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

Laboratory 2
Timing and Controlling Events in the Physical World

Issued: February 23, 2022 Due: March 9, 2022

GOALS: Understand and modify the monitor code running on your R31JP board
Understand the structure of an Intel Hex file
Learn to use interrupts and timers
Connect a peripheral chip to your R31JP
Take control of a physical system (LED lamp) using your R31JP
Learn about the PSoC "Programmable System on Chip" and the C programming language

PRIOR to starting this lab:
Read: Scherz (3rd Ed): pages 635 – 661, especially 638 and 644. Intel 8254 data sheet (skim), anything you didn’t finish reading from Lab 1.
Note that pages 2-21 – 2-24 in the MCS-51 Microcontroller User’s manual tell you how many machine cycles per instruction. PSoC background (see the 6.115 website).

NOTE: This Laboratory 2 includes our first exercise marked “KOVID KAPABLE:”. Future labs may have similarly marked exercises. Exercises marked this way are NOT optional. If you are well and can get to the 38-600 laboratory, please do so and complete the exercise. Alternatively, if you are generally “well” but unable to go to the lab, e.g., quarantining, AND you have made a reasonable effort to do the exercises on time, i.e., you did not leave it to the last day of the lab, AND you have let Professor Leeb and the staff know that you are unable to go to the lab, then please instead complete the “KOVID KAN’T:” part of the exercise.

ALSO NOTE: We use the phrase “hyperterminal” (a program available in some versions of Windows) to refer to your serial communication program. This term is a placeholder for your terminal program, e.g., TeraTerm, Putty, Realterm, etc.

EXERCISE 1: Check out more MINMON monitor code capability on your R31JP board

Currently, you have hopefully used your MINMON “D” and “G” commands.

KOVID KAPABLE: (APPLIES TO EXERCISE 1 ONLY!)

In this exercise, we’ll add “read and write” capabilities to MINMON. Use the currently free “R” and “W” command letters. These commands will be **very** useful for the remainder of the term, especially when testing new chips in your 6.115 kit.
Please do the following:

- Add a “read” command to MINMON. At the MINMON prompt, you should be able to type a command that reads the hexadecimal contents of a byte in external memory on the R31JP board. This hex byte should be printed on the PC screen. A typical interaction might look like this in a Hyperterminal window:

```
* R9000
FF
```

In this example, we’ve chosen to read the contents of memory location 9000. The R31JP board has read the contents of this memory location (FF in this example) and printed it on the PC terminal screen.

- Add a “write” command to MINMON. This command should load a hex byte into a specified location in external memory. A typical interaction might look like this in a Hyperterminal window:

```
* W9000=FF
*
```

In this example, we’re loading the memory at location 9000 with the byte FF (hex).

- 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. Read and write to several locations in memory to make sure you understand the behavior of your code. Burn your finished monitor code into your code EEPROM using the 6.115 chip programmer you saw during the laboratory/kit familiarization lecture. You will not be able to use W to write to every location in external memory. Be prepared to explain why at your check-off.

KOVID KAN’T: (APPLIES TO EXERCISE 1 ONLY!)

In the yellow parts box included in the kit issued to you, you will find a “pre-burned” MINMON 28C64 IC ROM. We have programmed this ROM with a “staff copy” of MINMON with additional commands “built-in.” Install the ROM in your R31JP very carefully – ASK FIRST if you are not sure how to do this after seeing the kit familiarization lecture. The “greeting” message for this “staff ROM” should include a smiley-face logo like this: :D

In this exercise, we will explore the “read and write” capabilities built into the staff MINMON. These commands will be **very** useful for the remainder of the term, especially when testing new chips in your 6.115 kit.

Please do the following:

- Test the “write” command in MINMON. This command should load a hex byte into a specified location in external memory. With the R31JP in MON, a typical interaction might look like this in a TeraTerm window:

```
* W9000=FF
*
```

In this example, we’re loading the memory at location 9000 with the byte FF (hex).

- Test the “read” command in MINMON. At the MINMON prompt, you should be able to type a command that reads the hexadecimal contents of a byte in external memory on the R31JP board. This hex byte should be printed on the PC screen. A typical interaction might look like this in a TeraTerm window:

```
*  R9000
FF
```

In this example, we’ve chosen to read the contents of memory location 9000. The R31JP board has read the contents of this memory location (FF in this example) and printed it on the PC terminal screen.

- Read and write to several locations in memory to make sure you understand the behavior of your code. You will not be able to use W to write to every location in external memory. Be prepared to explain why at your check-off.

EXERCISE 2: Reverse assemble an Intel hex file

Please examine the Intel Hex file shown in the text box below. This file was produced by AS31 from a text file containing an 8051 assembly language program.

```
:10000000740085E090D29280F75061727479206F0D
:0B0010006E20696E20362E3131352144
:00000001FF
```

Please do the following:

- "Reverse assemble" the program. That is, read the hex file, above, and write down the assembly language file that, assembled with AS31, would produce the file above.
- Assemble your assembly language file using AS31 and make sure that it produces the hex file in the text box.
- Explain what the program does.
- The assembly language file contained a secret message. If you are the **first** person (who has not already won a gift card) to correctly **e-mail** the message to Professor Leeb, **exactly**, then claim a $10 Amazon gift card from Professor Leeb.

EXERCISE 3: A simple calculator

Let's take your new MINMON for a test drive. Please do the following:

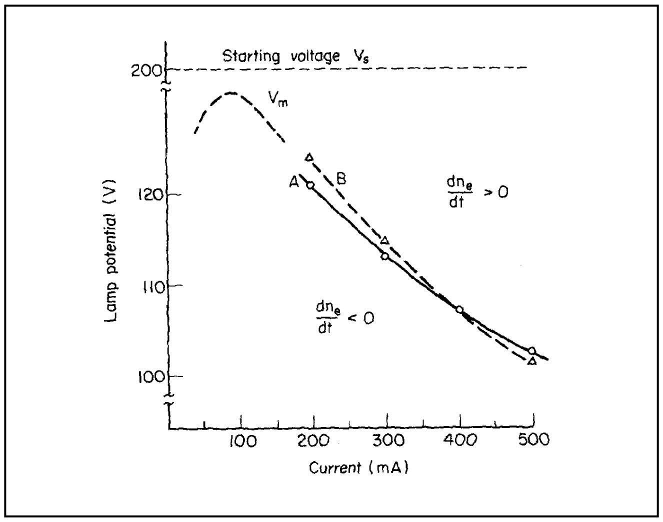
- Write a program that reads two bytes from sequential locations in external memory, e.g., 9000h and 9001h. The program should compute the sum, difference, product, and the quotient of the two numbers. Store the results, i.e., sum, difference, 16-bit product, and quotient and remainder in sequential external memory locations without overwriting the original two bytes. Make the program easy to use repeatedly. It should be executed using the MINMON "G" command, **not** by toggling the MON/RUN switch. That is, the R31JP board should stay in MON mode (red LED lit) at all times. You should enter the operands at 9000h and 9001h using your new "W" command. Complete the calculations using the "G" command, and read results from memory using your "R" command.

EXERCISE 4: Get comfortable with embedded lighting control

A key goal of 6.115 is to become familiar with the general technique of using an embedded controller to run a physical system. In this laboratory you'll use your R31JP board to control the operation of a lighting circuit. In lecture, we will look at controlling a fluorescent lamp. In the lab exercises here, you will use the R31JP to control an LED (solid-state) lamp. A key point is that "modern" light bulbs generally require a circuit to operate. This circuitry is typically more sophisticated than the circuitry used for incandescent bulbs. Why? The physics of the light producing mechanisms in fluorescent lamps and solid-state lamps, while different, both require a tailored control for the lamp voltage and current. Let's start by considering the fluorescent lamp we will look at in lecture (and again on the quiz!).

A fluorescent lamp is a sealed glass tube with electrodes at either end. During the manufacturing process, the tube is coated with a material called a phosphor, the white powder you may have noticed on the walls of a broken tube. The tube is evacuated of air, and refilled with mercury gas at a low pressure. In operation, an electrical arc or discharge fills the tube in response to a voltage difference between the electrodes. The arc excites the mercury gas, creating charged plasma that emits ultraviolet light, which stimulates the phosphors and makes them glow.

A fluorescent lamp is very different from an incandescent lamp. A fluorescent lamp bulb cannot simply be "plugged in" to the wall. It requires a special circuit called a "ballast". The reason that the lamp cannot simply be "plugged in" to a voltage source is indicated by the V-I characteristic for a typical fluorescent lamp, shown in the figure below (from "Electric Discharge Lamps" by Waymouth). When the bulb is "cold," i.e., off and not producing any light, it exhibits high impedance between its electrodes. A high voltage must be applied across the lamp to initiate an arc. This is the "Starting voltage" shown in the figure. Once the arc fires or "strikes", the gas in the tube begins to ionize. Once the gas ionizes, the



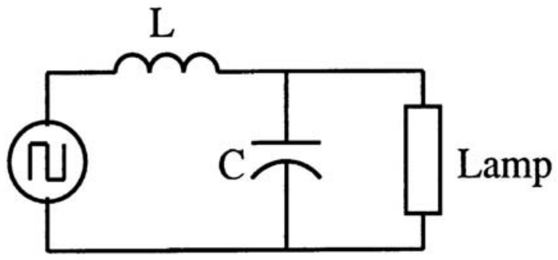
impedance of the lamp is relatively low compared to the cold tube. This is indicated by the "warm" or "struck" or "lit" V-I characteristics in the figure, which show the bulb performance for two different tube temperatures (40 Celsius for curve A, 23 Celsius for curve B). These curves show loci of equilibrium V-I points once the bulb has struck. If the starting voltage is maintained across the tube, an enormous current would eventually flow into the tube until something broke, e.g., a fuse blew, the filaments in the lamp broke, a wire melted, etc. So, once the bulb has "struck", the terminal voltage must be reduced to a point on the equilibrium curves that will produce the desired current and illumination.

Reaching a stable equilibrium point on the struck V-I curve is tricky. If you examine the curves in the figure carefully, you'll notice that, for useful current ranges where illumination is produced (between 100 and 500 mA) an **increase** in terminal voltage corresponds to a **decrease** in terminal current, and vice-versa. Roughly speaking, this happens because, as the current decreases in the tube, the number of charged carriers in the tube also decreases, decreasing the conductivity of the plasma column in the tube. So a higher voltage is needed to maintain the lower current! Increasing the current, on the other hand, increases the conductivity of the plasma. A lower voltage is required to sustain the higher current.

This odd behavior makes it difficult to keep the lamp stable on the V-I equilibrium curve. Imagine, for example, that we plug the lamp into a fixed voltage source and that, somehow, the lamp is struck and a perfect voltage is applied that will keep the lamp on the equilibrium curve. Now, imagine a slight, inevitable disturbance that momentarily increases the current in the bulb. This disturbance could be a slight change in exterior temperature, for example. The voltage across the tube remains fixed, but now we are "off" the equilibrium curve, with a larger number of charge carriers in the tube compared to before the disturbance. Off the equilibrium curve, this voltage will push yet more current into the bulb, further increasing the conductivity. If the voltage remains unchanged, the bulb enters a "runaway" condition where the current increases until something breaks.

A ballast circuit is used to start the lamp and keep it at a stable equilibrium point once struck. One easy way to solve the stability problem would be to put the lamp in series with a conventional linear resistor. For a suitable value of resistance, this series combination would appear to have an overall positive linear resistance characteristic. We could then apply a high voltage to strike the lamp/resistor combination, and lower the voltage to run at a stable equilibrium point in steady state. The problem with this plan is that the resistor dissipates power. This creates heat in the lamp fixture and also decreases the efficiency of the lamp.

A practical ballast circuit might look something like this:

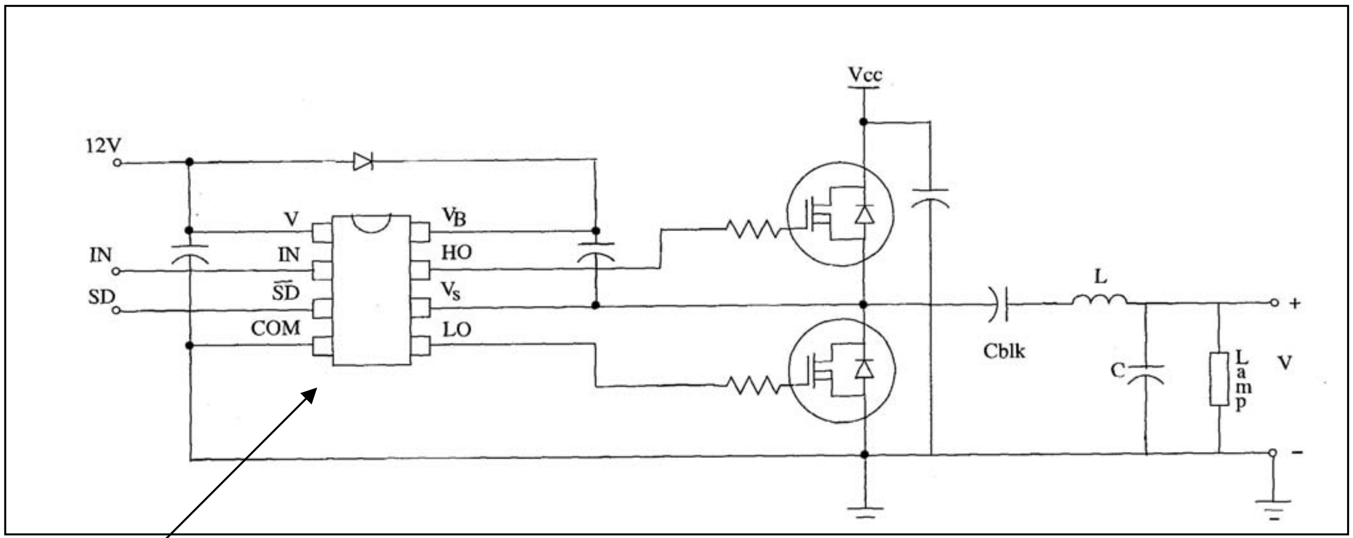


In this ballast, a square wave produced by the input voltage source drives a resonant circuit that consists of an inductor, a capacitor, and the lamp. When the lamp is cold and off, we can approximate it as an open circuit. In this case, the square wave source is driving an LC resonant circuit. If the square wave is at the resonant frequency, and the parasitic resistances in the inductor and capacitor are small, a huge voltage will be generated across the inductor and capacitor. The capacitor voltage will strike the lamp. Once the lamp is struck, it has a relatively low impedance. The capacitor is in parallel with this low impedance, and the resonant quality (Q) of the circuit is reduced dramatically. Roughly, you can imagine that the lamp "shorts out" the capacitor, leaving the circuit as an inductor in series with the lamp. The inductor serves as the "ballast" impedance that limits the current through the lamp. The square wave voltage in the circuit above must be large enough to drive the steady state current (between 100 and 500 mA) into the inductor.

An AC voltage is used for at least two reasons. First, AC permits "lossless" reactive elements to serve as ballast impedance. Specifically, an inductor can be used instead of a resistor to limit the current through the lamp. If we used a DC input voltage, the inductor would have essentially zero impedance, and would not serve to limit current. Second, the bulb lasts longer if the two filaments occasionally swap roles as electrode emitter and receiver (cathode and anode).

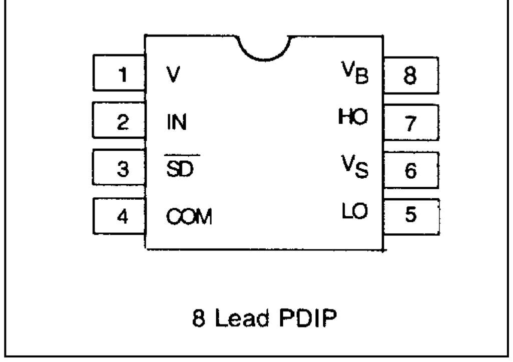
If the resonant circuit has a "high Q" (low-loss) and is driven near its resonant frequency, the capacitor voltage will be large enough to strike the lamp. This presents an interesting engineering economics and manufacturing problem when making a lamp ballast. To make sure that the lamp will strike in a range of customer operating environments, the LC tank should be tightly tuned with low loss. This will create a strong "peak" in capacitor voltage at the resonant frequency. Unfortunately, this approach also means that the frequency of the square wave must be tightly controlled to be very near the resonant frequency. Therefore, either the L and C values must be tightly controlled and unchanging, or the square wave frequency must adapt to changes or tolerances in the L and C values. Precision power-level inductors and capacitors are expensive, and incompatible with the production of cheap ballasts. It is generally cheaper overall to use lower tolerance inductors and capacitors, and then program a microcontroller to "find" the resonance frequency to strike the lamp. What would a circuit look like for driving the fluorescent lamp?

Here's a simplified circuit diagram of a fluorescent lamp ballast that we will look at in lecture:
(NOTE: Everything in the boxed circuit schematic below is potentially at high voltage! The lamp may need hundreds of volts to strike! That's why we will only look at this circuit together in lecture, but you will NOT build it.)

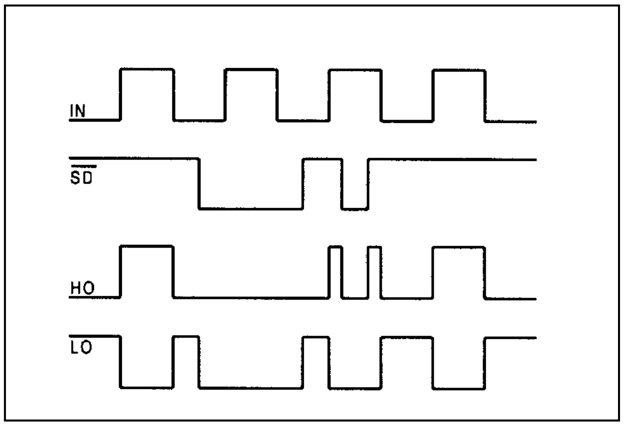


This chip (above this text) is the IR2104 from International Rectifier.

You can find out more about the IR2104 by examining its spec sheet on the International Rectifier web site, www.irf.com. The pinout of the IR2104 is:



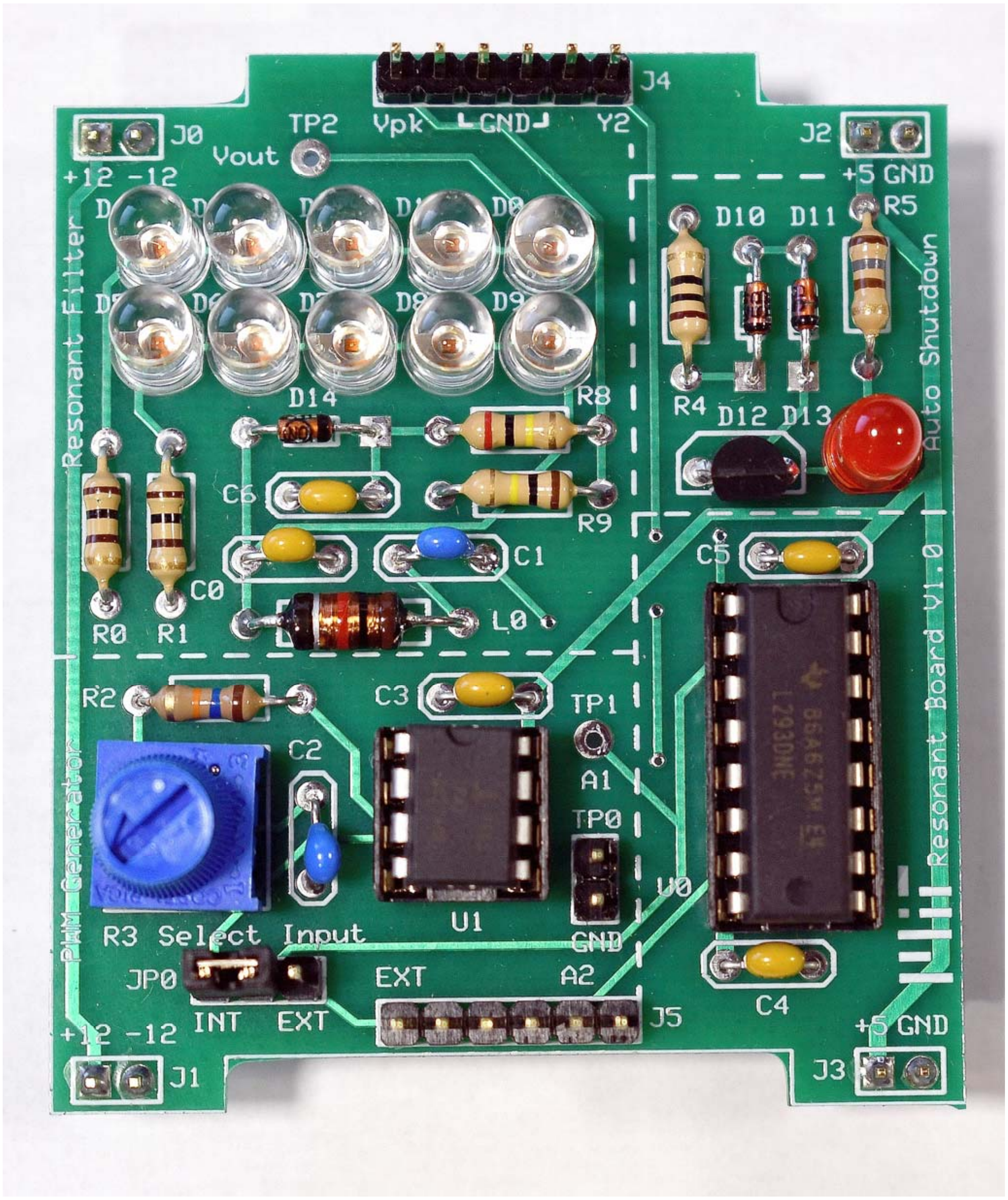
The key to understanding the ballast circuit is to understand how the IR2104 works. A timing diagram for the IR2104 is shown below:



On the 6.115 lecture ballast circuit board, the shutdown pin, #SD on the IR2104, is wired high, i.e., the chip is always enabled. When the IN signal is high, the HO signal is also high, activating the top MOSFET in the ballast circuit. This drives the L-C-Lamp circuit with a high voltage, Vcc. When the IN signal is low, the bottom MOSFET in the ballast circuit is activated, grounding the input of the L-C-Lamp circuit. In short, the entire circuit can be thought of as a square wave voltage source oscillating between levels of zero volts and Vcc volts. The DC blocking capacitor Cblk has a large value compared to C. The capacitor Cblk looks like a very low impedance at typical resonant frequencies for the ballast circuit (20-40 kHz). However, the DC component of the square wave produced by the IR2104 and MOSFETs appears across Cblk. The remainder of the resonant circuit, i.e., the L-C-Lamp combination, is effectively driven by a square wave of voltage oscillating between Vcc/2 and –Vcc/2. You control the frequency of this oscillation by varying the frequency of the TTL square wave that you apply to the IN pin on the signal connector – a perfect application for an inexpensive microcontroller. We will look at this in lecture.

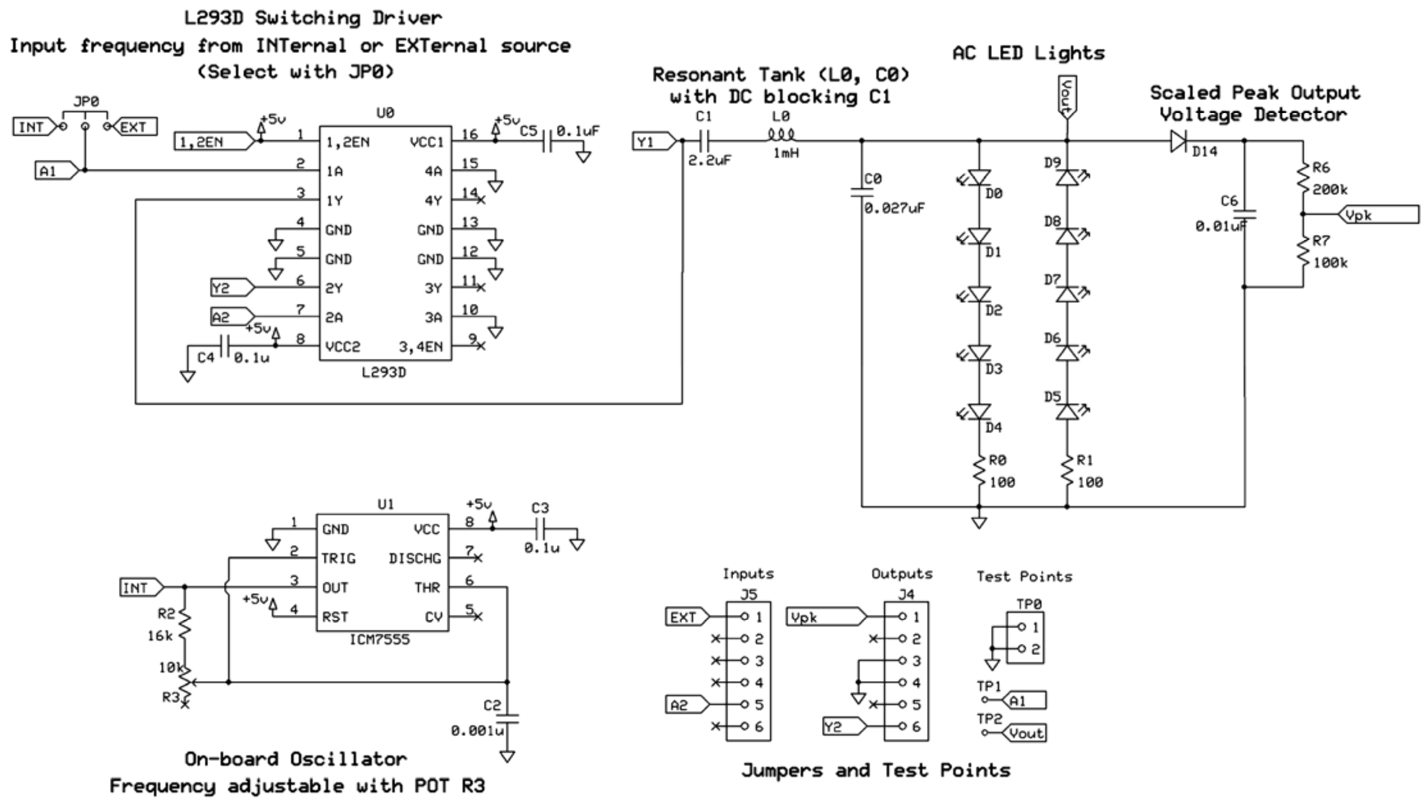
The “ballast” circuit used to control a fluorescent lamp is very similar to a circuit that we can use as a “driver” to control an LED lamp! A useful LED lamp can be made to run at lower voltages than the voltages necessary to “strike” a fluorescent lamp. So, we will work with an LED driver circuit in this lab to explore embedded control for lighting. But, recognize that the microcontroller’s job for the LED lamp that we will work with in your kit is very similar to the microcontroller’s job in a fluorescent lamp ballast.

A picture of the LED driver/lamp printed circuit board (PCB), available in your kit, is shown on the next page:



Be sure that you find the correct LED driver board in your kit! Notice the silkscreen labels on the board, e.g., “Resonant Board V1.0”. ASK questions if you are confused! Do not damage the board!

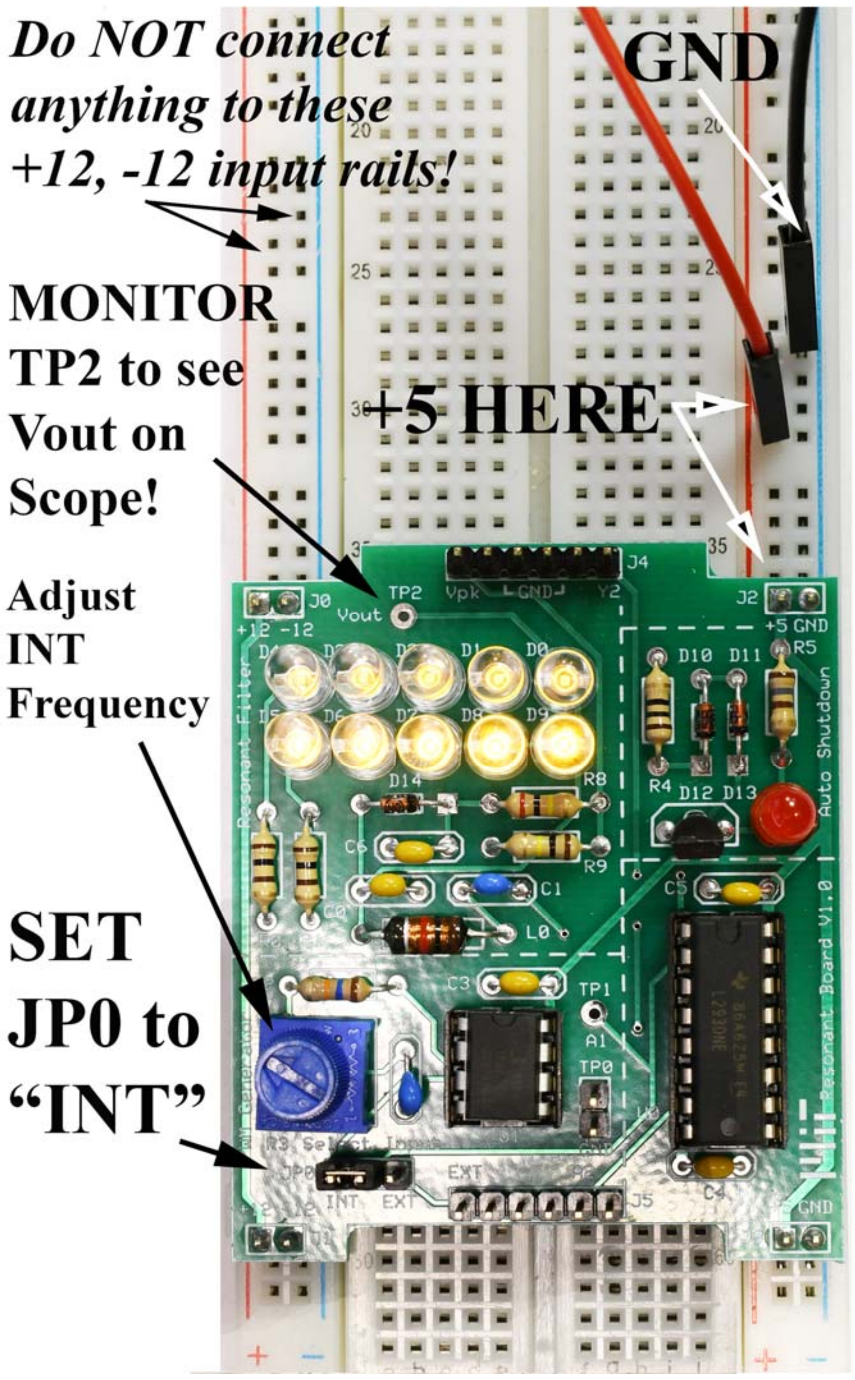
A simplified circuit schematic of the circuitry on this LED driver PCB is shown below. The details of an overvoltage protection circuit (in the area labeled “Auto Shutdown” on the PCB silkscreen) are not included in the schematic below, as we should not be activating this protection circuit.



Note how similar this circuit is to the fluorescent lamp driver! The LED driver again includes a resonant circuit formed from inductor L0 and capacitors C1 and C0. Capacitor C1 is a “DC blocking capacitor,” a relatively large capacitor compared to C0, tasked with absorbing the 2.5 volts of the DC square wave produced by the L293D and “passing through” only an AC square wave to L0 and C0. The inductor L0 and capacitor C0 resonate at a frequency substantially above DC. The resonant circuit is driven at the point labeled “Y1” by another high and low stack of switches (like the MOSFETS in the fluorescent ballast) provided by the L293D integrated circuit. The LED driver only uses one of the four switch stacks available on the L293D. An input square wave to the L293D controls the switches much in the same way as the IR2104. A TTL command or input square wave is applied to the “1A” pin of the L293D, and the “power level” output square wave, which oscillates between +5 volts and ground, is made available at the output of the L293D on the “1Y” pin. Notice that the input square wave can come from either one of two sources. If the JP0 jumper (located bottom left on the PCB) is connected to INT (internal), then the square wave is provided by an on-board, adjustable frequency square wave generator made with an ICM7555 timer IC. On the other hand, if the JP0 jumper is moved to the EXT (external) position, then a square wave applied to pin 1 of the J5 jumper on the PCB can be used to drive the L293D. We will use the R31JP to make square waves to apply to the EXT jumper pin on J5 to run the LED lamp. Notice how the ten total LEDs on the PCB are organized as two strings of five LEDs. The strings are oriented for opposing polarity. When the resonant circuit is resonating, it makes a sine wave of output voltage at Vout, available at a test point on the PCB. When the sine wave is positive, one of the strings of five LEDs will light. When the sine wave is negative, the other string will light. If the resonant frequency is relatively high (above 30 Hz or so), you will perceive the two strings as being “on” all the time, even though they are actually flickering at the resonant frequency. You adjust the brightness of the LEDs by adjusting the switch frequency of the INT or EXT source. For the INT source, you can move the frequency by playing with the R3 potentiometer connected to the ICM7555 timer.

The "Scaled Peak Output Voltage Detector" in the schematic indicates the peak voltage across the LEDs. The diode rectifies the sine wave, an C6, R6, and R7 provide a filter and divider that smoothes the rectified sine wave to a nearly DC level reduced or divided to "fit" between 0 and 5 volts. You can use the output voltage "Vpk" available on pin 1 of jumper J4 to monitor the peak voltage across the lamps, and therefore the lamp brightness.

BEFORE you connect anything to your R31JP, just get comfortable with the LED driver PCB. Use a scrap of breadboard from your box (NOT your Blackbird kit) to mount the LED driver PCB like this:


Do NOT connect anything to these +12, -12 input rails!
GND
MONITOR TP2 to see Vout on Scope!
+5 HERE
Adjust INT Frequency
SET JP0 to "INT"

BE CAREFUL! DO NOT DAMAGE THE PCB! Careless damage to the PCB will impact your check off. ASK QUESTIONS FIRST IF YOU HAVE THEM.

USE +5 and GND from your Blackbird kit, brought over as shown in the picture on red and black wires, to the breadboard scrap holding the LED PCB. THE PCB will ONLY mount on the breadboard scrap, it does not fit on the Blackbird breadboard!

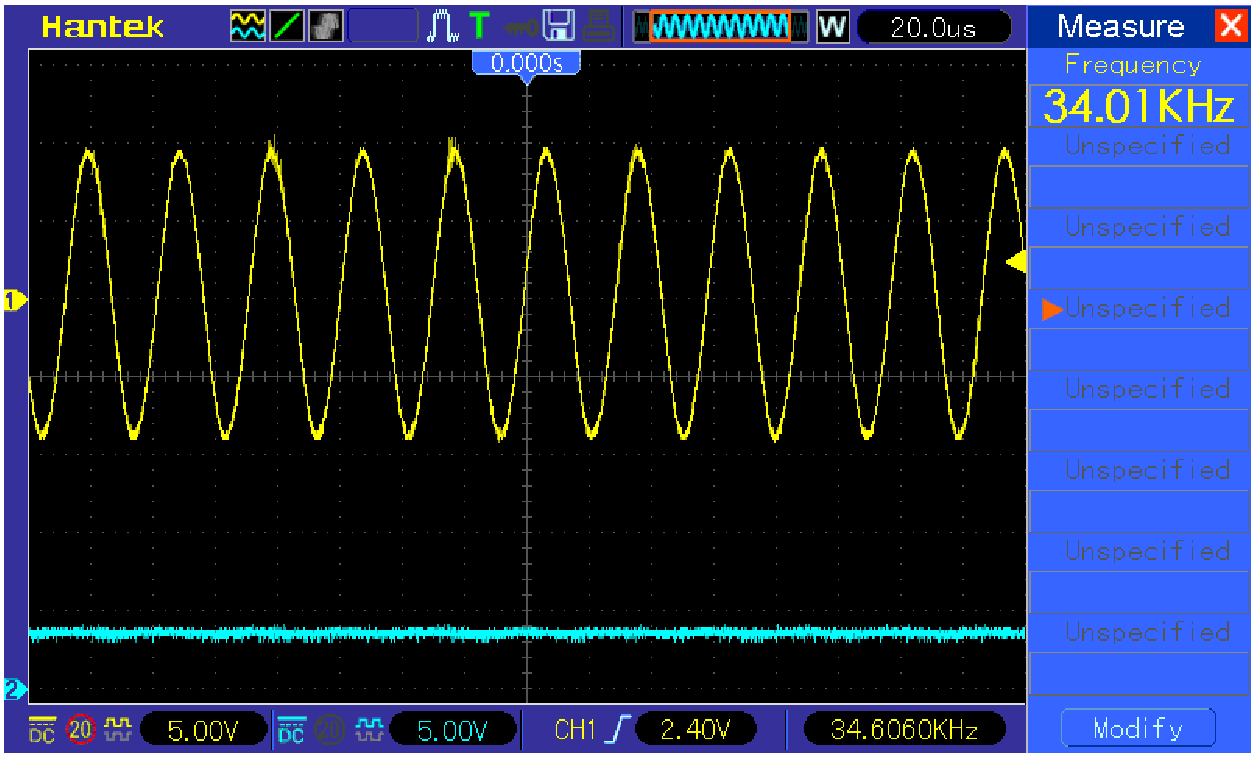
OBSERVE polarity CAREFULLY.
DO NOT connect power in reverse or you will destroy your PCB. DO NOT connect anything to the +12 and -12 rails on the PCB, they are not used.

SET JP0 to INT to get familiar with how the board works.

USE your oscilloscope to monitor TP2 (you can use a pin jumper in the TP2 hole to make a connection) so that you can see the output sine wave across the lights. (See Exercise 7 for a picture of this TP2 connection to a scope probe.)

You can also use a second scope probe to monitor the "Vpk" pin on J4 to see the "scaled peak voltage measurement" across the LED strings. (See Exercise 7 for a picture of J4 with a scope probe connected.)

Here's a picture from the staff oscilloscope while testing the LED PCB with five volts input and the INT setting on JP0. The top trace on channel 1 shows the "Vout" signal at TP2. The bottom trace on channel 2 shows the "Vpk" trace on J4:



Please do and answer the following:

- Connect your LED PCB on a breadboard scrap as shown above. Using the INT setting on JP0 and your oscilloscope, study the behavior of the circuit carefully. As you turn the blue POT for adjusting the ICM7555 drive frequency, what happens to the LED brightness? At what frequency are the LEDs brightest? What would you predict the peak brightness frequency to be analytically? Do the experiment and the prediction agree?
- Map the LED behavior as a function of frequency as you turn the POT. Make a table that includes ten levels of Vpk that correspond to ten different LED brightness levels. The lowest brightness level should be basically "off". At what frequency does this occur? The highest level should be the "brightest" you can get. At what frequency does this occur? Then, map eight other brightness levels and their associated frequencies, ranging smoothly from "off" to "brightest", for a total of ten entries in your table. Each table row should include frequency, Vpk, and qualitative brightness level as you assess it. The table has 10 rows. We will refer to it as the "brightness-frequency table".

EXERCISE 5: Make some square waves

Ultimately, we want the R31JP board to select the operating frequency of the LED driver. We will want at least ten different frequencies from the R31JP so that we can "sweep" through or "select" a value from your brightness-frequency table that you made in the previous exercise. For this exercise, we'll examine different ways to make square waves with the microcontroller. Use a Port 1 pin as your square wave output. You should not remove your 74C922 and keypad, you'll need it later. Just be sure to deactivate the 74C922 (put it in tristate mode) so that the chip is not attempting to drive a Port 1 pin that you are simultaneously attempting to drive with the microcontroller. We will NOT use the LED driver board in this exercise. Just work with the R31JP to practice making square waves in Exercise 5.

Please do and answer the following:

- Write a program that reads a byte from external memory. Count down from this number to zero. Every time you reach zero, toggle the state of the Port 1 pin, then begin counting from the number to zero again. In this way, you should be able to make a program that causes the microcontroller to provide a variable frequency square wave output. You should not have to recompile the program to change the frequency, i.e., the frequency should be controllable from MINMON. What is the highest frequency you can achieve? The lowest? What frequency resolution can you achieve, i.e, how finely can you adjust the frequency?
- Repeat this exercise, but this time use a two-byte count instead of a single byte. You will be graded for elegance, i.e., use the smallest amount of code and register space you can to do this. How does the 16-bit count effect your highest achievable frequency? Lowest? Resolution? Again, make changes possible using MINMON so that the code does not have to be reassembled to change frequencies.
- Write another program that generates variable frequency square waves. However, instead of using a count down loop, this time use the microcontroller's internal timer 0 configured in Mode 2, the auto re-load mode. Use a software timer interrupt to change the state of the Port 1 pin. Keep your code, particularly the interrupt service routine, as lean as possible to ensure fast execution. You will need to use the MON/RUN switch on your R31JP board to execute this program because interrupts are involved; that is, you cannot use the MINMON "G" command to run the program. Explain why. Nevertheless, if you are clever, you can make the frequency controllable by writing to a memory location using the MINMON "W" command before flipping the MON/RUN switch to RUN. This will avoid excessive assembling. What is the fastest frequency you can achieve? Slowest? Resolution?
- For the three pieces of code you just wrote (byte count, word count, and interrupt driven count), carefully analyze the timing of your code (number of machine cycles per instruction) and explain your observed highest and lowest frequencies and frequency resolution in terms of your code analysis.

EXERCISE 6: Understand the 8254 Counter/Timer chip

In this exercise we'll hook up a "peripheral" to the R31JP, the 8254 counter/timer chip. This chip will let us make square waves with very fine frequency control. Technically, you have already hooked up a peripheral to your R31JP, the 74C922 keypad controller we used in Lab 1. However, we connected the 74C922 to Port 1 with an "I/O mapped" connection scheme that uses a dedicated processor port. The 8051 microcontroller does not provide convenient support for simultaneously wiring lots of I/O mapped peripherals directly to the microcontroller through ports. So, in this exercise, we will explore a "memory mapped" peripheral connection. The 8254 will be addressed in the same manner as a memory location in RAM.

The 8254 is one chip in Intel's "8000-series" peripherals. The 8000 family of peripherals provides many different functions, and we will take advantage of some of them this term. For example, there are 8000-series chips that provide 8-bit parallel output ports like Port 1 on your R31JP (the 8255), serial ports, extra memory, or combinations of these. The 8254 provides a bank of three counter/timers that are similar in function to the two timers on the 8052 microcontroller in your R31JP board. Read the 8254 specification sheet. The 8254 is typical of a design philosophy that reappears throughout the 8000-

peripherals. If you master the hook-up and programming of the 8254, you'll find using the rest of the 8000-series relatively straightforward.

The 8254 contains four internal byte-wide registers that can be individually selected through two address lines on the chip, labeled A0 and A1 in the spec sheet. The 8254 can be configured to provide a number of different counting and timing operations. You can select the 8254 configuration by programming an 8-bit internal command register. The other three registers in the 8254 provide count or timing information for each of the three counter/timers in the 8254. Wire your 8254 into the memory address "hole" that can be addressed using the XIOSELECT line coming from the R31JP. Connect address lines A0 and A1 from the R31JP board to the A0 and A1 lines on the 8254. Use the XIOSELECT line from the Xilinx PLD on the R31JP to drive the chip select line on the 8254. Select appropriate lines from the R31JP to connect to the RD and WR pins on the 8254. For this laboratory, you will use Mode 3 of the 8254 to make a square wave at the frequency of your choice. Notice that you will need to hook a TTL crystal clock to one of the CLK pins on the 8254.

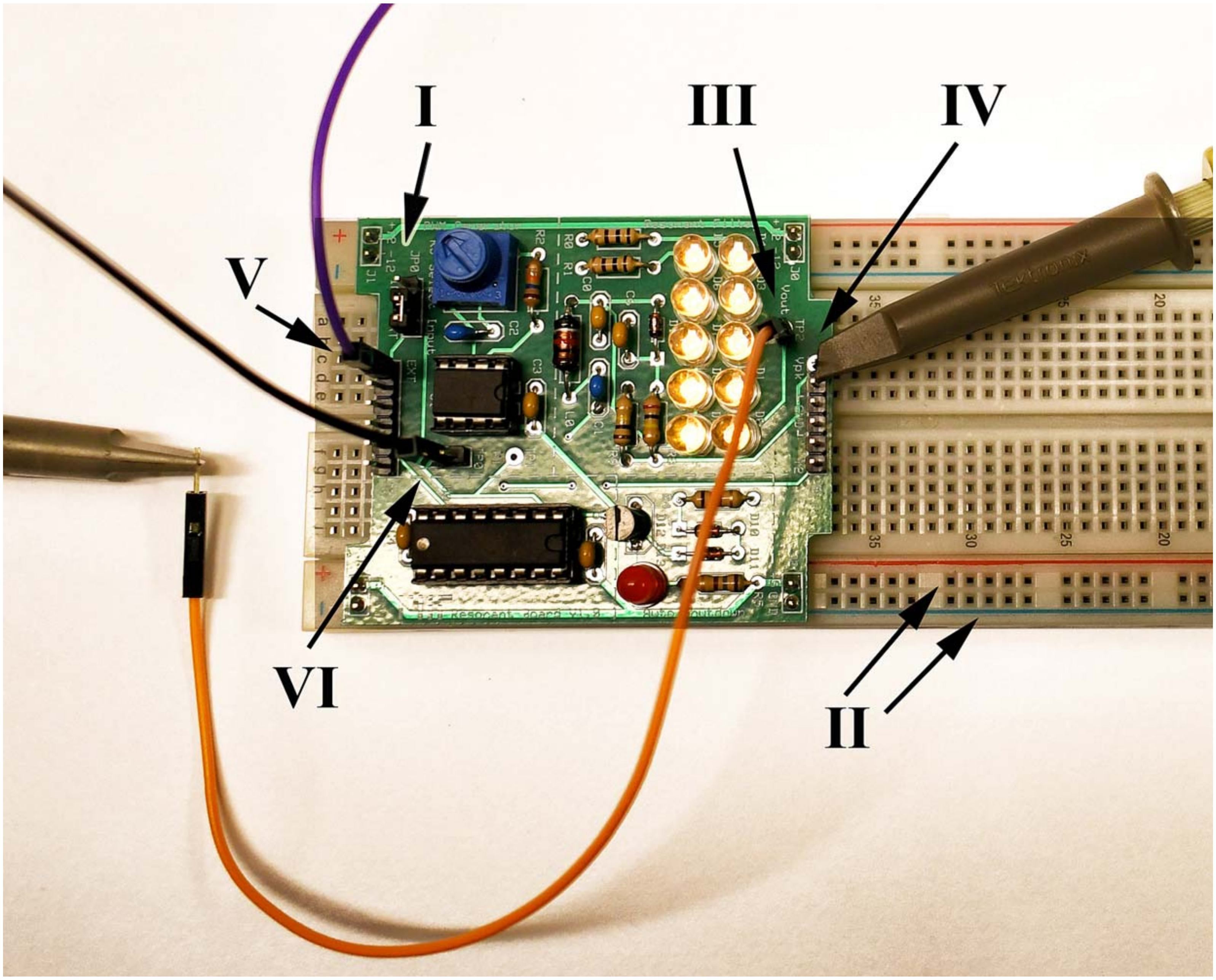
Do NOT use the LED driver board in this exercise. Just work with the R31JP and 8254 to make square waves.

Please do and answer the following:

- The XIOSELECT pin is active (low) for access to memory addresses in the range FE00h through FEFFh. How many different address locations in this range select the control register on your 8254 chip? What 8254 register is located at FE00h? What 8254 registers are located at FE01h, FE02h, and FE03h, respectively?
- For this question, imagine that we could drive the CLK0 pin either from a 1 MHz crystal or a 10 MHz crystal clock. Which choice gives the lowest possible 8254 output frequency on OUT0 in Mode 3? Which choice gives the highest possible output frequency? Which crystal provides the best output resolution around 30 kHz (where our LED light might operate)? That is, suppose the 8254 was operating in Mode 3 and producing an output square wave close to 30 kHz. Then suppose we change the "count" loaded into the timer register by adding or subtracting 1 from the count. Which input crystal, 1 MHz or 10 MHz, would provide the smallest change in output frequency?
- For this part, connect the 8254 CLK0 pin to a 10 MHz TTL crystal oscillator. Use your MINMON read and write commands to configure the 8254 for Mode 3 operation. Produce a square wave as close to the peak brightness frequency of your LED lamp as you can.
- Find the 8254 settings necessary to make each of the 10 frequencies in your brightness-frequency table. Expand the brightness-frequency table to include this valuable information for programming the 8254.

EXERCISE 7: Control the LED lamp

Now we will develop a control program that allows the microcontroller to automatically set and control the brightness of the LED lamp. Let's FIRST carefully prepare the LED PCB for connection to the R31JP. Do NOT connect the R31JP yet. Here is a picture of the staff LED PCB board ready for measurements and connection to an EXTernal square wave source (our R31JP), shown on the next page:



NOTE (following the Roman numerals in the picture – ALL connections to be made with power OFF):

I. With the power OFF, please carefully move the JP0 jumper to the EXT setting as shown.

II. CONTINUE to use power connections (not shown here) from your Blackbird KIT for +5 and GND. Same as in Exercise 4: GND goes to the OUTER rail as indicated. The +5 connection from the KIT goes to the INNER power rail as indicated. These connections are IMPORTANT in order to make sure that the LED PCB and the KIT and the R31JP (connected to the KIT) all share a common ground connection. You will destroy something if you do NOT have a common ground connection between the kit systems. Bypass your breadboard rails at II with capacitors (not shown in the picture above).

III. As before, you can measure the Vout output voltage across the LED strings at TP2. In the picture, we have used a pin-to-pin jumper to bring the TP2 signal over to a scope probe on the far left of the picture.

IV. Again as before, you can measure Vpk from the scaled peak detector circuit on the J4 jumper, shown with a directly connected scope probe on the right in the figure.

V. INJECT your EXTernal square wave from the 8254 (when you have one) on the EXT pin on J5. Use a socket-to-pin wire to make this connection to J5 and back to your Briefcase KIT. You can chain together socket-to-pin cables as needed to reach your KIT.

VI. AGAIN: Be sure to ground your scope, KIT, and LED PCB to the same ground connection. You should be able to make all needed ground connections along the OUTER ground rail indicated by II. But, FYI, you can also use a socket-to-pin cable to connect to the LED PCB ground at TP0, as indicated by the VI arrow in the picture.

IF you are confused, then STOP. ASK for help before you damage anything.

Please do and answer the following:

- Use your 8254 to create the EXTernal square wave for the LED PCB. If you have made the connections indicated above, it should be easy to test your work. Use your MINMON "W" command to configure the 8254 to make a square wave corresponding to the "resonance" or maximum brightness of your LED PCB. With power on, you should see the LED's glow brightly. You can of course try other frequencies using MINMON and verify that the LED brightness changes appropriately.
- Now, let's make a lighting controller. This is a common, lucrative commercial product. You can see some examples here:

  https://www.lutron.com/en-US/Pages/Dimmers/Dimmers.aspx

  Here's the plan: write an R31JP program in assembly that uses your keypad and 74C922 (still available on Port 1) to control the light. When you press one of the "number" keys 0-9, the LEDs should go to one of 10 brightness levels. The "0" key should set the lowest level (essentially off), and the "9" key should set the brightest level. The intermediate keys should set monotonically increasing brightness levels ranging from the "0" key to the "9" key.
- Now expand your program to add some of the other "professional features" you would find on something like the Lutron LED+ family of dimmers. When you press "A" on your keypad, your lights should ramp smoothly from "off" to "full bright" over a time period of about 10 seconds, and then remain on at full brightness. With the lights at "full brightness," when you press "B" on the keypad, your lights should ramp smoothly to "off" over a time period of about 10 seconds. Finally, again with the lights at "full brightness", when you press "C" on the keypad, your R31JP should keep track of how long the "C" key is held down, ranging between 0 to 10 seconds as decided by the user. Then, when the "C" key is released, the LEDs should remain "on" at full brightness over approximately the time period that the "C" key was held down, and then finally ramp to "off" over another period of about 10 seconds. These functions provide useful features, e.g., for "B" and "C," leaving you with some lighting as you walk out of a room, but then conserving energy by ramping the lights to off when the room is presumably empty. Or, with "A," providing a gradual increase in lighting as you enter to avoid a sudden blast of light to your eyes. Dimmers that provide these features are highly prized and command premium retail prices.

.

EXERCISE 8: Make sure you understand the 8051 hardware

As you know, we have provided you with a microcontroller board, the R31JP, which eliminates the tedium of wiring up six or seven chips to make a minimum system before you can play. However, it's important that you understand the hardware and the general features and details of microcontroller hardware. When you use microcontrollers to design a product, it's very likely that you will have to construct your own minimum system. In lecture, we'll look not only at the R31JP but also another single board computer (SBC) based on the 8051 microcontroller shown on the last two pages of this lab. One page shows the "processor core", the other shows the "memory subsytem" connected to the address bus and data bus provided by the processor core. Use these schematics to test your understanding of the 8051 architecture. Please look at the system shown on the last two pages, and answer the following questions in your lab report:

- How long (in seconds) is one machine cycle on this SBC: ______________________________

- The memory subsystem contains 3 memory chips: Two ROMS labeled C1 and C3 in the schematic, and one RAM labeled D3 in the schematic. Which chip would most profitably contain MINMON (circle your answer):

  C1 C3 D3 None of these

- Given the SBC as connected in the attached schematics, after a power-on reset, the first code-store fetch will be from (circle all which apply):

  Internal 8051 Rom C1 C3 D3 None of these

- The MINMON command "G8000" could be used to execute a program or routine on this SBC given only the chips and interconnect shown on the attached schematics:

  TRUE FALSE

- The MINMON command "G2000" would execute a program stored in (circle all which apply):

  Internal 8051 Rom C1 C3 D3 None of these

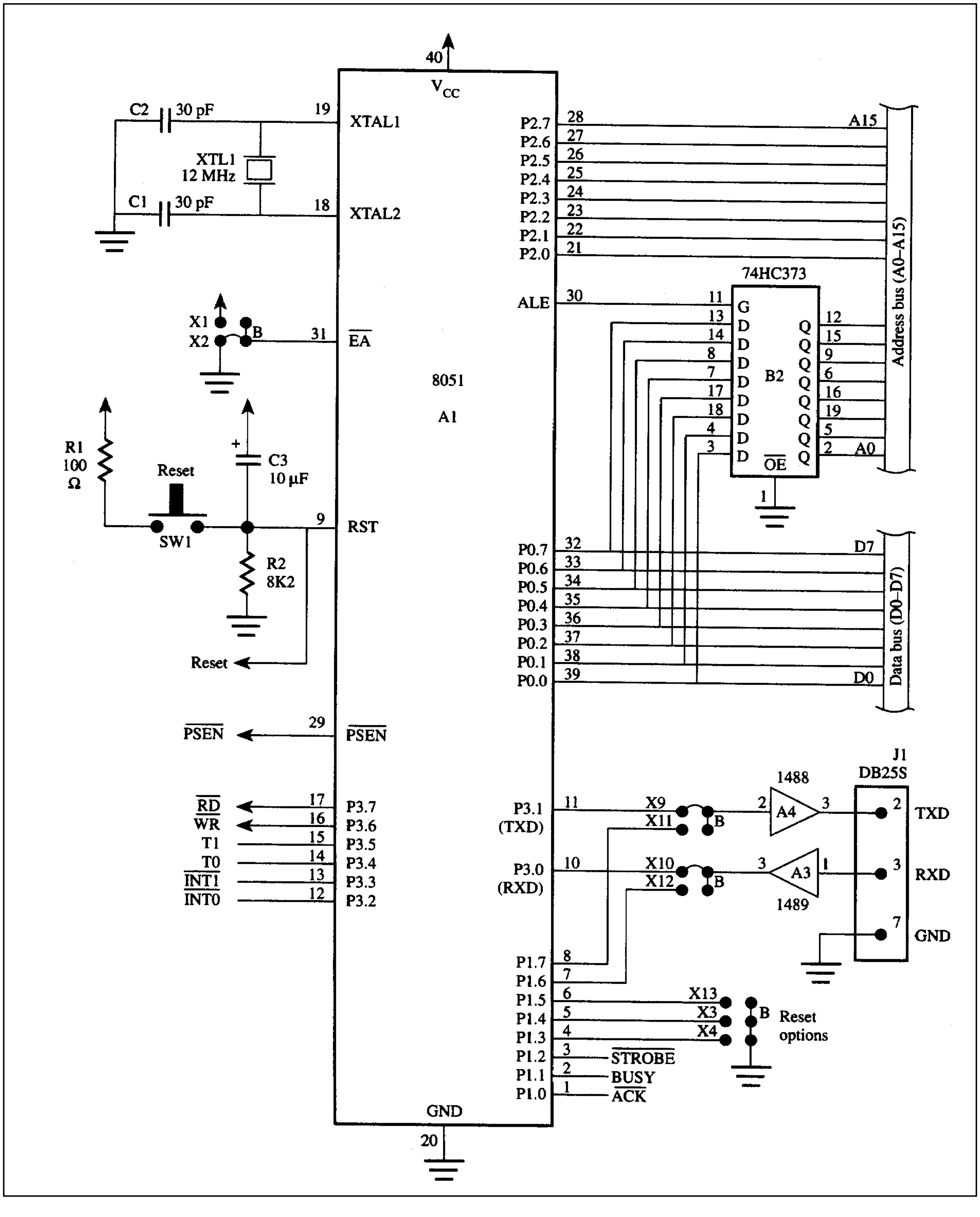
- Given the SBC as connected in the attached schematics, the 8051 instruction "MOVC A, @A+DPTR" could be used to read data bytes stored in (circle all which apply):

  Internal 8051 Rom C1 C3 D3 None of these

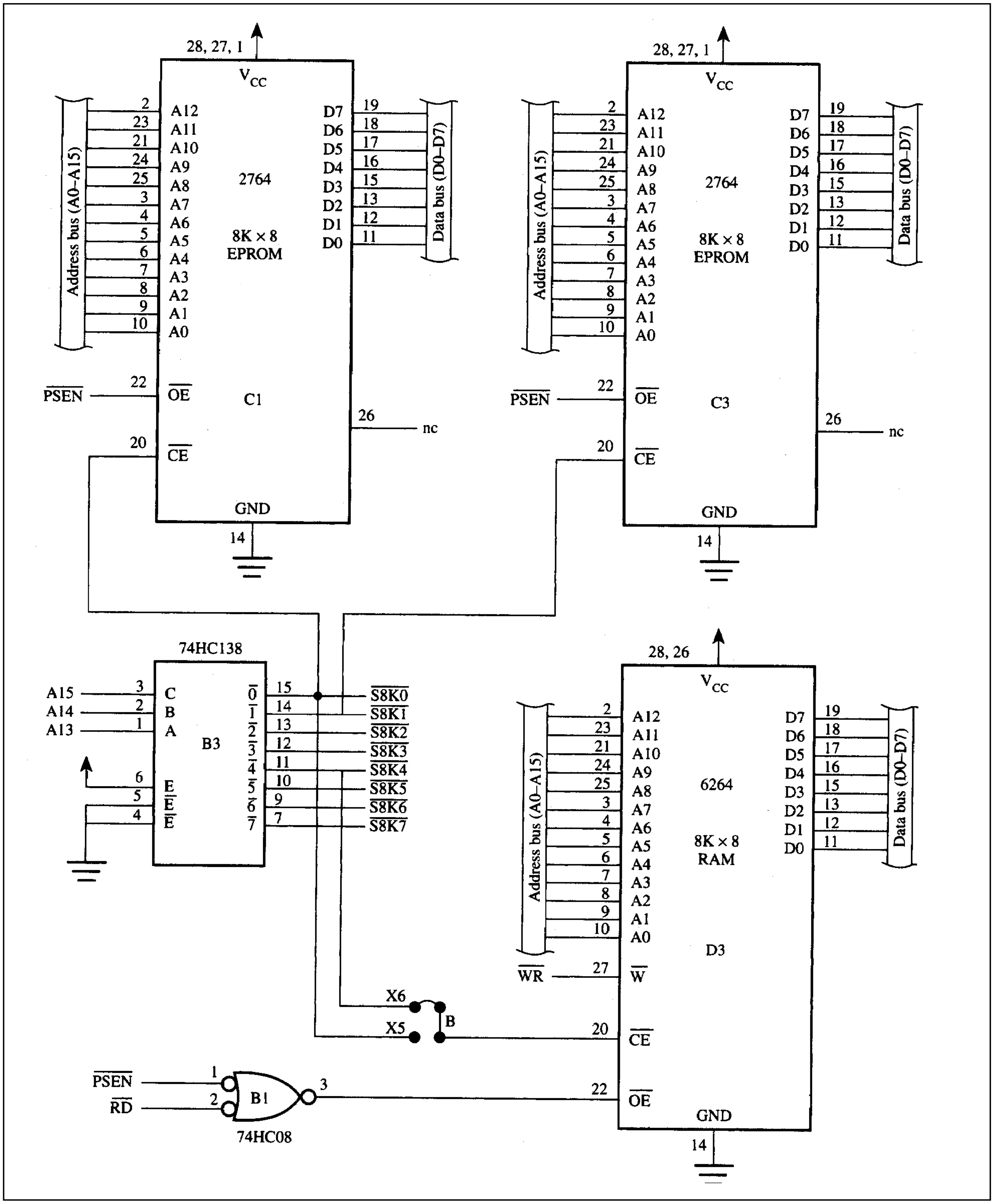
- Given the SBC as connected in the attached schematics, the 8051 instruction "MOVX A, @DPTR" could be used to read data bytes stored in (circle all which apply):

  Internal 8051 Rom C1 C3 D3 None of these

Single Board Computer: Processor Core



Single Board Computer: Memory Subsystem



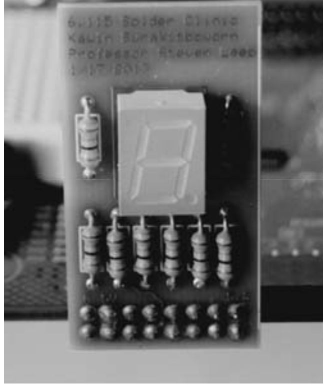
EXERCISE 9: Learn about the Cypress PSoC Big Board

In the previous lab, you started using the PSoC STICK, primarily focusing on the hardware. In this exercise, we will take a look at the PSoC Big Board and see how a PSoC is programmed in the C language with Creator. In the last decade, a new concept called "System on Chip" has begun to alter the way designers think about using microcontrollers. Cypress Semiconductor has created PSoC, a family of microcontrollers with an amazing set of hardware and software features on a single chip at relatively low prices (a few dollars per chip). In 6.115, you will have the chance to play with PSoC. On a single IC, a PSoC family microcontroller offers flash program memory, data memory, flexible digital blocks that can be used to implement all sorts of digital functions including serial ports, parallel ports, analog-to-digital converters, digital-to-analog converters, I2C, USB connectivity, and, amazingly, flexible op-amp blocks for handling analog signals as well. You decide how you want to use the hardware, and having a PSoC IC is like getting to pick from an electronics parts catalog with instant delivery. The PSoC family includes several different types of CPU cores, including a line of PSoCs with 8051 cores, and also a line of PSoCs with ARM cores (commonly used in cell-phones, for example).

We will play with the PSoC to learn about single-chip microcontrollers and also to experience "high-level" programming in a language called "C". Cypress provides a C compiler for use with PSoC in its "PSoC Creator" software. You installed your Creator software in previous labs. This software is free, and you are welcome to install copies on your home computer.

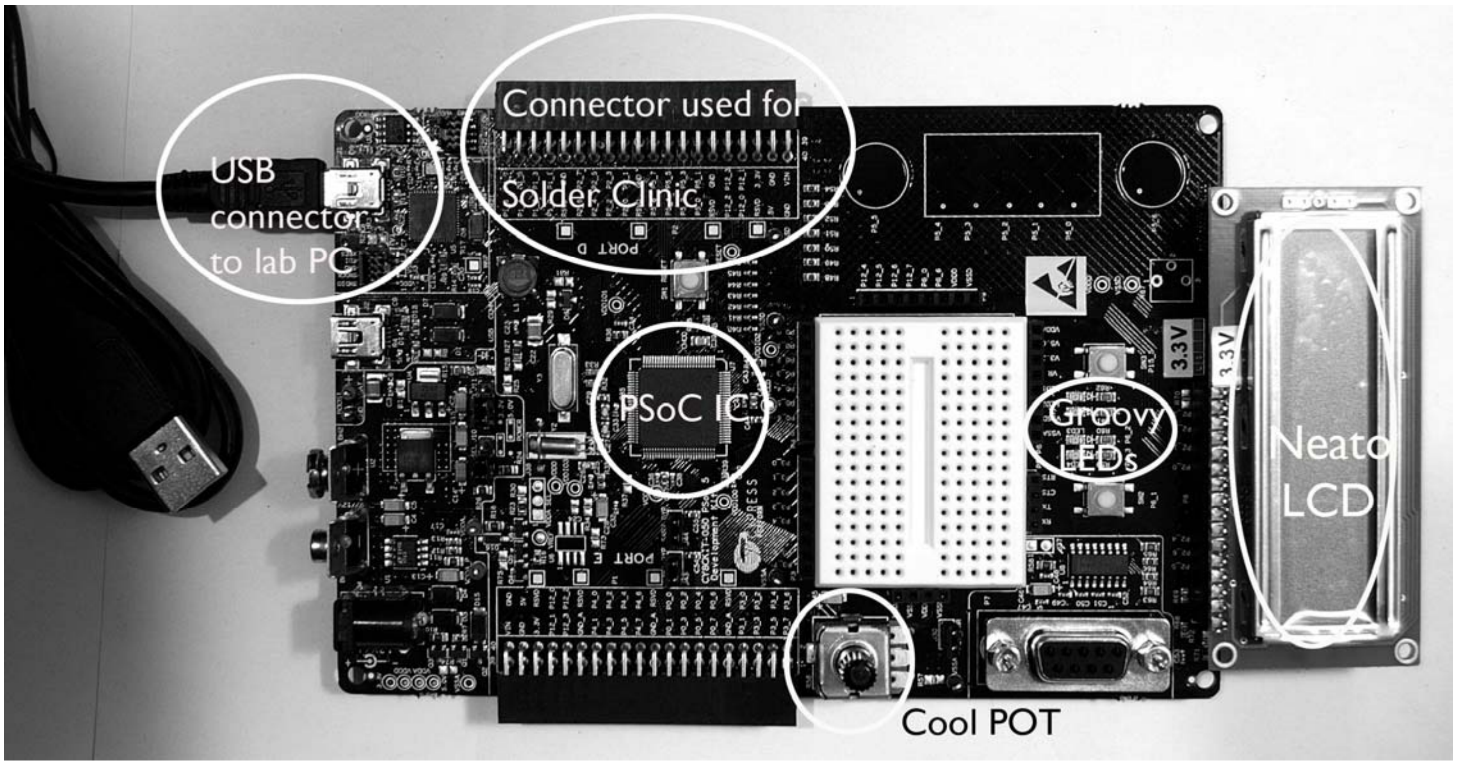
REMEMBER: The PSoC board is expensive, and we will NOT be able to replace it if you blow it up. Do not be careless with the board. There is no reason to have an accident with it, particularly during the laboratory exercises, if you are careful. ASK if you have questions before making a mistake with the board. Some tips to note:

1. ONLY power the board with the USB connection to the computer. This connection also allows you to program the board using Creator. ONLY use the top, left-side USB connector shown in the picture below.
2. Do NOT connect anything to the board other than the USB connector and the DIGIT LED peripheral board we will discuss below. Ask the staff if you have any confusion about how to connect the LED peripheral board for the following lab exercises. Do NOT connect the PSoC board to your Briefcase KIT, and do not use signals from the board to drive other parts, e.g., on your KIT.
3. These are precautions. We have found that a common reason why people destroy these expensive boards is that they fail to connect grounds between the PSoC evaluation board and other IC's on a kit or elsewhere. Also, the boards have been destroyed through the application of excessive voltage. For now, all of these and many other problems can be avoided by using the evaluation board "as is," with only the USB connector and the one addition of the 6.115 DIGIT LED peripheral board. Please be careful.
4. Note that, when you open an older PSoC project, you may be asked to "update components" by Creator. Please do so.
5. A picture of your DIGIT LED board is shown to the right. Please find this board in your kit. Your board may have minor cosmetic differences (on the silkscreen), but it should look essentially like the picture to the right. Normally, we would solder this board together at a "solder clinic" on campus. For "Kovid," we have soldered the board for you and enclosed it in your kit. But we will refer to the "solder clinic connector" below, as you will see in the pictures.

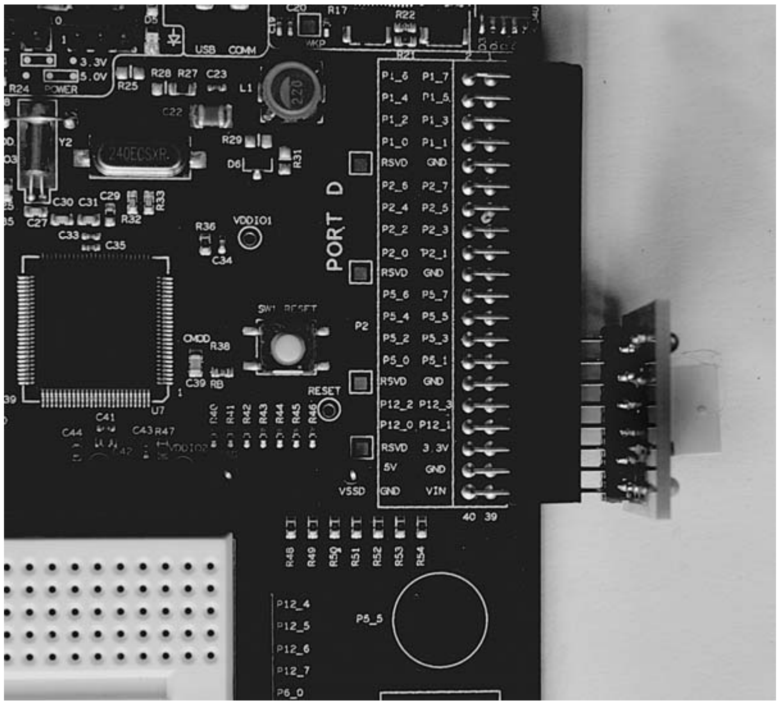


(We assume that there is no need to convince you of the importance of being able to use a digit LED display. Along with a blinking light, a digit display is one of the most important features for practically any project. Note, for example, that while the PSoC might be most suitable for a liquid-metal T1000 Terminator, even the T800 prefers digit displays, as shown in the HUD screen shot below. If you have **not** already won a gift certificate, be the first to e-mail Professor Leeb with the exact title of the immortal, must-see classic movie containing this scene and win a $10 Amazon certificate!)

A full picture of your PSoC evaluation Big Board is shown below, with area labels (the "lab PC" is your Windows 10 computer):



Here's what the Digit LED board looks like while it's being inserted into the PSoC evaluation board area labeled “Connector used for Solder Clinic.” NOTE that the Digit LED board has a double row of pins, which should be inserted into the double row of socket holes on the “Connector…”. Make sure that you study the pictures carefully, understand the orientations of the Big Board and Digit LED board, and ask questions as needed BEFORE you insert the Digit LED board. The picture below shows the Digit LED board in the middle of pushing it in for insertion. Make sure that your LED board is fully inserted before using the evaluation board.



The “7-segment” display on the Digit LED board is a convenient array of LEDs prepackaged to ease display applications. You can read more about this type of display here:

https://www.electronics-tutorials.ws/blog/7-segment-display-tutorial.html

Please do the following:

- Go to the "PSoC" page on the 6.115 course website: http://web.mit.edu/6.115/www/page/psoc-information.html
- Download the "Blinky" code for PSoC creator and make sure that you remember how to use Creator. Run Blinky so that you see the "6.115" message on your Digit LED display. In the Creator software, select the correct PSoC target device (CY8C5868AXI-LP035) using the "Device Selector…" under the "Project" menu tab. Use the "Build" menu tab to build Blinky, and use the "Debug" menu tab to program the PSoC evaluation board. Make sure you are comfortable "building" and "programming" the PSoC and that you can see the LED display working. In Creator, make sure that you can find and edit the files with ".cydwr", ".c", and ".cysch" for the project. These files are, respectively, the "design wide resources," the "C-language program code," and the internal "schematic" for the project.

- Now, from the 6.115 course website, download the "Exercise 1" for the PSoC (on the webpage right under the "Blinky" project). Complete Exercise 1 by following the instructions on the file page with a ".cysch" extension for Exercise 1. Build and program the project on your PSoC board. You should be rewarded with a regularly flashing light in your bank of "Groovy LEDs" on the evaluation board.
- Modify the c-code for the Exercise 1 project so that the light flashes a pattern corresponding to "SOS" in Morse Code: Three short flashes, followed by three long flashes, followed by three short flashes, followed by a 1 second pause. The SOS should repeat indefinitely. You may find the “CyDelay” command useful in your code.
- Save your modified project. Include a phone picture of the light flashing on, and also include your carefully commented, modified SOS C-code in your lab report.