I^2C NEMA Motor Controller Driver

Design Project #2

Mx. MMMMMMM KKKKKKKKK

PSU ID 666666666

Portland State University

Maseeh College of Engineering and Computer Science

ECE 372: Microprocessor Interfacing & Embedded Systems

Instructor: Dr. Douglas Hall

TA: Mr. Tyler Hull

Contents

0.1 Introduction

This project utilizes the Beaglebone Black platform featuring the ARM Cortex-A8 AM335x 1GHZ Sitara Processor. We will be utilizing the I2C2, GPIO, Clock, Timer5, Interrupt Controller peripherals (among others) embedded on the device.

BBB GPIO Pin 19 and 20 will be used to interface the I2C2 peripheral output with HW-052 (PCA9685). The HW-052 is a I2C addressable, 16-channel, PWM generating breakout for the PCA9685 IC. It features external driver power delivery and isolated control voltage. The slave address is adjustable via jumpers.

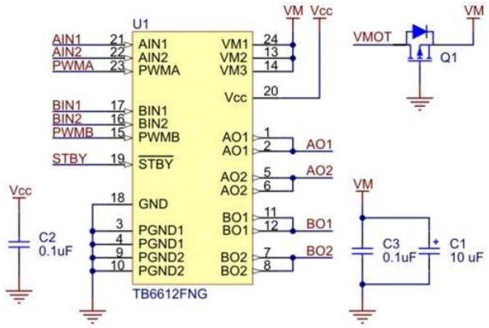

Figure 1: HW-048 Schematic

The PWM outputs from the HW-052 will be used as step functions to drive the AIN2, AIN1, BIN1, and BIN2 pins of a HW-048 (TB6612FNG). The HW-048 is a Polulu-clone "stepper motor driver" H-Bridge breakout for the TB6612FNG IC. It features EMF protection, voltage smoothing, and dual PWM-driven step function output.

Using both sets of the HW-048 output it is possible to have highly precise operation of a bipolar stepper motor. We will be using a Sainsmart 17-HSR-1006-25 BET nema motor (2014-05-14 manufacturing date).

H-SW Control Function

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

Figure 2: Truth Table for TB6612 IC

Part 1 is simply finding success with sending any data on the I2C output.

Part 2 will be utilizing the I2C protocol and peripherals to demonstrate controlled rotational movement of the NEMA motor shaft both in clockwise and counterclockwise directions.

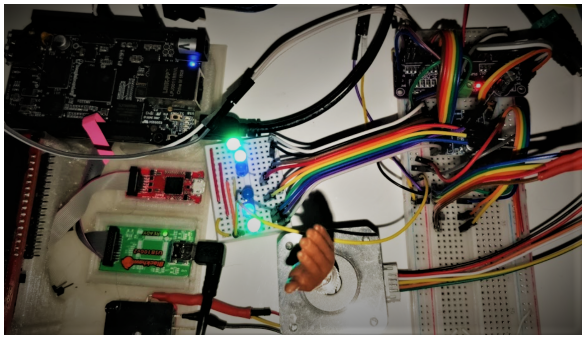

Figure 3: Development station showing Beaglebone Black, Logic Analyzer, Emulator/Debugger, PWM LED outputs, NEMA 17 motor, HW-052, HW-048 and +5 12v Barrel Jack VMot+ supply.

Some information did not make it into the direct design log, but migrated into supplementary logs. Full project files and text logs can be found here:

https://github.com/31415pi/ECE372_I2C2_BBB_NEMA_DRIVER

0.2 Part 1: I2C Initialization and Test

0.2.1 Introduction

Part 1 consists of initializing all the necessary peripheral modules and generating output on GPIO pin 19 and 20 of the BBB. We are using the I2C2 module.

0.2.2 High-Level Algorithm

1. Modify Startup/Bootloader
2. Initialize Stack
3. Initialize I2C2
4. Arm I2C
5. Delay
6. I2C Setup
7. Send Test Data

0.2.3 Medium-Level Algorithm

1. Modify Startup/Bootloader
    - Adjust startup file for _start to point to main in bootloader
    - Adjust startup file for int_handler takeover for interrupt
2. Initialize Stack
    - Load stack
    - Change processor state
    - Load stack
    - Change processor state
3. Initialize I2C2
    - Change Pinmux on Pin 19 to Mode 3, slow slew rate
    - Change Pinmux on Pin 20 to Mode 3, slow slew rate
    - Start I2C2 clock (CM_PER)
    - Reset I2C2 disable auto-idle
    - Set Prescalar Value
    - Set Scalar Low register
    - Set Scalar High register
    - Set Own I2C Address
    - Set Slave I2C Address
    - Set Dcount (data count)
4. Arm I2C
    - Disable IRQ
    - Enable bits for I2C protocol formatting
5. Delay
    - NOP for loop to burn time
6. I2C Setup
    - Check for BB clear
    - Set Dcount to 2 bytes
    - Set start and stop formatting
    - Wait for XRDY
7. Send Test Data
    - Send a Byte
    - Reset XRDY
    - Send another Byte
    - Reset XRDY

0.2.4 Low-Level Algorithm

1. Modify Startup/Bootloader
    - Adjust startup file for _start to point to main in bootloader
        - remove:
        - DP2_PT1:startup_ARMCA8.s:120 LDR r10, = _start
        - add:
        - DP2_PT1:startup_ARMCA8.s:120 LDR r10, = main
    - Adjust startup file for int_handler takeover for interrupt
        - add:
        - DP2_PT1:startup_ARMCA8.s:154 .extern int_handler
        - remove:
        - DP2_PT1:startup_ARMCA8.s:165 LDR pc, [pc,-8] @ 0x18 IRQ interrupt
        - add:
        - DP2_PT1:startup_ARMCA8.s:165 b int_handler
2. Initialize Stack
    - Load stack
    - Change processor state
    - Load stack
    - Change processor state
3. Initialize I2C2
    - Change Pinmux on Pin 19 to Mode 3, slow slew rate [0x33]
    - Change Pinmux on Pin 20 to Mode 3, slow slew rate [0x33]
    - Start I2C2 clock (CM_PER) + 0x44 [0x02]
    - Reset I2C2 disable auto-idle [0x02]
    - Set Prescalar Value [0x03]
    - Set Scalar Low register [0xB4]
    - Set Scalar High register [0xB8]
    - Set Own I2C Address [0x00]
    - Set Slave I2C Address [0x40]
    - Set Dcount (data count) [0x04]
4. Arm I2C
    - Disable IRQ [Set IRQSTATUS_RAW to 0x7FFF]

- Enable bits for I2C protocol formatting [Send 0x8600 to CON]

5. Delay
    - NOP for loop to burn time
6. I2C Setup
    - Check for BB clear
    - Set Dcount to 2 bytes
    - Set start and stop formatting [Send 0x8603 to CON]
    - Wait for XRDY [Check IRQSTATUS_RAW]
7. Send Test Data
    - Send a Byte [0x00 to DATA]
    - Reset XRDY [0x10 to IRQSTATUS]
    - Send another Byte [0xFF to DATA]
    - Reset XRDY [0x10 to IRQSTATUS]

Relevant Declarations:

	Label	Value
#define	CM_PER	0x44E00000
#define	CM_WKUP	0x44E04000
#define	CM_BASE	0x44E10000
#define	I2C2_BASE	0x4819C000
#define	SCL_MUX	0x97C
#define	SDA_MUX	0x978
#define	BB_SET	0x1000
#define	SYSC	0x10
#define	IRQSTATUS_RAW	0x24
#define	IRQSTATUS	0x28
#define	SYSS	0x90
#define	BUF	0x94
#define	CNT	0x98
#define	DATA	0x9c
#define	CON	0xa4
#define	OA	0xa8
#define	SA	0xac
#define	PSC	0xb0
#define	SCLL	0xb4
#define	SCLH	0xb8
#define	SYSTEST	0xbc

0.2.5 Part 1 Full Code

```
//  ***** PLACEHOLDER FOR INT_HANDLER *****
void int_handler(void){}

//  ***** BEGIN GLOBAL DEFINES  *****
#define HWREG(x) (*((volatile unsigned int *)(x)))
volatile unsigned int USR_STACK[100];
volatile unsigned int INT_STACK[100];

//  ***** DEFINE NECESSARY REGISTERS *****
#define CM_PER    0x44E00000
#define CM_WKUP   0x44E04000
#define CM_BASE   0x44E10000
#define I2C2_BASE 0x4819C000

//  ***** DEFINE CERTAIN VALUES *****
#define SCL_MUX        0x97C
#define SDA_MUX        0x978
#define BB_SET        0x1000

//  ***** DEFINE REGISTER OFFSETS *****
#define SYSC            0x10
#define IRQSTATUS_RAW   0x24
#define IRQSTATUS       0x28
#define SYSS            0x90
#define BUF             0x94
#define CNT             0x98
#define DATA            0x9c
#define CON             0xa4
#define OA              0xa8
#define SA              0xac
#define PSC             0xb0
#define SCLL            0xb4
#define SCLH            0xb8
#define SYSTEST         0xbc
```

```
//  ***** FUNCTION DEFINITIONS *****
void write_i2c(unsigned int FIRST);
void I2C_setup(void);
void I2C_init(void);
void I2C_arm(void);
void delay(int time);
void STACK_SETUP(void);

//  ***** BEGIN FUNCTIONS *****
void I2C_init()
{
    HWREG(CM_WKUP + 0xEC) = 0x2;        //wakeup timer 5
    HWREG(CM_WKUP + 0x518) = 0x2;       //set clock speed
    HWREG(CM_BASE + SCL_MUX)    = 0x33; //PIN MODE 3 - PULLUP SLOW SLEW SCL pin 19
    HWREG(CM_BASE + SDA_MUX)    = 0x33; //PIN MODE 3 - PULLUP SLOW SLEW INPUT SDA pin 20
    HWREG(CM_PER + 0x44)        = 0x02; //I2C2 CLK
    HWREG(I2C2_BASE + SYSC)     = 0x02; //RESET I2C2 disable autoidle
    HWREG(I2C2_BASE + PSC)      = 0x03; //SCALE BY 4 TO GET 400kHz
    HWREG(I2C2_BASE + SCLL)     = 0xB4; //SCALAR LOW REG
    HWREG(I2C2_BASE + SCLH)     = 0xB8; //SCALAR HIGH REG
    HWREG(I2C2_BASE + OA)       = 0x00; //-OWN ADDRESS-
    HWREG(I2C2_BASE + SA)       = 0x40; //-SLAVE ADDRESS-
    HWREG(I2C2_BASE + CNT)      = 0x04; // SET COUNT TO FOUR
}

void I2C_arm(void)
{
    HWREG(I2C2_BASE + IRQSTATUS_RAW)= 0x7FFF; // DIABLE IRQ
    HWREG(I2C2_BASE + CON)      = 0x8600;   //ENABLE BIT 9 10 15
}

void delay(int time)
{
    for(int delayval = time; delayval >= 0 ; delayval--)
    {
        asm("NOP");
    }
}
```

```
//  ***** I2C Setup is run before any reg+byte pair are sent *****
void I2C_setup(void)
{
    delay(500);
    while((HWREG(I2C2_BASE+IRQSTATUS_RAW) & BB_SET) ){;}
    HWREG(I2C2_BASE + CNT) = 0x02; // COUNT SET TO 2
    HWREG(I2C2_BASE + CON) = 0x8603;     // START VAL DELIVERY
    while(!(HWREG(I2C2_BASE + IRQSTATUS_RAW))){;}
}

//  ***** Called to write byte to the I2C Data Bus *****
void write_i2c(unsigned int FIRST)
{
    HWREG(I2C2_BASE + DATA) = FIRST; //send the bit
    delay(500); //wait
    HWREG(I2C2_BASE + IRQSTATUS) = 0x10; // RESET XREADY
    delay(500);
}

//  ***** Function to setup the stack and user space *****
void STACK_SETUP(void)
{
    asm("LDR R13, =USR_STACK");
    asm("ADD R13, R13, #0x100");
    asm("CPS #0x12");
    asm("LDR R13, =INT_STACK");
    asm("ADD R13, R13, #0x100");
    asm("CPS #0x13");
}
```

```
//  ***** Main is where all the magic happens *****
int main(void)
{
    STACK_SETUP();
    I2C_init();
    I2C_arm();
    delay(10);
    I2C_setup();
    write_i2c(0x00);
    write_i2c(0xFF);
    return 0;
}
```

0.3 Part 2: Motor Controller Driver

0.3.1 Introduction

Part 2 is expanding on Part 1 and creating the proper I2C signals on I2C peripheral Module 2 to generate the properly synchronized PWM signals from the HW-052 device in order to trigger half-step NEMA commands from the HW-048 device. The halfstep() function takes in an integer argument called dir. This argument directs the rotational direction of the rotor to produce clockwise or counterclockwise shaft torsion.

0.3.2 High-Level Algorithm

1. Modify Startup/Bootloader
2. Initialize I2C2
3. Initialize Timer5
4. Initialize Interrupt Controller
5. Enable IRQ Handler
6. Start Timer5
7. Arm I2C
8. Hand Off to NOP Loop
9. Interrupt Triggered
10. Setup PCA Device, Start Timer5, Increment State, Return to NOP
11. Interrupt Triggered
12. Call Half-Step Function Clockwise, Increment State, Start Timer5, Return to NOP
13. Interrupt Triggered
14. Call Half-Step Function Counter-Clockwise, Decrement State, Start Timer5, Return to NOP

0.3.3 Medium-Level Algorithm

1. Modify Startup/Bootloader
    - Adjust startup file for _start to point to main in bootloader
    - Adjust startup file for int_handler takeover for interrupt
2. Initialize I2C2
    - Change Pinmux on Pin 19 to Mode 3, slow slew rate
    - Change Pinmux on Pin 20 to Mode 3, slow slew rate
    - Start I2C2 clock (CM_PER)
    - Reset I2C2 & disable auto-idle
    - Set Prescalar Value
    - Set Scalar Low register
    - Set Scalar High register
    - Set Own I2C Address
    - Set Slave I2C Address
    - Set Dcount (data count)
3. Initialize Timer5
    - Wakeup Timer5
    - Set Clock Speed
    - Reset Timer5
    - Clear IRQ
    - Enable Overflow IRQ
4. Initialize Interrupt Controller
    - Reset Interrupt Controller
    - Unmask INTC_TINT5
5. Enable IRQ Handler
    - Grab CPSR
    - Clear 0x80
    - Write CPSR and Return
6. Start Timer5
    - Set Timer5 TCRR Counter Register
    - Start timer5
7. Arm I2C
8. Hand Off to NOP Loop - Call wait_loop()
9. Interrupt Triggered - Perform one of the following:
10. Setup PCA Device, Start Timer5, Increment State, Return to NOP
    - Set sleep and allcall response
    - Set Prescale mode for 1KHz
    - Restart, Allcall, Internal Clock setting, and normal mode
    - Set Totem Pole and non-inverted
    - Zero out PWM outputs
    - Set Timer5 TCRR Counter Register
    - Start Timer5
    - Set Current State Var
11. Call Half-Step Function Clockwise, Increment State, Start Timer5, Return to NOP
    - Call halfstep(1)
    - halfstep() starts standby PWMS.
    - Then increments 4 PWMs through 8 steps
    - Set Timer5 TCRR Counter Register
    - Start Timer5
    - Set Current State Var
12. Call Half-Step Function Counter-Clockwise, Decrement State, Start Timer5, Return to NOP
    - Call halfstep(0)
    - halfstep() starts standby PWMS.
    - Then decrements 4 PWMs through 8 steps
    - Set Timer5 TCRR Counter Register
    - Start Timer5
    - Set Current State Var

0.3.4 Low-Level Algorithm

1. Modify Startup/Bootloader
    - Adjust startup file for _start to point to main in bootloader
        - remove:
        - DP2_PT1:startup_ARMCA8.s:120 LDR r10, = _start
        - add:
        - DP2_PT1:startup_ARMCA8.s:120 LDR r10, = main
    - Adjust startup file for int_handler takeover for interrupt
        - add:
        - DP2_PT1:startup_ARMCA8.s:154 .extern int_handler
        - remove:
        - DP2_PT1:startup_ARMCA8.s:165 LDR pc, [pc,#-8] @ 0x18 IRQ interrupt
        - add:
        - DP2_PT1:startup_ARMCA8.s:165 b int_handler
2. Initialize I2C2
    - Change Pinmux on Pin 19 to Mode 3, slow slew rate [0x33]
    - Change Pinmux on Pin 20 to Mode 3, slow slew rate [0x33]
    - Start I2C2 clock (CM_PER) + 0x44 [0x02]
    - Reset I2C2 disable auto-idle [0x02]
    - Set Prescalar Value [0x03]
    - Set Scalar Low register [0xB4]
    - Set Scalar High register [0xB8]
    - Set Own I2C Address [0x00]
    - Set Slave I2C Address [0x40]
    - Set Dcount (data count) [0x04]
3. Initialize Timer5
    - Wakeup Timer5 [send 0x02 to CM_PER + 0xAC]
    - Set Clock Speed [send 0x02 to CM_PER + 0x]
    - Reset Timer5 [send 0x01 to TIMER5 + 0x10]
    - Clear IRQ [send 0x07 to TIMER5 + 0x28]
    - Enable Overflow IRQ [send 0x02 to TIMER5 + 0x2C]
4. Initialize Interrupt Controller

- Reset Interrupt Controller [send 0x02 to INTC + 0x10]
- Unmask INTC_TINT5 [send 0x20000000 to INTC + 0xC8]

5. Enable IRQ Handler
    - Grab CPSR [MSR CPSR to R0]
    - Clear 0x80 [BIC R0 with R0 and #0x80]
    - Write CPSR and Return [MSR CPSR_c with R0]
6. Start Timer5
    - Set Timer5 TCRR Counter Register [Store Count Up From val in TIMER5+ + 0x3C]
    - Start Timer5 [Send 0x01 to TIMER5 + 0x38]
7. Arm I2C
    - Disable IRQ [Set IRQSTATUS_RAW to 0x7FFF]
    - Enable bits for I2C protocol formatting [Send 0x8600 to CON]
8. Hand Off to NOP Loop
    - wait_loop();
9. Interrupt Triggered - Perform one of the following:
10. Setup PCA Device, Start Timer5, Increment State, Return to NOP
    - set sleep and allcall response
        - I2C_setup()
        - write_i2c(MODE1)
        - write_i2c(0x11)
    - Set Prescale mode for 1KHz
        - I2C_setup()
        - write_i2c(PRE_SLR)
        - write_i2c(0x05)
    - Restart, Allcall, Internal Clock setting, and normal mode
        - I2C_setup()
        - write_i2c(MODE1)
        - write_i2c(0x81)
    - Set Totem Pole and non-inverted
        - I2C_setup()
        - write_i2c(MODE2)
        - write_i2c(0x04)

- Zero out PWM outputs
  - I2C_setup()
  - write_i2c(ALL_LED_OFF_H)
  - write_i2c(0x00)
- Start Timer5
  - Set Timer5 TCRR Counter Register [Store Count Up From val in TIMER5+ + 0x3C]
  - Start Timer5 [Send 0x01 to TIMER5 + 0x38]
- current_state = 1

11. Call Half-Step Function Clockwise, Increment STate, STart Timer5, Return to NOP
    - halfstep(1) - Sets both PWMS high
      - For loop cycles through X Groups of Full Turns
      - For loop cycles through 8 steps per Full Turns
      - Each step calls chan_steps(stepno)
      - chan_steps() cycles through step number passed
      - For loop turns PWM9, 10, 11, and 12 on or off depending on step instruction
    - Start Timer5
      - Set Timer5 TCRR Counter Register [Store Count Up From val in TIMER5+ + 0x3C]
      - Start Timer5 [Send 0x01 to TIMER5 + 0x38]
    - current_state = 2
12. Call Half-Step Function Counter-Clockwise, Increment STate, STart Timer5, Return to NOP
    - halfstep(0) - Sets both STDBY PWM's high
      - For loop cycles through X Groups of Full Turns
      - For loop cycles through 8 steps per Full Turns
      - Each step calls chan_steps(stepno) in reverse
      - chan_steps() cycles through step number passed
      - For loop turns PWM9, 10, 11, and 12 on or off depending on step instruction
    - Start Timer5
      - Set Timer5 TCRR Counter Register [Store Count Up From val in TIMER5+ + 0x3C]
      - Start Timer5 [Send 0x01 to TIMER5 + 0x38]
    - current_state = 1

Relevant Declarations:

	Label	Value
#define	CM_PER	0x44E00000
#define	CM_WKUP	0x44E04000
#define	CM_BASE	0x44E10000
#define	I2C2_BASE	0x4819C000
#define	TIMER5	0x48046000
#define	INTC	0x48200000
#define	SCL_MUX	0x97C
#define	SDA_MUX	0x978
#define	BB_SET	0x1000
#define	SYSC	0x10
#define	IRQSTATUS_RAW	0x24
#define	IRQSTATUS	0x28
#define	SYSS	0x90
#define	BUF	0x94
#define	CNT	0x98
#define	DATA	0x9c
#define	CON	0xa4
#define	OA	0xa8
#define	SA	0xac
#define	PSC	0xb0
#define	SCLL	0xb4
#define	SCLH	0xb8
#define	SYSTEST	0xbc
#define	MODE1	0x00
#define	MODE2	0x01
#define	LED8_ON_H	0x27
#define	LED9_ON_H	0x2B
#define	LED10_ON_H	0x2F
#define	LED11_ON_H	0x33
#define	LED12_ON_H	0x37
#define	LED13_ON_H	0x3B
#define	ALL_LED_ON_L	0xFA
#define	ALL_LED_ON_H	0xFB
#define	ALL_LED_OFF_H	0xFD
#define	PRE_SLR	0xFE

0.3.5 Part 2 Full Code

```
//  ***** BEGIN GLOBAL DEFINES  *****
#define HWREG(x) (*((volatile unsigned int *)(x)))
volatile unsigned int USR_STACK[100];
volatile unsigned int INT_STACK[100];
int current_state = 0;

//  ***** DEFINE NECESSARY REGISTERS *****
#define CM_PER    0x44E00000
#define CM_BASE   0x44E10000
#define I2C2_BASE 0x4819C000
#define TIMER5    0x48046000
#define INTC      0x48200000

//  ***** DEFINE CERTAIN VALUES *****
#define SCL_MUX        0x97C
#define SDA_MUX        0x978
#define BB_SET        0x1000

//  ***** DEFINE REGISTER OFFSETS *****
#define SYSC            0x10
#define IRQSTATUS_RAW   0x24
#define IRQSTATUS       0x28
#define SYSS            0x90
#define BUF             0x94
#define CNT             0x98
#define DATA            0x9c
#define CON             0xa4
#define OA              0xa8
#define SA              0xac
#define PSC             0xb0
#define SCLL            0xb4
#define SCLH            0xb8
#define SYSTEST         0xbc
#define MODE1           0x00
#define MODE2           0x01
#define LED8_ON_H       0x27
#define LED9_ON_H       0x2B
```

```
#define LED10_ON_H      0x2F
#define LED11_ON_H      0x33
#define LED12_ON_H      0x37
#define LED13_ON_H      0x3B
#define ALL_LED_ON_L    0xFA
#define ALL_LED_ON_H    0xFB
#define ALL_LED_OFF_H   0xFD
#define PRE_SLR         0xFE

//  ***** FUNCTION DEFINITIONS *****
void write_i2c(unsigned int FIRST);
void I2C_setup(void);
void I2C_init(void);
void I2C_arm(void);
void delay(int time);
void cases(void);
void IntMasterIRQEnable(void);
void wait_loop(void);
void int_handler(void);
void halfstep(int dir);
void chan_step(int stepno);

//  ***** BEGIN FUNCTIONS *****
void I2C_init()
{
    HWREG(CM_PER + 0xEC)       = 0x02; //wakeup timer 5
    HWREG(CM_PER + 0x518)      = 0x02; //set clock speed
    HWREG(CM_BASE + SCL_MUX)    = 0x33; //PIN MODE 3 - PULLUP SLOW SLEW SCL pin 19
    HWREG(CM_BASE + SDA_MUX)    = 0x33; //PIN MODE 3 - PULLUP SLOW SLEW INPUT SDA pin 20
    HWREG(CM_PER + 0x44)        = 0x02; //I2C2 CLK
    HWREG(I2C2_BASE + SYSC)     = 0x02; //RESET I2C2 disable autoidle
    HWREG(I2C2_BASE + PSC)      = 0x03; //SCALE BY 4 TO GET 400kHz
    HWREG(I2C2_BASE + SCLL)     = 0xB4; //SCALAR LOW REG
    HWREG(I2C2_BASE + SCLH)     = 0xB8; //SCALAR HIGH REG
    HWREG(I2C2_BASE + OA)       = 0x00; //-OWN ADDRESS-
    HWREG(I2C2_BASE + SA)       = 0x40; //-SLAVE ADDRESS-
    HWREG(I2C2_BASE + CNT)      = 0x04; // SET COUNT TO FOUR
}
```

```
void I2C_arm(void)
{
    HWREG(I2C2_BASE + IRQSTATUS_RAW)= 0x7FFF; // DIABLE IRQ
    HWREG(I2C2_BASE + CON)      = 0x8600;   //ENABLE BIT 9 10 15
}

void delay(int time)
{
    for(int delayval = time; delayval >= 0 ; delayval--)
    {
        asm("NOP");
    }
}

//  ***** I2C Setup is run before any reg+byte pair are sent *****
void I2C_setup(void)
{
    delay(500);
    while((HWREG(I2C2_BASE+IRQSTATUS_RAW) & BB_SET) ){;}
    HWREG(I2C2_BASE + CNT) = 0x02; // COUNT SET TO 2
    HWREG(I2C2_BASE + CON) = 0x8603;     // START VAL DELIVERY
    while(!(HWREG(I2C2_BASE + IRQSTATUS_RAW))){;}
}

//  ***** Called to write byte to the I2C Data Bus *****
void write_i2c(unsigned int FIRST)
{
    HWREG(I2C2_BASE + DATA) = FIRST;    //send the bit
    delay(500);         //wait
    HWREG(I2C2_BASE + IRQSTATUS) = 0x10; // RESET XREADY
    delay(500);
}

//  ***** Setup PCA Device *****
void PCA_SETUP(void)
{
    I2C_setup();
    write_i2c(MODE1);    //Mode 1 reg
    write_i2c(0x11);    //Sleep, allcall sent
```

```
    I2C_setup();
    write_i2c(MODE1);    //Mode 1 reg
    write_i2c(0x81);    //Restart, allcall, internal clock, normal mode

    I2C_setup();
    write_i2c(PRE_SLR); //Prescalar Reg
    write_i2c(0x05);    //Set PS val

    I2C_setup();
    write_i2c(MODE2);
    write_i2c(0x04);    //Totem pole, non inverted

    I2C_setup();
    write_i2c(ALL_LED_OFF_H);
    write_i2c(0x00);    //Turn em all off
}

//  ***** Main is a workhorse *****
int main(void)
{
    asm("LDR R13, =USR_STACK");
    asm("ADD R13, R13, #0x100");
    asm("CPS #0x12");
    asm("LDR R13, =INT_STACK");
    asm("ADD R13, R13, #0x100");
    asm("CPS #0x13");

    I2C_init();

//  ***** Initialize Timer 5 *****
    HWREG(CM_PER + 0xEC) = 0x2;                 //wakeup timer 5
    HWREG(CM_PER + 0x518) = 0x2;                //set clock speed
    HWREG(TIMER5 + 0x10) = 0x1;                 //software reset
    HWREG(TIMER5 + 0x28) = 0x7;                 //clear irqs
    HWREG(TIMER5 + 0x2C) = 0x2;                 //enable overflow IRQ

//  ***** INTC INIT *****
    HWREG(INTC + 0x10) = 0x2;               //reset INTC
```

```
    HWREG(INTC + 0xC8) = 0x20000000;        //unmask INTC_TINT5

//  ***** ENABLE IRQ *****
    IntMasterIRQEnable();

//  ***** Start Timer 5 *****
    HWREG(TIMER5 + 0x3C) = 0xFFFFF000;       //set timer
    HWREG(TIMER5 + 0x38) = 0x1;              //start timer

    I2C_arm();

    wait_loop();
}

//  ***** Enable IRQ *****
void IntMasterIRQEnable(void)
{
    asm("   mrs     r0, CPSR\n\t"
        "   bic     r0, r0, #0x80\n\t"
        "   msr     CPSR_c, R0");
}

//  ***** The Infinity Loop *****
void wait_loop(void)
{
    while(1)
    {
        //do nothing loop
    }
}

//  ***** Interrupt Routine *****
void int_handler(void)
{
    if(HWREG(0x482000D8) == 0x20000000)
    {
        cases();
    }
    asm("LDMFD SP!, {LR}");
```

```
    asm("LDMFD SP!, {LR}");
    asm("SUBS PC, LR, #0x4");
}

//  ***** Checks the state and handles accordingly *****
void cases(void)
{
    HWREG(TIMER5 + 0x28) = 0x7;                  //clear timer5 interrupts
    HWREG(INTC + 0x48) = 0x1;                    //clear NEWIRQ bit in INTC

    switch(current_state)                        // Depending on state act accordingly
    {
        case (0):   PCA_SETUP();                 // Case clears after first run
                    current_state = 1;
                    break;

        case (1):   halfstep(1);                 // clockwise
                    current_state = 2;           // setup for ccw next
                    break;

        case (2):   halfstep(0);                 // ccw
                    current_state = 1;           // setup for clockwise next
                    break;
    }
    //reset timer
    HWREG(TIMER5 + 0x3C) = 0xFFFFFF00;   //set timer
    HWREG(TIMER5 + 0x38) = 0x1;              //start timer
}

//  ***** Run the halfstep rotation algo *****
void halfstep(int dir)
{
    I2C_setup();
    write_i2c(LED8_ON_H);
    write_i2c(0x01);
    I2C_setup();
    write_i2c(LED13_ON_H);
    write_i2c(0x01);
    delay(500);
```

```
    for(int cycle = 0 ; cycle < 201; cycle ++)
    {
        if(dir ==1)
        {
            for(int s = 0; s < 8; s++)
            {
                chan_step(s);
            }
        } else
        {
            for(int s = 7; s >= 0 ; s--)
            {
                chan_step(s);
            }
        }
    }
}

//  ***** Does the processing of each step instruction into PWM triggerin *****
void chan_step(int stepno)
{
    int regs[] = {LED9_ON_H, LED10_ON_H, LED11_ON_H, LED12_ON_H};
    int steps[8][4]=  {{1,0,1,0},{1,0,0,0},{1,0,0,1},{0,0,0,1},{0,1,0,1},{0,1,0,0},{0,1,1,0},{0,0,1,0}};
    for(int t = 0; t < 4; t++)
    {
         I2C_setup();
         write_i2c( regs[t] );
         write_i2c( steps[stepno][t] == 1? 0x01 : 0x00 );
         delay(50);
    }
    delay(9000);
}
```

0.4 Raw Design Log

Klementyn, Maddie

ECE 372

Design Project #2 - MOTOR CONTROLLER DRIVER

Design Log

January 6th 2021

Looked into various devices to use for the project.

Found 4 potential IC / breakout boards in my stock.

None of them corresponded to the chips used in the board supplied in the kit.

Bypassing the kit to use a pure PCA9685 based breakout + TB6612(FNG) based breakout.

Commencing ordering:

$/UNIT S/H CHIP URL - GLOBALGO STORE - CAINIAO SUPER ECONOMY 30-50DAYS

$1.96 $2.05 PCA9685 https://www.aliexpress.com/item/4001343775124.html

$1.48 $2.14 TB6612FNG https://www.aliexpress.com/item/1005001656690370.html

TOTAL FOR 1PC OF EACH $5.85

TOTAL FOR 5 SETS $24.25

March 1st

I guess the next five entries after Jan 6th were deleted somehow. GREAT.

There was some discussion about the 2 boards, the interfacing bits, and i2c addressing on them.

Processing samples and identifying important portions that I will need to use for my custom board setup.

High Level

Identify PIN MUX's

P9

i2c-2 SCL Pin 19

i2c-2 SDA Pin 20

Control Module

Pad Control Register Field Descriptions -

Pullup

Receiver Enabled

Clock

Clock to i2c2

Scale 48MHz to 12 MHz

400 kbps SCL for F/S mode

Polled Mode Transmit

No DMA yet

No i2c Interrupts yet

Initialize Controller

Start a transmission

Tx next byte when ready

Soldered up a HW-052 board, which houses the PCA9685 chip and breakout pins.

https://learn.adafruit.com/16-channel-pwm-servo-driver?view=all

Adding a Capacitor to the thru-hole capacitor slot

We have a spot on the PCB for soldering in an electrolytic capacitor. Based on your usage, you may or may

Tomorrow:

Identify all registers and read the rest of the code examples.

Solder Cap to board.

Design work rig

11pm.

March 2nd

Delays:

5k - between commands

40k -between steps

HW-048 - found datasheet - nope wrong part.

TB952A1

6612FNG

pins: 15, 16, 17, 19, 21, 22, 23

have internal 200k pull down resistance

24

Motor supply 25v~13.5V

HW-052

PCA9685

Setup for direct servo output - remember to hookup external psu to Toshiba only.

VSS power supply ground!

Pin Mux:

mode 0 Mode 3 PROC Name

P9 Pin 19 UART1_RTSN I2C2_SCL D17 I2C1_SDA

P9 Pin 20 UART1_CTSN I2C2_SDA D18 I2C1_SDA

CONTROL MODULE REGISTER:

978h conf_uart1_ctsn Section 9.3.1.50

97Ch conf_uart1_rtsn Section 9.3.1.50

Table 9-60. conf_<module>_<pin> Register Field Descriptions

PULLup ENABLE

RECEIVER ENABLES

FAST SLEW

SO...

WE NEED BITS 5 4 3 and 03 on 0-2

scl - pullup mode 03 fast slew

00000000000000000000000000011011

0x0000005B

sda - input pullup mode 03 fast slew

00000000000000000000000000111011

0x0000007B

March 4th

page 8 pca -- slave addressing / secondary address

7 bit count not 8 bit

coderstoolbox.net/number

controller handles SA

not in d count

not in array of data

set address / sa/ in setup and thats it

254 pre scale addressing

for each address has 4 registers --

can set timing sequence as ADSR

-->what address // what value // takes these two

poll busy

poll xready

send all data -- 10 or 100 bytes, send it all in loop

when:

d count reaches zero - gen stop n etc

then:

next send---

check bus busy

check xready

dont poll b/w bytes but bw all chunks or larger packets

Start & Stop events:

START: high to LO on SDA, SCL held HI | BB = 1 -- BUSY

STOP: low to HI on SDA, SCL held HI | BB = 0 -- FREE

Data TX:

MSB first.

Bytes on SDA are 8 bits long

DCOUNT is programmed for X number of bytes.

RX Mode sends AWK post byte.

Dataformats:

7b/10b

7b/10b with repeated start condition

First byte after a start condition (S) Always 8 bits.

RX/Awk mode - extra bit dedicated for AWK is inserted after each byte.

7b

7 MSB (tx first) slave address bits, and 1 LSB R/nW bit.

177777771 1 n 1 n 1 1

SSlaveAdRWACKDATAackDATAackP

Repeated start mode : bits

1777777711 repeats

Master Mode TX

SDA output

SCL sync data

SCL halts when XUDF / processor intervention required

Halts post byte of TX

Prescalar:

12-100MHz

24MHz ICLK recommended in case of F/S operation mode e.

4 bit register I2C_PSC

Divides sys clock SCLK to obtain internal required clock for I2C module

21.3.15.1 Module Configuration Before Enabling the Module

1. Program the prescaler to obtain an approximately 12-MHz I2C module clock (I2C_PSC = x; this value is to be calculated and is dependent on the System clock frequency).

2. Program the I2C clock to obtain 100 Kbps or 400 Kbps (SCLL = x and SCLH = x; these values are to be calculated and are dependent on the System clock frequency).

3. Configure its own address (I2C_OA = x) - only in case of I2C operating mode (F/S mode).

4. Take the I2C module out of reset (I2C_CON:I2C_EN = 1).

21.3.15.2 Initialization Procedure

1. Configure the I2C mode register (I2C_CON) bits.

2. Enable interrupt masks (I2C_IRQENABLE_SET), if using interrupt for transmit/receive data.

3. Enable the DMA (I2C_BUF and I2C_DMA/RX/TX/ENABLE_SET) and program the DMA controller) - only in case of I2C operating mode (F/S mode), if using DMA for transmit/receive data.

21.3.15.3 Configure Slave Address and DATA Counter Registers

In master mode, configure the slave address (I2C_SA = x) and the number of byte associated with the transfer (I2C_CNT = x).

21.3.15.4 Initiate a Transfer

Poll the bus busy (BB) bit in the I2C status register (I2C_IRQSTATUS_RAW). If it is cleared to 0 (bus not busy), configure START/STOP (I2C_CON: STT / I2C_CON: STP condition to initiate a transfer) - only in case of I2C operating mode (F/S mode).

offset register notes

10h I2C_SYSC 1100001100

9-8 keep clock active

4-3 no idle mode

2 wakeup enabled

1 softreset

0 auto idle disabled

94h I2C_BUF Buffer Config

98h I2C_CNT Data Counter

9Ch I2C_DATA Data Access

A4h I2C_CON Configuration

A8h I2C_OA Own Address

ACh I2C_SA Slave Address

B0h I2C_PSC Clock Prescaler

B4h I2C_SCLL SCL Low Time

B8h I2C_SCLH SCL High Time

BCh I2C_SYSTEST System Test

C0h I2C_BUFSTAT Buffer Status

C4h I2C_OA1Own Address 1

C8h I2C_OA2Own Address 2

CCh I2C_OA3Own Address 3

D0h I2C_ACTOA Active Own Address

D4h I2C_SBLO CKClock Blocking Enable

```
#define GPIO1BA 0x4804C000
#define GPIO_SET_DATA_OUT 0x194
#define GPIO_CLEAR_DATA_OUT 0x190
```

```
//INTC defines
#define INTCBA 0x48200000
//Timer 5 defines
#define TIMER5_BA 0x48046000
//other defines
#define CLKWKUPS 0x44E00000
#define LIGHT_BITS 0x01E00000
//i2c defines
#define CM_PER    0x44E00000
#define CM_WKUP   0x44E04000
#define I2C2_BASE 0x4819C000
//#define SCL_MUX        0x96C    // uart1_rtsn
//#define SDA_MUX        0x968    // uart1_ctsn
#define SCL_MUX        0x97C // uart1_rtsn incorrect
#define SDA_MUX        0x978 // uart1_ctsn incorrect
#define CM_BASE   0x44E10000


//i2c defines - regex parsed from pdf
#define I2C2_BASE 0x4819c000
#define REVNB_LO        0x00
#define REVNB_HI        0x04
#define SYSC            0x10
#define IRQSTATUS_RAW   0x24
#define IRQSTATUS       0x28
#define IRQENABLE_SET   0x2c
#define IRQENABLE_CLR   0x30
#define WE              0x34
#define DMARXENABLE_SET 0x38
#define DMATXENABLE_SET 0x3c
#define DMARXENABLE_CLR 0x40
#define DMATXENABLE_CLR 0x44
#define DMARXWAKE_EN    0x48
#define DMATXWAKE_EN    0x4c
#define SYSS            0x90
#define BUF             0x94
#define CNT             0x98
#define DATA            0x9c
#define CON             0xa4
```

```
#define OA              0xa8
#define SA              0xac
#define PSC             0xb0
#define SCLL            0xb4
#define SCLH            0xb8
#define SYSTEST         0xbc
#define BUFSTAT         0xc0
#define OA1             0xc4
#define OA2             0xc8
#define OA3             0xcc
#define ACTOA           0xd0
#define SBLOCK          0xd4

#define I2C_EN 15



CM_PER -  0X44e0_0000
cm_wkUP -  0X44E_0400

@enable Clocks
0h CM_PER_L4LS_CLKSTCTRL Section 8.1.12.1.1
@GPIO
@I2C
44H CM_PER_I2C2_CLKCTRL



@GPIO
@output enable on tx line
change to mode 3

Control Module
Pad Control Register Field Descriptions -
Pullup
Receiver Enabled
Clock
Clock to i2c2
Scale 48MHz to 12 MHz
400 kbps SCL for F/S mode
Polled Mode Transmit
```

No DMA yet

No i2c Interrupts yet

Initialize Controller

Start a transmission

Tx next byte when ready

C Code:

```
#define REGISTER 0x00000000

int main(void){
__asm
{
"LDR R1, =XYZ" @ FOLLOW THIS FORMAT TO IMPORT ASM
}
}
```

lookup extern & global.

poll two registers

when free

send val, dcount & [ ]

-

MARCH 14 & 15

sda 978h conf_uart1_ctsn Section 9.3.1.50

scl 97Ch conf_uart1_rtsn Section 9.3.1.50

WE NEED BITS 5 4 3 and 03 on 0-2

scl - pullup mode 03 slo slew

00000000000000000000000000011011

0x0000005B

sda - input pullup mode 03 slo slew

00000000000000000000000000111011

0x0000007B

254 pre scale addressing

for each address has 4 registers --

can set timing sequence as ADSR

‘‘‘

#define I2C2_BASE 0x4819c000

```
LDR R0, =0x44E00044  @ Load address for CM_PER+CM_PER_I2C2_CLKCTRL
MOV R2, #0x02 @ Enable
STRB R2, [R0] @ Write value to register
@ Change pin mode
LDR R0, =0x44E1097C @ Load address for conf_uart1_rtsn
MOV R2, #0x5b @ Value for  pullup mode 03 slo slew
STRB R2, [R0] @ Write value to register
LDR R0, =0x44E10958 @ Load address for conf_uart1_ctsn
MOV R2, #0x7b @ Value for input pullup mode 03 slo slew
STRB R2, [R0] @ Write value to register
@ reset
LDR R0, =0x4819C010 @ Load address for I2C_SYSC
MOV R2, #0x02 @ Value to set bit-2/SRST
STRB R2, [R0] @ Write value to register
@ Scale I2C clock to 12MHz
LDR R0, =0x4819C0B0  @ Load address for I2C_PSC
MOV R2, #0x03 @ Value to scale by 4
STRB R2, [R0] @ Write value to register
@ 100Kbps, fiddy pcent
LDR R0, =0x4819C0B4 @ Load address of I2C_SCLL
MOV R2, #0x35 @ Value for SCLL
STRB R2, [R0] @ Write value to register
LDR R0, =0x4819C0B8 @ Load address of I2C_SCLH
MOV R2, #0x37 @ Value for SCLH
```

```
STRB R2, [R0] @ Write value to register
@ Set I2C_OA
LDR R0, =0x4819C0A8  @ Load address for I2C_OA
MOV R2, #0x00 @ Value for own address
STRB R2, [R0] @ Write value to register
@Debug Config IN I2C_SYSTEST
LDR  R0, =0X4819C0BC @ Address of I2C_SYSTEST
LDR R2, [R0] @ LOAD value
AND  R2,R2, #0xFFFFBFFF @ Mask it for b14-Free Debug
STR R2, [R0] @ Write value to registert

@ Take I2C as master transmitter 7-bit data
LDR R0, =0x4819C0A4  @ Load address for I2C_CON
MOV R2, #0x8600 @ Value for bits 9, 10, 15
STR R2, [R0] @ Write value to register
@ Set bytes to transfer
LDR R0, =0x4819C098   @ Load address for I2C_CNT
MOV R2, #0x0A @ Value for bits 0-15/DCOUNT:10_D characters
STRB R2, [R0] @ Write value to register
@ Set slave address xxxxxxx
LDR R0, =0x4802A0AC  @ Load address for I2C_SA
MOV R2, #0xff @ Value for bits 0-7
STR R2, [R0] @ Write value to register
/* @ Clear interrupts
LDR R0, =0x4819C
.028 @ Load address for I2C_IRQSTATUS
LDR R2, =0x7FFF @ Load address for value to clear interrupts
STR R2, [R0] @ Write value to register  */



March 19th picking up with getting some bits to send repeatedly just to test i2c


setup my c incorrectly - was using ASM with C defines
reset it to just use c
noticed timer was going unrestrained.
it was actully my cm wkup was missing a zero.
cm wakeup might be incorrect addy.


March 20th
```

Fixed up my PT1 to use C vs ASM, and poking at getting polling done in C vs ASM before moving onto part 2

Information on HWREG and associated macros.

http://software-dl.ti.com/simplelink/esd/simplelink_msp432e4_sdk/2.30.00.14/docs/driverlib/msp432e4/html/

March 21st

PCA9685:

Slave address:

1 A5 A4 A3 A2 A1 A0 R/W

1 - fixed

A5-A0 - Hardware Selectable

R/W - write low.read high.

Registers/etc:

Copied from data sheet

regex find replace:

F: (^[0-9]*)(\s)(..)([0-9,\s]*)(\S*)(\s)(.*)

R: #define \5 \3 /\/\ \7

concatenate relevant lines

F: (^#)(.*)($)(\R)(^.*)

R: \1\2 \3 \5

Remove irrelevant lines:

F: (^#)(.*)($)|(.*)

R: \1\2 \3

Output copied to PCA9685.h

?

a. Set sleep and respond to All Call

--in mode1 this is bit 4 andbit 0

--hex value is 0x11 to 0x00

b. Set Prescale mode for 1 KHz [confused about this part]

confusing bc :val = round((osc clock) / (4096*update)) -1The example gives output freq of 200hz and cloc

were looking for an output frequency of ?? and clock of 25 mhz. Since we only need step functions without

c. Restart, AllCall, Internal Clock, normal mode

bit 7 is restartbit 6 low for int clockbit 4 low for normal modeso thats 0x81 to 0x00

d. Set for Totem Pole structure, non-inverted

This is governed by Mode Two register-bit 2 is totem, and bit 4 is inverted. so this is 0x04 to register

e. Zero ALL_LED_OFF_H , so just write to LEDn_ON registers.

This is 0x00 to the ALL_LED_OFF_H register at 0xFD.

1. 0x11 to mode1
2. 0x81 to mode1
3. 0x05 to 0xFE
4. 0x04 to mode 2
5. 0x00 to FD

Logic levels:

TB6612FNG 3.3/5 output - assuming input is the same. If I burn it, I have more.

PCA9685 1.6/2.3/4.0v output

5.5vmax 0.7+ is +, -0.5-0.3v is -

Slave address is 0x40.

Now I am receiving ack's and getting to sending commands.

No PWM output still.

Got some blips but might have fried my LA

Had the orientation incorrect with the I2C setup and the all on values

```
void Adafruit_PWMServoDriver::wakeup() {
  uint8_t sleep = read8(PCA9685_MODE1);
  uint8_t wakeup = sleep & ~MODE1_SLEEP; // set sleep bit low
  write8(PCA9685_MODE1, wakeup);
}
```

March 22nd

refactored some .h files

March 23rd

Got PWM out confirmed ...

```
void CW(void)
{
```

```
    I2C_setup();
write_i2c(ALL_LED_OFF_H);  //All off HIGH
write_i2c(0x00);     //Set

    I2C_setup();
write_i2c(ALL_LED_OFF_L);  //All off LOW
write_i2c(0x00);     //Set

I2C_setup();
write_i2c(LED0_ON_H);  //PWM 0 HIGH
write_i2c(0x01);     //Set ON

I2C_setup();
write_i2c(LED1_OFF_H);  //PWM 1 HIGH
write_i2c(0x00);     //Set

I2C_setup();
write_i2c(LED2_ON_H);  //PWM 2 HIGH
write_i2c(0x01);     //Set ON

I2C_setup();
write_i2c(LED3_ON_H);  //PWM 3 HIGH
write_i2c(0x01);     //Set ON
}
```

```
void CW(void)
{   //HLHH --> CLOCKWISE MOVEMENT

    I2C_setup();
    write_i2c(ALL_LED_OFF_H);   //All off HIGH
    write_i2c(0x00);            //Set
```

```
    I2C_setup();
    write_i2c(ALL_LED_OFF_L);   //All off LOW
    write_i2c(0x00);            //Set

    I2C_setup();
    write_i2c(LED0_ON_H);       //PWM 0 HIGH
    write_i2c(0x00);            //Set ON

    I2C_setup();
    write_i2c(LED1_OFF_H);      //PWM 1 HIGH
    write_i2c(0x01);            //Set

    I2C_setup();
    write_i2c(LED2_ON_H);       //PWM 2 HIGH
    write_i2c(0x00);            //Set ON

    I2C_setup();
    write_i2c(LED3_ON_H);       //PWM 3 HIGH
    write_i2c(0x00);            //Set ON
}

void CCW(void)
{   //LHHH --> Counter CLOCKWISE MOVEMENT

    I2C_setup();
    write_i2c(ALL_LED_OFF_H);   //All off HIGH
    write_i2c(0x00);            //Set

    I2C_setup();
    write_i2c(ALL_LED_OFF_L);   //All off LOW
    write_i2c(0x00);            //Set

    I2C_setup();
    write_i2c(LED0_OFF_H);      //PWM 0 HIGH
    write_i2c(0x01);            //Set OFF

    I2C_setup();
    write_i2c(LED1_ON_H);       //PWM 1 HIGH
```

```
    write_i2c(0x00);            //Set ON

    I2C_setup();
    write_i2c(LED2_ON_H);       //PWM 2 HIGH
    write_i2c(0x00);            //Set ON

    I2C_setup();
    write_i2c(LED3_ON_H);       //PWM 3 HIGH
    write_i2c(0x00);            //Set ON
}

void ALL_LED_OFF(void)
{
    I2C_setup();
    write_i2c(ALL_LED_OFF_H);   //All off HIGH
    write_i2c(0x01);            //Set
    I2C_setup();
    write_i2c(ALL_LED_OFF_L);   //All off LOW
    write_i2c(0x00);            //Set
}

void ALL_LED_ON(void)
{
    I2C_setup();
    write_i2c(ALL_LED_ON_H);   //All on HIGH
    write_i2c(0x01);            //Set
    I2C_setup();
    write_i2c(ALL_LED_ON_L);   //All on LOW
    write_i2c(0x01);            //Set
}



 * Step C0 C1 C2 C3
 *    1  1  0  1  0
 *    2  0  1  1  0
 *    3  0  1  0  1
 *    4  1  0  0  1

 DX    PWM
```

```
 0 10
 1  9
 2   11
 3 12

 8/13 PWM == SPEED.
 STBY -> HIGH

   STEP->1 2 3 4
P10 H L L H
P09 L H H L
P11 H H L L
P12 L L H H

P08 H H H H
P13 H H H H

STEP1[] = {1,0,1,0};
STEP2[] = {0,1,1,0};
STEP3[] = {0,1,0,1};
STEP4[] = {1,0,0,1};

CW[]  = {STEP1, STEP2, STEP3, STEP4};
CCW[] = {STEP4, STEP3, STEP2, STEP1};

int STEP_OFF[] = {LED10_OFF_H, LED9_OFF_H, LED11_OFF_H, LED12_OFF_H};
int STEP_ON[] = {LED10_ON_H, LED9_ON_H, LED11_ON_H, LED12_ON_H};

//DIRECTION=
// CW -> STEP1 -> PWM1 PWM2 PWM3 PMW4
//   STEP2
//   STEP3
//   STEP4

void CYCLE(int dir)
{
if(dir == 1){
DIRECTION = CW;
}
```

```
else
{
DIRECTION = CCW;
}

for (int)



for(int step = 0; step < 5; step ++)
{
for int(click = 0; click < 5 ; click++)
{
if(DIRECTION[step[click]] == 0)
{
write_i2c(STEP_OFF[DIRECTION[step[click]]]);
} else
{
write_i2c(STEP_ON[DIRECTION[step[click]]]);
}
}
}
}



///////

I2C_setup(); // setup send
if(STEP_VALS[step] == 0)
{
write_i2c(STEP_OFF[step]); // pick reg off if 0
} else
{
write_i2c(STEP_ON[step]); // pick reg on if 1
}
write_i2c(STEP_VALS[step]); // write enable val
}
}
///////
```

```
void CYCLE(int dir)
{
    int *DIRECTION;
    int *STEP_VAL;
    int STEP1[] = {1,0,1,0};
    int STEP2[] = {0,1,1,0};
    int STEP3[] = {0,1,0,1};
    int STEP4[] = {1,0,0,1};

    int *CW[]  = {STEP1, STEP2, STEP3, STEP4};
    int *CCW[] = {STEP4, STEP3, STEP2, STEP1};

    int STEP_OFF[] = {LED10_OFF_H, LED9_OFF_H, LED11_OFF_H, LED12_OFF_H};
    int STEP_ON[] = {LED10_ON_H, LED9_ON_H, LED11_ON_H, LED12_ON_H};

    //DIRECTION=
    //  CW -> STEP1 -> PWM1 PWM2 PWM3 PMW4
    //        STEP2
    //        STEP3
    //        STEP4

    if(dir == 1){
        DIRECTION = *CW;
    }
    else
    {
        DIRECTION = *CCW;
    }

    for(int step = 0; step < 5; step ++)
    {
        STEP_VAL = DIRECTION[step];

        for (int click = 0; click < 5 ; click++)
        {
            I2C_setup();

            if(STEP_VAL[click] == 0)
```

```
            {
                write_i2c(STEP_OFF[click]);
            } else
            {
                write_i2c(STEP_ON[click]);
            }
            write_i2c(STEP_VAL[click]);
        }
    }
}



void CYCLE(int dir)
{
    int STEP_VALS;
    int STEP1[] = {1,0,1,0};
    int STEP2[] = {0,1,1,0};
    int STEP3[] = {0,1,0,1};
    int STEP4[] = {1,0,0,1};

    int CW[][]  = {*STEP1, *STEP2, *STEP3, *STEP4};
    int CCW[][] = {*STEP4, *STEP3, *STEP2, *STEP1};

    int STEP_OFF[] = {LED10_OFF_H, LED9_OFF_H, LED11_OFF_H, LED12_OFF_H};
    int STEP_ON[] = {LED10_ON_H, LED9_ON_H, LED11_ON_H, LED12_ON_H};



    if(dir == 1)
    {
        for(int supernatural = 0 ; supernatural <5; supernatural ++)
        {
            int STEP_VALS[] = *CW[supernatural];
            for(int step = 0; step < 5 ; step ++)
            {

            }
        }

    }
```

}

Truth Table for all ins and outs.

intputs:

pwm: 10 9 11 12 10 9 12 11

AI1	AI2	PWMA	STBYA	BI1	BI2	PWMB	STBYB	AO1	AO2	BO2	BO1	STEP1	STEP2	STEP3	STEP4	S
H	L	H	H	H	L	H	H	H	L	H	L	HLHL				1
H	L	H	H	L	H	H	H	H	L	L	H	HLLH				2
L	H	H	H	L	H	H	H	L	H	L	H	LHLH				3
L	H	H	H	H	L	H	H	L	H	H	L	LHHL				4

//B A D C TOP TO BOTTOM.

MARCH 24

IT WORKS. but it dies after x mount of steps. adding a soft reset

Resets were uneccesary.

March 25th

Final demo signed off

Began project report in LaTeX.

March 26th @ 12:10 AM - FInished Report

Submitting via email to Mr. Tyler Hull.

Mawlee signing off.