(A/D and D/A Converters)

(Curtis Robinson, Khondakar Mujtaba)

(ELEN 120)

(Wednesday 2:15)

Objective:

Throughout this lab, our goal is to use the sine table, potentiometers, and oscilloscope to analyze and develop a code that correctly scales a waveform through our chip. The first thing we do is configure the potentiometer to turn on the Red and Green LEDs depending on the input. We must allow the potentiometer to be red in our input registers and use conditionals to distinguish between a number that enables or turns off the LEDs. After the later portions of the project, we must develop code that correctly scales our given sine wave from the sine table and outputs it to a pin that can be read by the oscilloscope. In the last part, we will change the sample rate of the saw wave and allow a timer to function in relation to the sine table and phase increment.

Problem 1

Include a circuit diagram of these potentiometers in your report.

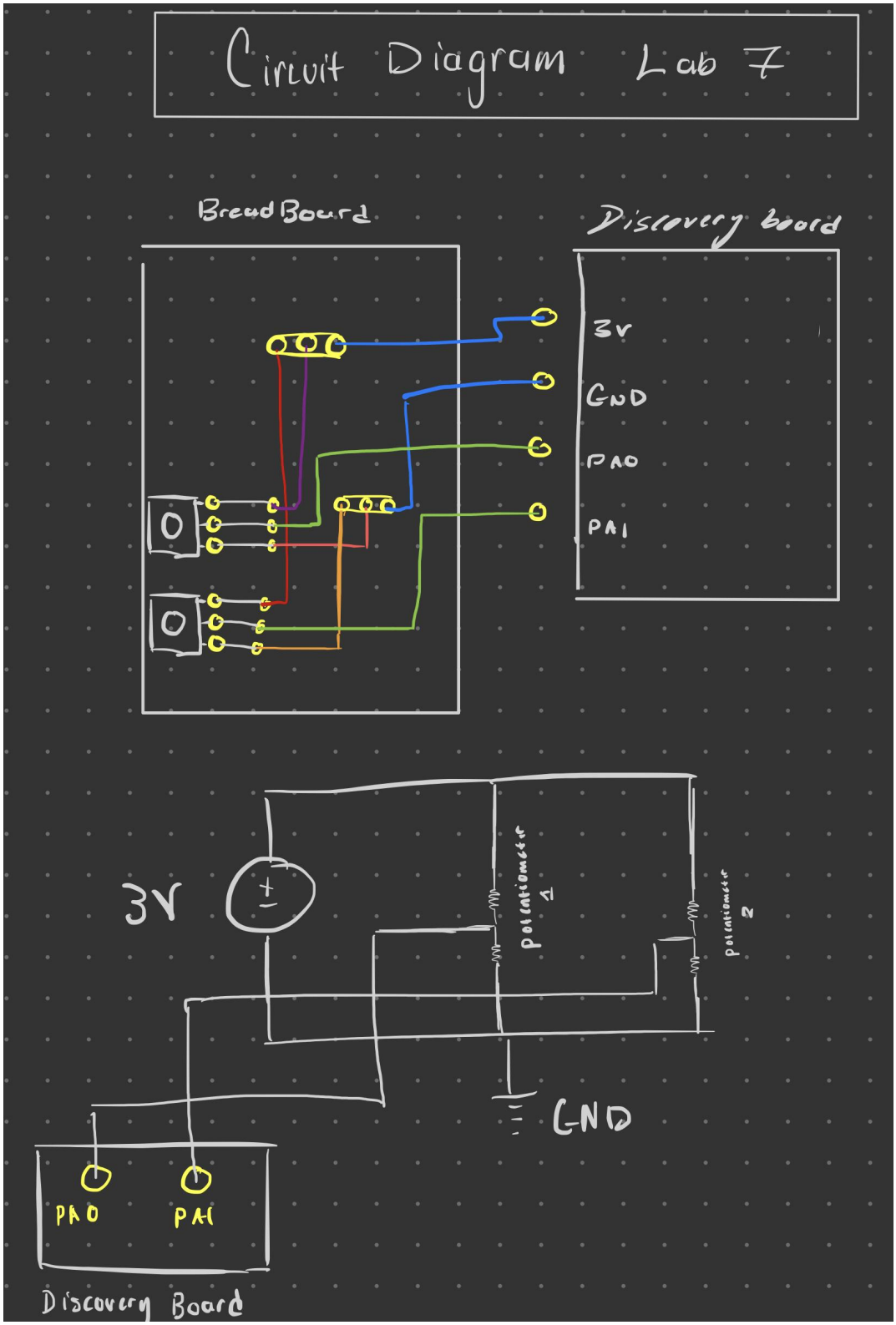
Adc read code:

```
108
109 adc_read    PROC        ;return ADC channel 6 value in r0
110                 EXPORT  adc_read
111 ;  This code appears to have been eaten by an internet worm.  Guess you need to rewrite it.
112         LDR r0,=ADC1_BASE
113         LDR r1,=(ADC1_BASE+ADC_CR); read ADC_CR
114         LDR r2,=ADC_CR_ADSTART; load ADC_CR_ADSTART
115         LDR r3,[r1];
116         BIC r3,#(0x1F); clear bits 0-5 and 31
117         BIC r3,#(0x01<<30);
118         ORR r3,r3,r2; set bit 2 and write back
119         STR r3,[r1];
120         LDR r3,=(ADC1_BASE+ADC_CSR)
121 worm    LDR r2, [r3,#ADC_CSR_EOC_MST]; check if ADC is clear or not
122         TST r2,#0x01
123         BEQ read;
124         b   worm
125
126 read    LDR r0,[r0, #ADC_DR]; read ADC_DR
127         dsb
128         bx      lr
129             ENDP
130             ALIGN
131             END
132
```

Main code:

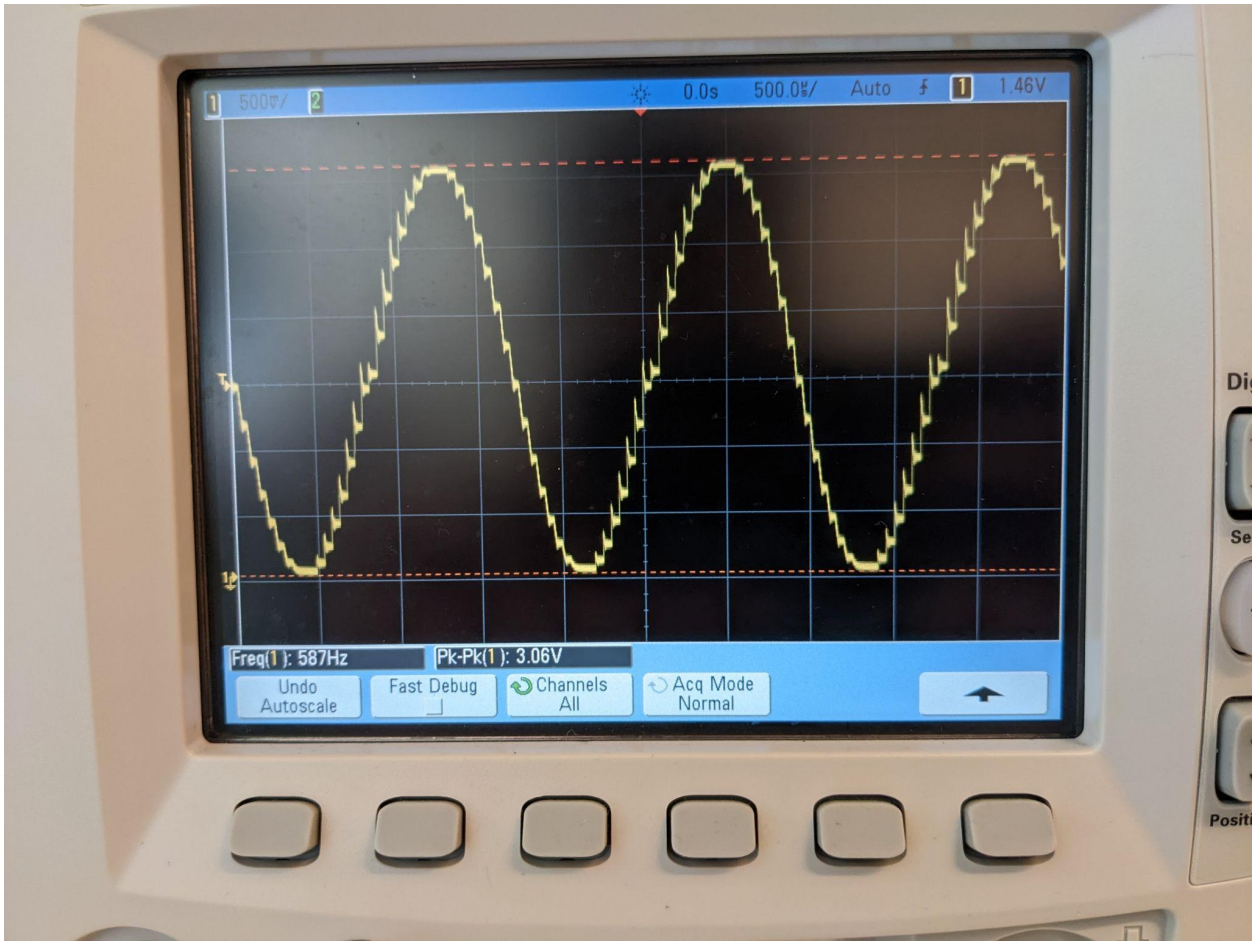
```
20
21              AREA    main, CODE, READONLY
22              EXPORT  __main
23              ENTRY
24
25  __main  PROC
26
27
28  ; Add code here to configure the proper GPIO port to drive the red LED.
29  ; You may use the routines provided in leds.s
30
31          LDR r0,=RCC_AHB2ENR_GPIOBEN; set clock in port B
32          bl      portclock_en
33
34          LDR r0,=GPIOB_BASE; initialize red LED
35          LDR r1,=GPIO_MODER_MODER2_0
36          bl      port_bit_pushpull;
37
38          bl      adc_init; initialize adc register
39  loop
40          bl      adc_read; read ADC
41
42          CMP r0,#2048;
43          BGE o_n
44          bl red_off; LED off
45          b   done;
46
47  o_n     bl  red_on; LED on
48          b   done;
49
50
51  done    b   loop
52
53  ; Add and/or modify code here to repeatedly read the A/D converter and turn the red LED on if
54  ; the reading is greater than or equal to 2048 and turn off the red LED is the reading is less than that.
55
56
57
58  endless b       endless
59          ENDP
60
61
```

Circuit Diagram:



Problem 2:

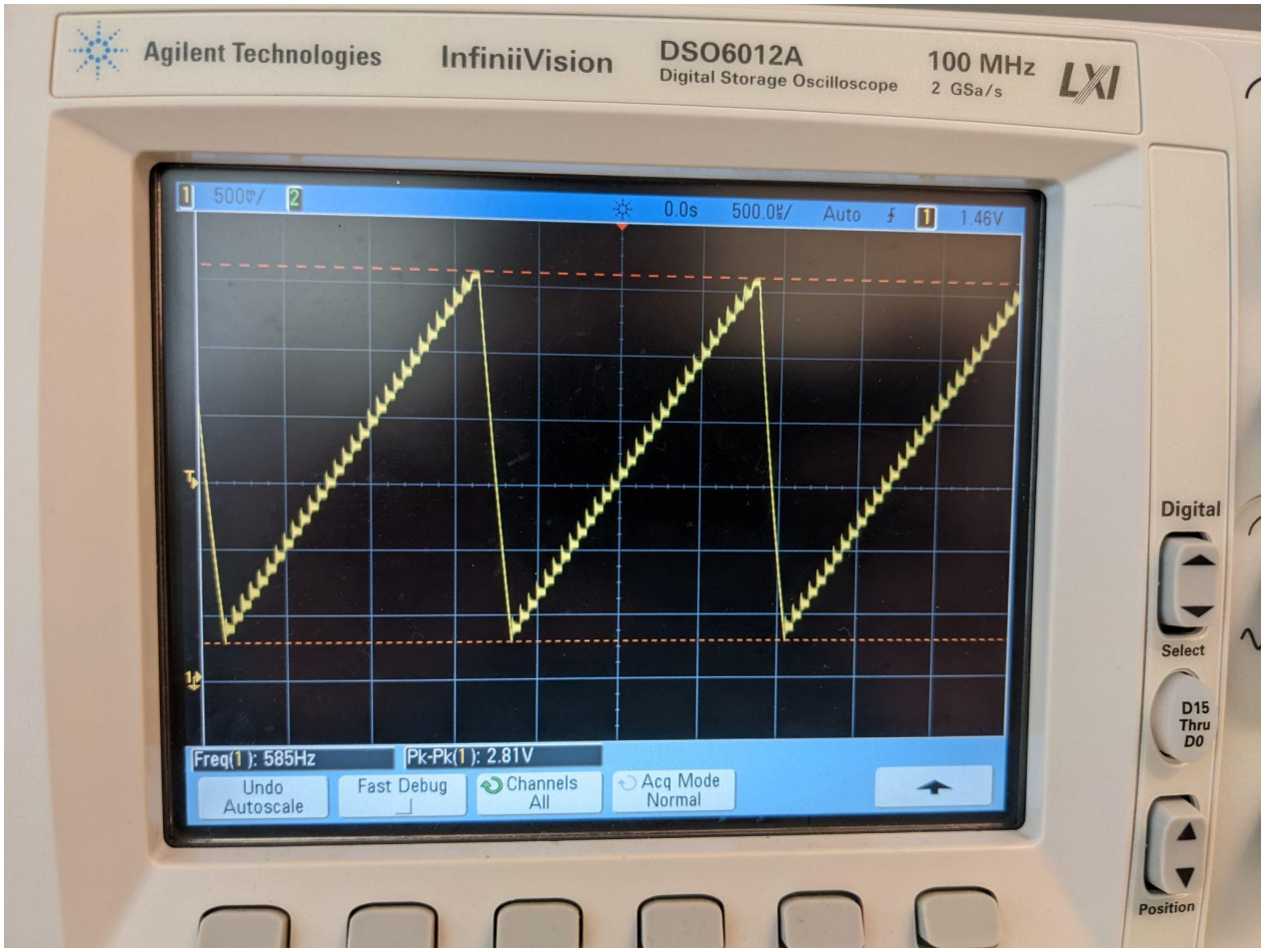
Demo for the TA and take a scope picture for your report. Turn in your code.



```
64  calc_phaseinc   PROC
65                      ; To calculate the phaseinc, take the new frequency (w)/sampling freq.(w0) * 1024
66                      ; to avoid precision issues - we will keep phase in 16ths then divide at the last minute
67                      ; w arrives in r0; phase increment returned in r0
68                      ; works from about 2Hz to sampling freq./2
69                      ; Assumes a wave table size of 1024 and a phase iterator scaled up by 16
70
71  ;Put your code here.
72          LDR r3,=phaseinc
73          MOV r2,#16
74          MUL r0,r0,r2;
75          LDR r1,=sample_freq;
76          MOV r2,#1024
77          MUL r0,r0,r2;
78          UDIV r0,r0,r1;
79          STR r0, [r3];
80
81
82          bx      lr
83          ENDP
84
85  update_phase    PROC
86                      ;recieves a pointer to phase in r0 and a pointer to phaseinc in r1
87                      ;adds phaseinc to phase
88  ;Put your code here.
89          MOV r2,r0;
90          LDR r0,[r0];
91          LDR r1,[r1];
92          ADD r0,r0,r1;
93          MOV r3,#(0x400<<4);
94          CMP r0,r3;
95          BLT inc
96          AND r0,r0,#0x0F;
97
98  inc
99          STR r0,[r2];
100         bx      lr
101         ENDP
102
103 get_tblval     PROC
104                     ;recieves a pointer to phase in r0 and a pointer to a wave table in r1
105                     ;Assume the wave table is 1024 entries; 16-bits each
106                     ;Assume the phase value is in 16ths.
107                     ;Return the sample in r0
108
109 ;Put your code here.
110         LDR r0,[r0];
111         MOV r2,#16;
112         UDIV r0,r0,r2;
113         MOV r2,#2;
114         MUL r0,r0,r2;
115         ADD r0,r1,r0;
116         LDRH r0,[r0];
117         bx      lr
118         ENDP
119
```

Problem 3:

Change sintbl in TIM2_IRQHandler to sawtbl. Determine what changed in your program.



The saw table changed the reference table to have values increase over each increment in a straight line instead of increasing and decreasing like a sine wave. This LED to a rising edge signal.

Problem 4:

New Main code:

```
__main  PROC

        bl      adc_init; initialize adc register

        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

loop    bl      adc_read; read ADC
        LDR     r1,=gain;
        STR     r0,[r1];

        b loop
;endless    b       endless
        ENDP
```

New TIM2_IRQHandler code:

```

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
        bl      get_tblval
        LDR     r2,=gain;
        LDR     r2,[r2];
        MUL     r0,r0,r2;
        MOV     r3,#4096;
        SUB     r3,r3,#0x01
        UDIV    r0,r0,r3;
        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
```