



University of California, Davis
Electrical and Computer Engineering



California State Summer School for Mathematics & Science

Cluster 8: Internet-of-Things

Lab 1: Embedded Systems Programming
Using the TI Microcontroller and Energia

Introduction:

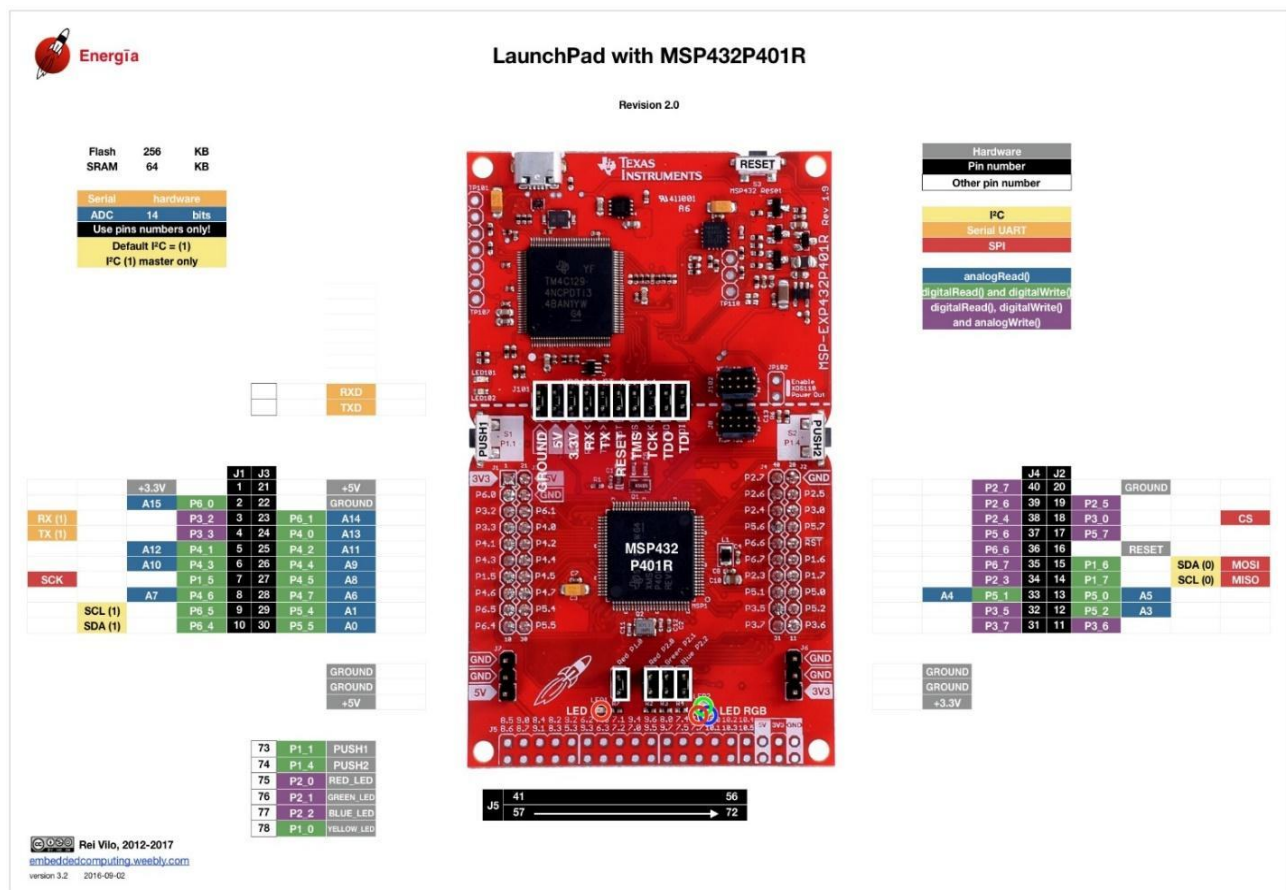
a. Purpose/ Scope

This lab will act as the foundation for the future projects and exercises throughout the course. During this lab, you will (1) familiarize yourself with the basics of embedded systems and their capabilities, namely how MCUs take signals/information in or transmit them; (2) learn the bare minimum required to write and execute programs for your microcontroller

b. Resources/Lab Materials

Lab Materials:

- 1x TI MSP432P401 LaunchPad Kit
- 1x Breadboard
- 1x Red LED and 1x Green LED
- Wire Jumpers
- 1x 330 Ohm resistor



Pin layout of MSP432

Part A: Installing Energia on your Personal Computer

Be sure to follow the instructions outlined below carefully as any deviation can result in large errors when using your board.

Links:

Energia Download on Canvas -> Files -> Lab 1 Materials

Video tutorial for Energia and driver installation for Windows: [Link](#)

Video tutorial for Energia installation for MacOS: [Link](#)

Procedure:

Step 1: Download Energia from Canvas. Use the most recent version of Energia and download the appropriate .zip file for your operating system. Extract the .zip file to wherever you think is best; your C drive is a common choice, but note there can be issues with the drivers if there are any spaces in the path name. Open the unzipped Energia folder and double click on “energia” file.

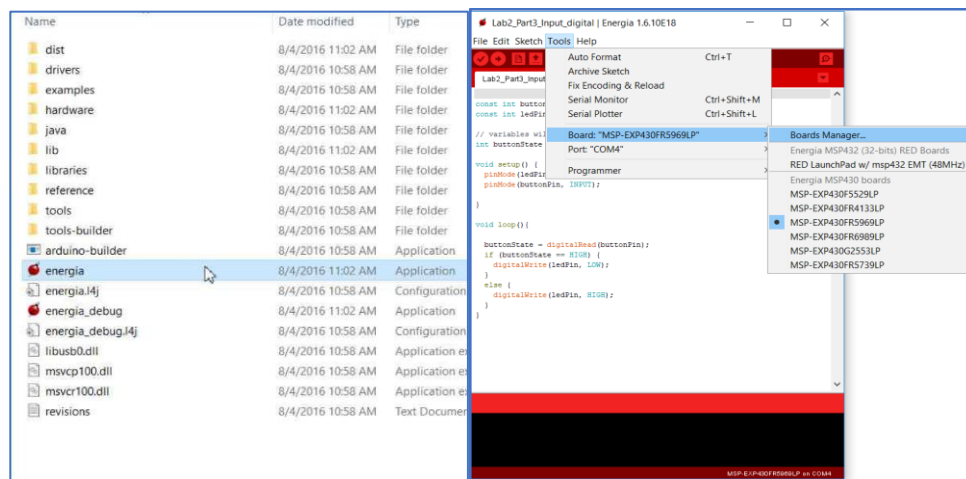


Fig. 1. Energia board selection

Step 2: You need to install the Energia package for MSP432P401R before you can upload code to your Launchpad.. Go to Tools **➤** Board: “MSP-EXP.....” **➤** Boards Manager. Search for “Energia MSP432 EMT RED Boards” and install the newest version. **NOTE: This installation can take quite a bit of time. DO NOT CANCEL THE INSTALL ONCE IT’S STATRTED. If the installation is canceled before it is finished your board will get errors when trying to run a program.**

Step 3: Connect your TI LaunchPad to the computer through a USB cable. Verify which Com your board is connected to in your Device Manager. Open Energia and in Energia, go to Tools ➤ Port “...” and select the correct port number.

For Windows Users:

Open Device Manager. To find it, type “Device Manager” in the Window search box. Select Ports (COM & LPT). The COM port number should show after the name: “ XDS110 Class Application/User UART”. If you do not have the drivers installed, they can be found on canvas in “xds110_drivers.zip”. Download the file, extract the whole “xds110_drivers” folder and run “DPinst” or “DPinst64” if you have a 32 bit or 64 bit processor respectively.

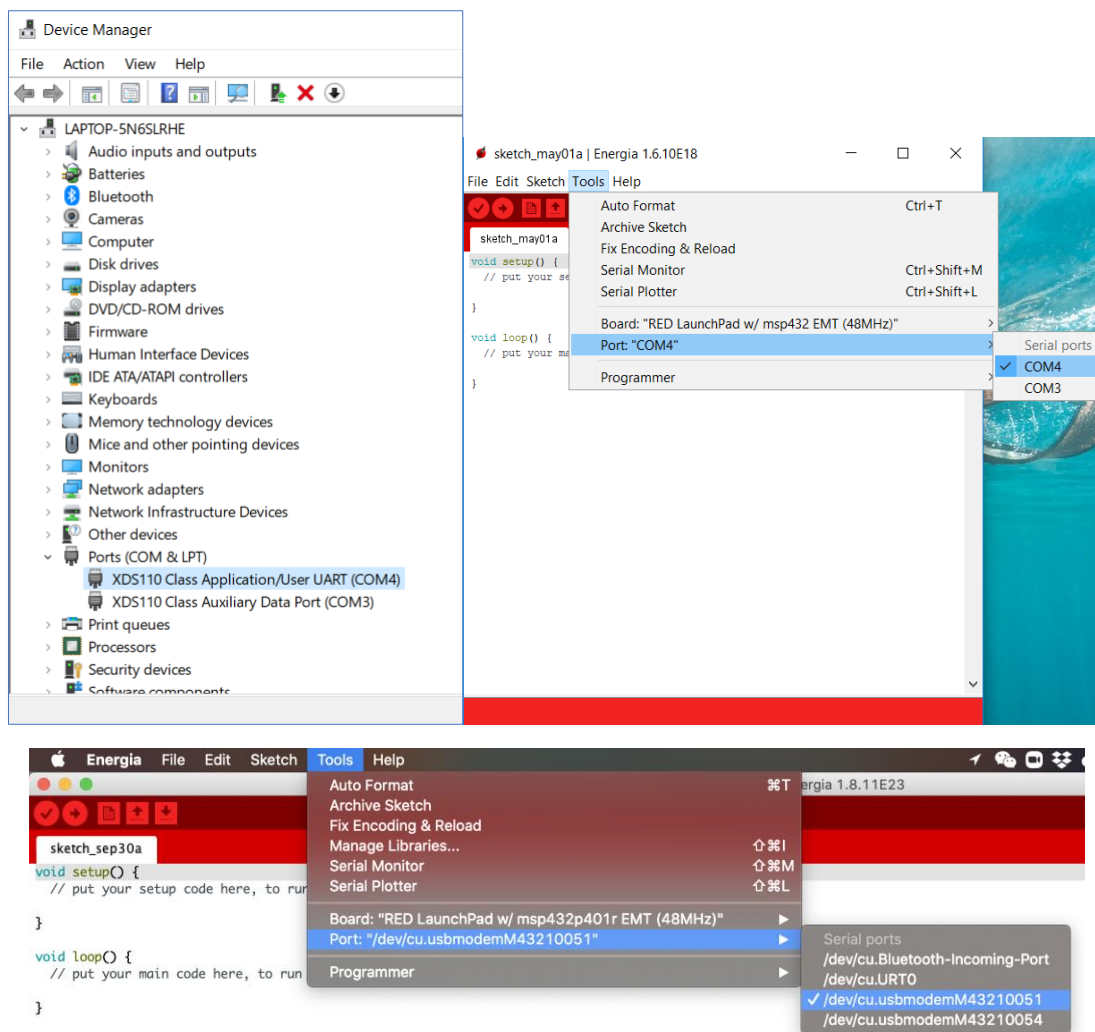


Fig. 2 Images of the Device manager showing the ports for the Launchpad and how to select it in Energia (a) Windows (b) MacOS.

For MAC users:

There is no driver needed for this board. Simply go to Tools ⑦ Port “...” and select the port name with “M432xxxxx“, which is the serial number of your board. Once you have done this, you may move onto part B which will be the exercises of the lab. Exercise one will make sure that Energia is installed and working properly.

Part B: Microcontroller exercises

Exercise 1: First program. Blink red LED

The following functions will be used during this exercise:

```
pinMode() :  
digitalWrite() :  
delay() :
```

Note that when noting specific functions or code we will use `Courier` fonts.

When beginning to write a program, Energia provides you the following template:



Fig. 3. Basic structure of Energia program

Elementary description of writing a program:

- (1) declare variables you intend on using at the beginning.
- (2) State their purpose within the `setup()` section.
- (3) Utilize them in the `loop()` section.

To see this and to test if Energia is working appropriately on your computer, we will make an LED blink on and then off using the three simple functions listed above.

Copy the following segment of code over the existing text in Energia. An image of what it should look like can be seen in Fig. 4.

```
//My first program

int LED = 75;

void setup() {
  // put your setup code here, to run once:
  pinMode(LED, OUTPUT); //variable LED (or pin 75 is set as an output)
}

void loop() {
  //make it blink!
  digitalWrite(LED, HIGH); //Turn the LED on
  delay(1000); //wait 1000ms
  digitalWrite(LED, LOW); //Turn the LED off
  delay(1000); //wait 1000ms
  /*loop tells the program to begin at the beginning of this section and
  repeat
  * the process continuously.
  */
}
```

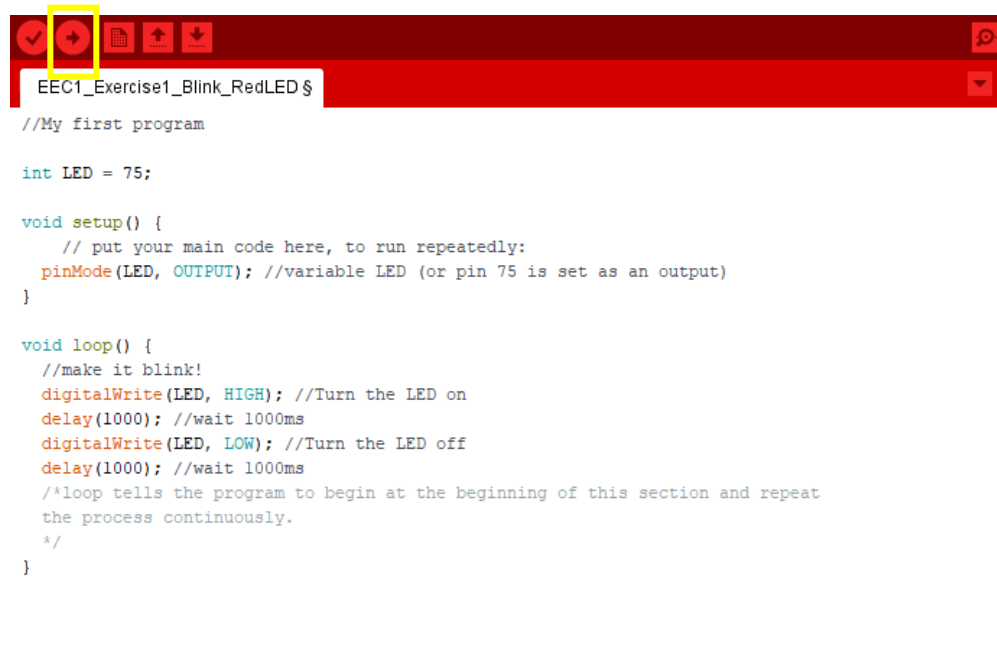


Fig. 4. Energia program to blink LED on MCU board. Upload button is outlined in yellow in upper left corner.

Click the play button outlined in yellow above and you can see red LED on your board begin to blink. Now we can break down what is happening here so you may write your own program. The first two minor points to mention is the semicolon (;) at the end of every line and the visible comments seen in light grey text. Programs will perform each instruction one at a time. To let the computer know that the instruction we are giving is done, we terminate the line with a semicolon.

The text seen throughout the program in light grey is a comment and it will not interfere with the program execution. Comments serve several convenient purposes as you write code such as leaving notes for yourself to read later, instructions on how your program works for others to follow, or as a way of blocking out code you don't want to run now. Comments can be written by first including `//` and then typing the comment on a single line or can be extended in paragraph over multiple lines using the tokens `/*` to start the multiline comment and `*/` to end it.

Moving forward with our analysis of the program, let us break it into three separate portions corresponding to our description of how to write an Energia program given above:



Fig. 5. Breakdown of LED blinking program

Red: This section states what numbers we intend on using by placing them in a variable and the type of value it will be. In this case, we have the line:

```
int LED = 75;
```

Here, we are assigning the value 75 to a variable to use later called LED and stating that the value 'LED' will hold will be in integer form. There are multiple data types that can be declared but we will only mention them as needed. Variables can be called just about anything you choose however they should make sense. For this line, we are saying that the pin we will be working with is pin 75 which is an LED pin (see Fig. 1. showing the pins of the MCU). Be careful when naming certain variables as certain words are reserved from the computer such as 'int', 'loop', 'delay', and others that you may use as commands within your code.

Green: This section tells the microcontroller what the value assigned to LED will be used for by using the `pinMode()` function. This function takes two values: a pin value and how it is configured. In our example, the pin value is given as LED and we are specifying it as an OUTPUT

meaning it will be sending a signal from the board to the red LED. We could have just as easily written `pinMode(75, OUTPUT)` however it is often easier to keep track of variables with meaningful names than numerical values associated with different pins.

Blue: Now that board is setup, we can tell it what to do to the designated pins. Here we use two functions: `digitalWrite()` and `delay()`. The function `digitalWrite()` will take two arguments, namely, a pin on the board and a value of 'HIGH' or 'LOW'. High and low refer to digital values which can be interpreted as on or off which is usually equivalent to 5 volts or 0 volts. For this case, 'HIGH' turns the LED on while 'LOW' turns it off. The following line uses the `delay` function which causes the program to pause for an amount of time, in milliseconds (ms), provided as an argument; in this case 1000 ms or 1 second. Given this information we can see how the program executes within the main portion of the code labeled '`loop()`'. First, the LED pin is assigned a high value to turn it on. Next, the program tells the MCU to wait 1 second. The following line tells the MCU to turn the LED off. The final line tells the MCU to once again wait for 1 second before returning back to the beginning of the `loop()` segment to repeat the process indefinitely.

QUESTION: Answer the following question once you complete the exercise.

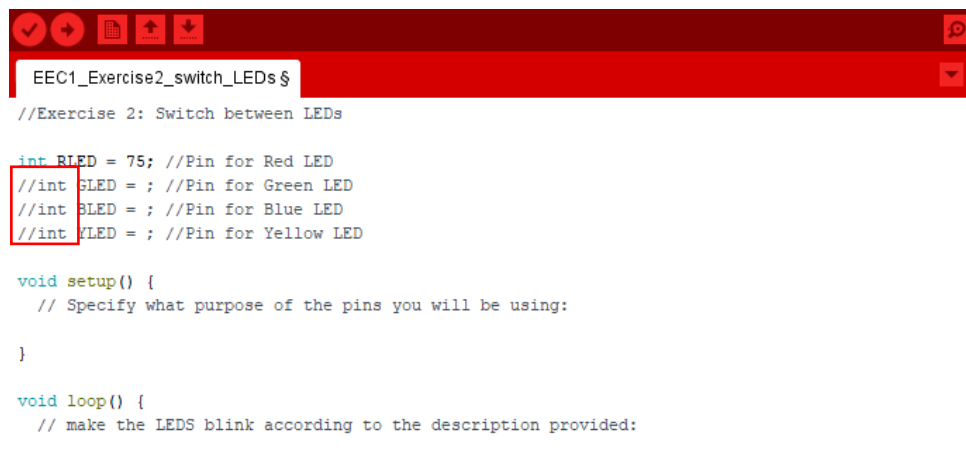
1. What would happen if you changed the variable "LED" from 75 to 77?

Exercise 2: Switching between LEDs

Description: For this exercise you will write your own short program which is an extension of the previous program starting from the template depicted below. **To demonstrate that you successfully completed this exercise, show one of the TAs that it is working correctly.** The MCU has 4 built-in LEDs of different colors: red, blue, green, and yellow (it says yellow but it is just another red LED). The goal of this program is blink between the different colors. The main program should start by turning the red LED on and then wait for half a second. Next, it should turn off the red LED and immediately turn on the green LED. This procedure should continue with the blue LED next and then the “yellow” LED. Once, the yellow LED turns off, the program should return to the beginning of the loop and repeat the processes.

Procedure: The procedure for this section will follow the outline for writing a program discussed previously:

1. Assign values to the variables for each LED. Use the pin layout diagram provided above to determine which pin to use. If you are having trouble, find which pin the red LED is assigned to, to help you read the diagram. The lines assigning the values for each variable are currently blocked out using comments (red block in image below). Remove the comment marks at the beginning of the line before running your code.



```
EEC1_Exercise2_switch_LEDs$
//Exercise 2: Switch between LEDs

int RED = 75; //Pin for Red LED
//int GREEN = ; //Pin for Green LED
//int BLUE = ; //Pin for Blue LED
//int YELLOW = ; //Pin for Yellow LED

void setup() {
  // Specify what purpose of the pins you will be using:
}

void loop() {
  // make the LEDs blink according to the description provided:
```

Fig. 6. Code structure for exercise 2

2. Using the `setup()` section, tell the MCU the purpose of each pin using the `pinMode()` function. You should have four separate lines in this section for declaring your pins. Hint: use and modify the code from exercise 1 above to make it work for multiple LEDs.

3. In the `loop()` section, write a block of code to perform the actions provided in the description. Begin by turning the red LED on and then wait for half a second. Then turn the red LED off, turn the green LED on, and wait for half a second. Follow this pattern so that the LEDs blink from red to green to blue to “yellow”.

Exercise 3: Blink external LED

Description: The following exercise is meant to illustrate how it is possible to interact with external components using your microcontroller and to provide an introduction into assembling circuits. In this exercise you will use the `digitalWrite()` function to turn an LED that is external to the board on and off in the same manner as the LEDs located on the MCU board.

Procedure:

1. Assemble the circuit on your breadboard using the image below as a guide. For this assembly, you will need jumper wires, a 330 ohm resistor, and a red LED. **Important:** pay close attention to the illustration of the red LED in the diagram below. If you look carefully, the right leg is bent at an angle. If you then look at the physical LED you are using to assemble the circuit you will notice one of the legs is longer than the other. The bent leg in the diagram corresponds to the longer leg on the actual device. This is an indicator of how to place the LED into the circuit. LEDs are polar devices. That means the direction is important! This is a topic that will be discussed in greater detail in later labs. In this diagram, the red wire is running from the pin labeled 3V3 on the MCU to the breadboard, the black wire is running from the pin labeled GND on the MCU to the breadboard, and the blue wire is running from the pin labeled P6.4 (left side, labeled 10 below) on the MCU to the breadboard.

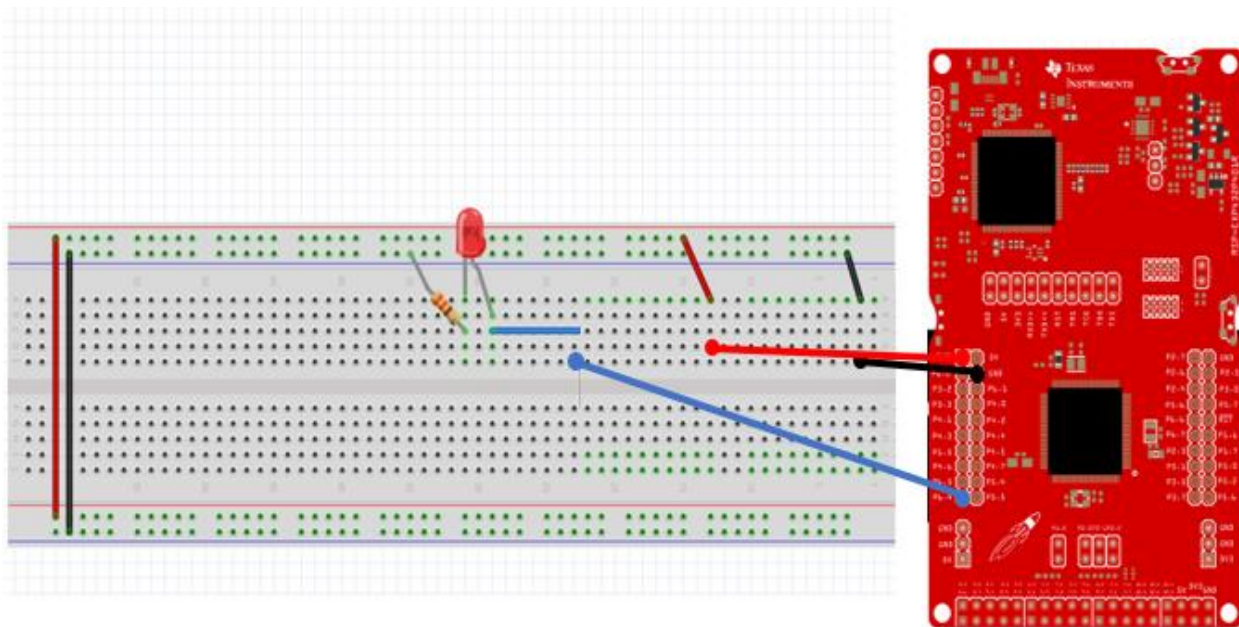


Fig. 7. Diagram illustrating how to connect the MCU to a breadboard to interface with an LED

2. Once the circuit is assembled, copy the code provided into Energia and run it. You should notice that the circuit is behaving identically to what you did in exercise 1. The only difference between the two is the declaration of which pin acts as a digital output and the peripheral circuitry.

```
//Exercise 3: Turn on external LED

int ext_LED = 10; //digital write pin

void setup() {
  // put your setup code here, to run once:
  pinMode(ext_LED, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(ext_LED, HIGH);
  delay(1000);
  digitalWrite(ext_LED, LOW);
  delay(1000);
}
```

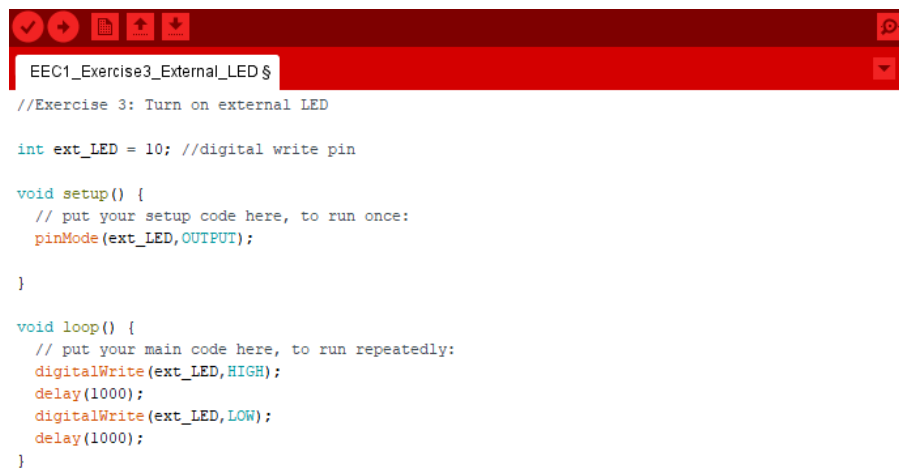


Fig. 8. Program used in exercise 3

QUESTIONS:

1. In the above code, we assign the variable `ext_LED` the value 10 because pin 10 on the MCU is a digital write pin. List one other pin that could be used as a digital write pin in this exercise.
2. Replace the red LED in the circuit with a green LED. Compare the brightness you see between the two LEDs. We will discuss this behavior further in future labs.

Exercise 4: Blink external LED with pushbutton (if/else statements)

Description: This program acts to introduce two concepts: (1) the ability to control external devices by sending a signal to the MCU, (2) the use of what are known in programming as control structures. There are many type of control structures used in programming that will be introduced throughout this lab but the first one we will introduce is the if/else statement. Problems in programming often occur that require different sets of instructions to execute. The if/else statements provide a means of controlling these events. Simply put, if event A happens do X, otherwise (else) do Y. In the case of the following program, we are interested if the state of the pushbutton on the MCU board changes.

Procedure:

1. For this exercise, you will be using the same circuit you built in the last step (using the red LED). Carefully read through the following program, copy it into Energia, and execute it.

```
//Exercise 4: Toggle external LED with pushbutton and if/else statements

const int buttonPin = 73; //equivalently PUSH1
const int ext_LED = 10; //digital write pin

int buttonState = 0;

void setup() {
  pinMode(ext_LED, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
}

void loop() {
  buttonState = digitalRead(buttonPin);
  if (buttonState == HIGH){
    digitalWrite(ext_LED, HIGH);
  }
  else{
    digitalWrite(ext_LED, LOW);
  }
}
```

```
//Exercise 4: Toggle external LED with pushbutton and if/else statements

const int buttonPin = 73; //equivalently PUSH1
const int ext_LED = 10; //digital write pin

int buttonState = 0;

void setup() {
  pinMode(ext_LED, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
}

void loop() {
  buttonState = digitalRead(buttonPin);
  if (buttonState == HIGH){
    digitalWrite(ext_LED, HIGH);
  }
  else{
    digitalWrite(ext_LED, LOW);
  }
}
```

Fig. 9. Program used for exercise 4.

Next, press the button on the MCU board depicted in the image below. When holding down the button, the LED should turn off, and when the button is released it should turn back on. Why this is happening is detailed in the setup() block and the loop() block. Notice in the setup block, the push button is declared as an “INPUT_PULLUP.” This means the value of this pin is left as a “high” signal. The loop block details the control flow of the program. One strategy to handling control statements is to use a variable representing the “state of the program” and then executing the control based on whether the state has changed.

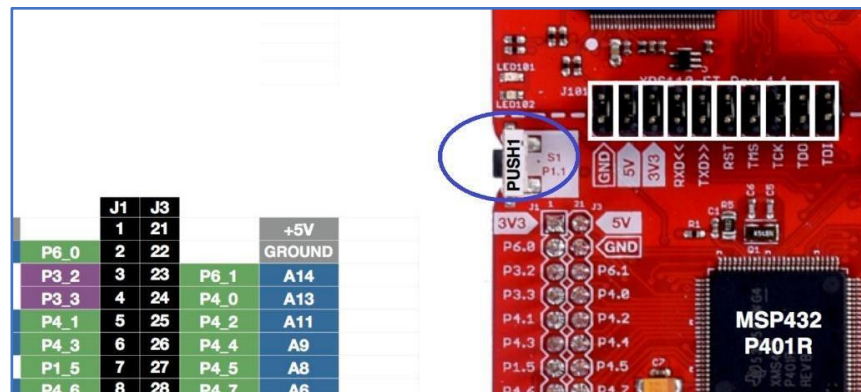


Fig. 10. MCU pushbutton

QUESTIONS:

1. In the *loop()* block of code, the first statement is *buttonState = digitalRead(buttonPin)*. What is the purpose of this line of code?
2. The next portion of this block is the if/else statement.
 - a. What is the if statement checking and if true, what is the following action?
 - b. When does the program execute the else statement?
3. Provide a brief explanation of how this program works to your TA.

Exercise 5: LED Brightness Control with analogWrite() and For Loops

Description: In this example, you will change LED brightness with the `analogWrite()` function. The difference between analog and digital signals is that analog signals vary with time and cover a range of values as opposed to digital signals which are left unchanging at a fixed value. The `analogWrite([pin], [number])` function uses Pulse Width Modulation (PWM) to create a signal which appears to be analog. In reality, a PWM is a periodic signal that switches between HIGH and LOW (or on and off) with a variable duty cycle that can be controlled by programming a signal number on the specified pin. The duty cycle is the percentage of each period that is HIGH or on, and the number written to it determines this percentage. When the number is equal to 0 the signal will be HIGH 0% of the cycle, and when the number is 255 the signal will be HIGH 100% of the cycle. By changing the duty cycle, `analogWrite()` is used to change the LED brightness (this can also be used to control the speed of a motor, or vary other controls). To understand `analogWrite()`, and PWM in more detail refer to the discussion provided in the prelab.

This program also utilizes one of the most important control structures in programming: For loops. Imagine you wanted to repeat a task a pre-determined number of times. When programming this command, you can copy the code such that it repeatedly executes one line after the other, however, this becomes impossible if you wanted to execute this command a million times. For loops offer a solution to this problem. A for loop begins with the word “for” followed by “(“ with three spaces to fill in separated by semicolons:

for (XX;YY;ZZ)

In this control structure XX acts to initialize the control variable that will be changing with each iteration of the loop, YY gives a logical condition that says how long the loop will go for and the loop will execute as long as that condition remains true, and ZZ gives instructions on how to modify the control variable at the end of each iteration. Consider the following example:

```
for (int i=0; i<100; i++){  
    (some instruction for the MCU);  
}
```

Breaking this snippet of code down, we first specify some control variable “i” that will have an integer value and is initially set to zero (blue part). The second entry (green part) says that whatever is defined in the loop will execute as long as “i” is less than 100. You may think at this point that we said “i” would always be less than 100, however, the third entry (purple part) states that at the end of *every* iteration of the loop, “i” will be incremented by 1. The statement, “i++” is equivalent to saying `i+1`. Just as well, one could replace the third entry as “i--” to decrease “i” by 1 at the end of each iteration. From this we can see that the example loop will execute whatever command is inside the loop, increment “i” by 1, and repeat until “i” equals 100.

Procedure:

The following program utilizes a similar circuit as the previous two exercises, however, we are now using an analog output as opposed to a digital input/output and so we must declare a new pin to use and move the wire connecting to the LED accordingly.

1. Copy the following code into Energia, but **don't execute it**. You'll notice that the variable declared `ext_LED` isn't assigned a value. We need to use this as the analog output. Refer to the pin out illustration of your MCU (Fig. 1) at the beginning of this document and select a pin that is functions as an analog output and declare `ext_LED` as that pin number.

```
//Exercise 5: LED brightness control with analogWrite() and For loops
int ext_LED = XX; //analog output pin
int val;

void setup() {
  pinMode(ext_LED, OUTPUT);
}

void loop() {
  for (val=255; val>0;val--)
  {
    analogWrite(ext_LED,val);
    delay(10);
  }
  for (val=0; val<255; val++)
  {
    analogWrite(ext_LED,val);
    delay(10);
  }
}
```



```
EEC1_Exercise5_LED_Brightness_analogout_For_Loops $
//Exercise 5: LED brightness control with analogWrite() and For loops
int ext_LED = ; //analog output pin
int val;

void setup() {
  pinMode(ext_LED, OUTPUT);
}

void loop() {
  for (val=255; val>0;val--)
  {
    analogWrite(ext_LED,val);
    delay(10);
  }
  for (val=0; val<255; val++)
  {
    analogWrite(ext_LED,val);
    delay(10);
  }
}
```

Fig. 12. Program for exercise 5

2. Before executing your code, remove the wire connected between the long leg of the red LED and pin 10 as used in the previous exercise. Move the wire to the pin of the MCU you specified in the code.

3. Execute the program and you'll notice the LED starts bright and then begins to dim. Once it turns off, it then starts to increase in intensity.

DEMONSTRATION: once you have verified that your circuit and program are working appropriately, show one the TAs present in the lab that your program is working **CORRECTLY**. Then answer the following questions:

1. Which pin (numerical value) did you assign to `ext_LED`?

2. There are two for loops used in the main loop block of the program. In no more than one paragraph, explain what the first for loop does and then what the second for loop does.