Editor's Note: For individuals looking to explore MCUs and MCU-based design principles, low-cost development kits offer an easy, low-risk entry into the intricacies of these complex ICs. In this detailed guide to the TI LaunchPad development kit, the authors break down hardware circuits and walk through MCU-based projects designed to help users work with the LaunchPad - the little red board that do so many big things, not the least of which is helping individuals gain knowledge and experience in MCU-based design. In a previous excerpt, the authors reviewed the basics of digital I/O and walked through examples of I/O design using the TI MSP430 MCU. This latest excerpt looks at LaunchPad programming with previous installments (Part 1, Part 2, Part 3 and Part 4) and concluding with this installment. Be sure to check out the previous excerpt on I/O design with examples using the LaunchPad.

Elsevier is offering this and other engineering books at a 25% discount. To use this discount, click on the coupon to the right and use code PBTY14 (valid through March 31, 2015) during checkout.

Adapted from "Getting Started with the MSP430 Launchpad, 1st Edition" by Adrian Fernandez and Dung Dang (Newnes)

Project 26: A Magic Eight-Ball Launchpad

Create a magic LaunchPad that displays a random response on the LCD display when the LaunchPad is shaken.

Ingredients

1. LaunchPad
2. Educational BoosterPack
   OR
   Breadboard, ADXL335 three-axis accelerometer, 1x1 kΩ, 2x10 kΩ, 1x15 kΩ resistors, 4x0.1 uF, 1x4.7 uF, 2x1 uF capacitors

In this particular project, we’ll be creating a fairly complex system that will require a three-axis accelerometer and the LCD. We will be combining a few concepts that we have recently learned about including analog signals from the accelerometer as well as SPI serial communication to an
First, let’s get the hardware set up. If you have the Educational BoosterPack, setup is quite easy. Simply plug in your Educational BoosterPack onto your LaunchPad and voila! Be sure that the two jumpers on the BoosterPack enable the Y and Z channels of the accelerometer to connect to the LaunchPad. Also be sure to remove the LED jumpers P1.0 and P1.6 on the LaunchPad. And lastly, position the TXD and RXD lines if you want to enable UART connection back to the computer.

For those who are Educational BoosterPack-less, we can recreate the required hardware using a breadboard, a few components, and some elbow grease! Below is a hardware diagram that shows how you can connect the accelerometer and the LCD module to the LaunchPad properly. Again, we’ve provided snapshots of the actual schematic as well as a diagram created using fritzing.
Let’s write some code!

Once your hardware is set up, let’s write some code!

Tip

Or copy some code. This is a long piece of code, so if you don’t feel like retyping 3 pages worth of code, you can go to the http://booksite.elsevier.com/9780124115880 and download the sketch in its entirety. We have placed the code in the book here for your reading convenience.
#include <SPI.h>
#include <LiquidCrystal.h>

const int xpin = A0;       // x-axis of the accelerometer
const int ypin = A1;       // y-axis
const int zpin = A2;       // z-axis
const int LCD_RS = 0;      // SS line is not used in our
const int LCD_EN = 1;      // hardware setup
const int LCD_RS = P2_3;   // RS pin of our LCD
LiquidCrystal lcd(LCD_RS, LCD_EN); // declare a LiquidCrystal
                                    // variable called 'lcd'

String response = "";  // this will hold our magic LaunchPad response
int result = 0;        // this will help us randomly choose a response
int startTime;         // this will help us periodically read our
                        // accel data

void setup() {
    lcd.begin(16, 2);     // initialize our LCD
    lcd.print("Shake the magic!"); // print out intro text
    lcd.setCursor(0,1);  // reposition cursor to next row
    lcd.print("LaunchPad!");
    startTime = millis();   // store start time. The millis() function returns how many
                            // milliseconds have elapsed since the program executed.*/
}

void loop() {
    /* these variables will hold the 3 axis accelerometer data sum
       will store our resultant vector */
    int a, b, c, sum;

    /* this checks to see if 100 ms have elapsed since we last checked
       accelerometer readings */
    if (millis()-startTime > 100) {
        a = analogRead(xpin) - 512; // read accel data
        b = analogRead(ypin) - 512; // offset the data to 'normalize'
        c = analogRead(zpin) - 512;

        /* use this formula to find the net force applied on all three
           vectors */
        sum = round(sqrt((a*a) + (b*b) + (c*c)));
    }

    if (sum<130) { // check to see if we exceeded threshold
        /* if so, trigger the Magic LaunchPad to return a random message!*/
        result = random(0, 3); // random(min, max); This function returns a random value from
                                // [min, max-1]. Then, pass in the random number between 0 & 3
                                // into a switch statement to return random response */
        switch (result) {
            case 0:
                response="Yes!";
                break;
            case 1:
                response="Maybe!";
                break;
            case 2:
                response="No!";
                break;
        }
        lcd.clear(); // clear display to show new random response!
        lcd.setCursor(0,0); // reset cursor
        lcd.print(response); // print our random response
        delay(3000); // show response for 3 seconds

        /* reset display and re-print the original prompt.*/
        lcd.setCursor(0,0);
        lcd.print("Shake the magic!");
        lcd.setCursor(0,1);
        lcd.print("LaunchPad!");
    }
    /* reset startTime. Wait for next 100ms before checking accel again.*/
    startTime = millis();
}

void displayPrompt() {
    lcd.clear(); // clear display to show new random response!
    lcd.setCursor(0,0); // reset cursor
    responses="Shake the magic!",
    lcd.print(responses);
    lcd.print(responses);
    response="LaunchPad!";
    lcd.setCursor(0,1); // reposition cursor to next row
    lcd.print(response);
}
Once you have the code in your LaunchPad, compile, download and run the code! Then, think of a question for our LaunchPad to answer and give it a shake!

**Challenges:**

- Using the `millis()` function and your new SPI skills, can you create a stopwatch using the LaunchPad? Try using the pushbuttons on the LaunchPad itself to function as the start and stop button of your stopwatch.
- Use the potentiometer as a dial to create a tea timer, while the LCD module displays the amount of time left until the buzzer will alarm. Based on the position of the potentiometer, set the amount of time your LaunchPad should countdown from. Once the potentiometer is positioned for the right time, use the onboard pushbutton on the LaunchPad as a button to start and pause the countdown. Once the timer has elapsed and counted down to zero, buzz the buzzer!

### 10.13 Other Languages

We’ve spent quite a lot of time going through two types of serial communication protocols that our LaunchPad is capable of—UART and SPI. Not too shabby! But wait, there are even more languages in our LaunchPad’s repertoire. One serial communication protocol that we won’t go into much detail in this book is I2C, or Inter-Integrated Circuit. This is a serial communication module that is similar to SPI in that it is a synchronous interface that requires a shared clock signal between all communicating parties. One major difference of I2C, however, is that it is only a two-wire protocol, unlike SPI which is primarily a four-wire interface.

Despite having just two wires, I2C is capable of enabling multislave communication with a Master, much like SPI. One main difference, however, is that SPI was able to individually communicate with each slave using individual Slave Select lines. I2C accomplishes this in a different way. Instead, our LaunchPad can individually communicate with each slave through I2C by sending out a slave “address,” or name before the actual command or payload. In addition to addressing each slave, the Master in the conversation can also specify if it wants to write (or talk) to a slave device or read from the slave device (aka listen).

I2C is capable of bi-directional communication and includes just two wires. One wire is to drive the clock signal (SCL) and the second wire is the data line (SDA). Once again, the Master drives the communication in this protocol.

### 10.14 Wrapping Up The Languages Of Launchpad

Our LaunchPad is a sociable device, capable of talking to external components such as LCDs, sensors, or other LaunchPads through various languages or serial communication protocols. We covered UART, SPI, and introduced I2C. We also learned how libraries can help us abstract away from the lower level communication protocols. This enables developers to focus more on the function and actions of our LaunchPad projects.