Lab 2H. Hardware Interfacing of LED, Switch (Spring 2025)

ECE319H students do Lab 2H by themselves (no partner for Lab 2H)

Preparation

1. Review Program 1.8.1 in the textbook
2. Read all of Chapter 2 in the textbook or ebook, with a particular emphasis on the not gate, Example 2.3.2
3. Read switch Data Sheet: http://users.ece.utexas.edu/~valvano/Datasheets/B3F-1059.pdf
4. Read LED Data Sheet: http://users.ece.utexas.edu/~valvano/Datasheets/LEDHLMP-4700.pdf
5. The CCS the starter project for Lab 2H is in the original downloads

Purpose

The purpose of this lab is to learn how to interface a switch and an LED and to program the LED to send Morse Code letters when triggered. You will also perform explicit measurements on the circuits in order to verify they are operational and to improve your understanding of how they work.

System Requirements

The primary task of the lab is to send a single word in Morse code. The **time unit** for Lab 2H is 100ms

- A *dot* is the LED on for one time unit
- A *dash* is the LED on for three time units
- The *inter-element gap* between the dots and dashes within a character is the LED off for one unit
- The *short gap* (between letters) is the LED off for three time units (not needed for Lab 2H)
- The *medium gap* (between words) is the LED off for seven time units (not needed for Lab 2H)

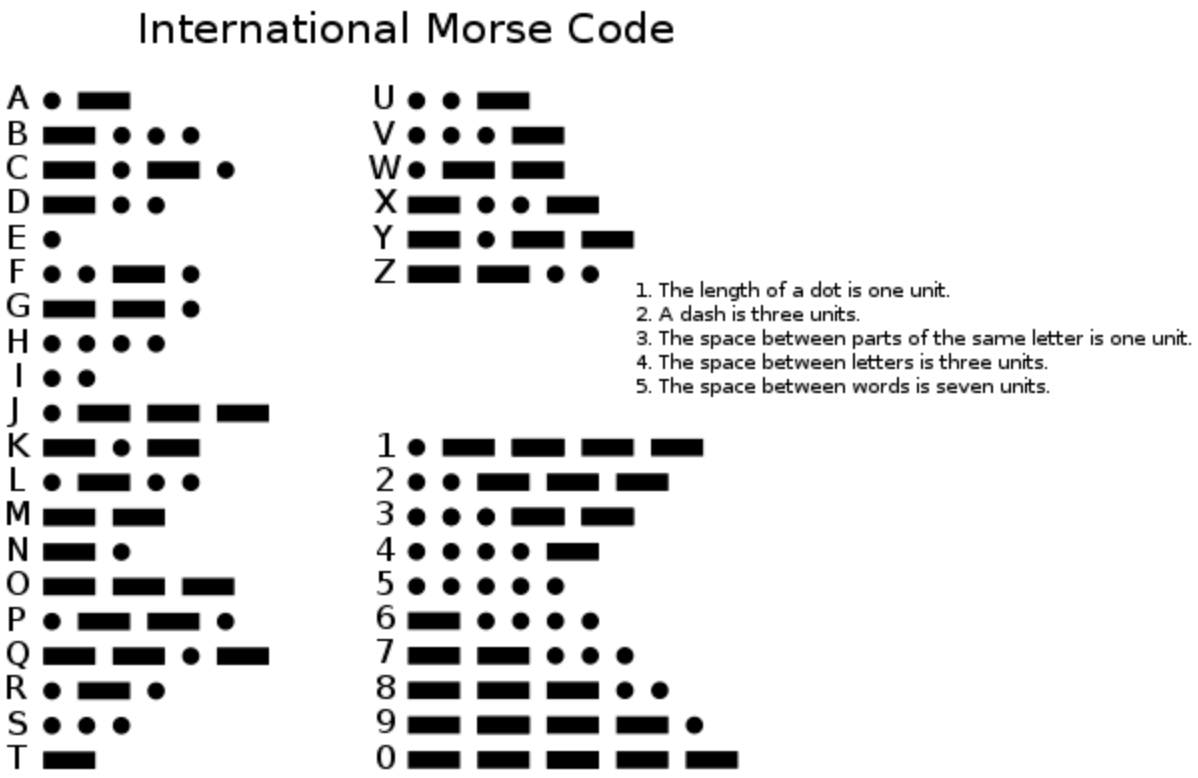


Figure 2.1. Morse Code https://en.wikipedia.org/wiki/Morse_code.

The external hardware for the lab includes one input and one output. Because you are interfacing the switch and LED externally you will need add resistors in the appropriate places. The switch and LED will be interfaced using positive logic. Students are randomly assigned which pin to use as input and which pin to use as output. Your Lab 2 function will be passed a variable length string containing one word in upper case, which you will transmit.

- The **Start** input switch on an input pin
- The **LED** output on an output pin

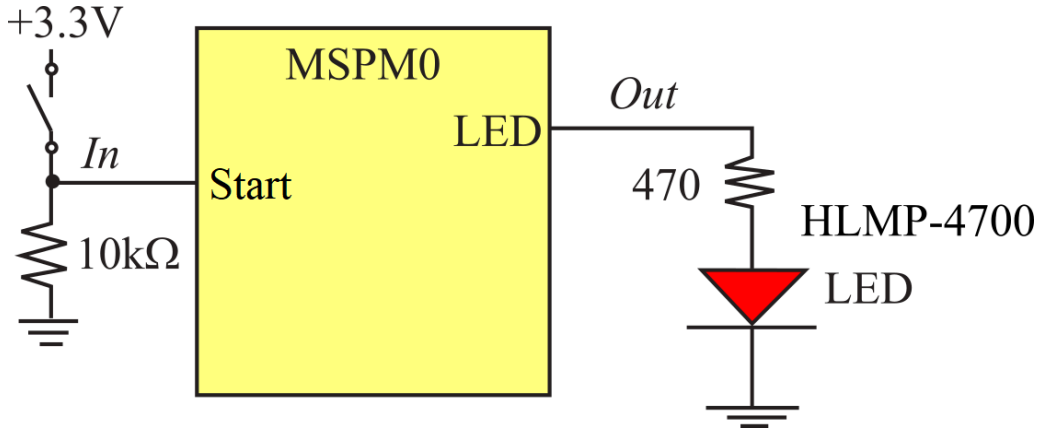


Figure 2.2. Block diagram of the Lab 2 system.

*The exact requirements for your lab will be revealed in the Terminal window when you enter your EID into the project and then build and call **Lab2Grader** with R0=0.*

If you do not see the Terminal window, execute **View->Terminal** and open a **Serial Terminal** with your COM port at 115200 baud rate. Overall functionality of this system is to repeat steps 1 - 4 over and over:

1. Your software will wait until the **Start** button is pressed
2. Call **Lab2Grader** with R0=0 for debug, R0=1 for logic analyzer, R0=2 for scope, or R0=10 for grading.
3. The grader will return in R0, a character from 'A' to 'Z' (upper case)
4. Your software will transmit the character.

Figure 2.3 Shows the desired LED output sending the word HELLO.

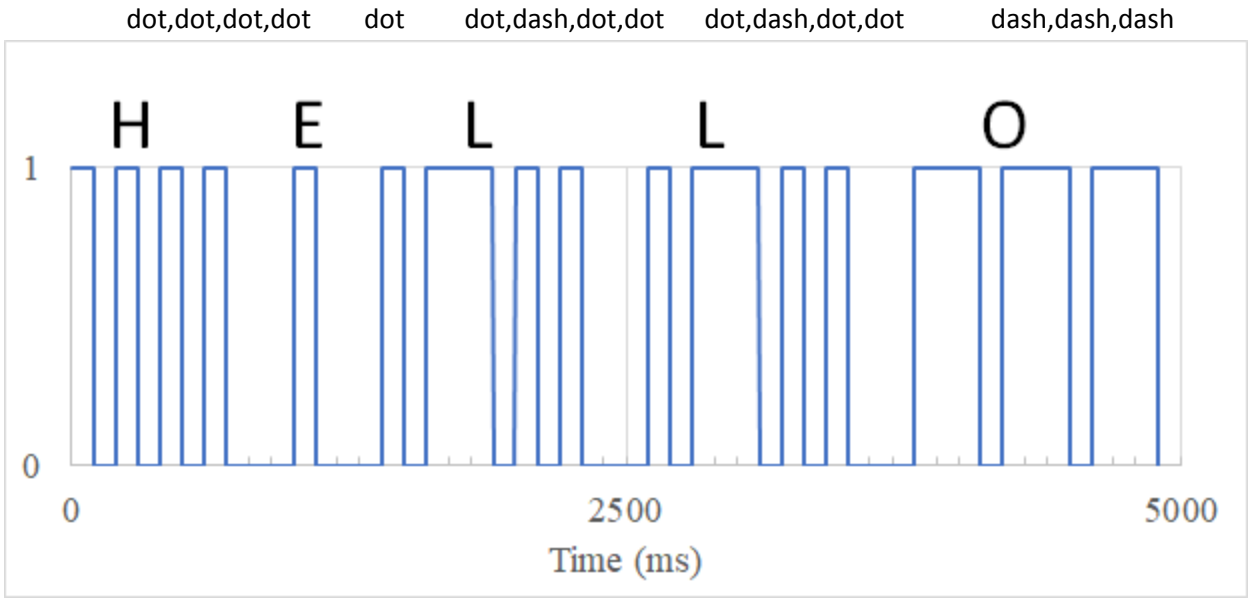


Figure 2.3. Theoretical LED output for the word HELLO in Morse Code.

Procedure

Part a - Determine the problem specifications

For all labs in ECE319K you will build and test your hardware for the real-board. Please enter your EID into the **ECE319K_Lab2.s** file as shown with the **red arrow** in Figure 2.3. The EID is used by a random number generator to select the pins you need to implement.

```
29          .text
30          .thumb
31          .align 2
32          .global EID
33  EID:    .string "ZZZ1234" // replace ZZZ123 with your EID here
```

Figure 2.3. Put your EID into your ECE319K_Lab2H.s file.

Initially, you should run **Lab2Grader** with R0=0 selected (**red arrow**), as shown in Figure 2.4.

```
54 //   configure interrupts  on TIMERG0 for grader or TIMERG1 for TExaS
55 //   initialize ADC0 PB20 for scope,
56 //   initialize UART0 for grader or TExaS
57      MOVS R0,#0  ; 0 for info, 1 debug with logic analyzer, 2 debug with scope, 10 for grade
58      BL   Lab2Grader
```

Figure 2.4. The function Lab2Grader takes an input parameter in R0.

Figure 2.4 shows a typical UART window when calling Lab2Grader with R0=0. The **red arrow** is your EID. The **purple arrow** is the pin specification for the **Start** switch. The **green arrow** is the pin specification for the LED output. With R0=0, the grader is not running so the Score will be 0. However, the

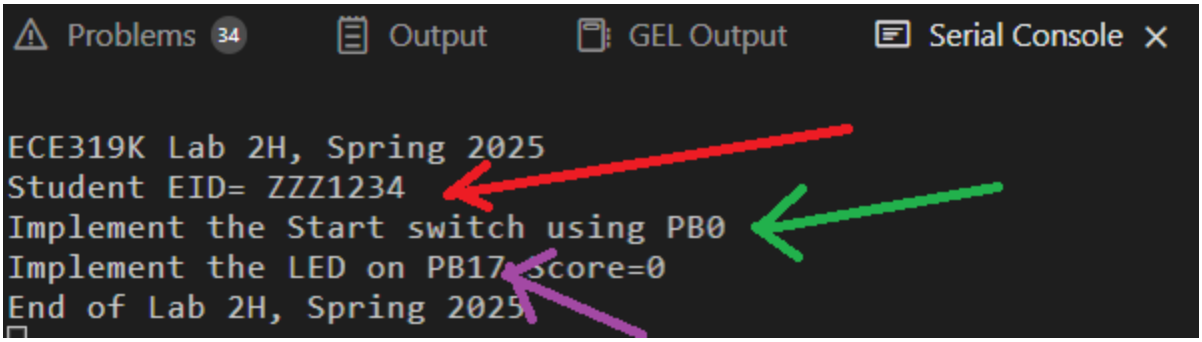


Figure 2.4. The Terminal window showing the problem specifications (please check to see if your EID is correct); it should show Lab2H, Spring 2025.

Part b - Read the data sheets and design the interfaces

Engineers must be able to read datasheets during the design, implementation and debugging phases of their projects. During the design phase, datasheets allow us to evaluate alternatives, selecting devices that balance cost, package size, power, and performance. Download the datasheet for the LEDs

http://users.ece.utexas.edu/~valvano/Datasheets/LEDHLMP-4700.pdf

An LED is a **diode** that emits light when an electric current passes through it, as shown in Figure 2.5. LEDs have polarity, meaning current must pass from anode to cathode to activate. Using the data sheet, hold an LED and identify which pin is the anode (+) and which is the cathode (-). Current flows from anode to cathode when the LED is on. The anode is labeled **a** or **+** , and cathode is labeled **k** or **-**. The cathode is the short lead and there may be a slight flat spot on the body of round LEDs. Thus, the anode is the longer lead. LEDs are not usually damaged by heat when soldering. Furthermore, LEDs will not be damaged if you plug it in backwards. However, LEDs won't work plugged in backwards.

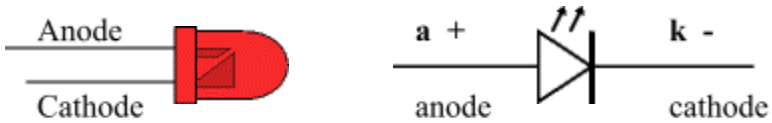


Figure 2.5. Left: a side view of an LED with leads labeled; Right: the corresponding circuit diagram

Because the LED requires less than 6 mA, we will not need a driver circuit like the ULN2003. Normally, we would pick a desired operating point (V_d, I_d), and find a resistor value to establish that point. However, because the ECE319K kit has six LEDs and six 470 ohm resistors, we will use 470 ohm resistors for the LEDs. Use Figure 2.6 to estimate the V_d, I_d resulting operating point for the red LED with the 470 ohm resistor. For now, assume output high voltage (V_{OH}) will be 3.2V. To simplify the math you can approximate the red LED curve as $I_d = 20*V_d-32$, where V_d is in volts and I_d is in mA for the values of V_d from 1.65 to 1.8V.

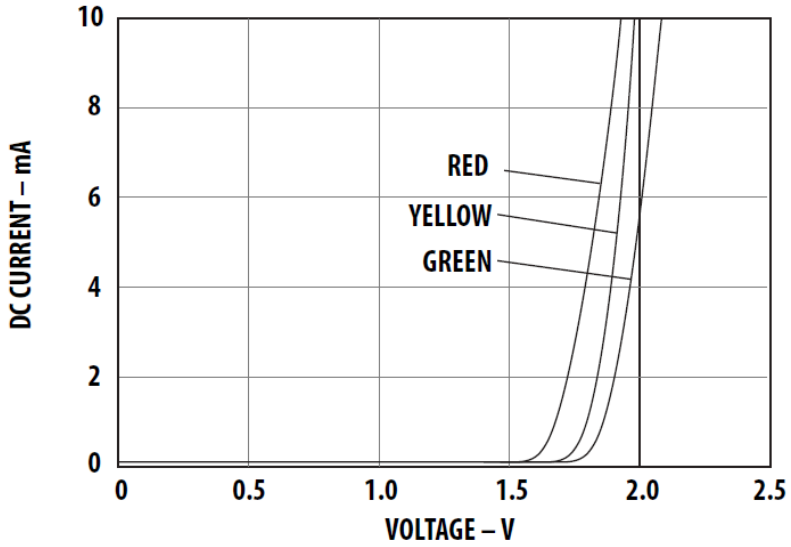


Figure 2.6. Find the operating point for this Red LED, the current will be around 3mA with 470 ohm resistor.

Sometimes we are asked to interface a device without a data sheet. Notice the switch has 4 pins in a rectangular shape, as shown in Figure 2.7. Each button is a single-pole single-throw normally-open switch. All four pins are connected to the switch. Using your ohmmeter, determine which pairs of pins are internally connected (having a very small resistance), and across which pair of pins is the switch itself. In particular, draw the internal connections of the switch, started in Figure 2.7, showing how the four pins are connected to the switch.

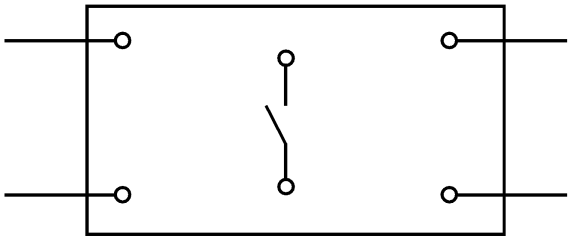


Figure 2.7. Connection diagram for the normally open switch.

Redraw the circuit in Figure 2.8 adding your pin connections, and include this drawing to your your Lab report

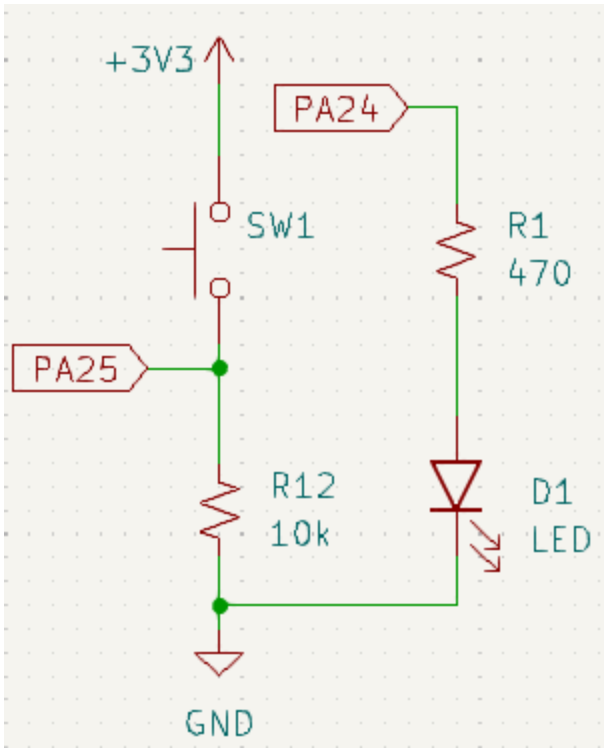


Figure 2.8. Circuit diagram showing the Lab 2 interface circuits drawn in KiCad. Refer to part a) to determine to which pins your switch and LED will be connected.

If you wish to learn KiCad, see the setup instructions for PCB camp.
https://docs.google.com/document/d/1mNN-iGtPof6gNw6qGtRm7M7LKCOSjDxiDxAOJB-Q2ys/edit

Part c - Build and test the switch circuit

Initially you can use the assembly **main** program and **Init** subroutine from the **NotGateAsm** project. This way you can do hardware debugging before writing any software. To run the **NotGateAsm** project connect the switch input pin to PB0.

Build the switch circuit on a solderless breadboard, connecting the input to PB0. To make connections to the MSPM0 we can run male-male solid wire from the bottom of the microcontroller board to the protoboard. WE WILL NOT CONNECT TO +5V IN THIS CLASS. Single step through the main loop and verify your software can detect whether or not the switch is pressed.

The input voltage is the signal that is connected to PB0. With a positive logic switch interface, the resistor current should be 0 when the switch is not pressed and about 3.3V/10kΩ when the switch is pressed. The voltages should be near +3.3 V when pressed and near 0 V when not pressed. The goal is to verify the input voltage is low when the switch is not pressed and high when the switch is pressed.

If you are unsure about the wiring, please show it to your TA before plugging in the USB cable. I like to place my voltmeter on the +3.3V power when I first power up a new circuit. If the +3.3V line doesn't immediately jump to +3.3V, I quickly disconnect power. If the green led in the upper portion of the LaunchPad does not turn on when you think you are applying power, you might have a short circuit.

Using a multimeter, collect measurements on your switch circuit. If you do not have access to an ammeter, you can measure the voltage drop around the resistor and divide by 10,000 ohms. Since no current enters the microcontroller input pin, the resistor current will equal the current through the switch.

Parameter	Value	Units	Conditions
Input Voltage on PB0		volts	With switch not pressed (measured with voltmeter)
Resistor current		mA	With switch not pressed (measured with ammeter)
Input Voltage on PB0		volts	With switch pressed (measured with voltmeter)
Resistor current		mA	With switch pressed (measured with ammeter)

Table 2.1. Switch measurements.

Do not place or remove wires on the protoboard while the power is on.

Part d - Build and test the LED circuit

Build the LED circuit on the solderless breadboard, connecting the output to PB1. You will test the LED using the unmodified **NotGateAsm** project. Single step through the main loop and verify your software can turn on and turn off the LED. The goal is to test the switch input and LED output.

Using a multimeter, collect measurements on your LED circuit. If you do not have access to an ammeter, you can measure the voltage drop around the resistor and divide by 470 ohms. Run the **NotGateAsm** program and stop when the LED is off to get measurements with the LED off. Run the test program and stop when the LED is onto get measurements with the LED on. Compare the measured values with the estimated values calculated in Part b) above.

Parameter	Value	Units	Conditions
LED Voltage, V_d		volts	With LED off (measured with voltmeter)
LED Current, I_d		mA	With LED off (measured with ammeter)
LED Voltage, V_d		volts	With LED on (measured with voltmeter)
LED Current, I_d		mA	With LED on (measured with ammeter)
LED Power, $V_d * I_d$		mW	With LED on (multiply two measurements)

Table 2.2. LED measurements.

Part e - Design a data structure to store Morse code

Figure 2.9 shows one possible data structure you could use to convert an ASCII character into Morse code. Each element of the list is five 32-bit numbers. Each number is a dot (100ms) or a dash (300ms). Therefore each element in the list is 20 bytes. There are 26 elements, one for each letter. Assume **Letter** is an ASCII character from 'A' to 'Z'. The index into the list is **Letter**-0x41, because 'A' is 0x41. Because the Morse Codes vary from 1 to 4 symbols, the dots and dashes have a null (0) termination. For example, the code for 'A' is dot-dash.

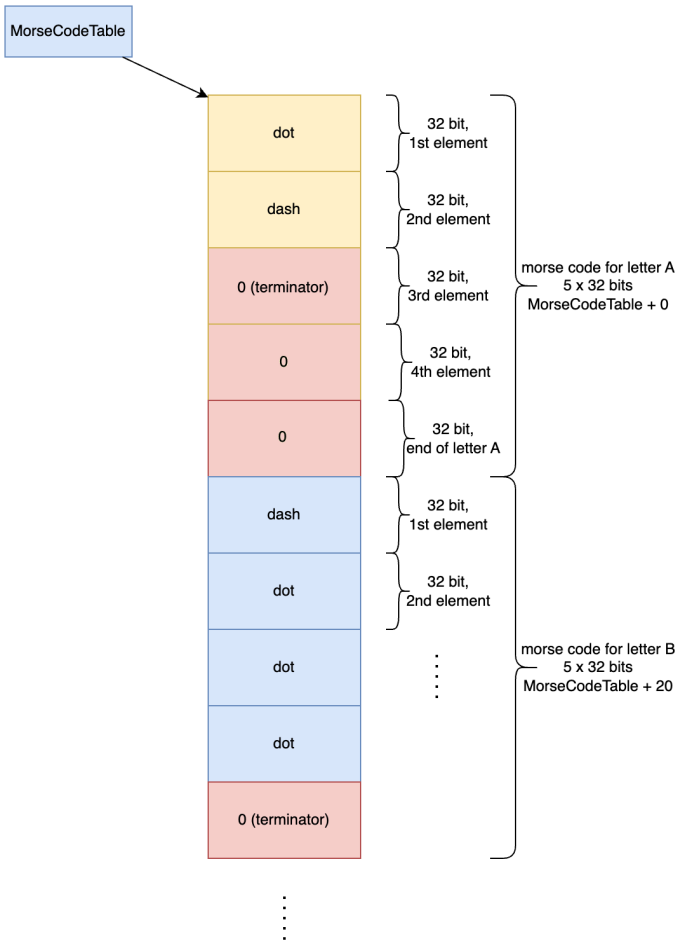


Figure 2.9. A list of 26 elements and each element is five 32-bit numbers.

The following code creates the data structure described in Figure 2.9. This code is part of your Lab 2H starter project. Edit this data structure as needed and use it to convert letters to Morse code. Feel free to organize the structure however you wish. It should be placed in ROM, because it is a constant. You can define values in milliseconds for **`dot`** and **`dash`** using **`.equ`**.

```
.equ dot,100   // msec
.equ dash,(3*dot)
.equ interelement,dot
Morse:
.long  dot,  dash,    0,    0, 0 // A
.long  dash,  dot,  dot,  dot, 0 // B
.long  dash,  dot, dash,  dot, 0 // C
```

```
  .long  dash,  dot,  dot,    0, 0 // D
  .long  dot,     0,    0,    0, 0 // E
  .long  dot,   dot, dash,  dot, 0 // F
  .long  dash, dash,  dot,    0, 0 // G
  .long  dot,   dot,  dot,  dot, 0 // H
  .long  dot,   dot,    0,    0, 0 // I
  .long  dot,  dash, dash, dash, 0 // J
  .long  dash,  dot, dash,    0, 0 // K
  .long  dot,  dash,  dot,  dot, 0 // L
  .long  dash, dash,    0,    0, 0 // M
  .long  dash,  dot,    0,    0, 0 // N
  .long  dash, dash, dash,    0, 0 // O
  .long  dot,  dash, dash,  dot, 0 // P
  .long  dash, dash,  dot, dash, 0 // Q
  .long  dot,  dash,  dot,    0, 0 // R
  .long  dot,   dot,  dot,    0, 0 // S
  .long  dash,    0,    0,    0, 0 // T
  .long  dot,   dot, dash,    0, 0 // U
  .long  dot,   dot,  dot, dash, 0 // V
  .long  dot,  dash, dash,    0, 0 // W
  .long  dash,  dot,  dot, dash, 0 // X
  .long  dash,  dot, dash, dash, 0 // Y
  .long  dash, dash,  dot,  dot, 0 // Z
```

Part f - Develop and debug the system using the logic analyzer or scope

The best approach to time delay would have been to use a hardware timer, which we will learn to use in Lab 4. In this lab we do not expect you to use the hardware timer. However, we expect you to write a software **Delay** function like Program 1.8.1 in the textbook. For example, to delay 10ms, one could

```
  LDR R0,=800000 // 10ms = (800,000 cycles)*12.5ns/cycle
  BL  Delay
```

You will need to use a logic analyzer or an oscilloscope to debug your system. We strongly encourage you to use the real scope, connecting one channel to the **Start** input and another channel to the **LED** output. Use TExaS logic analyzer only if you cannot come to campus.

We suggest developing the code in stages:

Stage 0: Output a 'dot','inter-element' over and over. I.e., make LED on for 100ms then off for 100ms. Use the scope in lab to check the timing.

```
Lab2: MOVS R0,#1      // TExaS in logic analyzer mode
      BL   Lab2Grader // Logic Analyzer, bus clock to 80 MHz
// Initialize your LED output
loop:
// turn LED on
// wait 100ms, dot timing
// turn LED off
// wait 100ms, inter-element timing
      B    loop
```

Use this stage to test the time delays for dot and inter-element gap.

Stage 1: Write code in Stage 1 so the main loop outputs a single character over and over. Stage 1 software could be implemented this way:

```
Lab2: MOVS R0,#1      // TExaS in logic analyzer mode
      BL   Lab2Grader // Logic Analyzer, bus clock to 80 MHz
// Initialize your LED output
loop:
// 1 second wait
// Send the letter L including the inter-element gaps
      BL Dot
      BL Dash
      BL Dot
      BL Dot
      B    loop
Dot:  PUSH {LR}
//    turn LED on
//    wait 100ms, dot timing
//    turn LED off
//    wait 100ms, inter-element timing
      POP {PC}
Dash: PUSH {LR}
//    turn LED on
//    wait 300ms, dash timing
//    turn LED off
//    wait 100ms, inter-element timing
      POP {PC}
```

How to run the logic analyzer: https://youtu.be/ZRa2pHbkXiM

Use this stage to test the time delays for dot dash and inter-element gap.

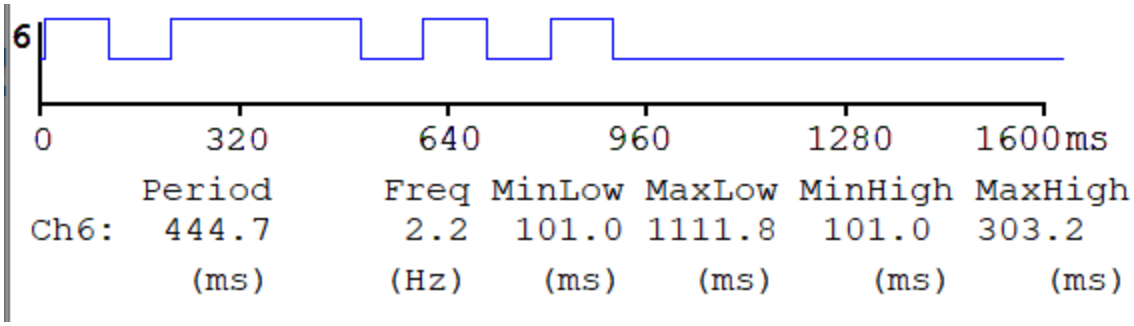


Figure 2.10. TExaS logic analyzer trace showing the letter L. Notice the dots are about 100ms long, the dashs are about 300ms long and the inter-element space is 100ms long. There is a rising-edge trigger on the LED signal.

Stage 2: Rewrite code in Stage 1 so the letter is sent every time you press the switch. Place the variable **Letter** in the *Expressions* window so you can see the random letter returned by the grader. Place the variable **Wave** in the *Expressions* window so you can decoding seen by the grader. Stage 2 software is written this way:

```
Lab2:
// Initialize your switch input and LED output
loop:
// wait for the switch to be released
```

```
// wait for the switch to be pressed
      MOVS R0,#1      // TExaS in logic analyzer mode
      BL   Lab2Grader // bus clock to 80 MHz
// Grader will return R0 as a random character from 'A' to 'Z'
// send that random one letter
      B    loop
```

Think about a helper function like this:

```
// R0=n is either 100 or 300
Out:  PUSH {LR}
//    turn LED on
//    wait n ms, dot or dash timing
//    turn LED off
//    wait 100ms, inter-element timing
      POP {PC}
```

Figure 2.11 shows the *Expressions* window. Figure 2.12 shows the input and output pins on the TExaS logic analyzer (call **Lab2Grader** with R0=1). In this figure, the LED output is on PB17, logic analyzer channel 5, and the Start input is on PB0, logic analyzer channel 0. I.e., PB18 is channel 6, PB17 is channel 5, PB16 is channel 4, PB3 is channel 3, PB2 is channel 2, PB1 is channel 1, and PB0 is channel 0. Alternatively, ask a TA to show you how to use the real oscilloscopes in the ECE319K lab.

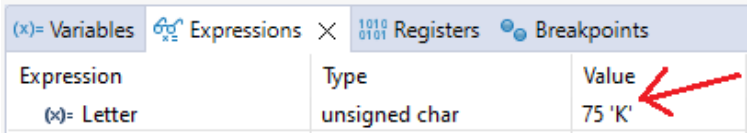


Figure 2.11. Every time you press the Start button you will get a random letter from 'A' to 'Z'.

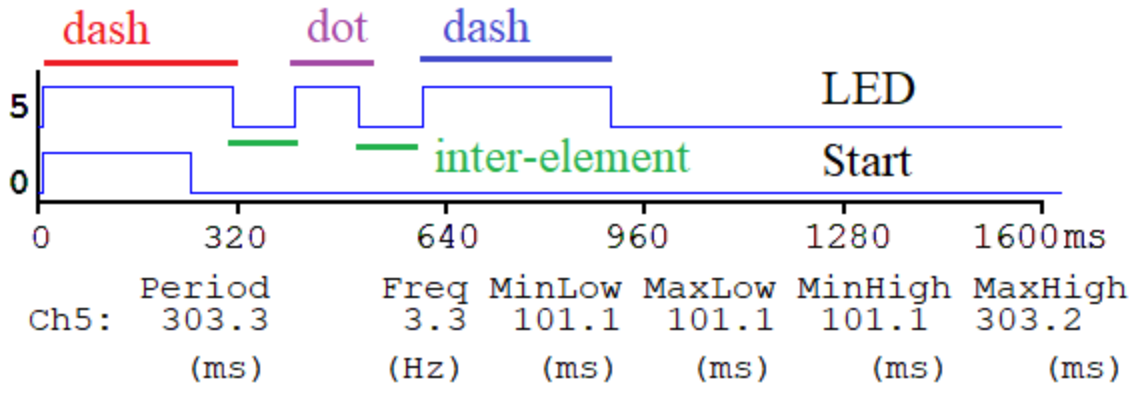


Figure 2.12. TExaS logic analyzer trace showing letter K, which is a dash-dot-dash. Again, the dots are about 100ms long, the dashs are about 300ms long and the inter-element space is 100ms long. There is a rising-edge trigger on the Start signal.

Part g- Run the grader

Please make sure your protocol is correct before running the grader. Place the string **Wave** in the *Expressions* window to observe how the debugger decoded your LED output.

- "**D**" is a dash

- ‘**d**’ is a dot
- ‘**.**’ is an inter-element space.

There are some error codes

- ‘*****’ is LED on for less than 75ms
- ‘**?**’ is LED on for 125 to 275ms or more than 325ms
- ‘@’ is LED off between dot/dash less than 75 ms
- ‘&’ is LED off between dot/dash more than 125 ms

Have both a wait for release and a wait for touch, so the grader has time to finish processing the previous code before you start another. You should wait at least 2 seconds before hitting the switch again. The grader should be run this way (*note*: you have to first push the Start button to begin grading):

```
Lab2:
// Initialize your switch input and LED output
loop:
  // wait for the switch to be released
  // wait for the switch to be pressed
      MOVS R0,#10      // TExaS in grader mode
      BL   Lab2Grader // bus clock to 80 MHz
// Grader will return R0 as a random character from ‘A’ to ‘Z’
// send that random one letter
      B    loop
```

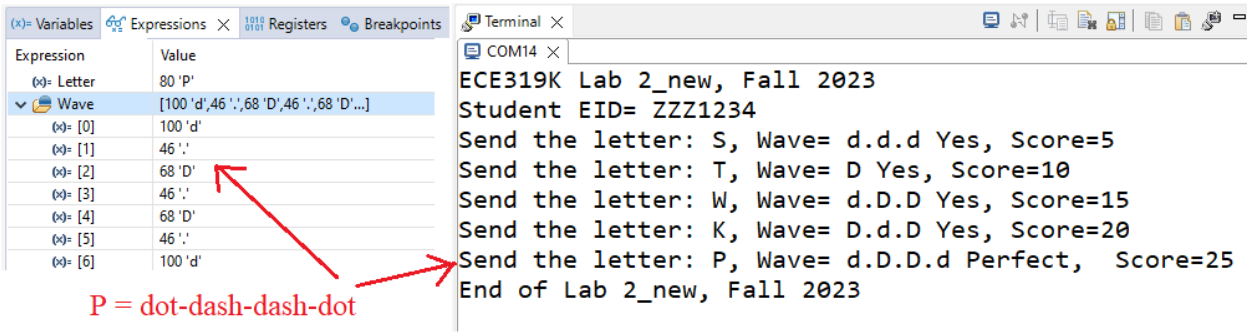


Figure 2.13. The Expressions and Terminal windows showing grader (it should say Lab2H Spring 2025

Demonstration

All lab assignments will follow a similar grading rubric:

- **20% Deliverables** - A collection of questions and code that will be submitted to Canvas (described below)
- **25% Performance** - The result returned by the grader. Does your code handle all situations correctly?
- **5% Code Standard Adherence** - The code you develop should be readable and well structured.
- **50% Demonstration** - The TA will ask you questions during your lab checkout about the assignment and your implementation. This also includes the visual correct execution of the lab assignment.

The TA may look at your data and expect you to understand how the data was collected and how the switch and LEDs work. Also be prepared to explain how your software works and to discuss other ways the problem could have been solved. We may test to see if you can measure voltage, current and/or resistance with your meter (so bring your meter to the demonstration).

Please make note of which TA checked you out. The name of the TA will greatly help you when resolving any grading issues later.

Do all these well in advance of your checkout

1. **Signup for a time with a TA. If you have a partner, then both must be present**
2. **Upload your software to canvas, make sure your names are on all your software**
3. **Upload your one pdf with deliverables to Canvas**

Do all these during the TA checkout meeting

1. **Have your one pdf with deliverables open on your computer**
2. **Have CCS Lab 2H open so TA can ask about your code**
3. **Start promptly, because we are on a schedule.**
4. **Demonstrate lab to TA**
5. **Answer questions from TA to determine your understanding**
6. **TA tells you your score (later the TA will upload scores to Canvas)**

Deliverables

*Upload your **ECE319K_Lab2H.s** file to Canvas. Combine the following components into one pdf file and upload this file also to Canvas. Have the pdf file and CCS open on your computer during demonstration*

1. Your name, professor, and EID
2. Circuit diagram (hand-drawn or optionally using KiCad), like Figure 2.8
3. Estimated LED voltage and current using the data sheet using Figure 2.6 and $I_d = 20*V_d-32$
4. Switch measurements (Table 2.1)
5. LED measurements (Table 2.2). Compare the LED on currents: mathematical analysis in Part b) using the simplified equation as $I_d = 20*V_d-32$, and measured current in Table 2.2.
6. A screenshot of a logic analyzer or scope showing one letter, like Figure 2.12
7. Terminal window of autograder, like Figure 2.13.

Optional Feedback : http://goo.gl/forms/rBsP9NTxSy

Precautions to avoid damaging your system

1. Do not attach or remove wires on the protoboard when power is on. Always shut power off when making hardware changes to the system.
2. Touch a grounded object before handling CMOS electronics. Try not to touch any exposed wires.
3. Do not plug or unplug the modules into the LaunchPad while the system is powered.
4. Do not use the MSPM0 with any external power sources, other than the USB cable. In particular, avoid connecting signals to the MSPM0 that are not within the 0 to +3.3V range. Voltages less than 0V or greater than +3.3V will damage the microcontroller.
5. In this 319H, we will only use 5V power pin for the IR LED interface in Lab 8. Otherwise, do not use 5V.

FAQ

The list of FAQ below are populated from forums over the semesters (thanks to the contributions of all past TAs and students). More questions may be posted so please check back regularly.

1. For my 1 ms delay, I'm having trouble getting the delay up to the right number of cycles. I've tried combining multiple delay subroutines to increase the delay, but I'm still quite a ways from 1ms. Can anyone share a smart way of increasing the delay?

Notice that the clock for this lab is running at 80 MHz instead. Your calculation in writing your delay loop count must account for this speed which implies each cycle is 12.5ns. Calling **Delay** with R0=80000 should wait 1 ms.

2. How are we supposed to determine which pair of pins has the switch across them using the ohmmeter? Each pair has the exact same resistance between them when I measure it.

   Were you measuring the resistance with the switch pressed? If so, all of the pins will be connected together and have the same, albeit small, resistance. Otherwise, if you were measuring with the switch not pressed, you should be measuring different resistances between different pairs. If you are absolutely sure this is not the case, it is possible that you could have a broken switch.

   The size of the resistance might also provide a hint as to what is going on (think about when the resistance should be very large and when it should be very small, assuming the switch is working correctly).

3. How do we measure current through a resistor?

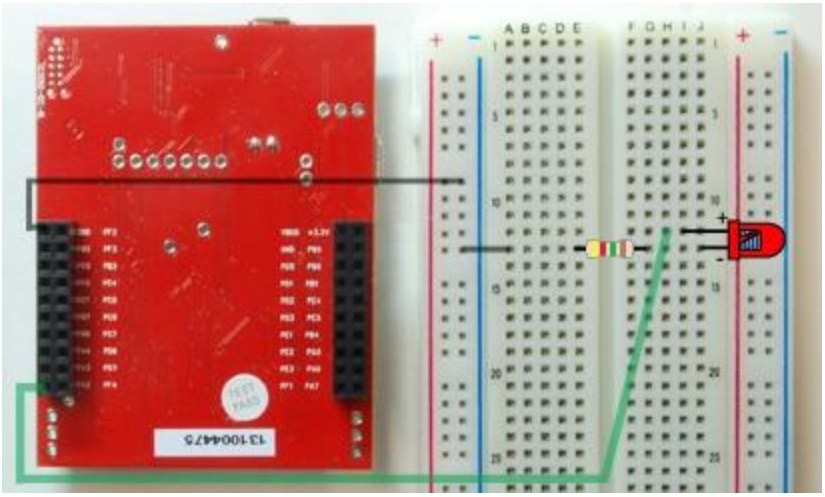
   See Tutorial below. You put the multimeter (in current mode) in series with the resistor. Or you can kind of "cheat" and measure the voltage difference across a resistor, then calculate the current as voltage difference divided by resistance. You should understand how to do both ways though. Recall Ohm's Law only applies to the resistor, not the LED.

4. After thoroughly checking my hardware wiring, I tried implementing it with the software and couldn't even get the LED to turn on. Are there any possible explanations for why this could be happening?

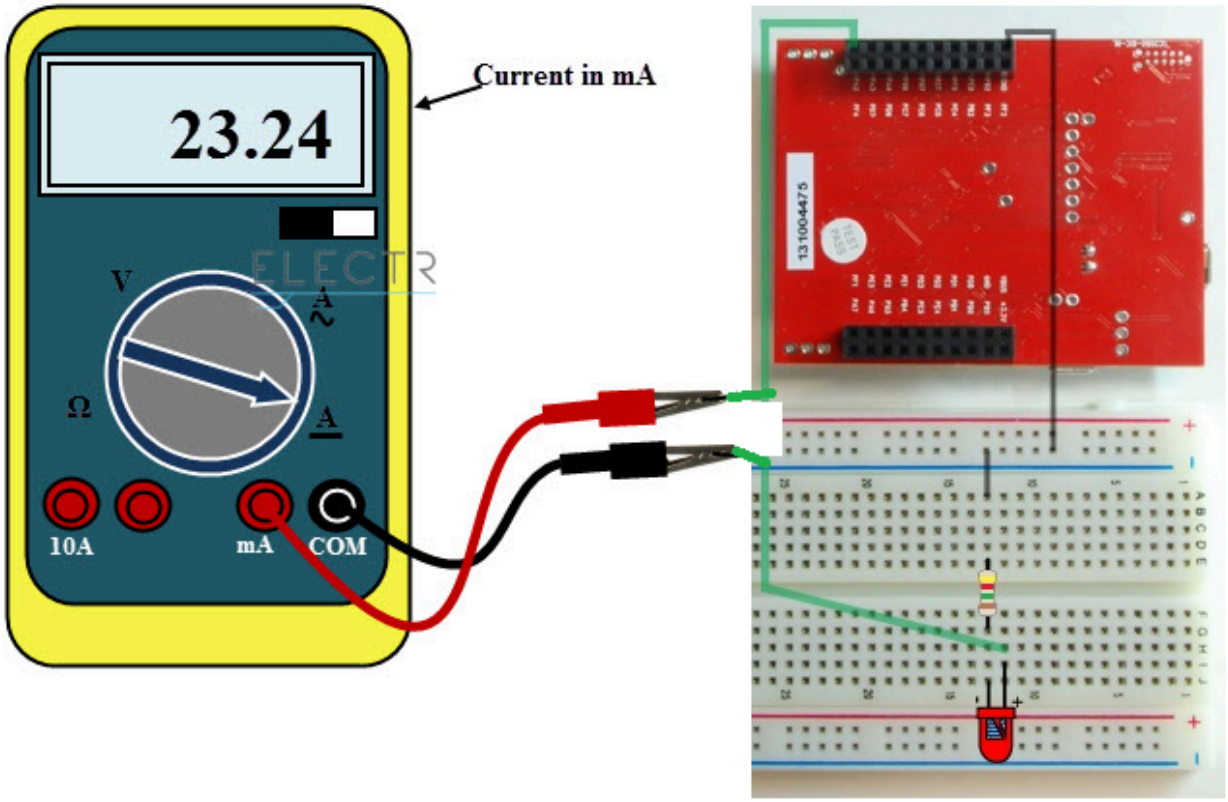
   You should test your switch and LED hardware using the **NotGateasm** project, connecting the input to PB0 and the output to PB1.

Tutorial on how to measure current through an LED

1) Start with a circuit that is operational. For example, output to the GPIO so the LED is on



2) Turn off the power, break the circuit, and connect the +probe and -probe of the DVM current meter in place of the break. The meter must be in current mode and not voltage mode. Most DVMs require you to connect the probes to the meter in a special place to measure current



https://www.electronicshub.org/current-measurement-using-multimeter/

3) Turn on the power (and make the GPIO pin high) and the system should run the same (LED on)