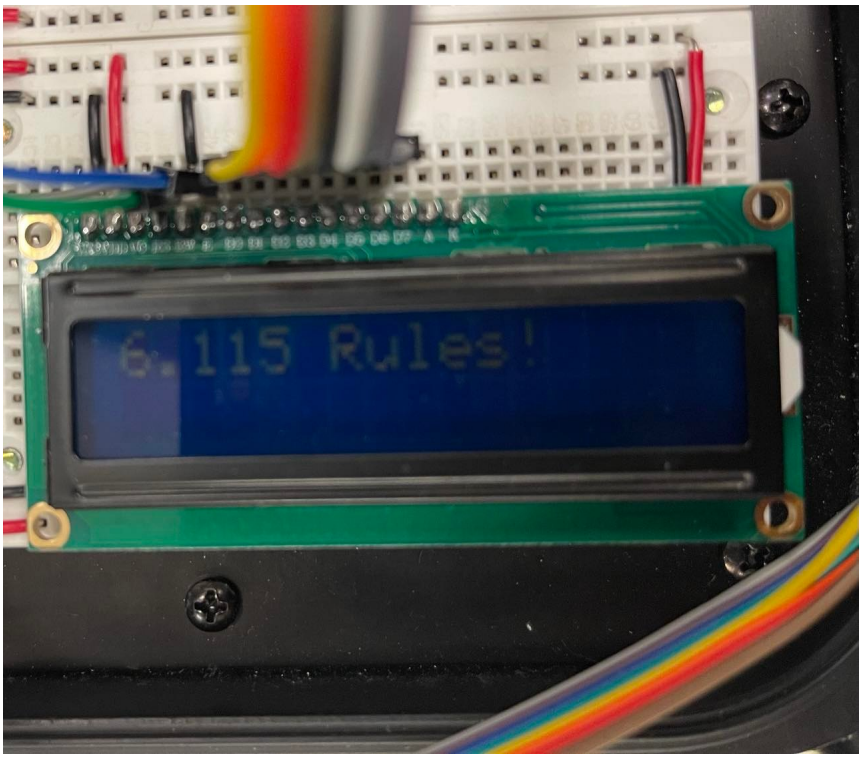
6.115 Laboratory 4: Electromechanical Actuators and Feedback Control

David Ologan

June 9, 2022

1 Exercise 1: Add More Useful Peripherals to Your Lab Kit

- Connect your LCD to the 8255 on your kit. Do not use the LCD backlight. Do make sure you adjust the LCD contrast so that you can see the letters and numbers on the screen. Test the LCD carefully, first with MINMON "W" commands, to make sure that you understand how the LCD works. Then, write a simple subroutine that you can re-use later that loads a string onto the display. You can use any approach you like. One method might be to have the subroutine load a fixed length string of character bytes from a location in memory. You could test this routine using your MINMON "V" command. Alternatively, you could have the subroutine always write a fixed string to the screen and recompile your code to change the text. Test your display with a program that writes "6.115 Rules!" to the LCD (You can be sure we'll want to see a demo of this).



2 Exercise 2: A Simple Linear Actuator: The Relay

- Determine what type of relay you have (how many contacts, normally open, normally closed etc.) by physically examining the relay.

The relay that I have has 2 contacts. The relay is normally open.

- Connect the relay to your R31JP through your 8255/ LM18293 combination as shown below: Explain why the 1N4001 is absolutely essential and cannot be eliminated (!) from this circuit. Run your LM18293 from the +V variable power supply set to 12 Volts. Write and test a simple program that turns the LM18293 on and off. You might make the program change the state of the LM18293 based on a key press, or have it toggle the LM18293 every 5 or 10 seconds. Use your multimeter (on continuity or Ohms setting) to see the contacts of the relay changing connections. Reverse the leads of the relay, (a and b), but leave all other connections the same. How does this change the behavior of the relay relative to the drive signal?

The 1N4001 is absolutely essential and cannot be eliminated from this circuit because when the relay coil is disconnected, the current needs somewhere to go. The diode provides protection from this scenario and makes sure you don't inadvertently fry your chip. Reversing the leads of the relay affects the relay but

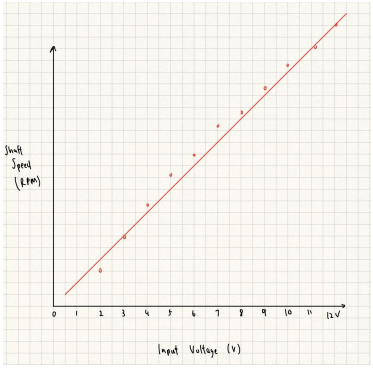
Listing 1: Relay Control

```
main:
    mov dptr, #0FE33h     ; Initialize the 8255 for 8 bit
    mov a, #80h
    movx @dptr, a
    loop:
        mov dptr, #0FE30h     ; Logic Level Low
        mov a, #00h
        movx @dptr, a
        lcall delay           ; Call Delay Function
        mov dptr, #0FE30h     ; Logic Level High
        mov a, #01h
        movx @dptr, a
        lcall delay           ; Call Delay Function
        sjmp loop             ; Repeat

delay:
    mov R0, #0FFh             ; Set Registers with Counters
    mov R1, #0FFh
    sups:
        DJNZ R0, sups         ; Nested DJNZ to Count out Time
        mov R0, #0FFh
        DJNZ R1, sups
    ret
```

3 Exercise 3: Get to know the DC Motor

- Use the "-V" variable kit supply to run the DC motor. We'll call this voltage source the "drive" voltage. Turn the voltage low, i.e., start at 0 volts. Turn the current limit all the way up (let the motor have all the current it can draw). Monitor the AC generator with an oscilloscope, and use the frequency readout measurement feature of the oscilloscope to determine the rotation speed of the shaft. Remember, the AC generator makes 8 sine wave periods for every one rotation of the mechanical shaft. You will need to convert to determine the actual shaft speed from the scope measurement. Gradually increase the motor voltage to 12 volts in steps of one volt, that is, run the motor at 0 volts, then 1 volt, 2 volts, etc. At each voltage step, let the motor settle to a steady-state speed. Record the AC generator voltage. Make a plot of shaft speed (in RPM, rotations per minute) versus drive voltage. Does your plot make sense to you?



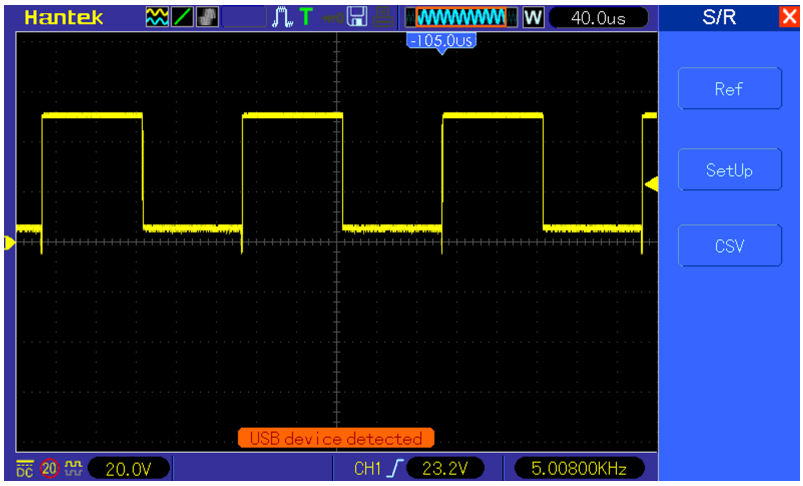
- Use your multimeter, set to measure current at a proper setting, to measure the current going into the DC motor. Continue measuring speed by observing the AC generator with your oscilloscope. With the voltage at 12 volts, gently "load" the motor with your fingers (Be careful, don't pinch yourself on the shaft or blister your finger; touch gently, or skip this step if you are not able to do so.). Watch the current display. What happens to the current flowing into the motor as you lightly increase the mechanical load on the shaft? What happens to the shaft speed?
- Think about what you saw. Looking at your graph from the first part of Exercise 3, how does the DC motor's terminal voltage affect speed? Terminal voltage on the motor appears to roughly influence what mechanical quantity? What mechanical quantity does terminal current influence?

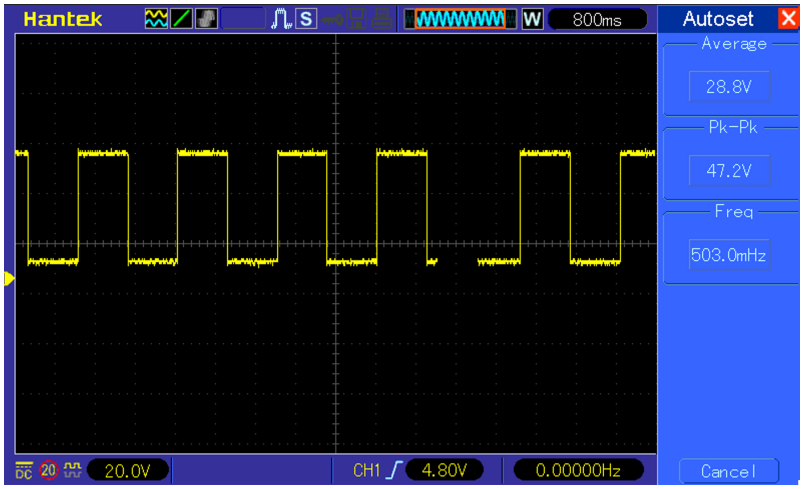
Current influences the torque of the motor while the voltage across the motor dictates its rotational velocity, w.

4 Exercise 4: Pulse Width Modulation

- Write two programs for the R31JP. One should create a PWM waveform at the output of one of your LM18293 channels with a duty cycle of 50 and a switch period of two seconds (0.5 Hz switch frequency). We'll call this the "low frequency" program. Also write a "high frequency" program that makes a PWM waveform with a duty cycle of 50 and a switch period of 0.2 milliseconds (switch frequency of 5000 Hz). Continue using the LM18293 with the "+V" variable kit supply voltage, set to 12 volts for this exercise, and apply the output voltage of your LM18293 to an LED in series with a current limiting resistor. Now, drive the LED with the low frequency program, and also with the high frequency program. What do you see in each case? Explain your observations? What is the low-pass filter in this system?

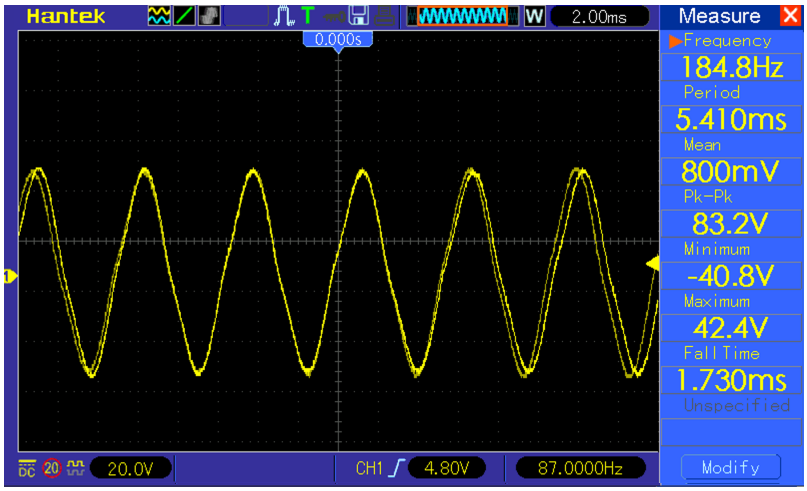
In the low frequency case, you can visually see the LED flickering on and off, but in the high frequency case, this effect is unnoticeable. The low pass filter in this system is your eyes, since your eyes can only register up to 30 Hz, and fluctuations above that frequency go unnoticed.

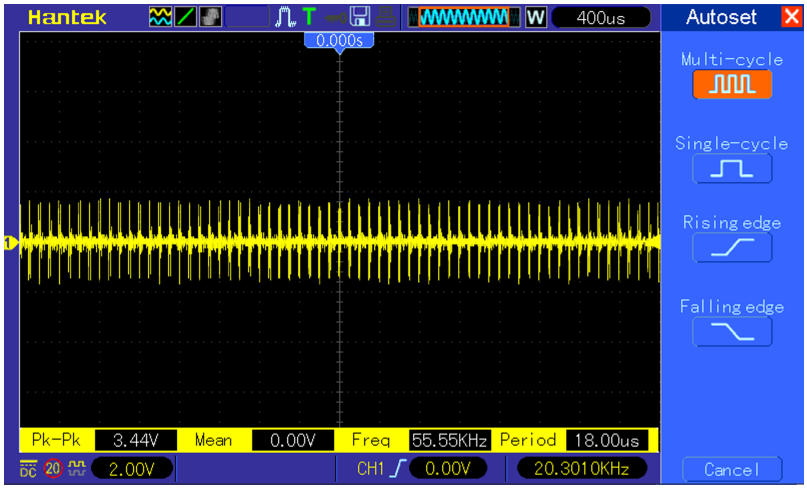




- Remove the LED/resistor combination from the output of the LM18293, and replace it with the motor/tachometer combination you used in Exercise 3. Drive the DC motor with the low and high frequency programs. Capture the AC generator/tachometer voltage waveform (speed) and drive voltage to the motor for each case. How does the motor respond to these two different drive programs? Explain your results. What provides low-pass filtering in this motor system?

At low frequency, the motor goes and then stops, so you can see it switching on and off. At high frequency the motor shaft stalls. In the motor system the low pass filtering is provided by the inertia of the shaft and friction.





- For the high frequency program, note the average tachometer or speed reading. To what drive voltage did this tachometer reading correspond to in Exercise 3 (use your graph of tach voltage versus drive voltage?) how does this drive voltage compare with the average voltage applied to your motor in this exercise.

This tachometer reading corresponded to a drive voltage of . This drive voltage is double the average voltage applied to the motor in this exercise which makes sense given our 50/50 duty cycle.

Listing 2: High Frequency

```
main:
    mov dptr, #0FE33h     ; Initialize the 8255 for 8 bit
    mov a, #80h
    movx @dptr, a
    loop:
        mov dptr, #0FE30h     ; Logic Level Low
        mov a, #00h
        movx @dptr, a
        lcall delay           ; Call Delay Function
        mov dptr, #0FE30h     ; Logic Level High
        mov a, #01h
        movx @dptr, a
        lcall delay           ; Call Delay Function
        sjmp loop             ; Repeat


delay:
    mov R0, #0FFh             ; Set Registers with Counters
    mov R1, #028h
    sups:
        DJNZ R1, sups         ; Single Delay tuned to desired high frequency
    ret                       ; return to lcall
```

Listing 3: Low Frequency

```
main:
    mov dptr, #0FE33h     ; Initialize the 8255 for 8 bit
    mov a, #80h
    movx @dptr, a
    loop:
        mov dptr, #0FE30h     ; Logic Level Low
        mov a, #00h
        movx @dptr, a
        lcall delay           ; Call Delay Function
        mov dptr, #0FE30h     ; Logic Level High
        mov a, #01h
        movx @dptr, a
        lcall delay           ; Call Delay Function
        sjmp loop             ; Repeat


delay:
    mov R0, #0FFh             ; Set Registers with Counters
    mov R1, #0FFh             ; Registers R1, R2 hold the desired repetitions
    mov R2, #07h
    sups:
        DJNZ R0, sups         ; Nested DJNZ to Count out Time
        mov R0, #0FFh         ; Longer Delay, Lower Frequency
        DJNZ R1, sups
        mov R1, #0FFh
        DJNZ R2, sups
```

```
27      ret
```

5 Exercise 5: Model the DC Motor

- In the block diagram above, determine the simplest description of the "unknown" signals indicated by the letters A, B, C, and D in terms of the DC motor circuit diagram.

A indicates the difference between $V_{lab} - V_{backemf}$, B represents the current through the motor, C represents the shaft torque and D represents $V_{backemf}$

- Determine a transfer function or equation that relates the motor shaft speed to the voltage Vlab applied to the DC motor.

$$\frac{\omega}{V_{lab}} = \frac{K}{K^2 + R_a(Js)}$$

- Determine a transfer function or equation that relates the output voltage Vout of the RC Circuit to the input voltage V applied to the circuit.

$$\frac{V_{out}}{V_{in}} = \frac{1}{1+(RC)s}$$

- Compare and Contrast the transfer relationships for the DC motor and the RC circuit.

The transfer function in the RC circuit just has a gain of 1.

6 Exercise 6: Motorized Drone Cart with PWM

- Connect each of the two motors in the robot cart to an LM18293 drive on your kit. One wire from each motor should go to an LM18293 channel. The other motor wire should go to ground. The hookup is essentially identical to the one we used for the relay, and you should use a 1N4001 flyback diode, just as we did with the relay, for each motor. With this arrangement, you will only be able to run the motors "forward," which will be fine for our purposes. By controlling the two motors independently, you should be able to move forward, turn left, and turn right. Continue to use the +V variable kit supply to power your LM18293 outputs, set for a maximum of 6 volts! Write a short R31JP program that slowly scans through the 2 channels, activating both, then one, then the other. That is, the cart should roll forward, then left, then right.

- Use your keypad to drive the cart. Your program does not have to service multiple simultaneous key presses. That is, you may assume that the user will only move one button at a time. Pressing "1" should cause the cart to move forward in a straight line. Pressing "2" should give a left turn, and pressing "3" should give a right turn. The R31JP should keep moving the cart until the key is released. Use a timer interrupt to scan or poll the control key pad at regular intervals. Drive your cart around a little, carefully. Don't overstretch the wires, and do NOT operate the cart on a tabletop or anywhere where it could fall or bump into something, someone, or a pet.

- Modify your control program to include PWM for the wheel motors. That is, the user should be able to select the duty cycle of a PWM waveform used to drive the motors selected by the keypad. This need not be too fancy. Provide at least three duty cycle choices, D = 0.5, D = 0.75, and D = 1, selected by keypad keys "4", "5", and "6." Use your LCD to display the selected duty cycle. Use a timer interrupt to implement the PWM scheme. Continue to use another timer interrupt to control the polling of the control keypad. If you select the low duty cycle, the cart should be able to go forward, left, or right slowly. If you select a D = 1, the cart should also go forward, left, or right (as selected on the keypad), but relatively quickly. Play with your program – drive the cart, attempting to repeat simple driving patterns. Mark a starting spot on your floor and try driving to a few different end points at different speeds.

- Finally, write a "canned" program that runs the cart at a duty cycle of your choice and drives the cart through a path you select that includes at least one "straight," one "left", and one "right," any order. Pressing "7" on the keypad should cause the cart to run through the pattern and then stop automatically. Run the program, marking the start and end point of the cart. Run the program several times, with the cart always starting from the same start position. How reliable is the cart in getting to the "first end point" that you found? Explain why closed-loop feedback control is generally used in industrial robot arms. (In a closed-loop control, the wheel rotations and distance traveled would be actively sensed, tracked, and corrected using some sort of position sensor or collection of sensors.) Based on your experiments, how easy is it to get the cart to repeat its behavior using our "open loop" (no sensors) approach?

Listing 4: PWM Drone Cart with Keypad

```
1  .org 0000h
2
3  main:
4      mov R0, #00h          ; Intialize the high bit of delay1
5      mov R1, #0FFh         ; Initialize the low bit of delay1
6      mov R2, #00h          ; Intialize the high bit of delay2
7      mov R3, #055h         ; Intialise the low bit of delay 2
8      lcall init
9      lcall getkey2
10     lcall getkey
11     sjmp main
12
13 init:
14     mov dptr, #0FE33h     ; Initialize the 8255 for 8 bit
15     mov a, #80h           ; Read 80h into the acc
16     movx @dptr, a         ; Move 80h into location of 8255
17     ret
18
19 forward:
20     mov R6, a             ; Move the button press value into R6
21     mov dptr, #0FE30h     ; Logic Level High
22     mov a, #03h           ; Activate the motor driver of both wheels
23     movx @dptr, a         ; Move value of acc into dptr location
24     lcall delay1          ; Call the first (on)delay
25     mov dptr, #0FE30h     ; Logic Level Low
26     mov a, #00h           ; Turn off the motor driver of both wheels
27     movx @dptr, a         ; Move acc value into dptr location
28     lcall delay2          ; Call the second (off) delay
29     mov a, R6             ; Return the button press value into acc
30     ret                   ; Return to LCALL location from stack
31
32 right:
33     mov R6, a             ; Move the button press value into R6
34     mov dptr, #0FE30h     ; Logic Level High
35     mov a, #01h           ; Activate the motor driver of the right wheel
36     movx @dptr, a         ; Move value of acc into dptr location
37     lcall delay1          ; Call the first (on) delay
38     mov dptr, #0FE30h     ; Logic Level Low
39     mov a, #00h           ; Turn off the motor driver of both wheels
40     movx @dptr, a         ; Move acc value into dptr locaiton
41     lcall delay2          ; Call the second (off) delay
42     mov a, R6             ; Return the button press value into acc
43     ret                   ; return to LCALL location
44
45 left:
46     mov R6, a             ; Move te button press value into R6
47     mov dptr, #0FE30h     ; Logic Level High
48     mov a, #02h           ; Activate the motor driver of the left wheel
```

```
49      movx @dptr, a          ; Move acc into dptr location
50      lcall delay1           ; Call the first (on) delay
51      mov dptr, #0FE30h      ; Logic Level Low
52      mov a, #00h            ; Turn off the motor driver of both wheels
53      movx @dptr, a          ; Move acc value into dptr location
54      lcall delay2           ; Call the second (off) delay
55      mov a, R6              ; Return the button press value into acc
56      ret                    ; Return to LCALL location
57
58  stops:
59      mov dptr, #0FE30h
60      mov a, #00h            ; Immediately kill the motor drivers of both wheels
61      movx @dptr, a          ; Send acc to dptr location
62      ret
63
64  ;   PWM is achieved through the relative sizing of two delays. If the on delay == the off
        delay, 50% Duty Cycle. Chagnging
65  ;   the relative ratios of the two delays allows us to manipulate the duty cycle of the motor
        drivers.
66
67  delay1:                ; Comparing with R0(high), R1(low)
68      mov R4, 0          ; Load R4, with the High Byte Timer Size
69      mov R5, 1          ; Load R5 with the Low Byte Timer Size
70      start:
71      CJNE R5, #00h, notsame0  ; Check when R5 hits 0
72      CJNE R4, #00h, notsame1  ; Check when R4 hits 0
73      ret
74      notsame0:
75          dec R5               ; Dec R5 and continue
76          sjmp start           ; Return to comparison
77      notsame1:
78          dec R4               ; Dec R4 and continue
79          mov R5, #0FFh        ; Reset R5 to its Max, if R4 hits 0, (16-bit counter)
80          sjmp start           ; Return to start
81
82
83  delay2:
84      mov R4, 2      ; Load R4 with the High Byte Timer Size
85      mov R5, 3      ; Load R5 with the Low Byte Timer Size
86      start2:
87      CJNE R5, #00h, notsame2   ; Check when R5 hits 0
88      CJNE R4, #00h, notsame3   ; Check when R4 hits 0
89      ret
90
91      notsame2:
92          dec R5               ; Dec R5 and continue
93          sjmp start2          ; Return to comparison
94      notsame3:
95          dec R4               ; Dec R4 and continue
96          mov R5, #0FFh        ; Reset R5 if R4 hits 0, (16-bit counter)
97          sjmp start2          ; Return to start
98
99  getkey:                ; Key Select for Direction
100     jnb P3.3, getkey       ; Jump if bit not set, wait for key press
101     mov P1, #0FFh          ; Set Port 1 high to be read
102     mov a, P1              ; Reading to Port 1
103     clr C
104     SUBB A, #0F0h          ; Make first nibble 0s
105     pressdone:
106         lcall choice
107         jb P3.3, pressdone ; If press detected, wait for it to end
108         lcall stops
```

```
109         ret
110
111 getkey2:                  ; Key Select for Duty Cycle
112     jnb P3.3, getkey2     ; Jump if bit not set, wait for key press
113     mov P1, #0FFh         ; Set Port 1 high to be read
114     mov a, P1             ; Reading to Port 1
115     clr C
116     SUBB A, #0F0h         ; Make first nibble 0s
117     pressdone2:
118         jb P3.3, pressdone2 ; If press detected, wait for it to end
119         lcall choice2
120         ret
121
122 choice:
123     straight:
124         CJNE A, #0Fh, turnr   ; Check Keypress, and lcall the desired direction
125         lcall forward
126         sjmp finished         ; Jump to Finish
127     turnr:
128         CJNE A, #0Eh, turnl
129         lcall right
130         sjmp finished
131     turnl:
132         CJNE A, #0Dh, D50
133         lcall left
134         sjmp finished
135     finished:
136         ret                   ; Return
137
138 choice2:
139     D50:
140         CJNE A, #0Bh, D75     ; 50/50 Duty Cycle if 4
141         mov R0, #00h
142         mov R1, #0FFh         ; Set Register Sizing for 50/50
143         mov R2, #00h
144         mov R3, #0FFh
145     D75:
146         CJNE A, #0Ah, D1      ; 75 Duty Cycle if 5
147         mov R0, #00h
148         mov R1, #0FFh         ; Set Register Sizing for 75/25
149         mov R2, #00h
150         mov R3, #055h
151     D1:
152         CJNE A, #09h, D7      ; 1 Duty Cycle if 6
153         mov R0, #0FFh
154         mov R1, #00h          ; Set Register Sizing for 1
155         mov R2, #00h
156         mov R3, #01h
157     D7:
158         CJNE A, #07h, finished2  ; Check for 7 Button Press
159         mov R7, #0FFh            ; Timer for R7
160         mov B, #02h              ; High Byte Timer
161         here:
162             lcall forward        ; Call Forward for Timer Length
163             DJNZ R7, here
164             mov R7, #0FFh
165             DJNZ B, here
166         mov R7, #0FFh
167         mov B, #02h
168         here1:
169             lcall right          ; Call Right for Timer Length
170             DJNZ R7, here1
```

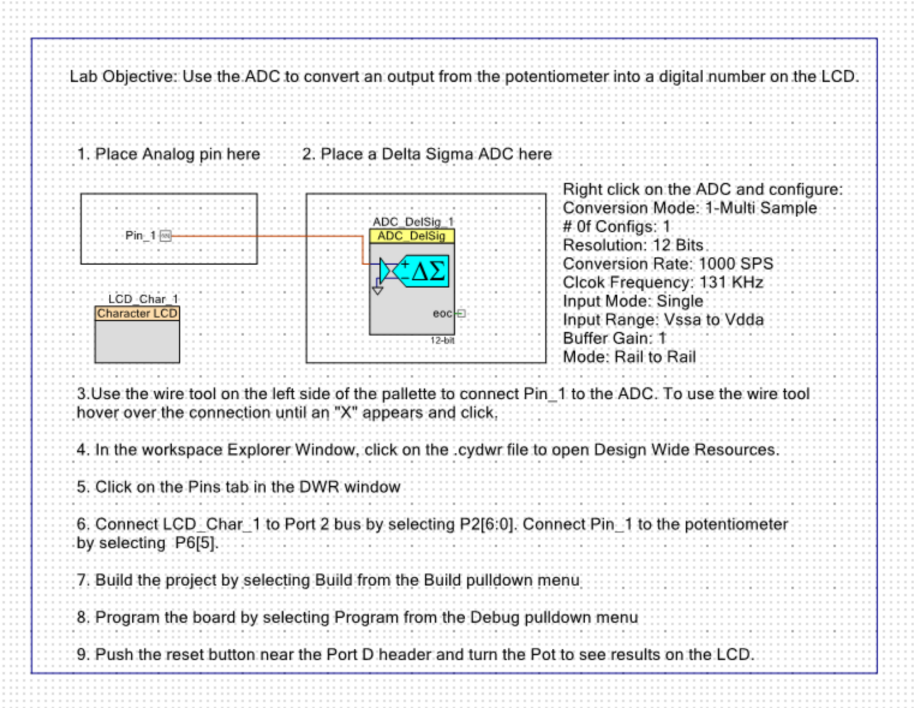
```
171             mov R7, #0FFh
172             DJNZ B, here1
173         mov R7, #0FFh
174         mov B, #02h
175         here2:
176             lcall left          ; Call Left for Length
177             DJNZ R7, here2
178             mov R7, #0FFh
179             DJNZ B, here2
180         lcall stops             ; Stop the Cart Entirely
181     finished2:
182         ret                     ; Return to Main
```

7 Exercise 7: Feedback Control for a First Order Plant

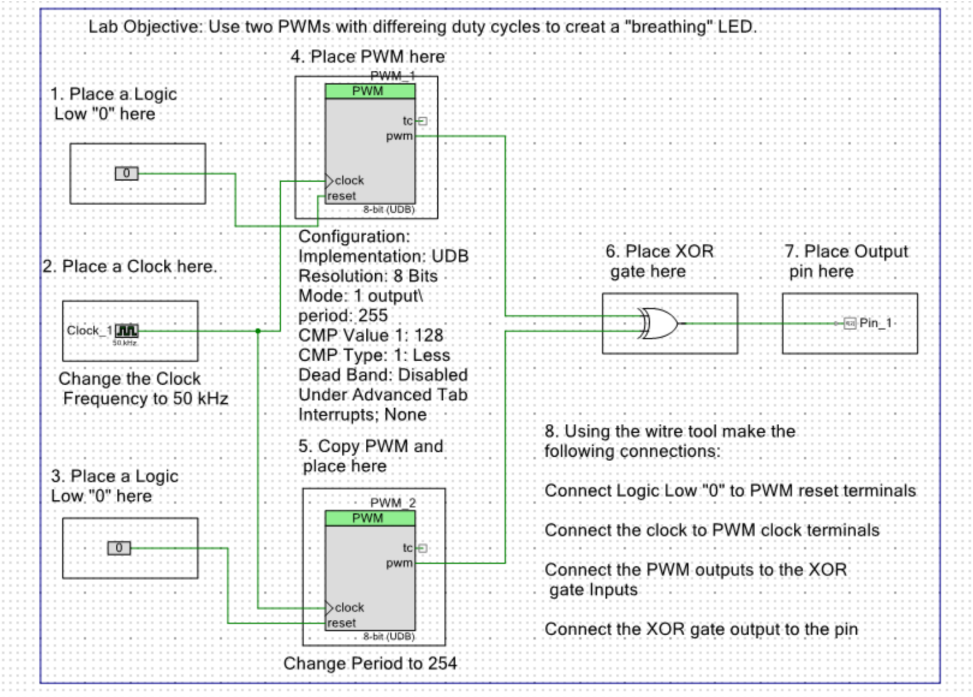
Yeah No.

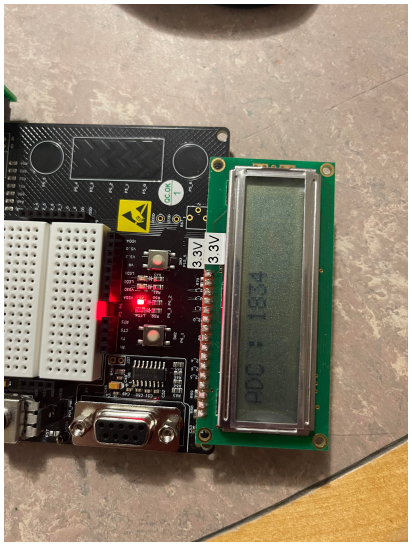
8 Exercise 8: But can I do it with PSoC?

- Now, from the 6.115 course website, download the "Exercise 2" for the PSoC (on the webpage under the "Blinky" project). Complete Exercise 2 by following the instructions on the file page with a ".cysch" extension for Exercise 2. You now know how to use the LCD for displaying text, and how to use an ADC to read an analog voltage! Include pictures and documentation of your code and TopDesign in your lab report.



- Now, from the 6.115 course website, download the "Exercise 3" for the PSoC (on the webpage under the "Blinky" project). Complete Exercise 3 by following the instructions on the file page with a ".cysch" extension for Exercise 2. Include pictures and documentation of your code and TopDesign in your lab report.





- Now, create your own program using your skills and tools you have just seen. For this program, make use of the "cool pot," the Digit LED display, the "Neato LCD" display," and the two "user buttons" on the "Big Board" that are located on either side of the "groovy LEDs". Here's what your program should do: Following a reset (using the reset button near the "PSoC IC" on the "Big Board"), your user (you in this case) should be able to turn the pot. As the pot rotates, the pot position should select one of at least 8 different letters or numbers (your choice) on the LCD display. The LCD display should show the character in a fixed location, changing the character as the pot rotates. When the user presses one of the "user buttons," let's call it "button 1," the PSoC should record the letter or number that was chosen, and display it on the Digit LED Display. The next letter chosen by the pot rotation should be shown in the "next" location on the LCD, so that a running record forms on the LCD display of the user choices. That is, the user should be able to repeat the pot-and-press process, storing several letters (you pick a maximum, at least five) shown sequentially on the LCD display as a historical record, with the most recent selection shown on the Digit LED Display. When the user presses the "other" user button, let's call it "button 2", the PSoC should stop accepting letter entry from the pot, and instead display the 14 sequence of letters that was entered, one at a time, on the Digit LED Display. Each letter should be displayed for one second, and the "message" should repeat indefinitely, cycling along on the Digit LED Display. A "reset" should start the whole process over again. (You may find looking at Exercise 1 again, under "Blinky", to be helpful....). READ the datasheets for the PSoC LCD display, and any other needed components, to learn about the API commands you will want in order to do things like manipulate the LCD display. Include pictures and documentation of your code and TopDesign in your lab report.

9 Exercise 9: Stepper Motors

- Reason out what type of stepper motor must be contained in SpinDude (unipolar-bipolar, etc.). Use a multimeter.

The Stepper Motor in Spin Dude is unipolar.

- Write a program for the R31JP that makes the stepper turn. Your program should wait to receive a serial character from Hyperterminal over the R31JP serial port. When the R31JP receives the character, it should turn the SpinDude turntable through exactly 24 steps, waiting approximately one second between steps. It should not matter what character is transmitted, any character (space bar, letter, etc.) should make the program start turning the table. A single transmitted character – one key press on the personal computer – should cause SpinDude to turn through 24 steps, completing a full circle. Make it possible to change the delay between steps. Your delay between steps should be able to range between a tenth of a second and three seconds. Watch the motor carefully, don't let it or the LM18293's get too hot. If anything seems warm,

check for mistakes. Also, lower the lab supply voltage to the lowest voltage that will permit the motor to turn.

Listing 5: Stepper Motors

```
main:
    mov dptr, #0FE33h     ; Initialize the 8255 for 8 bit
    mov a, #80h
    movx @dptr, a
    mov R3, #0Ch          ; Count to 12, since we do 2 moves per cycle, this gets us 24
    lcall init            ; Setup Serial Commands
    lcall getchr          ; Get and Input Character
    lcall sndchr          ; Display Character in Serial

    loop:
    mov dptr, #0FE30h     ; Initialize in the location
    mov a, #01h           ;
    movx @dptr, a         ; Start the first coil
    lcall delay           ; Input Delay

    mov dptr, #0FE30h     ; Initialize in the location
    mov a, #04h           ; Switch off to the 2nd coil
    movx @dptr, a
    lcall delay           ; Input Delay

    mov dptr, #0FE30h     ; Initialize in the location
    mov a, #02h           ; Switch off to the first coil
    movx @dptr, a
    lcall delay           ; Input Delay

    mov dptr, #0FE30h     ; Initialize the 8255 for 8 bit
    mov a, #08h           ; Switch off to the 2nd coil
    movx @dptr, a
    lcall delay           ; Input Delay

    DJNZ R3, loop         ; Repeat until 12

    loop2:
    sjmp loop2

delay:
    mov R0, #0FFh            ; Set Registers with Counters
    mov R1, #0FFh
    mov R2, #07h
    sups:
        DJNZ R0, sups        ; Nested DJNZ to Count out Time
        mov R0, #0FFh
        DJNZ R1, sups
        mov R1, #0FFh
        DJNZ R2, sups
    ret

init:
    mov tmod, #20h
    mov tcon, #40h
    mov th1, #0fdh
    mov scon, #50h
    ret

getchr:
    jnb ri, getchr
    mov a, sbuf
```

```
58      anl a, #7fh
59      clr ri
60      ret
61
62  sndchr:
63      clr scon.1
64      mov sbuf, a; Move contents of the accumulator into sbuf
65      txloop:
66          jnb scon.1, txloop
67      ret
```

- Compare the stepper and DC motors qualitatively. Why might one be preferred over the other, depending on the application?

The stepper motor is much more precise in its movement, so it would be ideal for tasks that require a high level of positional control. It also has more torque, but sacrifices speed as opposed to the DC motor.