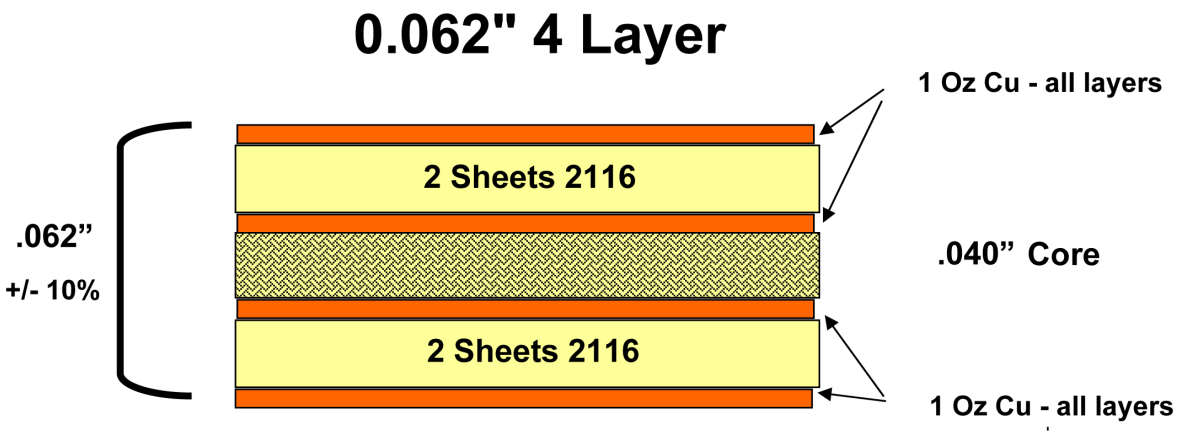


Figure 3: **Advanced Circuits (4PCB) Standard 4-Layer Stackup.** The top layer and layer 2 and the bottom layer and layer 3 are separated by 0.011" of prepreg, while layer 2 and layer 3 are separated by 0.040" of core material. Image source: http://www.4pcb.com/media/-062%204%20layer%206-9-2010.gif

Overall, the circuit board was laid out with digital components near the Digilent VHDCI connector at one end of the board, and analog components near the input BNC connectors at the other end of the board, with the DAC and ADCs in the middle separating the two sections (See Appendix B). A linear power supply (3.3V LDO, U1) was used to supply the ADC and DAC voltage rails, with additional LC filtering added to the ADCs' analog supply rails, while a switched capacitor inverter (IC1) was used to supply the -5V opamp reference rails and the Atlys board's internal 5V buck converter was used to supply the +5V opamp reference rails. Unfortunately, switching noise in these power supplies was high enough to introduce visible extra noise in the oscilloscope traces, and a linear bench supply was used to supply these voltages instead.

The 50-ohm line driver, IC2, which is necessary to buffer the ADCs' digital outputs so that they can drive the 50-ohm terminated traces to the FPGA on the Atlys board, is placed close to the VHDCI connector and requires high speed traces on both the top and bottom side of the board to connect to the VHDCI connector pins. In order to maintain the correct characteristic impedance for the bottom-layer traces, a section of ground plane is added to layer 3. Figure 4 shows the section of the board with this ground plane. Ground stitching vias between the ground plane in layer 2 and the section of ground plane in layer 3 are placed near every point where a signal via crosses through the two ground planes and the signal switches which plane it references. This technique ensures that the return currents under the traces that are routed on the bottom layer have a short path back underneath the traces once they switch reference planes, and that crosstalk between return currents flowing in the two reference planes next to each other is minimized.

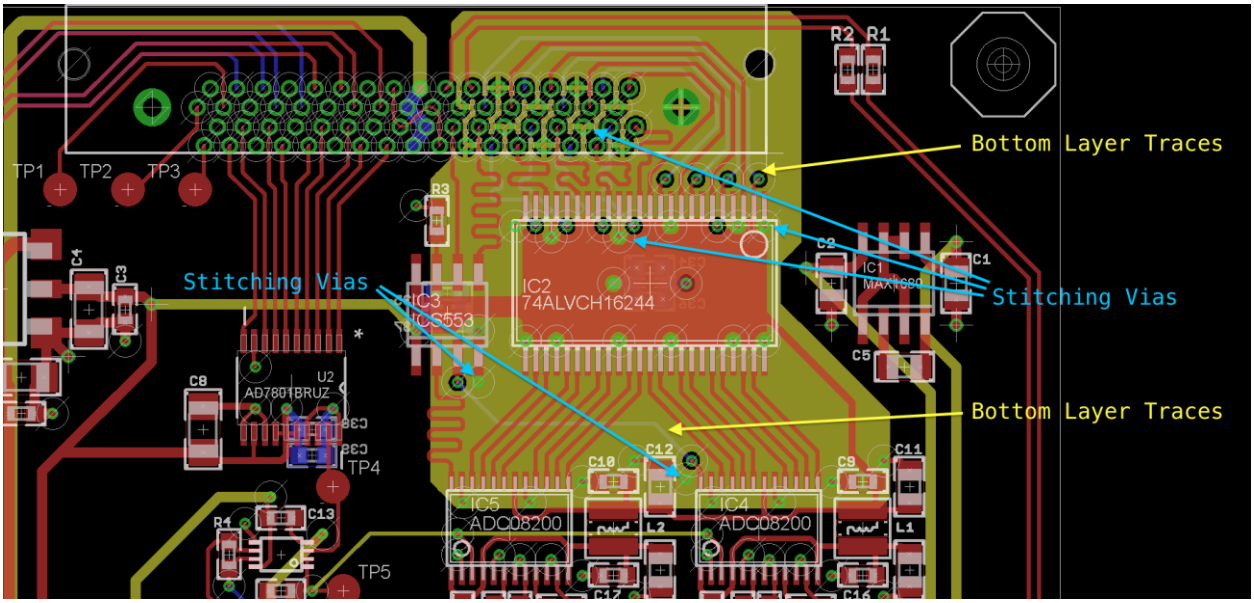


Figure 4: **PCB Artwork Showing the Extra Ground Plane and Ground Stitching Vias.** Stiching vias are placed wherever a trace switches reference planes. The ground connection pins on the VHDCI connector (top left) serve as the stitching vias at that end of the signal path.

In order to make sure that digital signals arrive at the correct time with respect to one another and that parallel busses are read correctly on rising clock edges, some degree of length matching between different signal traces was required. The speed of light (and therefore the speed of the propagation of an electromagnetic signal) in a non-magnetic dielectric material is given by the following equation (μ is the relative permeability and ϵ_r is the relative permittivity):

$$v_p = \frac{c}{\sqrt{\mu * \epsilon_r}}$$

In the FR-4 dielectric from the Advanced Circuits stackup, $\epsilon_r \approx 4.7$, thus we calculate that $v_p \approx 6"/ns$. In other words, for every inch of length mismatch, there is an additional 165ps of signal delay. Operating at a maximum frequency of $100MHz$, or a clock period of $10ns$, the difference made by even several inches of mismatch is not very significant, thus we can approximately match the parallel data traces and clock trace to $\pm 0.5"$ as a good design practice and not worry about the skew. However, the delay introduced by sending the clock signal to the ADCs from the FPGA and passing it through the clock buffer can be significant and as high as half a clock cycle (or even an entire clock cycle, if we someday decide to operate the ADCs at their maximum rate of $200MHz$). Therefore, one output of the clock buffer is sent back to the FPGA board so that sampling of the data bus can occur at the correct time. Figure 5 shows the section of the board that contains the clock buffer and length matched traces.

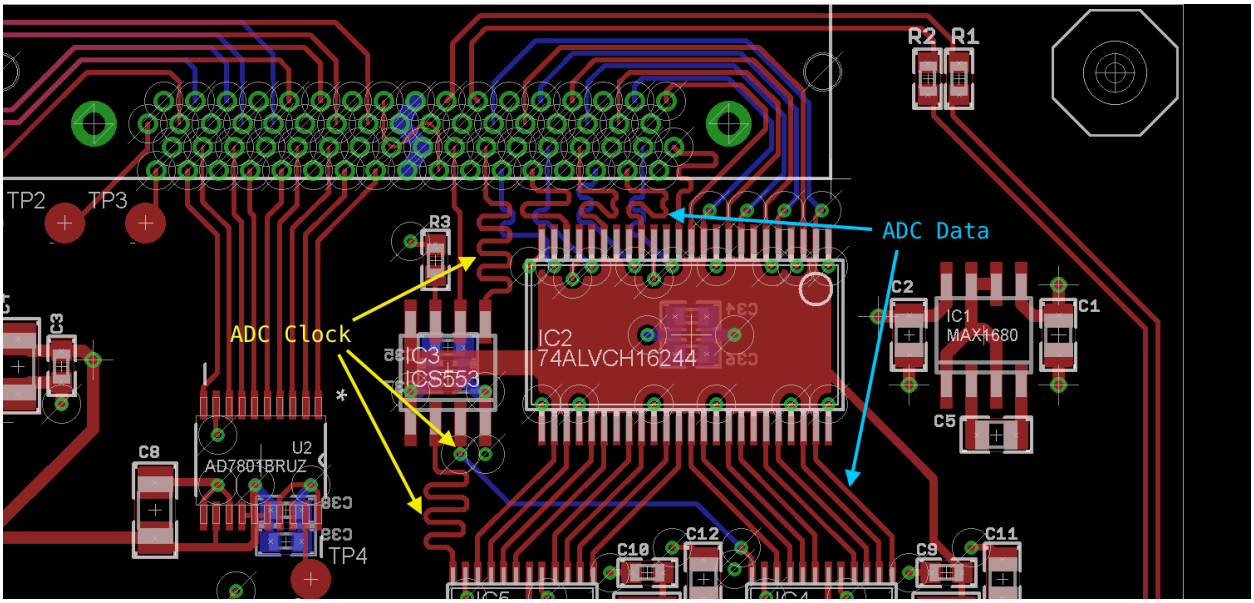


Figure 5: **PCB Artwork Showing the Clock Buffer and Length Matched Traces.** ADC data lines are matched to ± 0.5" and are not critical. The clock lines are precisely matched so that the ADCs sample concurrently and the buffered clock signal is sent back to the FPGA board so that the data bus is read at the correct time.

2.3 Assembly, Testing, and Debugging

The surface mount components on the board were soldered using solder paste and a heat gun, and components were added in sections and tested and an extra board was ordered in the event that irreparable damage was caused to the first board, such as a component pad overheating and falling off. While the digital sections of the board were brought up smoothly, there were issues with the analog section that required some extra deadbugged components to fix. Additionally, the switching power supplies were too noisy and an external linear supply had to be used to supply the reference voltages for the analog section.

One of the main problems encountered with the analog section was noise and oscillations in the high speed opamps. On one of the opamps in the preamplifier, for example, there was $80mV$ of offset between the two inputs and large amounts of high frequency oscillations superimposed on the input signal. The source of the offset voltage was improper decoupling; on the board, the opamps were only decoupled between $+5V$ and $-5V$, but not to the ground plane that the oscilloscope probe input is referenced to. Thus, large offset voltages were able to build up due to the stray inductance and resistance in the long return path the ground. Adding decoupling capacitors from $+5V$ and $-5V$ to the ground plane eliminated the offset voltage problem and reduced the oscillations. Figure 6 shows the extra decoupling capacitors deadbugged on the bottom side of the board.

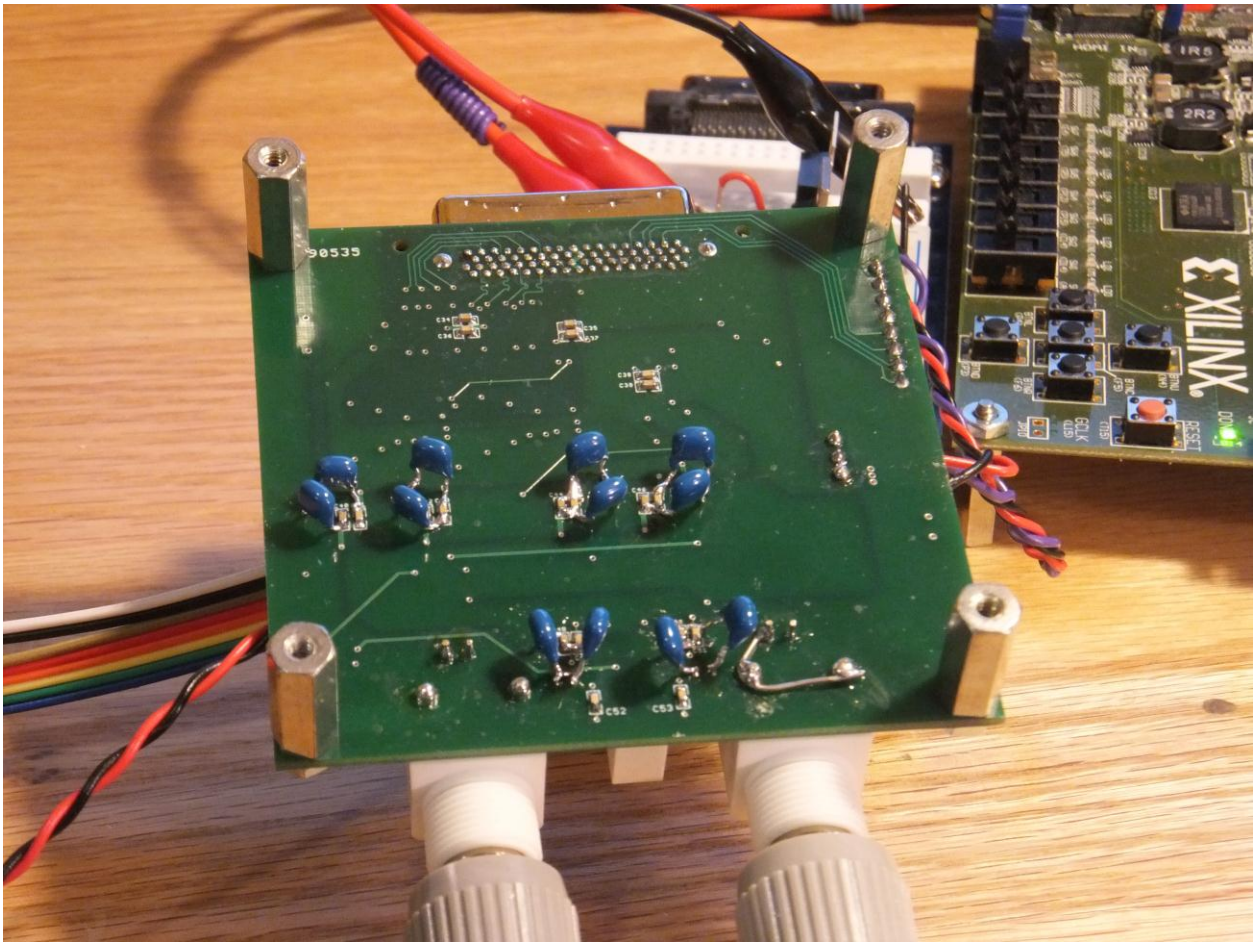


Figure 6: **Extra Decoupling Capacitors Soldered to the Bottom of the Board.** The high speed opamps were only decoupled between their supply rails, but not to the ground plane. This resulted in large offset voltage building up across the parasitic inductance and resistance in the ground return path, as well as increased noise.

An additional step that helped to reduce the oscillations in the opamps was to add input 330Ω pulldown resistors. These help to reduce the effect of stray capacitances that can couple the output back into the input of the opamps and create positive feedback. Figure 7 shows the completed ADC sampling board from the top with the pulldown resistors deadbugged across the affected opamps.

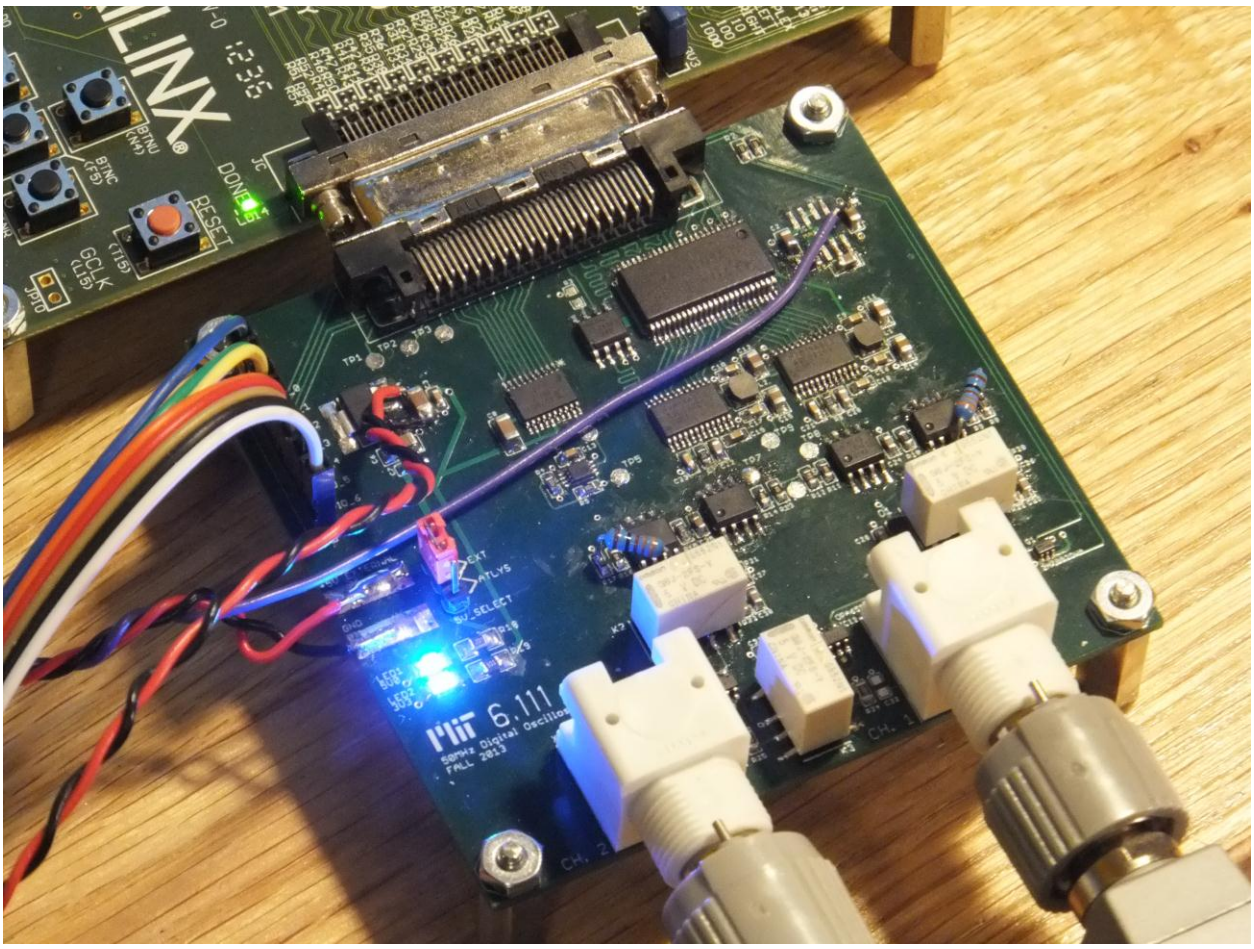


Figure 7: **Input Pulldown Resistors Soldered to the Top of the Board.** In order to reduce the effect of parasitic capacitances between the output and input of the opamps that were creating positive feedback and inducing oscillations, input pulldown resistors were added to the affected opamps. Additionally, the switching regulator (top right) for the negative supply rail was removed and replaced with a linear supply to reduce noise.

3. Software Implementation

3.1 Top Level Controller (top_block_controller.v)

Description: The top level controller instantiates instances of the lower-level modules and connects them together. Additionally, the top level controller is responsible for handling clock and reset generation, switch debouncing, and interfacing directly with the signals from the VHDCI connector and connecting them appropriately to the lower-level modules.

Implementation: The main subtleties in the top level controller lie in the clock generation. The Xilinx example HDMI transmitter code[3] contains an error in the clock division section and fails to compile. Instead of using a BUFIO2 to divide the $100MHz$ system clock (`sysclk`) by 2, which is a disallowed mode of operation for BUFIO2 due to the possibility of a glitch that results in a stuck state[4], a digital clock manager (DCM_CLKGEN) was used to directly divide down the system clock to the $85MHz$ pixel clock (`pclk`). Next, a PLL_BASE instance is used to generate the pixel clock multiplied by 2 and 10. This section is identical to the Xilinx example code.

Without extra buffering, the pixel clock is loaded down too much by the various BRAMs, data processing steps, and finite state machines that it runs, so a buffer tree made using BUFG instances is used to generate three extra pixel clock signals (`pclk_fsm_1`, `pclk_fsm_2`, `pclk_data`). Extra buffering is recommended when adding more modules to the design. Unlike the labkit's Virtex 2 FPGA, the Spartan 6 FPGA on the Digilent Atlys board is faster, but more sensitive to loading and timing issues on the clock signals. It is also necessary to buffer the input CMOS clock and the ADC return clock from the sampling board using IBUF instances.

3.2 Capture and Storage Block (capture_and_storage_block.v)

Description: The capture and storage block contains a $2K\ x\ 16bit$ BRAM for storing 8-bit samples from both channels. When a rising edge across the trigger threshold set by the data processing block is detected, the sample BRAM is filled, and then read back to the data processing module once the data is requested. The trigger flag is reset on the `vsync` pulse from the display controller so that triggering only happens once per frame.

Implementation: On the rising edge of `vsync`, the `triggered` and `sampling` flags is reset and the capture and storage module begins searching for a rising edge trigger event. Such an event occurs when the previous sample from the trigger channel was equal to the trigger threshold, and the current sample has surpassed the trigger threshold. The `triggered` and `sampling` flags are set, and so long as the `sampling` flag is set, the write address to the sample BRAM is incremented and the write enable signal is asserted. Once the BRAM has been filled, the `sampling` flag is reset, write enable is disasserted, and the BUFGMUX clock multiplexer switches the sample BRAM's clock from the $100MHz$ ADC

[3] http://www.xilinx.com/support/documentation/application_notes/xapp495_S6TMDS_Video_Interface.pdf
[4] http://www.xilinx.com/support/answers/56113.html

clock to the $85MHz$ pixel clock used by the data processing finite state machine. The $85MHz$ pixel clock is chosen for the clock of the data processing module because eventually it will need to write to the BRAM in the display controller, which operates at the pixel clock. A `done_sampling` signal (simply the inverse of the `sampling` flag) is sent to the data processing module to indicate when it has control of the sample BRAM and can read back data from it. The BRAM's address lines are multiplexed between the capture and storage module's sample write address and the data processing module's sample read address based on the `sampling` flag.

It is necessary to have both a `triggered` and `sampling` flag in order to make sure that a trigger event only initiates sampling into the BRAM once per frame, and to transfer control of the sample BRAM between the capture and storage module and the data processing module. Both of these functions cannot be done with only one flag.

3.3 Data Processing (data_processing_block.v)

Description: The data processing block processes the raw ADC data from the capture and storage block once triggering and sampling have occurred and writes the properly scaled and offset trace pixel values into the display controller block's frame BRAM. The data processing block also contains finite state machines that control the trigger level, vertical offset level, and vertical scale level based on user input from pushbuttons.

Implementation: The vertical offset finite state machine increments or decrements the value of the `v_offset` register based on whether the user is holding down the offset increment or offset decrement button (see Figure 8). In order to produce a smoother user interface, holding down the buttons results in a change, instead of each button press resulting in a change. This is implemented by polling the value of the buttons on every rising edge of `vsync`, which occurs at the 60Hz refresh rate of the screen.

In a similar manner to the vertical offset finite state machine, the trigger level finite state machine increments or decrements the trigger level based on whether the user is holding down the trigger level increment or decrement button. The one added feature in the trigger level finite state machine is that we would like the trigger level to remain on the screen for about a second after the user has released the increment and decrement buttons, like in a real oscilloscope. This functionality is accomplished by asserting the `draw_trig_lvl` signal that is sent to the display controller while the user is pressing the buttons, and then holding it high for `TRIG_HOLD_FRAMES` number of frames after the user has let go of the buttons. The `trig_hold_counter` is responsible for keeping track of the number of vsync pulses that have arrived since the user let go of the buttons. It is reset and the countdown restarts every time the user presses a button.

The volts per division finite state machine responds to user button presses on the V/div increment and decrement buttons by changing the ADC reference voltage using the reference DAC on the sampling board, and switching on or off the 1:50 switchable relay attenuator. The state of this finite state machine is sent to the display controller so that the volts per division value can be displayed on the screen. This is not currently implemented,

however, due to timing issues associated with drawing large amounts of text on the screen simultaneously.

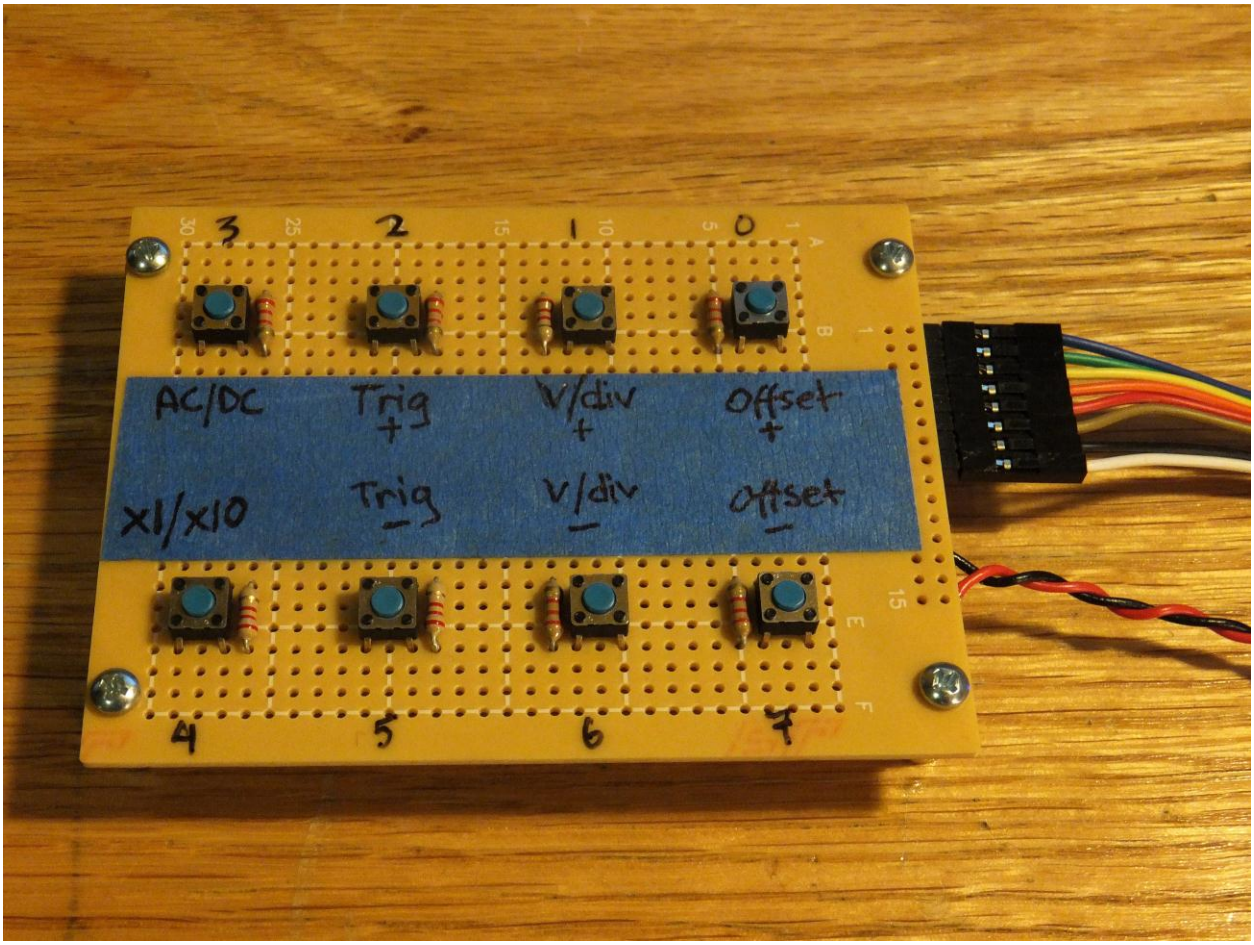


Figure 8: **User Interface Board with Pushbuttons.** Due to the time constraints of the project, a simple pushbutton user interface was constructed instead of the originally-planned front panel with rotary encoders and knobs. Veritcal offset, vertical scaling, and trigger level can be adjusted. The trigger channel is selected using the onboard slide switch on the Atlys board. The pushbutton board plugs into the GPIO pins on the front end board using a ribbon cable.

The final piece of the data processing block is the pixel calculator that reads data from the sample BRAM in the capture and storage block, shifts and scales it, and then writes it to the frame BRAM in the display controller block. When the `done_sampling` signal is asserted by the capture and storage block, indicating that it has triggered, filled the sample BRAM, and passed control of the sample BRAM to the data processing block, the pixel calculator scales, offsets, and then stores the samples to the frame BRAM using a 3-stage pipeline. Due to timing glitches when running at full speed (`pclk`), a 3-bit counter is used to slow down the read and write speed of the pixel calculator to 1/8th of the pixel clock. Once the frame BRAM has been filled, the data processing module resets the appropriate counters and flags and prepares for the next `done_sampling` event from the capture and storage block that occurs in the next frame.

3.4 Display Controller (display_controller_block.v)

Description: The display controller displays the trace pixel data, vertical offset, trigger level, and trigger channel to the output screen over HDMI.

Implementation: The HDMI/DVI serialization component of the display controller is taken from the previously mentioned Xilinx HDMI application note, with the clocking modified in the top level controller and custom timing parameters set up for the 1366x768 WXGA resolution of the OEM Dell laptop LCD screen that was used as the display of the oscilloscope. The Xilinx example code takes standard VGA timing parameters such as hcount, vcount, and pixel data as input, so the display controller works similarly to the lab 3 pong game, where all of the items on the screen are sprites that are determined by the hcount and vcount values, and combined using some combinatorial logic.

The trace is drawn by comparing the vcount value to the data stored in the frame BRAM that is currently being read. A double buffered frame BRAM switches between two BRAM modules, one is read by the display controller while the other is written to by the data processing module. The address of the BRAM that is read by the display controller is simply hcount. Figure 9 shows the output screen with a square wave signal on Channel 1.

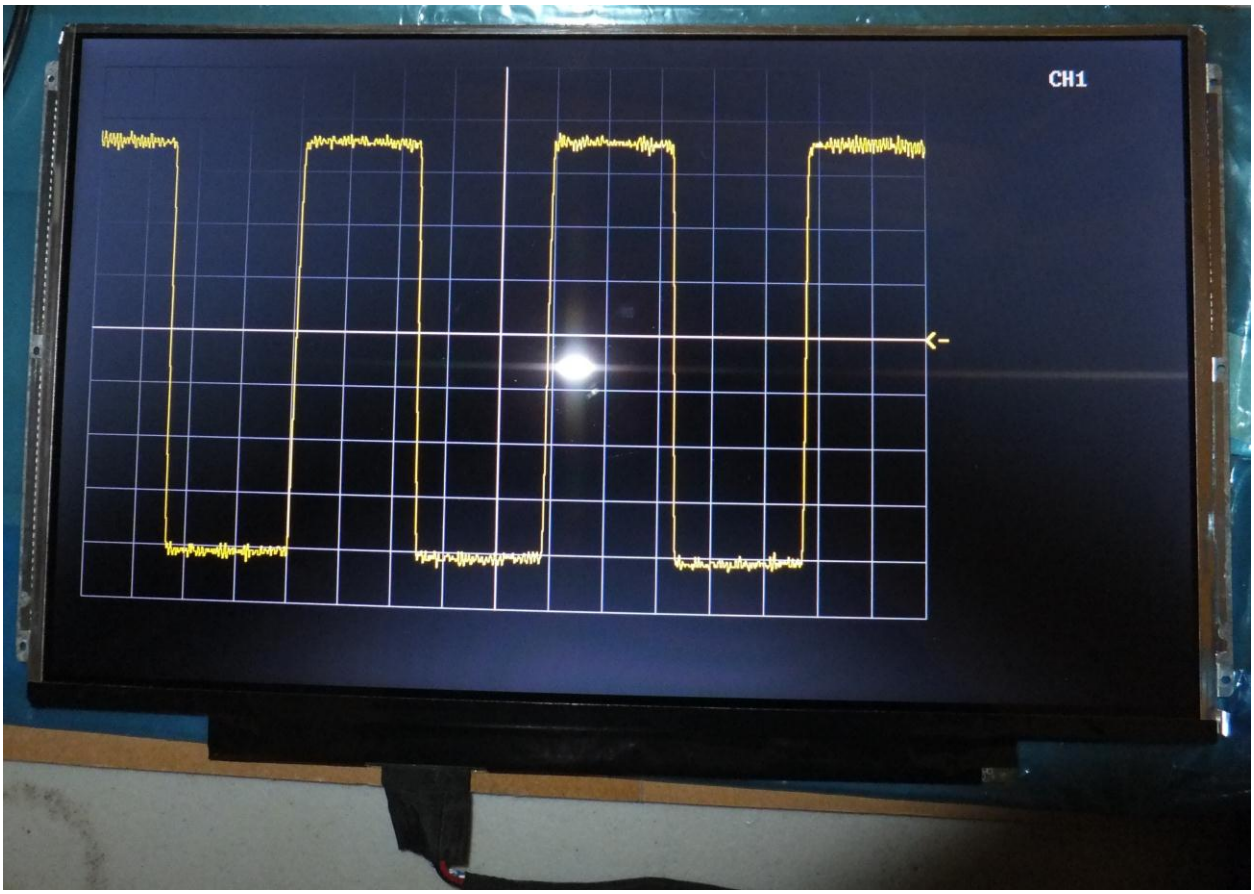


Figure 9: **Output Screen Display with Channel 1.** The output screen contains the trace superimposed on the graticule, a vertical offset marker, and the channel readout.

The vertical offset pointer and channel readout are generated using the `char_string_display` module from the 6.111 tools. The channel readout text and trace color are set using the `trig_ch_select` signal from the data processing block, and the vertical position of the vertical offset pointer is set by the `v_offset` signal from the data processing block.

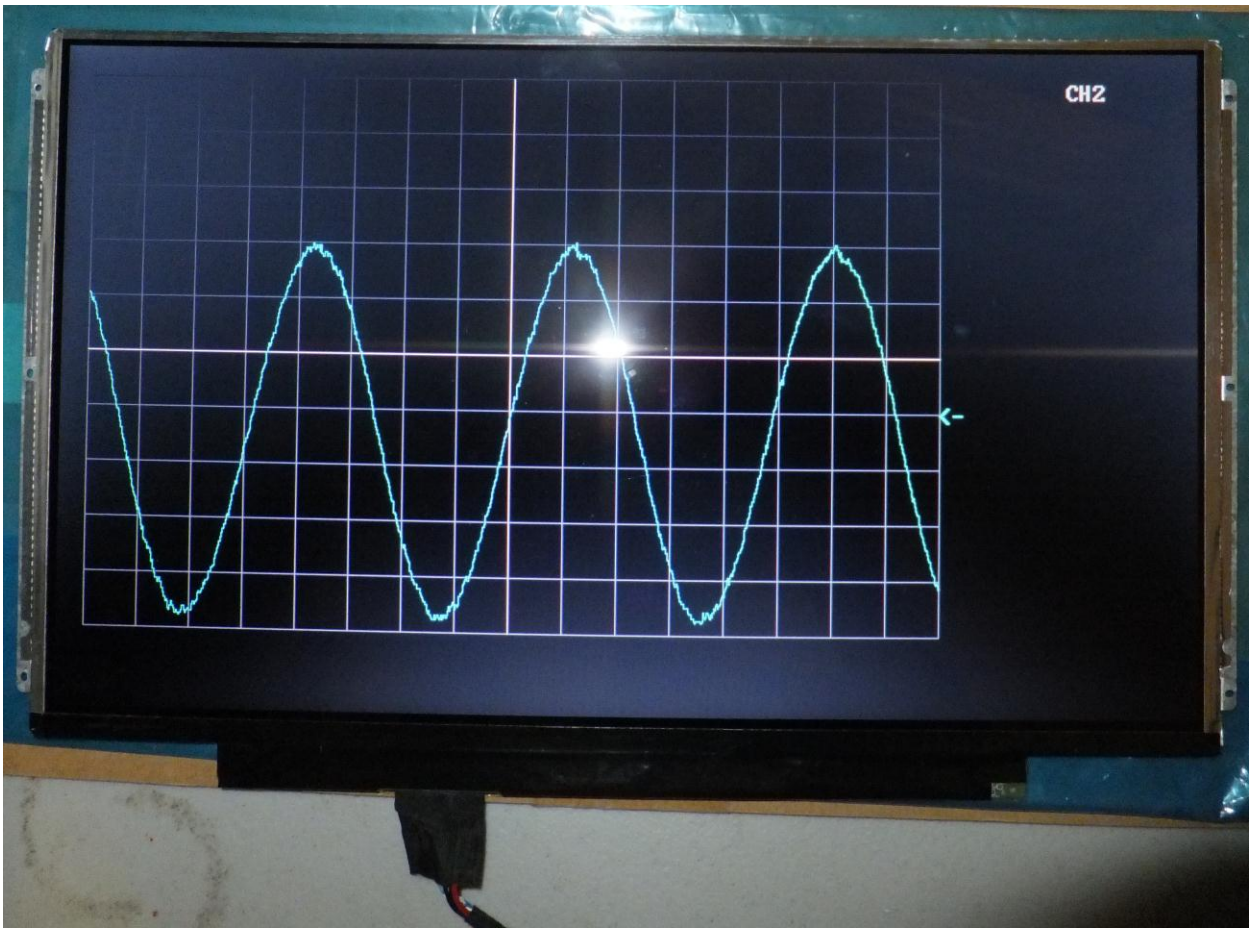


Figure 10: **Output Screen Display with Channel 2.** In this case, Channel 2 has a negative offset, as indicated by the marker on the side.

4. Conclusion and Future Work

Overall, the project achieved the goal of implementing the basic functionality of a relatively high speed digital oscilloscope with an HDMI display. Future work will be focused on adding more user interface features to the oscilloscope such as a front panel with rotary encoders for user input, additional onscreen readouts and measurements, simultaneous display of multiple channels, and horizontal scaling by downsampling and filtering the ADC data. Additionally, assuming that the timing of the FPGA can be improved to meet the higher clock frequency, the ADC sampling board can be operated as high as 200MSPS to double the analog bandwidth to 50MHz. Future work will be documented on the Techfair 2013 project blog linked below. This project would not have been possible without the funding and support of MIT's Techfair!

http://techfair.mit.edu/sr-blog/archives/category/student-projects/2013-2014/a-better-digital-oscilloscope

Appendix A: ADC Sampling Board Schematic

6.111 ADC SAMPLING BOARD

Daniel Kramnik

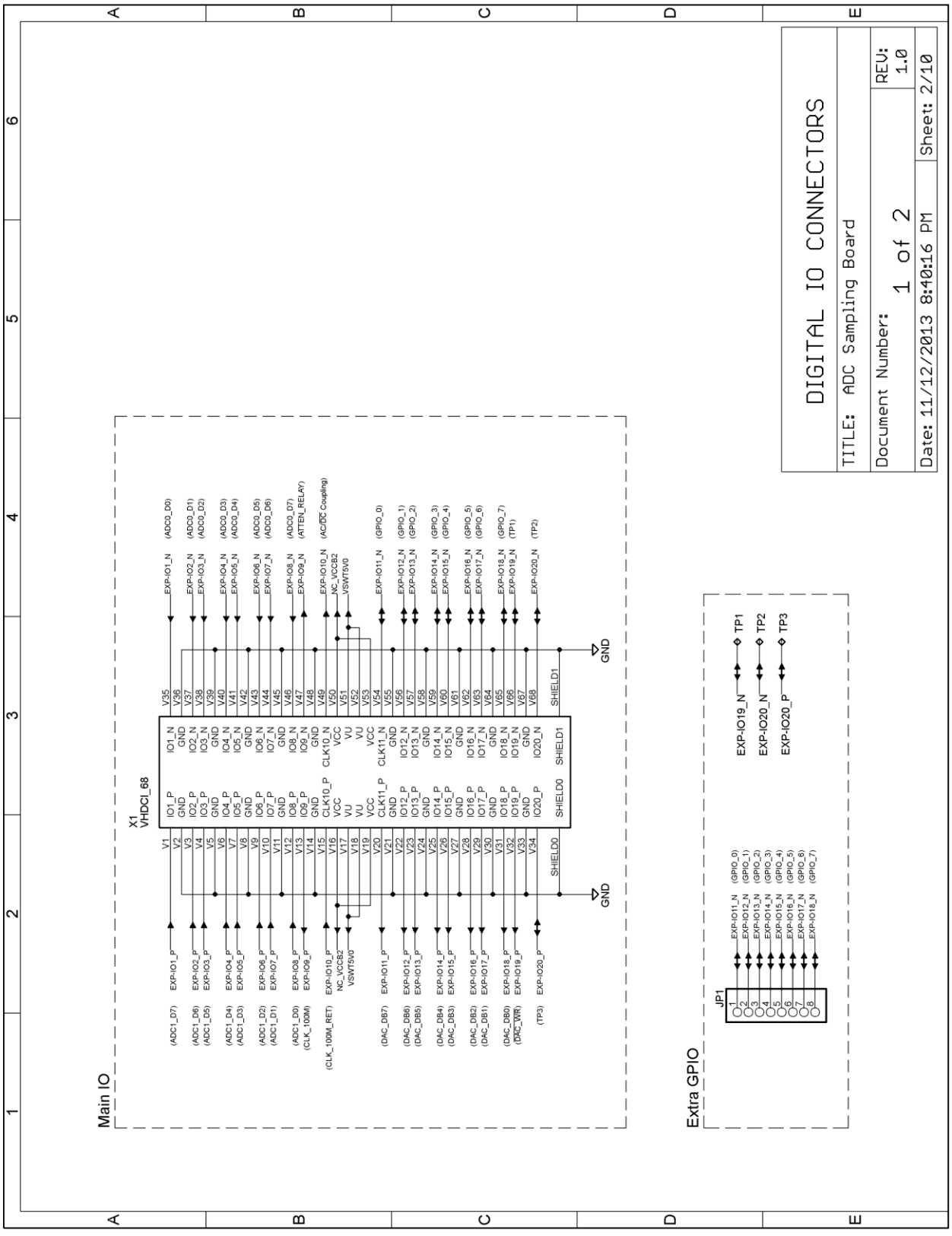
Routing Notes:

ADC DATA/CLK MAX LUMPED LENGTH: 3.5in.

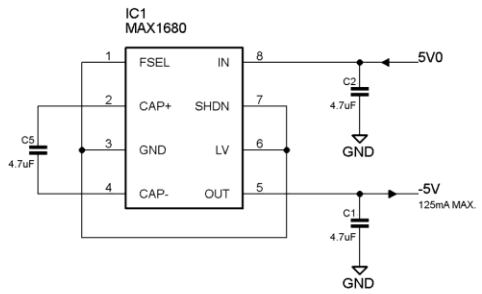
IMPEDANCE CALCULATIONS:
(T = 1oz/ft^2, H = 9.58mil, Er = 4.6)

8mil = 52.3 Ohm SE
9mil = 50.3 Ohm SE
10mil = 48.5 Ohm SE

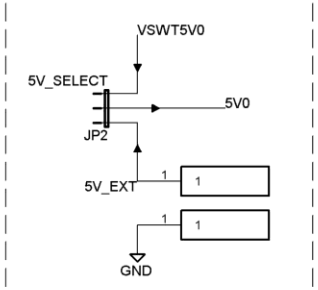
TABLE OF CONTENTS	
TITLE: ADC Sampling Board	
Document Number: 1 of 2	REV: 1.0
Date: 11/12/2013 8:43:26 PM	Sheet: 1/10


DIGITAL IO CONNECTORS
TITLE: ADC Sampling Board
Document Number: 1 of 2
REV: 1.0
Date: 11/12/2013 8:40:16 PM
Sheet: 2/10
Main IO
Extra GPIO
X1
VHDCI_68
JP1
GND
SHIELD0
SHIELD1
TP1
TP2
TP3

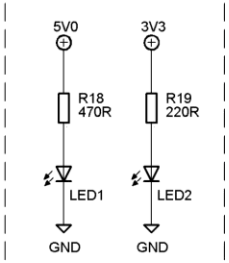
-5V Power Supply (Switched Capacitor, Inverting)



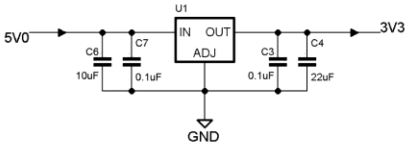
Input Power Select Jumper



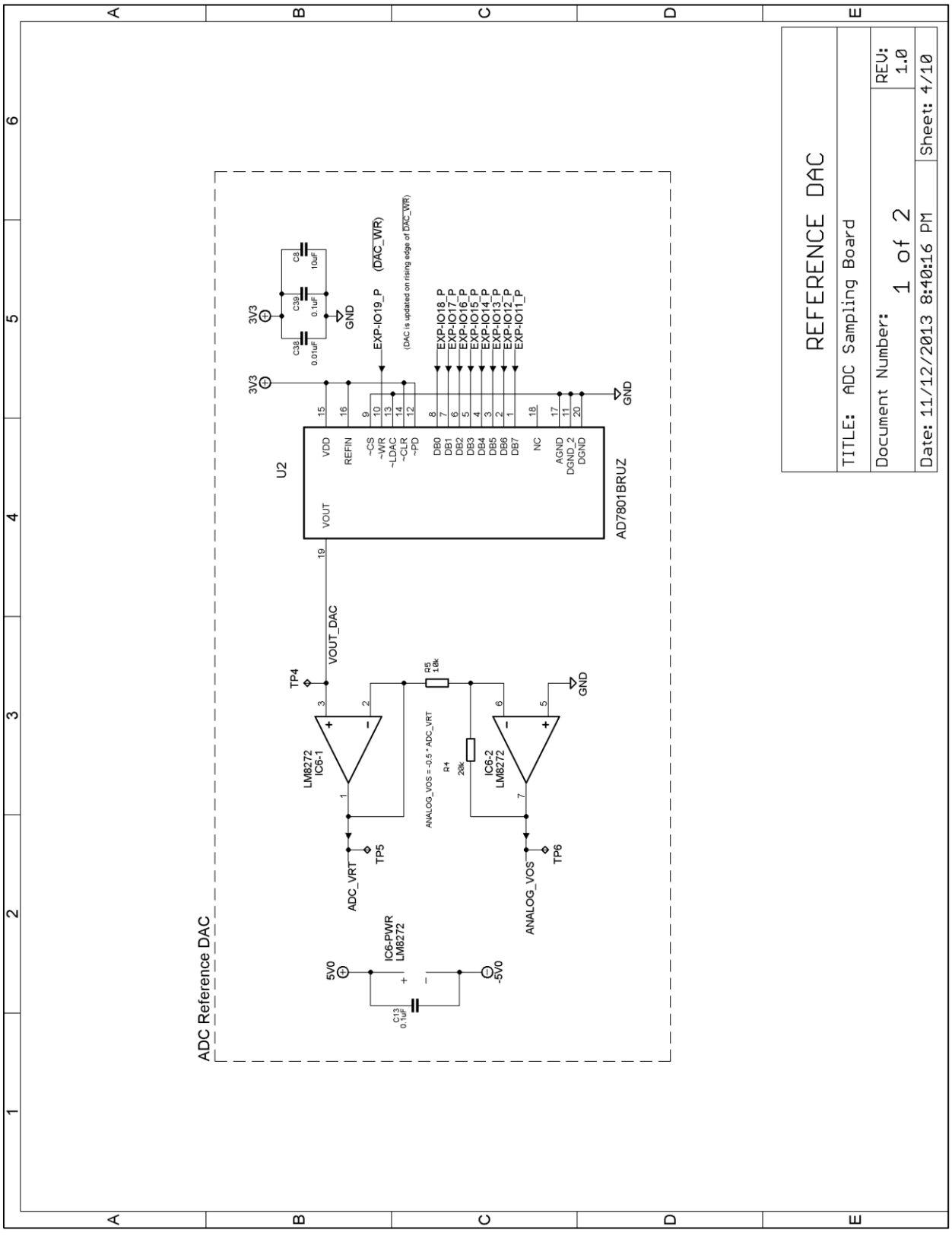
Power LEDs



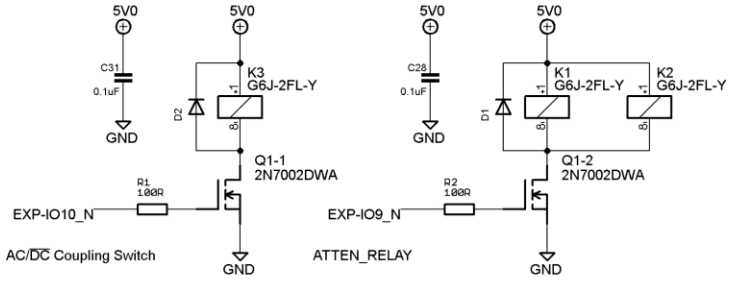
3.3V Power Supply (Linear)



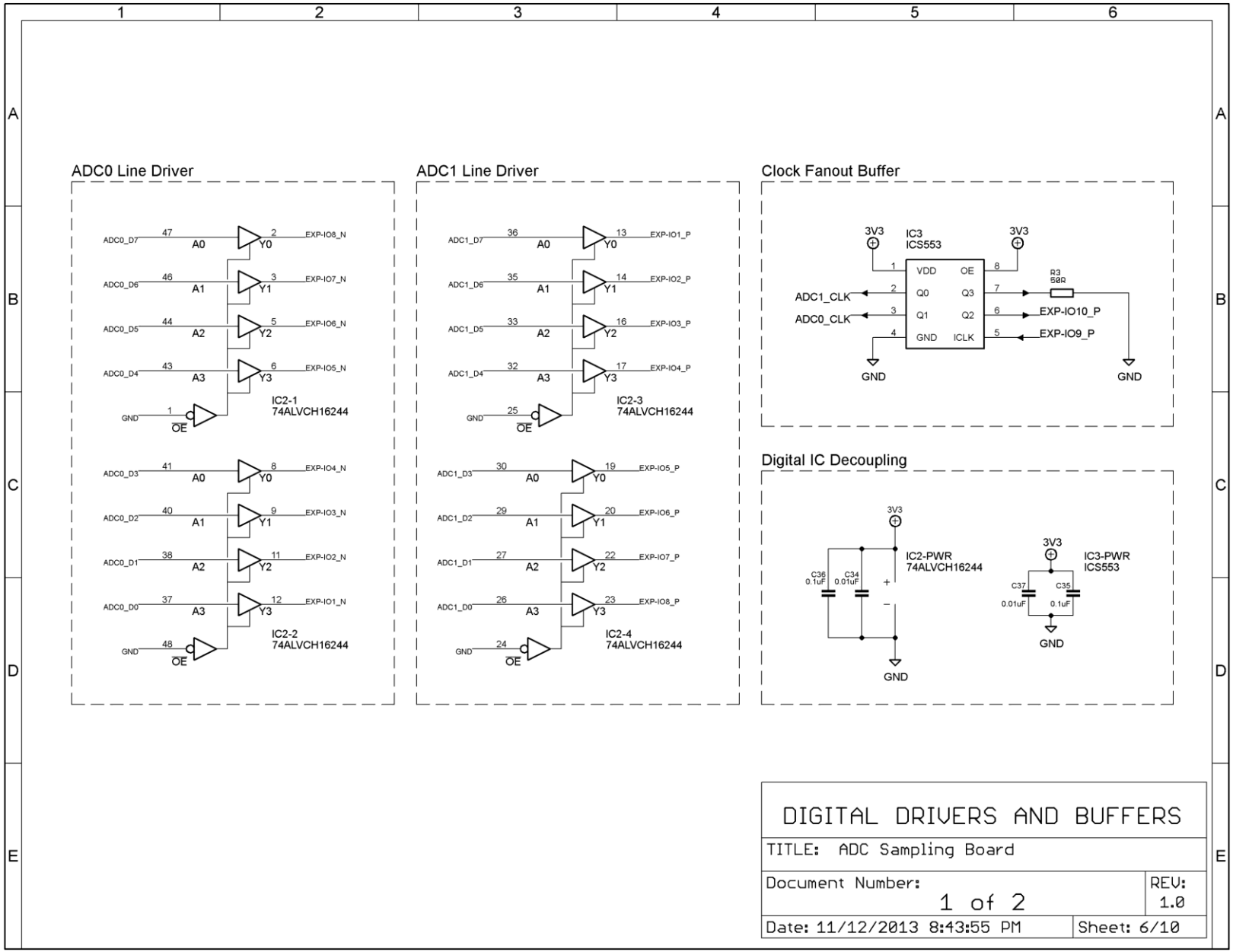
POWER SUPPLIES		
TITLE: ADC Sampling Board		
Document Number: 1 of 2		REV: 1.0
Date: 11/12/2013 8:40:16 PM	Sheet: 3/10	

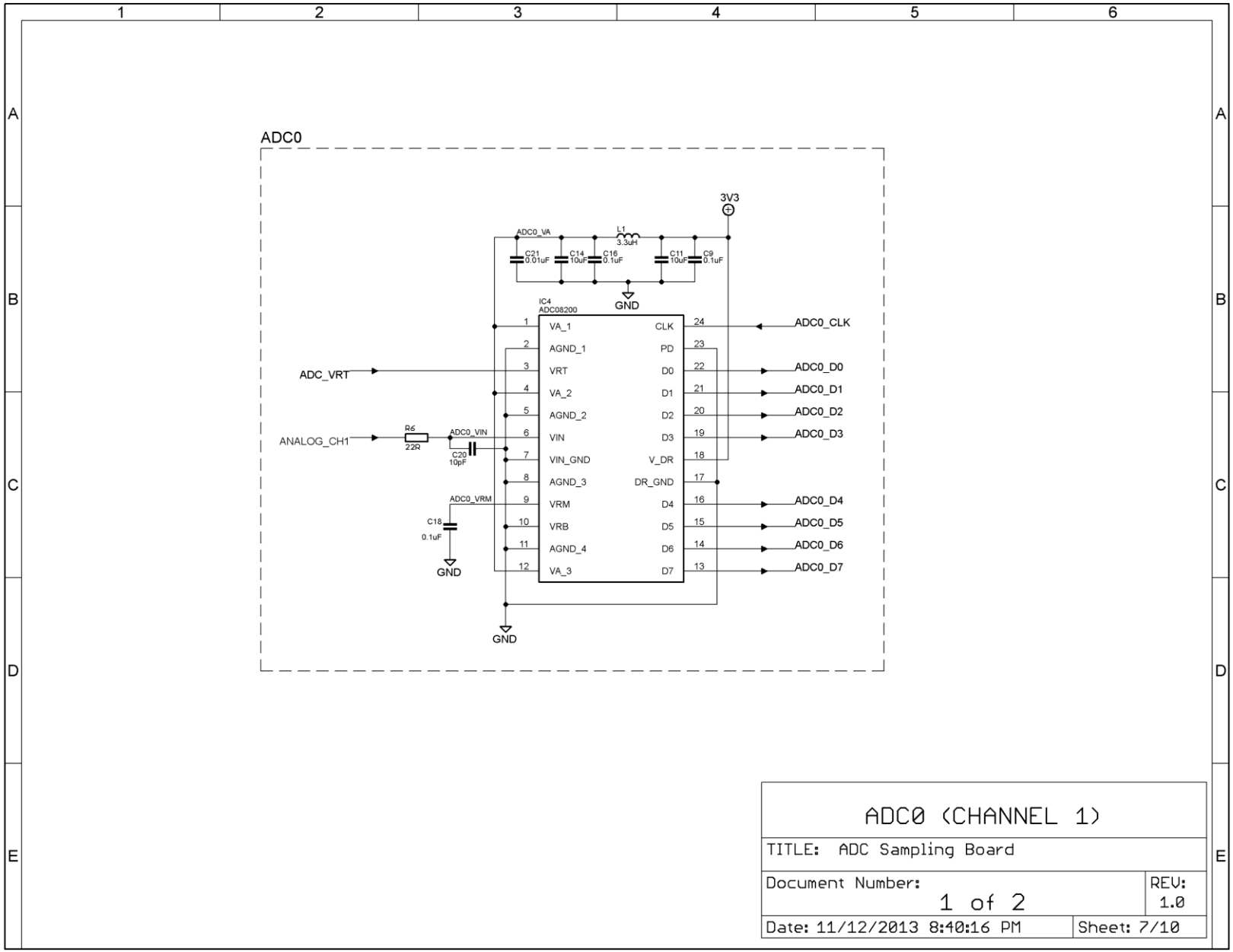

REFERENCE DAC
TITLE: ADC Sampling Board
Document Number: 1 of 2
REV: 1.0
Date: 11/12/2013 8:40:16 PM
Sheet: 4/10
ADC Reference DAC
U2
AD7801BRUZ
VDD
REFIN
VOUT
~CS
~WR
~LDAC
~CLR
~PD
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
NC
AGND
DGND_2
DGND
3V3
GND
EXP-IO19_P
(DAC_WR)
(DAC is updated on rising edge of DAC_WR)
EXP-IO18_P
EXP-IO17_P
EXP-IO16_P
EXP-IO15_P
EXP-IO14_P
EXP-IO13_P
EXP-IO12_P
EXP-IO11_P
C38 0.01uF
C39 0.1uF
C8 10uF
VOUT_DAC
TP4
R5 10k
R4 20k
ANALOG_VOS = -0.5 * ADC_VRT
LM8272 IC6-1
IC6-2 LM8272
ADC_VRT
TP5
ANALOG_VOS
TP6
5V0
-5V0
IC6-PWR LM8272
C13 0.1uF

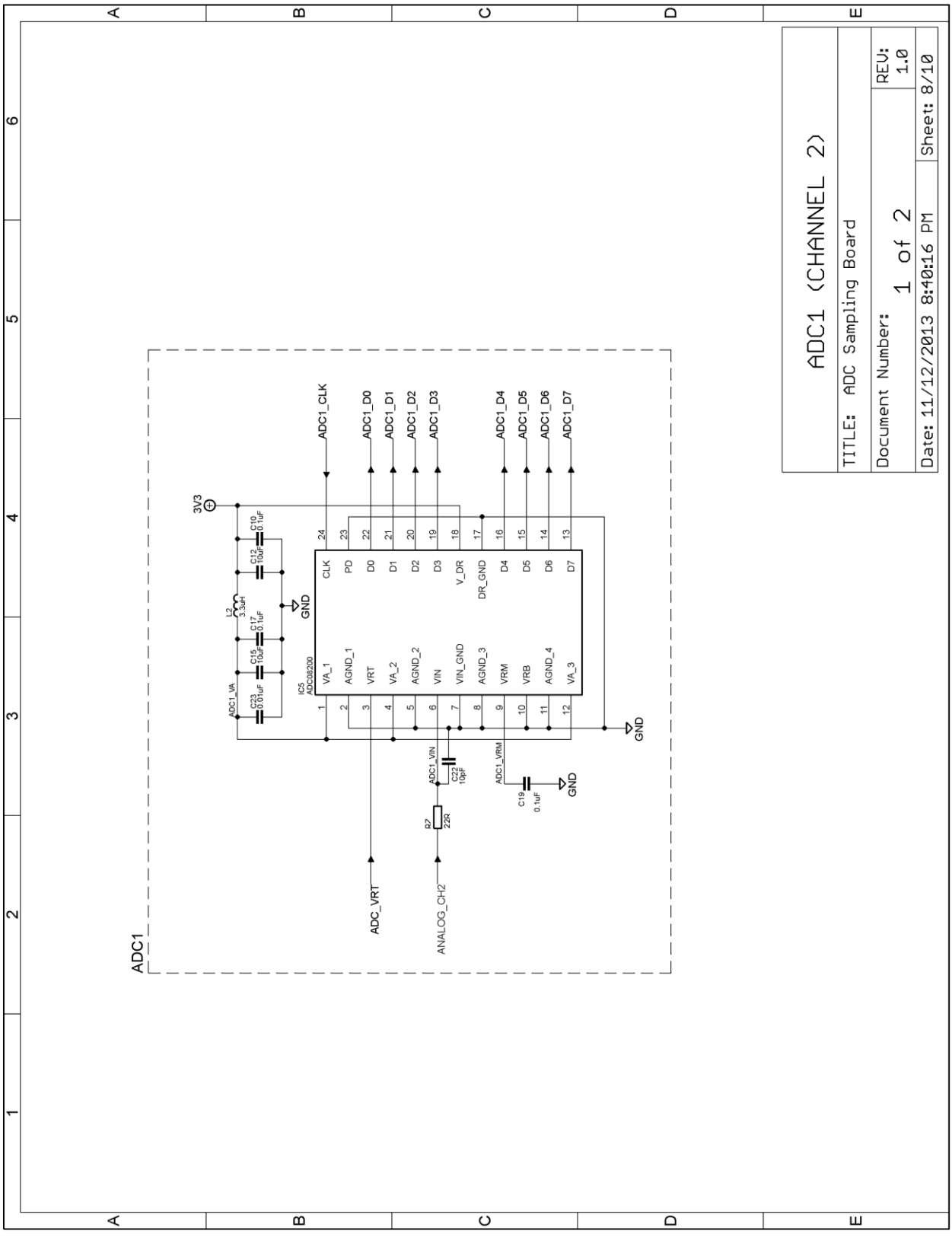
Analog Path Relay Controls



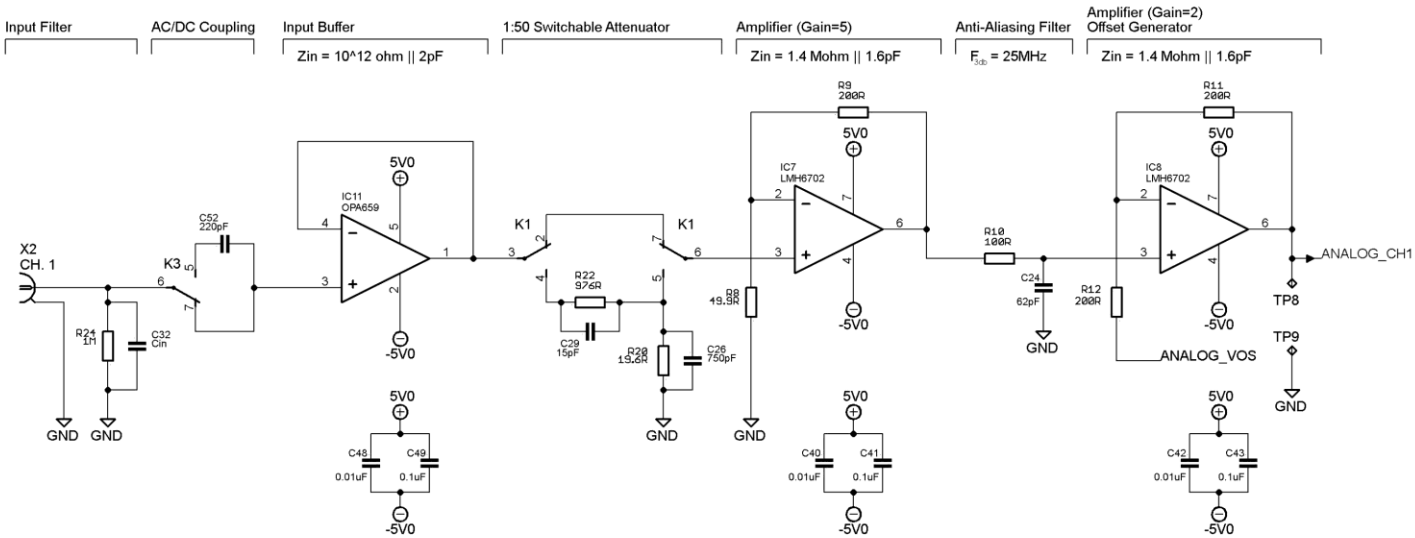
RELAY SWITCHES	
TITLE: ADC Sampling Board	
Document Number: 1 of 2	REV: 1.0
Date: 11/12/2013 8:40:16 PM	Sheet: 5/10


ADC0 Line Driver
ADC0_D7
ADC0_D6
ADC0_D5
ADC0_D4
ADC0_D3
ADC0_D2
ADC0_D1
ADC0_D0
A0
A1
A2
A3
Y0
Y1
Y2
Y3
OE
GND
EXP-IO8_N
EXP-IO7_N
EXP-IO6_N
EXP-IO5_N
EXP-IO4_N
EXP-IO3_N
EXP-IO2_N
EXP-IO1_N
IC2-1
74ALVCH16244
IC2-2
74ALVCH16244
ADC1 Line Driver
ADC1_D7
ADC1_D6
ADC1_D5
ADC1_D4
ADC1_D3
ADC1_D2
ADC1_D1
ADC1_D0
EXP-IO1_P
EXP-IO2_P
EXP-IO3_P
EXP-IO4_P
EXP-IO5_P
EXP-IO6_P
EXP-IO7_P
EXP-IO8_P
IC2-3
74ALVCH16244
IC2-4
74ALVCH16244
Clock Fanout Buffer
3V3
IC3
ICS553
VDD
OE
Q0
Q3
Q1
Q2
GND
ICLK
ADC1_CLK
ADC0_CLK
R3
50R
EXP-IO10_P
EXP-IO9_P
Digital IC Decoupling
IC2-PWR
74ALVCH16244
C36
0.1uF
C34
0.01uF
IC3-PWR
ICS553
C37
0.01uF
C35
0.1uF
DIGITAL DRIVERS AND BUFFERS
TITLE: ADC Sampling Board
Document Number:
1 of 2
REV:
1.0
Date: 11/12/2013 8:43:55 PM
Sheet: 6/10


1
2
3
4
5
6
A
B
C
D
E
ADC0
3V3
ADC0_VA
L1
3.3uH
C21
0.01uF
C14
10uF
C16
0.1uF
C11
10uF
C9
0.1uF
GND
IC4
ADC08200
VA_1
AGND_1
VRT
VA_2
AGND_2
VIN
VIN_GND
AGND_3
VRM
VRB
AGND_4
VA_3
CLK
PD
D0
D1
D2
D3
V_DR
DR_GND
D4
D5
D6
D7
ADC0_CLK
ADC0_D0
ADC0_D1
ADC0_D2
ADC0_D3
ADC0_D4
ADC0_D5
ADC0_D6
ADC0_D7
ADC_VRT
ANALOG_CH1
R6
22R
ADC0_VIN
C20
10pF
ADC0_VRM
C18
0.1uF
GND
GND
ADC0 (CHANNEL 1)
TITLE: ADC Sampling Board
Document Number:
1 of 2
REV:
1.0
Date: 11/12/2013 8:40:16 PM
Sheet: 7/10

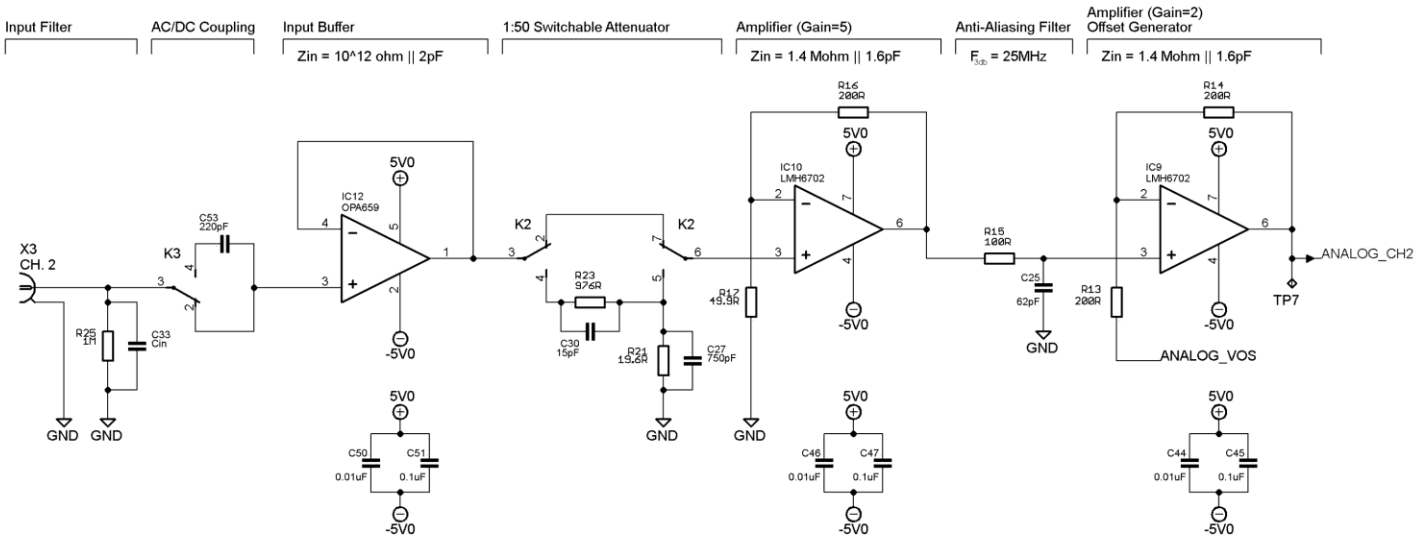

ADC1 (CHANNEL 2)
TITLE: ADC Sampling Board
Document Number:
1 of 2
REV: 1.0
Date: 11/12/2013 8:40:16 PM
Sheet: 8/10
ADC1
3V3
GND
ADC1_CLK
ADC1_D0
ADC1_D1
ADC1_D2
ADC1_D3
ADC1_D4
ADC1_D5
ADC1_D6
ADC1_D7
ADC1_VA
ADC1_VIN
ADC1_VRM
ADC_VRT
ANALOG_CH2
IC5
ADC08200
CLK
PD
D0
D1
D2
D3
V_DR
DR_GND
D4
D5
D6
D7
VA_1
AGND_1
VRT
VA_2
AGND_2
VIN
VIN_GND
AGND_3
VRM
VRB
AGND_4
VA_3
C10 0.1uF
C12 10uF
L2 3.3uH
C17 0.1uF
C15 10uF
C23 0.01uF
C22 10pF
C19 0.1uF
R7 22R

Channel 1 Analog Processing



ANALOG PROCESSING (CHANNEL 1)	
TITLE: ADC Sampling Board	
Document Number: 1 of 2	REV: 1.0
Date: 11/12/2013 8:40:16 PM	Sheet: 9/10

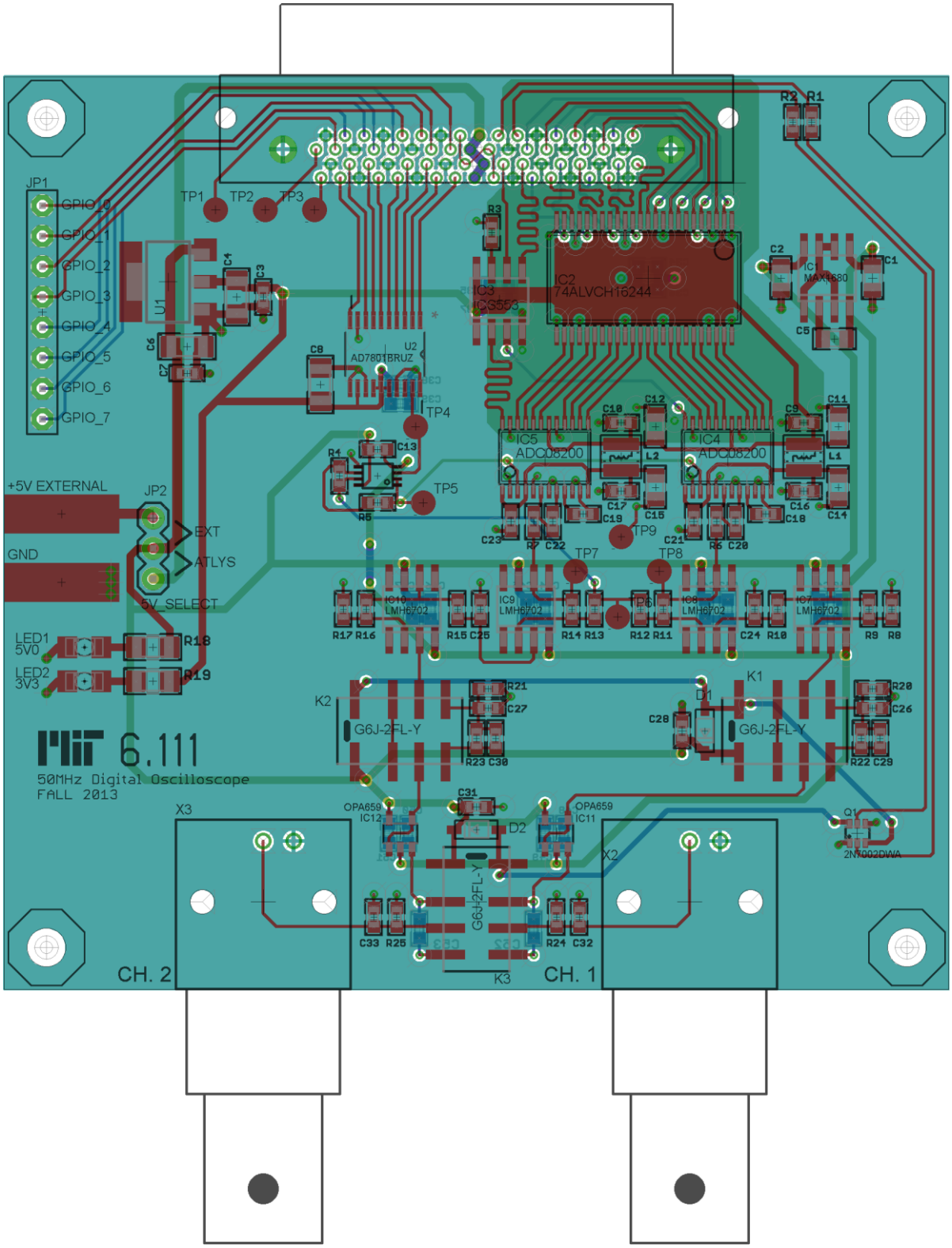
Channel 2 Analog Processing



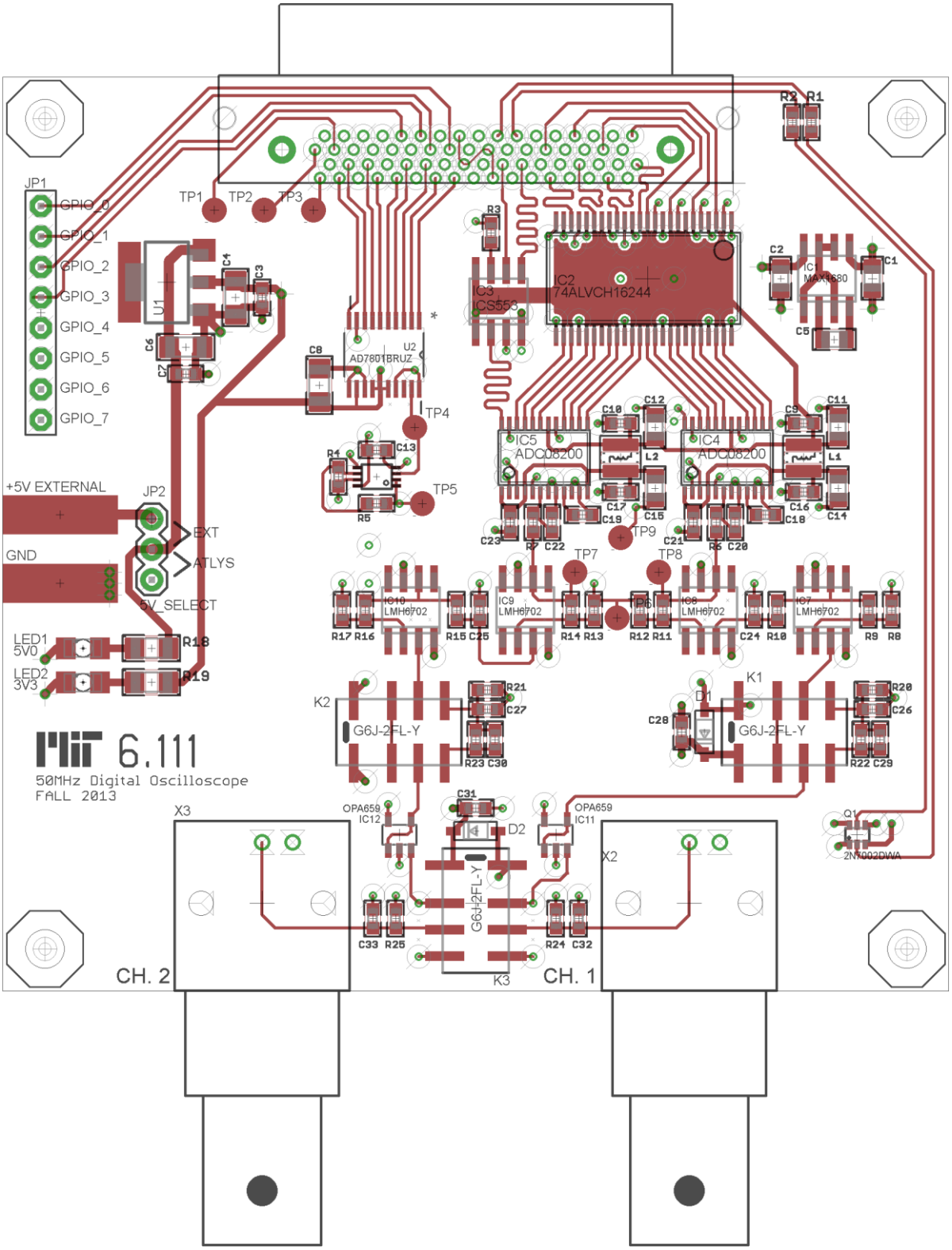
ANALOG PROCESSING (CHANNEL 2)		
TITLE: ADC Sampling Board		
Document Number: 1 of 2		REV: 1.0
Date: 11/12/2013 8:40:16 PM	Sheet: 10/10	

Appendix B: ADC Sampling Board PCB Layout

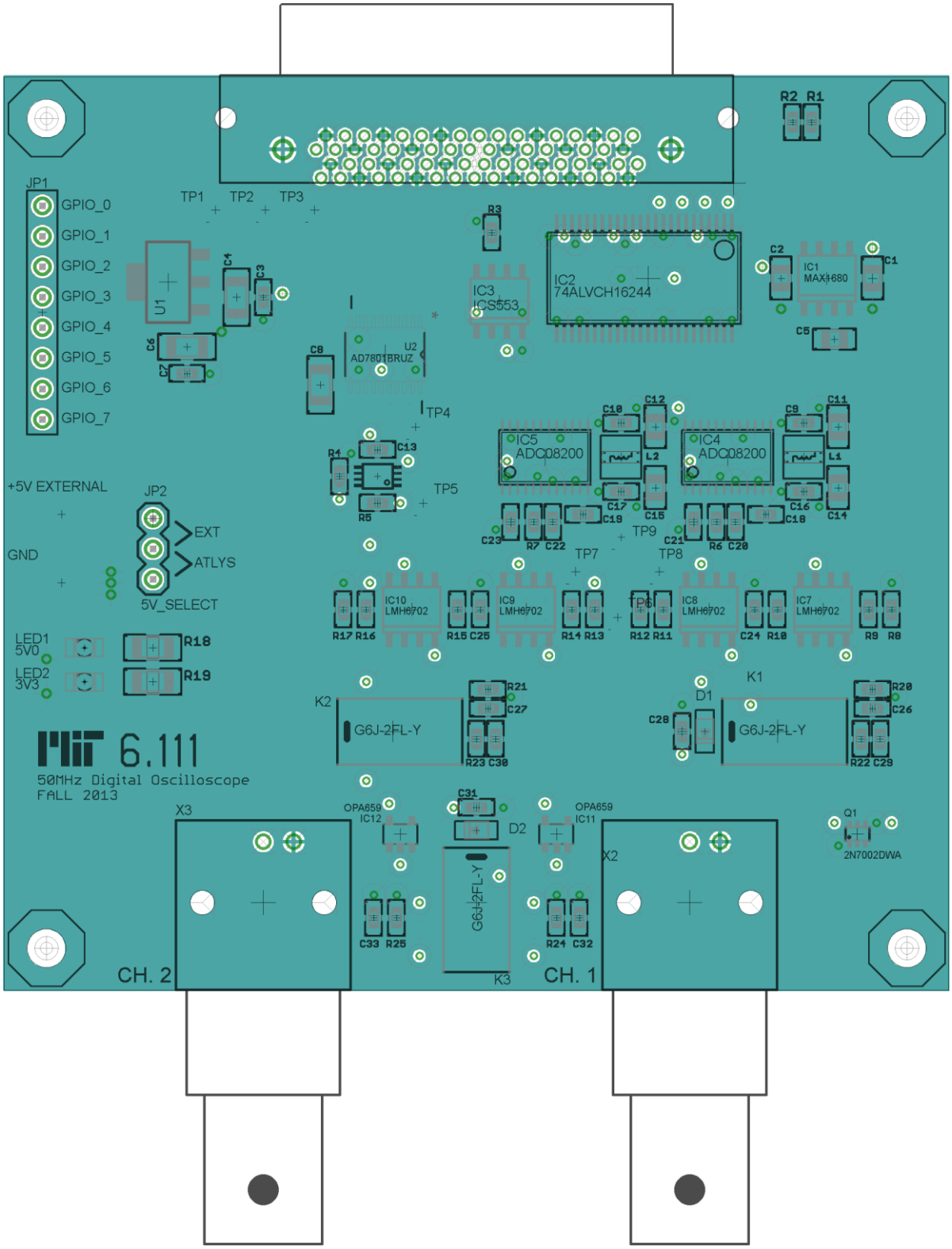
1. Complete View



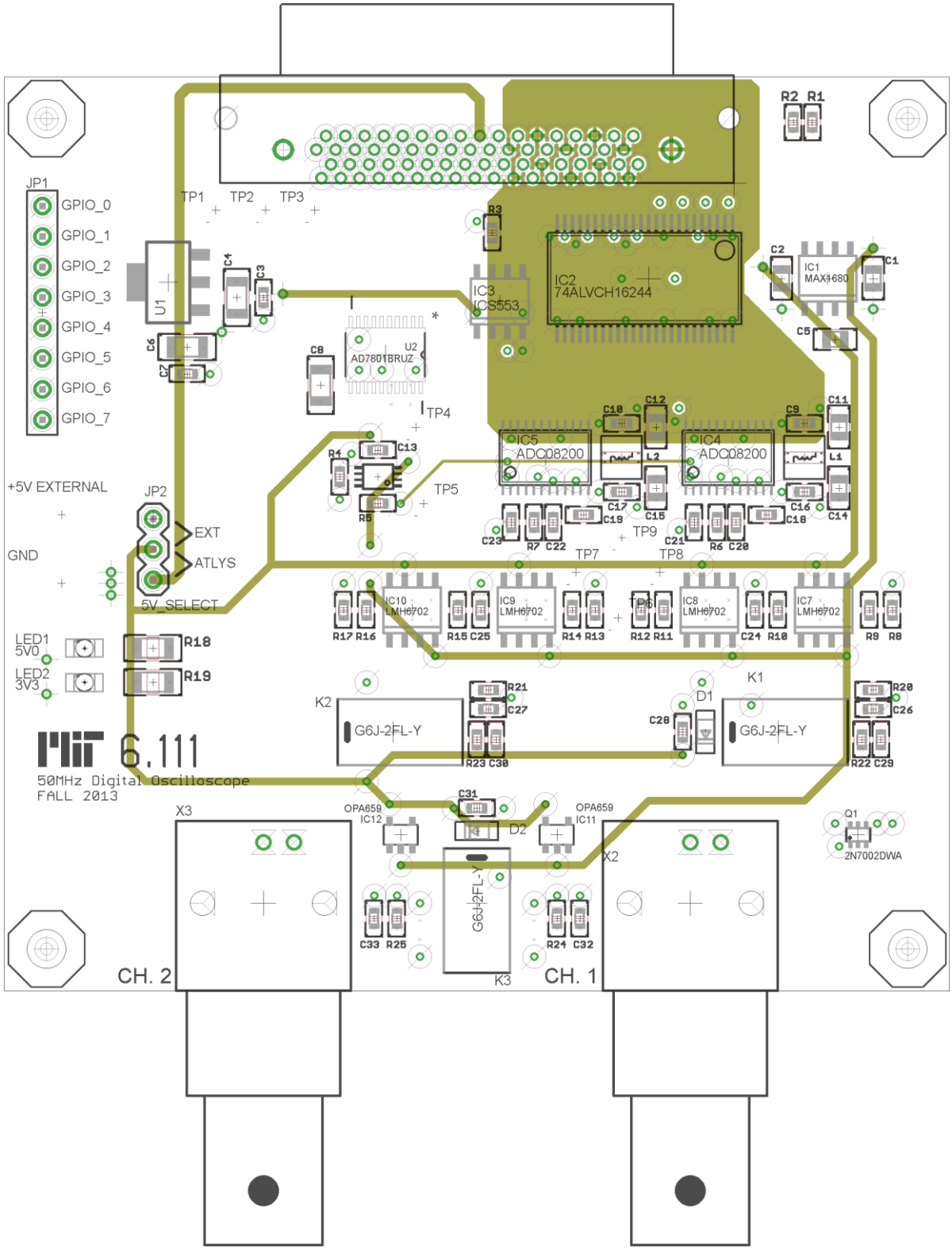
2. Top Silkscreen and Layer 1



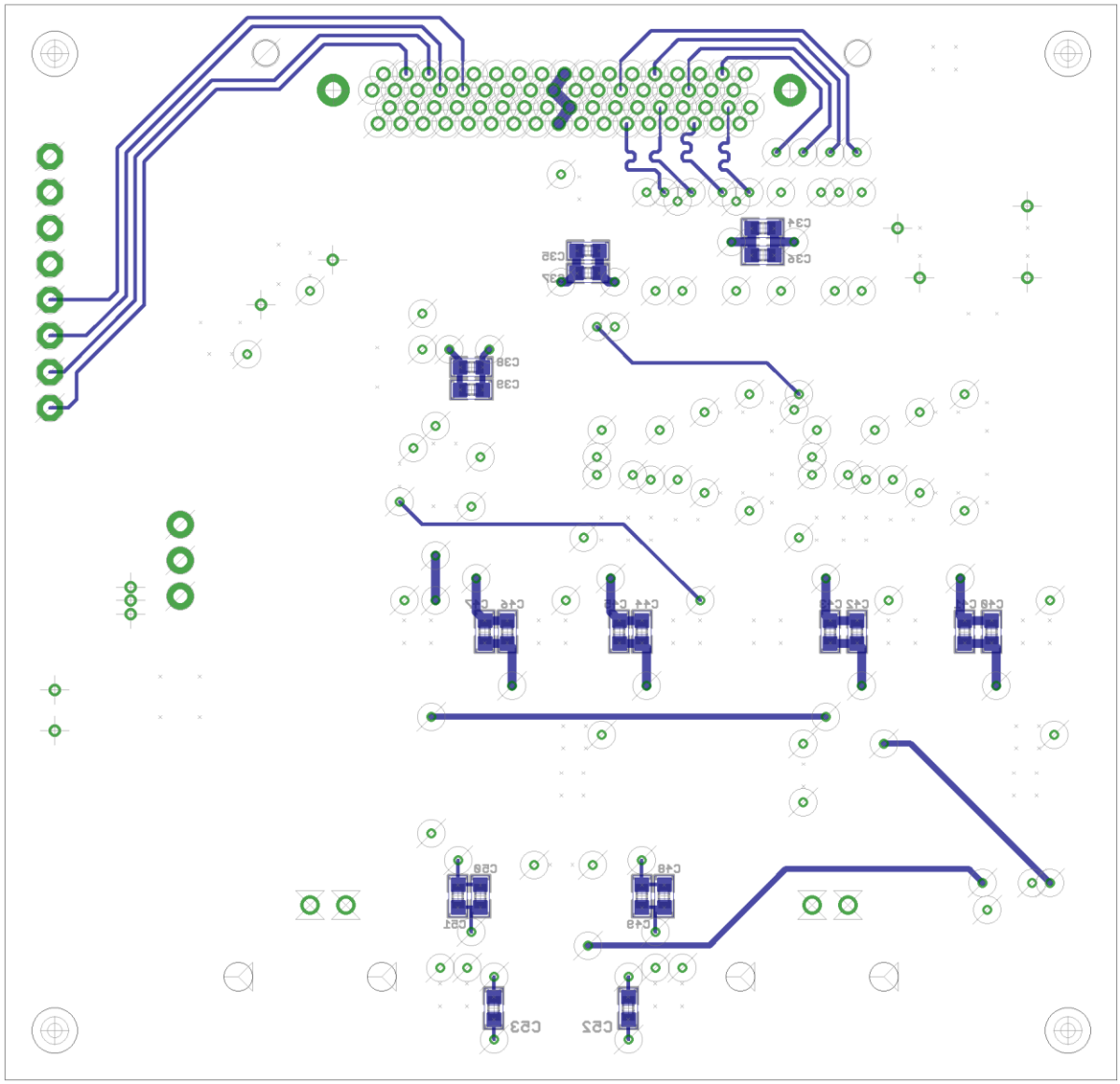
3. Top Silkscreen and Layer 2



4. Top Silkscreen and Layer 3



4. Bottom Silkscreen and Layer 4



Appendix C: Verilog Code

C1. Top-Level Controller (top_block_controller.v)

```
‘timescale 1ns/1ps

//`include "cstringdisp.v"
//`include "font_rom.v"

//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: KRAMNIK
//
// Create Date:    20:58:49 11/18/2013
// Design Name:
// Module Name:    top_block_controller
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module top_block_controller(
                input  wire SYS_CLK,

                input  wire [5:0] BTN,                      // BTN[5] = CENTER
(RSTBTN), BTN[4] = RIGHT, BTN[3] = DOWN, BTN[2] = LEFT, BTN[1] = UP, BTN[0] = M0/RESET
                input  wire [7:0] SW,                       // [7:4] used to set pong
ball speed, [3:0] used to set resolution

                // VHDCI Connector

                input wire [7:0] VHDCIIOP_ADC1_D, // ADC1 data output
                output wire VHDCIIOP_CLKOUT,                        // 100MHz clock
output
                input wire  VHDCIIOP_CLKIN,                         // 100MHz clock
return
                output wire [7:0] VHDCIIOP_DAC_DB,          // DAC data input
                output wire VHDCIIOP_DAC_WR_BAR,
                output wire VHDCIIOP_TP3,

                input wire [7:0] VHDCIION_ADC0_D, // ADC0 data output
                output wire VHDCIION_ATTEN_RLY,
                output wire VHDCIION_AC_EN_RLY,
                input wire [7:0] VHDCIION_GPIO,
                output wire VHDCIION_TP1,
                output wire VHDCIION_TP2,

                output wire [3:0] TMDS,
                output wire [3:0] TMDSB,
                output wire [3:0] LED

        );
```

```
assign VHDCIION_TP1 = 0;          // Do nothing with TP1 and 2
assign VHDCIION_TP2 = done_sampling;

// DAC
// Refresh DAC on VSYNC (60Hz frame refresh rate), echo to TP3 for probing
assign VHDCIIOP_TP3 = !display_vsync;

// LEDs
assign LED[1] = btn_gpio[0];
assign LED[0] = btn_gpio[7];

assign LED[2] = VHDCIION_AC_EN_RLY;
assign LED[3] = frame_triggered;

//////////////////////////////////////////////////////////////////////
// Create global clock and synchronous system reset.                //
//////////////////////////////////////////////////////////////////////
wire locked, reset, pwrup, sysclk;

IBUF sysclk_buf (.I(SYS_CLK), .O(sysclk));

wire pclk_lckd;

SRL16E #(.INIT(16'h1)) pwrup_0 (
        .Q(pwrup),
        .A0(1'b1),
        .A1(1'b1),
        .A2(1'b1),
        .A3(1'b1),
        .CE(pclk_lckd),
        .CLK(sysclk),
        .D(1'b0)
);

//////////////////////////////////////////////////////////////////////
// Send and receive 100MHz ADC clock
//////////////////////////////////////////////////////////////////////

assign VHDCIIOP_CLKOUT = sysclk;

wire ADC_clk;
IBUF ADC_clk_buf (.I(VHDCIIOP_CLKIN), .O(ADC_clk));

//////////////////////////////////////////////////////////////////////
// Generate pixel clock by dividing 100MHz sysclk.                  //
//////////////////////////////////////////////////////////////////////
wire clkfx, pclk;
DCM_CLKGEN #(
        .CLKFX_DIVIDE (40),
        .CLKFX_MULTIPLY (34),
        .CLKIN_PERIOD(20.000)
)
PCLK_GEN_INST (
        .CLKFX(clkfx),
        .CLKFX180(),
        .CLKFXDV(),
        .LOCKED(pclk_lckd),
        .PROGDONE(),
        .STATUS(),
        .CLKIN(sysclk),
        .FREEZEDCM(1'b0),
```

```
                .PROGCLK(1'b0),
                .PROGDATA(),
                .PROGEN(),
                .RST(1'b0)
        );

        //*****************************************************************//
        // Generate 2x pixel clock and 2x pixel clock using PLL.           //
        //*****************************************************************//
        wire pllclk0, pllclk1, pllclk2;
        wire pclkx2, pclkx10, pll_lckd;
        wire clkfbout;

        ////////////////////////////////////////////////////////////////////
        // Pixel Rate clock buffer
        //
        ////////////////////////////////////////////////////////////////////
        BUFG pclkbufg (.I(pllclk1), .O(pclk));

        ////////////////////////////////////////////////////////////////////
        // 2x pclk is going to be used to drive OSERDES2
        // on the GCLK side
        ////////////////////////////////////////////////////////////////////
        BUFG pclkx2bufg (.I(pllclk2), .O(pclkx2));

        ////////////////////////////////////////////////////////////////////
        // 10x pclk is used to drive IOCLK network so a bit rate reference
        // can be used by OSERDES2
        ////////////////////////////////////////////////////////////////////
        PLL_BASE # (
                .CLKIN_PERIOD(13),
                .CLKFBOUT_MULT(10), //set VCO to 10x of CLKIN
                .CLKOUT0_DIVIDE(1),
                .CLKOUT1_DIVIDE(10),
                .CLKOUT2_DIVIDE(5),
                .COMPENSATION("INTERNAL")
        ) PLL_OSERDES (
                .CLKFBOUT(clkfbout),
                .CLKOUT0(pllclk0),
                .CLKOUT1(pllclk1),
                .CLKOUT2(pllclk2),
                .CLKOUT3(),
                .CLKOUT4(),
                .CLKOUT5(),
                .LOCKED(pll_lckd),
                .CLKFBIN(clkfbout),
                .CLKIN(clkfx),
                .RST(~pclk_lckd)
        );

        wire serdesstrobe;
        wire bufpll_lock;
        BUFPLL #(.DIVIDE(5)) ioclk_buf (.PLLIN(pllclk0), .GCLK(pclkx2),
.LOCKED(pll_lckd),
                                        .IOCLK(pclkx10), .SERDESSTROBE(serdesstrobe),
.LOCK(bufpll_lock));

        synchro #(.INITIALIZE("LOGIC1"))
        synchro_reset (.async(!pll_lckd),.sync(reset),.clk(pclk));

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

```
        // Use buffers to drive FSM clocks
        //*********************************************************************//

        wire pclk_fsm_1;
        BUFG pclk_fsm_1_bufg (.I(pllclk1), .O(pclk_fsm_1));

        wire pclk_fsm_2;
        BUFG pclk_fsm_2_bufg (.I(pllclk1), .O(pclk_fsm_2));

        wire pclk_data;
        BUFG pclk_data_bufg (.I(pllclk1), .O(pclk_data));

        //*********************************************************************//
        // Debounce input buttons and switches
//
        //*********************************************************************//

        wire sysclk_dbnce;
        BUFG sysclk_dbnce_bufg (.I(sysclk), .O(sysclk_dbnce));

        // FPGA Board Buttons
        wire btn_up, btn_down, btn_center, btn_left, btn_right;

        debounce btn_up_dbnce    (.reset(reset), .clock(sysclk_dbnce), .noisy(BTN[1]),
.clean(btn_up));
        debounce btn_down_dbnce  (.reset(reset), .clock(sysclk_dbnce), .noisy(BTN[3]),
.clean(btn_down));
        debounce btn_center_dbnce(.reset(reset), .clock(sysclk_dbnce), .noisy(BTN[5]),
.clean(btn_center));
        debounce btn_left_dbnce  (.reset(reset), .clock(sysclk_dbnce), .noisy(BTN[2]),
.clean(btn_left));
        debounce btn_right_dbnce (.reset(reset), .clock(sysclk_dbnce), .noisy(BTN[4]),
.clean(btn_right));

        // GPIO Board Buttons
        wire [7:0] btn_gpio;

        debounce gpio0_dbnce(.reset(reset), .clock(sysclk_dbnce),
.noisy(VHDCIION_GPIO[0]), .clean(btn_gpio[0]));
        debounce gpio1_dbnce(.reset(reset), .clock(sysclk_dbnce),
.noisy(VHDCIION_GPIO[1]), .clean(btn_gpio[1]));
        debounce gpio2_dbnce(.reset(reset), .clock(sysclk_dbnce),
.noisy(VHDCIION_GPIO[2]), .clean(btn_gpio[2]));
        debounce gpio3_dbnce(.reset(reset), .clock(sysclk_dbnce),
.noisy(VHDCIION_GPIO[3]), .clean(btn_gpio[3]));
        debounce gpio4_dbnce(.reset(reset), .clock(sysclk_dbnce),
.noisy(VHDCIION_GPIO[4]), .clean(btn_gpio[4]));
        debounce gpio5_dbnce(.reset(reset), .clock(sysclk_dbnce),
.noisy(VHDCIION_GPIO[5]), .clean(btn_gpio[5]));
        debounce gpio6_dbnce(.reset(reset), .clock(sysclk_dbnce),
.noisy(VHDCIION_GPIO[6]), .clean(btn_gpio[6]));
        debounce gpio7_dbnce(.reset(reset), .clock(sysclk_dbnce),
.noisy(VHDCIION_GPIO[7]), .clean(btn_gpio[7]));

        //*********************************************************************//
        // Intermediate-level block instantiation.
//
        //*********************************************************************//

        wire [7:0] trig_lvl;
        wire draw_trig_lvl;
```

```
wire [9:0] trig_lvl_scaled;
wire frame_triggered;

wire [9:0] v_offset;

wire [3:0] v_per_dv_fsm_state;

wire [10:0] sample_addr;
wire [15:0] sample_data;
wire done_sampling;

wire [10:0] fbuf_wr_addr;
wire [9:0]  fbuf_wr_data;

wire display_vsync;

capture_and_store_block capture_and_store(
                .ADC_clk(ADC_clk),
                .pclk(pclk_data),
                .vsync(display_vsync),

                .ADC0_data(VHDCIION_ADC0_D),
                .ADC1_data(VHDCIIOP_ADC1_D),

                .trig_ch_select(SW[0]),
                .trig_lvl(trig_lvl),

                .sample_rd_addr(sample_addr),
                .sample_rd_data(sample_data),

                .done_sampling(done_sampling),
                .frame_triggered(frame_triggered)
        );

data_processing_block data_processing(
                .pclk_data(pclk),
                .pclk_fsm(pclk_fsm_1),
                .vsync(display_vsync),

                .btn_up(btn_gpio[2]),
                .btn_down(btn_gpio[5]),
                .btn_center(btn_center),
                .btn_left(btn_left),
                .btn_right(btn_right),
                .btn_ac_couple(btn_gpio[3]),

                .ac_couple(VHDCIION_AC_EN_RLY),
                .atten_en(VHDCIION_ATTEN_RLY),

                .dac_data(VHDCIIOP_DAC_DB),
                .dac_wr_bar(VHDCIIOP_DAC_WR_BAR),

                .trig_ch_select(SW[0]),
                .trig_lvl(trig_lvl),
                .trig_lvl_scaled(trig_lvl_scaled),
                .draw_trig_lvl(draw_trig_lvl),

                .v_offset_inc(btn_gpio[0]),
                .v_offset_dec(btn_gpio[7]),
                .v_offset(v_offset),
```

```
                        .btn_x1x10(btn_gpio[4]),
                        .v_per_dv_inc(btn_gpio[1]),
                        .v_per_dv_dec(btn_gpio[6]),
                        .v_per_dv_fsm_state(v_per_dv_fsm_state),

                        .done_sampling(done_sampling),

                        .sample_addr(sample_addr),
                        .sample_data(sample_data),

                        .fbuf_wr_addr(fbuf_wr_addr),
                        .fbuf_wr_data(fbuf_wr_data)
                );

        display_controller_block display_controller(
                        .pclk(pclk),
                        .pclkx2(pclkx2),
                        .pclkx10(pclkx10),
                        .pclk_fsm(pclk_fsm_2),

                        .reset(reset),
                        .serdesstrobe(serdesstrobe),
                        .bufpll_lock(bufpll_lock),

                        .trig_ch_select(SW[0]),
                        .trig_lvl_scaled(trig_lvl_scaled),
                        .draw_trig_lvl(draw_trig_lvl),

                        .v_offset(v_offset),

                        .v_per_dv_fsm_state(v_per_dv_fsm_state),

                        .fbuf_wr_addr(fbuf_wr_addr),
                        .fbuf_wr_data(fbuf_wr_data),

                        .vsync_out(display_vsync),

                        .TMDS(TMDS),
                        .TMDSB(TMDSB)
                );

endmodule
```

C2. Capture and Storage (capture_and_storage_block.v)

```
`timescale 1ns/1ps

//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: KRAMNIK
//
// Create Date:    20:59:11 11/18/2013
// Design Name:
// Module Name:    capture_and_store_block
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module capture_and_store_block(
                        input wire ADC_clk,
                        input wire pclk,

                        input wire vsync,

                        input wire [7:0] ADC0_data,
                        input wire [7:0] ADC1_data,

                        input wire trig_ch_select,                  // 0 = ADC0
(CH.1), 1 = ADC1 (CH.2)
                        input wire [7:0] trig_lvl,

                        input  wire [10:0] sample_rd_addr,
                        output wire [15:0] sample_rd_data,

                        output wire done_sampling,
                        output reg frame_triggered
    );

        reg triggered, sampling;
        initial triggered = 0;
        initial sampling = 0;

        assign done_sampling = !sampling;

        //////////////////////////////////////////////////////////////////////////////
        // Sample Buffer
        //////////////////////////////////////////////////////////////////////////////

        wire [10:0] sample_storage_addr;
        reg  [10:0] sample_wr_addr;
        wire [15:0] sample_wr_data;

        wire bram_clk;
```

```
        BUFGMUX BUFGMUX_bram (    // Write to bram at a rate of ADC_clk, read from
bram at a rate of pclk
                .O(bram_clk),             // Clock MUX output
                .I0(pclk),                        // Clock0 input
                .I1(ADC_clk),             // Clock1 input
                .S(sampling)              // Clock select input
        );

        bram #(.LOGSIZE(11),.WIDTH(16))

        sample_storage(.addr(sample_storage_addr),.clk(bram_clk),.we(sampling),.din(sa
mple_wr_data),.dout(sample_rd_data));

        assign sample_storage_addr = sampling ? sample_wr_addr : sample_rd_addr;
        // When sampling, store to sample_wr_addr (specfied by counter), else read
from sample_rd_addr (specified by data processing)

        //////////////////////////////////////////////////////////////////////
        // Triggering and Capture
        //////////////////////////////////////////////////////////////////////

        reg [7:0] ADC0_data_last, ADC1_data_last;

        reg vsync_last;

        assign sample_wr_data = {ADC0_data, ADC1_data};

        always @ (posedge ADC_clk) begin
                if(!triggered && (ADC0_data > trig_lvl) && (ADC0_data_last ==
trig_lvl) && !trig_ch_select) begin                        // Channel 1 (ADC0) trigger
selected
                        triggered <= 1;
                        sampling <= 1;
                        sample_wr_addr <= 11'b0;
                end
                else if(!triggered && (ADC1_data > trig_lvl) && (ADC1_data_last ==
trig_lvl) && trig_ch_select) begin                 // Channel 2 (ADC1) trigger
selected
                        triggered <= 1;
                        sampling <= 1;
                        sample_wr_addr <= 11'b0;
                end
                else if(!vsync_last && vsync) begin                // Reset trigger
flag every frame (we want to trigger once per frame)
                        triggered <= 0;
                        sampling <= 0;
                end
                else if(sampling) begin  // Fill up sample BRAM, increment the
counter every ADC clock cycle
                        if(sample_wr_addr == 11'b11111111111) begin
                                sampling <= 0;
                                sample_wr_addr <= 11'b0;
                        end
                        else begin
                                sample_wr_addr <= sample_wr_addr + 1;
                        end
                end

                ADC0_data_last <= ADC0_data;
                ADC1_data_last <= ADC1_data;
```

```
                vsync_last <= vsync;
        end

endmodule
```

C3. Data Processing (data_processing_block.v)

```
`timescale 1ns/1ps

//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: KRAMNIK
//
// Create Date:    20:59:28 11/18/2013
// Design Name:
// Module Name:    data_processing_block
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module data_processing_block(
                        input wire pclk_data,
                        input wire pclk_fsm,
                        input wire vsync,

                        input wire btn_up, btn_down, btn_center, btn_left, btn_right,

                        input wire btn_ac_couple,
                        output reg ac_couple,
                        output reg atten_en,

                        output reg [7:0] dac_data,
                        output wire dac_wr_bar,

                        input wire trig_ch_select,
                        output reg [7:0] trig_lvl,
                        output reg [9:0] trig_lvl_scaled,
                        output reg draw_trig_lvl,

                        input wire v_offset_inc,
                        input wire v_offset_dec,
                        output reg [9:0] v_offset,

                        input wire btn_x1x10,
                        input wire v_per_dv_inc,
                        input wire v_per_dv_dec,
                        output reg [3:0] v_per_dv_fsm_state,

                        input wire done_sampling,

                        output reg [10:0] sample_addr,
                        input wire [15:0] sample_data,    // {ADC0_data, ADC1_data}

                        output reg [10:0] fbuf_wr_addr,
                        output reg [9:0]  fbuf_wr_data
    );
```

```
        reg vsync_last;

        //////////////////////////////////////////////////////////////////////
        // Vertical Offset FSM
        //////////////////////////////////////////////////////////////////////

        reg v_offset_inc_last, v_offset_dec_last;
        initial v_offset = 10'd340;

        always @ (posedge pclk_fsm) begin
                if(v_offset_inc && (v_offset < 10'd640) && vsync && !vsync_last)
begin
                        v_offset <= v_offset + 1;
                end
                else if(v_offset_dec && (v_offset > 10'd0) && vsync && !vsync_last)
begin
                        v_offset <= v_offset - 1;
                end

                v_offset_inc_last <= v_offset_inc;
                v_offset_dec_last <= v_offset_dec;
        end

        //////////////////////////////////////////////////////////////////////
        // Trigger Level FSM
        //////////////////////////////////////////////////////////////////////

        reg btn_up_last, btn_down_last;
        initial trig_lvl = 8'b01111111;

        parameter TRIG_HOLD_FRAMES = 8'd60;                // Number of frames the the
trigger line is displayed on the screen after adjustment is set
        reg [7:0] trig_hold_counter;

        always @ (posedge pclk_fsm) begin          // Press up = move trigger level
up, press down = move trigger level down
                if(btn_up && (trig_lvl < 8'b11111111) && vsync && !vsync_last) begin
                        trig_lvl <= trig_lvl + 1;
                        draw_trig_lvl <= 1;
                        trig_hold_counter <= 8'b0;
                end
                else if(btn_down && (trig_lvl > 8'b0) && vsync && !vsync_last) begin
                        trig_lvl <= trig_lvl - 1;
                        draw_trig_lvl <= 1;
                        trig_hold_counter <= 8'b0;
                end
                else if (vsync && !vsync_last && draw_trig_lvl) begin
        // Keep the trigger line displayed on the screen for TRIG_HOLD_FRAMES after
the user changes the level
                        if(trig_hold_counter == TRIG_HOLD_FRAMES) begin
                                draw_trig_lvl <= 0;
                        end
                        else begin
                                trig_hold_counter <= trig_hold_counter + 1;
                        end
                end

                trig_lvl_scaled = ((5 * {2'b0, trig_lvl}) >> 1) + v_offset - 10'd340;

                btn_up_last <= btn_up;
                btn_down_last <= btn_down;
```

```
        end

        //*****************************************************************//
        // Volts per Div FSM
        //*****************************************************************//

        initial v_per_dv_fsm_state = MV_PER_DV_100;
        initial atten_en = 0;
        initial dac_data[7:0] = 8'd78;             // 78/255 * 3.3V = 1.01V

        reg v_per_dv_inc_last, v_per_dv_dec_last;

        // Parameter names referenced to x10 scope probe (handle x1 probe by changing
the screen text in the display controller)
        parameter MV_PER_DV_2000 = 4'b0101;
        parameter MV_PER_DV_1000 = 4'b0100;
        parameter MV_PER_DV_500  = 4'b0011;
        parameter MV_PER_DV_100  = 4'b0010;
        parameter MV_PER_DV_50   = 4'b0001;
        parameter MV_PER_DV_10   = 4'b0000;

        always @(posedge pclk_fsm) begin
                if (v_per_dv_inc && !v_per_dv_inc_last) begin
                        case(v_per_dv_fsm_state)
                                MV_PER_DV_2000: begin
                                        // Do nothing, the call is out of bounds.
                                end
                                MV_PER_DV_1000: begin
                                        v_per_dv_fsm_state <= MV_PER_DV_2000;
                                        atten_en <= 1;
                                        dac_data <= 8'd78;        // 78/255 * 3.3V =
1.01V

                                end
                                MV_PER_DV_500: begin
                                        v_per_dv_fsm_state <= MV_PER_DV_1000;
                                        atten_en <= 1;
                                        dac_data <= 8'd39;        // 39/255 * 3.3V =
0.51V

                                end
                                MV_PER_DV_100: begin
                                        v_per_dv_fsm_state <= MV_PER_DV_500;
                                        atten_en <= 1;
                                        dac_data <= 8'd16;                // 16/255
* 3.3V = 0.21V

                                end
                                MV_PER_DV_50: begin
                                        v_per_dv_fsm_state <= MV_PER_DV_100;
                                        atten_en <= 0;
                                        dac_data <= 8'd78;        // 78/255 * 3.3V =
1.01V

                                end
                                MV_PER_DV_10: begin
                                        v_per_dv_fsm_state <= MV_PER_DV_50;
                                        atten_en <= 0;
                                        dac_data <= 8'd39;        // 39/255 * 3.3V =
0.51V
```

```
                                end
                                default: begin                          // Just in
case...
                                        v_per_dv_fsm_state <= MV_PER_DV_100;
                                        atten_en <= 0;
                                        dac_data <= 8'd78;      // 78/255 * 3.3V =
1.01V
                                end
                        endcase
                end
                else if (v_per_dv_dec && !v_per_dv_dec_last) begin
                        case(v_per_dv_fsm_state)
                                MV_PER_DV_2000: begin
                                        v_per_dv_fsm_state <= MV_PER_DV_1000;
                                        atten_en <= 1;
                                        dac_data <= 8'd39;      // 39/255 * 3.3V =
0.51V

                                end
                                MV_PER_DV_1000: begin
                                        v_per_dv_fsm_state <= MV_PER_DV_500;
                                        atten_en <= 1;
                                        dac_data <= 8'd16;              // 16/255
* 3.3V = 0.21V

                                end
                                MV_PER_DV_500: begin
                                        v_per_dv_fsm_state <= MV_PER_DV_100;
                                        atten_en <= 0;
                                        dac_data <= 8'd78;      // 78/255 * 3.3V =
1.01V

                                end
                                MV_PER_DV_100: begin
                                        v_per_dv_fsm_state <= MV_PER_DV_50;
                                        atten_en <= 0;
                                        dac_data <= 8'd39;      // 39/255 * 3.3V =
0.51V

                                end
                                MV_PER_DV_50: begin
                                        v_per_dv_fsm_state <= MV_PER_DV_10;
                                        atten_en <= 0;
                                        dac_data <= 8'd19;      // 16/255 * 3.3V =
0.25V

                                end
                                MV_PER_DV_10: begin
                                        // Do nothing, the call is out of bounds.
                                end
                                default: begin                          // Just in
case...
                                        v_per_dv_fsm_state <= MV_PER_DV_100;
                                        atten_en <= 0;
                                        dac_data <= 8'd78;      // 78/255 * 3.3V =
1.01V
                                end
                        endcase
                end

                v_per_dv_inc_last <= v_per_dv_inc;
```

```
                v_per_dv_dec_last <= v_per_dv_dec;
        end

        // AC/DC coupling

        always @(posedge btn_ac_couple) begin
                ac_couple <= !ac_couple;
        end

        // Reference DAC Update
        assign dac_wr_bar = !vsync;

        //////////////////////////////////////////////////////////////////////
        // Pixel Calculator
        //////////////////////////////////////////////////////////////////////

        reg done_writing_fbuf;
        initial done_writing_fbuf = 0;

        initial sample_addr = 11'b0;
        initial fbuf_wr_addr = 11'b0;

        reg [10:0] sample_data_scaled;
        initial sample_data_scaled = 11'b0;

        reg [7:0] sample_data_last;
        initial sample_data_last = 8'b0;

        reg [2:0] interpolate_counter;
        initial interpolate_counter = 3'b0;

        always @ (posedge pclk_data) begin
                if(done_sampling && !done_writing_fbuf) begin
                        if(fbuf_wr_addr == 11'b11111111111) begin
                                sample_addr                     <= 11'b0;
                                fbuf_wr_addr            <= 11'b0;
                                done_writing_fbuf <= 1;
                                interpolate_counter <= 3'b0;
                        end
                        else begin              // Write trigger channel to the
screen (1 = CH.2, 0 = CH.1), fbuf_wr_data is 10 bits so pad with zeros
                                // Pipeline scaling and shifting the sample data
                                sample_data_scaled <= trig_ch_select ? ((5 * {2'b0,
sample_data[7:0]}) >> 1) : ((5 * {2'b0, sample_data[15:8]}) >> 1);

                                interpolate_counter <= interpolate_counter + 1;

                                fbuf_wr_data <= sample_data_scaled + v_offset -
10'd340; // 5/2 * 256 = 640

                                sample_addr  <= &interpolate_counter ? (sample_addr
+ 1)  : sample_addr;
                                fbuf_wr_addr <= &interpolate_counter ? (fbuf_wr_addr
+ 1) : fbuf_wr_addr;

                                //fbuf_wr_data <= fbuf_wr_addr;   // TEST PATTERN
(Use to make sure display controller is working correctly, draws a ramp)
                        end
                end

                if (vsync && !vsync_last) begin
```

```
                done_writing_fbuf <= 0;
            end

            vsync_last <= vsync;
        end

endmodule
```

C4. Display Controller (display_controller_block.v)

```
`timescale 1ns/1ps

//`include "font_rom.v"

//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    20:59:46 11/18/2013
// Design Name:
// Module Name:    display_controller_block
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module display_controller_block(
                input wire pclk,
                input wire pclkx2,
                input wire pclkx10,
                input wire pclk_fsm,

                input wire reset,
                input wire serdesstrobe,
                input wire bufpll_lock,

                input wire trig_ch_select,
                input wire [9:0] trig_lvl_scaled,
                input wire draw_trig_lvl,

                input wire [9:0] v_offset,

                input wire [3:0] v_per_dv_fsm_state,

                input wire [10:0] fbuf_wr_addr,
                input wire [9:0]  fbuf_wr_data,

                output wire vsync_out,

                output wire [3:0] TMDS,
                output wire [3:0] TMDSB
        );

        assign vsync_out = VGA_VSYNC_INT;

        //*************************************************************************//
        // Frame Buffers
        //*************************************************************************//
```

```
        reg fbuf_wr_loc;                    // 0 = write to FB0, read from FB1, 1 =
write to FB1, read from FB0
        initial fbuf_wr_loc = 0;

        always @ (posedge vsync_out) fbuf_wr_loc <= !fbuf_wr_loc;   // Swap frame
buffers every frame

        wire [10:0] fbuf_addr_0, fbuf_addr_1;
        wire [9:0]  fbuf_mem_out_0, fbuf_mem_out_1, fbuf_rd_data;

        /*              Alternate bram generated using Xilinx tool
        bram_frame
frame_buffer_0(.clka(pclk),.wea(!fbuf_wr_loc),.addra(fbuf_addr_0),.dina(fbuf_wr_data),
.douta(fbuf_mem_out_0));
        bram_frame frame_buffer_1(.clka(pclk),.wea(fbuf_wr_loc)
,.addra(fbuf_addr_1),.dina(fbuf_wr_data),.douta(fbuf_mem_out_1));
        */

        bram #(.LOGSIZE(11),.WIDTH(10))

        frame_buffer_0(.addr(fbuf_addr_0),.clk(pclk),.we(!fbuf_wr_loc),.din(fbuf_wr_da
ta),.dout(fbuf_mem_out_0));

        bram #(.LOGSIZE(11),.WIDTH(10))

        frame_buffer_1(.addr(fbuf_addr_1),.clk(pclk),.we(fbuf_wr_loc),.din(fbuf_wr_dat
a),.dout(fbuf_mem_out_1));


        assign fbuf_addr_0 = fbuf_wr_loc ? bgnd_hcount : fbuf_wr_addr;
        assign fbuf_addr_1 = fbuf_wr_loc ? fbuf_wr_addr : bgnd_hcount;
        assign fbuf_rd_data = fbuf_wr_loc ? fbuf_mem_out_0 : fbuf_mem_out_1;

        //////////////////////////////////////////////////////////////////////
        // Pixel Generation Logic
        //////////////////////////////////////////////////////////////////////

        wire [10:0] bgnd_hcount, bgnd_vcount;

        wire [7:0] red_data, green_data, blue_data;

        reg [7:0] red_data_ingrid, green_data_ingrid, blue_data_ingrid;

        reg [9:0] fbuf_rd_data_last;
        always @ (posedge pclk) fbuf_rd_data_last <= fbuf_rd_data;

        parameter H_GRID_OFFSET = 11'd34;
        parameter V_GRID_OFFSET = 11'd34;
        parameter PX_PER_DIV = 11'd64;

        wire draw_trace_px;
        assign draw_trace_px = (bgnd_vcount == (10 * PX_PER_DIV - fbuf_rd_data_last +
V_GRID_OFFSET));

        wire [3:0] draw_connecting_px;
        wire [11:0] last_px, this_px;

        assign last_px = 10 * PX_PER_DIV - fbuf_rd_data_last + V_GRID_OFFSET;
        assign this_px = 10 * PX_PER_DIV - fbuf_rd_data              + V_GRID_OFFSET;

        assign draw_connecting_px[0] = (bgnd_vcount <= this_px);
```

```
        assign draw_connecting_px[1] = (bgnd_vcount >= last_px);

        assign draw_connecting_px[2] = (bgnd_vcount >= this_px);
        assign draw_connecting_px[3] = (bgnd_vcount <= last_px);

        wire px_in_grid_range;
        assign px_in_grid_range = (bgnd_hcount >= (H_GRID_OFFSET)) &&

(bgnd_vcount >= (V_GRID_OFFSET)) &&

(bgnd_hcount <= (H_GRID_OFFSET + 16 * PX_PER_DIV)) &&

(bgnd_vcount <= (V_GRID_OFFSET + 10 * PX_PER_DIV));

        always @ (posedge pclk) begin
                if(px_in_grid_range) begin
                        if((bgnd_vcount == (10 * PX_PER_DIV - trig_lvl_scaled +
V_GRID_OFFSET)) && draw_trig_lvl) begin            // Draw trigger level
                                red_data_ingrid   <= 8'b00000000;
                                green_data_ingrid <= 8'b11111111;
                                blue_data_ingrid  <= 8'b00000000;
                        end
                        else if((&draw_connecting_px[3:2]) ||
(&draw_connecting_px[1:0])) begin
                                red_data_ingrid   <= trig_ch_select ? 8'b00000000 :
8'b11111111;                                                                    // CH1
                                green_data_ingrid <= trig_ch_select ? 8'b11111111 :
8'b11111111;
                                blue_data_ingrid  <= trig_ch_select ? 8'b11111111 :
8'b00000000;
                        end
                        else if((bgnd_hcount == (H_GRID_OFFSET + 8  * PX_PER_DIV))
||
                                        (bgnd_vcount == (V_GRID_OFFSET + 5  *
PX_PER_DIV))) begin
                                red_data_ingrid   <= 8'b11111111;
                                green_data_ingrid <= 8'b11111111;
                                blue_data_ingrid  <= 8'b11111111;
                        end
                        else if((bgnd_hcount == (H_GRID_OFFSET))
                                        (bgnd_hcount == (H_GRID_OFFSET +
PX_PER_DIV))  ||
                                        (bgnd_hcount == (H_GRID_OFFSET + 2  *
PX_PER_DIV))  ||
                                        (bgnd_hcount == (H_GRID_OFFSET + 3  *
PX_PER_DIV))  ||
                                        (bgnd_hcount == (H_GRID_OFFSET + 4  *
PX_PER_DIV))  ||
                                        (bgnd_hcount == (H_GRID_OFFSET + 5  *
PX_PER_DIV))  ||
                                        (bgnd_hcount == (H_GRID_OFFSET + 6  *
PX_PER_DIV))  ||
                                        (bgnd_hcount == (H_GRID_OFFSET + 7  *
PX_PER_DIV))  ||
                                        //(bgnd_hcount == (H_GRID_OFFSET + 8  *
PX_PER_DIV))||           // Major grid line, skip it
                                        (bgnd_hcount == (H_GRID_OFFSET + 9  *
PX_PER_DIV))  ||
                                        (bgnd_hcount == (H_GRID_OFFSET + 10 *
PX_PER_DIV))  ||
```

```
                                        (bgnd_hcount == (H_GRID_OFFSET + 11 *
PX_PER_DIV))  ||
                                        (bgnd_hcount == (H_GRID_OFFSET + 12 *
PX_PER_DIV))  ||
                                        (bgnd_hcount == (H_GRID_OFFSET + 13 *
PX_PER_DIV))  ||
                                        (bgnd_hcount == (H_GRID_OFFSET + 14 *
PX_PER_DIV))  ||
                                        (bgnd_hcount == (H_GRID_OFFSET + 15 *
PX_PER_DIV))  ||
                                        (bgnd_hcount == (H_GRID_OFFSET + 16 *
PX_PER_DIV))  ||

                                        (bgnd_vcount == (V_GRID_OFFSET))
                                        (bgnd_vcount == (V_GRID_OFFSET +
PX_PER_DIV))  ||
                                        (bgnd_vcount == (V_GRID_OFFSET + 2  *
PX_PER_DIV))  ||
                                        (bgnd_vcount == (V_GRID_OFFSET + 3  *
PX_PER_DIV))  ||
                                        (bgnd_vcount == (V_GRID_OFFSET + 4  *
PX_PER_DIV))  ||
                                        //(bgnd_vcount == (V_GRID_OFFSET + 5  *
PX_PER_DIV))||           // Major grid line, skip it
                                        (bgnd_vcount == (V_GRID_OFFSET + 6  *
PX_PER_DIV))  ||
                                        (bgnd_vcount == (V_GRID_OFFSET + 7  *
PX_PER_DIV))  ||
                                        (bgnd_vcount == (V_GRID_OFFSET + 8  *
PX_PER_DIV))  ||
                                        (bgnd_vcount == (V_GRID_OFFSET + 9  *
PX_PER_DIV))  ||
                                        (bgnd_vcount == (V_GRID_OFFSET + 10 *
PX_PER_DIV))) begin
                                red_data_ingrid   <= 8'b01111111;
                                green_data_ingrid <= 8'b01111111;
                                blue_data_ingrid  <= 8'b01111111;
                        end
                        else begin
                                red_data_ingrid   <= 8'b00000000;
                                green_data_ingrid <= 8'b00000000;
                                blue_data_ingrid  <= 8'b00000000;
                        end
                end
                else begin
                        red_data_ingrid   <= 8'b00000000;
                        green_data_ingrid <= 8'b00000000;
                        blue_data_ingrid  <= 8'b00000000;
                end
        end

        // 0V Reference Arrow Display (Points to Vertical Offset)

        wire [63:0] gnd_pointer_string = "<-";
        wire [2:0] gnd_pointer_px;

        char_string_display gnd_pointer_text(
                        pclk_fsm,
                        bgnd_hcount,
                        bgnd_vcount[9:0],
                        gnd_pointer_px,
                        gnd_pointer_string,
```

```
                        11'd960,
                        10 * PX_PER_DIV - v_offset + V_GRID_OFFSET + 10'd10 );
        // y

        // Channel Display

        wire [63:0] ch_readout_string = trig_ch_select ? "CH2" : "CH1";
        wire [2:0]  ch_readout_px;

        char_string_display ch_readout_text(
                        pclk_fsm,
                        bgnd_hcount,
                        bgnd_vcount[9:0],
                        ch_readout_px,
                        ch_readout_string,
                        11'd1128,               // x
                        10'd34 );               // y

        // Timing error!
        // Even instantiating the following modules will cause the trace to stop
displaying.

        // Volts per Vertical Division Display
        /*
        // Parameter names referenced to x10 scope probe (handle x1 probe by changing
the screen text in the display controller)
        parameter MV_PER_DV_2000 = 4'b0101;
        parameter MV_PER_DV_1000 = 4'b0100;
        parameter MV_PER_DV_500  = 4'b0011;
        parameter MV_PER_DV_100  = 4'b0010;
        parameter MV_PER_DV_50   = 4'b0001;
        parameter MV_PER_DV_10   = 4'b0000;

        reg [63:0] v_per_div_string = "100mV/dv";
        initial v_per_div_string = "100mV/dv";

        always @(posedge vsync_out) begin
                case(v_per_dv_fsm_state)
                        MV_PER_DV_2000: v_per_div_string <= "2V/dv";
                        MV_PER_DV_1000: v_per_div_string <= "1V/dv";
                        MV_PER_DV_500:  v_per_div_string <= "500mV/dv";
                        MV_PER_DV_100:  v_per_div_string <= "100mV/dv";
                        MV_PER_DV_50:   v_per_div_string <= "50mV/dv";
                        MV_PER_DV_10:   v_per_div_string <= "10mV/dv";
                endcase
        end

        wire [2:0]  v_per_div_px;
        char_string_display v_per_div_text(
                        pclk_fsm,
                        bgnd_hcount,
                        bgnd_vcount[9:0],
                        v_per_div_px,
                        v_per_div_string,
                        11'd1128,               // x
                        10'd64 );               // y
        */
        // Time per Horizontal Division Display
        /*
        wire [63:0] t_per_div_string = "70ns/dv";
        wire [2:0]  t_per_div_px;
```

```
        char_string_display t_per_div_text(
                        pclk,           // NOTE! Not pclk_fsm!
                        bgnd_hcount,
                        bgnd_vcount[9:0],
                        t_per_div_px,
                        t_per_div_string,
                        11'd1128,               // x
                        10'd98 );               // y

        wire or1_red, or1_green, or1_blue;
        wire or2_red, or2_green, or2_blue;
        wire or3_red, or3_green, or3_blue;
        //wire [7:0] or4_red, or4_green, or4_blue;
        */
        /*
        assign or1_red   = gnd_pointer_px[2] | ch_readout_px[2];
        assign or1_green = gnd_pointer_px[1] | ch_readout_px[1];
        assign or1_blue  = gnd_pointer_px[0] | ch_readout_px[0];

        assign or2_red   = t_per_div_px[2] | v_per_div_px[2];
        assign or2_green = t_per_div_px[1] | v_per_div_px[1];
        assign or2_blue  = t_per_div_px[0] | v_per_div_px[0];

        assign or3_red   = or1_red   | or2_red;
        assign or3_green = or1_green | or2_green;
        assign or3_blue  = or1_blue  | or2_blue;
        */

        /*
        assign or4_red   = red_data_trace   | red_data_trigger;
        assign or4_green = green_data_trace | green_data_trigger;
        assign or4_blue  = blue_data_trace  | blue_data_trigger;
        */


        //assign red_data   = {8{or1_red}}   | red_data_ingrid;
        //assign green_data = {8{or1_green}} | green_data_ingrid;
        //assign blue_data  = {8{or1_blue}}  | blue_data_ingrid;


        assign red_data   = red_data_ingrid   | {8{ch_readout_px[2]}} |
{8{!trig_ch_select & gnd_pointer_px[2]}};
        assign green_data = green_data_ingrid | {8{ch_readout_px[1]}} |
{8{gnd_pointer_px[1]}};
        assign blue_data  = blue_data_ingrid  | {8{ch_readout_px[0]}} |
{8{trig_ch_select & gnd_pointer_px[0]}};

        /*
        assign red_data   = red_data_ingrid   | {8{v_per_div_px[2]}};
//{8{ch_readout_px[2]}} | {8{gnd_pointer_px[2]}};
        assign green_data = green_data_ingrid | {8{v_per_div_px[1]}};
//{8{ch_readout_px[1]}} | {8{gnd_pointer_px[1]}};
        assign blue_data  = blue_data_ingrid  | {8{v_per_div_px[0]}};
//{8{ch_readout_px[0]}} | {8{gnd_pointer_px[0]}};
        */

        //////////////////////////////////////////////////////////////////////
        // VGA Timing Parameters
        //////////////////////////////////////////////////////////////////////
```

```
        //      1366x768 (pclk = 85MHz, M=34, D=40 in top_block_controller)
        wire hsync_polarity = 1'b1;       //1-Negative, 0-Positive
        wire vsync_polarity = 1'b0;

        parameter HPIXELS = 11'd1366;     // Horizontal Live Pixels
        parameter VLINES  = 11'd768;  // Vertical Live Pixels
        parameter HSYNCPW = 11'd130;  // HSYNC Pulse Width
        parameter VSYNCPW = 11'd5;    // VSYNC Pulse Width
        parameter HFNPRCH = 11'd80;   // Horizontal Front Porch
        parameter VFNPRCH = 11'd3;    // Vertical Front Porch
        parameter HBKPRCH = 11'd200;  // Horizontal Back Porch
        parameter VBKPRCH = 11'd22;   // Vertical Back Porch

        //      1024x768 (pclk = 65MHz, M=13, D=20 in top_block_controller)
        /*
        wire hsync_polarity = 1'b1;       //1-Negative, 0-Positive
        wire vsync_polarity = 1'b1;

        parameter HPIXELS = 11'd1024;     // Horizontal Live Pixels
        parameter VLINES  = 11'd768;  // Vertical Live Pixels
        parameter HSYNCPW = 11'd136;  // HSYNC Pulse Width
        parameter VSYNCPW = 11'd6;    // VSYNC Pulse Width
        parameter HFNPRCH = 11'd24;   // Horizontal Front Porch
        parameter VFNPRCH = 11'd3;    // Vertical Front Porch
        parameter HBKPRCH = 11'd160;  // Horizontal Back Porch
        parameter VBKPRCH = 11'd29;   // Vertical Back Porch
        */

        // Code from this point on is from Xilinx app. note

        wire [10:0] tc_hsblnk, tc_hssync, tc_hesync, tc_heblnk, tc_vsblnk, tc_vssync,
tc_vesync, tc_veblnk;

        assign tc_hsblnk = HPIXELS - 11'd1;
        assign tc_hssync = HPIXELS - 11'd1 + HFNPRCH;
        assign tc_hesync = HPIXELS - 11'd1 + HFNPRCH + HSYNCPW;
        assign tc_heblnk = HPIXELS - 11'd1 + HFNPRCH + HSYNCPW + HBKPRCH;
        assign tc_vsblnk =  VLINES - 11'd1;
        assign tc_vssync =  VLINES - 11'd1 + VFNPRCH;
        assign tc_vesync =  VLINES - 11'd1 + VFNPRCH + VSYNCPW;
        assign tc_veblnk =  VLINES - 11'd1 + VFNPRCH + VSYNCPW + VBKPRCH;

        //////////////////////////////////////////////////////////////////////
        // VGA Timing Generator
        //////////////////////////////////////////////////////////////////////

        wire VGA_HSYNC_INT, VGA_VSYNC_INT;
        //wire [10:0] bgnd_hcount, bgnd_vcount;
        wire bgnd_hsync, bgnd_hblnk, bgnd_vsync, bgnd_vblnk;

        timing timing_inst (
                .tc_hsblnk      (tc_hsblnk),            // input
                .tc_hssync      (tc_hssync),            // input
                .tc_hesync      (tc_hesync),            // input
                .tc_heblnk      (tc_heblnk),            // input
                .hcount         (bgnd_hcount),          // output
                .hsync          (VGA_HSYNC_INT), // output
                .hblnk          (bgnd_hblnk),           // output
                .tc_vsblnk      (tc_vsblnk),            // input
                .tc_vssync      (tc_vssync),            // input
                .tc_vesync      (tc_vesync),            // input
```

```
        .tc_veblnk      (tc_veblnk),            // input
        .vcount         (bgnd_vcount),          // output
        .vsync          (VGA_VSYNC_INT), // output
        .vblnk          (bgnd_vblnk),           // output
        .restart        (reset),
        .clk                    (pclk) );

//*********************************************************************//
// V/H SYNC and DE Generator
//*********************************************************************//

assign active = !bgnd_hblnk && !bgnd_vblnk;

reg active_q;
reg vsync, hsync;
reg VGA_HSYNC, VGA_VSYNC;
reg de;

always @ (posedge pclk)
begin
        hsync <= VGA_HSYNC_INT ^ hsync_polarity ;
        vsync <= VGA_VSYNC_INT ^ vsync_polarity ;
        VGA_HSYNC <= hsync;
        VGA_VSYNC <= vsync;

        active_q <= active;
        de <= active_q;
end

//*********************************************************************//
// DVI Encoder.
//*********************************************************************//

wire [4:0] tmds_data0, tmds_data1, tmds_data2;

dvi_encoder enc0 (
        .clkin      (pclk),
        .clkx2in    (pclkx2),
        .rstin      (reset),
        .blue_din   (blue_data),
        .green_din  (green_data),
        .red_din    (red_data),
        .hsync      (VGA_HSYNC),
        .vsync      (VGA_VSYNC),
        .de         (de),
        .tmds_data0 (tmds_data0),
        .tmds_data1 (tmds_data1),
        .tmds_data2 (tmds_data2));

wire [2:0] tmdsint;

wire serdes_rst = ~bufpll_lock;

serdes_n_to_1 #(.SF(5)) oserdes0 (
        .ioclk(pclkx10),
        .serdesstrobe(serdesstrobe),
        .reset(serdes_rst),
        .gclk(pclkx2),
        .datain(tmds_data0),
        .iob_data_out(tmdsint[0])) ;
```

```
        serdes_n_to_1 #(.SF(5)) oserdes1 (
                .ioclk(pclkx10),
                .serdesstrobe(serdesstrobe),
                .reset(serdes_rst),
                .gclk(pclkx2),
                .datain(tmds_data1),
                .iob_data_out(tmdsint[1])) ;

        serdes_n_to_1 #(.SF(5)) oserdes2 (
                .ioclk(pclkx10),
                .serdesstrobe(serdesstrobe),
                .reset(serdes_rst),
                .gclk(pclkx2),
                .datain(tmds_data2),
                .iob_data_out(tmdsint[2])) ;

        OBUFDS TMDS0 (.I(tmdsint[0]), .O(TMDS[0]), .OB(TMDSB[0])) ;
        OBUFDS TMDS1 (.I(tmdsint[1]), .O(TMDS[1]), .OB(TMDSB[1])) ;
        OBUFDS TMDS2 (.I(tmdsint[2]), .O(TMDS[2]), .OB(TMDSB[2])) ;

        reg [4:0] tmdsclkint = 5'b00000;
        reg toggle = 1'b0;

        always @ (posedge pclkx2 or posedge serdes_rst) begin
                if (serdes_rst)  toggle <= 1'b0;
                else                                    toggle <= ~toggle;
        end

        always @ (posedge pclkx2) begin
                if (toggle)                     tmdsclkint <= 5'b11111;
                else                                    tmdsclkint <= 5'b00000;
        end

        wire tmdsclk;

        serdes_n_to_1 #(
                .SF             (5))
        clkout (
                .iob_data_out (tmdsclk),
                .ioclk        (pclkx10),
                .serdesstrobe (serdesstrobe),
                .gclk         (pclkx2),
                .reset        (serdes_rst),
                .datain       (tmdsclkint));

        OBUFDS TMDS3 (.I(tmdsclk), .O(TMDS[3]), .OB(TMDSB[3])) ;    // HDMI clock
output

endmodule
```

C4. Constraint File (atlys_digital_scope.ucf)

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

##########################################
# Setting VCCAUX for different SP601 board
##########################################
VCCAUX = 3.3;

#######################################
# Reset button and LEDs
#######################################
//NET "RSTBTN"   LOC = "N4";
//NET "LED<0>"   LOC = "H12";
//NET "LED<1>"   LOC = "G13";
//NET "LED<2>"   LOC = "E16";
//NET "LED<3>"   LOC = "E18";

 NET "LED<0>" LOC = "U18"; # Bank = 1, Pin name = IO_L52N_M1DQ15,   Sch name = LD0
 NET "LED<1>" LOC = "M14"; # Bank = 1, Pin name = IO_L53P,          Sch name = LD1
 NET "LED<2>" LOC = "N14"; # Bank = 1, Pin name = IO_L53N_VREF,     Sch name = LD2
 NET "LED<3>" LOC = "L14"; # Bank = 1, Pin name = IO_L61P,          Sch name = LD3

##############################################################################
# SYSCLK Input
##############################################################################

NET "SYS_CLK"    LOC = "L15";

##############################################################################
# Pushbuttons (BTN)
##############################################################################

NET "BTN<0>" LOC = "T15" | IOSTANDARD = LVCMOS33 ;  # Bank = 2, Pin name =
IO_L1N_M0_CMPMISO_2,      Sch name = M0/RESET
NET "BTN<1>" LOC = "N4"  | IOSTANDARD = LVCMOS33 ;  # Bank = 3, Pin name = IO_L1P,
                          Sch name = BTNU
NET "BTN<2>" LOC = "P4"  | IOSTANDARD = LVCMOS33 ;  # Bank = 3, Pin name = IO_L2P,
                          Sch name = BTNL
NET "BTN<3>" LOC = "P3"  | IOSTANDARD = LVCMOS33 ;  # Bank = 3, Pin name = IO_L2N,
                          Sch name = BTND
NET "BTN<4>" LOC = "F6"  | IOSTANDARD = LVCMOS33 ;  # Bank = 3, Pin name =
IO_L55P_M3A13,          Sch name = BTNR
NET "BTN<5>" LOC = "F5"  | IOSTANDARD = LVCMOS33 ;  # Bank = 3, Pin name =
IO_L55N_M3A14,          Sch name = BTNC


##############################################################################
# Mechanical Switches (SW)
##############################################################################

NET "SW<0>" LOC = "A10" | IOSTANDARD = LVCMOS33 ;  # Bank = 0, Pin name =
IO_L37N_GCLK12,           Sch name = SW0
NET "SW<1>" LOC = "D14" | IOSTANDARD = LVCMOS33 ;  # Bank = 0, Pin name =
IO_L65P_SCP3,             Sch name = SW1
NET "SW<2>" LOC = "C14" | IOSTANDARD = LVCMOS33 ;  # Bank = 0, Pin name =
IO_L65N_SCP2,             Sch name = SW2
NET "SW<3>" LOC = "P15" | IOSTANDARD = LVCMOS33 ;  # Bank = 1, Pin name =
IO_L74P_AWAKE_1,        Sch name = SW3
NET "SW<4>" LOC = "P12" | IOSTANDARD = LVCMOS33 ;  # Bank = 2, Pin name = IO_L13N_D10,
                  Sch name = SW4
```

```
NET "SW<5>" LOC = "R5"  | IOSTANDARD = LVCMOS33 ;  # Bank = 2, Pin name = IO_L48P_D7,
                  Sch name = SW5
NET "SW<6>" LOC = "T5"  | IOSTANDARD = LVCMOS33 ;  # Bank = 2, Pin name =
IO_L48N_RDWR_B_VREF_2, Sch name = SW6
NET "SW<7>" LOC = "E4"  | IOSTANDARD = LVCMOS33 ;  # Bank = 3, Pin name =
IO_L54P_M3RESET,       Sch name = SW7

##########################################################################
# VHDCI Connector (VHDCIIIO)
# Channnel 1 connects to P signals, Channel 2 to N signals
##########################################################################

 NET "VHDCIIOP_ADC1_D<7>"  LOC = "U16"; # Bank = 2,  Pin name = IO_L2P_CMPCLK,
 NET "VHDCIIOP_ADC1_D<6>"  LOC = "U15"; # Bank = 2,  Pin name = *IO_L5P,
 NET "VHDCIIOP_ADC1_D<5>"  LOC = "U13"; # Bank = 2,  Pin name = IO_L14P_D11,                Sch nam
 NET "VHDCIIOP_ADC1_D<4>"  LOC = "M11"; # Bank = 2,  Pin name = *IO_L15P,
 NET "VHDCIIOP_ADC1_D<3>"  LOC = "R11"; # Bank = 2,  Pin name = IO_L16P,
 NET "VHDCIIOP_ADC1_D<2>"  LOC = "T12"; # Bank = 2,  Pin name = *IO_L19P,
 NET "VHDCIIOP_ADC1_D<1>"  LOC = "N10"; # Bank = 2,  Pin name = *IO_L20P,
 NET "VHDCIIOP_ADC1_D<0>"  LOC = "M10"; # Bank = 2,  Pin name = *IO_L22P,
 NET "VHDCIIOP_CLKOUT"    LOC = "U11"; # Bank = 2,  Pin name = IO_L23P,
 NET "VHDCIIOP_CLKIN"     LOC = "R10"; # Bank = 2,  Pin name = IO_L29P_GCLK3,              Sch nam
 NET "VHDCIIOP_DAC_DB<7>"          LOC = "U10"; # Bank = 2,  Pin name =
IO_L30P_GCLK1_D13,                 Sch name = EXP-IO11_P
 NET "VHDCIIOP_DAC_DB<6>"          LOC = "R8";  # Bank = 2,  Pin name =
IO_L31P_GCLK31_D14,                Sch name = EXP-IO12_P
 NET "VHDCIIOP_DAC_DB<5>"          LOC = "M8";  # Bank = 2,  Pin name = *IO_L40P,
                           Sch name = EXP-IO13_P
 NET "VHDCIIOP_DAC_DB<4>"          LOC = "U8";  # Bank = 2,  Pin name = IO_L41P,
                           Sch name = EXP-IO14_P
 NET "VHDCIIOP_DAC_DB<3>"          LOC = "U7";  # Bank = 2,  Pin name = IO_L43P,
                           Sch name = EXP-IO15_P
 NET "VHDCIIOP_DAC_DB<2>"          LOC = "N7";  # Bank = 2,  Pin name = *IO_L44P,
                           Sch name = EXP-IO16_P
 NET "VHDCIIOP_DAC_DB<1>"          LOC = "T6";  # Bank = 2,  Pin name = IO_L45P,
                           Sch name = EXP-IO17_P
 NET "VHDCIIOP_DAC_DB<0>"          LOC = "R7";  # Bank = 2,  Pin name = IO_L46P,
                           Sch name = EXP-IO18_P
 NET "VHDCIIOP_DAC_WR_BAR" LOC = "N6";  # Bank = 2,  Pin name = *IO_L47P,
 NET "VHDCIIOP_TP3"                      LOC = "U5";  # Bank = 2,  Pin name =
IO_49P_D3,                              Sch name = EXP-IO20_P

 NET "VHDCIION_ADC0_D<0>"  LOC = "V16"; # Bank = 2,  Pin name = IO_L2N_CMPMOSI,             Sch nam
 NET "VHDCIION_ADC0_D<1>"  LOC = "V15"; # Bank = 2,  Pin name = *IO_L5N,
 NET "VHDCIION_ADC0_D<2>"  LOC = "V13"; # Bank = 2,  Pin name = IO_L14N_D12,                Sch nam
 NET "VHDCIION_ADC0_D<3>"  LOC = "N11"; # Bank = 2,  Pin name = *IO_L15N,
 NET "VHDCIION_ADC0_D<4>"  LOC = "T11"; # Bank = 2,  Pin name = IO_L16N_VREF,               Sch nam
 NET "VHDCIION_ADC0_D<5>"  LOC = "V12"; # Bank = 2,  Pin name = *IO_L19N,
 NET "VHDCIION_ADC0_D<6>"  LOC = "P11"; # Bank = 2,  Pin name = *IO_L20N,
 NET "VHDCIION_ADC0_D<7>"  LOC = "N9";  # Bank = 2,  Pin name = *IO_L22N,
 NET "VHDCIION_ATTEN_RLY"  LOC = "V11"; # Bank = 2,  Pin name = IO_L23N,
 NET "VHDCIION_AC_EN_RLY"          LOC = "T10"; # Bank = 2,  Pin name = IO_L29N_GCLK2,
        Sch name = EXP-IO10_N
 NET "VHDCIION_GPIO<0>"   LOC = "V10"; # Bank = 2,  Pin name = IO_L30N_GCLK0_USERCCLK,      Sch nam
 NET "VHDCIION_GPIO<1>"   LOC = "T8";  # Bank = 2,  Pin name = IO_L31N_GCLK30_D15,         Sch nam
 NET "VHDCIION_GPIO<2>"   LOC = "N8";  # Bank = 2,  Pin name = *IO_L40N,
 NET "VHDCIION_GPIO<3>"   LOC = "V8";  # Bank = 2,  Pin name = IO_L41N_VREF,               Sch nam
 NET "VHDCIION_GPIO<4>"   LOC = "V7";  # Bank = 2,  Pin name = IO_L43N,
 NET "VHDCIION_GPIO<5>"   LOC = "P8";  # Bank = 2,  Pin name = *IO_L44N,
 NET "VHDCIION_GPIO<6>"   LOC = "V6";  # Bank = 2,  Pin name = IO_L45N,
 NET "VHDCIION_GPIO<7>"   LOC = "T7";  # Bank = 2,  Pin name = IO_L46N,
```

```
 NET "VHDCIION_TP1"                          LOC = "P7";  # Bank = 2,  Pin name =
*IO_L47N,                                    Sch name = EXP-IO19_N
 NET "VHDCIION_TP2"                          LOC = "V5";  # Bank = 2,  Pin name =
IO_49N_D4,                                   Sch name = EXP-IO20_N

##############################################################################
# Debug Port # JA1
##############################################################################
//NET "DEBUG[0]" LOC = "B12" | IOSTANDARD = LVCMOS33;
//NET "DEBUG[1]" LOC = "B11" | IOSTANDARD = LVCMOS33;

##############################################################################
# DCM/PLL/BUFPLL positions
##############################################################################
#INST "PCLK_GEN_INST" LOC = "DCM_X0Y3";
#INST "PLL_OSERDES"   LOC = "PLL_ADV_X0Y1";
#INST "ioclk_buf"     LOC = "BUFPLL_X1Y0";

##########################################
# Timing Constraints
##########################################

# NET "sysclk" TNM_NET = "TNM_sysclk";
# NET "sysclk_dbnce" TNM_NET = "TNM_sysclk";
# TIMESPEC "TS_sysclk" = PERIOD "TNM_sysclk" 100MHz HIGH 50% PRIORITY 0 ;

NET "ADC_clk" TNM_NET = "TNM_ADC_clk";
TIMESPEC "TS_ADC_clk" = PERIOD "TNM_ADC_clk" 100MHz HIGH 50% PRIORITY 0 ;

NET "pclk" TNM_NET = "TNM_PCLK";
NET "pclk_fsm_1" TNM_NET = "TNM_PCLK";
NET "pclk_fsm_2" TNM_NET = "TNM_PCLK";
NET "pclk_data" TNM_NET = "TNM_PCLK";
TIMESPEC "TS_PCLK" = PERIOD "TNM_PCLK" 85 MHz HIGH 50 % PRIORITY 0 ;

NET "pclkx2" TNM_NET = "TNM_PCLKX2";
TIMESPEC "TS_PCLKX2" = PERIOD "TNM_PCLKX2" TS_PCLK * 2;

NET "pclkx10" TNM_NET = "TNM_PCLKX10";
TIMESPEC "TS_PCLKX10" = PERIOD "TNM_PCLKX10" TS_PCLK * 10;

#
# Multi-cycle paths
#
TIMEGRP "bramgrp" = RAMS(display_controller/enc0/pixel2x/dataint<*>);
TIMEGRP "fddbgrp" = FFS(display_controller/enc0/pixel2x/db<*>);
TIMEGRP "bramra" = FFS(display_controller/enc0/pixel2x/ra<*>);

TIMESPEC "TS_ramdo" = FROM "bramgrp" TO "fddbgrp" TS_PCLK;
TIMESPEC "TS_ramra" = FROM "bramra" TO "fddbgrp" TS_PCLK;

############################
# TMDS pairs on the top
############################
#NET "TMDS(0)"   LOC = "C7" | IOSTANDARD = TMDS_33 ; # Blue
#NET "TMDSB(0)"  LOC = "A7" | IOSTANDARD = TMDS_33 ;
#NET "TMDS(1)"   LOC = "D8" | IOSTANDARD = TMDS_33 ; # Red
#NET "TMDSB(1)"  LOC = "C8" | IOSTANDARD = TMDS_33 ;
#NET "TMDS(2)"   LOC = "B6" | IOSTANDARD = TMDS_33 ; # Green
#NET "TMDSB(2)"  LOC = "A6" | IOSTANDARD = TMDS_33 ;
#NET "TMDS(3)"   LOC = "B8" | IOSTANDARD = TMDS_33 ; # Clock
```

```
#NET "TMDSB(3)"  LOC = "A8" | IOSTANDARD = TMDS_33 ;

NET "TMDS(0)"    LOC = "D8" | IOSTANDARD = TMDS_33 ; # Blue
NET "TMDSB(0)"   LOC = "C8" | IOSTANDARD = TMDS_33 ;
NET "TMDS(1)"    LOC = "C7" | IOSTANDARD = TMDS_33 ; # Red
NET "TMDSB(1)"   LOC = "A7" | IOSTANDARD = TMDS_33 ;
NET "TMDS(2)"    LOC = "B8" | IOSTANDARD = TMDS_33 ; # Green
NET "TMDSB(2)"   LOC = "A8" | IOSTANDARD = TMDS_33 ;
NET "TMDS(3)"    LOC = "B6" | IOSTANDARD = TMDS_33 ; # Clock
NET "TMDSB(3)"   LOC = "A6" | IOSTANDARD = TMDS_33 ;

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