UNITED STATES
AIR FORCE
ACADEMY

ECE 383 - Embedded Computer Systems II
Lecture 19 - Lab 3 - Software control of a datapath

- see Teams / Lab3
- see Lab3_Install_short_version.pdf

Bitbucket: too much to push? (path length too long for git?)

- vhdl, xdc, .c, README
  2 AXI fiks!

L20	Soft CPU		HW #11	BOC L21
L21	Lab3 - O'scope control	Final Project Ideas		
L22	Lab3 - O'scope control		Gate Check 1	End of L22
L23	Lab3 - O'scope control		Gate Check 2	End of L23
L24	Lab3 - O'scope control		Gate Check 3 ~~Lab3 Functionality~~	COB L24
L25	Lab3 - O'scope control		Lab 3 Functionality	COB L25
L26	Direct Digital Synthesis	Final Project Ideas	Lab3 Write-up Final Project Proposal Section 2	COB L26 BOC L27

HW12 BOC L27 Alot Due!

- No team projects
- Everyone does a different project
- Difficulty Level

- 383 website / Datasheets → hints: NES, IR Controller and PS2 Mouse

UNITED STATES AIR FORCE ACADEMY

Lesson Outline

- **Lab 3 – Software control of a datapath**

Lab 3 – Software control of a datapath

UNITED STATES AIR FORCE ACADEMY

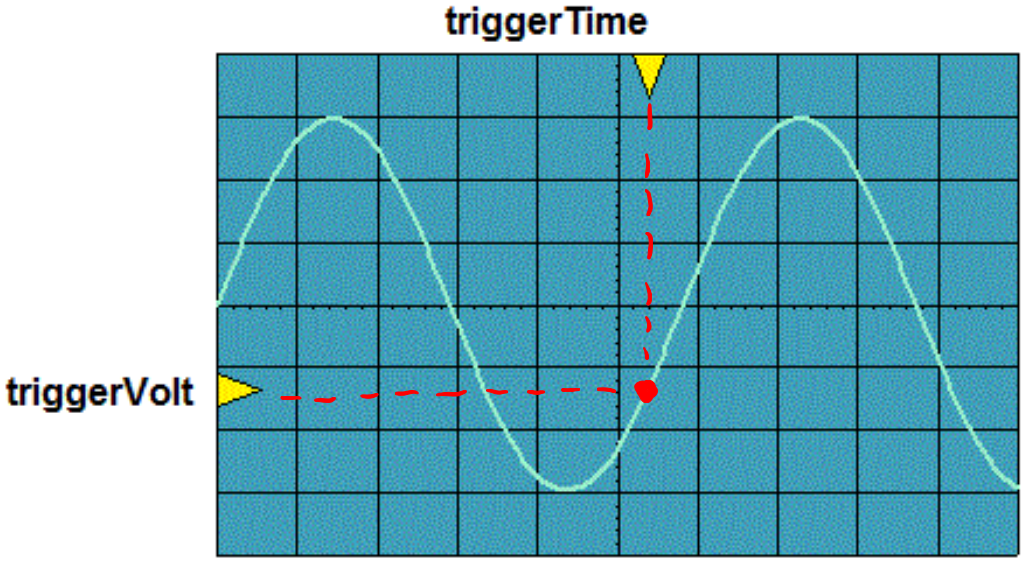
Lab 3 – Lab Overview

- Lab Overview - In this lab we will integrate the video display controller developed in Lab 2 with the MicroBlaze processor built using the fabric of the FPGA. In the preceding lecture we have learned about the SDK tool chains, now its time to put that knowledge to the test by building a software controlled datapath. Lab 2 revealed some shortcomings of our oscilloscope that this lab intends on correcting.
  - Using both trigger volt and trigger time for trigger
  - Using polling and/or interrupts
  - Ability to enable and disable channels to display
  - ~~Ability to trigger off channel 2~~ ← Bonus
  - ~~Ability to change the slope direction for the trigger.~~

UNITED STATES AIR FORCE ACADEMY

Lab 3 – Lab Overview

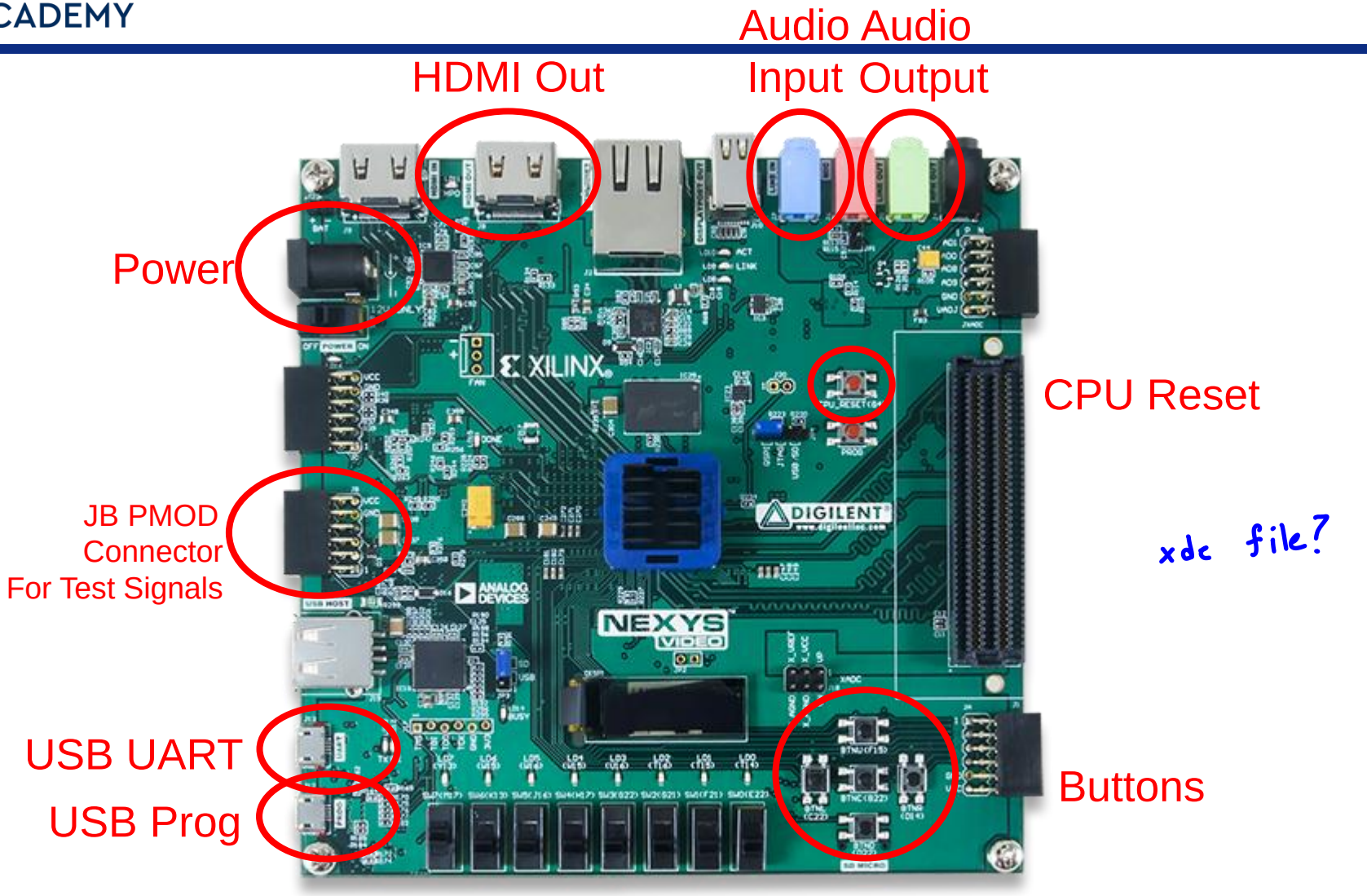
- The following figure shows required functionality - your program should allow the user to change the position of the triggerVolt and triggerTime indicators with the result that the waveform should be drawn so that the periodic waveform is increasing through that voltage at that time.



Before After?

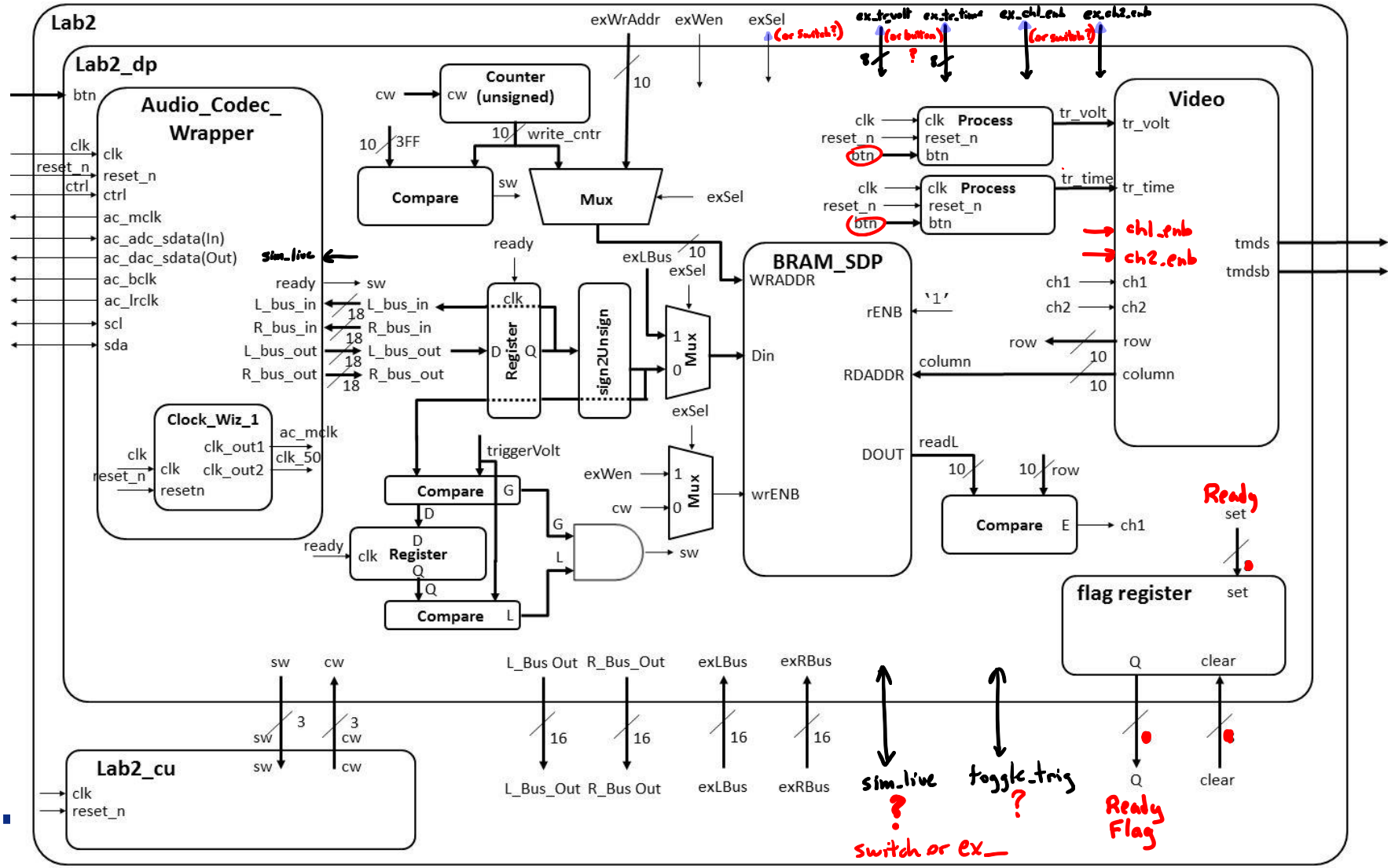
UNITED STATES AIR FORCE ACADEMY

Lab 3 – Connections

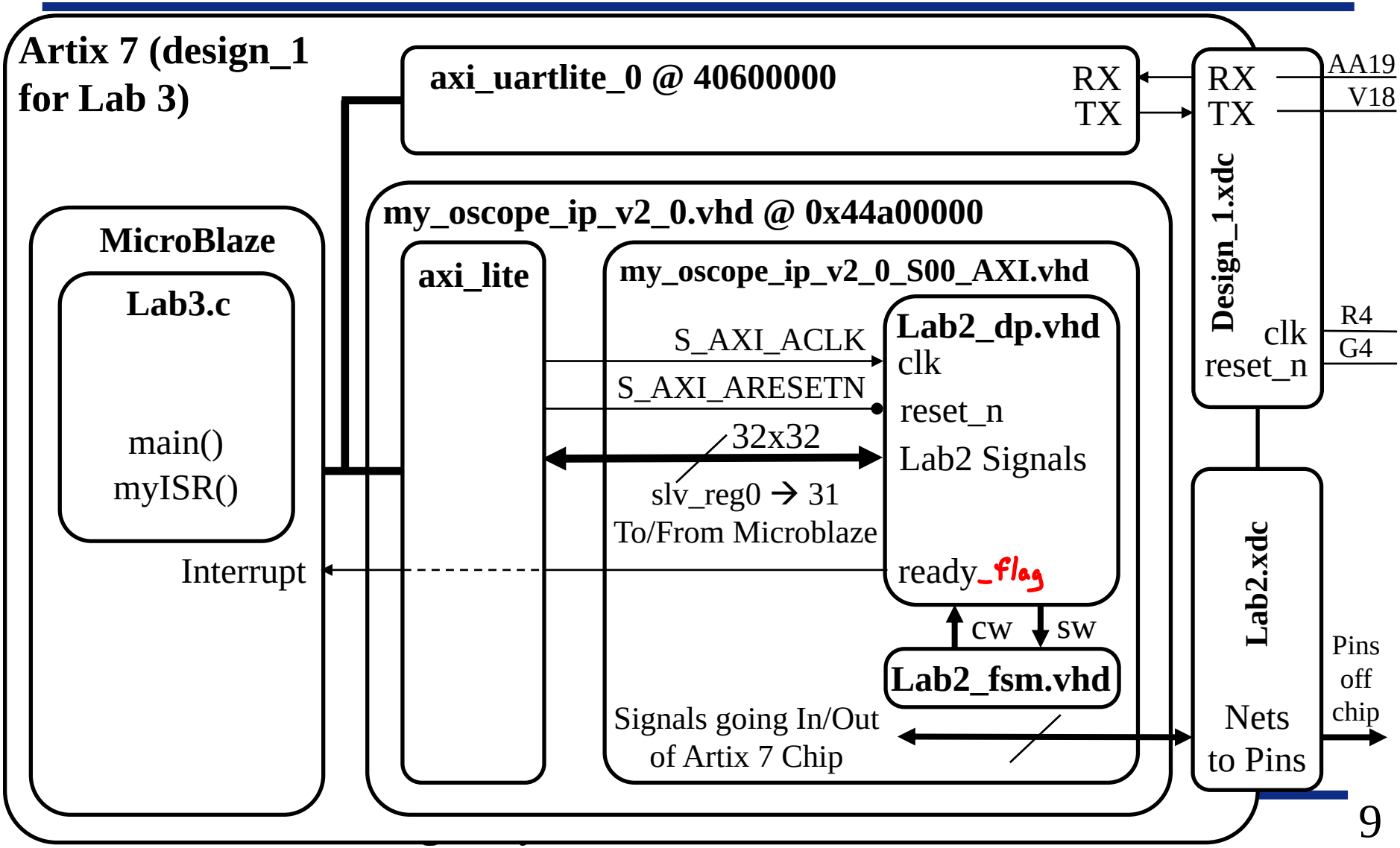


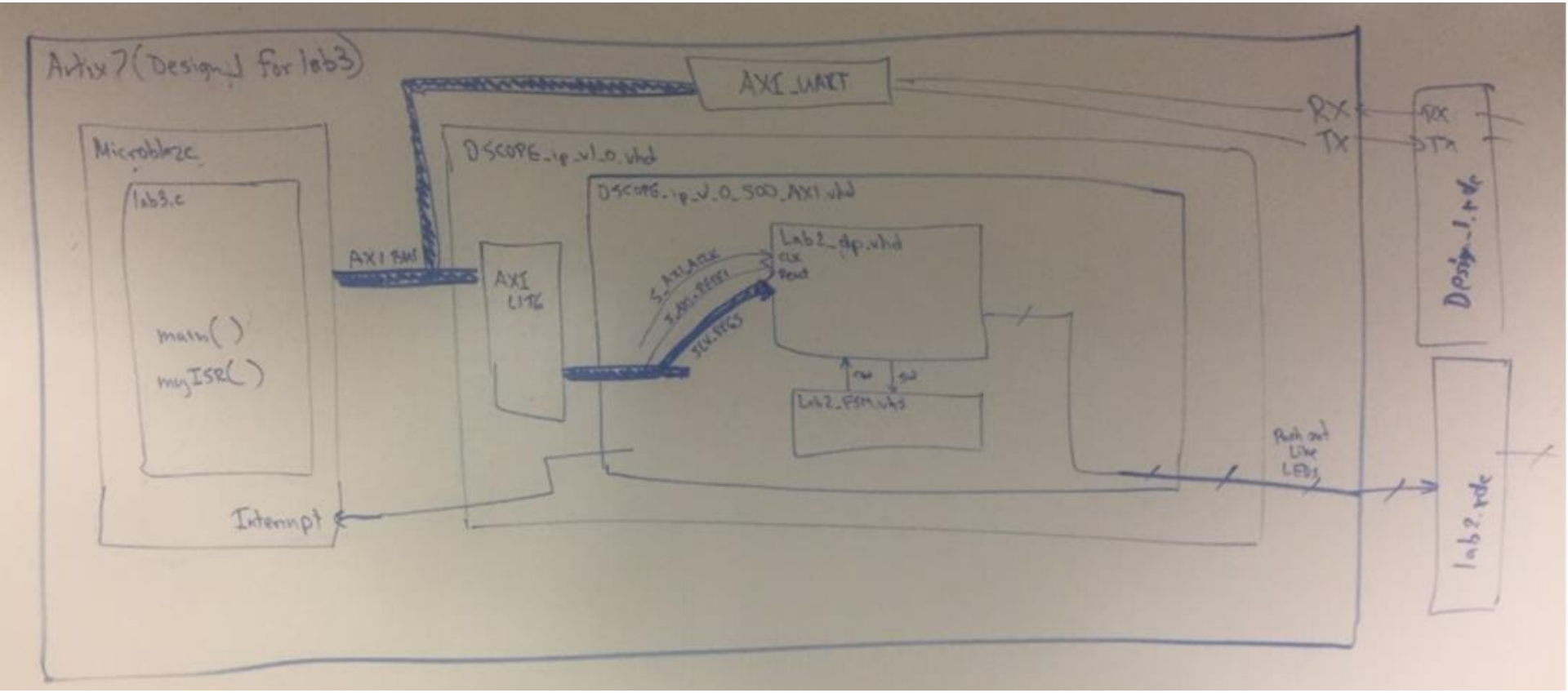
UNITED STATES AIR FORCE ACADEMY

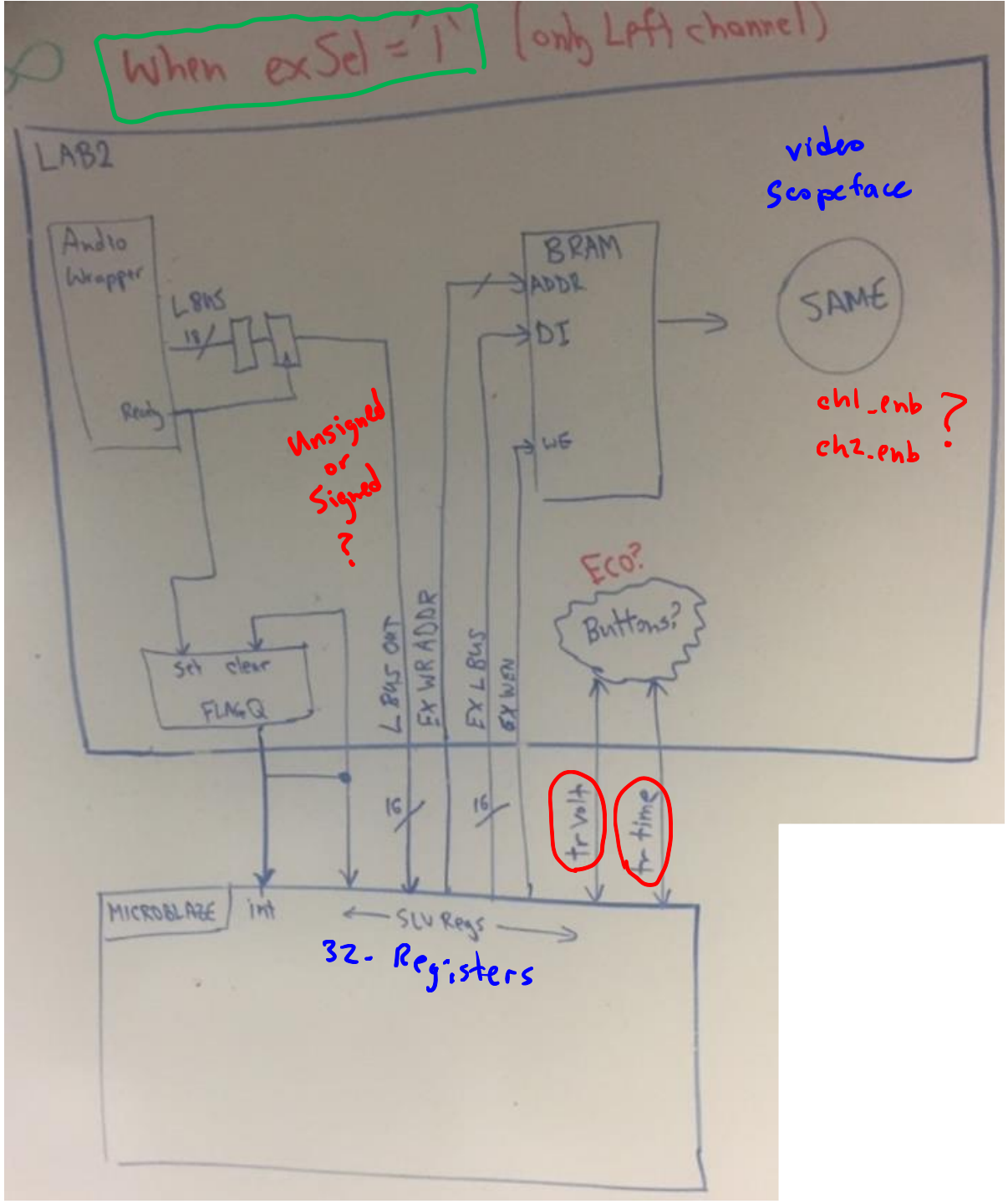
Lab 2 – Architecture



Lab 3 – Architecture



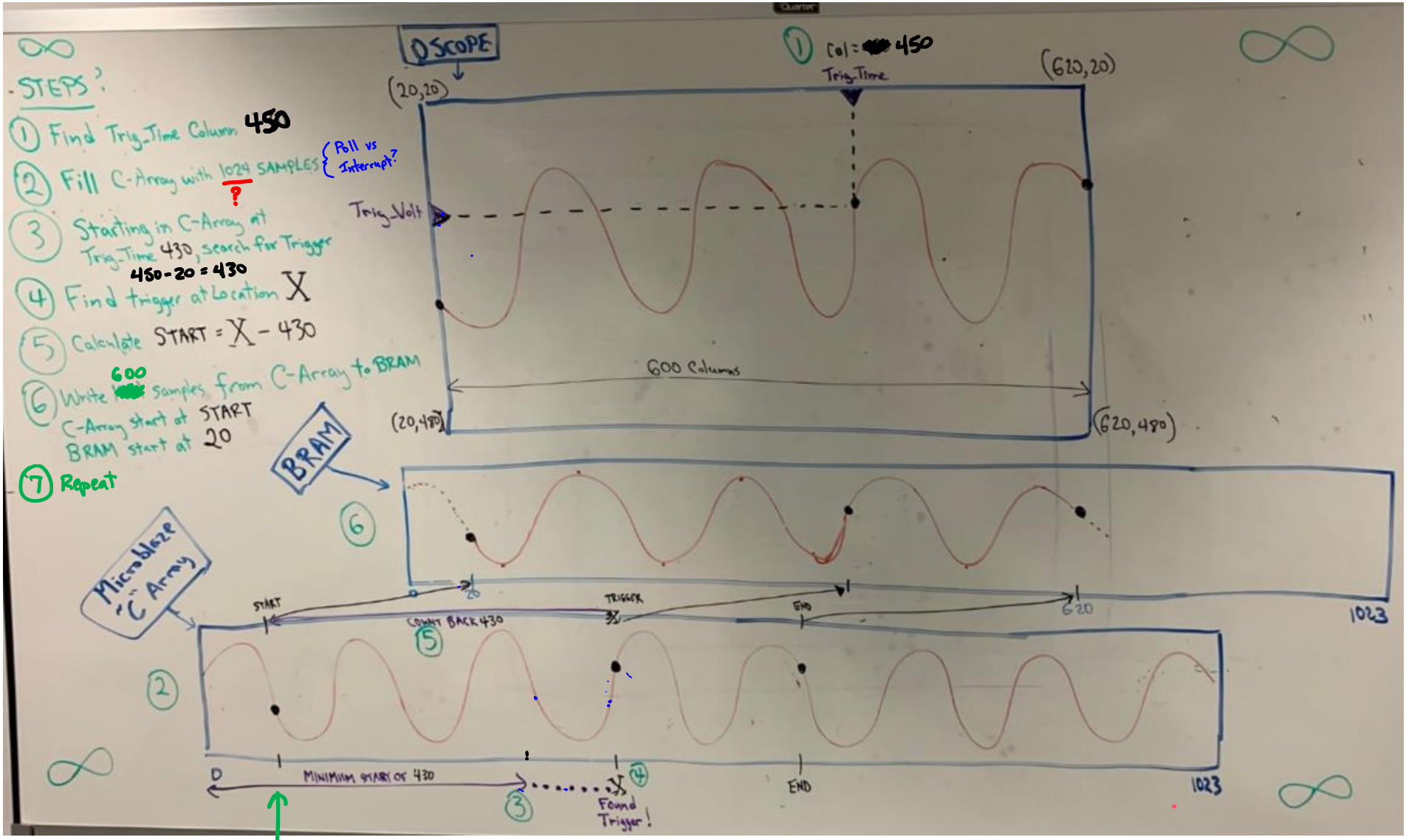

Artix7 (Design) for lab3
AXI_UART
RX
TX
Microblaze
lab3.c
main()
myISR()
AXI BUS
AXI LITE
DSCOPE_ip_v1_0.vhd
DSCOPE_ip_V_0_S00_AXI.vhd
Lab2_dp.vhd
clk
Reset
SLV_REGS
Lab2_FSM.vhd
Push out Live LEDs
Interrupt
lab2.xdc


When exSel = '1' (only Left channel)
LAB2
video
Scopeface
Audio
Wrapper
L BUS
18
Ready
BRAM
ADDR
DI
WE
SAME
chl_enb ?
ch2_enb
Unsigned
or
Signed
?
ECO?
Buttons?
Set clear
FLAG
L BUS OUT
EX WR ADDR
EX L BUS
EX WEN
16
16
tr volt
tr time
MICROBLAZE
int
SLV Regs
32- Registers

DESIGN?

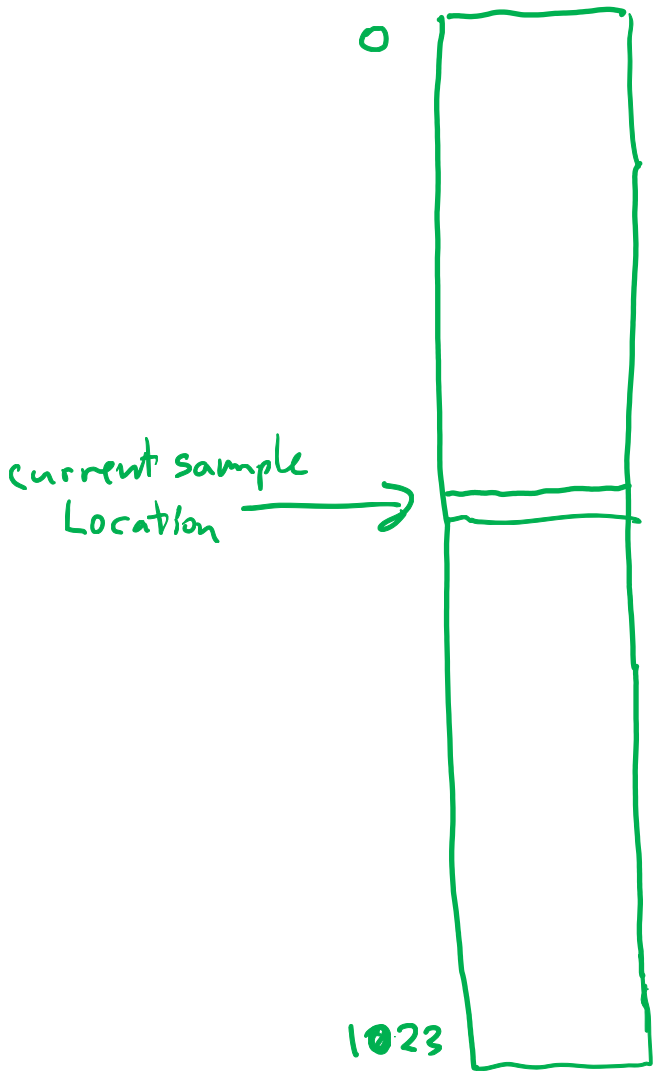
Hint: see Lab3_MiniC_psuedocode.c

LINEAR ARRAY APPROACH



WHAT IF NO "HIT" in step 3?
WHAT DO YOU DO AFTER STEP 6?

CIRCULAR ARRAY ?



X → trigger location?

Start pointer = X - 430

End pointer = start + 600

When get to END,
then write START to END
to BRAM

Everytime get a new sample
- see if new trigger_point X
- update start pointer
- update end pointer
- update new sample location

UNITED STATES AIR FORCE ACADEMY

Lab 3 – UART Note

- **Note:** In your program, you may want to check if the users has hit the key on the keyboard without having to actually read the key. For these cases the following command will prove useful. Note that "uartRecReg" is a constant, the address of the uart.

XUartLite_IsReceiveEmpty(uartRecReg);

See 383 website / datasheets

Reading UART keyboard without being locked out

https://georgeyork.github.io/ECE383_web/datasheets/Using%20keyboard%20buffer.pdf

Also see "lessons learned" example on 383 website at the end of the assignment

UNITED STATES AIR FORCE ACADEMY

Lab 3 – Hardware

- With the expectation of the following engineering change orders (ECO) in the table below, the hardware you developed in lab2 will be unchanged. For the following ECO, please refer to the high-level architecture in lab 2.

Name:	Trigger Voltage, Trigger Time
Scope:	lab2_dp and lab2
Type:	Change to the entity descriptions.
Details:	• Inside the lab2_dp component, remove the logic driving the triggerVolt and triggerTime signals into the video component. • Remove the buttons signal from the lab2 and lab2_dp entities. • Remove the buttons signal from the ucf file. • Add the triggerVolt and triggerTime signals to the lab2 and lab2_dp entity descriptions. • Drive the triggerTime and triggerVolt inputs on the video component with the corresponding signals on the lab2_dp entity.

(or Option for 2021: Keep lab 2 buttons, and input Trigger Volt and Time Into Microblaze using slv_regs

Lab 3 – Signals

Hint: use lab3_signal_mapping.xlsx

- Your first step will be to create a component for your lab2 component in your Vivado repository. This will require you to think about what signals are routed to the MicroBlaze and those going outside the Artix 7 chip.

Design

To/From MicroBlaze	Outside Artix 7 → to xdc
exWrAddr	clk ?
exWen	reset ?
exSel ? or a switch →	ac_mclk
L_bus_out, R_bus_out	ac_adc_sdata
exLbus, exRbus	ac_dac_sdata
flagQ	ac_bclk
flagClear	ac_lrclk
triggerTime or buttons ? →	sda
triggerVolt	scl
ready	tmds
Ch1_enb, Ch2_enb ? ? or switches →	tmdsb

- sim_live?
- toggle_trigger?

Design table
- µBlaze and/or xdc
- number of bits
- direction, in or out?
- slv_reg used and bits used

Gate Check 1 - Complete Port Mapping Table
- update lab2 block diagram (slide 8)

MicroBlaze Registers [in = read; out = write]		
Signal	Direction	Register/Bits
exWrAddr		
exWen	write	slv_reg7[0]
L_bus_out	read	slv_reg5[15:0]
R_bus_out		
exLbus		
exRbus		
flagQ_ready		
flagClear		
Trig_Volt?		
Trig_Time?		
ch1_enb?		
ch2_enb?		
exSel?		
sim_live?		
toggle_trig?		

Outside Artix 7 (lab2.xdc)		
Signal	Type	Package Pin
clk?	clock	R4
reset?	button	G4
ac_mclk	Audio Codec	U6
ac_adc_sdata	Audio Codec	T4
ac_dac_sdata	Audio Codec	W6
ac_bclk	Audio Codec	T5
ac_lrclk	Audio Codec	U5
sda	QSPI	V5
scl	QSPI	W5
tmds[3:0]	HDMI out	T1 AB3 AA1 W1
tmdsb[3:0]	HDMI out	U1 AB2 AB1 Y1
up?	button	F15
left?	button	C22
down?	button	D22
right?	button	D14
ch1_enb?		
ch2_enb?	switch	F21
exSel?		
sim_live?		
toggle_trig?		

Hint: use lab3_signal_mapping.xlsx

Build HW once? or Many?

UNITED STATES AIR FORCE ACADEMY

Lab 3 – Requirements Gate Check 1

- **Gate Check 1**
  - By the conclusion of lesson 22, you need to have all of your Lab 2 functionality implemented with the Microblaze.
  - That is, you need to export your Lab 2 design into the SDK and be able to achieve the same functionality as you did in Lab 2.
  - You should be able to set exSel = '0' from your microblaze C program

Lab 3 – Requirements
Gate Check ~~2~~ 3

- **Gate Check ~~2~~ 3**
  - Task 1: By ~~the conclusion of~~ taps lesson ~~23~~ 24, you need to be able to send UART commands to your FPGA to adjust the trigger on the screen. The trigger on the screen should properly react to moving the trigger either up or down. arrows and left/right.

(or Option for 2021: Keep lab 2 buttons, and input Trigger Volt and Time Into Microblaze using slv_regs. Show on UART display you can read Trigger Volt and Time

Task 2: You must implement at least one of the baby-step test such as (see slides 28-29)
- microblaze write 600 samples to BRAM → see on scope face
- microblaze read 1024 samples from audio codec

UNITED STATES AIR FORCE ACADEMY

Lab 3 – Requirements Required Functionality

- **Required Functionality**
  - In order to make required functionality you will need to properly trigger the oscilloscope on channel 1 using a positive edge trigger. Control of this process is to be performed using the MicroBlaze. The main tasks of the MicroBlaze will include:
    - Moving audio samples into a pair of circular buffer. These circular buffers will be maintained in the address space of the MicroBlaze. That is you should have two big arrays defined in your program. Use polling of the ready bit of the flag register.
    - Examining the samples looking for a trigger event.
    - Fill the remaining sample slots in memory.

Bonus

(or Option for 2021: You can use a Linear Buffer instead of a Circular Buffer

UNITED STATES AIR FORCE ACADEMY

Lab 3 – Requirements Required Functionality Cont 1

- **Required Functionality cont**
  - Move the appropriate buffer values into the display memory of the oscilloscope (lab2) component.
  - Provide a user menu (through the terminal) allowing the user to adjust the trigger voltage and trigger time.

(or Option for 2021: Keep lab 2 buttons, and input Trigger Volt and Time into Microblaze using slv_regs. Show on UART display you can read Trigger Volt and Time

- Must work in Continuous Mode (no UART Blocking)
- Partial Credit for test cases that work

```
case 'm'
case 'p'
etc
```

Lab 3 – Requirements

A functionality

- Achieve required functionality.

  Use the ready bit of the flag register to trigger an interrupt. The ISR should store the samples (left and right), look for a triggering even, and signal when the stored samples should be transferred to the BRAM in the oscilloscope component.

  Add means to control Ch1_enb and Ch2_enb either by adding two FPGA board switches or controlled by microblaze terminal interface.

BONUS: add ability to change trigger from Ch1 to Ch2 using either switches or user menu

UNITED STATES AIR FORCE ACADEMY

Lab 3 – Requirements Turn In

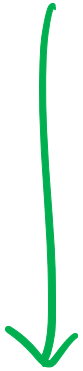
- **Turn In Requirements**
  - All your work in this lab is to be submitted using Bitbucket
  - README: only needs
    - Design
      - Updated lab2 block diagram with correct signals
      - Mapping of 32 AXI registers to lab2 signals
    - Evidence of completing Gate Checks 1 and 2, and 3 basic functionality, and A-functionality, along with the date/time achieved
    - Issues and lessons learned

UNITED STATES AIR FORCE ACADEMY

Dummy Waveforms for Testing

- Audio_Codec_Wrapper has sim_live
  - Use switch or C-code to toggle sim_live?

- Walk through example C-code test commands
  - see website lab2 testing section!
  - lessons learned the hard way



Implementation and Testing

Lessons from previous years:

- For all the lab2 signals you are reading into Microblaze AXI registers, add them to your C-code menu, using printf to print their values when you type the "?" command. This is very useful in debugging

Build your C-code incrementally, with baby-step tests such as these [and add each of these tests as one of your C-code menu options]:

- Draw a horizontal line on channel 1 and a diagonal line on channel 2, by writing the proper values to the BRAM [this test does not require your interface with the audio codec or the interrupt to work, and tests if you can write to the BRAM, and see the correct output on the scopeface]

```
pseudo code:
        case 'd':     // some of these will be XIL commands
             for (i=0;i<1024;i++) {
                          exWrAddr = i; // set BRAM address
                          exLBus = 185; // row for horz line
                                        // need to shift to upper 10bits?
                          exRBus = i;   // diagnonal line [need to shift?]
                          exWen = 1;    // write data to address in BRAM
                          exWen = 0;    // turn off write
             }
                        break;
```

- Use polling (polling the ready flag) to grab each sample and write them into your C-array. [this is a good test to see if you are getting the ready flag, able to read live samples, able to clear the flag, and repeat to fill your array]

```
pseudo code:
        case 'm':     // some of these will be XIL commands
             for (i=0;i<1024;i++) {
               while(flagQ==0){}; //wait on ready
               array_L[i] = LbusReg; // read audio values
               array_R[i] = RbusReg;
               ClearFlag = 1;   //clear the flag
               ClearFlag = 0;   //release the clear, so can be set again
             }
                        break;
```

- Use printf to print all the values in your array to the UART terminal [this is a good test to see if you are successfully reading audio codec samples in and storing them in your c arrary, or not]

```
case 'p':    // some of these will be XIL commands
               for (i=0; i<1024; i++){
                       printf("%x\r\n",[redacted]array_L[i][redacted]);
               }
               break;
```

- Write your C-array to BRAM (not triggered) [this is a good test to use, after you fill the array in the test above, to see if you can write it to the BRAM and see it appear on scopeface... and fix any calibration issues] simple change to case 'd'
- Use interrupts (ISR) to fill the C-array with samples [then use command above to write the array to BRAM]
- Write a command to search through the C-array to find the trigger point, and printf this location to the terminal
- Given this trigger location in the array, write the appropriate data values in the array to the BRAM, so the sine wave appears triggerd.
- Now you have all the pieces to create the "continuous" mode with interrupts and triggering

Note: When you are finding your trigger point in lab3, just like in lab2, you need to compare apples to apples. If your number in your C-array is 16-bits, it will be a large number in decimal like 26572, and this large 16-bit value you will want to write to your BRAM (as your VHDL code will later pull out the upper 10 or 9 bits) Meanwhile your triggerVolt is a small 10-bit number, like decimal 220. So to compare apples to apples in your C-code, (just for the purpose of finding the trigger) you might what to change the value in your C-array to the scale of the trigger volt by grabbing only its upper 10-bits (or in math, shift right 6 times, or divide by 2^6). Remember, in lab2 you probably also added (or subtracted?) an offset like the number like 34 (for the DC offset), so you would also need to add or subtract this from either trigvolt or the scale array number when you are doing the search for the trigger value.

Note: In the past, some students in their datapath hooked up L_Bus_out/R_Bus_Out directly to the 18-bit "signed" value coming out of the audio codec, not the 18-bit value converted to "unsigned". If you do this, your sine wave will look strange (or maybe off the screen)... this can also impact your triggering. At some point, you need to convert from signed to unsigned. If you didn't do this in your VHDL datapath, you could always do this in your C-code. If you are doing this conversion in C, remember you cannot just cast a variable from signed to unsigned. casting to unsigned in C would not do what you want (as we explained back in lab2). For example, if you have a signed value X = -7, and you cast X to an unsigned variable, what would -7 become? Unsigned values must be Zero or greater. -7 is undefined. If you go back to slide 11 in the lab2.pdf, you can see the pattern of this conversion with the 5 example numbers. It looks like all the lower bits stay the same, and only the MSB changes... it flips its value. I know two simple ways to flip the MSB (there are some other ways also), either (1) inverting the bit with a bitwise operator like XOR, or (2) adding a special number that keeps all the lower bits the same but only changes the MSB For method (2), adding the special number, look at the slide 11, you should be able to figure it out. For method (1), using XOR, ask yourself (a) what happens to a bit if I XOR it with ZERO?, and (b) what happens to a bit if I XOR it with ONE?

Note: The XIL in and out functions are either 8-bit, 16-bit, or 32-bit [like Xil_Out16(exWraddr, i], while you have some lab2 values that are different sizes (like trigvolt is 10-bits), so you may need to append bits in your VHDL code to make them match

See 383 website for other cases

- case 'w': W
  and see it
- case 't': Gi
- case 'z': G
- case 'g': N
- case 'i': Us
- case 'c': N

Lessons Learned the hard way

- when doing triggering math, use unsigned variables (u16, u8) not signed variables (int)
- Samples are 16-bits. Trig-volt is 10-bits. For triggering math, need to shift and offset to compare apples to apples
- Do not do anything slow in ISR or during grabbing samples from code like PRINTF()
- When grabbing 1024 samples, whether ISR or Polling, ONLY grab samples
  - Don't do anything else (like write to BRAM) cause too slow → miss samples → wave look higher freq
- Don't have ISR and POLLING sampling at same time!
  - Both try to clear flag → miss samples
  - disable_interrupts()
- Code for Sine Wave (see Dr York) → on 383 website

```
u16 sinFunc[64] = {128,141,153,165,177,189,200,210,219,227,235,241,246,250,253,255,
255,254,252,248,244,238,231,223,214,205,194,183,171,159,147,134,
122,109, 97, 85, 73, 62, 51, 42, 33, 25, 18, 12,  8,  4,  2,  1,
1,  3,  6, 10, 15, 21, 29, 37, 46, 56, 67, 79, 91,103,115,128};
```