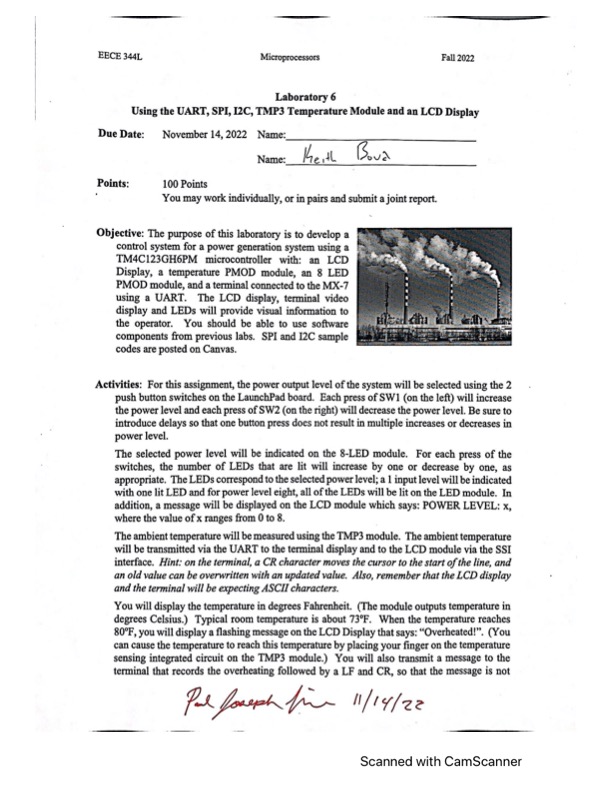
Graphical user interface, text

Description automatically generated



**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:

Diagram

Description automatically generated

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:

A picture containing text, electronics

Description automatically generated

The buttons properly incremented the power level:

A picture containing text, monitor, electronics, display

Description automatically generated

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

A picture containing text

Description automatically generated

**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.