Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.115 Microprocessor Project Laboratory Spring 2022

Laboratory 3
Digital Waveform Synthesis

Issued: March 9, 2022 Due: March 30, 2022

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, 8255 data sheet, LM358 data sheet, LM386 data sheet, Intel 8051 data book as needed. Review in Yeralan: p. 73—82, p. 164—167; in Scherz (3rd ed), p. 635-661, p. 820-829.

EXERCISE 1: Further explore MINMON

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 explore one last command, provided under the "T" command for KOVID KAPABLE and "V" command letter otherwise. 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 market these days; see here, for example:

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

KOVID KAPABLE (APPLIES TO EXERCISE 1 ONLY!): Please do the following:

- 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. A typical interaction might look like this in a SecureCRT window:

```
MINMON>
* T94
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 T94 (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. The next memory location (9401) is automatically displayed on a new line, prompting the user to enter the next byte of data. 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.

KOVID KAN'T (APPLIES TO EXERCISE 1 ONLY!): Please do the following:

- With your MINMON ROM from your yellow parts box inserted in your R31JP: Using the "V" (vector table write) command 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, terminating the entry process. The hex bytes should be printed on the PC screen. A typical interaction might look like this in TeraTerm:

```
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.). 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.

Try it! Enter a table of data for at least 10 or so addresses. Then use the MINMON "R" command to inspect the memory locations and make sure that you understand how the "V" command works!

EXERCISE 2: Add more useful peripherals to your lab kit

You will want to connect new peripherals to make your microcontrollers perform in new, neat-o applications. The PSoC makes this easy, as we will see later in this lab. The R31JP is wonderful because it exposes how the hardware connections for new peripherals work "at the metal" level. For this lab, we will add two digital-to-analog converters (DACs) to your R31JP kit. We will also add an 8255 parallel port, which effectively gives us three more ports similar to the "Port 1" already provided by the 8051/R31JP. ***Read the spec sheet on your AD558 DAC carefully. Read the spec sheet on the 8255 carefully.*** Please do not destroy these expensive chips! Currently, you should still have your keypad and 74C922 controller wired to 8051/R31JP Port 1. You should also have an 8254 wired up on your board, hogging all of the 256 available addresses for memory-mapped I/O devices, i.e., currently the 8254 is directly selected by the XIOSELECT line from the R31JP.

Please do the following:

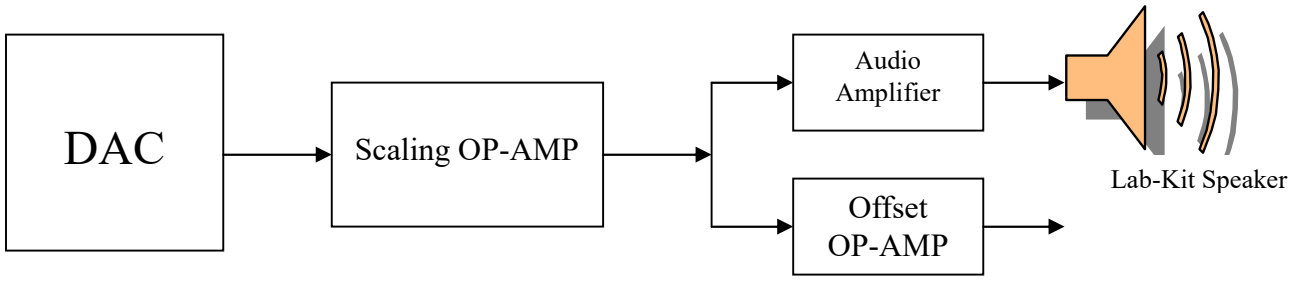
- Connect your two DACs so that they are available through memory-mapped I/O addresses. Configure your two DACs for a 0 to 2.5 volt output range. Do not remove the 8254 or 74C922 from your kit. Instead, use a 74138-type logic part that will use the XIOSELECT line and the address lines cleverly so that 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 datasheet carefully), address lines A4, A5, and A6, and the XIOSELECT line. Include a detailed circuit schematic in your lab write-up. Bypass!

- 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 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 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.

EXERCISE 3: Make a sine wave generator

We will NOT make you add the following additional hardware to your kit, but be aware that you could add an "Offset OP_AMP" and also an LM386 audio amplifier to your kit, creating two DAC circuits that might look like this:



The "Offset OP-AMP" shown above would likely be an LM358 op-amp configured as a subtractor. Using scaling and offset OP-AMPS, you could have an output that can be varied in amplitude and offset

between 2.5 and –2.5 volts, just like a commercial signal generator. The "Audio Amplifier" block might be an LM386 audio amplifier running between 12 volts and ground and configured with a volume control. You may want this for your final project – if so, **READ** the LM386 specification sheet – it includes reference designs from which you could select a useful circuit! In combination with the scaling OP-AMP and LM386 volume control, the LM386 would 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. If you make use of the LM386 during your final project, **USE** bypass capacitors, positioned close to the LM386 power terminals. The LM386 is very picky about having good bypassing.

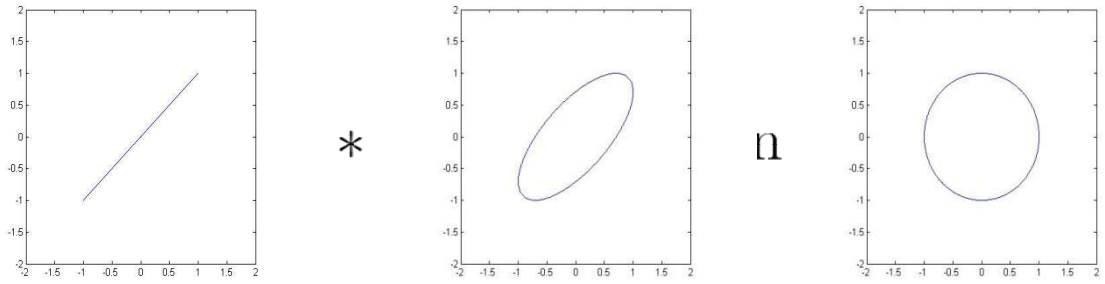
Please do the following:

- 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.
- 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.
- 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, 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.

EXERCISE 4: An "application-specific" signal generator: Lissajous Curves.

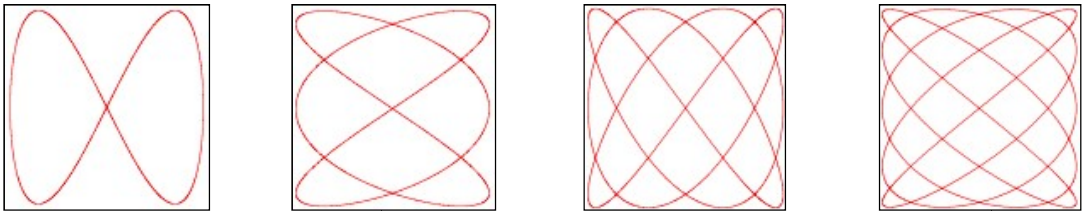
In lecture we will examine an experimental setup for laser light drawing called **Lazerdillo**. Lazerdillo is a wonderful tool for making "laser light shows" like you might have seen at the Science Museum or elsewhere. As we will discuss in lecture, the physical construction of Lazerdillo limits how quickly its

mirrors can be moved, and therefore how quickly the laser beam can be "dragged" across a projection surface. For maximum flexibility, let's build a similar light show using the oscilloscope in X-Y mode as a display! Figure out how to put your scope in X-Y mode. You will be able to make the beam dot move around the screen by putting different voltages on Ch1 and Ch2 of the scope. The scope is faster than Lazerdillo; nevertheless, we will design our light show to meet Lazerdillo's speed limitations.



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:

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, the variable 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 oscilloscope operating in X-Y mode. Use one of your scaling OP-AMP outputs from one of your DAC circuits to drive oscilloscope Channel 1. Use the other scaling OP-AMP output from the "other" DAC to drive oscilloscope Channel 2. With the scope properly configured, you should be able to use MINMON make a point of light "move around" the X-Y space of the oscilloscope display. We will pretend that the scope is limited to the same dynamic speeds as Lazerdillo.

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 1/20th of a second. When the figure takes longer than 1/20th 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 on your kit. 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:

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

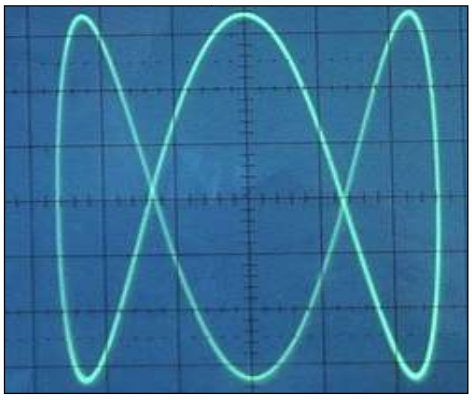
$$\mathrm{x} = \sin(2\pi * 54t) \qquad \mathrm{y} = \sin(2\pi * (81t - t))$$

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

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. Will the oscilloscope give the same behavior?

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

Using the oscilloscope in X-Y mode (instead of Lazerdillo), you are going to produce a TIE fighter (as shown to the right), a circle, and a figure with a=1, b=4, phi=0 (i.e. no rotation). You will then make a program that combines these three shapes and selects between them using the keypad.



Please do the following:

- 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.
- 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.

(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.)

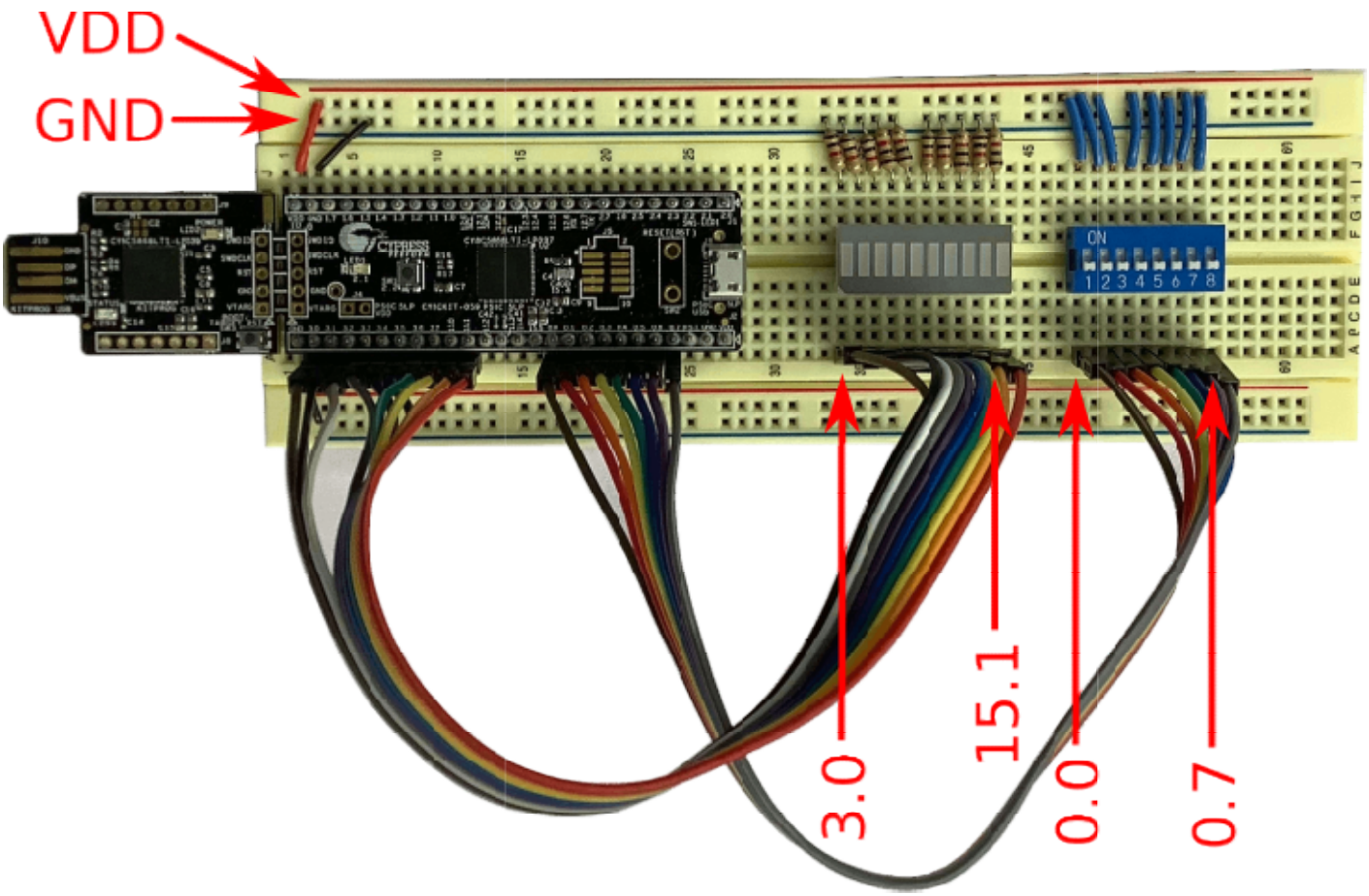
(Laser light shows are a timeless favorite in every future! If you have not already won a gift certificate, be the first to e-mail Professor Leeb with the exact title of the movie with this scene to claim your $10 Amazon certificate.)

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. Compress your PDF lab report to make sure that the pictures do not make the lab report larger than 2MB, e.g.,:
  https://smallpdf.com/compress-pdf