Lab 6

Using the UART, PSI, I2C, TPM3 Temperature Module and an LCD

Keith Bova

EECE 344L Microprocessors Fall 2022

Laboratory 6
Using the UART, SPI, I2C, TMP3 Temperature Module and an LCD Display

Due Date: November 14, 2022 Name:_____

Name: Keith Bova

Points: 100 Points
You may work individually, or in pairs and submit a joint report.

Objective: The purpose of this laboratory is to develop a control system for a power generation system using a TM4C123GH6PM microcontroller with: an LCD Display, a temperature PMOD module, an 8 LED PMOD module, and a terminal connected to the MX-7 using a UART. The LCD display, terminal video display and LEDs will provide visual information to the operator. You should be able to use software components from previous labs. SPI and I2C sample codes are posted on Canvas.

Activities: For this assignment, the power output level of the system will be selected using the 2 push button switches on the LaunchPad board. Each press of SW1 (on the left) will increase the power level and each press of SW2 (on the right) will decrease the power level. Be sure to introduce delays so that one button press does not result in multiple increases or decreases in power level.

The selected power level will be indicated on the 8-LED module. For each press of the switches, the number of LEDs that are lit will increase by one or decrease by one, as appropriate. The LEDs correspond to the selected power level; a 1 input level will be indicated with one lit LED and for power level eight, all of the LEDs will be lit on the LED module. In addition, a message will be displayed on the LCD module which says: POWER LEVEL: x, where the value of x ranges from 0 to 8.

The ambient temperature will be measured using the TMP3 module. The ambient temperature will be transmitted via the UART to the terminal display and to the LCD module via the SSI interface. *Hint: on the terminal, a CR character moves the cursor to the start of the line, and an old value can be overwritten with an updated value. Also, remember that the LCD display and the terminal will be expecting ASCII characters.*

You will display the temperature in degrees Fahrenheit. (The module outputs temperature in degrees Celsius.) Typical room temperature is about 73°F. When the temperature reaches 80°F, you will display a flashing message on the LCD Display that says: "Overheated!". (You can cause the temperature to reach this temperature by placing your finger on the temperature sensing integrated circuit on the TMP3 module.) You will also transmit a message to the terminal that records the overheating followed by a LF and CR, so that the message is not

Paul Joseph 11/14/22

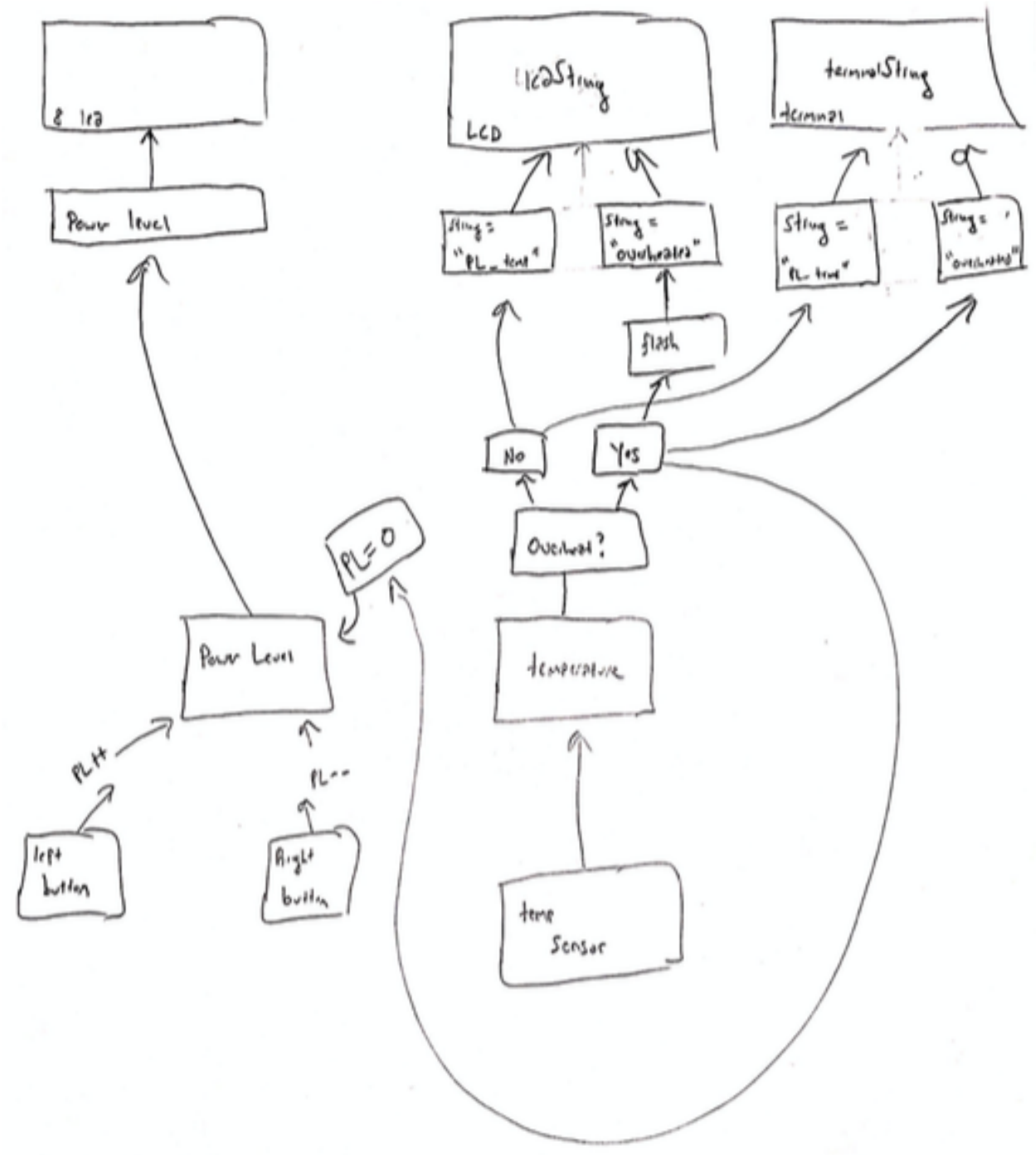
ECE344l Lab 6

Introduction:

The purpose of this lab is to familiarize students with the use of the GPIO, UART and I2C on the Texas Instruments TM4C123GH6PM microcontroller. Students will write a program that implements a basic power control system.

Body:

This program uses a baudrate of 9600 to communicate with the TM4C123GH6PM. The design of the power control system follows the block diagram, as shown below:



The system collects input from the two buttons on the microcontroller, and from the temperature sensor. The system displays output on the digilent 8led modue, the terraterm terminal, and the lcd display. Input from the buttons increments and decrements the current power state. If the input from

the temperature sensor exceeds 80 degrees F, the current power state gets reset to zero. The 8 led module displays the current power state. The lcd displays the current power state, temperature, and flashes if the system is overheating. The Terraterm displays the current power state, temperature, and if the system is overheating.

Pseudocode:

The block diagram for the project can be implemented as follows:

```
getButtonInput()
if(leftButtonIsPressed):
    if(currentPowerLevel + 1 < 8):
        currentPowerLevel += 1
if(rightButtonIsPressed):
    if(currentPowerLevel -1 >= 0):
        currentPowerLevel -= 1

checkIfSystemIsOverheating()

if(systemIsOverheating):
    currentPowerLevel = 0
    sendAlert()

displaySystemStatusTo8LedModule()
displaySystemStatusToLcdDisplay()
displaySystemStatusToTerraterm()
```

Source Code:

```
/*****************************************************************************/
/* ECE 344L - Microprocessors - Fall 2022      */
/*            */
/*            */
/* lab6.c     */
/*            */
/*            */
/*****************************************************************************/
/* Author(s):  Keith Bova      */
/*                                          */
/*****************************************************************************/
/*   Detailed File Description:        */
/*  This program implements a basic power control system  */
```

```
/*      */
/*            */
/*************************************************************************/
/*  Revision History:  11/14/22  */
/*        */
/*            */
/*************************************************************************/

#include <stdint.h>
#include <stdbool.h>                                  // needed for
compatibility
#include <stddef.h>
#include <string.h>
#include <stdio.h>
#include <tm4c123gh6pm.h>

#define SLAVE_ADDR 0x4F     /* 0100 1111 */
#define DELAY_VALUE 0xF9F           // 0xF9F = 1 mSec delay at 4 MHz
#define SYSDIV2 4

/* ************************ Prototypes ***************************** */
void I2C1_init(void);
char I2C1_byteWrite(int slaveAddr, char data);
char I2C1_burstWrite(int slaveAddr, int byteCount, char* data);
char I2C1_read(int slaveAddr, int byteCount, char* data);
void init_SSI0(void);
void SSI0Write(unsigned char data);
void putsSPI0(size_t buflen, char * buffer);
void SysTick_init(void);
void SysTick_mSecDelay(uint32_t delay);

/* 1 mSec Time delay using busy wait.   */
void SysTick_mSecDelay(uint32_t delay){
  uint32_t i;
  for(i=0; i<delay; i++){
    NVIC_ST_RELOAD_R = DELAY_VALUE;     // wait one cycle = DELAY_VALUE
    NVIC_ST_CURRENT_R = 0;
    while((NVIC_ST_CTRL_R & 0x00010000) == 0){  };
  }
}


/* initialize I2C1 as master and the port pins */
void I2C1_init(void)
{
```

```
    SYSCTL_RCGCI2C_R |= 0x02;           // enable clock to I2C1
    SYSCTL_RCGCGPIO_R |= 0x01;          // enable clock to GPIOA

    /* PORTA 7, 6 for I2C1 */
    GPIO_PORTA_AFSEL_R |= 0xC0;       /* PORTA 7, 6 for I2C1 */
    GPIO_PORTA_PCTL_R &= ~0xFF000000; /* PORTA 7, 6 for I2C1 */
    GPIO_PORTA_PCTL_R |= 0x33000000;
    GPIO_PORTA_DEN_R |= 0xC0;         /* PORTA 7, 6 as digital pins */
    GPIO_PORTA_ODR_R |= 0x80;         /* PORTA 7 as open drain */

    I2C1_MCR_R = 0x10;           /* master mode */
    I2C1_MTPR_R = 39;             /* 100 kHz @ 80 MHz */
}

/**************************************************************************** */
/* Wait until I2C master is not busy and return error code */
/* If there is no error, return 0 */
static int I2C_wait_till_done(void)
{
    while(I2C1_MCS_R & 1);   /* wait until I2C master is not busy */
    return I2C1_MCS_R & 0xE; /* return I2C error code */
}

/**************************************************************************** */

/* Write one byte only */
/* byte write: S-(saddr+w)-ACK-maddr-ACK-data-ACK-P */
char I2C1_byteWrite(int slaveAddr, char data)
{
    char error;

    /* send slave address and starting address */
    I2C1_MSA_R = slaveAddr << 1;
    I2C1_MDR_R = data;
    I2C1_MCS_R = 7;                      /* S-(saddr+w)-ACK-maddr-ACK */

    error = I2C_wait_till_done();       /* wait until write is complete */
    if (error) return error;

    return 0;       /* no error */
}

/**************************************************************************** */

/* Use burst write to write multiple bytes to consecutive locations */
```

```
/* burst write: S-(saddr+w)-ACK-maddr-ACK-data-ACK-data-ACK-...-data-ACK-P */
char I2C1_burstWrite(int slaveAddr, int byteCount, char* data)
{
    char error;
    if (byteCount <= 0)
        return -1;                  /* no write was performed */

    /* send slave address and starting address */
    I2C1_MSA_R = slaveAddr << 1;
    I2C1_MDR_R = *data++;
    I2C1_MCS_R = 3;                  /* S-(saddr+w)-ACK-maddr-ACK */
    byteCount--;                    // send first char with start & ACK
    error = I2C_wait_till_done();   /* wait until write is complete */
    if (error) return error;

    /* send remaining data one byte at a time */
    while (byteCount > 1)
    {
        I2C1_MDR_R = *data++;             /* write the next byte */
        I2C1_MCS_R = 1;                   /* -data-ACK by slave- */
        error = I2C_wait_till_done();
        if (error) return error;
        byteCount--;
    }

    /* send last byte and a STOP */
    I2C1_MDR_R = *data++;                 /* write the last byte */
    I2C1_MCS_R = 5;                       /* -data-ACK-P */
    error = I2C_wait_till_done();
    while(I2C1_MCS_R & 0x40);             /* wait until bus is not busy */
    if (error) return error;

    return 0;       /* no error */
}

/*************************************************************** */

/* Read memory */
/* read: S-(saddr+w)-ACK-maddr-ACK-R-(saddr+r)-ACK-data-ACK-data-ACK-...-
data-NACK-P */
char I2C1_read(int slaveAddr, int byteCount, char* data)
{
    char error;

    if (byteCount <= 0)
```

```
        return -1;         /* no read was performed */

    /* configure bus from for read, send start with slave addr */
    I2C1_MSA_R = (slaveAddr << 1) + 1;   /* restart: -R-(saddr+r)-ACK */

    if (byteCount == 1)             /* if last byte, don't ack */
        I2C1_MCS_R = 7;              /* -data-NACK-P */
    else                            /* else ack */
        I2C1_MCS_R = 0xB;            /* -data-ACK- */
    error = I2C_wait_till_done();
    if (error) return error;

    *data++ = I2C1_MDR_R;            /* store the data received */

    if (--byteCount == 0)           /* if single byte read, done */
    {
        while(I2C1_MCS_R & 0x40);    /* wait until bus is not busy */
        return 0;       /* no error */
    }

    /* read the rest of the bytes */
    while (byteCount > 1)
    {
        I2C1_MCS_R = 9;              /* -data-ACK- */
        error = I2C_wait_till_done();
        if (error) return error;
        byteCount--;
        *data++ = I2C1_MDR_R;        /* store data received */
    }

    I2C1_MCS_R = 5;                  /* -data-NACK-P */
    error = I2C_wait_till_done();
    *data = I2C1_MDR_R;              /* store data received */
    while(I2C1_MCS_R & 0x40);        /* wait until bus is not busy */

    return 0;       /* no error */
}

struct ButtonInput
{
    bool leftSwitchPressed;
    bool    rightSwitchPressed;
    bool bothSwitchesPressed;
    bool nothingPressed;
};
```

```
void getInputFromButtons(struct ButtonInput * myInput)
{
    int input = GPIO_PORTF_DATA_R & 0x11;  // raw input bits - others masked

    switch (input)
        {
            case 0:                          // sw1 & sw2 pressed
                myInput->bothSwitchesPressed = true;
        break;
            case 1:                          // sw1 pressed
                myInput->leftSwitchPressed = true;
              break;
      case 16:                        // sw2 pressed
                myInput->rightSwitchPressed = true;
                break;
      default:
                myInput->nothingPressed = true;
    }
    SysTick_mSecDelay(20);
}

void resetButtonsToZero(struct ButtonInput * myInput)
{
    myInput->leftSwitchPressed = false;
    myInput->rightSwitchPressed = false;
    myInput->bothSwitchesPressed = false;
    myInput->nothingPressed = false;
}




void init_SSI0(void)
{
    SYSCTL_RCGCSSI_R |= 1;          // enable clock to SSI0
    SYSCTL_RCGCGPIO_R |= 0x1;       // Enable clock to PORT A

    /* configure PORTA 2..5 for SSI0 clock, FS, Tx & Rx */
    GPIO_PORTA_AMSEL_R = 0;             // turn off analog function
    GPIO_PORTA_DEN_R |= 0x3C;       // make PA2..PA5 digital
    GPIO_PORTA_AFSEL_R = 0x3C;      // make PA2.. PA5 alternate function
    GPIO_PORTA_PCTL_R = 0x00222200;   // configure PA2..PA5 as SSI0

    /* SPI Master, POL = 0, PHA = 0, SysClk = 80 MHz, 8 bit data */
```

```
    SSI0_CR1_R = 0;          // disable SSI and make it master
    SSI0_CC_R = 0;           // use system clock
    SSI0_CPSR_R = 0x64;      // prescaler divided by 100
    SSI0_CR0_R = 0x0707;     // 800KHz/8 = SSI clock, SPI mode, 8 bit data
    SSI0_CR1_R |= 2;         // enable SSI0
}

/* This function writes one byte to a slave device via the SSI0 interface
*/
void SSI0Write(unsigned char data)
{
    while((SSI0_SR_R & 2) == 0); // wait until FIFO not full
    SSI0_DR_R = data;            // transmit high byte
    while(SSI0_SR_R & 0x10);     // wait until transmit complete
}

/*  ------------------------------------------------------------------------
--     */
/* This function writes the characters in a string to the SPI 0 interface
*/
/* The input arguments are a character count and the start address of the
buffer    */
/* As SS01Write() waits for the FIFO buffer to not be full, no waiting is
*/
/* Needed in this routine.
*/
/*  ------------------------------------------------------------------------
--     */
 void putsSPI0(size_t buflen, char * buffer) {
    char * i;
    for (i = buffer; i < (buffer + buflen); i++)
        {
         SSI0Write(*i); /* write a character */
        }
};

void SysTick_delay(void)
{
    NVIC_ST_CURRENT_R = 0;                      // 1.   clear CURRENT by
writing any value
    while((NVIC_ST_CTRL_R & 0x00010000)==0)     // 2. wait for count flag to
be set
        {

        }
```

```
}

void SysTick_init(void){
    NVIC_ST_CTRL_R = 0;                 // 1. disable SysTick before
configuring
    NVIC_ST_RELOAD_R = DELAY_VALUE;     // 2. set to desired delay value (1
mSec)
    NVIC_ST_CURRENT_R = 0;              // 3. clear CURRENT by writing any
value
    NVIC_ST_CTRL_R &= ~0x00000004;      // 4. set clock to POSC/4
    NVIC_ST_CTRL_R |= 0x00000001;       // 5. enable SysTick timer
}




void configureSystemClockFor80MhzOperation()
{
    SYSCTL_RCC2_R |= 0x80000000;
// Use RCC2
  SYSCTL_RCC2_R |= 0x00000800;
// Bypass PLL while initializing it

// Select crystal value and osc source
  SYSCTL_RCC_R = (SYSCTL_RCC_R & ~0x000007C0)
// clear bits 10-6
                + 0x00000540;
// 10101 configure for 16Mhz XTL
  SYSCTL_RCC2_R &= ~0x00000070;
//  Use main oscillator
  SYSCTL_RCC2_R &= ~0x00002000;
//  Activate PLL - clear PWRDN
  SYSCTL_RCC2_R |= 0x40000000;
//  Set system divider
  SYSCTL_RCC2_R = (SYSCTL_RCC2_R & ~0x1FC00000) +(SYSDIV2<<22);

}

int convertDegreesCelsiusToDegreesFahrenheit(int inputTemperature)
{
    return ((inputTemperature * 9)/ 5) + 32;
}


/**********************Global Variables ****************************** */
```

```
    char i2c_data[4];           // buffer for date read from or written to
I2C
    char *i2c_char_p = &i2c_data[0];

/*********************************************************************** */

int getCurrentTemperatureFromSensorInFahrenheit(const char slaveAddress)
{
    int raw_temp;
// raw temp data in int form
    short int a, b, currentTempInDegCelsius;
// temperature variables
    I2C1_read(slaveAddress, 2, i2c_char_p);
  a = (short int) i2c_data[0];
// cast bytes to 16 bits
    b = (short int) i2c_data[1];
    raw_temp = b | (a << 8);
// combine bytes to 9 bit result
    currentTempInDegCelsius = raw_temp >>
8;
// shift out ls 7 bits of 0 divide by 256 -> 0 Deg C
  return convertDegreesCelsiusToDegreesFahrenheit(currentTempInDegCelsius);
}

void configureI2C()
{
    i2c_data[0] = 0;
// select register 0 on TMP3 module
}

void enablePhaseLockLoopByClearingBypass()
{
    SYSCTL_RCC2_R &= ~0x00000800;
//  Enable PLL by clearing BYPASS
}

void waitForPhaseLockLoopToLock()
{
    while((SYSCTL_RIS_R&0x00000040)==0)
    {


};
//  Wait for PLL to lock - poll PLLRIS
}
```

```
void displayStringToLcdModule(char * stringToDisplayOnModule)
{
    SSI0Write(0x1b);
// Display reset - write an escape character
    char * buffer = {"this is a command string"};
    buffer="[j";
// command sequence for clear screen

// and reset cursor
    putsSPI0(2,buffer);
// write out string
    SysTick_mSecDelay(10);
// approximately .01 s

  putsSPI0(strlen(stringToDisplayOnModule),stringToDisplayOnModule);
// write out string
}




void displayDecimalNumberOnGpioBoardInBinary(uint8_t
decimalToDisplayInBinary)
{
    const uint32_t decimalToDisplayInBinaryMod256 = decimalToDisplayInBinary
% 256;
    GPIO_PORTE_DATA_R = decimalToDisplayInBinary & 0xF;          // lower 4
count bits - PE0..PE3
    GPIO_PORTD_DATA_R = (decimalToDisplayInBinary & 0xF0) >> 4;  // upper 4
count bits - PD0..PD3
}

void UART2Tx(char c)
{
  while((UART2_FR_R & 0x20) != 0);     /* wait until Tx buffer not full */
  UART2_DR_R = c;                      /* before giving it another byte */
}

void displayStringToConsole(char * stringToDisplay)
{
    char * i;
    const int numberOfElementsInArray = strlen(stringToDisplay);
```

```
    for(i = stringToDisplay; i < (stringToDisplay + numberOfElementsInArray);
i++)
    {
        UART2Tx(*i);

    }
}


void initializeUartAndGpioForUseWithTerraterm()
{
    SYSCTL_RCGCUART_R |= 0x04;  // provide clock to UART2
  SYSCTL_RCGCGPIO_R |= 0x8;  // Enable clock to PORTD


    UART2_CTL_R = 0;         // disable UART2
  UART2_IBRD_R = 520;      // 80MHz/16=5MHz, 5MHz/520=>9600 baud rate
  UART2_FBRD_R = 53;       // fraction part, .8333*64 + 0.5
  UART2_CC_R = 0;          // use system clock
  UART2_LCRH_R = 0x60;     // 8-bit, no parity, 1-stop bit
  UART2_CTL_R = 0x301;     // enable UART2, TXE, RXE

    /* UART2 TX5 and RX5 use PC7 and PC6. Set them up. */
  GPIO_PORTD_LOCK_R = 0x4C4F434B;   //  Unlock the port register
  GPIO_PORTD_CR_R = 0xFF;           // Allow changes to PD7..PD0

  GPIO_PORTD_DEN_R |= 0xC0;          /* make PD7, PD6 as digital */
  GPIO_PORTD_AMSEL_R = 0;           /* turn off analog function */
  GPIO_PORTD_AFSEL_R = 0xC0;        /* use PD7, PD6 alternate function */
  GPIO_PORTD_PCTL_R = 0x11000000;   /* configure PD7, PD6 for UART2 */
}

void initializeGPIOPortsFor8LedModule()
{
    SYSCTL_RCGCGPIO_R |= 0x38;  // activate port D,E & F clocks

  GPIO_PORTF_DIR_R |= 0x04;   // make PF2 out (built-in blue LED)
    GPIO_PORTE_DIR_R |= 0x0F;   // make PE0..PE3 out
  GPIO_PORTD_DIR_R |= 0x0F;   // make PD0..PD3 out

  GPIO_PORTF_AFSEL_R &= ~0x04;// disable alt funct on PF2
    GPIO_PORTE_AFSEL_R &= ~0x0F;// disable alt funct on PE0..PE3
  GPIO_PORTD_AFSEL_R &= ~0x0F;// disable alt funct on PD0..PD3

  GPIO_PORTF_DEN_R |= 0x04;   // enable digital I/O on PF2
```

```
    GPIO_PORTE_DEN_R |= 0x0F;   // enable digital I/O on PE0..PE3
  GPIO_PORTD_DEN_R |= 0x0F;   // enable digital I/O on PD0..PD3

                              // configure PF2 as GPIO (Selectively - others
left unchanged)
  GPIO_PORTF_PCTL_R = (GPIO_PORTF_PCTL_R&0xFFFFF0FF)+0x00000000;
                                                  // configure PE0..PE3 as GPIO
(Selectively - others left unchanged)
    GPIO_PORTE_PCTL_R = (GPIO_PORTF_PCTL_R&0xFFFF0000)+0x00000000;
                         // configure PD0..PD3 as GPIO (Selectively - others
left unchanged)
  GPIO_PORTD_PCTL_R = (GPIO_PORTD_PCTL_R&0xFFFF0000)+0x00000000;

  GPIO_PORTF_AMSEL_R = 0;     // disable analog functionality on PF
    GPIO_PORTE_AMSEL_R = 0;     // disable analog functionality on PE
  GPIO_PORTD_AMSEL_R = 0;     // disable analog functionality on PD

}

bool checkIfSystemIsOverheating(const int currentTemperatureInFahrenheit)
{
    const int temperatureThreshold = 80;
    if(currentTemperatureInFahrenheit > temperatureThreshold)
    {
        return true;
    }
    return false;
}

int main(void)
{

    int len;
// string length
    int currentTemperatureInFahrenheit;
// temperature data
    char i;
  char buffer[40] = {"hello world - it's a fine day   "};
    char * cbuffer = {"this is a command string"};                      //
buffer for commands
    char slaveAddress = SLAVE_ADDR;
// I2C address of TMP module
    int currentPowerLevel = 0;
    struct ButtonInput inputFromButtons;
    int input;
```

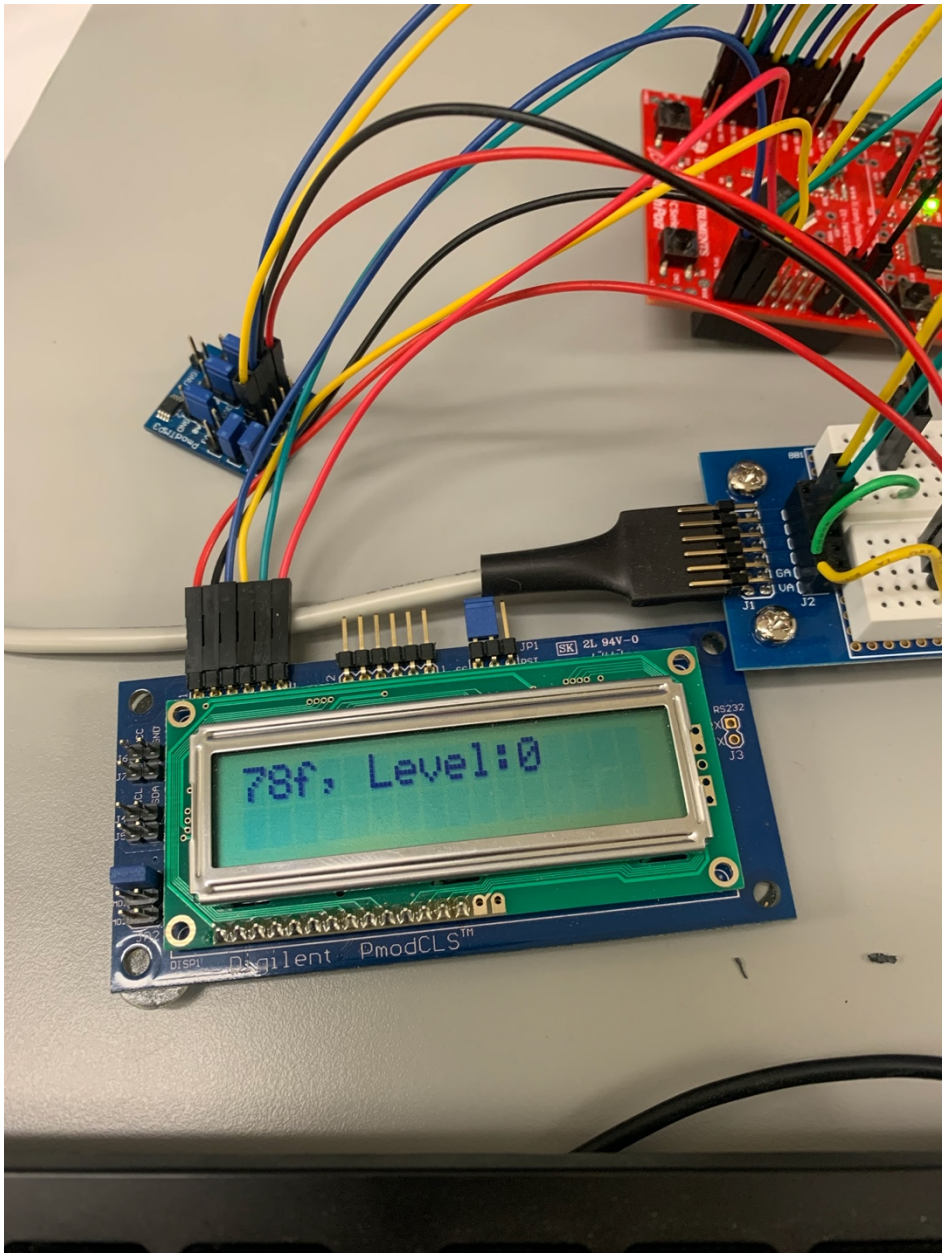
```
    configureSystemClockFor80MhzOperation();
    waitForPhaseLockLoopToLock();
    enablePhaseLockLoopByClearingBypass();
    SysTick_init();
// initialize SysTick timer

    initializeUartAndGpioForUseWithTerraterm();

    SysTick_delay();
/* wait for output line to stabilize */



    init_SSI0();
// Configure and initialize SSI1 interface
  I2C1_init();
// Configure & Initialize I2C1 interface
    configureI2C();


    /* command TMP3 to read from register 0 for desired temperature format */

  I2C1_byteWrite(slaveAddress, i2c_data[0]);
// configuration command

    char lcdDisplayMessage[16], consoleDisplayMessage[30];

    SYSCTL_RCGCGPIO_R |= 0x00000020; // (a) activate clock for port F

    /* PORTF0 has special function, need to unlock to modify */
    GPIO_PORTF_LOCK_R = 0x4C4F434B;   /* unlock commit register */
    GPIO_PORTF_CR_R = 0x01;           /* make PORTF0 configurable */
    GPIO_PORTF_LOCK_R = 0;            /* lock commit register */
    /* configure PORTF for switch input and LED output */
    GPIO_PORTF_DIR_R &= ~0x11;         /* make PORTF 4, 0 input for switch */
    GPIO_PORTF_DIR_R |= 0x0E;          /* make PORTF3, 2, 1 output for LEDs
*/
        GPIO_PORTF_AFSEL_R &= ~0x11;     // disable alt funct on PF4, PF0
    GPIO_PORTF_DEN_R |= 0x1F;          /* make PORTF4-0 digital pins */
    GPIO_PORTF_PUR_R |= 0x11;          /* enable pull up for PORTF4, 0 */
    GPIO_PORTF_DATA_R = 0; // lights out!

        initializeGPIOPortsFor8LedModule();
```

```
    while(1)
    {
        input = GPIO_PORTF_DATA_R & 0x11;  // raw input bits - others masked
        currentTemperatureInFahrenheit =
getCurrentTemperatureFromSensorInFahrenheit(slaveAddress);
        switch (input){
        case 0: // sw1 & sw2
                GPIO_PORTF_DATA_R = 4; // light blue


                break;
        case 1:                         // sw1 pressed
                GPIO_PORTF_DATA_R = 8;      // light red
                currentPowerLevel++;

                break;
        case 16:                        // sw2 pressed
                GPIO_PORTF_DATA_R = 2;      // light green
                if(currentPowerLevel -1 >= 0)
                    {
                    currentPowerLevel--;
                    }
                    break;
        default:
                GPIO_PORTF_DATA_R = 0; // lights out!
        break;
        }


   SysTick_mSecDelay(10);
        //displayDecimalNumberOnGpioBoardInBinary(currentPowerLevel);
        sprintf(consoleDisplayMessage,"%df,
Level:%d",currentTemperatureInFahrenheit, currentPowerLevel );

        bool systemIsOverheating =
checkIfSystemIsOverheating(currentTemperatureInFahrenheit);
        if(systemIsOverheating)
        {
            currentPowerLevel = 0;
            sprintf(consoleDisplayMessage,"overheat                ");
        }

        displayDecimalNumberOnGpioBoardInBinary(1 << currentPowerLevel);
```

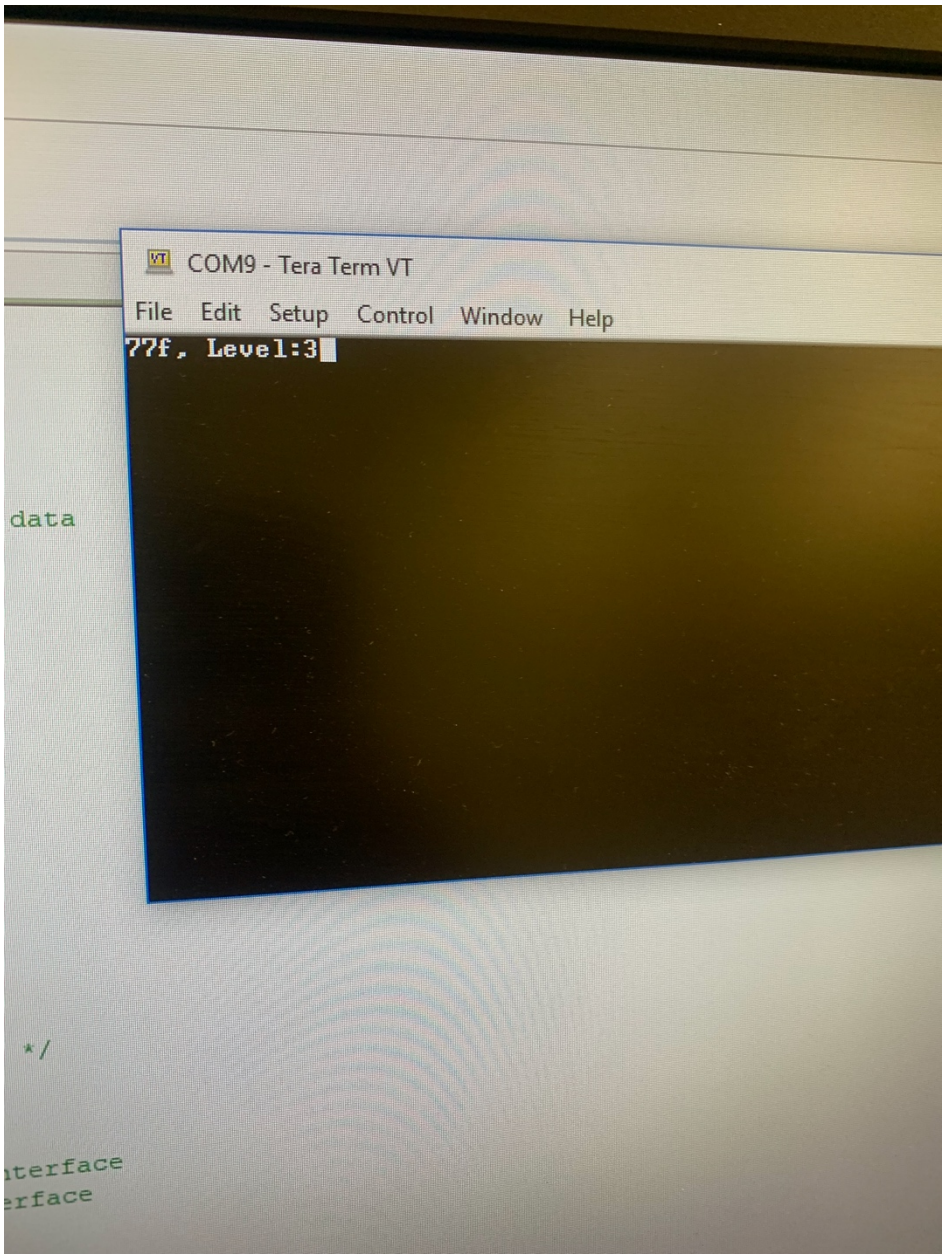
```
        sprintf(lcdDisplayMessage,"%df,
Level:%d",currentTemperatureInFahrenheit, currentPowerLevel );









        displayStringToLcdModule(lcdDisplayMessage);
        displayStringToConsole(consoleDisplayMessage);
        SysTick_mSecDelay(200);
// delay before reading new temperature
UART2Tx(0x0D);
    }




}
```

Testing:

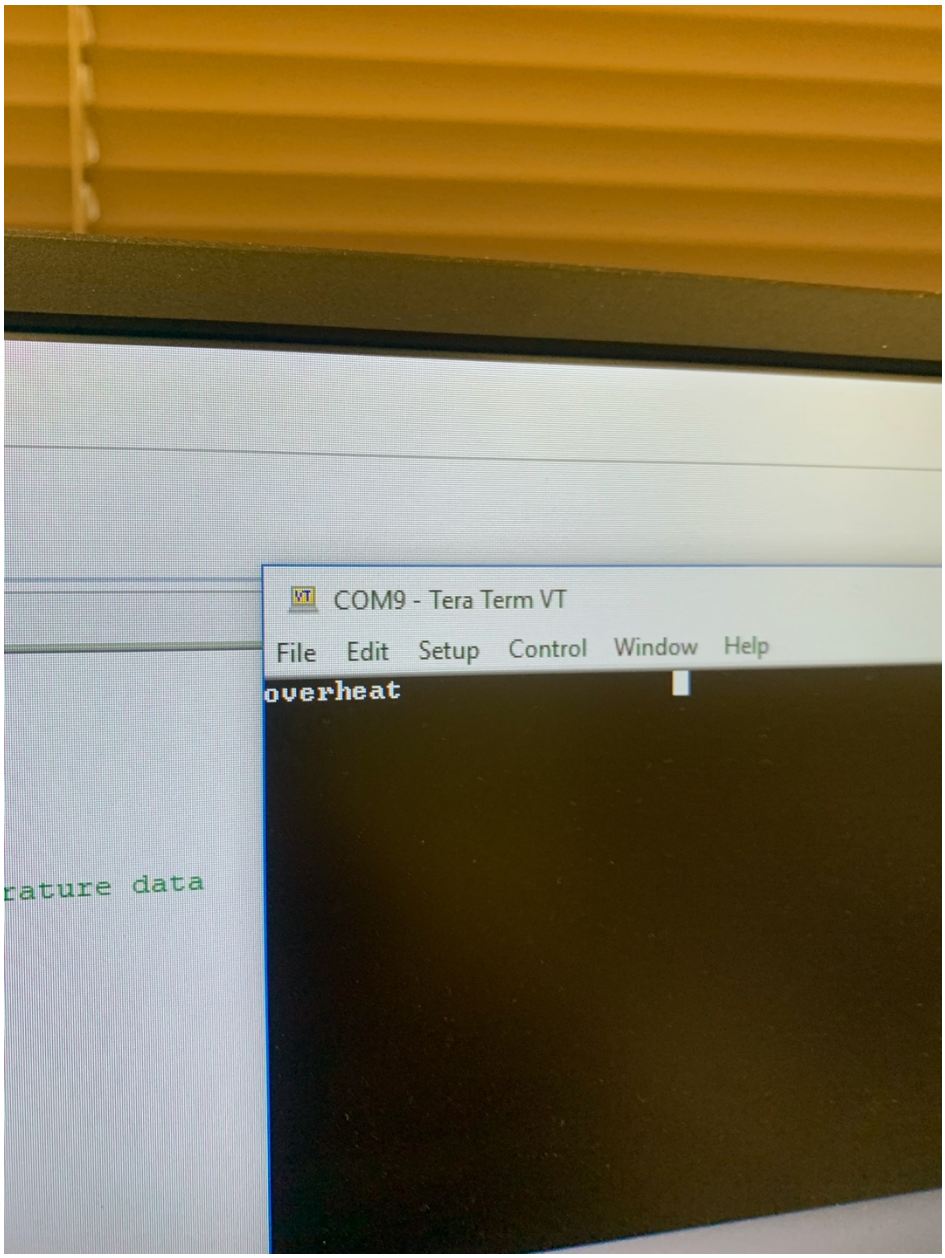
The program performed as expected. The LCD properly displayed the current temperature and power level:



The buttons properly incremented the power level:



And exceeding the temperature threshold reset the power level and displayed a message:



Conclusion:

The TM4C123GH6PM is a very capable device and can effectively handle input from multiple sources--in real time. This feedback control mechanism could be useful working with motors, or other devices that are digitally controlled and cannot excced a particular threshold.