6.115 Laboratory 3: Digital Waveform Synthesis

David Ologan

June 9, 2022

1 Exercise 1: Further explore MINMON

- Add a "T" (table write) command to MINMON. At the MINMON prompt, you should be able to type a command that loads a hex byte into a specified location in memory. MINMON should continue to automatically prompt you for data for the next address byte in memory, until 256 bytes have been entered, or the user hits return without entering any data. The hex bytes should be printed on the PC screen.

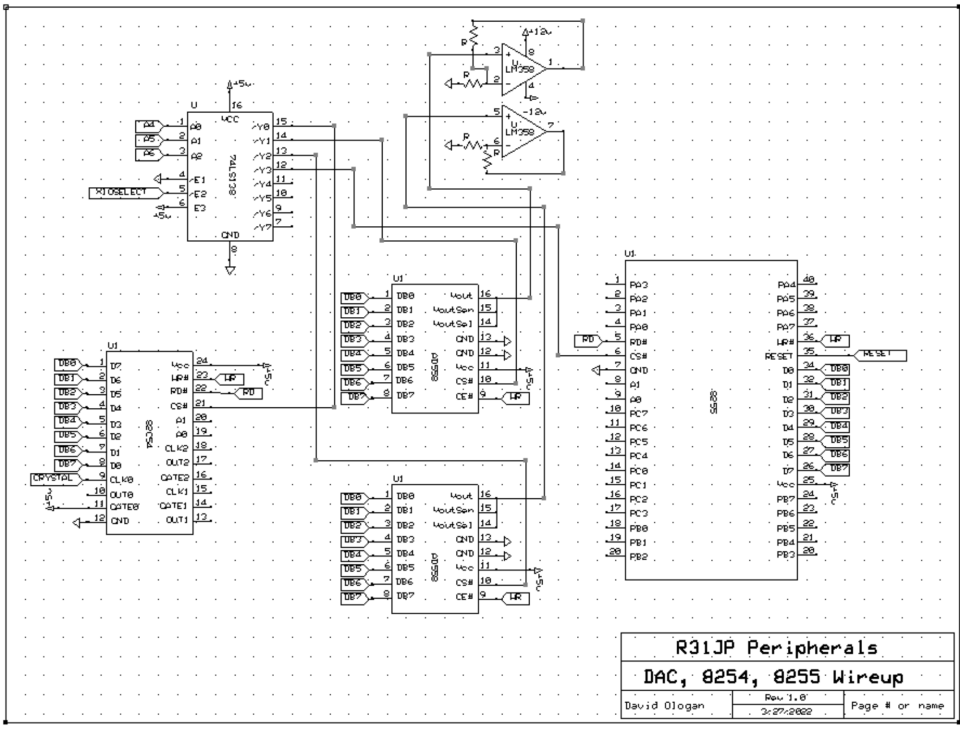
Listing 1: Table MINMON Command

```
table:
   lcall getbyt         ; get address high byte
   mov dph, a           ; get the first half
   lcall prthex         ; print the first half
   mov a, 00h           ; second half is always zero
   mov dpl, a           ; move 00 into the low byte
   lcall crlf           ; get a newline
   mov R0, #0FFh        ; Set the 256 byte counter in R0
   rdata:
        mov a, dph           ; Move high byte into acc
        lcall prthex         ; print high byte to serial
        mov a, dpl           ; Move low byte into acc
        lcall prthex         ; Print low byte
        mov a, #20h          ; Move hex code for a space in ascii
        lcall sndchr         ; Add Space between the address and the written byte
        lcall getbyt         ; get the given byte
        lcall prthex         ; Print the given byte to serial
        mov R1, a
        movx @dptr, a        ; Write that byte to the location given byte dptr
        lcall crlf           ; Get a newline
        inc dpl              ; increment the low byte and start over
        DJNZ R0, rdata       ; Decrement R0, that counts 256 entered bytes
   ljmp endloop              ; End loop at 256 entered bytes
```

2 Exercise 2: Add More Useful Peripherals to your Lab Kit

- Connect your two DAC's so that they are available through memory-mapped I/O addresses. Configure your two DAC's for a 0 to 2.5 volt output range. Do not remove the 8254 or 74C922 from your kit. Instead, use a 74138-type logic part that will use the XIOSELECT line and the address lines cleverly so that you can chip select 8 different I/O peripheral chips in the memory-mapped I/O space. Make sure and keep the 8254 and each new DAC wired up to different addresses in this space. This should leave 5 other select lines for future expansion. Recall that 8000 series peripherals sometimes use more than one address. Leave eight custom addresses available for each of the 8 peripherals. For example, the addresses FE00h to FE07h could all result in chip selection of the same 8254. The 6.115 staff solved this problem using a 74138 chip (see our Lecture notes and the 8051 board schematics you reviewed in Exercise 8 of Lab 2 for

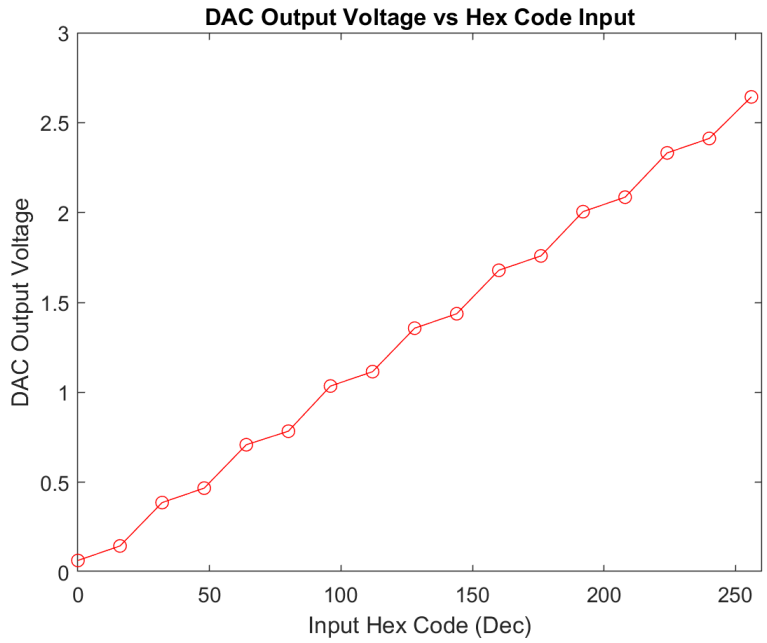
hints on how to use the 74138, but also check the 74138 data sheet carefully), address lines A4, A5, and A6, and the XIOSELECT line. Include a detailed circuit schematic in your lab write-up.



- Test your 8254 to make sure that it still works. Run the 8254 in square wave mode, and make a few different square waves at different frequencies. You should not need to write a program to do this. Just use your MINMON "W" command to exercise the chip and make sure it works correctly.

- Test your new DACs. Again, use MINMON. Protect the output of each DAC with a non-inverting OP-AMP gain block. Let's call this the "scaling" OP-AMP. You can use plus and minus 12-volt power rails FOR THE OP-AMPs ONLY. Get the plus and minus 12 from the fixed power supplies on the side of your kit (near the power switch)! The +12 should already be available on your kit breadboard near the R31JP ribbon cable connector – don't mess with the +12 banana jack on the side of the kit. Do run a wire from the -12 banana jack on the side of the kit to the breadboard, neatly. Do not use the kit variable supplies for this. Do not get the plus or minus 12 volts anywhere near your digital logic, and do NOT damage your R31JP or other peripheral chips. Set the gain so that each DAC and OP-AMP combination can produce an output voltage between 0 and 5 volts. Use the OPAMP output to drive an LED in series with an appropriate resistor (Do NOT use one of the KIT LEDs. Wire up your own LED, and calculate a series resistance value to limit the current through the LED to 10 mA at a peak of 5 volts across the LED/resistor combination.). Use MINMON to write commands to each DAC in order to set the voltage on the LED/resistor series circuit. You should be able to adjust the brightness of the light using MINMON commands. Do NOT drive the LED directly with the DAC, you will burn out the DAC – the DAC spec sheet should convince you that it cannot provide enough current to reliably drive the LED.

- Check the DAC output with an oscilloscope, and make a graph of voltage output values for 16 different digital command bytes ranging between 00h and FFh

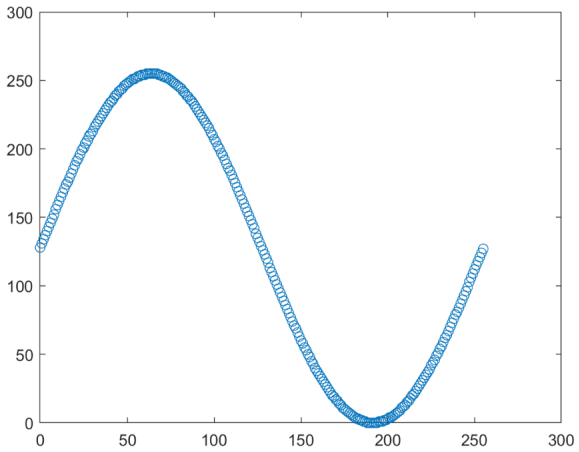


- Use another select line from your new 74138 addition to also add an 8255 parallel port IC to your R31JP/Kit. Don't forget about the reset line on the 8255 that we discussed in lecture. Use 8255 "Mode 0" (see the datasheet) to configure the 8255 as three "output" ports, and test your system with MINMON. You should be able to write values to the 8255 pins and check the 8255 outputs with your oscilloscope.

3 Exercise 3: Make a Sine Wave Generator

- Make a hexadecimal sine wave table that contains 256 amplitude values of a sine wave distributed sequentially over a single sine wave period. You may use the software of your choice to do this - we recommend OCTAVE/MATLAB or EXCEL or PYTHON or a C program. (OCTAVE, listed under the r-Octave menu item, is available at web.mit.edu/cdev.) If you are clever with your program and a text editor, you will be able to cut and paste your table into an assembly program without actually retyping the 256 entries. Think carefully about the scaling and offset of your sine table. We will want to use this table to drive the DAC. Recall that your DAC expects digital values between 00h and FFh, and, with the scaling OP-AMP, will produce an analog output that should be between 0 and 5 volts. Hence, your analog output from the DAC will have an offset, which will be fine for our purposes, but could be removed if we had bothered to add the Offset OP-AMP. The sine wave table entries should vary between 00h and FFh. The table should be a circular buffer; in other words, do not repeat the first entry of the table in the last entry. You should be able to make a continuous sine waveform by reading through the table entries, and, at the end, return to the beginning of the table to repeat.

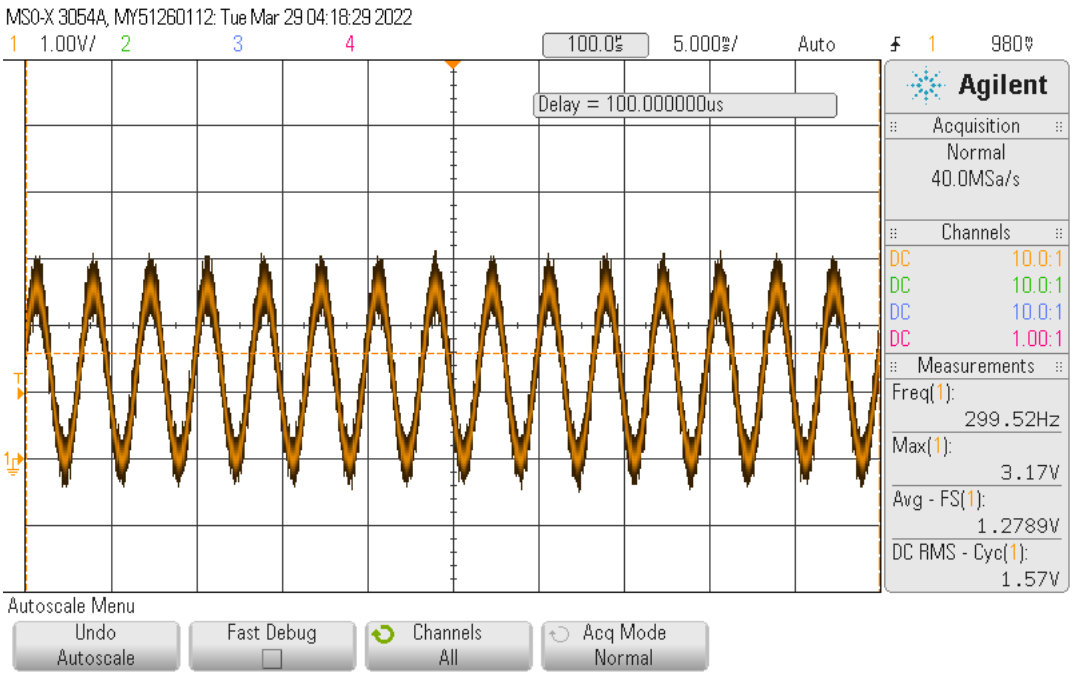
```
x = linspace(0,2*pi, 256);
t = 0:1:255;
y = zeros(1,256);
for i = 1:length(x)
    y(i)= sin(x(i));
    y(i)= y(i)*255/2;
    y(i) = y(i)+255/2;
end
y=round(y);
plot(t,y, 'o')
title('Scaled Hex Sine Wave')
```



- Write an assembly language program that turns your R31JP board into a sine wave generator. Use your table from above and the db command to provide the sine sample values for both of your DAC/OP-AMP circuits. Have your program loop to make an analog sine wave that repeats indefinitely. Make sure that there are no glitches in the sine wave, e.g., your code flow should create no extraneous "jumps" in the output waveform as you cross from end to beginning in the sine wave table.

Listing 2: Sine Wave Generator

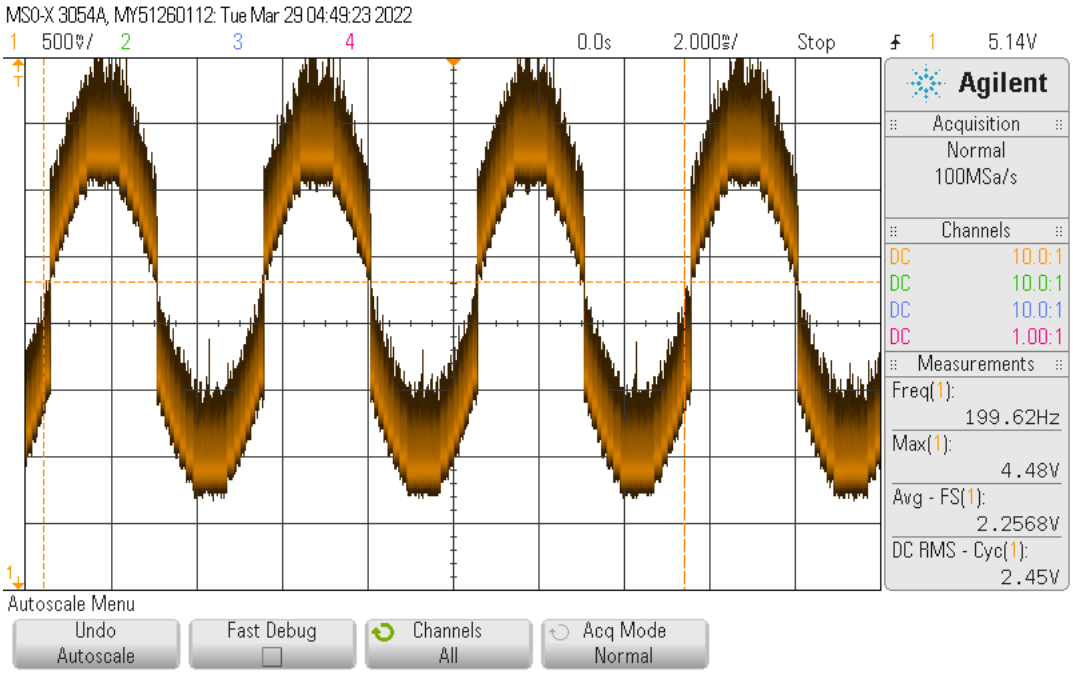
```
1  .org 0000h
2
3  main:
4      mov R0, #0FFh          ; Initialize Counter for 256 bytes
5      mov R1, #00h           ; Initialize Counter to Keep track of position in data table
6      start:
7          mov a, R1          ; Load Position in data table into acc
8          mov dptr, #sine    ; Move address of sine table into data pointer
9          movc A, @A+dptr    ; Get byte (sine value) at that location
10         mov dptr, #0FE20h  ; Move DAC address into dptr
11         movx @dptr, a      ; Move the sine value into the DAC
12         inc R1             ; Increment position counter
13     sjmp start             ; Repeat
14
15 .org 0500h
16 sine:
17     .db
           128,131,134,137,140,143,146,149,152,156,159,162,165,168,171,174,176,179,182,185,188,191,193,196,199,201,
```



- Carefully analyze the timing (number of machine cycles) of each instruction in your sine-wave program. Explain your observed sine-wave frequency in the laboratory in terms of your timing analysis.

Each instruction in my sine-wave program takes 2 machine cycles except for the inc R0, for a total of 17 machine cycles to get through the program. Given that the observed frequency is about 300 Hz, $(11.0592 * 10^6)/(256 * 12 * 17) = 211Hz$

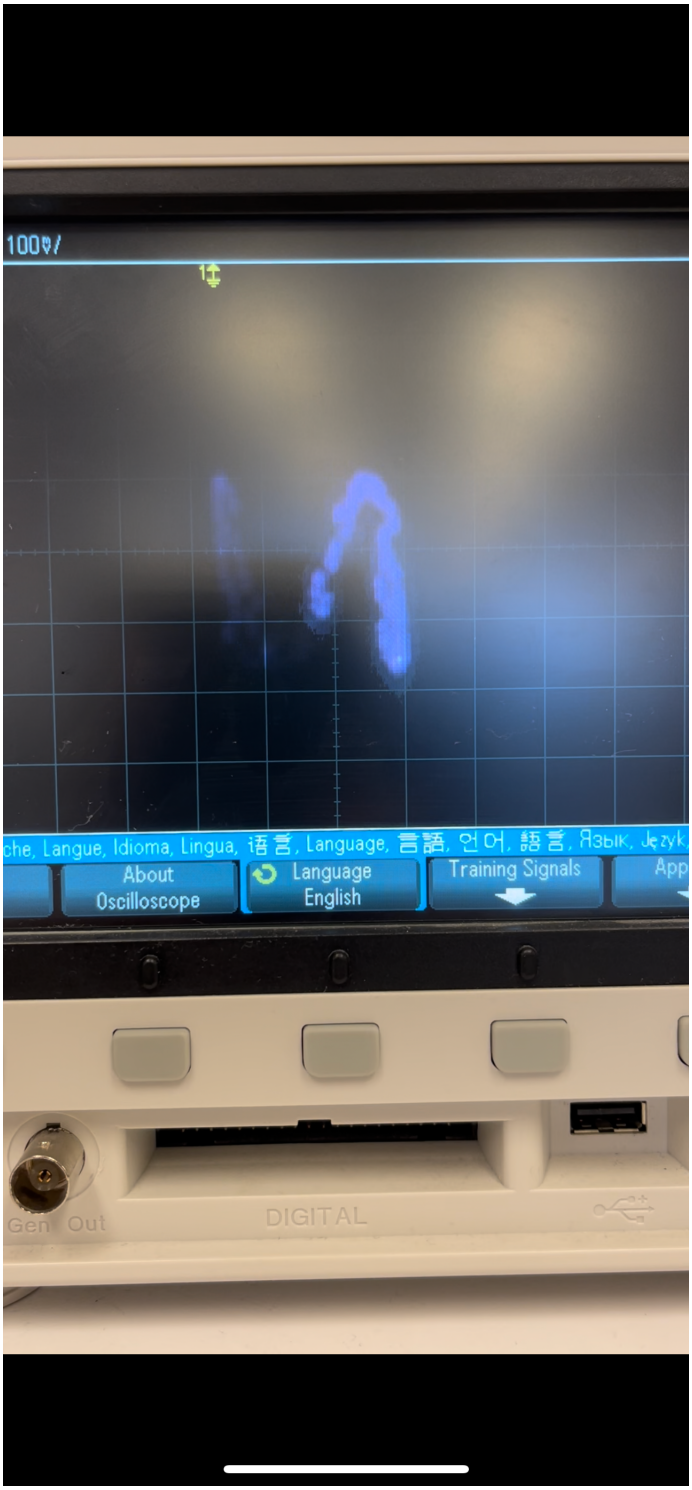
- Modify your program so that it divides each sample entry by 10 before writing the sample to the DAC. Then, modify your scaling OP-AMP to restore this attenuation, i.e., change the OP-AMP gain to multiply the signal by a gain of 10, so that the peak amplitude of the waveform is about the size of the peak, unsaturated sine wave you made in the previous step. What does this new wave form look like on the scope? Carefully (with detailed timing analysis) explain any changes in frequency, wave shape, and amplitude from the previous step.



- You should be able to use your MINMON "T" or "V" command to overwrite the sine wave table "on the fly" (without using AS31 to re-assemble your signal generator program) to create an "arbitrary"waveform generator that repeats any shape you enter. Try it! For example, you could overwrite the first 10 or 20 entries of your sine wave table (in a known location in memory) to create a "flat spot" in the waveform. Include scope photos of a change like this in your lab report.

4 Exercise 4: An "Application-Specific" Signal Generator: Lissajous Curves

- Set your two scaling OP-AMP gains so that each offers a gain of 2, i.e., so that your DAC signal generators can make signals between 0 and 5 volts using the "full range" 00h-FFh of DAC inputs. Implement an R31JP program that makes the oscilloscope draw a TIE fighter without rotation. Use 16-bit counters to step through and keep place in the sine table.

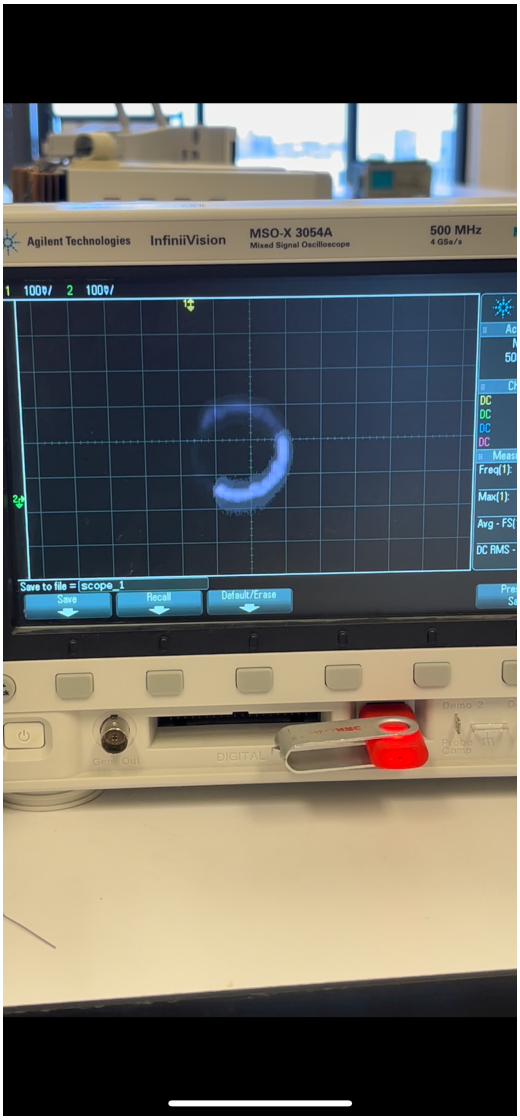


Listing 3: TIE Fighter

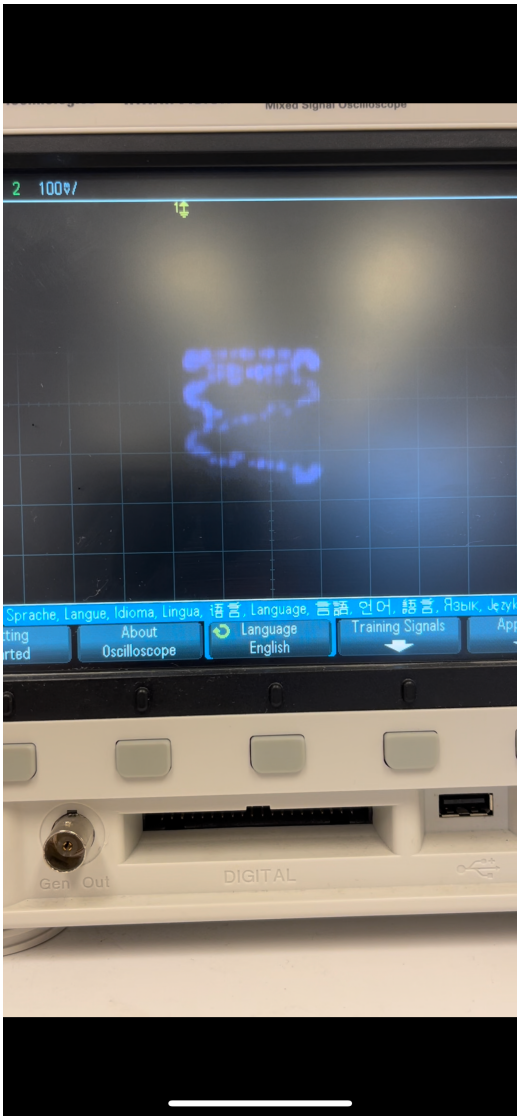
```
1  .org 00h
2  ljmp main
3
4  .org 00Bh
5
6  T0ISR:            ; Interrupt Service Routine
7      lcall draw
8      reti
```

```
9
10  .org 0030h
11
12  main:
13      mov TMOD, #02h     ; Set Mode, Initialize the Interrupt
14      mov TH0, #4Ch      ; Set Count
15      setb TR0
16      mov IE, #82h
17      mov R4, #00h       ; DAC 1 Current Position High Byte
18      mov R5, #00h       ; DAC 1 Current Position Low Byte
19      mov R6, #0C0h       ; DAC 2 Current Position High Byte
20      mov R7, #00h       ; DAC 2 Current Position Low Byte
21      loop:
22          sjmp loop      ; Loop Back
23
24
25  .org 0500h
26  sine:
27      .db
            128,131,134,137,140,143,146,149,152,156,159,162,165,168,171,174,176,179,182,185,188,191,193,196,199,201,
28
29  draw:
30      mov R0, #03h       ; DAC 1 Counter High Byte
31      mov R1, #00h       ; DAC 1 Counter Low Byte
32
33      mov R2, #01h       ; DAC 2 Counter High Byte
34      mov R3, #00h       ; DAC 2 Counter Low Byte
35
36      clr C              ; DAC 1 Calculation
37      mov a, R5          ; Move low byte current position into Acc
38      add a, R1          ; Add low byte first
39      mov R5, a          ; Update with the result of the low byte addition into the low byte
            counter
40      mov a, R4          ; Move the high byte current position into acc
41      addc a, R0         ; Add to High
42      mov R4, a          ; Update with the result of the high byte addition into the high byte
            counter
43
44      clr C              ; DAC 2 Calculation
45      mov a, R7          ; Move low byte Current Position into Acc
46      add a, R3          ; Add low byte firstlow
47      mov R7, a          ; Update with the result of the low byte addition into the low byte
            counter
48      mov a, R6          ; Move high byte current position into acc
49      addc a, R2         ; Add to High
50      mov R6, a          ; Update with the result of the high byte addition into the high byte
            counter
51
52                         ; Loading DAC 1
53      mov a, R4          ; Obtain integer of high byte position for DAC 1
54      mov dptr, #sine    ; Move address of data table into dptr
55      movc A, @A+dptr    ; Load acc with the correct position in the sine table
56      mov dptr, #0FE10h  ; Load dptr with the DAC location
57      movx @dptr, a      ; Load sine value into DAC 1
58
59                         ; Loading DAC 2
60      mov a, R6          ; Obtain integer of high byte position for DAC 2
61      mov dptr, #sine    ; Move address of data table into dptr
62      movc A, @A+dptr    ; Loadd acc with the correct position in the sine table
63      mov dptr, #0FE20h  ; Load dptr with the DAC location
64      movx @dptr, a      ; Load sine value into DAC 2
65      ret                ; Return to ISR
```

- Now create a program that draws a circle, without rotation.



- Create a program that draws a Lissajous curve with a=1, b=4, phi=0 (i.e. no rotation).



- See if the oscilloscope lets you experience the "rotation" effect. Write a program that allows the TIE fighter to rotate at different speeds. A good range for rotation is between 0 Hz and 1.5 Hz.

- Combine your programs into a program that can choose between three shapes to draw (the TIE fighter, the circle, and the Lissajous curve with a=1, b=4) all with no rotation. Use your keypad to select between the three possibilities.

Listing 4: Keypad 4 Shapes

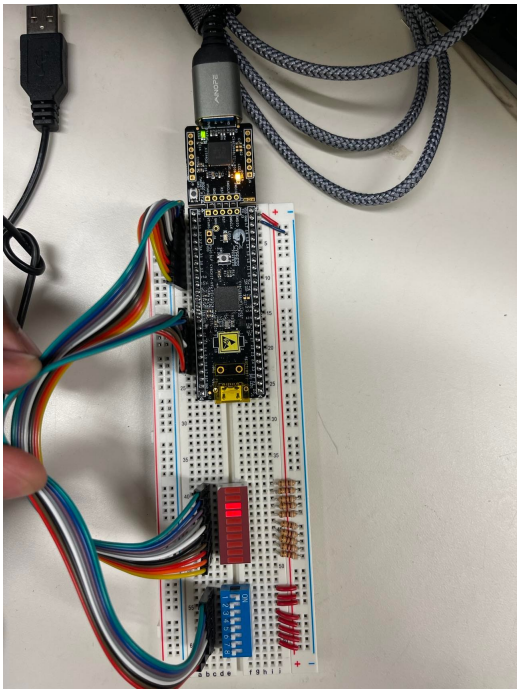
```
1  .org 00h
2  ljmp main
3
4  .org 00Bh
5
6  T0ISR:            ; Interrupt Service Routine
7      lcall draw
8      reti
9
10 .org 0030h
11
12 main:
13     mov TMOD, #02h     ; Set Mode, Initialize the Interrupt
14     mov TH0, #4Ch      ; Set Count
15     setb TR0
16     mov IE, #82h
```

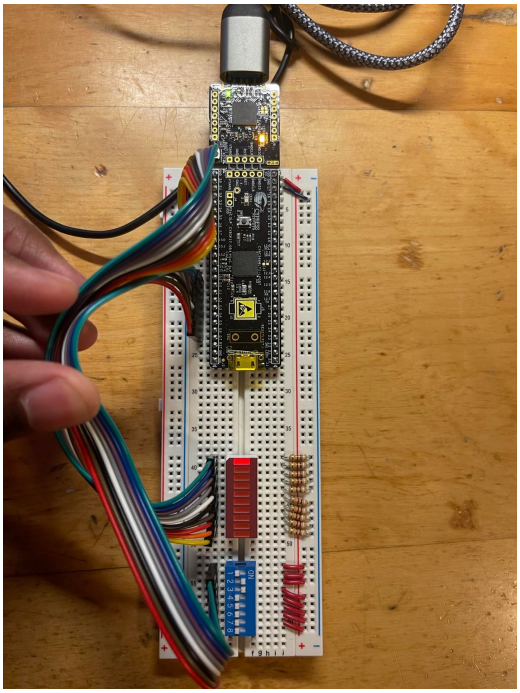
```
17      loop:
18          lcall getkey   ; Get key and update the Registers based on the selection
19          sjmp loop      ; Loop Back
20
21
22  .org 0500h
23  sine:
24      .db
            128,131,134,137,140,143,146,149,152,156,159,162,165,168,171,174,176,179,182,185,188,191,193,196,199,201,
25
26  draw:
27
28      clr C              ; DAC 1 Calculation
29      mov a, R5          ; Move low byte current position into Acc
30      add a, R1          ; Add low byte first
31      mov R5, a          ; Update with the result of the low byte addition into the low byte
            counter
32      mov a, R4          ; Move the high byte current position into acc
33      addc a, R0         ; Add to High
34      mov R4, a          ; Update with the result of the high byte addition into the high byte
            counter
35
36      clr C              ; DAC 2 Calculation
37      mov a, R7          ; Move low byte Current Position into Acc
38      add a, R3          ; Add low byte firstlow
39      mov R7, a          ; Update with the result of the low byte addition into the low byte
            counter
40      mov a, R6          ; Move high byte current position into acc
41      addc a, R2         ; Add to High
42      mov R6, a          ; Update with the result of the high byte addition into the high byte
            counter
43
44                         ; Loading DAC 1
45      mov a, R4          ; Obtain integer of high byte position for DAC 1
46      mov dptr, #sine    ; Move address of data table into dptr
47      movc A, @A+dptr    ; Load acc with the correct position in the sine table
48      mov dptr, #0FE10h  ; Load dptr with the DAC location
49      movx @dptr, a      ; Load sine value into DAC 1
50
51                         ; Loading DAC 2
52      mov a, R6          ; Obtain integer of high byte position for DAC 2
53      mov dptr, #sine    ; Move address of data table into dptr
54      movc A, @A+dptr    ; Loadd acc with the correct position in the sine table
55      mov dptr, #0FE20h  ; Load dptr with the DAC location
56      movx @dptr, a      ; Load sine value into DAC 2
57      ret                ; Return to ISR
58
59
60  getkey:
61      jnb P3.3, getkey       ; Jump if bit not set, wait for key press
62      mov P1, #0FFh          ; Set Port 1 high to be read
63      mov a , P1             ; move P1 into acc
64      clr C
65      SUBB A, #0F0h          ; Make first nibble 0s
66      mov P1, a              ;
67      clr TR0                ; Stop Clock Momentarily
68      B1: ; TIE Fighter
            CJNE a, #0Fh, B2
70          mov R0, #03h       ; DAC 1 Counter High Byte
71          mov R1, #00h       ; DAC 1 Counter Low Byte
72          mov R2, #01h       ; DAC 2 Counter High Byte
73          mov R3, #00h       ; DAC 2 Counter Low Byte
```

```
74          mov R4, #00h      ; DAC 1 Current Position High Byte
75          mov R5, #00h      ; DAC 1 Current Position Low Byte
76          mov R6, #0C0h      ; DAC 2 Current Position High Byte
77          mov R7, #00h      ; DAC 2 Current Position Low Byte
78          sjmp stoppoint
79      B2: ; Circle
80          CJNE a, #0Eh, B3
81          mov R0, #01h      ; DAC 1 Counter High Byte
82          mov R1, #00h      ; DAC 1 Counter Low Byte
83          mov R2, #01h      ; DAC 2 Counter High Byte
84          mov R3, #00h      ; DAC 2 Counter Low Byte
85          mov R4, #00h      ; DAC 1 Current Position High Byte
86          mov R5, #00h      ; DAC 1 Current Position Low Byte
87          mov R6, #40h      ; DAC 2 Current Position High Byte
88          mov R7, #00h      ; DAC 2 Current Position Low Byte
89          sjmp stoppoint
90      B3: ; Special
91          CJNE a, #0Dh, B4
92          mov R0, #01h      ; DAC 1 Counter High Byte
93          mov R1, #00h      ; DAC 1 Counter Low Byte
94          mov R2, #04h      ; DAC 2 Counter High Byte
95          mov R3, #00h      ; DAC 2 Counter Low Byte
96          mov R4, #00h      ; DAC 1 Current Position High Byte
97          mov R5, #00h      ; DAC 1 Current Position Low Byte
98          mov R6, #00h      ; DAC 2 Current Position High Byte
99          mov R7, #00h      ; DAC 2 Current Position Low Byte
100         sjmp stoppoint
101     B4: ; Rotating TIE
102         CJNE a, #0Bh, stoppoint
103         mov R0, #03h      ; DAC 1 Counter High Byte
104         mov R1, #06h      ; DAC 1 Counter Low Byte
105         mov R2, #01h      ; DAC 2 Counter High Byte
106         mov R3, #00h      ; DAC 2 Counter Low Byte
107         mov R4, #00h      ; DAC 1 Current Position High Byte
108         mov R5, #00h      ; DAC 1 Current Position Low Byte
109         mov R6, #0C0h      ; DAC 2 Current Position High Byte
110         mov R7, #00h      ; DAC 2 Current Position Low Byte
111         sjmp stoppoint
112     stoppoint:
113     pressdone:
114         jb P3.3, pressdone ; If press detected, wait for it to end
115         mov a, P1         ; Reading to Port 1
116         setb TR0          ; Restart Clock
117         ret
```

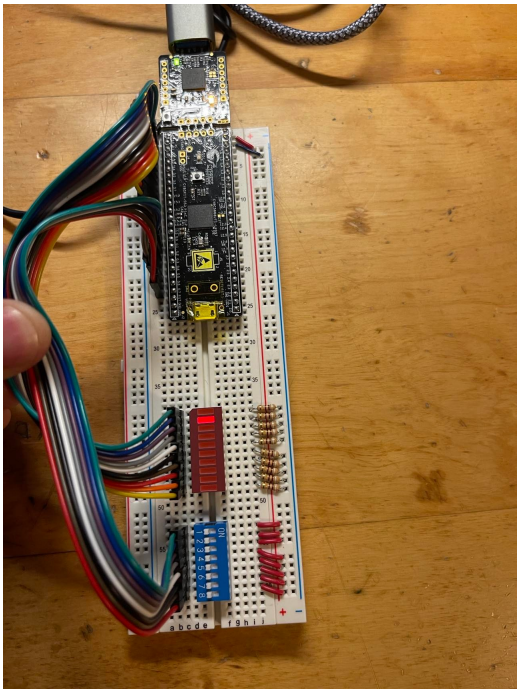
5 Exercise 5: Secrets of the Squirrel!

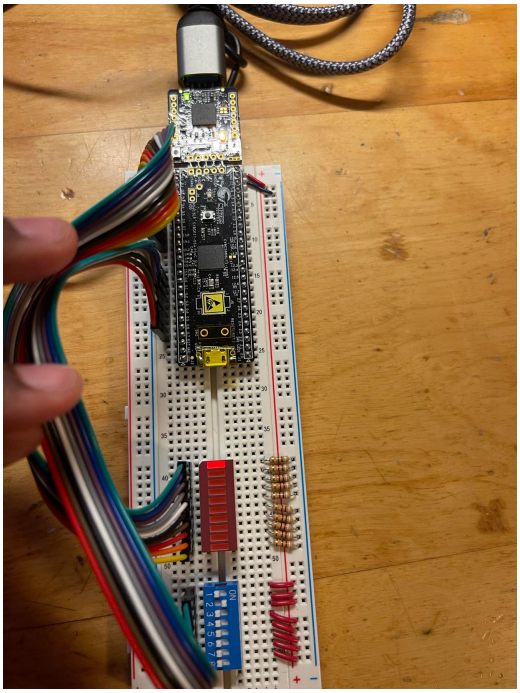
- Complete VE Lab 8 and Lab 9. Document each lab with a photograph of your Konsole board demonstrating a correct result for each lab activity.

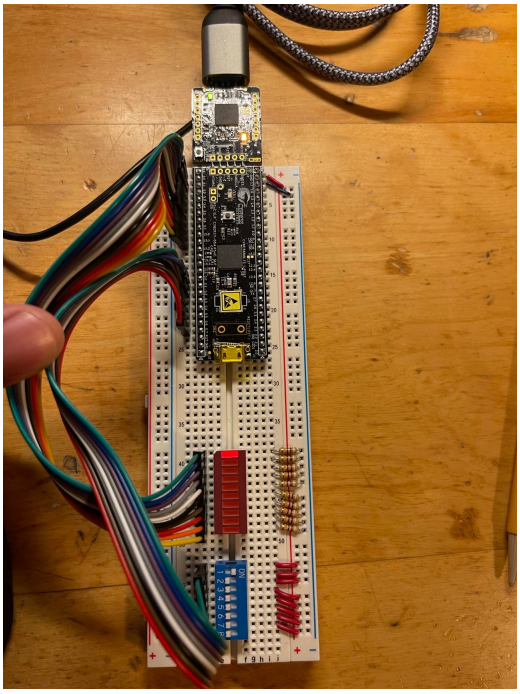


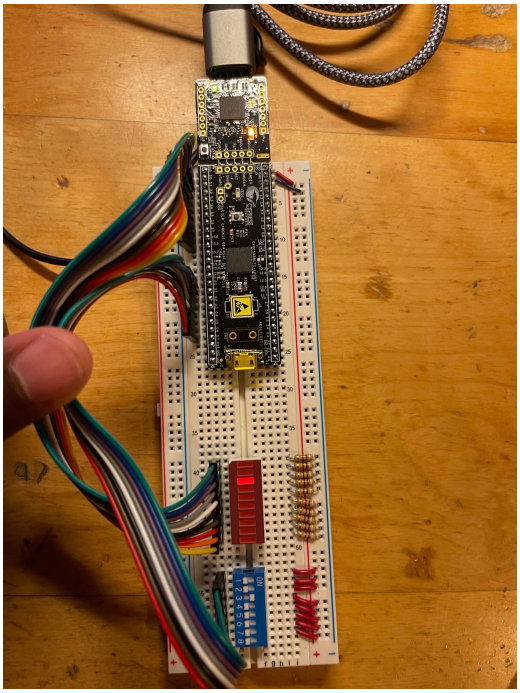


- Complete VE Lab 17, 18, 19, 20. Document each lab with a photograph of your Konsole board demonstrating a correct result for each lab activity.









- Complete VE Lab 16, 25, 26, 27 and 28. Document each lab with a photograph of your Konsole board demonstrating a correct result for each lab activity. Compress your PDF lab report to make sure that the pictures do not make the lab report larger than 2 MB.