Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.115 Microprocessor Project Laboratory Spring 2024

Laboratory 3 and SPRING BREAK!

We do NOT expect you to work over Spring Break!

We do expect you to plan your time wisely, please.

Laboratory 3 is a "normal two week effort" for us, just like the other lab assignments, Lab 1 – Lab 4. However, Lab 3 is not "due" until three weeks after the date of issue. The "third week" is spring break. Notice, please, that Lab 3 is due on April 2, a few days after the end of spring break.

It is not possible to start the lab AFTER spring break and expect to finish. Please start Lab 3 early if you are traveling. Please think ahead for travel and weather issues. Travel delays will not receive extensions for the lab due date.

If you are planning to use spring break to do the lab – THIS IS NOT REQUIRED AND NOT OUR EXPECTATION, but fine if you want to – then please be aware that the department changes the 38-600 lab hours over spring break. Make sure you know when the 38-600 lab will be open so that you are not surprised.

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.115 Microprocessor Project Laboratory Spring 2024

Laboratory 3
Digital waveform synthesis

Issued: March 12, 2024 Due: April 2, 2024

GOALS: Further understand and modify the monitor code
Finish getting familiar with 8051-assembly language
Connect multiple peripherals to the R31JP
Make a collection of digital waveform synthesizers
Explore the amazing PSoC!

PRIOR to starting this lab:
Read: AD558 DAC data sheet,, LM358 data sheet, LM386 data sheet, LM18293/L293D datasheet, 8255 datasheet, Intel 8051 data book as needed. Review in Yeralan: p. 73—82, p. 164—167; in Scherz: "Operational Amplifiers," "Analog-to-Digital and Digital-to-Analog Conversion," and "Driving LCDs".

ON-LINE CODE SUBMISSION: Please **electronically submit** your **final** ASCII assembly (.asm) file of the "combined" program that you write in Exercise 5 to the course website **by 1 pm on April 2, 2023**.

EXERCISE 1: Modify the MINMON monitor code on your R31JP board

By now, your personal MINMON code has (at least) commands under the letters "D," "G," "R," and "W," for downloading code, executing from an address, reading an external memory byte, and writing an external memory byte, respectively. Let's add one more command, using the free "V" command letter. This new command will permit us to enter a table or vector of data bytes into external memory. Among other applications, you could use this table command to enter vectors of data that will describe "arbitrary" waveforms we might want our R31JP signal generator to reproduce as analog waveforms. Arbitrary waveform generators are a "hot" item on the test equipment market these days; see here, for example:

https://www.keysight.com/us/en/products/arbitrary-waveform-generators.html

Please do the following:

- Add a "V" (vector 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. A typical interaction might look like this in a terminal window:

```
MINMON>
* V94
9400 01
9401 02
9402
*
```

In this example, the user has chosen to write a table of two bytes to memory, beginning at location 9400. The activity begins when the user types V94 (This MINMON implementation automatically assumes an LSB of 00 for the external start address, you can too if you wish.). MINMON prints the address, e.g., 9400, and the user types the data, e.g., 01 in this case. Program entry stops after 256 bytes, or, as in this example, when the user hits return without entering data for a particular address.

- You can test your new MINMON commands by first using the existing MINMON on your R31JP board to download your new test monitor code to RAM, and then executing the test code in RAM by flipping the MON/RUN switch. Test your code carefully. After entering some data, use the "R" command we coded in the last lab to check the memory addresses you filled.
- Burn your finished monitor code into your EEPROM using the 6.115 chip programmer you saw during the laboratory familiarization.

EXERCISE 2: Analog output from a digital squirrel!

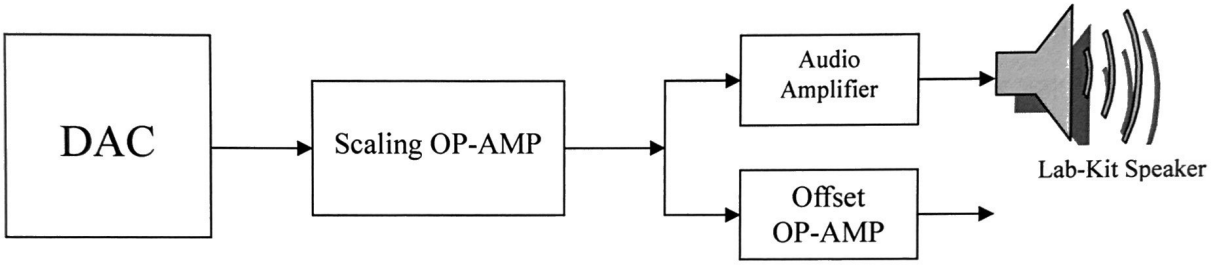
You will need to connect new peripheral chips throughout the term to make your R31JP perform in new, neat-o applications. For this lab, we need to add a digital-to-analog converter (DAC) to your kit. ***Read the spec sheet on your DAC carefully.*** Please do not destroy this expensive chip! Currently, you should still have your keypad and 74C922 controller wired to Port 1. You should also have an 8254 counter/timer wired up on your board, hogging all of the 256 available addresses for memory-mapped I/O devices. That is, the 8254 is directly selected by the XIOSELECT line from the R31JP.

Please do the following:

- Connect your DAC, the AD558, so that it is available through a memory-mapped I/O address. Configure your DAC for a 0 to 2.5 volt output range. Do not remove the 8254 or 74C922 from your board. 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 the new DAC wired up to different addresses in this space. This should leave 6 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 (read about it in Scherz, p. 337, but also check the 74138 datasheet, Scherz has some typos), 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 DAC. Again, use MINMON. Protect the output of the DAC with a non-inverting OP-AMP gain block. Let's call this the "scaling" OP-AMP. Set the gain so that the DAC and OP-AMP combination can produce an output voltage between 0 and 5 volts. Use the OP-AMP output to drive an LED in series with an appropriate resistor (Do NOT use one of the kit LEDs. Wire up your own, 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 the 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.

EXERCISE 3: Make a sine wave generator

Using the DAC, you can make your own sine wave signal generator. You already have the ability to adjust the amplitude of your generated waveform using the scaling op-amp. Let's add a little more analog circuitry to adjust the "offset" of your output waveform, and also to listen to your output waveform on a speaker. In block form, we want to create a circuit with the following functional blocks:



Please do the following:

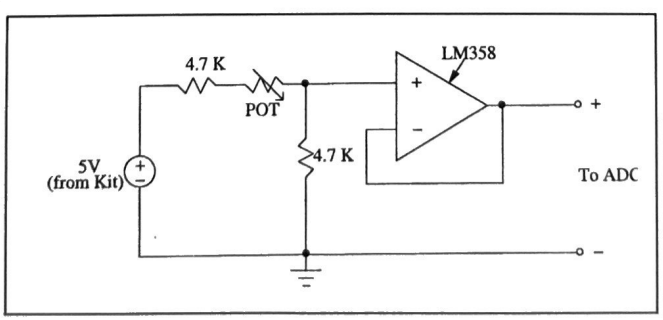
- Add the "Offset OP-AMP" shown above using an op-amp configured as a subtractor. Demonstrate that, with your scaling and offset OP-AMPS, you have an output that can be varied in amplitude and offset between 2.5 and –2.5 volts, just like a commercial signal generator.
- Add the "Audio Amplifier" block using an LM386 audio amplifier running between 12 volts and ground and configured with a volume control. **READ** the LM386 specification sheet – it includes reference designs from which you should select a useful circuit! In combination with the scaling OP-AMP and volume control, the LM386 should allow you to create a sine wave output to a speaker with no offset voltage and no saturation; in other words, it should look like a sine wave on the scope. **USE** bypass capacitors, positioned close to the LM386 power terminals.
- 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 MATLAB or a C program. 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. 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.
- 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 your DAC/OP-AMP circuit. Have your program loop to make an analog sine wave that repeats indefinitely. Make sure that there are no glitches in the sinewave, 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.
- Use your LM386 to drive the kit speaker, and listen to your sine wave. What happens to the sound if you increase the volume until the LM386 saturates? Explain.
- 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.
- Modify your program so that it divides each sample entry by 10 before writing the sample to the DAC. Then, use your scaling OP-AMP to restore this attenuation, i.e., use the OP-AMP to multiply the signal by an additional 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 sound like compared to the sine wave? What does it look like on the scope? Carefully (with detailed timing analysis) explain any changes in frequency, wave shape, and amplitude from the previous step.

EXERCISE 4: Add more useful peripherals to your lab kit

To make some of the experiments we will do in future exercises easier to understand, it will be helpful to have a convenient "readout" or display that you can use to indicate gain settings, command set points, measured values, etc. So, let's add a flexible display that's become relatively inexpensive these days: a liquid crystal display. We will issue you a special LCD for the R31JP, with pins that fit (be careful!) into a kit breadboard strip – this is NOT the LCD in the PSoC Big Board kit! The LCD is a small embedded system! Besides the actual liquid crystal screen, the display contains a simple microcontroller with on-board memory. To use the display, you first "program" it for the display mode of your choice, and then load data to be displayed in a memory bank in the LCD. The suggested reading section in Scherz will tell you what you need to know to connect and "program" the LCD display. As discussed in lecture, connect the LCD directly to the 8255 on your kit. Use MINMON to "play" with the LCD, poking instructions and data at the display through the 8255 until you are comfortable with it.
Please do the following:

- Connect the 8255 in the memory-mapped I/O space on your kit. This IC effectively adds three new "Port 1-style" general purpose I/O ports to your kit, as we discussed in lecture.
- 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).
- Connect your ADC0804 in the memory-mapped I/O space on your kit. Configure your ADC so that it accepts input voltages in the range from 0 to 5 volts, mapping these to digital bytes in the range from 00h to FFh. Make **VERY** certain that you never apply less than 0 volts or more than 5 volts to the ADC. These chips are expensive. Always check signals you apply to the input before you connect them to the ADC to make **sure** they remain in the acceptable range. Test your ADC by writing a program for the R31JP that samples the input to the ADC and displays the sampled byte on the Port 1 lights. Use the op-amp circuit shown below to drive the analog input of the ADC (create the variable resistor by using two leads on a potentiometer).



Use ground and your 5 volt power supply to power the LM358, ensuring that it's output voltage will be safe (below 5 volts). If you pick a reasonable pot, you should be able to get a variable voltage

source connected to the ADC that cannot go above 2.5 volts (a conservative choice for testing purposes), and which can get close to zero volts. Apply some DC levels to the analog input of the ADC and then compare the byte on Port 1 to the measurement of a real voltmeter connected to the input of the ADC0804. If everything is working, you should, for example, see a light pattern corresponding to 128 decimal on the port 1 lights when the voltmeter indicates 2.5 volts applied to the ADC input.

- Write a program for the R31JP that samples the input to the ADC and displays the voltage on the LCD screen. That is, make a simple voltmeter. Your voltmeter should be sensitive to one digit after the decimal, which is well within the resolution of your ADC. For example, if you apply 2.10 volts exactly to the ADC input, your voltmeter should display "2.1" on the LCD panel. Test your ADC by applying some DC levels to the analog input and then comparing your LCD reading to a real voltmeter connected to the input of the ADC0804. Once again, use this op-amp circuit shown above to drive the analog input of the ADC.
- READ the L293D datasheet. (NOTE: You may have an **LM18293 or an L293D** in your kit. They are the same for our purposes, and **we will refer to them interchangeably**.) You should have at least one of your three 8255 ports free and available. Pick four convenient 8255 output lines and wire them to the inputs of the four buffers on an L293D driver IC, pins 2, 7, 10, and 15. The L293D is a "power buffer" that provides more current than the 8255 can provide. Also, you get to select the output voltage of the L293D. Ground the four ground pins of the L293D. Connect L293D pins 1, 9, and 16 (two enable lines and a logic power input) to 5 volts on your kit. Connect the drive power input pin 8 to the 3003-style variable power supply on your bench Make sure that the 3003 power supply is grounded to your L293D and your kit. BYPASS! By adjusting the variable voltage, you can set the output voltage of the L293D. When an input to the L293D is logic high, the output of that L293D buffer will be at +V. A low input brings the associated L293D output to ground.
- For now, keep the variable kit power supply voltage limited to 5 volts. Test your system, e.g., use the L293D to drive some suitable resistors to make sure that your system works. Again, use MINMON's "R" and "W" commands to write to the 8255 and thereby set the status of the L293D outputs.
- Check to make sure that everything else on the kit still works, e.g., the 8254, DACs, etc.

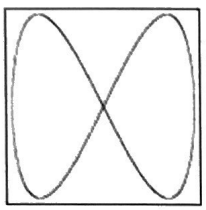
EXERCISE 5: An "application-specific" signal generator: Lissajous Curves.

ATTENTION: the Lazerdillo lab station is extremely delicate. We will tolerate no abuse of this hardware. Be especially careful to avoid shorting the Lazerdillo board or mechanically mishandling the optics or mechanical mechanism. The last page of this lab contains a schematic of the circuit built on to the Lazerdillo lab station. Be sure you understand it, particularly the DAC interface to the R31JP ("Data in" and "Control in" in the schematic). PLEASE do not destroy or damage the lab stations! Use Lazerdillo's built-in laser as little as possible, leave it off when you do not need to see the beam by using the LSR control line (see below).

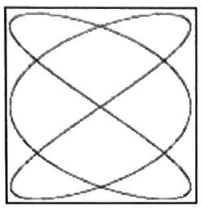
Lissajous curves are a staple of laser light shows everywhere. Generated parametrically by sine waves, surprisingly intricate figures can be generated through the calculated modulation of their relative frequencies. As discussed in lecture (review the lecture notes), the basic Lissajous equations are:

$$\mathrm{x} = A\sin(2\pi * at) \qquad \mathrm{y} = B\sin(2\pi * (bt + \phi))$$

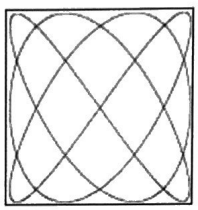
The structure of a Lissajous curve is defined by the ratio of the two frequencies, a and b. Some examples of Lissajous figures are depicted below with their associated frequency ratio:



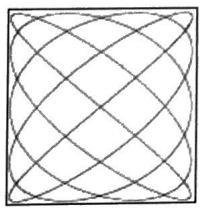
$\frac{a}{b} = \frac{1}{2}$



$\frac{a}{b} = \frac{3}{2}$



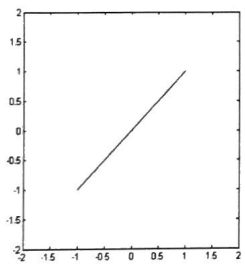
$\frac{a}{b} = \frac{3}{4}$



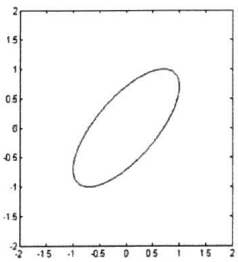
$\frac{a}{b} = \frac{5}{4}$

In the Lissajous equations, phi (ϕ) specifies the relative phase offset of the sinusoids and therefore determines the orientation of the figure. Below, we review examples of a Lissajous figure with frequency ratio a/b=1/1 for different values of phi:

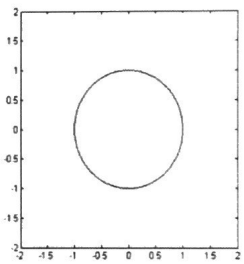
$\phi = 0$



$\phi = 1/8$



$\phi = 1/4$



In this exercise, we will produce Lissajous curves by combining the signal generator functionality of the R31JP with the Lazerdillo's display.

For our purposes, Lazerdillo has several constraints that influence its ability to generate Lissajous figures. The frequency response of the Lazerdillo is limited by the inertia of the mirrors. Above 80 Hz the Lazerdillo's stepper motors cannot apply enough torque to oscillate them fast enough. As a result the magnitude of the output drops off significantly and Lazerdillo is unable to depict the figure. The second constraint deals with the optical refresh rate. To meet the blending limitations imposed by your eye, Lazerdillo must complete the entire figure within $1/20^{th}$ of a second. When the figure takes longer than $1/20^{th}$ of a second to complete, each frame (or section of the Lissajous figure) will become distinct and the entire display will seem to flicker or rotate.

To generate the sinusoidal signals you will be using a 256-byte sine wave table to select values for the two AD558 digital-to-analog converters built into the Lazerdillo. Use the MOVX A, @A+DPTR instruction to access your stored data. As you know, the entry in DPTR points to the base of the table, and A is the offset used to select the value of interest from the table.

To calculate the table advance for a specific frequency we need to calculate the interrupt rate. The R31JP has an 11.0592 MHz clock that runs one machine cycle every 12 clock cycles. When operating Timer 0 in Mode 2, register TL0 counts up to 255, at which point it overflows and can generate an interrupt. On an overflow, the value in TH0 is placed in the counter TL0 and TL0 counts up to 255 again. It is important that you leave enough time for computation between interrupts. This time ensures that the previous interrupt is handled before the next one begins. An interrupt frequency of TH0 = #04Ch is recommended because it provides adequate time for computation and makes the calculations easier to implement.

This frequency results in an interrupt rate of:

$$11.0592M\frac{\text{clock cycles}}{\text{second}} \div 12\frac{\text{clock cycles}}{\text{machine cycles}} \div 180\frac{\text{machine cycles}}{\text{interrupt}} = 5120\frac{\text{interrupts}}{\text{second}}$$

From the interrupt rate we can calculate the table advance needed to maintain a specific sine wave.
For 80 Hz we have:

$$256\frac{\text{table elements}}{\text{period}} \div 5120\frac{\text{interrupts}}{\text{second}} * 80\frac{\text{periods}}{\text{second}} = 4\frac{\text{table elements}}{\text{interrupt}}$$

This is fine for the 80 Hz sine wave, because the number of table elements per interrupt is an integer. If we try to make a 54 Hz sine wave, though, we run into problems.

$$256\frac{\text{table elements}}{\text{period}} \div 5120\frac{\text{interrupts}}{\text{second}} * 54\frac{\text{periods}}{\text{second}} = 2.7\frac{\text{table elements}}{\text{interrupt}}$$

We cannot step through the sine wave table with a step rate of 2.70 steps per interrupt using integer steps. In order to solve this problem, we can use a 16-bit count. As discussed in lecture, instead of using one register, we use two. One register stores the "integer" part of the necessary step rate (0-255) while another one stores the "fractional" part of the step rate (0/256 - 255/256, i.e., 0.000 - 0.996). In the case of a 54 Hz sine wave, the high byte would be #02h, while the low byte would be #0B3h (Why?).

If the frequency with which we draw our figure is not fast enough to cover the whole figure, due to the mechanical constraints of the mirrors, the figure will appear to rotate.
For example, with the equations below, we will try a figure with a = 54 and b = 80. This figure has many lobes in both x and y dimensions and is hard to visualize. It is also difficult for Lazerdillo to draw. The mirrors will not move fast enough to complete a full 54 lobe by 80 lobe image in the fusion time for the eye, about 1/20th of a second. We can model what will happen by realizing that a 54/80 figure is essentially a 54/81 figure (a/b=2/3) with a time-varying phase shift:

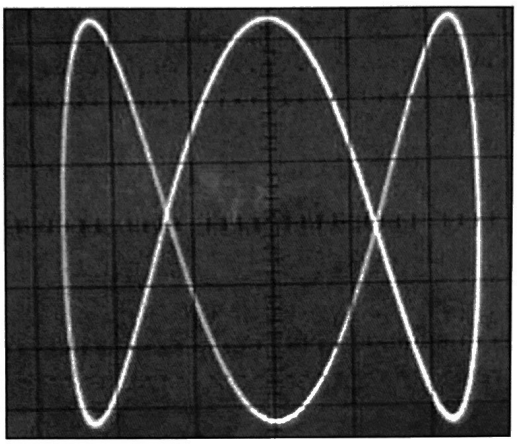
$$\mathrm{x} = \sin(2\pi * 54t) \qquad \mathrm{y} = \sin(2\pi * 80t)$$

$$\mathrm{x} = \sin(2\pi * 54t) \qquad \mathrm{y} = \sin(2\pi * (81t - t))$$

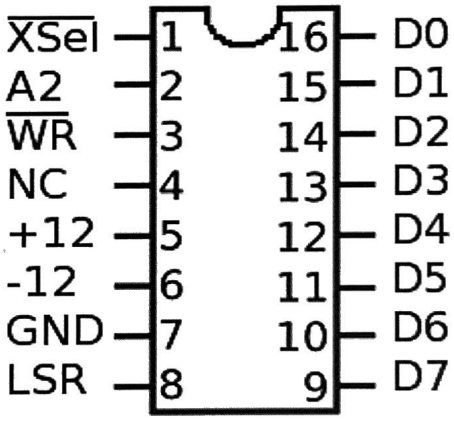
$$\phi_{\mathrm{y}}(t) = -t \qquad \text{frequency} = 1 \text{ Hz}$$

Lazerdillo will effectively render the 54/80 equations as a 2/3 Lissajous figure with a time-varying phase shift that causes the figure to appear to rotate once per second. If we use a 16-bit counter, as described above, we can produce rotation frequencies at fractions of a Hertz.

Now that you understand how Lissajous figures work, we're going to make a TIE fighter:



Here is the pinout of Lazerdillo:



The XSel# pin is like a "chip select" that enables Lazerdillo to receive new DAC values. It is active low. When the pin is high, Lazerdillo will not accept new X and Y DAC values. Lazerdillo's XSel# line takes logic values as an input. (+5 or 0)

The address pin A2 selects between the X and Y DACs built into the Lazerdillo. Each DAC controls a voice coil actuator (in this case, a stepper motor that we will use without making full rotations). This voice coil actuator rotates a mirror through a few degrees of rotation based on DAC command voltage. Like XSel#, A2 takes a logic value as an input.

The WR# pin triggers a write to the DAC currently selected by A2. When this pin goes low, the value stored in the DAC is sent to the stepper motor. This also takes a logic value as an input.
NC means "not connected". Do not use NC pins.

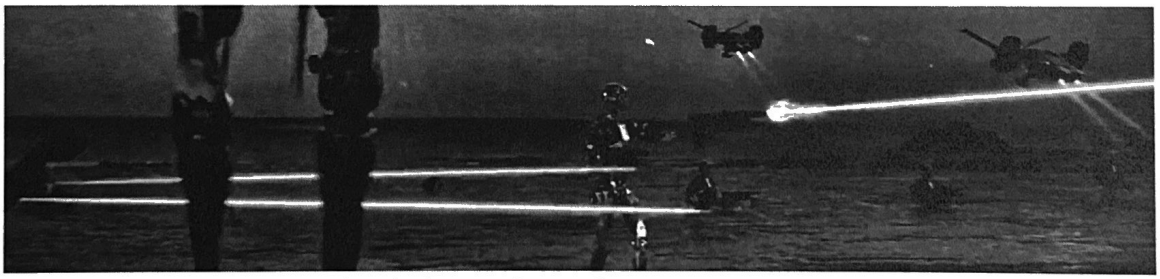
The LSR pin triggers the laser to operate. It takes a logic value (+5 or 0) as an input. When the pin is high, the laser is on. **PLEASE minimize your use of the laser to preserve it**. Keep LSR low and the beam off when you do not need it.

Using Lazerdillo, you're going to produce a TIE fighter, a circle, and a figure with a=1, b=4, phi=0 (i.e. no rotation). You will then make the figures rotate. You will then make a program that combines these three shapes and selects between them, with a selectable rotation rate, by using your keypad.

Please do the following:

- Implement an R31JP program that makes Lazerdillo draw a TIE fighter without rotation. Use 16-bit counters to step through and keep place in the sine table.
- 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).
- Now you are able to draw the three figures mentioned above. Write a program that allows the TIE fighter to rotate. Choose a rotation frequency between 0 Hz and 1.5 Hz. You should be able to change the rotation frequency by editing the program and re-assembling.
- 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) and a rotation frequency. Use Lazerdillo to draw the shapes, and use the keypad to select between the possibilities. Any of the 3 shapes should be selectable from the keypad, and any of the three shapes should be able to rotate if desired. Allow the shapes to rotate at a frequency selected between 0 Hz and 1.5 Hz. **This is the "combined" program you should submit online to the course website.**

(NOTE: if you have trouble with your keypad, do not let this stop you from getting the light show working. For example, you could make an "emergency no-keypad program" that simply switches between the three figures every few seconds. But the keypad version is preferred for checkoff.)



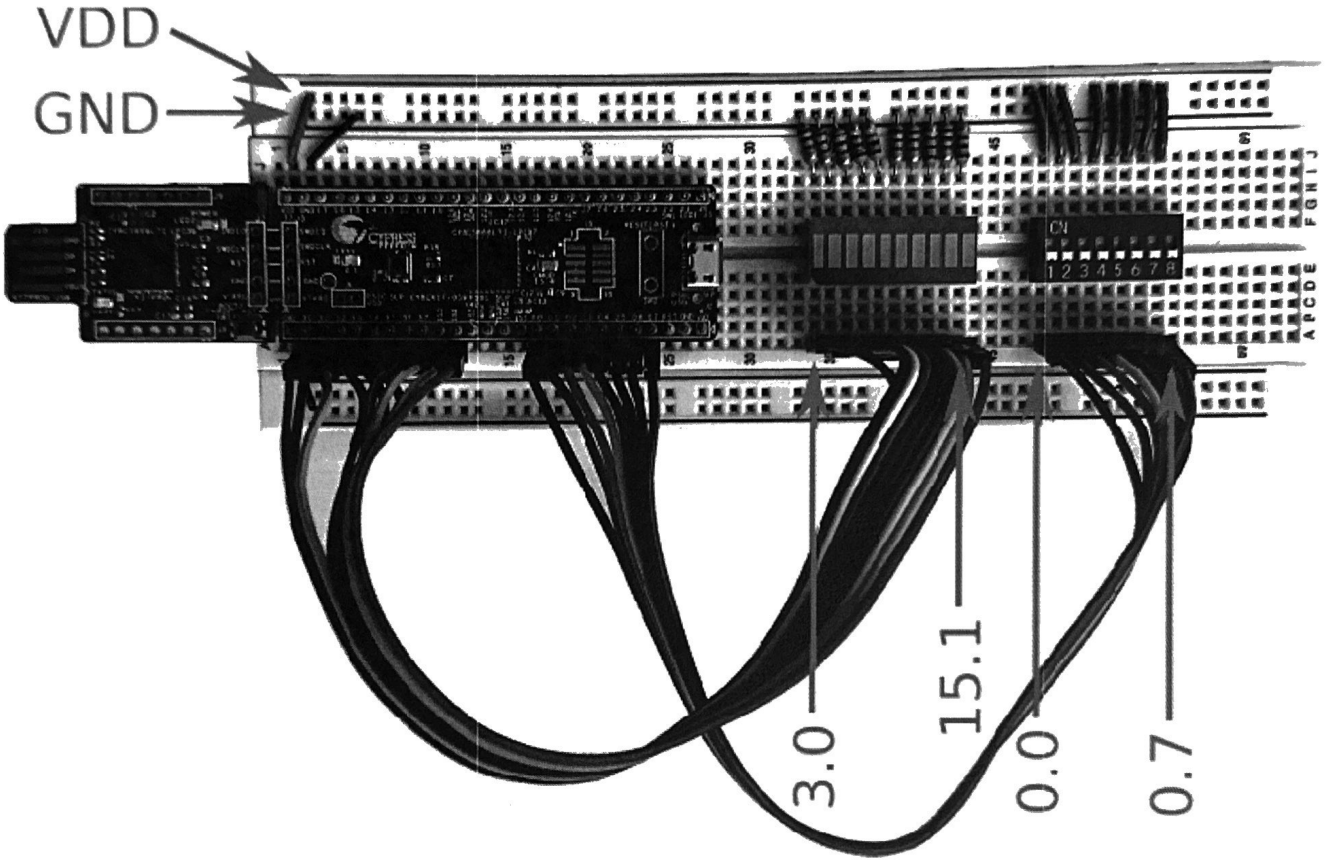
(Laser light shows are a timeless favorite in every future! If you have not already won a gift certificate, be the first to e-mail Professor Leeb with the exact title of the movie with this scene to claim your $10 Amazon certificate.)

EXERCISE 6: Secrets of the Squirrel!

You may have been wondering how a CPU squirrel is actually implemented. The answer is that most CPU squirrels are sophisticated "Finite State Machines" (FSM's). Classic squirrels were implemented with finite state machines built around a memory or "look-up table" to define behavior. The entries in the look-up table were a kind of program sometimes called "microcode". More recent squirrels are constructed with complex arrangements of digital gates and latches that permit the direct implementation of one of several types of FSMs. We can experiment with both of these techniques using exercises from the VE book and the magic of PSoC Creator!

In overview: we will work through a collection of lab exercises in VE. In each exercise, you will connect some PSoC hardware in Creator to make different arrangements of digital gates. To exercise the digital gate circuits that you build, you will need some switches to set some "inputs" high, and some LEDs with current limiting resistors to see or display the outputs of your digital circuit creations.

The input switches and output LED's are provided by the Kovid Konsole that you built in Exercise 6 of Lab 1. A photograph of the staff Konsole is shown again for reference below. You tested your version of this back in Lab 1:



For now, power your Kovid Konsole through the STICK's USB programming connector that hangs off the left edge of the breadboard in the picture; *add no other source of power*! You can use the USB extension cable in your kit to both program and power the Konsole from your PC.

Our intent here is to learn and complete the circuit lessons in VE; do not worry if a detail in VE cannot be followed precisely. For example, the VE book refers to a "DigitalTestBench" Creator project that will NOT be provided to you. This is not required and can easily be set up manually. However, take careful note of the correct PSoC parts and connections shown in the VE circuit diagrams.

Please do the following:

- 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, and 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.

Lazerdillo PC Board Schematic: