Digital Scope Implemented on Altera DE1-SoC

A Design Project Report

Presented to the

School of Electrical and Computer Engineering of Cornell University

In Partial Fulfillment of the Requirements for the Degree of

Master of Engineering, Electrical and Computer Engineering

Submitted by:

Hanchen Jin

Date Submitted:

May 15, 2016

MEng Field Advisor:

Dr. Bruce R. Land

Project Title:

Digital Scope Implemented on Altera DE1-SoC

Author:

Hanchen Jin

Program:

Master of Engineering Program, School of Electrical and Computer Engineering,
Cornell University

Abstract:

The objective of this project is to implement a digital oscilloscope based on Alter DE1-SoC board. Since DE1-SoC has a low-noise, eight-channel CMOS 12-bit analog-to-digital converter (LTC2308) with a conversion throughput (Sample Rate) up to 500Ksps, it can be an appropriate hardware platform for implementing a digital oscilloscope. Also a 15-pin D-SUB connector is available on this DE1-SoC board which is driven by the ADV7123—triple 10-bit high-speed video DAC (transforms signals from digital to analog). This DAC can support the SXGA standard (1280x1024) with signals transmitted under a speed up to 100MHz. For implementing the stable displaying, VGA mode (640x480) within a 25MHz pixel clock is chose as the display configuration for this design. The ADC values are stored into two register arrays—one for stable displaying, one for buffering ADC data for next frame; then these data are operated by FPGA logic and used to draw waveform on monitor through VGA port. Since FPGA logic is able to do parallel floating operations effectively, it can calculate the frequency of the waveform and also update the display on the monitor quickly. So this design builds a digital scope including the functions like measuring and storing voltage for DC or AC value, adjusting trigger value, adjusting horizontal position of the waveform, calculating the frequency, run/stop, reseting the whole system and selecting ADC value input channels by the combination of switches and buttons. It is mainly comprised of ADC measuring module, data storage and calculation module and VGA displaying module. All of these functions are implemented in FPGA logic.

Key words:

ADC converter, VGA display, ADC value storage, frequency calculation

Executive summary:

This design implements basic functions of an oscilloscope based on Altera DE1-SoC. The ADC value is measured by 12-bit 500Ksps ADC converter LTC2308. Then these raw ADC values are stored in register arrays. After the data processing through parallel calculation by FPGA logic, the monitor will display the waveform and its corresponding parameters through VGA port driven by triple 10-bit high speed video DAC—ADV7123. So this digital scope is able to measure DC/AC value, display waveform, detect peak voltage, calculate frequency etc. Nearly all fundamental functions of the oscilloscope are implemented in this design. Furthermore, based on the high-precision of this 12-bit ADC converter, high-speed FPGA parallel operation and appropriate calculating algorithms, this digital can even perform better than the traditional oscilloscope in the lab to some extent.

So this report is mainly about principles and details about implementing the digital scope. In the part of introduction, it covers basic background information about the hardware platform—DE1-SoC, the primary principle of ADC converter, the fundamental displaying method through VGA port and the motivation for this idea. Then in the second part, ADC data measuring module is detailed introduced including the timing requirement of LTC2308, designing and testing strategy and the configuration of channel selection. The third part comes to discuss the displaying module through VGA port. Firstly, it presents some information about the video DAC—ADV7123 related to this design. Then the content is mainly about the incremental design from displaying static image to DC/AC waveform on the monitor. Finally, it tells the implementation of some useful functions for adjusting the waveform like trigger function and horizontal position adjustment. Then next part is mainly about the ADC data processing by some specific algorithms. Data processing includes calculating the peak voltage and the frequency for AC inputs. Also the displaying strategy for the dynamic figures is introduced in this part. Then the reset function is unified to one single button for recovering the whole system. This function is important when some unpredicted phenomenon happened to the measurement and there are some tricky points there about correctly implementing it. The final part is the summary of this design and the corresponding evaluation of its display and calculations. Also some possible further design improvement/ideas are provided in this part.

Table of Contents:

1. Introduction

As the first chapter in this report, it covers general background information for implementing this design. In specific, this part is divided into three parts. The first part introduces all functions implemented in this design. Then the second part covers basic information of the DE1-SoC board, within some details about its hardware modules applied in this design like ADC converter and VGA port. And the last part introduces the motivation for this design.

1.1 Overview of the final design

The final aim of this design is to implement basic functions from the oscilloscope. Considering the available hardware resource on DE1-SoC, this design implements the functions including run&stop, voltage cursor, trigger adjustment, horizontal position adjustment, input channel selection and reset. Figure A-1 shows the User Guide interface implemented in this design which covers detailed information about controlling this digital scope.

From Figure A-1 we can learn that all ten switches and four buttons are used for controlling this digital scope. Since there is no knob on the DE1-SoC, functions like trigger voltage adjustment and horizontal position adjustment are achieved by the combination of switch and buttons.

All hardware required including ADC converter, video DAC, VGA port, switches and buttons are embedded in DE1-SoC, so finally it can be a portable and independent design. For implementing this design, three main modules are required—ADC value measurement, data storage & processing and VGA output. Figure 1-1 shows the schematic of data flow for this design.

When using or testing this design, the ADC value will be input to DE1-SoC through 2x5 pin headers; next the ADC converter will convert the analog input into 12-bit ADC values with the sample rate up to 625Ksps; then these 12-bit ADC data will be stored into register arrays and do some high-speed parallel operation to calculate some important parameters; finally the VGA module will generate output signals based on raw ADC values and calculating results. Through the video DAC, some helpful notes, waveform and parameters of waveform will be displayed on the screen. Also the reset function is used to restart the whole system when meet some accidents (e.g. measuring freeze). So key points are ADC data measurement, VGA display, data processing and reset function which are discussed in following chapter 2,3,4,5 separately.

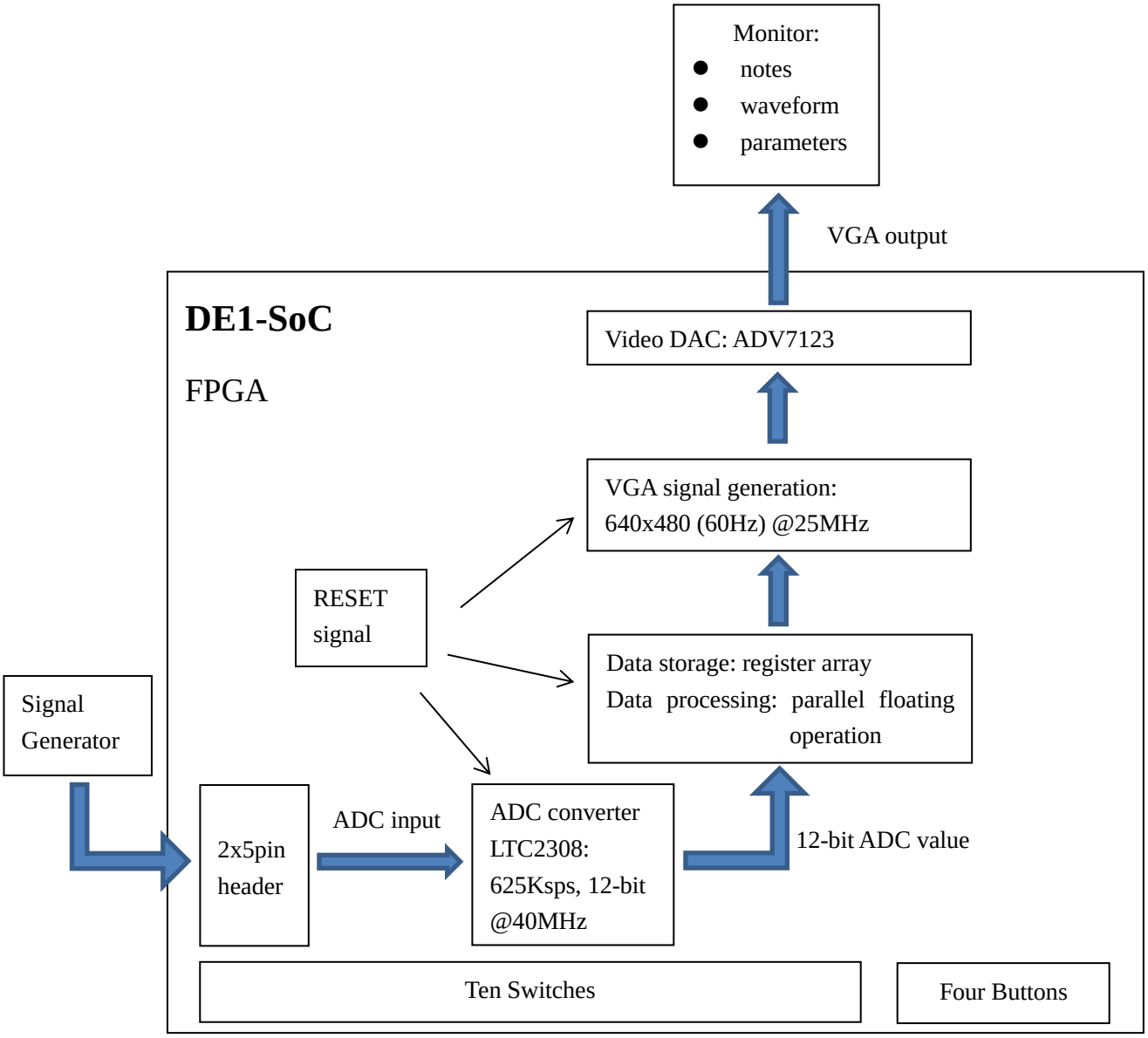

Figure 1-1: Schematic of data flow for the digital scope

1.2 Background

As an integrated design, all hardware modules required for implementing these functions are available on DE1-SoC Development Kit. Just like Figure 1-2 shows, hardware resource in DE1-SoC is abundant including high speed DDR3 memory, video audio capabilities, Ethernet port, USB port etc. This design is mainly based on FPGA logics, key parts applied in this design are the ADC headers and VGA output port. In the design, ADC module is used to transfer analog input into digital values. And VGA is a display method by drawing pictures by pixels. Switches and buttons are applied for controlling the digital scope. LEDs and seven segment LEDs are useful for testing and debugging this design. So in all, DE1-SoC is a resourceful and powerful board for implementing some interesting projects like building a digital scope.

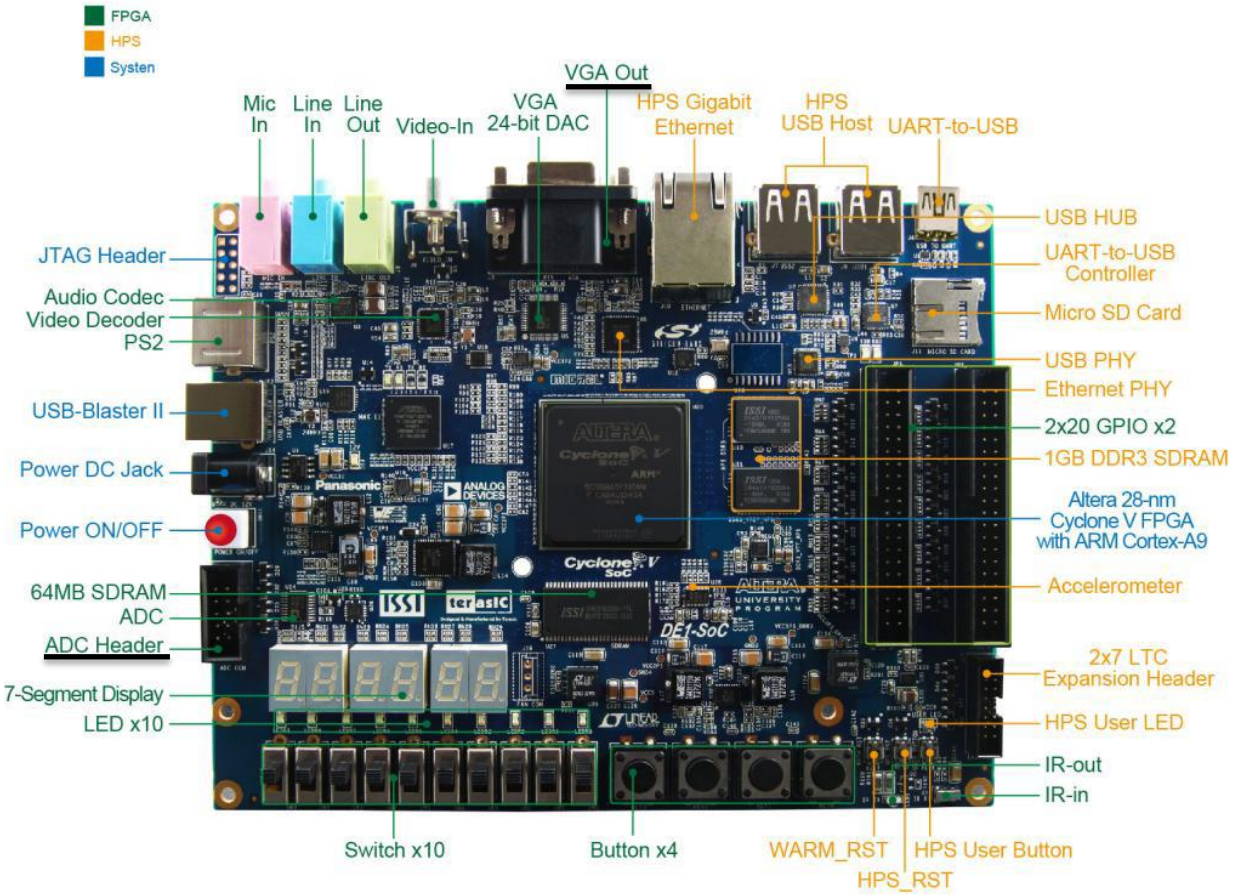

Figure 1-2: DE1-SoC development board[1]

1.3 Motivation

So based on this advanced DE1-SoC board, it is interesting and challenging to design some awesome products. According to the user manual, in this Cyclone V SoC 5SCEMA5F31 Device, there are 85K programmable logic elements, about 4450 Kbits embedded memory and up to 6 fractional PLL[1]. Based on the characteristic of parallel computing on FPGA logic, it can do heavy floating operations in a high speed. So FPGA is able to serve as the fixed-function co-processor accelerator to CPUs. But there is no free lunch in the real world—high speed parallel computation consumes much more hardware resource. But with high-density integrated FPGA logics on DE1-SoC, data processing like calculating the frequency for AC inpus can be performed precisely with a relative high speed. Therefore, with the embedded ADC module and VGA output port, it is possible to build a digital oscilloscope on it. Also many switches and buttons on the board can be used to control it. So sometimes this awesome design implemented on the small board can even replace the heavy oscilloscope in the lab!

2. ADC data measurement

For this project, getting the correct ADC value is the first milestone. So this part introduces modules for ADC data measurement. In specific, brief introduction of the embedded ADC converter (LTC2308) is in the first part; next it covers the discussion about applying and exploring this ADC converter based on its timing requirement; then design and testing strategy is presented; finally it illustrates how to configure and select different channels as the ADC value input.

2.1 ADC converter (LTC2308)

The embedded ADC converter—LTC2308 is a low noise, 500Ksps, 8-channel, 12-bit ADC module. Its internal conversion clock enables the external serial output data clock (SCK) up to operate at the frequency up to 40MHz. It operates from a single 5V supply and draws just 3.5mA at a sample rate of 500Ksps. It has internal 2.5V reference with 8-channel multiplexer to select input channel.

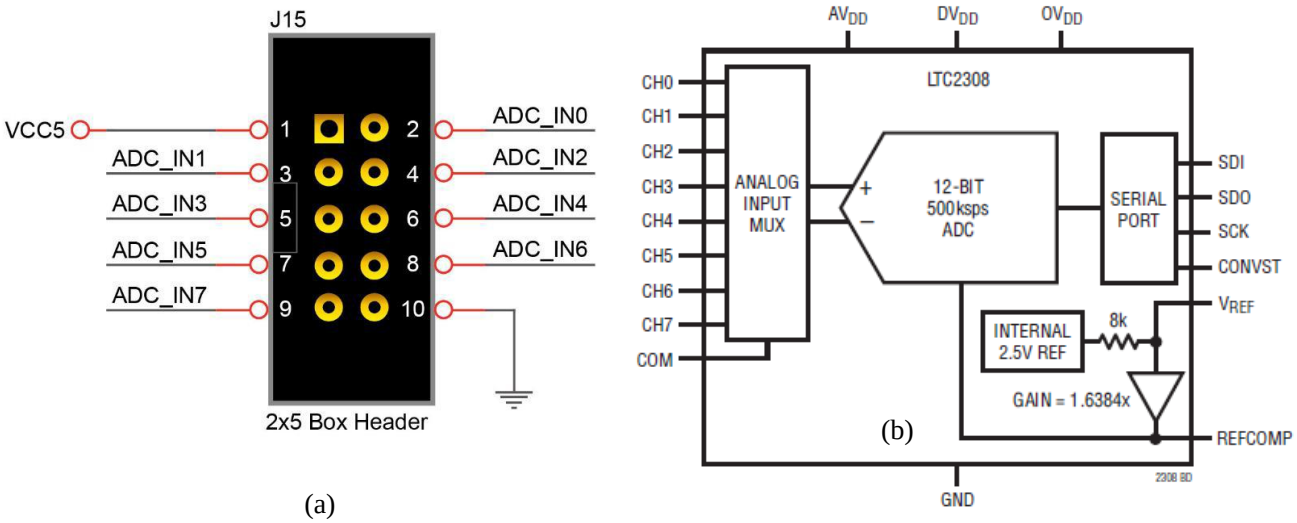

Figure 2-1: Signals of the 2x5 Header[1] (a) Block diagram for LTC2308[4] (b)

Just as Figure 2-1 shows, 8 pin headers marked ADC_IN0~7 (a) corresponds 8 channels Ch0~7 (b).The analog input mux will choose one channel as input. In LTC2308, under the control of configuration signals and internal clock, 12-bit ADC data can be measured within a sample rate 500Ksps.

2.2 Discussion on Timing requirement

So this part covers more details with timing issues which is essential for the configuration of ADC converter. Basically LTC2308 is able to work under 500Ksps, but from its data sheet, it seems that users can explore its performance by changing some default settings.

The LTC2308 has two typical converting modes with different CONVST pulse. For higher sample rate to acquire more ADC values, the converting with short CONVST pulse is selected. As Figure 2-2 shows, time required for measuring one ADC data is this:

$$t_{cyc} = t_{CONV} + t_{WLCONVST} = 1.3 + 0.41 = 1.71\text{s}$$

The possible minimal values are provided in Figure 2-3. The typical conversion time is 1.3μs which is also the minimum one. But the CONVEST low time which is used to output 12-bit ADC data seems longer than it required. Theoretically period of twelve clock cycles (SCK) is enough to output the serious data (SDO). So this is the minimum sample period for this ADC converter in theory:

Minimum t_{cyc}= t_{CONV}+12t_{SCK} = 1.3 + 12x1/40M=1.6μs

Highest sample Rate: 1/t_{cyc} = 625Ksps

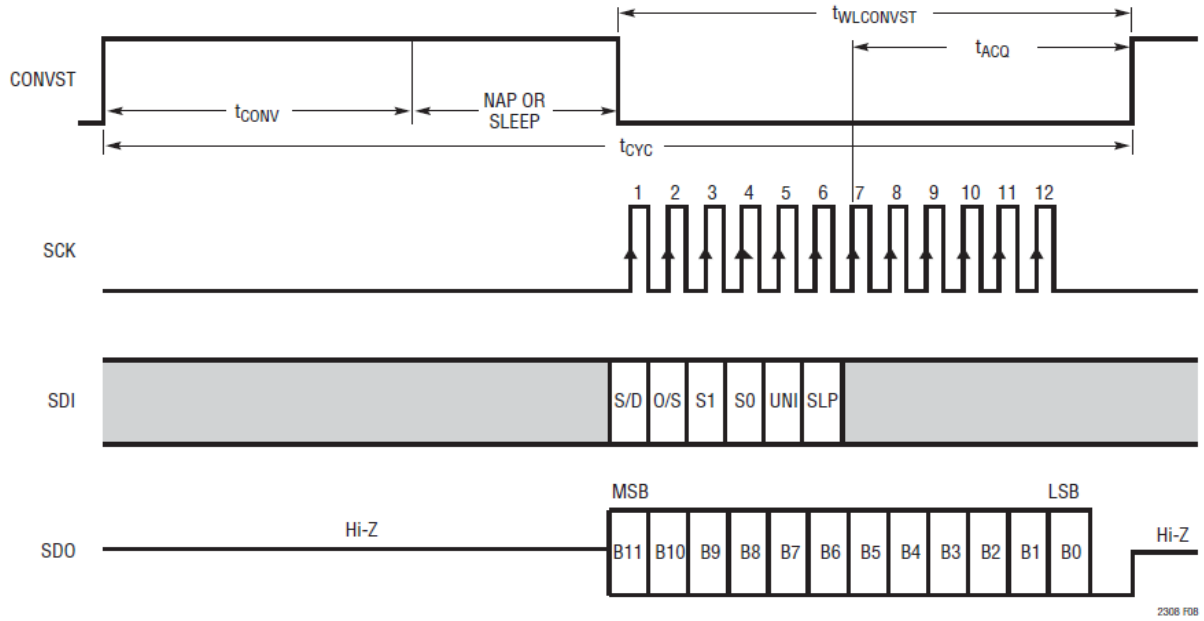

Figure 2-2: LTC2308 Timing with a long CONVST Pulse[4]

SYMBOL	PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
$f_{SMPL(MAX)}$	Maximum Sampling Frequency		●			500	kHz
f_{SCK}	Shift Clock Frequency		●			40	MHz
t_{WHCONV}	CONVST High Time	(Note 9)	●	20			ns
t_{HD}	Hold Time SDI After SCK↑		●	2.5			ns
t_{SUDI}	Setup Time SDI Valid Before SCK↑		●	0			ns
t_{WHCLK}	SCK High Time	$f_{SCK} = f_{SCK(MAX)}$	●	10			ns
t_{WLCLK}	SCK Low Time	$f_{SCK} = f_{SCK(MAX)}$	●	10			ns
$t_{WLCONVST}$	CONVST Low Time During Data Transfer	(Note 9)	●	410			ns
$t_{HCONVST}$	Hold Time CONVST Low After Last SCK↓	(Note 9)	●	20			ns
t_{CONV}	Conversion Time		●		1.3	1.6	μs
t_{ACQ}	Acquisition Time	7th SCK↑ to CONVST↑ (Note 9)	●	240			ns
$t_{REFWAKE}$	REFCOMP Wakeup Time (Note 12)	$C_{REFCOMP}$ = 10μF, C_{REF} = 2.2μF			200		ms
t_{dDO}	SDO Data Valid After SCK↓	C_L = 25pF (Note 9)	●		10.8	12.5	ns
t_{hDO}	SDO Hold Time After SCK↓	C_L = 25pF	●	4			ns
t_{en}	SDO Valid After CONVST↓	C_L = 25pF	●		11	15	ns
t_{dis}	Bus Relinquish Time	C_L = 25pF	●		11	15	ns
t_r	SDO Rise Time	C_L = 25pF			4		ns
t_f	SDO Fall Time	C_L = 25pF			4		ns
t_{CYC}	Total Cycle Time				2		μs

Figure 2-3: LTC2308 Timing parameters[4]

2.3 Design and Test

There is an awesome example provided in DE1-SoC CD which measures the ADC data and output them by a Nios II window. But it is too complex for this design. So I started to read this complicated example and try to get rid of redundant modules to save hardware resources for my further design.

Firstly the software core Nios II is useless for my design. So I just disabled the Nios II connection with the Qsys GUI interface because that took some hardware resource. For Qsys reset and switches control module, I just left it there but without connection to real signals. PLL module is essential for generating the reference clock (SCK) during serialization. Also since one PLL is able to generate three different stable clocks, besides the ADC clock (40MHz), it provides reference clock for VGA module (25MHz) as well.

Then the key point is to modify the state machine in adc_ltc2308.v to enable continuous measurement. The basic state machine moves among three states: measuring start, waiting measuring done, storing them into FIFO. For my design, only the raw ADC values are useful. So the method is waiting for the flag wait_measure_done. When this flag is "0", it means the measurement of ADC data is done. So under that condition, it is safe to store the available raw ADC data into another register for further usage. So in the bottom module, I build a sequential circuit under the control of ADC clock (40MHz) to store data into a register called ADCout under the condition that wait_measure_done is "0". Then transfer this value to other modules for processing.

Another problem here is that for exploring the performance of this ADC converter, manually reset is required for restarting the ADC converter. As showed in Figure 2-2, after 1.6μs, the ADC data is ready and has been saved in the register. At that time just reset the ADC converter to let it measure next value, thus increase the sample rate to 625Ksps. So a counter is built in the top level to count the time and generate the reset signal every 1.6μs.

For verifying the ADC value, seven segment LEDs are applied for displaying it. Firstly, a module which transfers binary value to decimal value is required. In this module, circuits include dividers and remainders are used to calculate four digits in decimal. Then this decimal value is transmitted to seven segment LEDs displaying module. After successfully compilation and configuration, input DC value by DC voltage source into channel 0 (default setting), the correct DC voltage is displayed on seven segment LEDs (Figure 2-4). If input voltage is changed, the displays value will also change quickly.

2.4 Channel selection

Since channel 0 has been successfully applied to measuring data, in this section, it will cover detailed about configuration of enabling other seven channels. As Figure 2-5 shows, the embedded ADC converter's eight channels have been connected to 2x5 pin headers (Figure 2-1-a) with RC filter circuit to eliminate the high-frequency noise. In the LTC2308, an eight to one multiplexer is applied to choose the input channel. So from the programmer's perspective, only a three-bit register called measure_ch is visible for channel selection.

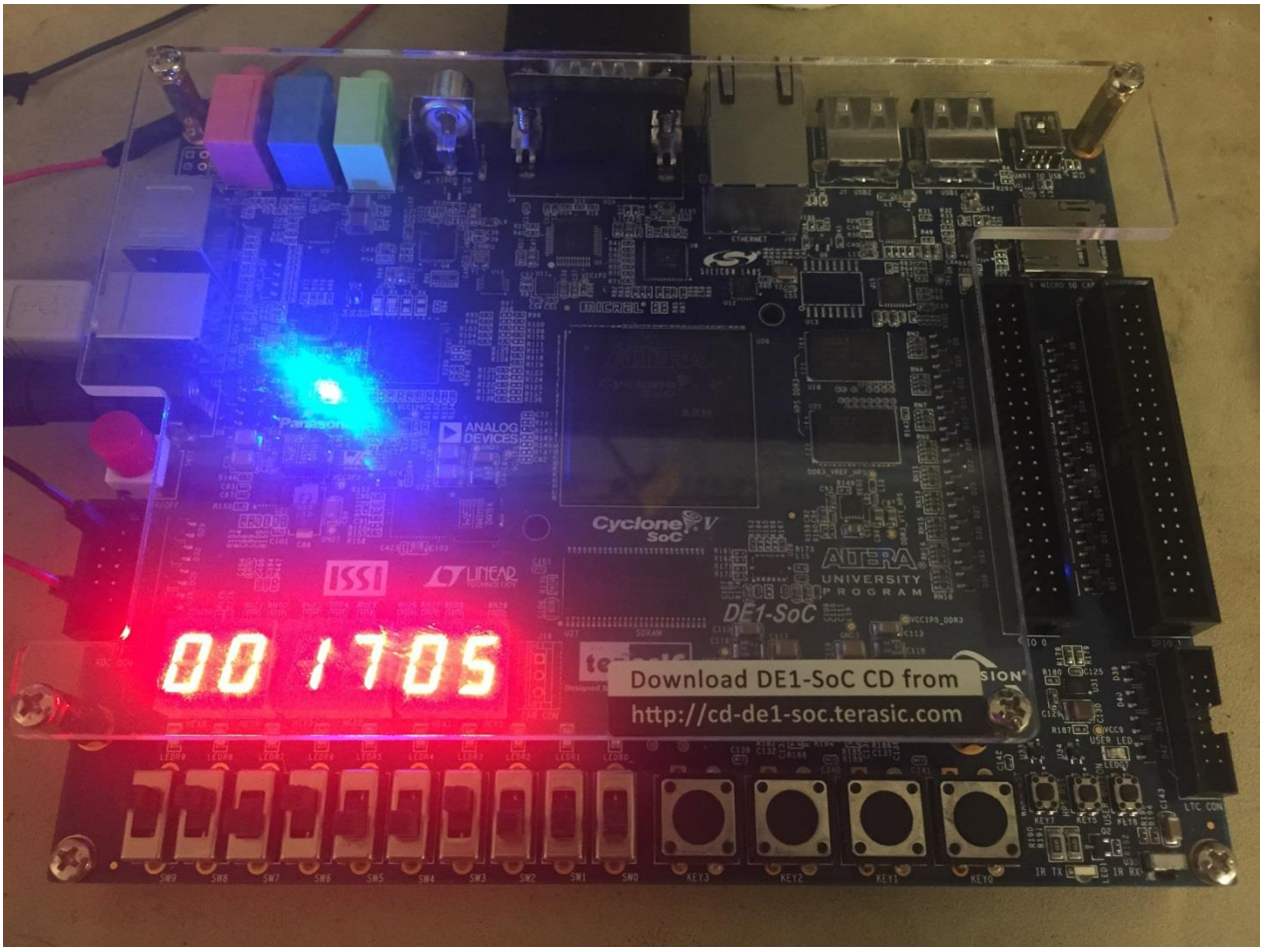

Figure 2-4: diagram with DC value displayed by seven segment LEDs

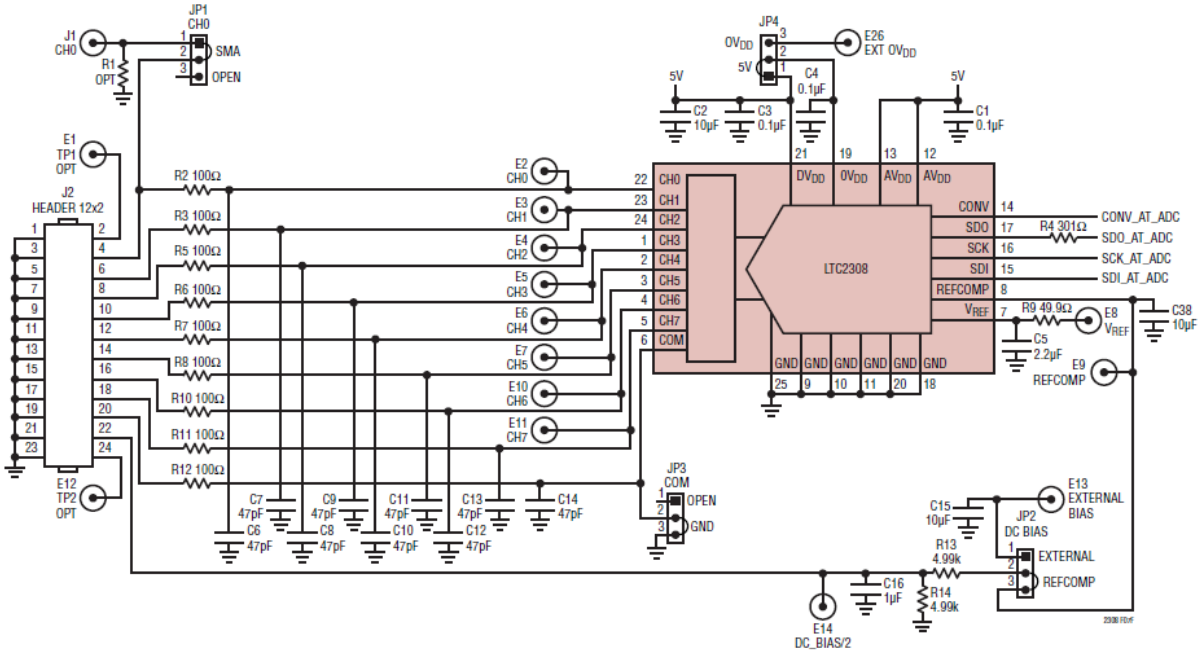

Figure 2-5: LTC2308 connecting circuit[4]

The first method to apply these eight channels is to measure values in a polling way. That means the setting of measure_ch will be changed with an interval of sampling time. But the key problem is that the actual sample rate for each channel will be divided by eight since eight channel are sharing one ADC converter. Considering the sample rate for LTC2308 can only up to 625Ksps, for precisely measuring waveform with high frequency, the idea for this design is to select one ADC channel as the input; then continuously measuring ADC value from that one channel within full speed (625Ksps). And switch 0,1,2 are applied to select different channels. From the hardware level, these three switches are connected to the three bits register which selects one channel from the 8 to 1 multiplexer as input.

So now the ADC module has been successfully built and verified, then these ADC values can be stored into FPGA register arrays. Next milestone is the display of the waveform through VGA port which will be introduced in next chapter.

3. Basic VGA display

Since the ADC value is ready for usage, next milestone is to draw the waveform on the monitor through VGA port. So this chapter covers the basic knowledge of displaying through VGA port, the process of displaying static images and dynamic waveforms. Furthermore, functions like trigger and horizontal position adjustment which relate to basic display strategies are also discussed in this chapter.

3.1 VGA timing specification

Before starting practice, it is better to figure out the timing issues on VGA display. Firstly, I pick up the VGA mode showed in Figure 3-1. It is a selection from VGA timing specification with 60Hz update rate (one second 60 frames) under the control of a 25MHz Pixel clock. Figure 3-1 shows the same timing specification with different directions. (a) shows the horizontal timing of each state inμs; (b) defines the vertical timing which applies horizontal lines as basic unit.

(a)

VGA mode		Horizontal Timing Spec				
Configuration	Resolution(HxV)	a(us)	b(us)	c(us)	d(us)	Pixel clock(MHz)
VGA(60Hz)	640x480	3.8	1.9	25.4	0.6	25

(b)

Configuration	Resolution(HxV)	a(lines)	b(lines)	c(lines)	d(lines)	Pixel clock(MHz)
VGA(60Hz)	640x480	2	33	480	10	25

Figure 3-1: VGA vertical timing selected in this design[1]

So the basic idea for VGA display is to draw 640x480 pixels for one picture, then update the display in a speed of 60Hz for playing videos. As for one single picture, displaying method is to draw 480 rows from the top of the screen to its bottom within 640 pixels for each row. The process of drawing each row follows the rule of horizontal timing specification (Figure 3-2).

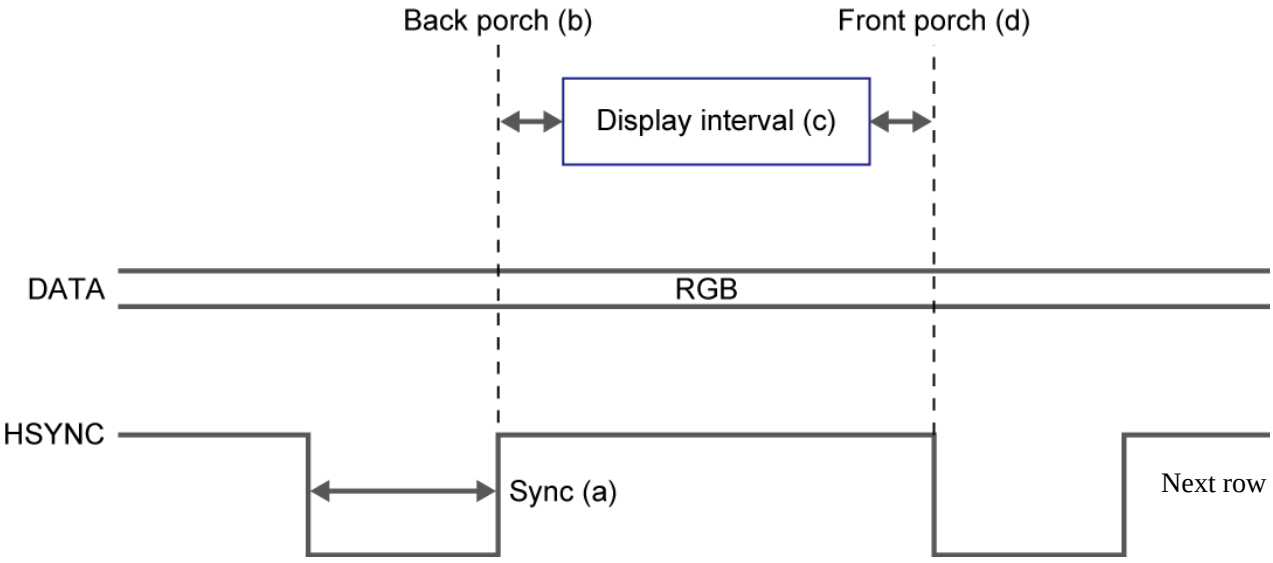

Figure 3-2: VGA Horizontal Timing Specification[1]

As Figure 3-4 shows, (a) is called horizontal synchronization period; (b) is the horizontal back period; (c) is the visible area when RGB data is transmitted; (d) is the front area. The basic structure of timing specification of vertical (Figure 3-3) is nearly the same with horizontal one. But in display interval, it should display 480 lines for one frame. After displaying the whole picture, it draws next frame immediately for playing videos. So for displaying one line, 800 pixel cycles are required; for one frame, 525 lines are required, time for one frame is like this:

$$t_{frame} = N_{lines} \times N_{cyles/line} \times t_{pixel\ clock} = 525 \times 800 \times 1/25M = 0.0168s$$

So the update rate is:

$$f_{update\ rate} = 1/t_{frame} = 1/0.0168 \approx 60Hz$$

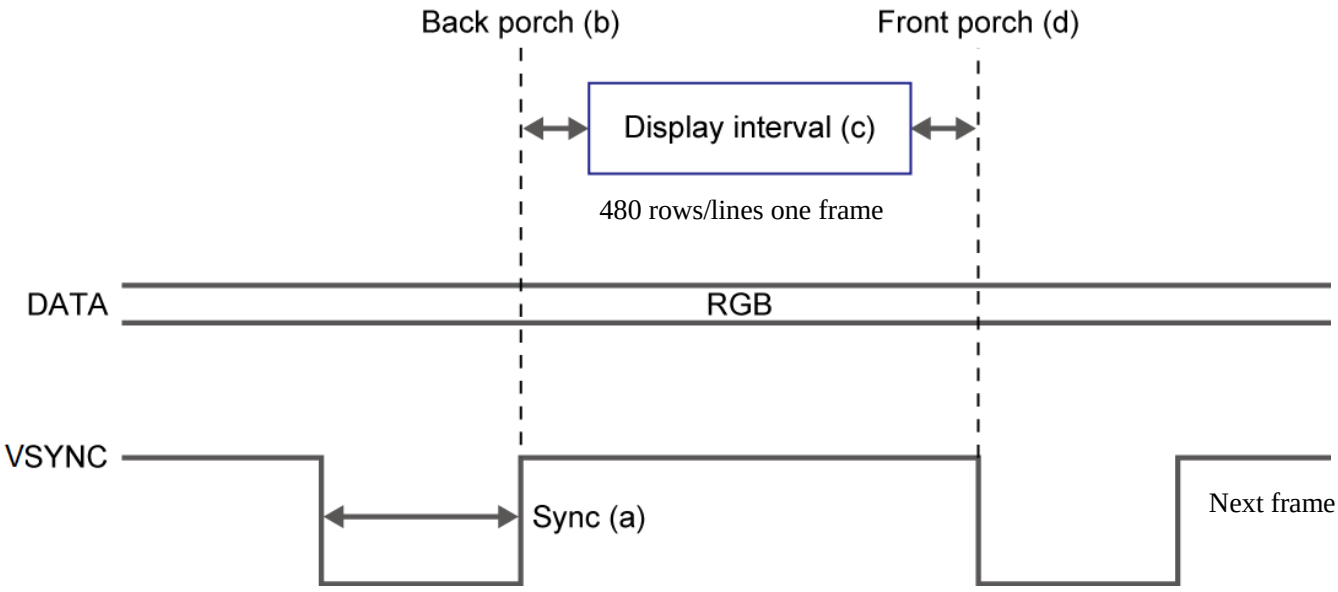

Figure 3-3: VGA Vertical Timing Specification[1]

3.2 VGA display testing—colored bar

As introduced before, the VGA port required a video DAC to drive it. Video DAC ADV7123 is embedded in the board and the connection between this video DAC and VGA pins are fixed connection in the board so designers should not worry too much about that. The basic connection from FPGA logic and VGA port is showed in Figure 3-4.

Then to get things started, I start to build a simple test project which displays colored bar on the monitor. So I need a control module to generate HSYNC and VSYNS signals under the control of pixel clock according to the standard VGA timing specification introduced before. The pixel clock (25MHz) can be generated by PLL used in ADC module. Sequential circuits with counter are able to continuously produce correct HSYNC and VSYNS control signals. At the same time, the coordinate of pixels are stored into registers called iCoord_X and iCoord_Y. These output coordinates determines 24-bit RGB values. So for testing my understanding, I built a simple design which outputs colors bars on the monitor (Figure 3-5). An important note here, since all signals in VGA display module should

follow the strict timing specification. All circuits in this module should be sequential circuits within the control of the pixel clock.

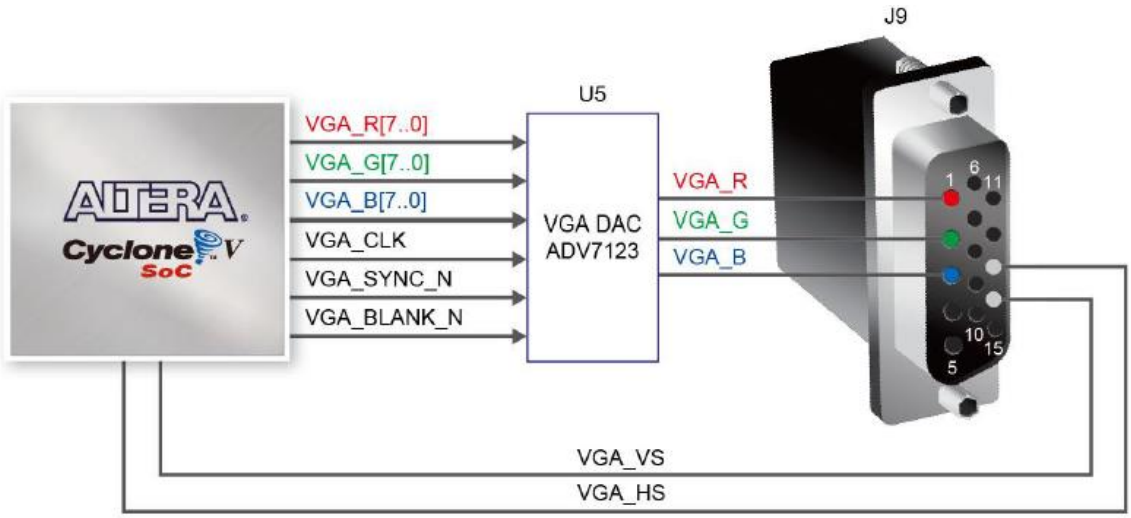

Figure 3-4: Connections between the FPGA and VGA[1]

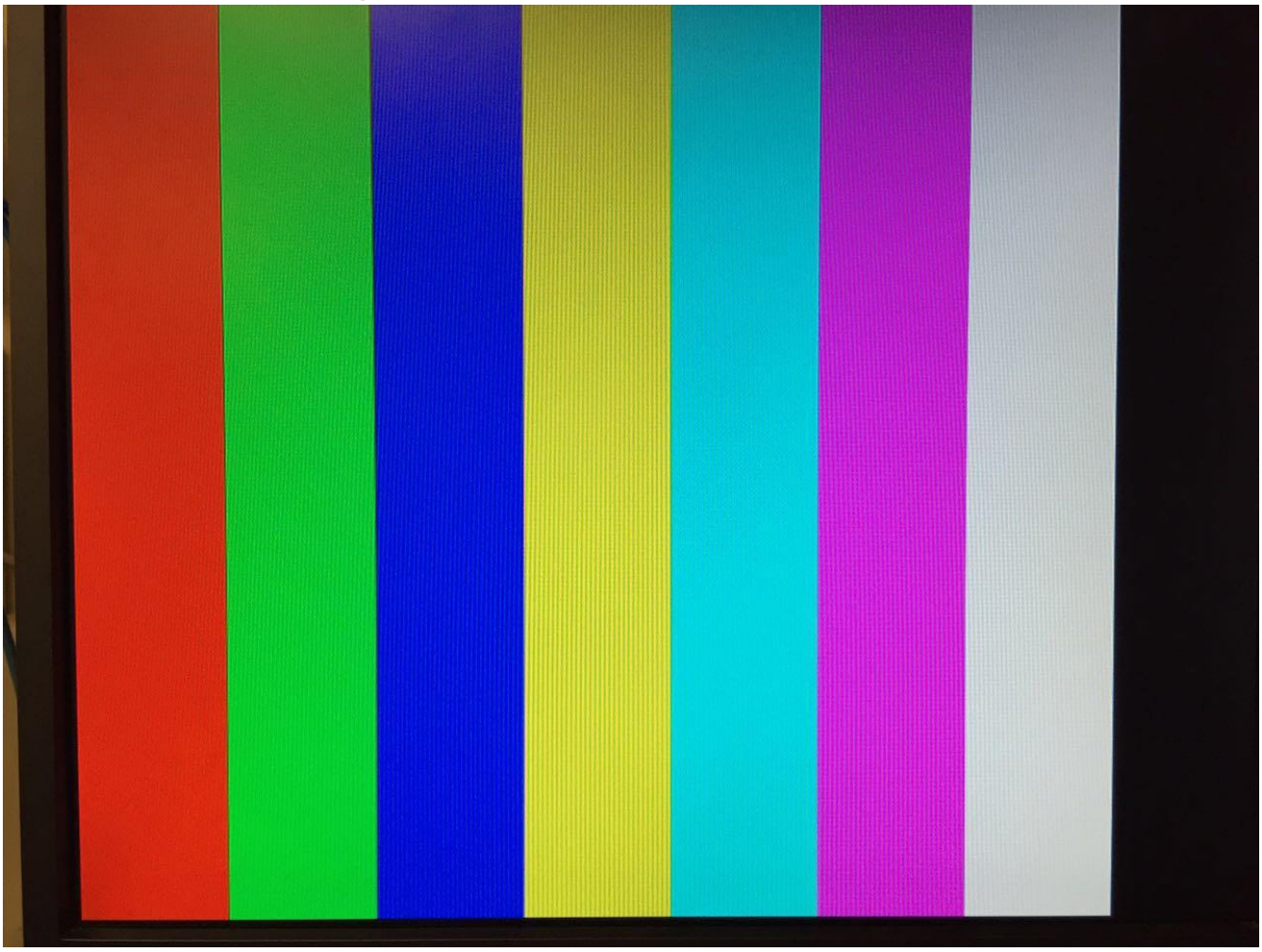

Figure 3-5: Simple test design—colored bar

3.3 Displaying static image—notes

So now I start to design useful functions for my digital scope. Considering method of incremental design, I decide to firstly display some static notes including the information of measuring range and the coordinate axis. The strategy of image displaying is a little different. The format of VGA output is 640x480, so it is better to make a reasonable plan for exploiting this limited displaying area. Figure 3-6 is the design of displaying plan. Since the measuring range for ADC converter is 0 to 4.096 volt which means the output of 12-bit ADC values range from 0 to 4096. So it is proper to just scale it to 0~410 for drawing waveform on the monitor (area 4). The function of coordinate axis (area 3) is in the left side of the screen which can not only help users to read the voltage of the input waveform, but also reminds users the measuring range of this design from a graphic method. The notes in area 1 provide the information of measuring range and enable this design seems much more formal. Area 2 is reserved for displaying important values of the waveform, detailed discussion about this will be in Chapter 4.

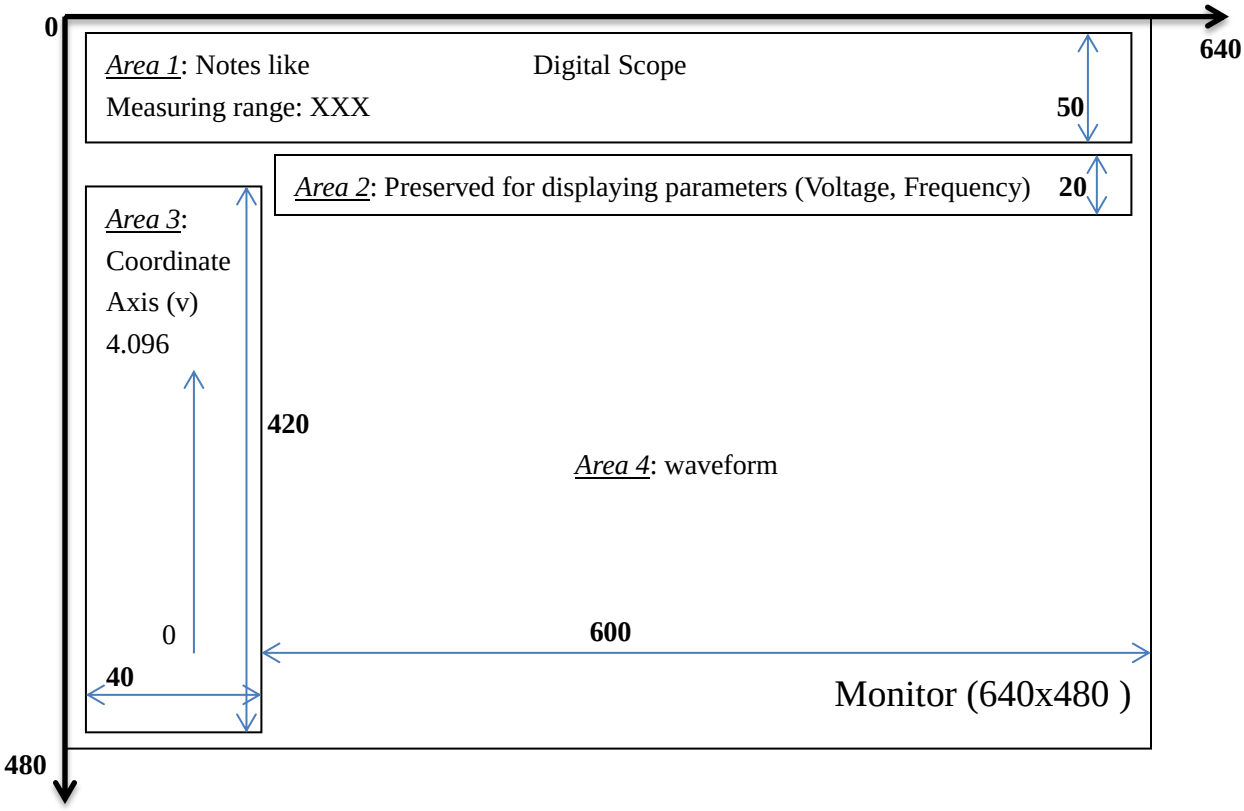

Figure 3-6: Displaying plan on the monitor

As we know, VGA output draw output pictures by pixels. So for displaying note in a specific area, font library should be built including 26 letters and other useful signals. Then designers can choose to display the message they want. Actually I can do that but it is really a huge work. More importantly, for these static pictures, what I can do is just draw a 640x50 picture and then display them on the screen because no changes for this part. So key points here are building specific pictures and displaying them.

For area 1, I have to build a picture with a size of 640x50. "*Paint*" is an awesome software for building

this. Resize the sketchpad to 640x50, then write notes on it within a proper font. But the building of coordinate axis is not an easy work. Since the coordinate has to be used to reflect true ADC values. So the calibrations on the coordinate axis should be drawn strictly according to its position—pixels on the picture. Fortunately, *Paint* will tell users the position of their mouse based on pixels. So by carefully following the pixel information about the mouse provided in "*Paint*", I can put each calibration in the right place of this coordinate axis. Then the key point is to transform this picture into pixel values for VGA display.

A software called "BmpToMif" resolves this problem. Actually it can only change .bmp file to .mif file MIF file contains initial values for ROM. This software can extract each pixel of a picture and generate a MIF file for configuring memory which meets the requirement for my design. For saving some FGPA logic, I build each MIF file with a width of one. Then I build ROMs for storing pixel data of these notes; also ROM module called index_ROM is built which transforms one bit pixel value to 24-bit VGA output. According to the coordinate generated by VGA timing specification, pixel values are extracted from ROM to draw notes on the monitor.

3.4 Displaying DC waveform

Then I start to display the DC waveform. Since the correct ADC values have been stored into a register, what I should do here is to create a display buffer which stores one tenth of the ADC values (scale the raw ADC value 0~4096 to 0~410). Then when the Y-axis (iCoord_Y) of output equals that value, it output white on the screen. Just like Figure 3-7 shows, the white line on the screen reflects the input value. The measured value is 3V which is really precise compared with input value (3.02V). By the way, the white line pointed to 4.1V provides users a feeling of maximum measurable voltage.

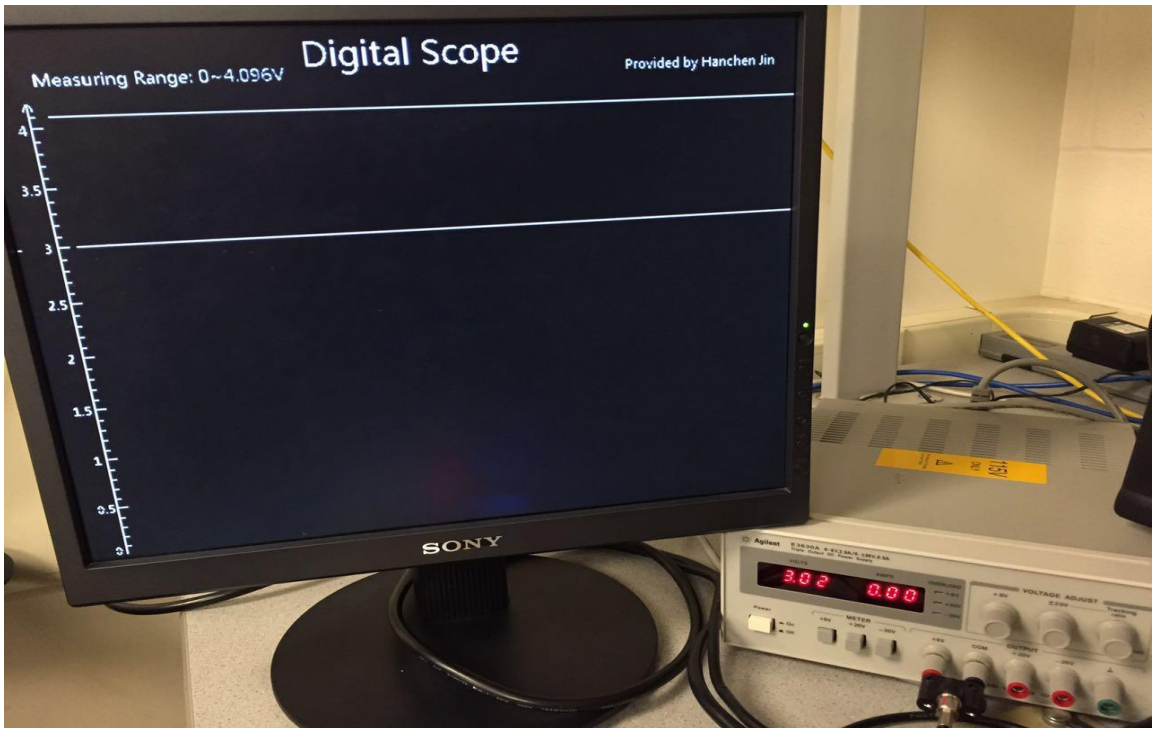

Figure 3-7: Displaying DC values on the monitor

3.5 Displaying AC waveform

Now the key point is to draw AC waveform on the screen. The basic idea is same as drawing DC lines. Firstly build a 12-bit 640 elements register array for storing enough ADC values for one frame. Then I store ADC values with a method of circular storage. Within the sample rate of 625Ksps, data in this register array will be updated every 1.024ms. On displaying control part, X coordinate (iCoord_X) is used as index to access ADC value register array. When this value equals iCoord_Y, this point will be drawn white. So the final waveform is composed of such 640 points. It is a pretty straightforward idea but actually the waveform on the screen now is unstable—it oscillates violently. The reason is obvious, the ADC values stored in the register array are not the same every time and also the display for one frame is much longer than the update rate of register array, so the display is messy. Only when the waveform with a frequency about 980Hz (nearly the same with update rate for register array), the display is a good waveform sine under this condition the data stored in register array are nearly the same every time. So the trigger should be added for displaying stable waveform under any frequencies.

3.6 Triggering function

The trigger function implemented in this design is follows this basic thoughts—firstly choose a specific value; then when the input hit it, start storing ADC values into register array so that the first ADC values stores into register arrays is nearly the same. In specific, the first state detects the current mode selection—"AC" mode or "DC" mode. If the current input mode is selected to "DC", the state machine will jump to an idle state, do nothing then jump back to the mode detection state. If the current input mode is selected to "AC", it will pick one memory and prepare to store data in it. For more stable display, I implement two register arrays, one for storage and one for display—mem0 and mem1 in the state machine. The mem0 and mem1 represent two register arrays, they have the same size 640x12-bit. So when mem0 is updating its data, display module will pick static data stored in mem1; and when mem0 finish the process of data updating, display module start to pick data from mem0 and update the ADC value in mem1 at the same time. The default settings for memory selection is mem0, which means at first cycle, ADC values will be stored in mem0 and the display module will draw waveform by data in mem1. But now there is no valuable data in mem1, there is no display on the screen. After about 1ms, 640 ADC values have been stored in mem0 then the waveform will be displayed on the monitor. So after user input values they desire to be measured, the delay will be about 1ms because of the latency for acquiring enough ADC values for first frame.

Every time when the state machine select one register array to store data in, it will go down to the corresponding branch. The states in these two branches are the same because they perform the same action but only store data in different register arrays. For example, just assume the state machine is storing data into mem0. Firstly the state machine will stay in this state until the ADC value measured is equal to the trigger value. Then it will jump to next state which judges whether this value is in the rising edge. If the answer is no, it will jump back to the state of waiting value again to wait for the correct value. If the answer is yes, it will start to store data into mem0. When the memory is full, it will jump to the state which clears all registers and flags applied during the storing process. Then jump to the initial state for next cycle of storing. So with this trigger function, waveform with different frequencies can be stably drawn on the screen (Figure 3-8).

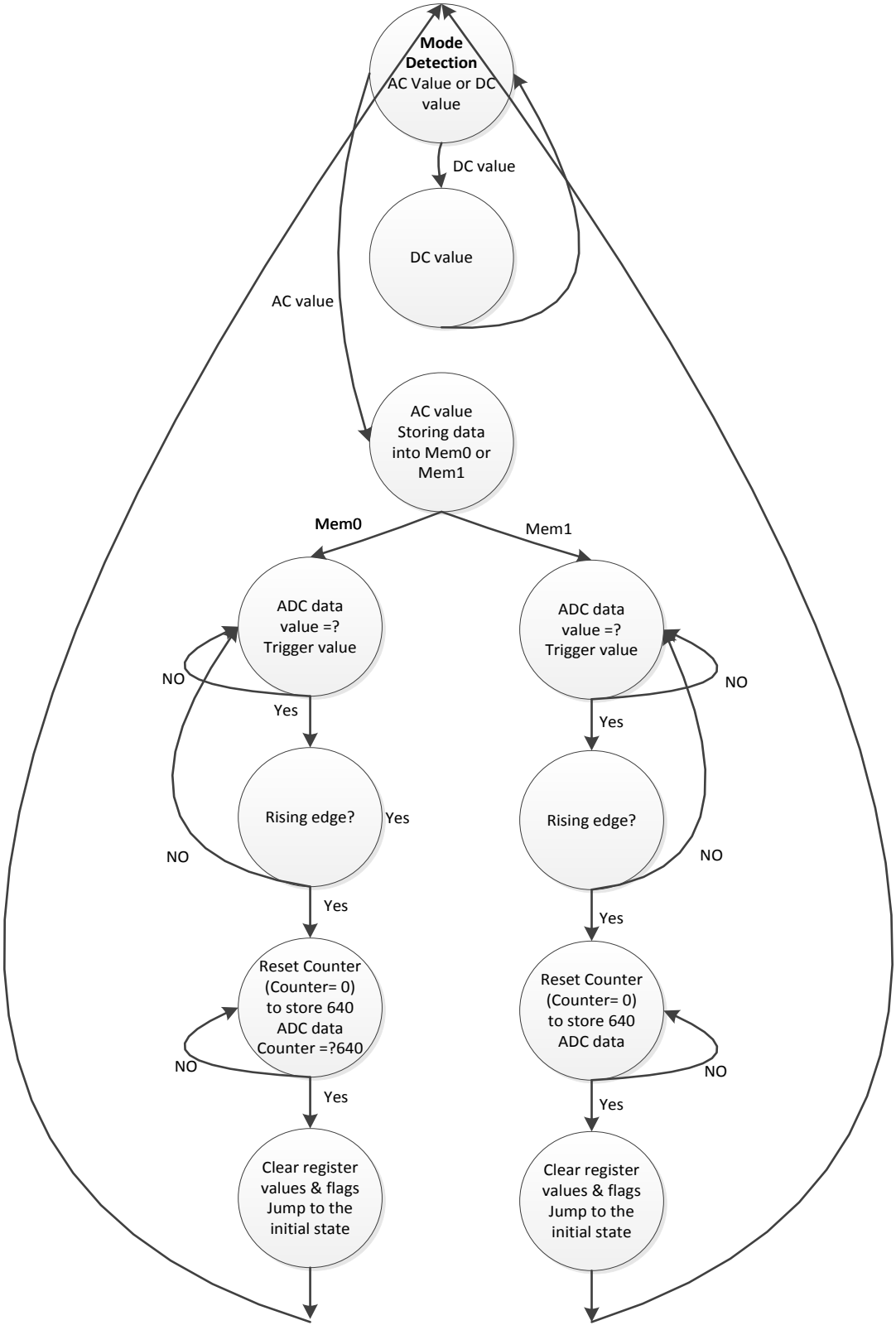

Figure 3-8: State machine for trigger function

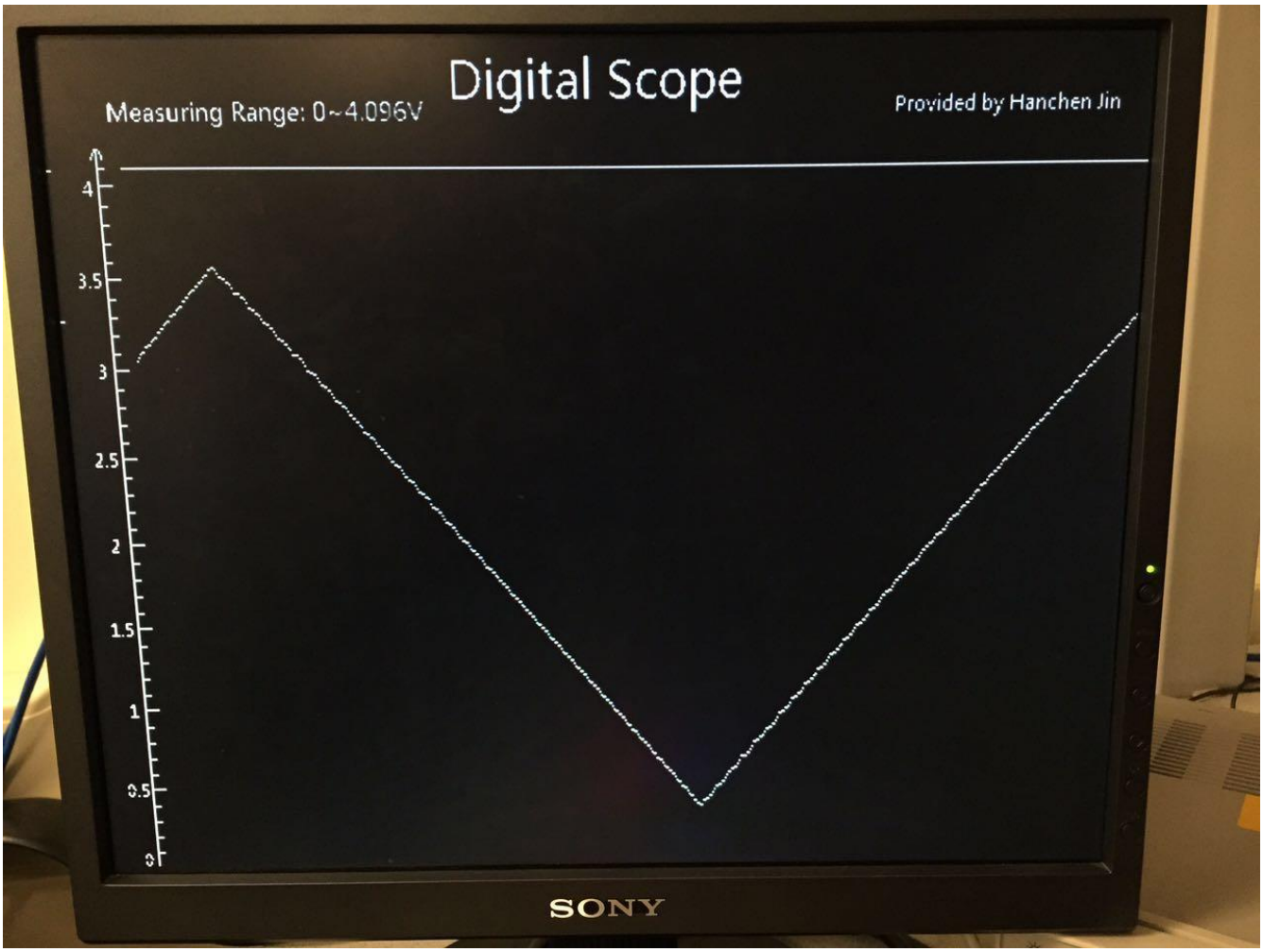

Figure 3-9: Stable AC value display—triangular waveform

Since the trigger value is important to get stable waveforms, the function of trigger voltage adjustment is essential for this design. Suppose if the default trigger value (2V) was out the range of input waveform's peak voltages, none of ADC values will be stored in register arrays (mem0&mem1) thus no waveform displayed on the screen. So firstly, I choose one switch to enable the display of trigger voltage on the screen—one horizontal line and the trigger voltage can be read according to the coordinate axis. Then I choose two push buttons as input. Because of the high speed of inner clock, one press is recognized as multiple presses. So the debouncing should be added for pressing buttons. The state machine for this function is simple—one initialized state (idle) for detecting button presses, next jump to the state of altering the trigger voltage, then jump to waiting state to wait for about 0.2s as bouncing time (counting clock as timer), finally return back to idle state for next input. So now when user presses button for one time, it will change the trigger voltage for 0.01 volt (for sensitive trigger voltage adjustment); and when user holds the button, the trigger voltage can be changed continuously in a speed of approximately 0.05v/s. It is worth to mention that in my design, users can only change the trigger voltage when it is visible (the horizontal line).This is quite reasonable because it is sensible to change trigger voltage according to its value and the measured value.

3.7 Horizontal position/time scale adjustment

After the trigger function has been successfully implemented, the displaying waveform has been quite stable. However, when measuring waveform with higher frequency more than 10KHz, there are too many cycles of waveform in one screen which makes the display a little messy. So the function like adjusting the waveform's horizontal position is required. Here the horizontal position adjustment is like the function of adjusting the time scale.

Due to the constraint of the embedded ADC converter LTC2308's sample rate, it is impossible to get more data from higher frequency input for drawing better waveform. So the basic idea is to enlarge one piece of the original waveform to get a better visual effect. Under this condition, only a part of ADC values stored in register array is applied for drawing the waveform. Since there is no knob on the board, I applied one switch and two buttons to simulate the knob. A register named hortizontal_multiple is implemented to store the scale of enlarging the display. For example, if hortizontal_multiple is two, that means half value stored in the register array is used to draw waveform on the whole screen. Also two buttons with debouncing are able to alter the scale to achieve different kind of enlarging.

Figure 3-10 shows the display within the function of horizontal position adjustment. From this picture we can learn that if the input waveform's frequency became larger, the ADC values measured in one cycle became less because of the limited sample rate. Under this condition, the frequency calculated and displayed in this picture also has a larger error. Details about it will be covered in next chapter.

The same kind of problem happens when measuring waveform with lower frequency. When the frequency is lower than 1KHz, the 640 ADC values stored in the register array can only constitute half cycle of the waveform or less. Referred the method of horizontal position adjustment for larger frequencies, I also combine one switches and two buttons for this control. Push buttons on DE1-SoC are limited so I apply switches to enable different functions. The basic idea is to lower the storing speed. The sample rate of the ADC converter remains the same, I just slow down the speed of storing these values into register arrays. I build a counter and apply the variable hortizontal_multiple as threshold. Before I build this function, the data will be stored into register array within the same speed of ADC values' measuring. And now with the variable hortizontal_multiple, the speed of storing process will become 1/(hortizontal_multiple+1) of the normal storing speed (same with sample rate 625Ksps). For example, if the value of hortizontal_multiple is two, the counter will start count from zero under the condition that ADC value is available, only when the counter reaches two, the ADC value measured will be stored in the register array for drawing waveform. So under this condition, only one ADC value from a serious of measured ADC values will be stored in the register array. So finally the 640 values in the register array can draw a better waveform (within one or two cycles of the input waveform).

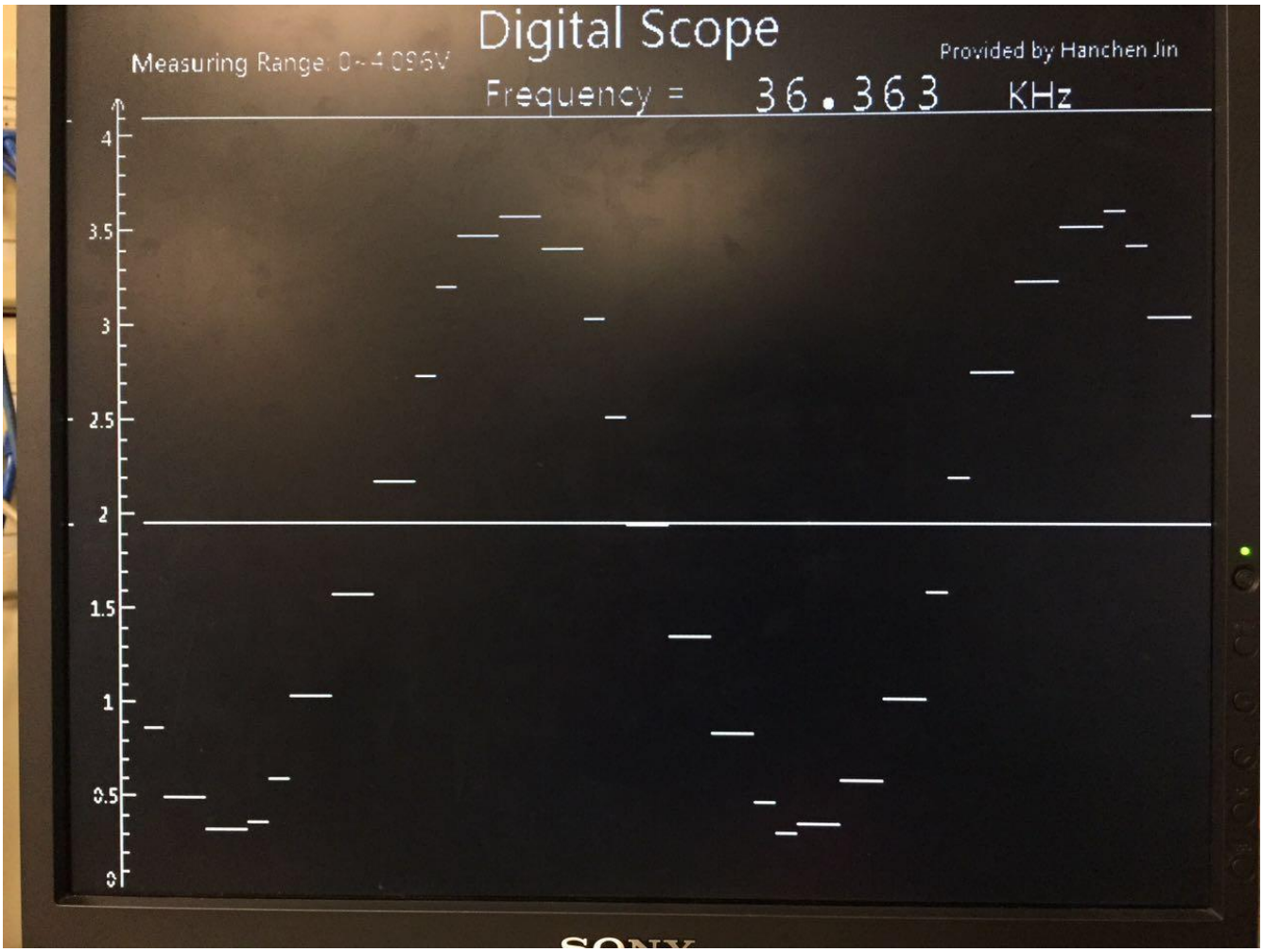

Figure 3-10: Waveform with higher frequency

4. Data processing & displaying

So in this chapter, it firstly introduces details about the strategy of displaying DC voltages. Next it covers an algorithm is applied to find the peak voltage in AC values. Then the implementation of calculating frequency is a critical part in this design. Finally the strategy of displaying frequencies is discussed, which is similar to the basic method of DC voltage display but much more complex.

4.1 DC voltage displaying strategy

For measuring DC values, even if the sample rate is up to 625Ksps which means the system can get one new ADC data per 1.6μs, the display update rate is only 60Hz so that any kind of display update rate is constraint by it. The voltage is the most important parameter of DC input, so this design will display the input voltage in area2 (Figure 3-7).

The display of static note like "Voltage=" is simple referred to Chapter 3.3. Just build the image file and change it to a mif file, next build a ROM to store the notes, finally output data under the control of pixel's address. So the key point here is the dynamic display of output ADC values. A font library which contains ten numbers (0~9) and other critical notes like the dot and 'v' for displaying is required. They are all built by "paint" because it is easier to control the dynamic display when the elements in the font library have a same size. In my design, each of them is a 20x18 picture and they are stored in the ROM and output values according to output pixel coordinates. Then the control signals which are applied to select correct numbers are important. Due to 12-bit input ADC values, circuits convert binary data to decimal data are built by mode module and divider. Finally four numbers will be selected to correctly display the voltage (Figure 4-1).

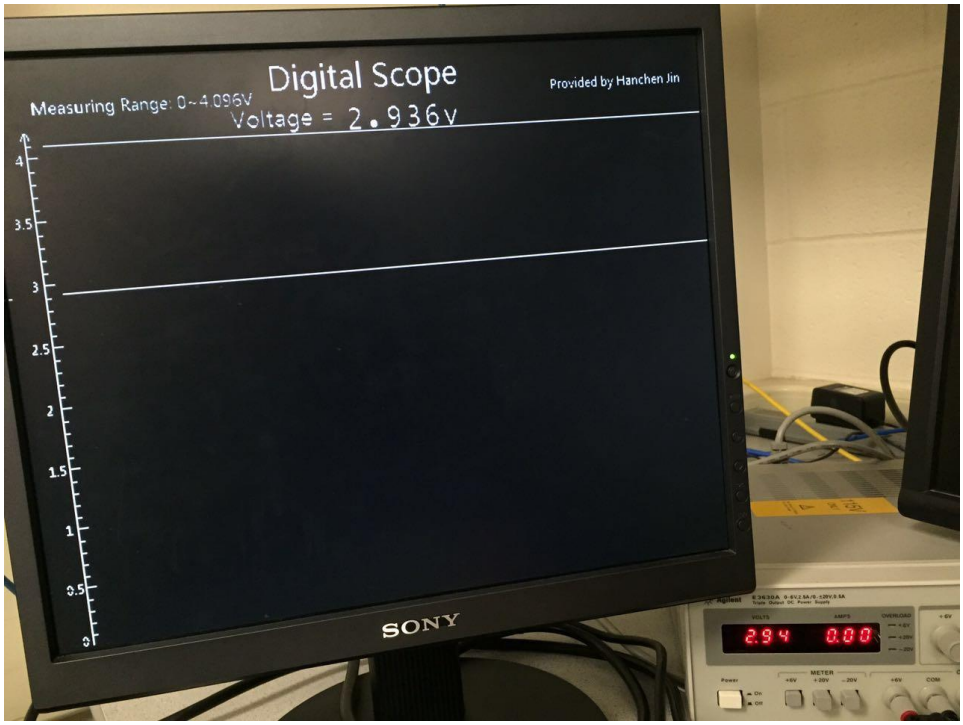

Figure 4-1: DC voltage displayed on the screen

The measuring range for this ADC converter ranges from 0 to 4.096V, so the measured value holds three digits after the decimal point. From Figure 4-1 we can learn that the voltage measured is pretty precise, even more accurate than the display on the Power Supply.

4.2 Calculating peak voltage

So this is a function which helps user to measure the peak voltage for waveforms. Basic idea is to find the maximum and minimum values among the 640 ADC values stored in two register arrays (mem0 and mem1). This process can be performed at the same time when data stored into these two register arrays. Each new ADC value will be compared with temporary maximum and minimum vales, if it is larger than the temporary maximum one or it is smaller than the minimum one, the corresponding temporary value will be replaced by this new value. By this method, the peak voltages from two register arrays will be stored into two groups of registers. Given the limited area on the screen, these values are showed by two horizontal lines so that users can read the value through coordinate axis on the left side of the screen. Also for stable displaying of these two horizontal lines, results comes from displaying register array are used. Figure 4-2 shows this function. A switch is applied to enable this function. Compared with the oscilloscope, the cursor here will automatically indicate the peak voltage.

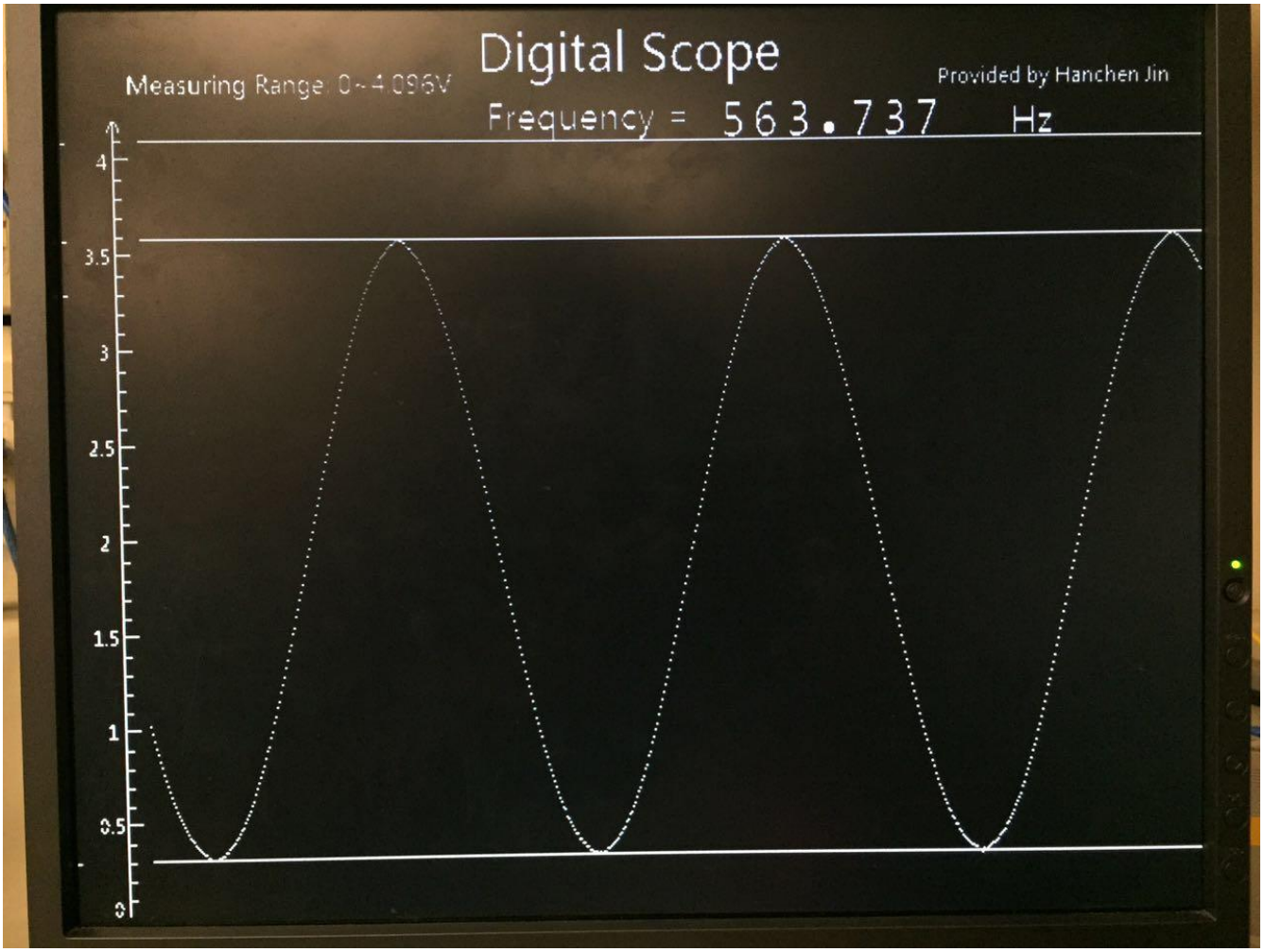

Figure 4-2: Function of peak voltage cursor

4.3 AC frequency calculating algorithm

Now the focus of this design is about the calculation about frequency. Since FPGA logic is able to do some high-speed parallel floating operation, frequencies of waveform can be calculated quickly. The basic strategy is to detect intersections between waveform and trigger voltage. A counter is applied to record the period of the waveform, also a divider is required for calculating frequencies.

Figure 4-4 is the state machine for frequency calculating algorithm. In the first state, it will determine the current measuring mode; if DC mode is selected, this module will do nothing to save energy. When setting the AC mode, trigger voltage is updated and reserved as reference for detecting cross points. This state is critical because for many states during the process of frequency measurement, this referred voltage (trigger value in the figure) should be kept the same value or else the result will be wrong. Then the state machine will move down to find the cross point. For example, assuming a sine wave is being measured (Figure 4-3), the value of first ADC data determines which branches it jumps in. If the first ADC value (value from area ①) is larger than trigger voltage, the state machine will jump into left branch and waiting in that state for intersection L1. Then the state machine will wait until the ADC value reach area ②. So if the current ADC value is smaller than the trigger value that means the first intersection L1 is detected, the state machine will start the counter and apply the same principle to detect the second intersection L2 and the third intersection L3. When the third intersection is detected, it will stop the counting process and apply the value in counter to calculate the frequency. And if the first ADC (value from area ②) is lower than trigger value, the state machine will jump into right branch and waiting for detecting cross points R1, R2 and R3 based on the same principle.

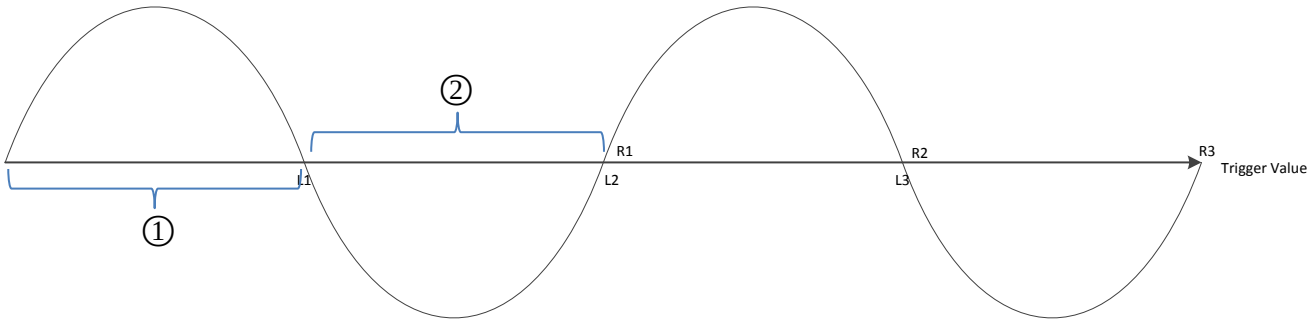

Figure 4-3: Diagram of sine wave input_v1

Since the counter is counting the ADC clock (40MHz). So we know that:

Frequency = 40M/counter value

However the division takes longer time than the counting process when higher frequency waveform is input, thus a state waiting for the complement of this operation is required. So I build a divider by subtractors for getting signals showing its completion. Even if some extra time (roughly in the magnitude of ms) is added to the whole calculation, it is worth to add it because the calculating system becomes more stable and accurate.

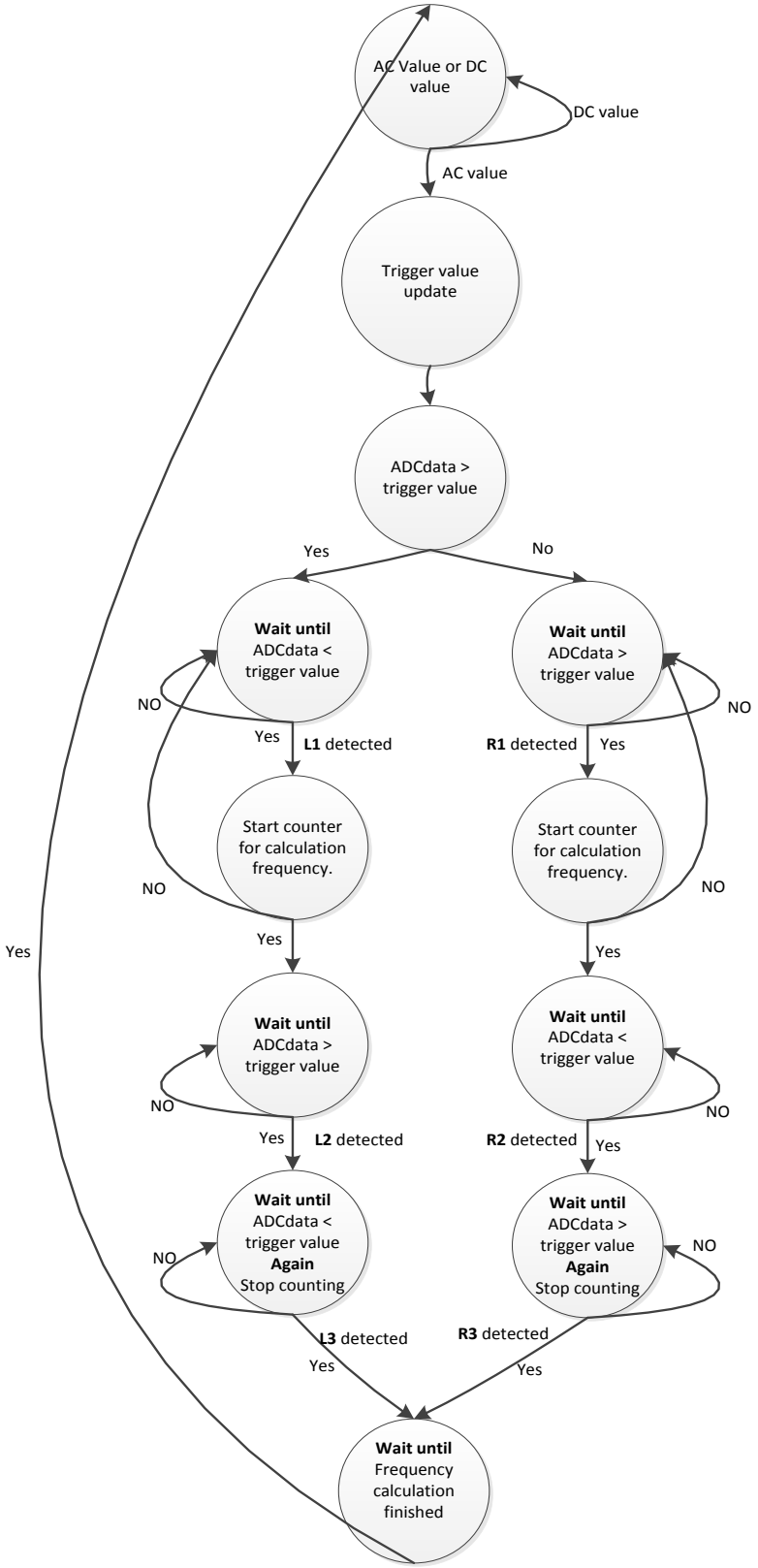

Figure 4-4: State machine for frequency calculation_v1

With the frequency calculation_v1, most results are correct expect for some low frequency inputs. So the frequency calculation_v2 which includes different thresholds (trigger value max/min) for detecting cross points is implemented. The reason for causing wrong result in frequency calculation_v1 is that if simply applied trigger value as referred voltage, errors will occur when inputting waveform with low frequency (lower than 50Hz). When the input waveform is lower than 50Hz, the result is always more than 50KHz. The displaying waveform provides me some hints. The display is nearly one horizontal line on the screen which means that 640 ADC values in one screen nearly has the same value. And due to the tiny error of the measurement by the ADC converter or even the error caused by signal generator, the measured values slightly fluctuate between a small range. Given that the algorithm is based on the judgement of input values, the oscillation near the trigger value always lead to the wrong determination about intersections. The value in the counter is incorrect thus the final result is also wrong. For resolving this problem, I defined minimum and maximum trigger value as threshold voltages.

$$\text{Trigger voltage max} = \text{Trigger voltage} + 0.01$$

$$\text{Trigger voltage min} = \text{Trigger voltage} - 0.01$$

A diagram for illustrating this improved algorithm is showed in Figure 4-5. For instance, if the trigger value (the dotted line) is 2V, trigger voltage max/min (solid lines) will be 2.01 and 1.99 separately. Suppose the first input value is in area ①, when input value is larger than 1.09V, it will wait for the intersection. And when the ADC value is lower than 1.09V, at that point the first intersection L1 is detected. Then it will start to detect second intersection. The oscillation is exist so the next ADC value will become larger than 1.09V (maybe 1.091V or more but lower than 2.01V), but within different threshold, this input value will not be viewed as the second intersection because this value is lower than trigger voltage max. So the correct second intersection L2 will be detected until the value is larger than 2.01V. And from Figure 4-5, we can learn that the time between L1 and L3 is the period of this waveform. Same period happens from intersection R1 to R3. So the false judgement of intersections will never happen again. The basic structure of state machine for frequency calculation_v2 (Figure 4-6) is nearly the same with frequency calculation_v2.

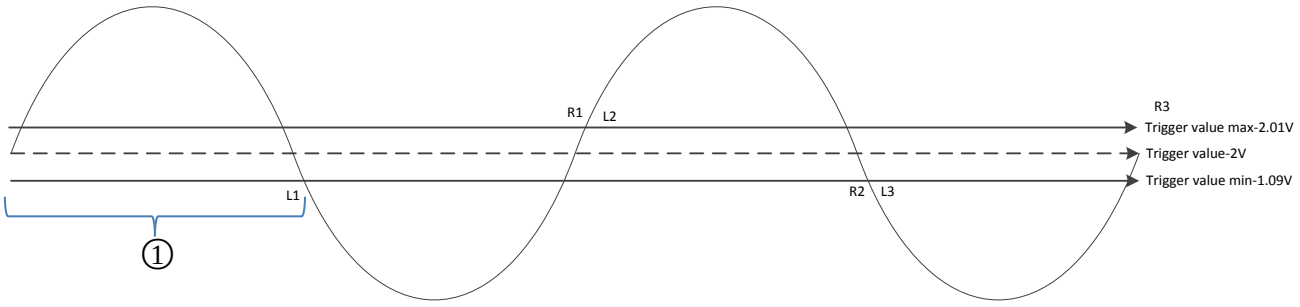

Figure 4-5: Diagram of sine wave input_v2

Also I display the frequency on seven segment LEDs to verify it. All frequencies are displayed with significant digits ranges from four to six (three digits after the decimal point). So a three-bit variable is applied to record the actual value of the frequency to distinguish the unit for this frequency (Hz or KHz). This is a useful control signal for the display on the monitor which is introduced in next section.

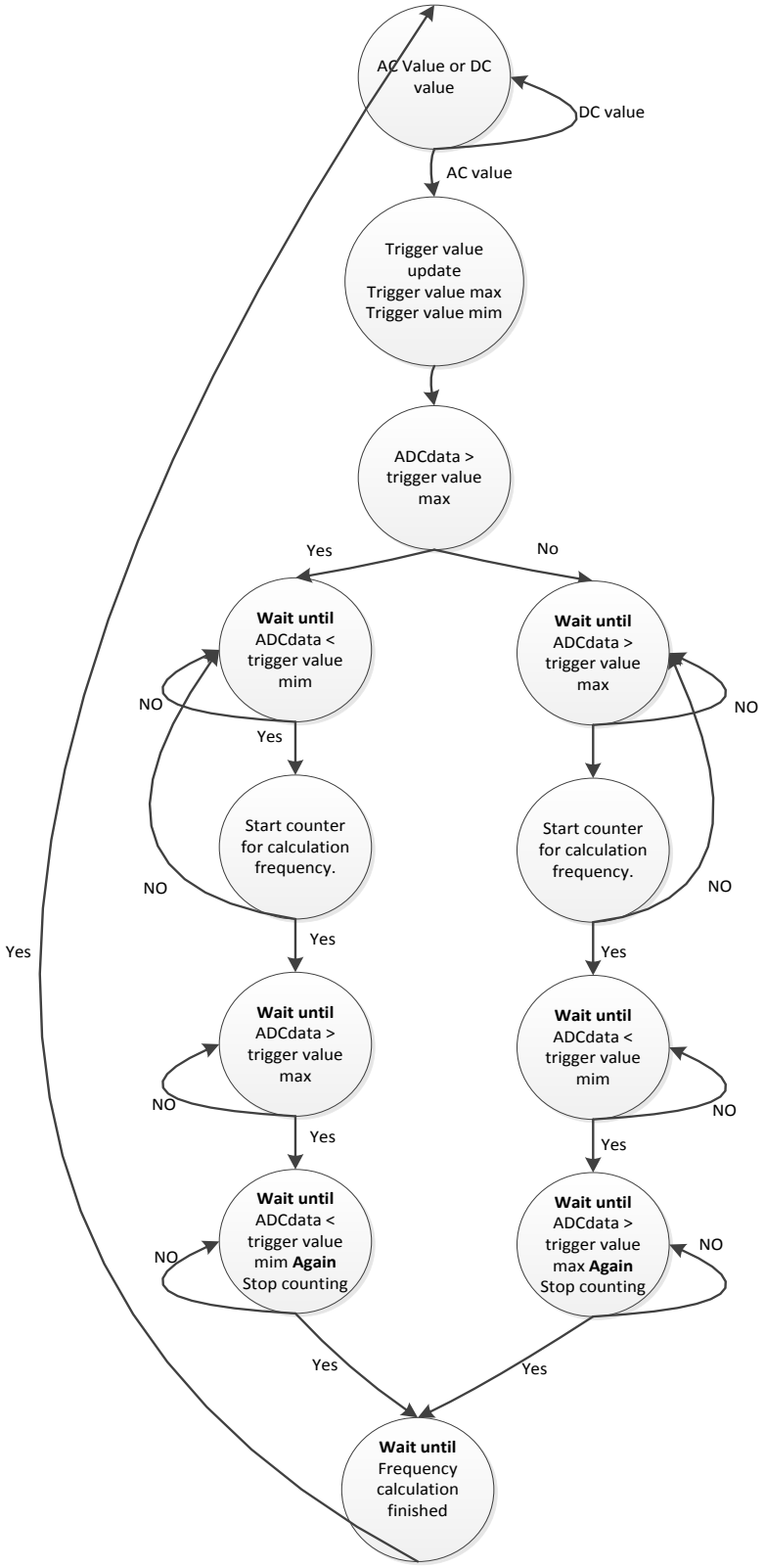

Figure 4-6: State machine for frequency calculation_v2

4.4 AC frequency displaying strategy

The frequency displaying strategy is similar to the voltage display under DC mode but a little harder. Different frequencies have various significant digits with different units (Hz or KHz). So I have to explore my existing font library—adding notes like “Frequency=”, “KHz” and “Hz” to it. The variable called “unit” which records the actual magnitude of the frequency plays an indispensable role, which determines whether some specific digits should be visible or not, especially top two digits. Note “Frequency=” is always displayed there as an indication for the meaning of following numbers. Note for showing the unit of frequency is also selected by the variable “unit”. So now the frequency can be displayed on the monitor. The time of calculation is smaller than the interval of displaying, so the update rate of frequency is also restricted by the VGA update rate (60Hz). Figure 4-7 shows the displayed frequency with a square wave input. A small trick here: if you want to transfer data through two modules with different clocks, it is better to build a register array or FIFO which serves as a buffer to help the system transmit data fluently.

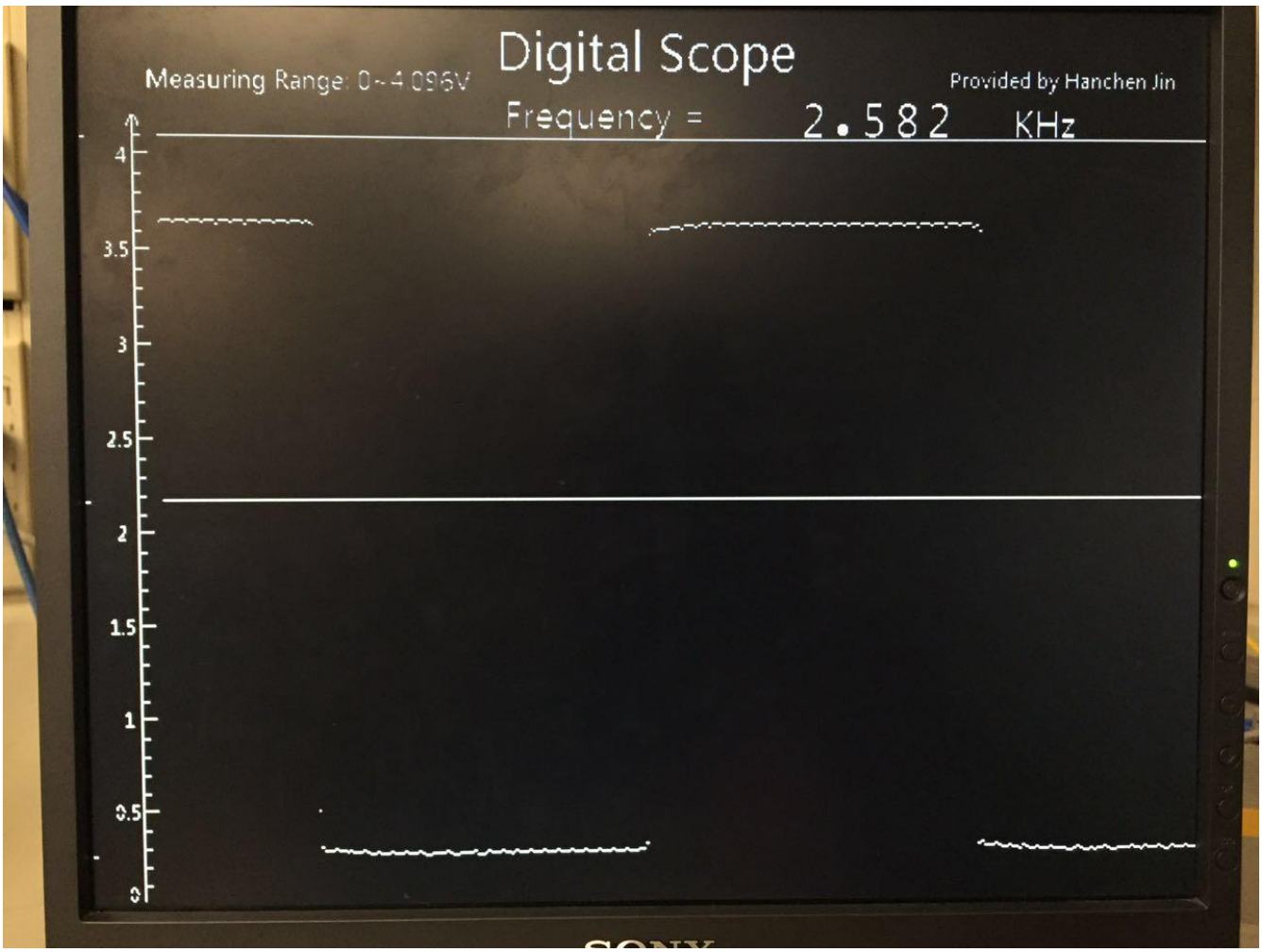

Figure 4-7: Frequency displaying with square wave input

5. Reset function

Reset function is so important that I decide to discuss it in a separate chapter. Sometimes with wrong input or mal-operations, the design may freeze and stop working. It is possible to resolve this problem by reconfiguring the FPGA with the programmer, but it takes some time and also this design lost its own independency towards the computer. Thus the reset function triggered by external hardware control is essential for recovering the whole system in a lower consumption.

5.1 Reset ADC module

For exploiting the sample rate of LTC2308, the ADC module is reset every 1.6μs to continuously convert the analog input to 12-bit ADC values. It clears all control registers for the measuring state machine so that it can start a new measuring immediately. Actually the hard reset switch does not influence this module. Because the reset function here triggers the procedure of measuring ADC values, which is the fundamental function for the digital scope. So it is controlled by fixed circuits and users can not interrupt it by external control from switches and buttons.

5.2 Reset VGA display module

The aim of applying a switch to reset the whole system is to recover the accidental freeze of the design. The reason is mainly caused by the stuck in the state machine for trigger function (Figure 3-9). Suppose the digital scope is working in AC mode and in the state of waiting data equal to trigger value, now the user change it to DC mode which input a voltage and it is not equal to the trigger value. Under that condition, the whole design will stuck in that state until AC values input again and with a value equals to the trigger value. So if that happens, it will keep the display of AC value but not the expected DC values on the screen. This is an extreme phenomenon but it does happen. So the best way to resolve this problem is resetting the whole state machine to let it be able to judge the type of input value again. By the way, when reset the state machine, it is sensible to reset all registers related to it or else it may stick into another unexpected state.

5.3 Reset data processing module

Same ideas for this module, in the frequency calculation state machine (Figure 4-3), when the digital scope starts countering for frequency, the mode modification will cause wrong results. But this will recover by itself since the period for frequency measurement is small. So the reset function here is just served as reserved method for recovering the calculating process.

6. Result & Evaluation

All functions covered in previous chapters have been successfully implemented in this board. So in this chapter, it summarizes the key points of this design, evaluates the performance of it and provides some general ideas for further improvement.

6.1 Result summary

In this design project, I implement a digital scope based on the DE1-SoC. This board is the only hardware required for this design since it has the hardware module including ADC converter for measuring analog input, abundant FPGA programmable logic for storing and processing ADC values, and the VGA output port for displaying on the monitor. It can measure either DC value or AC value. Also it will display voltage for DC values and frequencies for AC values. The build-in interface User Guide can help users learn how to control this design. Eight input channels are available for selection. Other useful functions like run_stop, peak voltage cursor, trigger voltage adjustment, horizontal position adjustment are also implemented in this design. All this functions are implemented by FPGA logic and nearly 20% of the whole FPGA logic on the board is used for this design.

6.2 Evaluation

- Waveform display

Figure 6-1 is the display of final output. From that picture we can clearly learn that the input is a Sine wave even if this waveform is composed of many points. It is accessible to explicitly display the waveform with its one or two cycles. Since the horizontal position of the display can be adjusted in this design, waveform with any frequencies can be clearly displayed on the screen. However, the display is still restricted by the sample rate of ADC converter. If the frequency of the input waveform is higher than 30KHz, the output waveform seems not very well (Figure 3-11). It is reasonable since the sample rate is only 625Ksps, the raw ADC values for display is limited so it is a hardware restriction. This also causes the larger error for frequency calculation.

- Accuracy of parameters

Two key parameters here—one is the voltage, another is the frequency for AC input. The measuring range is 0 to 4.096V within 12-bit values, so the accuracy for measuring voltage is 0.001V which is pretty precise. But the error of frequency increased with the input value due to the limited sample rate.

Table 6-1 shows the result of frequency calculation and the corresponding error under different input. A line chart is drew based on data in table 6-1 (Figure 6-2). The red line shows the errors within different input, the black line shows the tendency of this line chart. Within the higher input frequency, the errors become larger. Since the sample rate is 625KHz, from the chart we can also learn that when the input is larger than 62.5KHz, the error tends to be larger than 2% which can not be ignored any more. Actually, under 10KHz, the error is nearly lower than 0.5% which performs really good. Since with higher input frequency, the data can be measured in one cycle is limited (like Figure 3-11 shows) which leads to the larger error. Thus this is constricted by the hardware module—the sample rate of ADC converter.

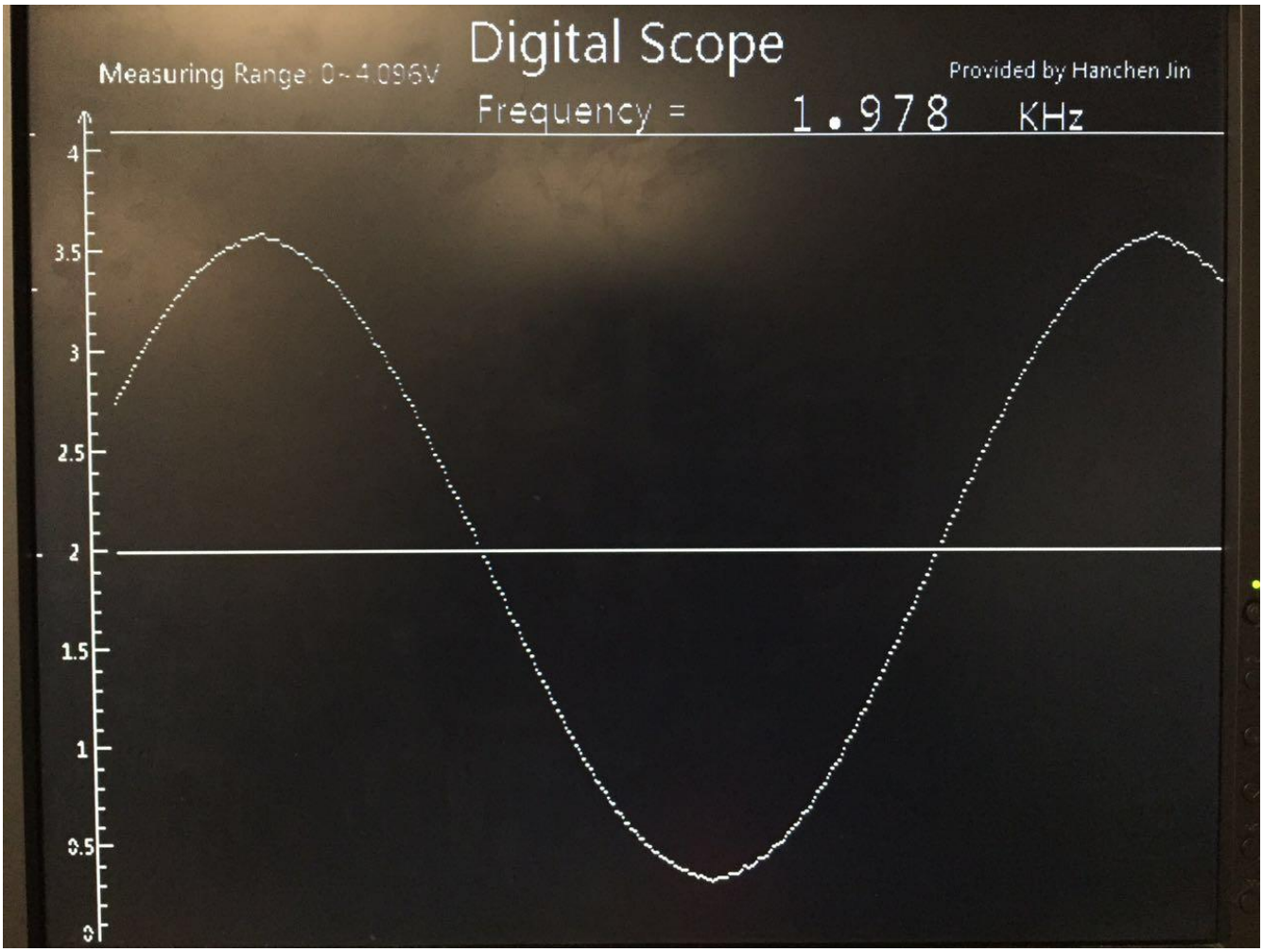

Figure 6-1: Display on the screen within a sine wave input

Table 1 errors of frequency calculation under different input

Actual value (KHz)	Calculated result (KHz)	Error (%)
1.702	1.705	0.18
5.753	5.716	0.64
9.483	9.498	0.16
14.821	15.060	1.61
20.464	20.586	0.60
25.460	25.740	1.10
31.360	30.888	1.51
41.590	41.194	0.95
51.500	51.546	0.09
60.570	61.823	2.07
70.540	68.728	2.57
98.29	103.092	4.89

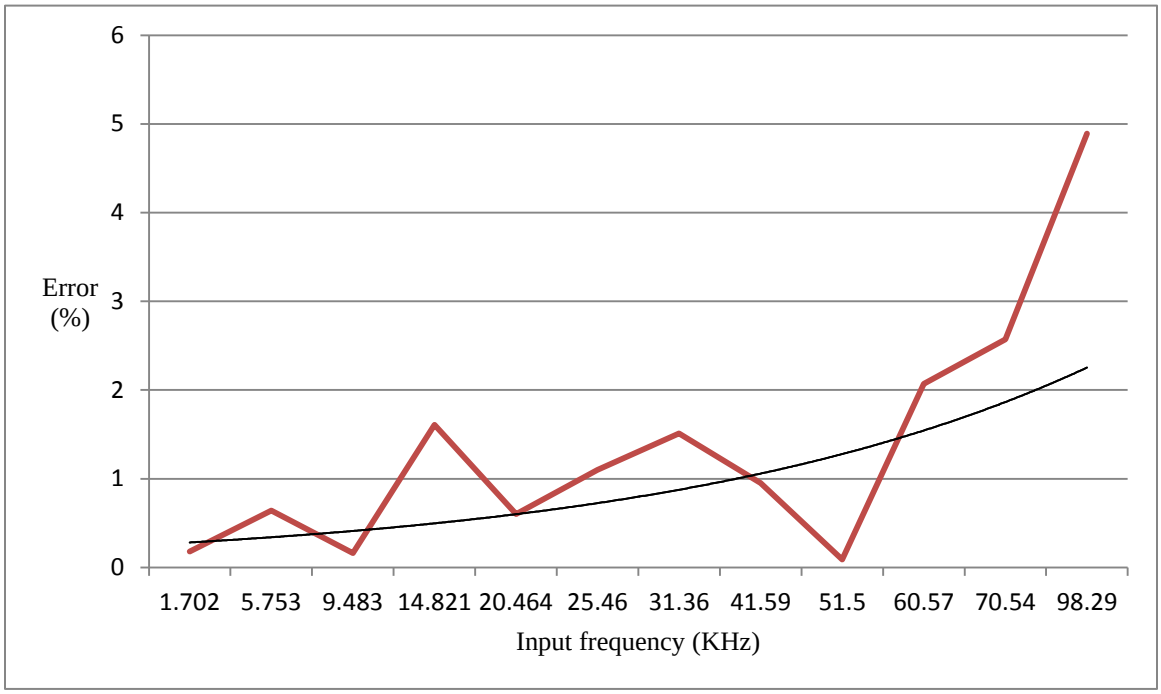

Figure 6-2: Display on the screen within a sine wave input

6.3 Further improvement

For current design, nearly 20% of the FPGA logic is applied. Also there are many other advanced modules on DE1-SoC like SDRAM, Advanced Microcontroller Bus Architecture (AMBA) Advanced eXtensible Interface (AXI) bridge, dual-ARM core, etc. Thus here I provide two possible directions for improving the current design.

- Improving current functions

From the evaluation, we can learn that the limited sample rate caused larger error when calculating frequency within larger input. Even though the error of frequency calculation might be reduced by exploiting some advanced algorithms, the display of the waveform is hard to be improved because of the limited ADC values measured in one cycle. Also the measuring range of input voltage which is 0 to 4.096V is really small. However, these are hardware limitations so the fundamental solution is to leverage a more powerful ADC converter.

- ADC data transmission to laptop

Compared with oscilloscope, one function this design does not have is storing the picture of the waveform into our own laptop. Even though the smart phone can take high-quality photos of the display on the monitor, this function is also desired to be implemented. For implementing this, we can store ADC values into 64MB SDRAM, and these data can be sent to the HPS part through the AMBA AXI bridge. Under the control of ARM core, these data can be transmitted to our laptop/PC through USB port. Furthermore, we can build a small software to draw waveforms on PCs. So this may also be a really interesting project.

Acknowledge

Firstly, I appreciate the hardware platform for this design (DE1-SoC) and the lad equipment provided by Cornell ECE department. DE1-SoC is a good platform to implement some interesting designs and lab equipment like oscilloscope and signal generator are essential for testing and debugging this design.

Then I really appreciate the generate help from my MEng advisor—Prof. Bruce Land. Basically, he told me that this board could be applied as a digital scope since the embedded ADC converter is able to acquire ADC values and VGA port can be used for display. So according to his basic introduction of the board, I decided to take on this interesting project—building a digital scope on DE1-SoC. During the design, difficulties appeared nearly every week. It is my advisor who helped me to resolve these problems and provided me confidence to keep going. Without his help, it will be hard for me to finish this design in time. So I feel grateful to have a chance work with him.

References

All references below come from the DE1-SoC_v.5.0.1_HWrevF_SystemCD
http://www.terasic.com/downloads/cd-rom/de1-soc/

- User manuals:

[1] DE1-SoC_User_manual
[2] DE1-SoC_Getting_Started_Guide

- Schematics:

[3] DE1-SoC

- Datasheets:

[4] ADC converter: LTC2308fb
[5] Video DAC: ADV7123

- Demonstrations[4]—FPGA:

[6] my_first_fpga
[7] DE1_SoC_ADC
[8] DE1_SoC_Default
[9] DE1_SoC_golden_top
[10] DE1_SoC_TV
[11] DE1_SoC_SDRAM_RTL_Test

Appendix

1. Demo video:
https://www.youtube.com/watch?v=cc3TSWgk0vQ&list=PLE0F52F4DD6990C7A&index=11
2. Source code for the whole project:
https://www.dropbox.com/sh/59jz4tha4j60xm8/AAA82S8yNqyFBNY8mCpjxQTVa?dl=0
3. Figure for User Guide Interface

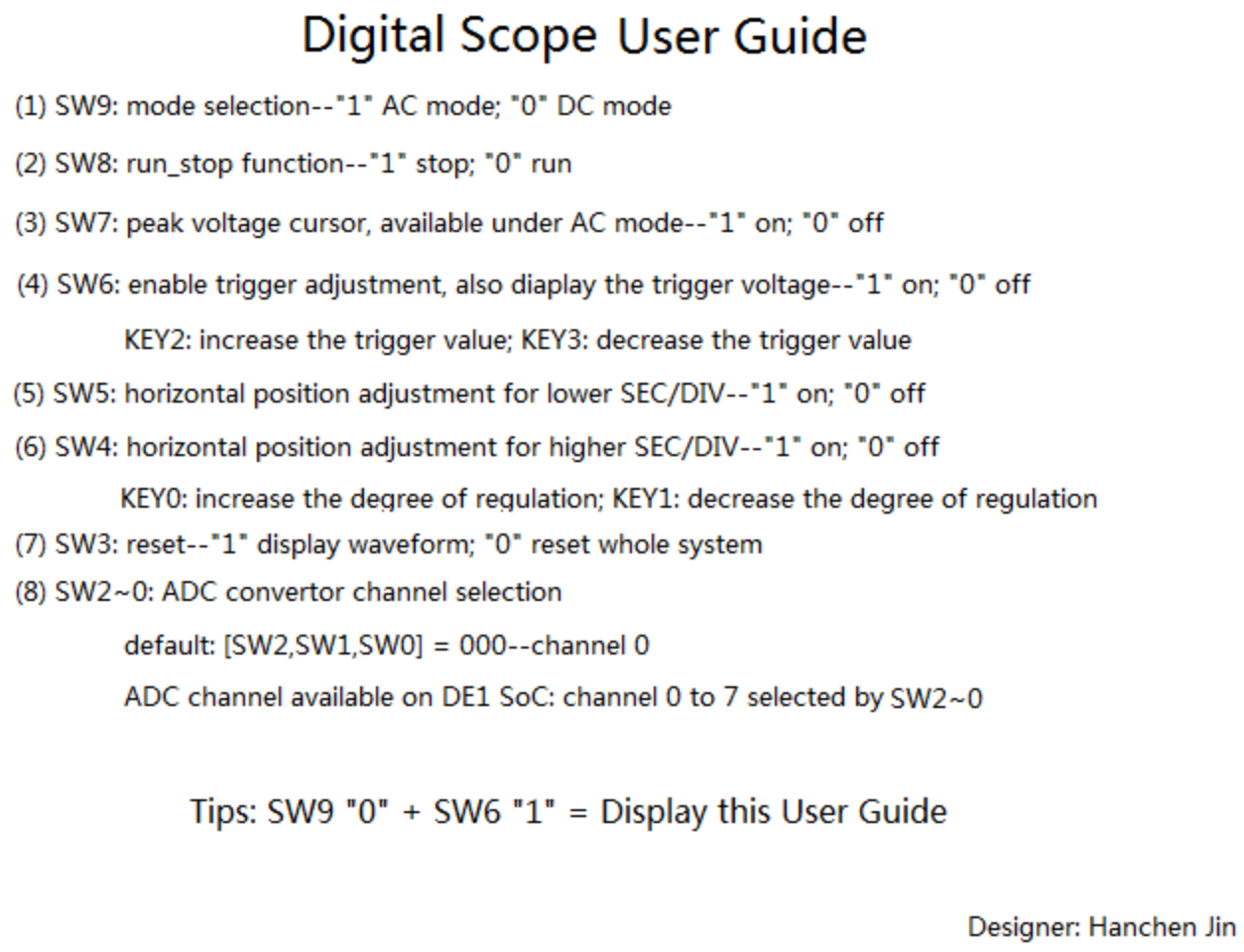

Figure A-1: User Guide interface of digital scope