Demonstration: (NOTE: these change based on the lab - enter the demo tasks specified in each lab)

1. Demonstrate the series commands getting executed with respect to red and green LEDs (Requirements 1, 2, 3, 8, and 9)

2. Demonstrate the printing of acknowledgment or errors when the commands are executed (Requirements 4 and 5)

3. Demonstrate the printing of temperature and accelerometer reading when interrupt driven push buttons are pressed. (Requirements 6, 7, 8, and 10)

Requirements:

1. When the command SR1 n R is typed on the serial communication console, the red LED (J4.39) should blink based on the series an = 2an-1 + an-2 where a1 = 1 and a2 = 1. (Example values of n: if n = 1 then 1 blink, if n = 2 then 1 blink, if n = 3 then 3 blinks, and so on)

2. When the command SR2 n G is typed on the serial communication console, the green LED (J4.38) should blink based on the series an = an-1 + an-2 where a1 = 1 and a2 = 1.

3. The blinking should not be accomplished using delay functions

4. When a command is sent through the serial communication console, the board should acknowledge by sending “Executed” to the serial communication console after the command is executed

5. Any command that is not recognized by the board should be acknowledged by printing “Invalid Command”

6. Display the object temperature on the serial console when the push button S1 (J4.33) is pressed

7. Display the accelerometer (x, y, and z axis) values when the push button S2 (J4.32) is pressed

8. Only one command or operation can be given or performed at a time and the end of a command on the serial console is determined by pressing the enter key on the keyboard

9. Make sure every command is entered on a new line and every response message should be displayed on a new line

10. Make sure the push button inputs (S1 (J4.33) and S2 (J4.32)) are configured as interrupt-based inputs.

Learning Objectives:
This lab introduces serial communication, temperature sensor interfacing, accelerometer interfacing, and interrupts using the Ti MSP432 board and the booster pack MKII.

General Steps:
1. Setup and run the example codes after reading the MK II setup document posted on Canvas
2. Test examples AccelerometerSerial and TMP006 from File → Examples → 09.EducationalBP MKII.
3. Use the serial functions in this link to process the serial input commands
4. Make sure to setup the ADC resolution to 12 bits when using the accelerometer
5. Make sure all the push button inputs (S1 (J4.33) and S2 (J4.32)) are configured as interrupt-based inputs. Refer to the link for more information.
6. Using the example codes, complete the lab to meet all the requirements
7. Record your observations in the report.

Detailed Steps:

```
if(Serial.available() > 0 && readflag == 0 ) //input read
{
  Serial_series = Serial.readStringUntil(' '); // first command
  Serial_n = Serial.parseInt();                // second command
  Serial_led = Serial.readStringUntil('\n');   // third command
  Serial_led.trim(); // removes spaces
  enable = 1; // enable toggle
  readflag == 1; // disables new commands until finished
}
```

This is my input command reader, it takes in a command by user input, and looks for three things, a string, int, and string(but it requires only a char when it checks later). The program stores these values into the variables listed in the snippet. The flag, “enable” is thrown up to allow the case statements to acknowledge a new command has been entered to be completed. readflag is thrown when a new command is input, this will reject new commands from occurring during an uncompleted cycle, fulfils only one command at a time requirement.

```
if((Serial_series == "SR1") && (Serial_n > 0) && (Serial_led == "R") && (enable == 1)) // Case 1
{

      blink = series1(Serial_n); // Series 1} // if command instruction completes resets flags and counter
      currentMillis = millis();

      if(currentMillis - previousMillis > interval)
      {
          blinkLed(ledRed,ledStateRed);
          previousMillis = currentMillis;
           if(ledStateRed == LOW){cntl++;}        // cntl counts the cycles or in this case blinks
           if(cntl >= blink){enable = 0; cntl=0; readflag=0; Serial.println("Executed"); // prints only on last cycle of instruction
           }

      } // closes Millis()

}
```

This is my case statement, there are two case statements that are nearly identical except for the command, led color, and series. Case 1 is only entered if the command parsed matches the conditional checks AND the flag “enable” is thrown so it can reenter the state to finish cycling through the blinks.

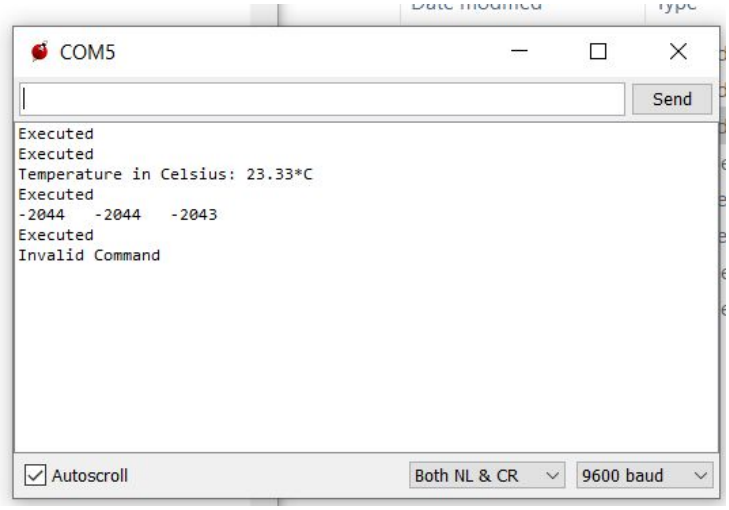
The N value the user inputs gets fed through the Series1 function and returns into the blink variable. This blink variable is the number blinks for the led to blink, I am using it as a max value for my counter to count to.

Once the led blinks its state is checked when its in the LOW state, my counter cntr1 increments to keep track of a completed blink. No delays, while loops, or for loops were used. Each blink will occur when the loop() cycles through. The millis() is used to control blink timing.

```
attachInterrupt(buttonTwo, accel, LOW);

attachInterrupt(buttonOne, tempr, LOW);
if (! tmp006.begin(TMP006_CFG_8SAMPLE))
  {
    Serial.println("No sensor found");
    while (1);
  }
}
```

* Interrupt settings



Observations:
Some important observations while completing/testing this lab were how many traditional programmers rely on dead loops to complete tasks. The number of checks required to do things efficiently instead of waiting for things to complete.

Summary:
In this lab, we learned how to implement a task to be completed without deadlocking the processor, this allows you the flexibility if needed to loop a task we could utilize the main loop to loop instead of a dead loop inside the loop waiting to complete. Learning to program efficiently maximized clock cycles that were already going to occur. I also learned how to implement the interrupt feature instead of constantly polling.