ECE 372
Design Project 2, Part 1
Phil Nevins

Design Log Part 1

Entry Date 2/26/2023

The project instructions for Design Project 2, Part 1 and it seems very straight forward. I am reading through them and will notate important aspects below.

1. I2C uses 2 signal lines + ground, SCL, SDA
2. MSB is sent first
3. Address size is 7 bits
4. Data is transferred over an I2C bus serially in bytes
5. An I2C message may contain any number of bytes

To start a transmission, the master pulls the SDA line from H to L while the SCL line is H.
The master then pulses the SCL line and shifts out the data bits on SDA synchronously with the SCL pulses.
If the slave receives the 8 data bits correctly, it synchronously pulls the SDA line L as an acknowledge signal to the master.
If the slave needs time to process the received byte, it can hold the SCL line L to force the master to insert wait states.
When the slave releases the SCL line and its pulled H by the pull-up resistor, the master can send another byte.

Stop condition: the master allows the SDA line to go H while the SCL line is H.
Repeated start condition: the SDA line goes L while SCL line is H.

For any transmission on the bus, the master will first send out an address byte.
Each device on an I2C bus can be internally programmed with a 7-bit address.
The upper 7 bits of the first byte sent out by the master will contain the address of the slave that is to be written to or read from.
The LSB of this byte will be 0 for a write and 1 for a read.

After continuing to read through the rest of the instructions and handouts, here are the important aspects I will need to write the program.

The function we will use to enable us to write values to an address is this function:
HWREG(x) (*((volatile unsigned int *)(x)))

Using this, we can call:
HWREG(BASE_ADDRESS + OFFSET) ='value we want to write to "addr + offset" ' ;

Base addresses needed	
CM_PER_BASE	0x44E00000
CONTROL_BASE	0x44E10000
I2C2_BASE	0x4819C000

Registers Needed	Calculated Offsets
I2C2_SCL	0x954
I2C2_SDA	0x950
I2C_PSC	0xB0
I2C_SCL_L	0xB4
I2C_SCL_H	0xB8
I2C_CON	0xA4
I2C_IRQSTATUS_RAW	0x24
I2C_OA	0xA8
I2C_CNT	0x98
I2C_DATA	0x9C
I2C_SA	0xAC
I2C_SYSC	0x10
I2C_CLK	0x44

We will also need the table I made below for the initialization of the I2C

I2C Register	Value Needed	What does value do
SCL	0x32	Enable register
SDA	0x32	Enable register
I2C_CLK	0x02	Wake up I2C Clock
I2C_OA	0x00	Reset value
I2C_PSC	0x00	Calculated pre-scaler
I2C_SCL_L	0x08	Time L
I2C_SCL_H	0x0A	Time H
I2C_CON	0x8600	Configure I2C
I2C_SYSC	0x02	Reset Software
I2C_SA	0xE0	Set slave addr as 0xE0

Entry Date 2/27/2023

Now we can build our Low and High level algorithms

High Level Algorithm Part 1

```
Initialize and Define, global statements
        Initialize HWREG C command
        Define base addr
        Define reg offsets
        Define I2C values

Main
        Initialize I2C reg
        Call Functions
Function: Initiate
Function: Transfer Data
```

Low Level Algorithm Part 1

```
Initialize and Define, global statements
Initialize HWREG C command
        Define BASE addr
                CM_PER_BASE addr        0x44E00000
                CONTROL_BASE addr       0x44E10000
                I2C2_BASE addr          0x4819C000

        Define Register offsets
                I2C_SYSC                0x10
                I2C_IRQSTATUS_RAW       0x24
                I2C_IRQSTATUS           0x28
                I2C_CLK                 0x44
                I2C_CNT                 0x98
                I2C_CON                 0xA4

                I2C_PSC                 0xB0
                I2C_SCL_L               0xB4
                I2C_SCL_H               0xB8
                I2C_DATA                0x9c

                I2C2_SCL                0x950
                I2C2_SDA                0x954
```

```
Define global functions
        Initiate transfer on I2C
        Transmit data on I2C

Define Main Function
        Initialize I2C Registers using HWREG
 (CONTROL_BASE + SCL)              =0x32          Enable register
 (CONTROL_BASE + SDA)              =0x32          Enable register
 (CM_PER_BASE + I2C2_CLK)           =0x02         Wake up I2C Clock
 (I2C2_BASE + I2C_OA)              =0x00          Reset value
 (I2C2_BASE + I2C_PSC)             =0x03          Calculated pre-scaler
 (I2C2_BASE + I2C_SCL_L)           =0x08          Time L
 (I2C2_BASE + I2C_SCL_H)           =0x0A          Time H
 (I2C2_BASE + I2C_CON)             =0x8600        Configure I2C
(I2C2_BASE + I2C_SA)                = 0xE0          Set slave address as 0xE0
        Call Functions
                Initiate transfer function
                Transfer data function

Function to initialize transfer
        If statement to see if the bus is ready
                Write 0x01 to I2C_CON
                Call Transmit Data
        Else check bus again


Function to transfer data
        If statement to see if the bus is ready
                Load data to I2C_DATA
        Else return
```

Code Part 1

```
//Phil Nevins
//ECE 372 DP2 Part 1
//I2C2 driver

//Define statementsf
//C function to write to assembly addr
#define HWREG(x) (*((volatile unsigned int *)(x)))

//Base addr
#define CM_PER_BASE 0x44E00000
#define CONTROL_BASE 0x44E10000
#define I2C2_BASE 0x4819C000

//Offsets
#define I2C2_SCL          0x954
#define I2C2_SDA          0x950
#define I2C_PSC           0xB0
#define I2C_SCL_L         0xB4
#define I2C_SCL_H         0xB8
#define I2C_CON           0xA4
#define I2C_IRQSTATUS_RAW 0x24
#define I2C_OA            0xA8
#define I2C_CNT           0x98
#define I2C_DATA          0x9C
#define I2C_SA            0xAC
#define I2C_SYSC          0x10
#define I2C_CLK           0x44

//Functions
void INIT_I2C2();
void TRANSMIT(int address, int data);

int main(void)
{
    INIT_I2C2();
    TRANSMIT(ALL_OFF, 0x10);
    return 0;
}


void INIT_I2C2()
{
    HWREG(CM_PER_BASE + 0x44) = 0x2;             //Turn on I2C2 Clock Module
    HWREG(CONTROL_BASE + I2C2_SCL) = 0x32;      //Configure Pin 21 as I2C2_SCL
    HWREG(CONTROL_BASE + I2C2_SDA) = 0x32;    //Configure Pin 22 as I2C2_SDA
    HWREG(I2C2_BASE + I2C_SYSC) = 0x02;           //Software reset
    HWREG(I2C2_BASE + I2C_PSC) = 0x00;           //Configure Pre-Scale I2C2 register
    HWREG(I2C2_BASE + I2C_SCL_L) = 0x08;         //Configure the I2C low time
register
    HWREG(I2C2_BASE + I2C_SCL_H) = 0x0A;          //Configure the I2C high time
register
```

```
    HWREG(I2C2_BASE + I2C_OA) = 0x0;              //Configure I2C Own Address 
register
    HWREG(I2C2_BASE + I2C_SA) = 0xE0;             //Set slave address as 0xE0
    HWREG(I2C2_BASE + I2C_CON) = 0x8600;        //Configure I2C_CON Register
}

void TRANSMIT(int address, int data)
{
    HWREG(I2C2_BASE + I2C_CNT) = 0x02;     //configure I2C_CNT register with number 
of bytes to be transfered

    if(!(HWREG(I2C2_BASE + I2C_IRQSTATUS_RAW) & 0x1000))    // If the bus busy Bit 12 
= 0
    {
        HWREG(I2C2_BASE + I2C_CON) = 0x8603;        //Write 0x01 to CON register to 
initiate start transfer condition

        if((HWREG(I2C2_BASE + I2C_IRQSTATUS_RAW) & (0x10)))        // If XRDY bit 4 = 
1
        {
            HWREG(I2C2_BASE + I2C_DATA) = address;            //Load I2C Data reg 
with address to send out on SDA
            DELAY_LOOP(5000);

            HWREG(I2C2_BASE + I2C_DATA) = data;            //data to output
            DELAY_LOOP(5000);
        }

        else
        {

            TRANSMIT(address, data);    //jump back to top of send_out function and 
chekck bits 12 and 4 again)
        }
    }
    else
    {
        TRANSMIT(address, data);
    }
}
```

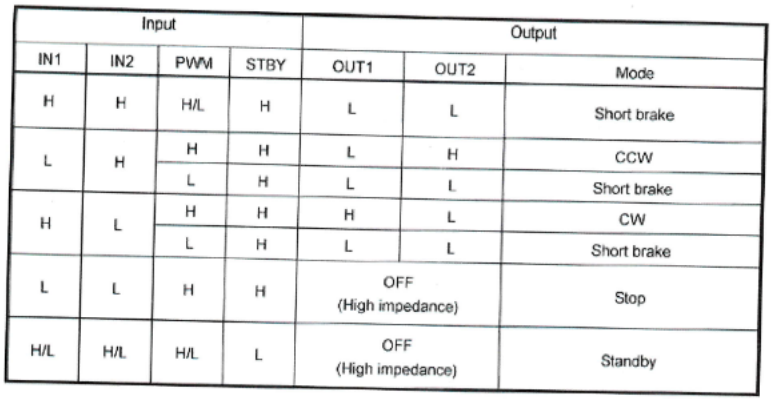
ECE 372
Design Project 2, Part 2
Phil Nevins

Design Log Part 2

Entry 3/5/2023

I am reading through the design project instructions, and it seems straight forward. We are tasked with using the I2C2 driver we made in Part 1 to step the motor 180 degrees when a button interrupt is pressed. We will utilize the button interrupt code we did in Project 1, using the “asm” command for C code.

Important details, gathered from the datasheets of the BBB, PCA9865 and stepper motor:



Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

(-> means connected to)
IC3 pins -> PCA9685 pins

AIN1 -> PWM4 (pin 10)
AIN2 -> PWM3 (pin 9)
PWMA -> PWM2 (Pin 8)

BIN1 -> PWM5 (pin 11)
BIN2 -> PWM6 (pin 12)
PWMB -> PWM7 (pin 13)

Initialize PCA9865

1. Set prescale mode for 1Khz

    To prescale, you need to be in Mode 2 (0x01) because the datasheet says Writes to PRE_SCALE register are blocked when SLEEP bit is logic 0 (MODE 1).

    Write 1Khz (0x05) to PRE_SCALE (0xFE)

2. Set for Totem Pole structure, non inverted

    Write to MODE 2 (0x01) the value to turn off INVRT (Output logic is not inverted, bit 4, default) and OUTDRV (Totem pole structure, bit 2, default), which is 00000100 (0x4). Since both of these settings are default, it should always be on but we will write to it just to ensure its correct.

3. Zero ALL_LED_OFF_H, so you just have write to LED_ON registers

    Write 0001 0000 (0x10) to ALL_LED_OFF_H (0xFD)

**Use 5000 for delay loop – stated in instructions. This works!

Initialize I2C2:

Action	Value
Turn on I2C2 Clock Module	0x2
Configure Pin 21 as I2C2_SCL	0x32
Configure Pin 22 as I2C2_SDA	0x32
Software reset	0x02
Configure Pre-Scale I2C2 register	0x00
Configure the I2C low time register	0x08
Configure the I2C high time register	0x0A
Configure I2C Own Address register	0x0
Set slave address	0xE0
Configure I2C_CON Register	0x8600

Initialization for slave addresses:

LED2_ON_H	0x0F	PWMA
LED2_OFF_H	0x11	PWMA
LED3_ON_H	0x13	BIN2
LED3_OFF_H	0x15	BIN2
LED4_ON_H	0x17	AIN2
LED4_OFF_H	0x19	AIN2
LED5_ON_H	0x1B	BIN1
LED5_OFF_H	0x1D	BIN1
LED6_ON_H	0x1F	AIN1
LED6_OFF_H	0x21	AIN1
LED7_ON_H	0x23	PWMB
LED7_OFF_H	0x25	PWMB
ALL_LED_ON_H	0xFB	All LED on
ALL_LED_OFF_H	0xFD	All LED off

PRE_SCALE	0xFE	Pre-Scale value for PCA
MODE_1	0x00	Mode 1 register
MODE_2	0x01	Mode 2 register

Functions needed:

```
void INIT_I2C2();
void INIT_PCA9865();
void STEP1();
void STEP2();
void STEP3();
void STEP4();
void DELAY_LOOP();
void TRANSMIT(int address, int data);
void TURN_OFF(void);
void INIT(void);
void WAIT_LOOP();
void INT_DIRECTOR();
void PASS_ON();
void BUTTON_SVC();
void STEP_LOOP();
void INIT_INTERRUPT();
void RESET_INTERRUPT();
```

Stack designation:|

```
volatile unsigned int SVC_STACK[1000];
volatile unsigned int IRQ_STACK[1000];
```

Entry Date 3/12/2023

Now that we have all of our information that is needed and our I2C driver built, we will write the low and high level algorithms for Part 2.

High Level Algorithm Part 2

```
Initialize and Define, global statements
    Initialize HWREG C command
    Define base addr
    Define reg offsets
    Define I2C values

Initialize Functions
```

```
Function INIT_I2C2
Function INIT_PCA9865
Function STEP1
Function STEP2
Function STEP3
Function STEP4
Function DELAY_LOOP
Function TRANSMIT
Function TURN_OFF
Function INIT
Function WAIT_LOOP
Function INT_DIRECTOR
Function PASS_ON
Function BUTTON_SVC
Function STEP_LOOP
Function INIT_INTERRUPT
Function RESET_INTERRUPT

Designate Stack Arrays for SVC and IRQ

Main:
    Initialize Interrupt
    Initialize I2C2
    Initialize PCA9865
    Go To Wait Loop to wait for interrupt

Step Loop:
    For Loop
        Step1
        Step2
        Step3
        Step4
    Turn off all signals
    Reset interrupt
    Return to wait loop

Reset Interrupt, button_svc, pass_on, int_director from Design Project 1 ECE372 file

Step1:
    Led3
    delay
    led5
    delay

Step2:
    Led4
    Delay
    Led6

Step3:
    Led6
    Delay
    Led3
    Delay

Step4:
    Led5
    Delay
    Led4
    Delay

Turn_off:
    Turn off all LEDs

Init:
    Turn off all LEDs
    Hold PMWB high
    Send x10 to Led2 driver out (pmwA)

Transmit data:
    Function to initialize transfer
    If statement to see if the bus is ready
        Write 0x01 to I2C_CON
        Call Transmit Data
    Else check bus again


Function to transfer data
    If statement to see if the bus is ready
        Load data to I2C_DATA
    Else return

Delay loop
```

Low Level Algorithm Part 2

```
#include stdio.h

define HWREG(x) (*((volatile unsigned int *) (x)))

Base Addr
```

CONTROL_BASE	0x44E10000
CM_PER_BASE	0x44E00000
I2C2_BASE	0x4819C000

Offsets

I2C2_SCL	0x954
I2C2_SDA	0x950
I2C_PSC	0xB0
I2C_SCL_L	0xB4
I2C_SCL_H	0xB8
I2C_CON	0xA4
I2C_IRQSTATUS_RAW	0x24
I2C_OA	0xA8
I2C_CNT	0x98
I2C_DATA	0x9C
I2C_SA	0xAC
I2C_SYSC	0x10
I2C_CLK	0x44

Slave Addresses

LED2_ON_H	0x0F
LED2_OFF_H	0x11
LED3_ON_H	0x13
LED3_OFF_H	0x15
LED4_ON_H	0x17
LED4_OFF_H	0x19
LED5_ON_H	0x1B
LED5_OFF_H	0x1D
LED6_ON_H	0x1F
LED6_OFF_H	0x21
LED7_ON_H	0x23
LED7_OFF_H	0x25
ALL_ON	0xFB
ALL_OFF	0xFD
PRE_SCALE	0xFE
MODE1	0x00
MODE2	0x01

INIT_INTERRUPT() **From Design Project 1 ECE372 using asm("assembly line of code");

```
//Function Initialization
void INIT_I2C2
void INIT_PCA9865
void STEP
void STEP2
void STEP3
void STEP4
void DELAY_LOOP
void TRANSMIT(int address, int data)
void TURN_OFF
void INIT
void WAIT_LOOP
void INT_DIRECTOR
void PASS_ON
void BUTTON_SVC
void STEP_LOOP
void INIT_INTERRUPT
void RESET_INTERRUPT

volatile unsigned int SVC_STACK[1000]
volatile unsigned int IRQ_STACK[1000]

MAIN
   INIT_INTERRUPT
   INIT_I2C2
   INIT_PCA9865
   WAIT_LOOP


STEP_LOOP
         For Loop using I = 26
                    STEP1
                    STEP2
                   STEP3
                   STEP4

         TRANSMIT ALL_OFF 0x10
         RESET_INTERRUPT
         WAIT_LOOP


RESET_INTERRUPT **From Design Project 1 ECE372 using asm(“assembly line of code“);

WAIT_LOOP
   While loop, wait for interrupt

INT_DIRECTOR **From Design Project 1 ECE372 using asm(“assembly line of code“);

PASS_ON **From Design Project 1 ECE372 using asm(“assembly line of code“);

BUTTON_SVC **From Design Project 1 ECE372 using asm(“assembly line of code“);
```

```
   STEP_LOOP

INIT_I2C
  Using HWREG:
        CM_PER_BASE + I2C_CLK)          = 0x2           Turn on I2C2 Clock Module
        CONTROL_BASE + I2C2_SCL)        = 0x32          Configure Pin 21 as I2C2_SCL
        CONTROL_BASE + I2C2_SDA)        = 0x32          Configure Pin 22 as I2C2_SDA
        I2C2_BASE + I2C_SYSC)           = 0x02          Software reset
        I2C2_BASE + I2C_PSC)            = 0x00          Configure Pre-Scale I2C2 register
        I2C2_BASE + I2C_SCL_L)          = 0x08          Configure the I2C low time register
        I2C2_BASE + I2C_SCL_H)          = 0x0A          Configure the I2C high time register
         I2C2_BASE + I2C_OA)            = 0x0           Configure I2C Own Address register
        I2C2_BASE + I2C_SA)             = 0xE0          Set slave address as 0xE0
         I2C2_BASE + I2C_CON)           = 0x8600        Configure I2C_CON Register

INIT_PCA9865
        TRANSMIT(MODE1, 0x11)           Send 0x11 to MODE1 enabling sleep reg to enable
write on PRE_SCALE register
        TRANSMIT(PRE_SCALE, 0x05)       Setting pre-scale for 1kHz
         TRANSMIT(MODE1, 0x01)          Taking MODE1 out of sleep and maintaining
response to all call
         TRANSMIT(MODE2, 0x04)          Set totem pole
        INIT

TRANSMIT(int address, int data)

   (I2C2_BASE + I2C_CNT) = 0x02    configure I2C_CNT register with number of bytes to be
transfered

   If (!(I2C2_BASE + I2C_IRQSTATUS_RAW) & 0x1000))          If the bus busy Bit 12 = 0
   Then (I2C2_BASE + I2C_CON) = 0x8603                      Write 0x01 to CON register
to initiate start transfer condition

      If ((I2C2_BASE + I2C_IRQSTATUS_RAW) & (0x10))         If XRDY bit 4 = 1
      Then (I2C2_BASE + I2C_DATA) = address                 Load I2C Data reg with
address to send out on SDA
        DELAY_LOOP

        (I2C2_BASE + I2C_DATA) = data                       data to output
        DELAY_LOOP
      }

      Else TRANSMIT(address, data)   jump back to top of send_out function and check bits 12
and 4 again)
  Else TRANSMIT(address, data)
```

```
STEP1
   TURN_OFF
   TRANSMIT(LED3_ON_H, 0x10)
   DELAY_LOOP
   TRANSMIT(LED5_ON_H, 0x10)
   DELAY_LOOP

STEP2
   TURN_OFF
   TRANSMIT(LED4_ON_H, 0x10)
   DELAY_LOOP
   TRANSMIT(LED6_ON_H, 0x10)
   DELAY_LOOP

STEP3
   TURN_OFF()
   TRANSMIT(LED6_ON_H, 0x10)
   DELAY_LOOP
   TRANSMIT(LED3_ON_H, 0x10)
   DELAY_LOOP

STEP4
   TURN_OFF
   TRANSMIT(LED5_ON_H, 0x10)
   DELAY_LOOP
   TRANSMIT(LED4_ON_H, 0x10)
   DELAY_LOOP

TURN_OFF
   TRANSMIT(LED6_ON_H, 0x00)
   DELAY_LOOP
   TRANSMIT(LED5_ON_H, 0x00)
   DELAY_LOOP
   TRANSMIT(LED4_ON_H, 0x00)
   DELAY_LOOP
   TRANSMIT(LED3_ON_H, 0x00)
   DELAY_LOOP

INIT
   TRANSMIT(ALL_ON, 0x00)           Turning off off all call
   TRANSMIT(ALL_OFF, 0x00)          turning off on all LED outputs
   DELAY_LOOP
   TRANSMIT(LED7_ON_H, 0x10)        Send 0x10 to LED7 to hold PWMB high
```

```
  DELAY_LOOP
  TRANSMIT(LED2_ON_H, 0x10)          Send 0x10 to LED2 driver out (PWMA)
  DELAY_LOOP
DELAY_LOOP
  For loop: (int n = 0; n < 5000; n++)
    asm("NOP")
```

Code Part 2

```
//Phil Nevins
//ECE 372 DP2 Part 2
//This project will turn a stepper motor 180 degrees using I2C from the B3 board via
PSA9865 LED controller
//It will have a push button that will do this
//We use code from ECE 372 DP1

#include <stdio.h>

//Given Functinon from handout
//C function to write to assembly addr
#define HWREG(x) (*((volatile unsigned int *) (x)))

//Base Addr
#define CONTROL_BASE    0x44E10000
#define CM_PER_BASE     0x44E00000
#define I2C2_BASE       0x4819C000

//Offsets
#define I2C2_SCL          0x954
#define I2C2_SDA          0x950
#define I2C_PSC           0xB0
#define I2C_SCL_L         0xB4
#define I2C_SCL_H         0xB8
#define I2C_CON           0xA4
#define I2C_IRQSTATUS_RAW 0x24
#define I2C_OA            0xA8
#define I2C_CNT           0x98
#define I2C_DATA          0x9C
#define I2C_SA            0xAC
#define I2C_SYSC          0x10
#define I2C_CLK           0x44

//Slave Addresses
#define LED2_ON_H   0x0F
#define LED2_OFF_H  0x11
#define LED3_ON_H   0x13
#define LED3_OFF_H  0x15
#define LED4_ON_H   0x17
#define LED4_OFF_H  0x19
#define LED5_ON_H   0x1B
#define LED5_OFF_H  0x1D
#define LED6_ON_H   0x1F
#define LED6_OFF_H  0x21
#define LED7_ON_H   0x23
#define LED7_OFF_H  0x25
#define ALL_ON      0xFB
#define ALL_OFF     0xFD
#define PRE_SCALE   0xFE
#define MODE1       0x00
#define MODE2       0x01
```

```
void INIT_INTERRUPT()
{
//Button + Debounce Init
asm("LDR R0, =0x4804C000");
asm("LDR R1, =0x44E000AC");         //Address of CM_PER_GPIO1_CLKCTRL
asm("LDR R2, =0x00040002");         //Turn on Aux Funct CLK, bit 18 and CLK
asm("STR R2, [R1]");                //Write value to CMP_PER_GPIO_CLKCTRL
asm("ADD R1, R0, #0x0150");         //Addr of GPIO1_DEBOUNCABLE
asm("MOV R2, #0x00000008");         //Load value of GPIO1 for bit 3
asm("STR R2, [R1]");                //Enable GPIO1_3 debounce
asm("ADD R1, R0, #0x154");          //Addr of GPIO1_DEBOUNCING TIME
asm("MOV R2, #0xA0");               //Value for 31 Micro-Seconds debounce interval
asm("STR R2, [R1]");                //Write to GPIO1_DEBOUNCING TIME)

//Detect Falling Edge on GPIO1_3 and eable to assert POINTRPEND1
asm("ADD R1, R0, #0x14C");      //R1 = ADDR of GPIO1_FALLINGDETECT Register
asm("MOV R2, #0x00000008");     //Load value for Bit 3 (GPIO1_3)
asm("LDR R3, [R1]");            //Read GPIO1_FALLINGDETECT register
asm("ORR R3, R3, R2");          //Modify (set bit 3)
asm("STR R3, [R1]");            //Write back
asm("ADD R1, R0, #0x34");       //Addr of GPIO1_IRQSTATUS_SET_0 Register
asm("STR R2, [R1]");            //Enable GPIO1_3 request on POINTRPEND1

//Initialize INTC
asm("LDR R1, =0x482000E8");     //ADDR of INTC_MIR_CLEAR3 Register
asm("MOV R2, #0x04");           //Value to unmask INTC INT 98, GPIONT1A
asm("STR R2, [R1]");            //Write to INTC_MIR_CLEAR3 Register

//Make sure processor IRQ enabled in CPSR
asm("MRS R3, CPSR");            //Copy CPSR to R3
asm("BIC R3, #0x80");           //Clear bit 7
asm("MSR CPSR_c, R3");          //Write back to CPSR

asm("LDR R13, =SVC_STACK");        //Point to base of STACK1 for SVC mode
asm("ADD R13, R13, #0x1000");   //Point to top of STACK1
asm("CPS #0x12");               //Switch to IRQ mode
asm("LDR R13, =IRQ_STACK");        //Point to IRQ STACK2
asm("ADD R13, R13, #0x1000");   //Point to top of STACK2
asm("CPS #0x13");               //Back to SVC mode
}

//Functions
void INIT_I2C2();
void INIT_PCA9865();
void STEP1();
void STEP2();
void STEP3();
void STEP4();
void DELAY_LOOP();
void TRANSMIT(int address, int data);
void TURN_OFF(void);
void INIT(void);
void WAIT_LOOP();
void INT_DIRECTOR();
void PASS_ON();
void BUTTON_SVC();
void STEP_LOOP();
void INIT_INTERRUPT();
void RESET_INTERRUPT();

volatile unsigned int SVC_STACK[1000];
volatile unsigned int IRQ_STACK[1000];

//Main
int main(void)
{
    INIT_INTERRUPT();
    INIT_I2C2();
    INIT_PCA9865();

    //Wait for interrupt
    WAIT_LOOP();
}

//Step Motor Loop
void STEP_LOOP()
{
    for(int i = 0; i < 26; i++) //Step Loop 13 + Delay 2500 = fast. Only works with
the RESET_INTERRUPT function  || Step loop 26 + Delay 5000 = slow
    {
        STEP1();
        STEP2();
        STEP3();
        STEP4();
    }

TRANSMIT(ALL_OFF, 0x10);
RESET_INTERRUPT();
WAIT_LOOP();
}

void RESET_INTERRUPT()
{
    asm("MRS R3, CPSR");            //Copy CPSR to R3
    asm("BIC R3, #0x80");           //Clear bit 7
    asm("MSR CPSR_c, R3");          //Write back to CPSR

//turn off NEWIRQA bit in INTC_CONTROL, so processor can respond to new IRQ
    asm("LDR R0, =0x48200048");     //ADDR of INTC_CONTROL Register
    asm("MOV R1, #0x01");           //Value to clear bit 0
    asm("STR R1, [R0]");            //Write to INTC_CONTROL Register
}

//Wait for interrupt
void WAIT_LOOP()
{
    while(1);
}
```

```
void INT_DIRECTOR()
{
    asm("LDR R13,=SVC_STACK");  //Push registers on stack
    asm("LDR R0, =0x482000F8");     //ADDR of INTC_PENDING_IRQ3 Register
    asm("LDR R1, [R0]");            //Read INTC_PENDING_IRQ3 Register
    asm("TST R1, #0x00000004");     //Test Bit 2
    asm("BEQ PASS_ON");             //Not from GPIOINT1A, go to wait loop, Else
    asm("LDR R0, =0x4804C02C");     //Load GPIO1_IRQSTATUS_0 Register ADDR
    asm("LDR R1, [R0]");            //Read STATUS Register
    asm("TST R1, #0x00000008");     //Test if bit 3 = 1
    asm("BNE BUTTON_SVC");          //If 1, go to button_svc
    asm("LDR R0, =0x48200048");     //Else, go back. INTC_CONTROL Register
    asm("MOV R1, #0x1");            //Value to clear bit 0
    asm("STR R1, [R0]");            //Write to INTC_CONTROL Register
    asm("LDR R13, =SVC_STACK");  //Restore Registers
    asm("SUBS PC, LR, #4");         //Pass execution to wait loop for now
}

void PASS_ON()
{
    asm("MOV R1, #0x00000008");     //Value to turn off GPIO1_3 & INTC Interrupt
request
    asm("STR R1, [R0]");            //Write to GPIO1_IRQSTATUS_0 Register

//turn off NEWIRQA bit in INTC_CONTROL, so processor can respond to new IRQ
    asm("LDR R0, =0x48200048");     //ADDR of INTC_CONTROL Register
    asm("MOV R1, #0x01");           //Value to clear bit 0
    asm("STR R1, [R0]");            //Write to INTC_CONTROL Register


    asm("LDR R13,=SVC_STACK");  //Restore Registers
    asm("SUBS PC, LR, #4");         //Pass execution onto wait LOOP
}

void BUTTON_SVC()
{
    asm("MOV R1, #0x00000008");     //Value to turn off GPIO1_3 & INTC Interrupt
request
    asm("STR R1, [R0]");            //Write to GPIO1_IRQSTATUS_0 Register

//turn off NEWIRQA bit in INTC_CONTROL, so processor can respond to new IRQ
    asm("LDR R0, =0x48200048");         //ADDR of INTC_CONTROL Register
    asm("MOV R1, #0x01");               //Value to clear bit 0
    asm("STR R1, [R0]");                //Write to INTC_CONTROL Register
    STEP_LOOP();
}


//Initialize I2C2
void INIT_I2C2()
{
    HWREG(CM_PER_BASE + I2C_CLK)    = 0x2;            //Turn on I2C2 Clock Module
    HWREG(CONTROL_BASE + I2C2_SCL)  = 0x32;           //Configure Pin 21 as I2C2_SCL
    HWREG(CONTROL_BASE + I2C2_SDA)  = 0x32;           //Configure Pin 22 as I2C2_SDA
    HWREG(I2C2_BASE + I2C_SYSC)     = 0x02;           //Software reset
    HWREG(I2C2_BASE + I2C_PSC)      = 0x00;           //Configure Pre-Scale I2C2
register
    HWREG(I2C2_BASE + I2C_SCL_L)    = 0x08;           //Configure the I2C low time
register
    HWREG(I2C2_BASE + I2C_SCL_H)    = 0x0A;           //Configure the I2C high time
register
    HWREG(I2C2_BASE + I2C_OA)       = 0x0;            //Configure I2C Own Address
register
    HWREG(I2C2_BASE + I2C_SA)       = 0xE0;           //Set slave address as 0xE0
    HWREG(I2C2_BASE + I2C_CON)      = 0x8600;         //Configure I2C_CON Register
}

//Initialize PCA9865
void INIT_PCA9865(void)
{
    TRANSMIT(MODE1, 0x11);          //Send 0x11 to MODE1 enabling sleep reg to enable
write on PRE_SCALE register
    TRANSMIT(PRE_SCALE, 0x05);      //setting pre-scale for 1kHz
    TRANSMIT(MODE1, 0x01);          //Taking MODE1 out of sleep and maintaining
response to all call
    TRANSMIT(MODE2, 0x04);          //Set totem pole
    INIT();
}

//Transmit Data
void TRANSMIT(int address, int data)
{
    HWREG(I2C2_BASE + I2C_CNT) = 0x02;     //configure I2C_CNT register with number
of bytes to be transfered

    if(!(HWREG(I2C2_BASE + I2C_IRQSTATUS_RAW) & 0x1000))    // If the bus busy Bit 12
= 0
    {
        HWREG(I2C2_BASE + I2C_CON) = 0x8603;        //Write 0x01 to CON register to
initiate start transfer condition

        if((HWREG(I2C2_BASE + I2C_IRQSTATUS_RAW) & (0x10)))         // If XRDY bit 4 =
1
        {
            HWREG(I2C2_BASE + I2C_DATA) = address;            //Load I2C Data reg
with address to send out on SDA
            DELAY_LOOP();

            HWREG(I2C2_BASE + I2C_DATA) = data;            //data to output
            DELAY_LOOP();
        }

        else
        {

            TRANSMIT(address, data);    //jump back to top of send_out function and
check bits 12 and 4 again)
        }
    }
    else
    {
        TRANSMIT(address, data);
    }
}

//Step 1
void STEP1(void)
{
    TURN_OFF();
    TRANSMIT(LED3_ON_H, 0x10);
    DELAY_LOOP();
    TRANSMIT(LED5_ON_H, 0x10);
    DELAY_LOOP();
}

//Step 2
void STEP2(void)
{
    TURN_OFF();
    TRANSMIT(LED4_ON_H, 0x10);
    DELAY_LOOP();
    TRANSMIT(LED6_ON_H, 0x10);
    DELAY_LOOP();
}

//Step 3
void STEP3(void)
{
    TURN_OFF();
    TRANSMIT(LED6_ON_H, 0x10);
    DELAY_LOOP();
    TRANSMIT(LED3_ON_H, 0x10);
    DELAY_LOOP();
}

//Step 4
void STEP4(void)
{
    TURN_OFF();
    TRANSMIT(LED5_ON_H, 0x10);
    DELAY_LOOP();
    TRANSMIT(LED4_ON_H, 0x10);
    DELAY_LOOP();
}

//Turn All Off
void TURN_OFF(void)
{
    TRANSMIT(LED6_ON_H, 0x00);
    DELAY_LOOP();
    TRANSMIT(LED5_ON_H, 0x00);
    DELAY_LOOP();
    TRANSMIT(LED4_ON_H, 0x00);
    DELAY_LOOP();
    TRANSMIT(LED3_ON_H, 0x00);
    DELAY_LOOP();
}

//
void INIT(void)
{
    TRANSMIT(ALL_ON, 0x00);       //Turning off off all call
    TRANSMIT(ALL_OFF, 0x00);      //turning off on all LED outputs
    DELAY_LOOP();
    TRANSMIT(LED7_ON_H, 0x10);    // Send 0x10 to LED7 to hold PWMB high
    DELAY_LOOP();
    TRANSMIT(LED2_ON_H, 0x10);    // Send 0x10 to LED2 driver out (PWMA)
    DELAY_LOOP();
}

void DELAY_LOOP()
{
    for(int n = 0; n < 5000; n++) //Change 5000 to change delay loop timing
    {
        asm("NOP");
    }
}
```

Signed Statement

By signing this statement, I affirm that I did not give any help to any other person, did not receive any help from any other person, except TA and Instructor and did not obtain any information from the Internet or other sources.

Signature:______Philip A Nevins________________________

Date:______3/23/2023_______