SANTA CLARA UNIVERSITY
Electrical and Computer Engineering Department

ELEN 120 – Embedded Computing Systems

Lab 7 – A/D and D/A Converters

Dylan Thornburg, Sal Martinez, Gigi Patmore

Assignment: In this assignment you will use the A/D and D/A converters. You will make A/D readings from a potentiometer and use them. You will also use the D/A to synthesize waveforms that you can see on a scope. Finally, you will use these together.

Prelab:

Read Chapters 20 and 21 in the book. It is not critical to understand everything, but be familiar with what is taught there.

Review the register specifications and pre-supplied code discussed in the lab below.

Review this handout and develop a plan for writing and testing your code.

Problem 1

In the first problem, you will use the A/D converter to read the voltage on the center pin of a potentiometer. The potentiometer looks like this:

The specifications are here: https://www.mouser.com/ProductDetail/Bourns/PTV09A-4020FB503?qs=%2Fha2pyFaduhvapXs4is1IEX8wO9%252B7dElzCmfkzV%2FXz34t3woJ4J9WQ%3D%3D

The Lab 7 files directory has the datasheet.

Take 2 potentiometers. Bend back the two mounting pins on the sides and insert the 3 signal pins into the breadboard on 3 different rows, knob facing outwards. Make sure it is in tight – they don't fit well and will be a problem if they are not making contact.

Connect pin 1 of each potentiometer to the 3V supply output of the Discovery board. Connect pin 3 to ground. Connect pin 2 of one potentiometer to PA0 and pin 2 of the other potentiometer to PA1. Set each potentiometer to around the middle of its rotation.

Include a circuit diagram of these potentiometers in your report.

Copy the project shell ADC - redacted to your directory, unzip it, and open it.

I have provided an A/D converter initialization routine adc_init. It initializes the ADC subsystem to perform single conversions on ADC channel 6 which is connected to pin PA1. The range is approximately 0-3V and the readings will be 12-bits, right-aligned. You can and should use this routine to initialize the A/D converter.

I also wrote a routine called adc_read. adc_read performs a single conversion on channel 6 and returns the value in r0. Somehow you didn't get the code I wrote. You will need to rewrite it.

The ADC register specifications are in chapter 16 of the STM32L476VGT6 Reference manual.

Here are the steps you need to include to read channel 6 of the A/D converter: • You need to write a 1 to the ADC_CR_ADSTART bit in the ADC_CR register. This is actually harder than it sounds. The ADC_CR register has some bits that are normal r/w and should not be changed. Other bits, including ADC_CR_ADSTART are r/s type bits that take an action when written with a 1. You don't want to alter any of the r/w bits and you don't want to write any r/s bits other than the one you intend. So:

- Read the ADC_CR register
- Clear bits 0-5 and 31 in your copy. These are the r/s bits you don't want to write by mistake.
- Set bit 2 in your copy. (ADC_CR_ADSTART)
- Write your copy back to the ADC_CR register.

• Next you need to check if the A/D converter is busy and keep checking until it is not. You can do that by reading the ADC_CSR_EOC_MST bit in the ADC_CSR register. It will be set when the A/D conversion is complete. If it is cleared, you must wait.

• Finally, you can read the measurement value from the ADC_DR register and return it in r0.

Add all of these steps to adc_read so that it works again.

Now, test adc_read from main. It should return a measurement. When you turn the proper potentiometer (the one connected to PA1), the measurement reading should change.

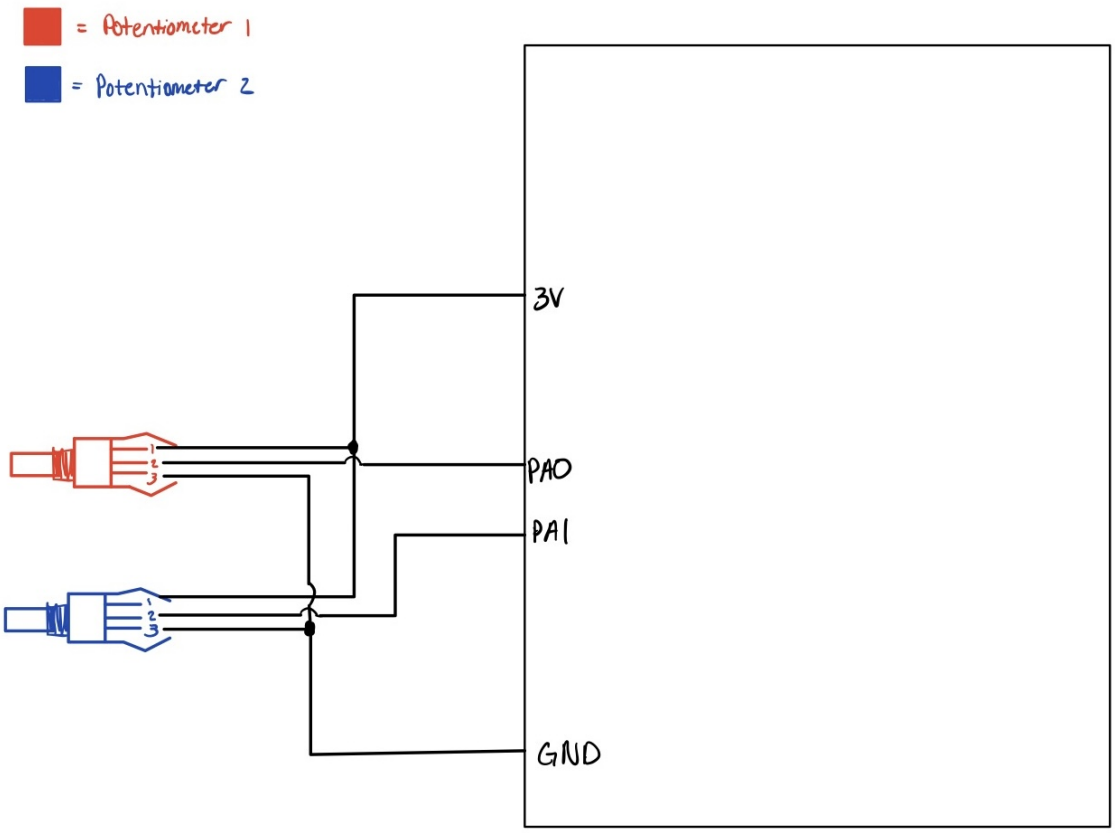
Now, add code to main that does the following:

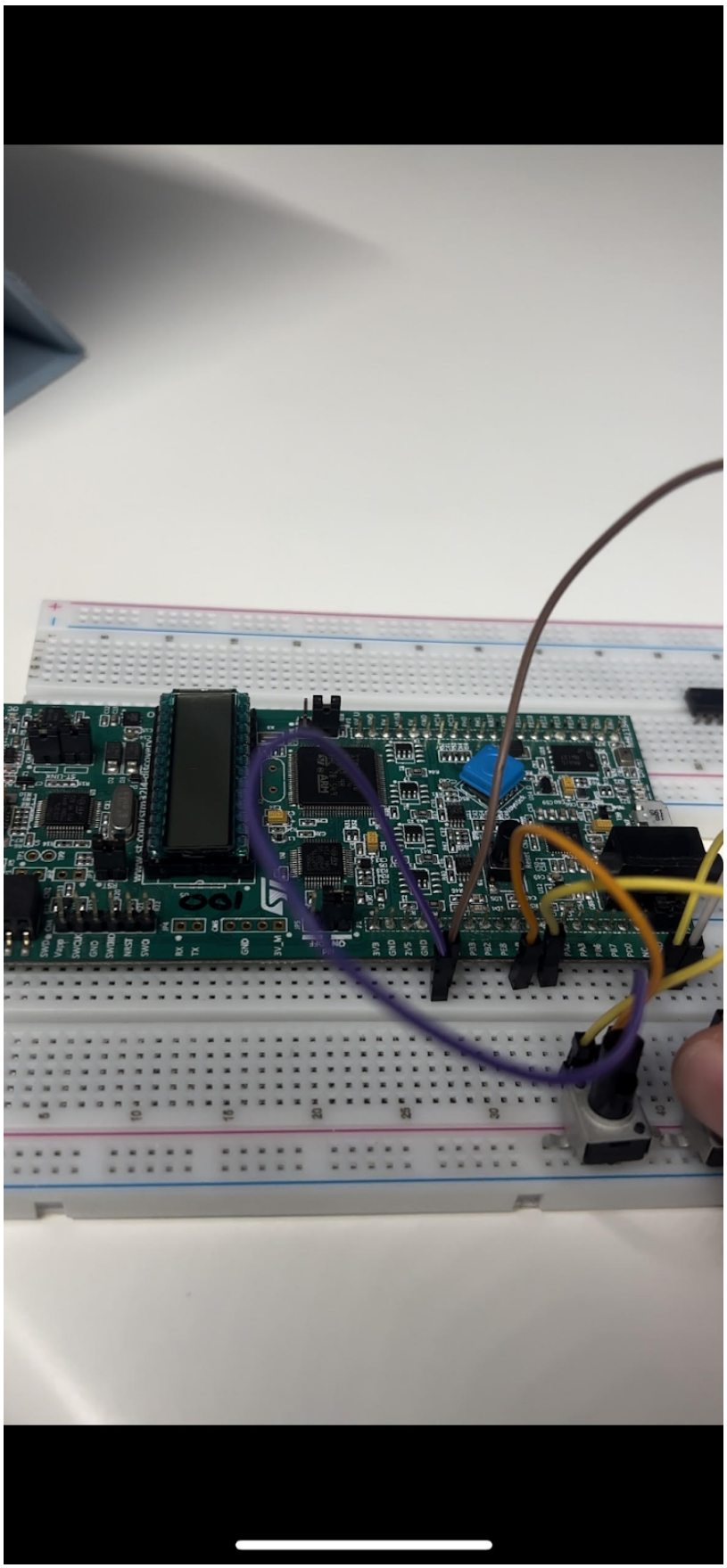
• Read the potentiometer reading with the A/D
• If the reading is >= 2048, turn the red LED on.
• If the reading is <2048, turn the red LED off.

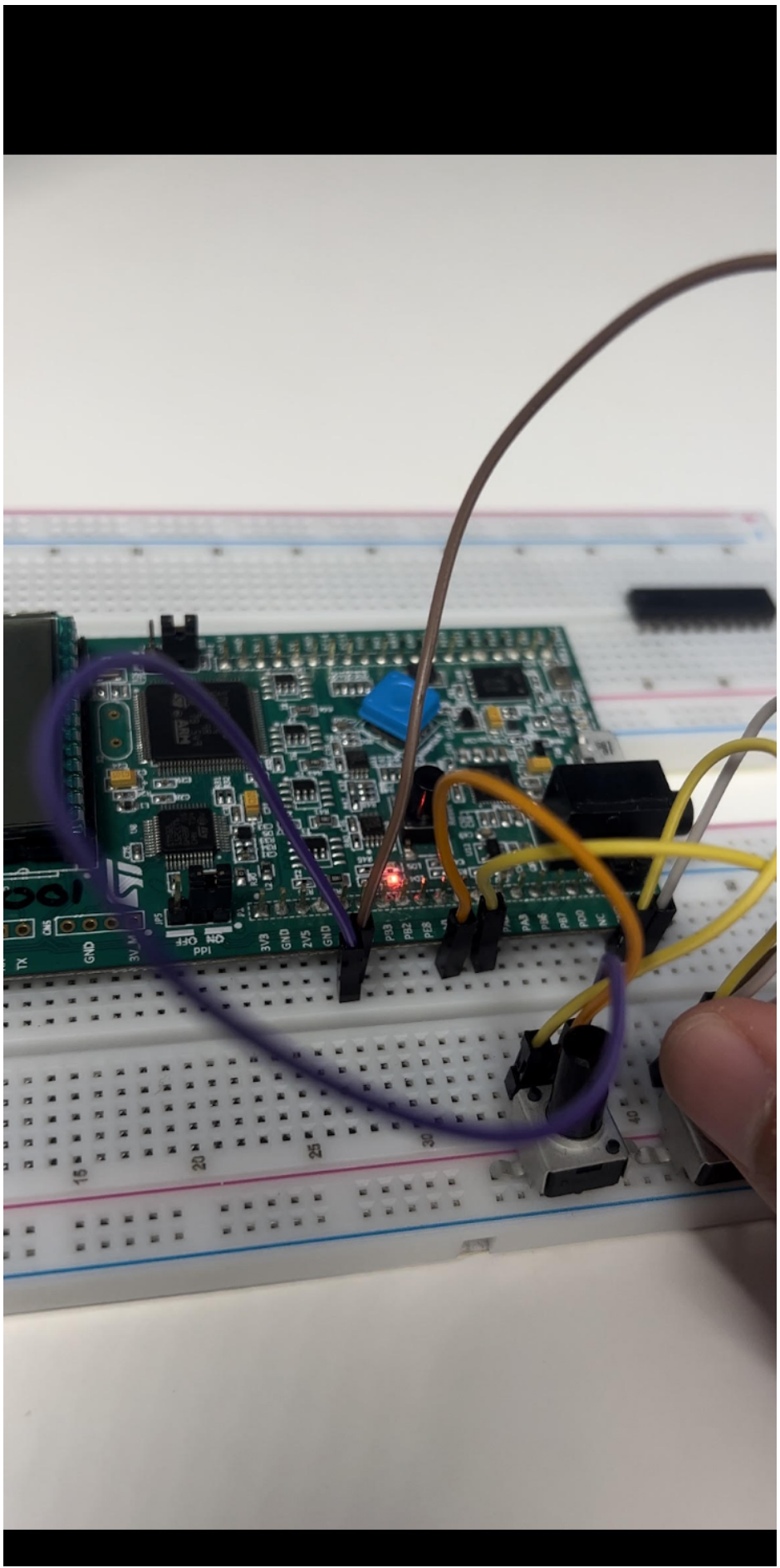
• Repeat forever.

You will need to add code to configure and control the red LED. You may use the routines I have provided in leds.s.

When you have it all running, turning the potentiometer knob should turn the red LED on and off. Demo for the TA and make a video for your report. Turn in your code.






GND
3V3
GND
2V5
GND
PB3
PB2
PB8
PA3
PB6
PB7
PD0
NC
PPI
OFF
ON
JP5
CN5
TX
15
20
25
30
40

Mains.s:

```
;******************** (C) Andrew Wolfe ********************************
; @file    main_hw_proto.s
; @author  Andrew Wolfe
; @date    August 29, 2019
; @note
; This code is for the book "Embedded Systems with ARM Cortex-M
; Microcontrollers in Assembly Language and C, Yifeng Zhu,
; ISBN-13: 978-0982692639, ISBN-10: 0982692633 as used at Santa Clara University
;**********************************************************************
        INCLUDE core_cm4_constants.s            ; Load Constant Definitions
        INCLUDE stm32l476xx_constants.s
        INCLUDE leds.h
        INCLUDE     adc.h

                    AREA    main, CODE, READONLY
                    EXPORT      __main
                    ENTRY
__mainPROC
; Add code here to configure the proper GPIO port to drive the red LED.
; You may use the routines provided in leds.s
                LDR r0, =(RCC_BASE + RCC_AHB2ENR) ;activates red clock
                LDR r1, [r0]
                ORR r1, #RCC_AHB2ENR_GPIOBEN
                STR r1, [r0]
                LDR r0, =(GPIOB_BASE+GPIO_MODER) ;sets up digital output
                LDR r1, [r0]
                BIC r1, r1, #(0x03<<(2*2))
                ORR r1, r1, #(1<<(2*2))
                STR r1, [r0]
                bl              adc_init

loop            bl              adc_read
                cmp r0, #2048
                blt     off
                bge on

off     bl red_off
                b loop
```

```
on              bl red_on
                b loop

; Add and/or modify code here to repeatedly read the A/D converter and turn the red LED on if
; the reading is greater than or equal to 2048 and turn off the red LED is the reading is less than
that.

endlessb                endless
                ENDP
                        ALIGN
                        AREA    myData, DATA, READWRITE
                        ALIGN
        END
```

adc.s:

```
adc_read        PROC            ;return ADC channel 6 value in r0
                                EXPORT      adc_read
                ldr r1, =(ADC1_BASE+ADC_CR)
                ldr r2, [r1]
                bic r2, #0x80000000
                bic r2, #0x0000003f
                orr r2, #ADC_CR_ADSTART
                str r2, [r1]
loop    ldr r1, =(ADC1_BASE + ADC_CSR)
                ldr r2, [r1]
                cmp r2, #ADC_CSR_EOC_MST
                bne out
                b loop
out             ldr r1, =(ADC1_BASE+ADC_DR)
                ldr r0, [r1]

                bx              lr
```

Problem 2

For problem 2 you will use the DAC to create a sine wave using a wave table and an arbitrary waveform generator. Since you are real engineers, you will generate samples at a known, fixed frequency so that you can analyze the Nyquist rate and someday (not in this lab) build a proper filter.

I have provided you with a project framework called wave3_redacted. (Don't ask about wave1 or wave2, it's not something you want to know. Kind of like those first-attempt clones that are always locked in the basement of every sci-fi story.) Copy the framework and rename the directory.

The overall project works as follows:

- We will configure a timer interrupt to interrupt the CPU at a fixed rate (20KHz).
- The DAC will output data at this rate from within the timer interrupt handler.
- A wavetable will provide the signal data. You must properly sample the wavetable to provide a target output frequency.
- The DAC output will appear at pin PA5. You can connect this to an oscilloscope to see the waveform you are producing.

I have actually given you the code for the __main procedure. The target frequency (test_freq) is used to calculate a phase increment value that is the amount that one must step through the waveform table. The waveform table contains 1024 entries corresponding to an entire 2π radian cycle. Each entry is 2 bytes (16-bits) with the first 4 bits being 0 and the last 12 being a waveform value ranging from 0-4095. Based on this information, the output sample rate, and the target signal frequency, the distance between output samples from the table must be calculated.

Since we would get some weird results at certain frequencies if we used integers, we use 16ths instead. I.E., the distance between entry 1 and entry 2 in the wave table is 16. Routine calc_phaseinc computes the distance between subsequent DAC output values in 16ths. Or at least it will once you fill in the code. Pass the target frequency in r0 and return the phase increment (in 16ths) in r0. Write and test this code.

Next, the code initializes the DAC to output on pin PA5. I have given you this code. You should read and understand it as part of the prelab.

Next, the timer (Timer 2) is initialized and set to a 20KHz interrupt rate.
This project is configured for the default 4MHz clock frequency. I have provided you with two routines to configure Timer 2. Routine tim2_init configures the registers for Timer 2 and enables the interrupt. It sets the timer count rate to 1µs for a 4MHz clock and sets up a 1ms timer interrupt rate as the default.

Routine tim2_freq resets the Timer 2 interrupt rate. Provide a period in µs in r0 and call this routine. The defined values sample_freq and sample_per in main.s define a 20KHz interrupt rate.

You will use this as already present in main to set the interrupt rate to 20KHz.

Once these are called, you enter the endless loop, but 20K times per second, the timer will trigger the timer interrupt TIM2_IRQHandler to output a sample to the DAC. It saves the lr for later so it can call other routines. It then loads a pointer to the wavetable into r1 and the current phase (the offset into the wavetable, in 16ths) into r0 and calls get_tblval. This routine looks up the corresponding output value in the wave table and returns it in r0. Or at least it will when you write it.

This output value is then sent to the DAC. After that, "phase" must be incremented. That means:

- Add the phase increment to phase
- If it exceeds (1024 16ths)-1. Then it needs to be wrapped around (that's what periodic means)

The routine update_phase does this. Or at least it will when you write it.

Test all of the code you wrote then get it all running. If you get it right, the code will produce a 600Hz Sine wave at pin PA5.

Use an oscilloscope to view the signal.

Demo for the TA and take a scope picture for your report. Turn in your code.

Code:

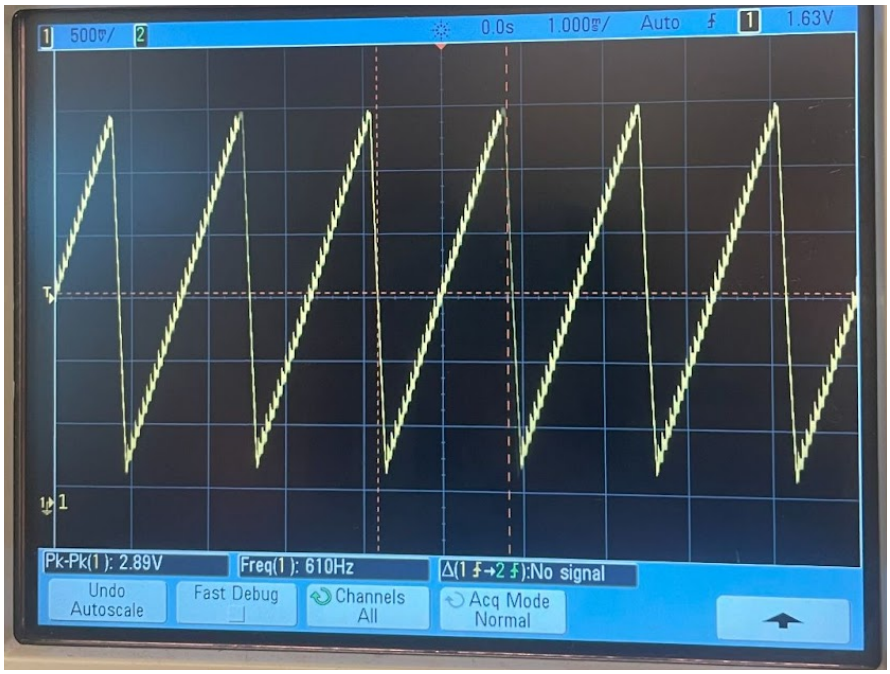
```
;******************** (C) Andrew Wolfe ****************************************
; @file    main_hw_proto.s
; @author  Andrew Wolfe
; @date    August 18, 2019
; @note
;         This code is for the book "Embedded Systems with ARM Cortex-M
;         Microcontrollers in Assembly Language and C, Yifeng Zhu,
;         ISBN-13: 978-0982692639, ISBN-10: 0982692633 as used at Santa Clara University
;******************************************************************************


  INCLUDE core_cm4_constants.s          ; Load Constant Definitions
  INCLUDE stm32l476xx_constants.s
  INCLUDE timer.h
  INCLUDE leds.h
  INCLUDE   dac.h
  IMPORT    sintbl
  IMPORT    sawtbl

sample_freq   equ   20000         ;20KHz sampling rate
sample_per    equ   1000000/sample_freq
test_freq     equ   600

              AREA    main, CODE, READONLY
              EXPORT      __main
              ENTRY

__mainPROC
        ldr           r0,=test_freq
        bl            calc_phaseinc  ;compute the phase increment value (phaseinc)
        ldr           r2,=phaseinc
        str           r0,[r2]        ;store the phase increment value in memory

        bl            dac_init                  ;initialize dac
        bl            tim2_init                 ;initialize timer interrupt
        ldr           r0,=sample_per            ;set output rate to 20KHz
        bl            tim2_freq



endlessb              endless
```

```
        ENDP


TIM2_IRQHandler PROC
        EXPORT      TIM2_IRQHandler
        push    {lr}
        ldr             r0,=phase                   ;get a pointer to the current phase
        ldr             r1,=sintbl                  ;Get pointer to waveform table | change to
 sawtbl for saw table values
        bl              get_tblval
        bl              dac_set
        ldr             r1,=phaseinc            ;load phase increment
        ldr             r0,=phase                   ;reload last phase value
        bl              update_phase
        pop             {lr}
        ldr             r2,=(TIM2_BASE+TIM_SR) ;reset pending interrupt for TIM2
        mov             r1,#~TIM_SR_UIF
        str             r1,[r2]
        dsb
        bx              lr
        ENDP

calc_phaseinc  PROC
        ; To calculate the phaseinc, take the new frequency (w)/sampling freq.(w0) * 1024
        ; to avoid precision issues - we will keep phase in 16ths then divide at the last minute
        ; w arrives in r0; phase increment returned in r0
        ; works from about 2Hz to sampling freq./2
        ; Assumes a wave table size of 1024 and a phase iterator scaled up by 16
        ;Put your code here.

        LDR r1, =sample_freq
        LSL r0, #4
        LSL r0, #10
        UDIV r0, r0, r1

        bx              lr
        ENDP

update_phase  PROC
                        ;receives a pointer to phase in r0 and a pointer to phaseinc in r1
                        ;adds phaseinc to phase
;Put your code here.
```

```
        LDR r2, [r0]
        LDR r3, [r1]
        ADD r3, r2
        BIC r3, #0x4000
        STR r3, [r0]
        bx          lr
        ENDP

get_tblval      PROC
                ;receives a pointer to phase in r0 and a pointer to a wave table in r1
                ;Assume the wave table is 1024 entries; 16-bits each
                ;Assume the phase value is in 16ths.
                ;Return the sample in r0

;Put your code here.
        LDR r2, [r0]
        LSR r2, #4
        LSL r2, #1
        ADD r1, r2
        LDRH r0, [r1]
        bx          lr
        ENDP


        ALIGN
        AREA    myData, DATA, READWRITE

        ALIGN
phase   dcd     0                       ;maintain in 16ths.
phaseinc dcd    160

 END
```

Problem 3

Change sintbl in TIM2_IRQHandler to sawtbl. Determine what changed in your program. Demo for the TA and take a scope picture for your report.



Graph changed from a sine wave to a sawtooth graph.

```
;******************** (C) Andrew Wolfe ****************************************
; @file    main_hw_proto.s
; @author  Andrew Wolfe
; @date    August 18, 2019
; @note
;         This code is for the book "Embedded Systems with ARM Cortex-M
;         Microcontrollers in Assembly Language and C, Yifeng Zhu,
;         ISBN-13: 978-0982692639, ISBN-10: 0982692633 as used at Santa Clara University
;******************************************************************************



        INCLUDE core_cm4_constants.s        ; Load Constant Definitions
        INCLUDE stm32l476xx_constants.s
        INCLUDE timer.h
        INCLUDE leds.h
        INCLUDE     dac.h
        IMPORT      sintbl
        IMPORT      sawtbl

sample_freq   equ     20000          ;20KHz sampling rate
sample_per    equ     1000000/sample_freq
test_freq     equ     600
```

```
                AREA    main, CODE, READONLY
                EXPORT      __main
                ENTRY

__mainPROC
                ldr         r0,=test_freq
                bl          calc_phaseinc ;compute the phase increment value (phaseinc)
                ldr         r2,=phaseinc
                str         r0,[r2]       ;store the phase increment value in memory

                bl          dac_init                ;initialize dac
                bl          tim2_init               ;initialize timer interrupt
                ldr         r0,=sample_per          ;set output rate to 20KHz
                bl          tim2_freq



endlessb            endless
                ENDP


TIM2_IRQHandler PROC
                EXPORT      TIM2_IRQHandler
                push    {lr}
                ldr         r0,=phase               ;get a pointer to the current phase
                ldr         r1,=sawtbl              ;Get pointer to waveform table |
change to sawtbl for saw table values
                bl          get_tblval
                bl          dac_set
                ldr         r1,=phaseinc        ;load phase increment
                ldr         r0,=phase           ;reload last phase value
                bl          update_phase
                pop         {lr}
                ldr         r2,=(TIM2_BASE+TIM_SR) ;reset pending interrupt for TIM2

                mov         r1,#~TIM_SR_UIF
                str         r1,[r2]
                dsb
                bx          lr
                ENDP

calc_phaseinc  PROC
        ; To calculate the phaseinc, take the new frequency (w)/sampling freq.(w0) * 1024
```

```
        ; to avoid precision issues - we will keep phase in 16ths then divide at the last minute
        ; w arrives in r0; phase increment returned in r0
        ; works from about 2Hz to sampling freq./2
        ; Assumes a wave table size of 1024 and a phase iterator scaled up by 16

;Put your code here.
                LDR r1, =sample_freq
                LSL r0, #4
                LSL r0, #10
                UDIV r0, r0, r1

                bx              lr
                ENDP

update_phase  PROC
        ;receives a pointer to phase in r0 and a pointer to phaseinc in r1
        ;adds phaseinc to phase

;Put your code here.
                LDR r2, [r0]
                LDR r3, [r1]
                ADD r3, r2
                BIC r3, #0x4000
                STR r3, [r0]
                bx              lr
                ENDP

get_tblval              PROC
        ;receives a pointer to phase in r0 and a pointer to a wave table in r1
        ;Assume the wave table is 1024 entries; 16-bits each
        ;Assume the phase value is in 16ths.
        ;Return the sample in r0

;Put your code here.
                LDR r2, [r0]
                LSR r2, #4
                LSL r2, #1
                ADD r1, r2
                LDRH r0, [r1]
                bx              lr
                ENDP


                        ALIGN
```

```
                AREA    myData, DATA, READWRITE

                ALIGN
phase           dcd         0                               ;maintain in 16ths.
phaseinc        dcd         160

        END
```

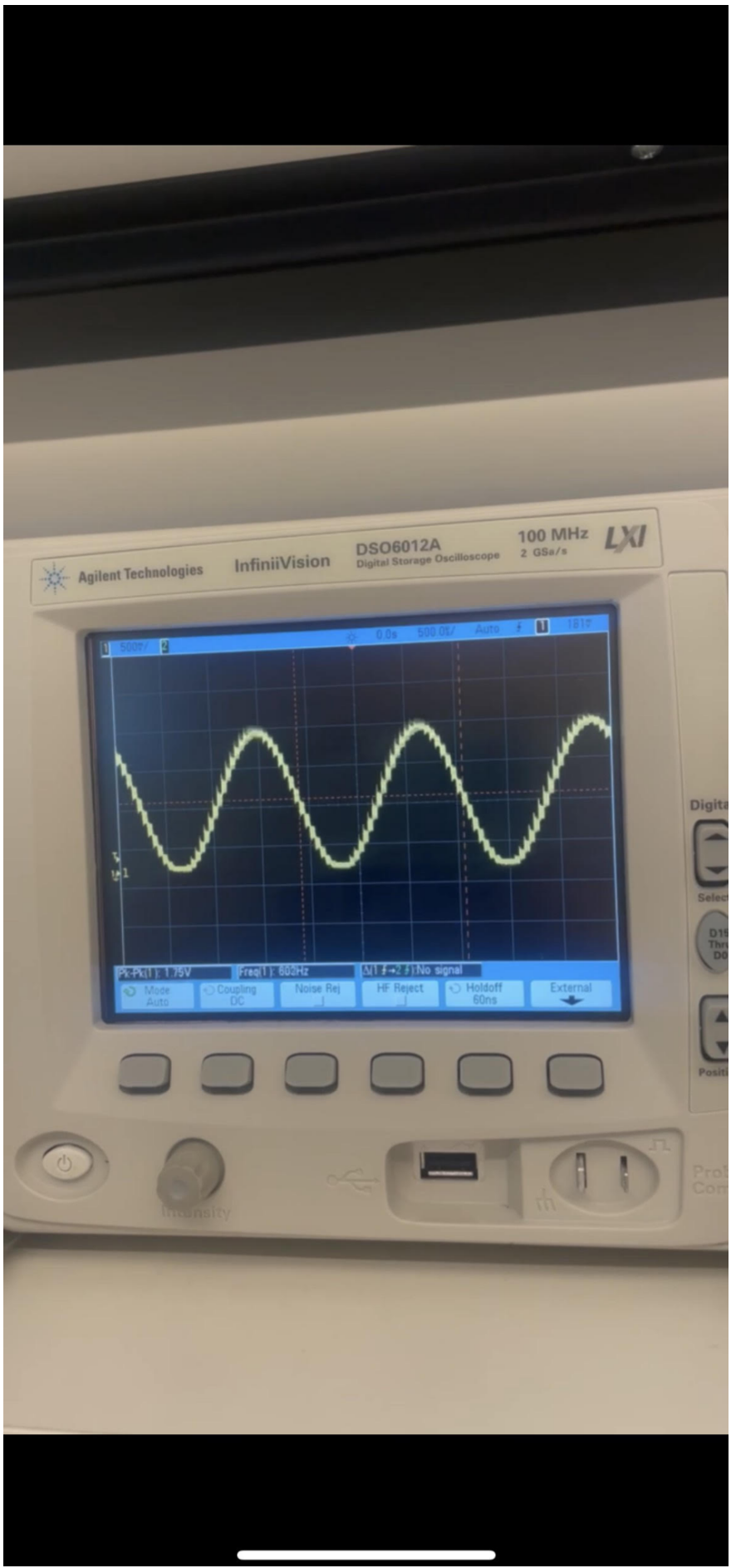
Problem 4

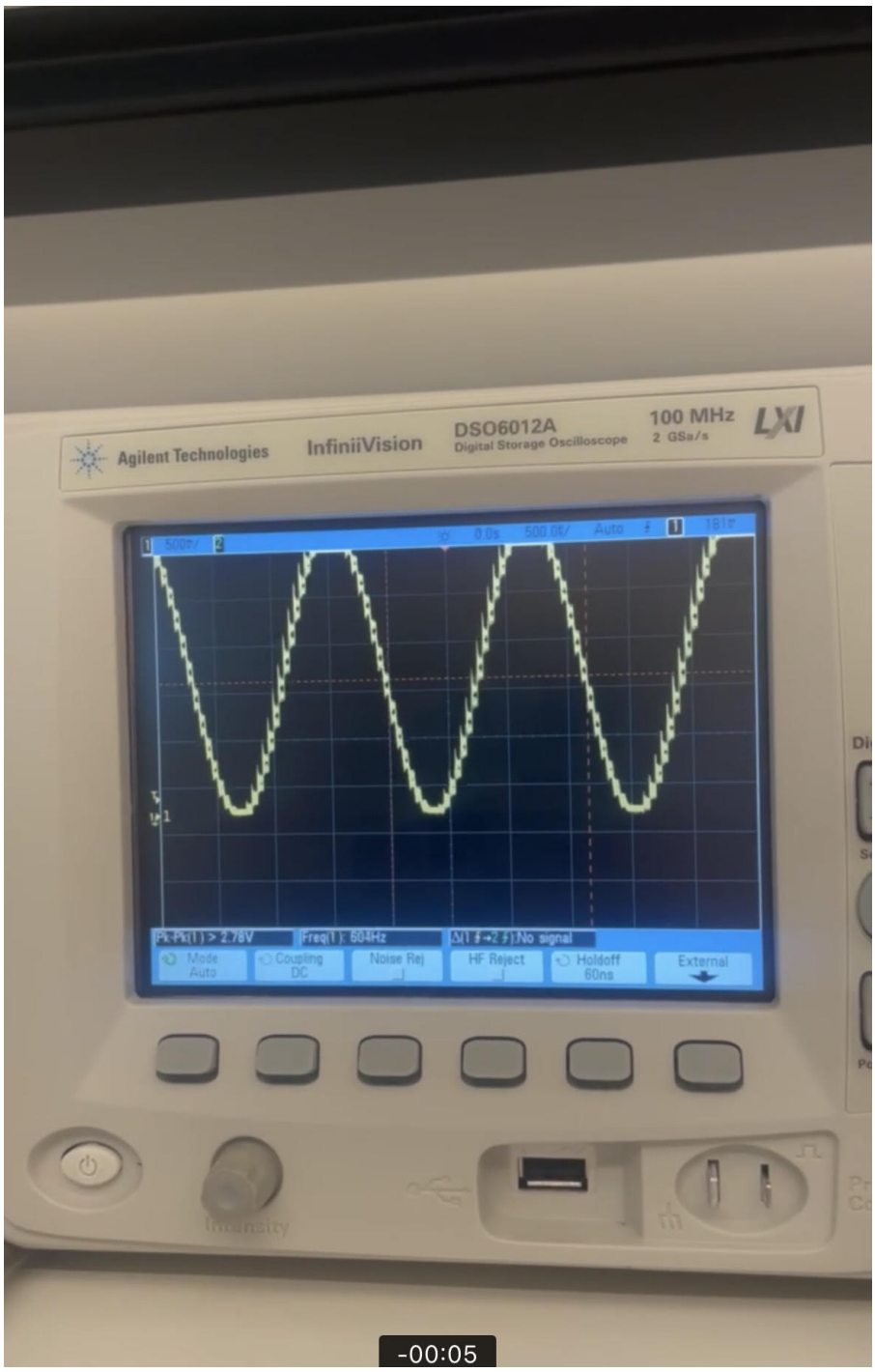
Now make a copy of your problem 2 solution (change back to sintbl). Add adc.h and adc.s to this project. Add a word variable called gain to the data section.

Now, in the main part of your program, read the A/D converter channel 6 repeatedly and store the value in gain.

Modify the code so that the value sent to the D/A is multiplied by gain/4096. Think about how to do this math properly. Once it works, the potentiometer connected to PA1 should become a volume control that makes your output louder and softer.

Demo for the TA and take a scope video for your report. Turn in your code.


Agilent Technologies
InfiniiVision
DSO6012A
Digital Storage Oscilloscope
100 MHz
2 GSa/s
LXI
Pk-Pk(1): 1.75V
Freq(1): 603Hz
No signal
Mode
Auto
Coupling
DC
Noise Rej
HF Reject
Holdoff
60ns
External
Intensity



Main code:

```
;******************** (C) Andrew Wolfe ************************************
; @file    main_hw_proto.s
; @author  Andrew Wolfe
; @date    August 18, 2019
; @note
;          This code is for the book "Embedded Systems with ARM Cortex-M
;          Microcontrollers in Assembly Language and C, Yifeng Zhu,
;          ISBN-13: 978-0982692639, ISBN-10: 0982692633 as used at Santa Clara University
```

```
;*******************************************************************************


        INCLUDE core_cm4_constants.s        ; Load Constant Definitions
        INCLUDE stm32l476xx_constants.s
        INCLUDE timer.h
        INCLUDE leds.h
        INCLUDE     dac.h
        INCLUDE     adc.h
        IMPORT      sintbl
        IMPORT      sawtbl

sample_freq     equ     20000           ;20KHz sampling rate
sample_per      equ     1000000/sample_freq
test_freq       equ     600

                AREA    main, CODE, READONLY
                EXPORT      __main
                ENTRY

__mainPROC
                ldr             r0,=test_freq
                bl              calc_phaseinc       ;compute the phase increment value
(phaseinc)
                ldr             r2,=phaseinc
                str             r0,[r2]                     ;store the phase increment value in
memory

                bl              dac_init                    ;initialize dac
                bl              tim2_init                   ;initialize timer interrupt
                ldr             r0,=sample_per              ;set output rate to 20KHz
                bl              tim2_freq

                bl              adc_init
loop    bl              adc_read
                ldr r1, =gain
                str r0, [r1]
                b loop
```

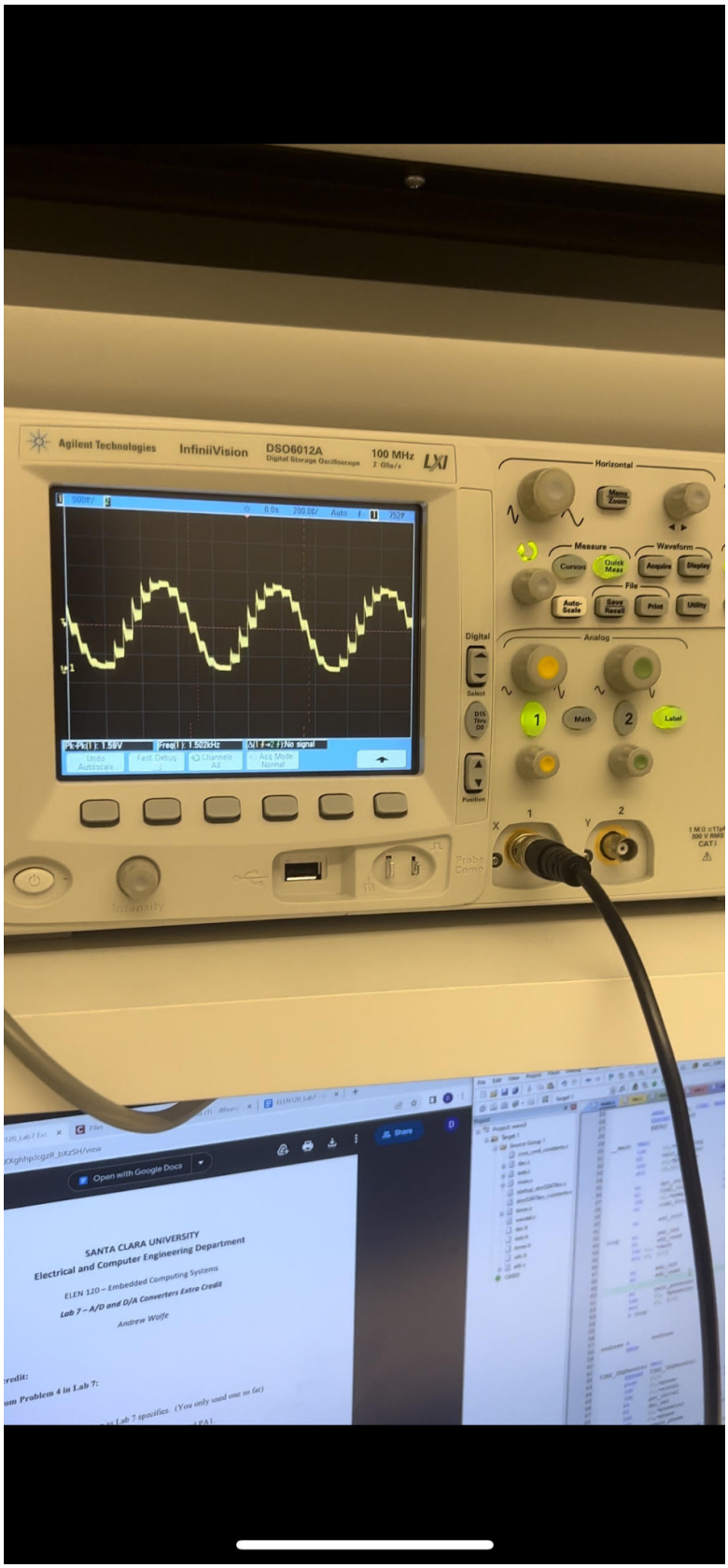
```
endlessb        endless
                ENDP


TIM2_IRQHandler PROC
                EXPORT      TIM2_IRQHandler
                push    {lr}
                ldr         r0,=phase                   ;get a pointer to the current phase
                ldr         r1,=sintbl                  ;Get pointer to waveform table |
change to sawtbl for saw table values
                bl          get_tblval
                bl          dac_set
                ldr         r1,=phaseinc        ;load phase increment
                ldr         r0,=phase                   ;reload last phase value
                bl          update_phase
                pop         {lr}
                ldr         r2,=(TIM2_BASE+TIM_SR) ;reset pending interrupt for TIM2

                mov         r1,#~TIM_SR_UIF
                str         r1,[r2]
                dsb
                bx          lr
                ENDP

calc_phaseinc  PROC
                                ; To calculate the phaseinc, take the new frequency
(w)/sampling freq.(w0) * 1024
                                ; to avoid precision issues - we will keep phase in 16ths then
divide at the last minute
                                ; w arrives in r0; phase increment returned in r0
                                ; works from about 2Hz to sampling freq./2
                                ; Assumes a wave table size of 1024 and a phase iterator
scaled up by 16

;Put your code here.
                LDR r1, =sample_freq
                LSL r0, #4
                LSL r0, #10
                UDIV r0, r0, r1

                bx          lr
                ENDP
```

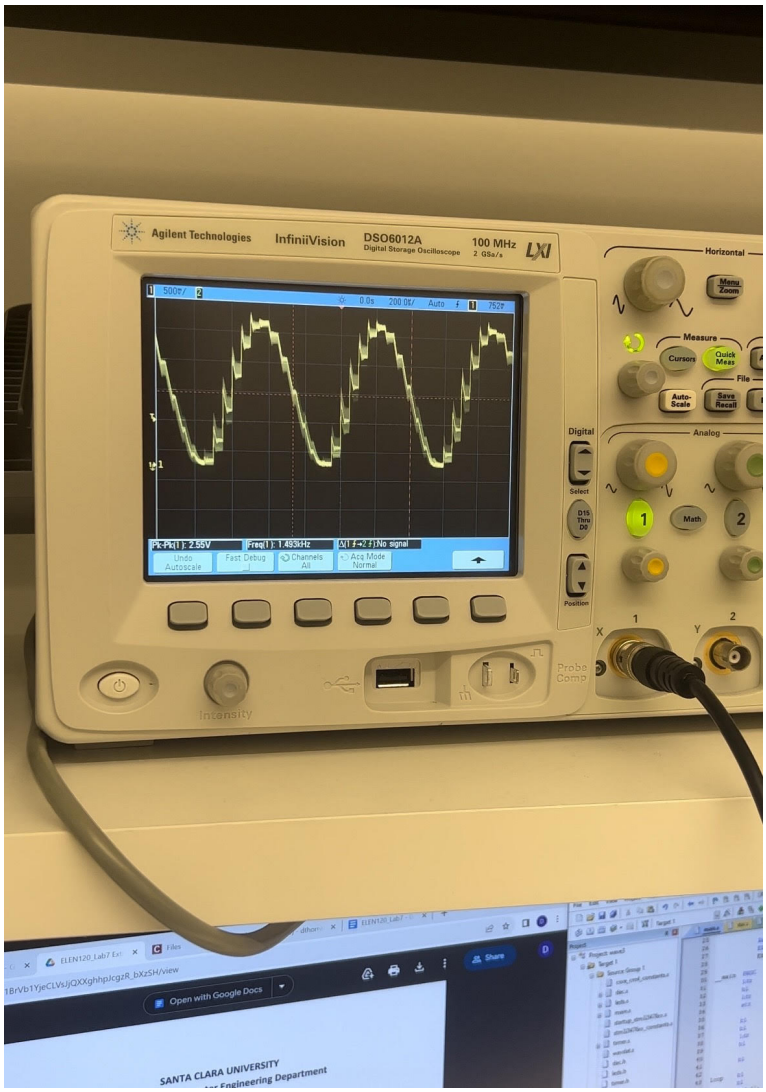
```
update_phase  PROC
                                        ;recieves a pointer to phase in r0 and a pointer to phaseinc in
r1
                                        ;adds phaseinc to phase
;Put your code here.
                LDR r2, [r0]
                LDR r3, [r1]
                ADD r3, r2
                BIC r3, #0x4000
                STR r3, [r0]
                bx              lr
                ENDP

get_tblval              PROC
                                        ;recieves a pointer to phase in r0 and a pointer to a wave
table in r1
                                        ;Assume the wave table is 1024 entries; 16-bits each
                                        ;Assume the phase value is in 16ths.
                                        ;Return the sample in r0

;Put your code here.
                LDR r2, [r0]
                LSR r2, #4
                LSL r2, #1
                ADD r1, r2
                LDRH r0, [r1]

                LDR r8, =gain
                LDR r9, [r8]
                MUL r0, r9
                LSR r0, #12
                bx              lr
                ENDP


                        ALIGN
                        AREA    myData, DATA, READWRITE

                        ALIGN
phase           dcd             0                               ;maintain in 16ths.
phaseinc        dcd             160
```

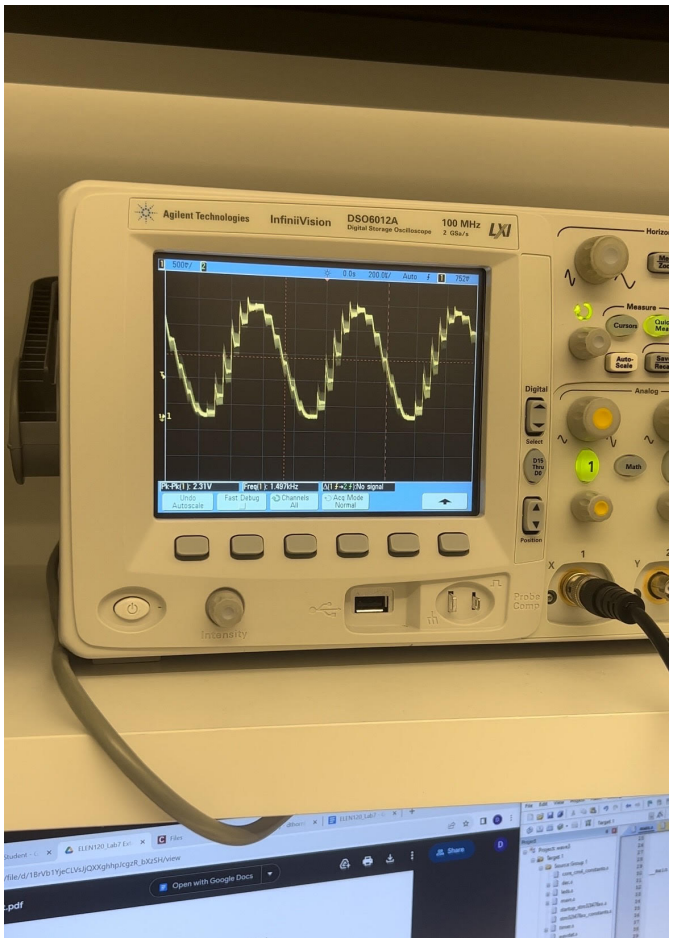
```
gain            dcd     0

        END
```

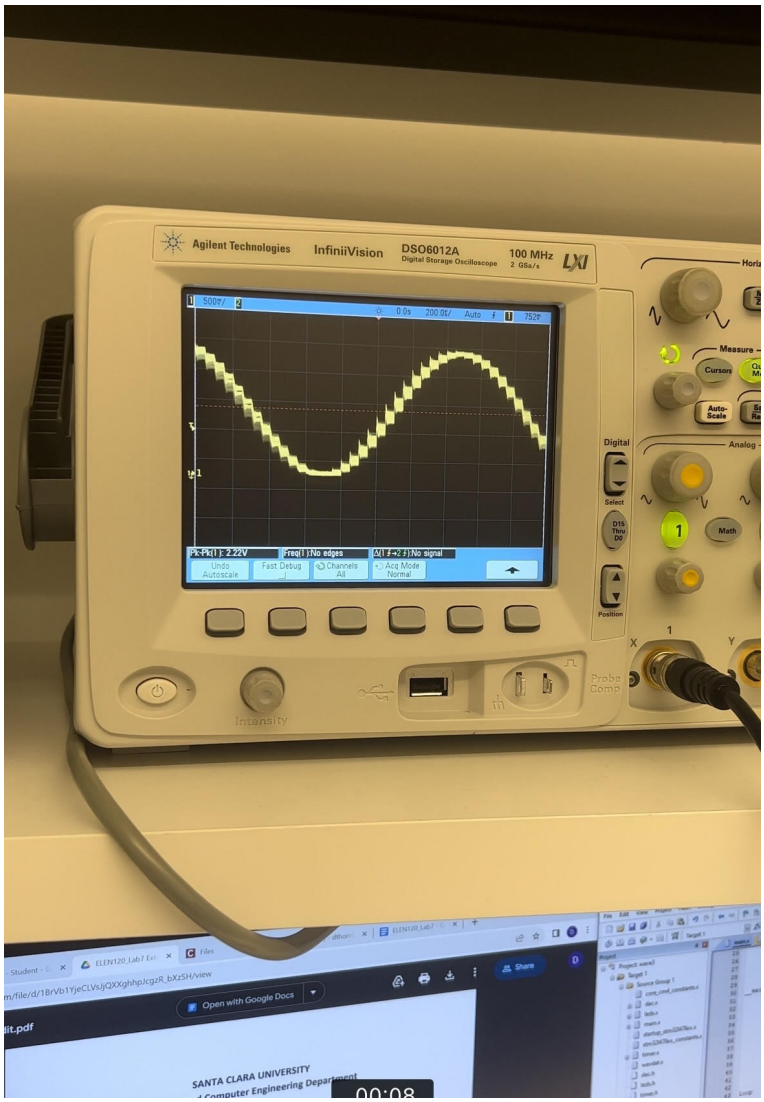
Problem 5 (extra credit):

Problem 5
You need 2 potentiometers hooked up as Lab 7 specifies. (You only used one so far)
• Modify adc_init so that it enables and configures both PA0 and PA1.
• Modify adc_init so that it configures the analog switch for both pins
• Modify adc_init so that it configures both channels 5 and 6 to single-ended mode
• Modify adc_init so that it configures the sample time for both channels 5 and 6
• Write a routine adc_ch5 that switches the active ADC channel to channel 5
• Write a routine adc_ch6 that switches the active ADC channel to channel 6
• Modify your main loop to measure PA0 on channel 5 as well as PA1 on channel 6. Test to make sure you can measure both potentiometers.
• Use the value measured on PA0 to vary the frequency of your sine wave from 300-600Hz. (PA1 still adjusts the amplitude)
Demo for the TA and take a scope video for your report. Turn in your code.


Agilent Technologies
InfiniiVision
DSO6012A
Digital Storage Oscilloscopes
100 MHz
2 GSa/s
LXI
Horizontal
Measure
Waveform
File
Analog
Digital
Intensity
Probe Comp
Open with Google Docs
SANTA CLARA UNIVERSITY
Electrical and Computer Engineering Department
ELEN 120 – Embedded Computing Systems
Lab 7 – A/D and D/A Converters Extra Credit
Andrew Wolfe


Agilent Technologies
InfiniiVision
DSO6012A
Digital Storage Oscilloscope
100 MHz
2 GSa/s
LXI
Horizontal
Menu Zoom
Measure
Cursors
Quick Meas
File
Auto-Scale
Save Recall
Analog
Digital
Select
Math
Position
Intensity
Probe Comp
Open with Google Docs
SANTA CLARA UNIVERSITY


Agilent Technologies
InfiniiVision
DSO6012A
Digital Storage Oscilloscope
100 MHz
2 GSa/s
LXI
Measure
Cursors
Auto-Scale
Digital
Analog
Math
Intensity
Probe Comp
Undo Autoscale
Fast Debug
Channels All
Acq Mode Normal
Open with Google Docs
Share



Adc code:

```
        INCLUDE core_cm4_constants.s        ; Load Constant Definitions
        INCLUDE stm32l476xx_constants.s

        AREA    main, CODE, READONLY

; Interface routines for ADC1 Channel 6
; Available at pin PA 1

; ADC intitialization derived from Figure 20-12 in Zhu
adc_init        PROC
                EXPORT      adc_init
                ldr         r2,=(RCC_BASE+RCC_CR)       ;Turn on HSI clock
                ldr         r1,[r2]
                orr         r1,#RCC_CR_HSION
                str         r1,[r2]
adcw1  ldr          r1,[r2]                             ;Wait for HSI Ready
```

```
                tst             r1,#RCC_CR_HSIRDY
                beq             adcw1
                ldr             r2,=(RCC_BASE+RCC_AHB2ENR)              ;Turn on port A
clock (bit 0)
                ldr             r1,[r2]
                orr             r1,#RCC_AHB2ENR_GPIOAEN
                str             r1,[r2]
                ldr             r2,=(GPIOA_BASE+GPIO_MODER)             ;program bits
2-3 in GPIOA_MODER for pin 1 analog mode (11)
                ldr             r1,[r2]
                orr             r1,#(0xf)
                str             r1,[r2]
                ldr             r2,=(GPIOA_BASE+GPIO_ASCR)          ;Enable the analog
switch ASC1
                ldr             r1,[r2]
                orr             r1,#GPIO_ASCR_EN_0
                orr             r1,#GPIO_ASCR_EN_1
                str             r1,[r2]
                ldr             r2,=(RCC_BASE+RCC_AHB2ENR)              ;Turn on ADC
clock
                ldr             r1,[r2]
                orr             r1,#RCC_AHB2ENR_ADCEN
                str             r1,[r2]
                ldr             r2,=(RCC_BASE+RCC_AHB2RSTR)             ;reset ADC
                ldr             r1,[r2]
                orr             r1,#RCC_AHB2RSTR_ADCRST
                str             r1,[r2]
                mov             r0,#10
adcw11          subs    r0,#1
                bne             adcw11
                bic             r1,#RCC_AHB2RSTR_ADCRST
                str             r1,[r2]
                ldr             r2,=(ADC1_BASE+ADC_CR)              ;Disable ADC1
                ldr             r1,[r2]
                bic             r1,#ADC_CR_ADEN
                str             r1,[r2]
                ldr             r2,=(SYSCFG_BASE+SYSCFG_CFGR1)          ;Enable analog
switch booster
                ldr             r1,[r2]
                orr             r1,#SYSCFG_CFGR1_BOOSTEN
                str             r1,[r2]
                ldr             r2,=(ADC1_BASE+ADC_CCR)             ;Turn on vfrefen;set
prescaler to 0000;set ckmode; independent mode
                ldr             r1,[r2]
                orr             r1,#ADC_CCR_VREFEN
                bic             r1,#ADC_CCR_PRESC
                bic             r1,#ADC_CCR_CKMODE
                orr             r1,#ADC_CCR_CKMODE_0
                bic             r1,#ADC_CCR_DUAL
```

```
                orr             r1,#6
                str             r1,[r2]
                ldr             r2,=(ADC1_BASE+ADC_CR)          ;power up ADC1 and
voltage regulator
                ldr             r1,[r2]
                bic             r1,#ADC_CR_DEEPPWD
                str             r1,[r2]
                orr             r1,#ADC_CR_ADVREGEN
                str             r1,[r2]
                mov             r1,#(28 * 20)                   ;About 22µs delay at
80MHz
adcw2  subs     r1,#1
                bne             adcw2
                ldr             r2,=(ADC1_BASE+ADC_CFGR)        ;set resolution to 12
bits, right aligned
                ldr             r1,[r2]
                bic             r1,#ADC_CFGR_RES
                bic             r1,#ADC_CFGR_ALIGN
                str             r1,[r2]

                ldr             r2,=(ADC1_BASE+ADC_DIFSEL)      ;set channel 6 to
single-ended
                ldr             r1,[r2]
                bic             r1,#ADC_DIFSEL_DIFSEL_5
                bic             r1,#ADC_DIFSEL_DIFSEL_6
                str             r1,[r2]
                ldr             r2,=(ADC1_BASE+ADC_SMPR1)       ;set channel 6 sample
time to 47 clocks
                ldr             r1,[r2]
                bic             r1,#ADC_SMPR1_SMP5
                orr             r1,#(3<<15)
                bic             r1,#ADC_SMPR1_SMP6
                orr             r1,#(3<<18)
                str             r1,[r2]
                ldr             r2,=(ADC1_BASE+ADC_CFGR)        ;select discontiguous
mode; software trigger
                ldr             r1,[r2]
                bic             r1,#ADC_CFGR_CONT
                bic             r1,#ADC_CFGR_EXTEN
                str             r1,[r2]
                ldr             r2,=(ADC1_BASE+ADC_CR)          ;Enable ADC1
                ldr             r1,[r2]
                orr             r1,#ADC_CR_ADEN
                str             r1,[r2]
                ldr             r2,=(ADC1_BASE+ADC_ISR)         ;wait for ADC ready
adcw3  ldr              r1,[r2]
                tst             r1,#ADC_ISR_ADRDY
                beq             adcw3
```

```
                bx              lr
                ENDP

adc_ch5     PROC
                EXPORT adc_ch5
                ldr             r2,=(ADC1_BASE+ADC_SQR1)            ;set conversion
sequence to 1 channel, channel 5
                ldr             r1,[r2]
                bic             r1,#ADC_SQR1_L
                bic             r1,#ADC_SQR1_SQ1
                orr             r1,#(5 << 6)
                str             r1,[r2]
                bx lr

                ENDP



adc_ch6     PROC
                EXPORT adc_ch6
                ldr             r2,=(ADC1_BASE+ADC_SQR1)            ;set conversion
sequence to 1 channel, channel 6
                ldr             r1,[r2]
                bic             r1,#ADC_SQR1_L
                bic             r1,#ADC_SQR1_SQ1
                orr             r1,#(6 << 6)
                str             r1,[r2]
                bx lr
                ENDP

adc_read        PROC            ;return ADC channel 6 value in r0
                                EXPORT      adc_read
                ldr r1, =(ADC1_BASE+ADC_CR)
                ldr r2, [r1]
                bic r2, #0x80000000
                bic r2, #0x0000003f
                orr r2, #ADC_CR_ADSTART
                str r2, [r1]
loop    ldr r1, =(ADC1_BASE + ADC_CSR)
                ldr r2, [r1]
                tst r2, #ADC_CSR_EOC_MST
                bne out
                b loop
out             ldr r1, =(ADC1_BASE+ADC_DR)
                ldr r0, [r1]

                bx              lr
                ENDP
```

```
                ALIGN
                END
```

Main code:

```
;******************** (C) Andrew Wolfe
*****************************************
; @file    main_hw_proto.s
; @author  Andrew Wolfe
; @date    August 18, 2019
; @note
;         This code is for the book "Embedded Systems with ARM Cortex-M
;         Microcontrollers in Assembly Language and C, Yifeng Zhu,
;         ISBN-13: 978-0982692639, ISBN-10: 0982692633 as used at Santa Clara University
;*******************************************************************************
*



        INCLUDE core_cm4_constants.s        ; Load Constant Definitions
        INCLUDE stm32l476xx_constants.s
        INCLUDE timer.h
        INCLUDE leds.h
        INCLUDE     dac.h
        INCLUDE     adc.h
        IMPORT      sintbl
        IMPORT      sawtbl

sample_freq     equ     20000           ;20KHz sampling rate
sample_per      equ     1000000/sample_freq
test_freq       equ     600

                AREA    main, CODE, READONLY
                EXPORT      __main
                ENTRY

__mainPROC
                ldr             r0,=test_freq
                bl              calc_phaseinc       ;compute the phase increment value
(phaseinc)
                ldr             r2,=phaseinc
                str             r0,[r2]                     ;store the phase increment value in
memory

                bl              dac_init                    ;initialize dac
                bl              tim2_init                   ;initialize timer interrupt
                ldr             r0,=sample_per              ;set output rate to 20KHz
                bl              tim2_freq
```

```
                bl              adc_init

loop    bl              adc_ch6
                bl              adc_read
                ldr r1, =gain
                str r0, [r1]

                bl              adc_ch5
                bl              adc_read

                bl              calc_phaseinc
                ldr             r2, =phaseinc
                str     r0, [r2]
                b loop



endlessb                endless
                ENDP


TIM2_IRQHandler PROC
                EXPORT      TIM2_IRQHandler
                push    {lr}
                ldr             r0,=phase                       ;get a pointer to the current phase
                ldr             r1,=sintbl                      ;Get pointer to waveform table |
change to sawtbl for saw table values
                bl              get_tblval
                bl              dac_set
                ldr             r1,=phaseinc            ;load phase increment
                ldr             r0,=phase                       ;reload last phase value
                bl              update_phase
                pop             {lr}
                ldr             r2,=(TIM2_BASE+TIM_SR) ;reset pending interrupt for TIM2

                mov             r1,#~TIM_SR_UIF
                str             r1,[r2]
                dsb
                bx              lr
                ENDP

calc_phaseinc  PROC
                                ; To calculate the phaseinc, take the new frequency
(w)/sampling freq.(w0) * 1024
                                ; to avoid precision issues - we will keep phase in 16ths then
divide at the last minute
                                ; w arrives in r0; phase increment returned in r0
                                ; works from about 2Hz to sampling freq./2
```

```
                                        ; Assumes a wave table size of 1024 and a phase iterator
scaled up by 16

;Put your code here.
                LDR r1, =sample_freq
                LSL r0, #4
                LSL r0, #10
                UDIV r0, r0, r1

                bx              lr
                ENDP

update_phase  PROC
                                        ;recieves a pointer to phase in r0 and a pointer to phaseinc in
r1
                                        ;adds phaseinc to phase
;Put your code here.
                LDR r2, [r0]
                LDR r3, [r1]
                ADD r3, r2
                BIC r3, #0x4000
                STR r3, [r0]
                bx              lr
                ENDP

get_tblval              PROC
                                        ;recieves a pointer to phase in r0 and a pointer to a wave
table in r1
                                        ;Assume the wave table is 1024 entries; 16-bits each
                                        ;Assume the phase value is in 16ths.
                                        ;Return the sample in r0

;Put your code here.
                LDR r2, [r0]
                LSR r2, #4
                LSL r2, #1
                ADD r1, r2
                LDRH r0, [r1]

                LDR r8, =gain
                LDR r9, [r8]
                MUL r0, r9
                LSR r0, #12
                bx              lr
                ENDP


                ALIGN
                AREA    myData, DATA, READWRITE
```

```
                ALIGN
phase       dcd         0                       ;maintain in 16ths.
phaseinc    dcd         160
gain        dcd     0

    END
```