A11 - Pulse Width Modulation and Servos

Students: Zach Bunce & Garrett Maxon
Class: CPE 329-03
Professor: Gerfen, Jeffrey

Youtube Demo: https://youtu.be/Dy0PBAWmUvA

C Code Used

```
/*
 * main.c
 * Runs the keypad to servo PWM interface
 *
 * Date: June 6, 2018
 * Author: Zach Bunce, Garrett Maxon
 */

#include "msp.h"
#include "keypad.h"
#include "delays.h"
#include "set_DCO.h"

void setDC();
void TIMER_INIT();

int highCnt = 4500;
int lowCnt = 55500;
char vals[2] = {0,9};
char valBuf[2] = {0,9};
int valCnt = 0;

void main()
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     //Stop watchdog timer

    int CLK = 30;
    char keyVal = K_NP;

    set_DCO(CLK);   //Sets clocks to 3MHz
    TIMER_INIT();   //Initializes timer used for PWM
    KEYPAD_INIT();  //Initializes keypad

    __enable_irq(); //Enables global interrupt

    NVIC->ISER[0] = 1 << ((TA0_0_IRQn) & 31);

    P6 -> DIR |= BIT1;
    P6 -> OUT &= ~BIT1;

    while(1)
    {
        keyVal = K_NP;

        //Waits for first key press
        while(keyVal == K_NP) {
```

```
            keyVal = get_Key();
        }
        //Waits for 1st key press
        while(valCnt == 0 && !(keyVal == K_1   || keyVal == K_0 ||
                               keyVal == K_Ast || keyVal == K_Pnd)) {
            keyVal = get_Key();
        }
        //Waits for 2nd key press
        while(valCnt == 1 && vals[0] == K_1 && keyVal == K_9) {
            keyVal = get_Key();
        }
        if(keyVal == K_Ast) { //Rotate 10 Degrees CW
            if(valBuf[1] == K_0 && valBuf[0] == K_1) {
                valBuf[0] = K_0;
                valBuf[1] = K_9;
            }
            else if(valBuf[1] != K_0) {
                valBuf[1] -= 1;
            }
            setDC();
        }
        else if(keyVal == K_Pnd) { //Rotate 10 Degrees CCW
            if(valBuf[1] == K_9) {
                valBuf[0] = K_1;
                valBuf[1] = K_0;
            }
            else if(!(valBuf[0] == K_1 && valBuf[1] == K_8)) {
                valBuf[1] += 1;
            }
            setDC();
        }
        else {
            vals[valCnt] = keyVal;
            valCnt++;
        }

        if(valCnt >= 2) {
            valBuf[0] = vals[0];
            valBuf[1] = vals[1];
            setDC();
            valCnt = 0;
        }
        delay_ms(700, CLK);
    }
}

void TIMER_INIT()
{
    TIMER_A0 -> CCTL[0] = TIMER_A_CCTLN_CCIE;     //TACCR0 interrupt enabled
    TIMER_A0 -> CCR[0]  = 60000;
```

```
    TIMER_A0 -> CTL     = TIMER_A_CTL_SSEL__SMCLK |   //Runs the timer on SMCLK
                          TIMER_A_CTL_MC__CONTINUOUS; //Uses continuous mode
}

//Sets the duty cycle high count and low count based on the key pressed
void setDC()
{
    switch(valBuf[1]) {
    case(K_0):
        if(valBuf[0] == K_0)
            highCnt = 3000;
        else
            highCnt = 6000;
        break;
    case(K_1):
        if(valBuf[0] == K_0)
            highCnt = 3300;
        else
            highCnt = 6300;
        break;
    case(K_2):
        if(valBuf[0] == K_0)
            highCnt = 3600;
        else
            highCnt = 6600;
        break;
    case(K_3):
        if(valBuf[0] == K_0)
            highCnt = 3900;
        else
            highCnt = 6900;
        break;
    case(K_4):
        if(valBuf[0] == K_0)
            highCnt = 4200;
        else
            highCnt = 7200;
        break;
    case(K_5):
        if(valBuf[0] == K_0)
            highCnt = 4500;
        else
            highCnt = 7500;
        break;
    case(K_6):
        if(valBuf[0] == K_0)
            highCnt = 4800;
        else
            highCnt = 7800;
        break;
```

```
    case(K_7):
        if(valBuf[0] == K_0)
            highCnt = 5100;
        else
            highCnt = 8100;
        break;
    case(K_8):
        if(valBuf[0] == K_0)
            highCnt = 5400;
        else
            highCnt = 8400;
        break;
    case(K_9):
        highCnt = 5700;
        break;
    default:
        highCnt = highCnt;
        break;
    }
    lowCnt = 60000 - highCnt;   //Calculates low count of duty cycle
}

void TA0_0_IRQHandler()
{
    TIMER_A0->CCTL[0] &= ~TIMER_A_CCTLN_CCIFG;
    static int PREV_ST = 1;

    //Toggles output state
    if(PREV_ST) {
        PREV_ST = 0;
        P6 -> OUT |= BIT1;
        TIMER_A0->CCR[0] += highCnt;
    }
    else {
        PREV_ST = 1;
        P6 -> OUT &= ~BIT1;
        TIMER_A0->CCR[0] += lowCnt;
    }
}
```

Keypad Library

```
/*
 * keypad.h
 * Header file for the keypad driver functions.
 *
 * Date: April 13, 2018
 * Author: Zach Bunce, Garrett Maxon
 */

#ifndef KEYPAD_H_
#define KEYPAD_H_

#define ROWA    BIT0
#define ROWB    BIT1
#define ROWC    BIT2
#define ROWD    BIT3

#define COL1    BIT2
#define COL2    BIT1
#define COL3    BIT0

//Rows: A-D; Columns 1-3
//First row
#define A1      0x13
#define A2      0x15
#define A3      0x16

//Second row
#define B1      0x23
#define B2      0x25
#define B3      0x26

//Third row
#define C1      0x43
#define C2      0x45
#define C3      0x46

//Fourth row
#define D1      0x83
#define D2      0x85
#define D3      0x86

//ASCII hex values for keypad characters
#define K_1     0x31
#define K_2     0x32
#define K_3     0x33

#define K_4     0x34
#define K_5     0x35
#define K_6     0x36
```

```
#define K_7     0x37
#define K_8     0x38
#define K_9     0x39

#define K_Ast   0x2A
#define K_0     0x30
#define K_Pnd   0x23

#define K_NP    0x10

void KEYPAD_INIT();
uint8_t get_Key();
uint8_t chk_Keypad();
uint8_t KEY_LOCATE();

#endif /* KEYPAD_H_ */
```

```
/*
 * keypad.c
 * Call KEYPAD_INIT to enable use of keypad
 * chk_Keypad meant for external use
 * Returns ASCII code of symbol pressed (Including numbers)
 *
 * Date: April 13, 2018
 * Author: Zach Bunce, Garrett Maxon
 */

#include "msp.h"
#include "keypad.h"

static uint8_t keyOut = K_NP;
static int keyFlag = 0;

uint8_t get_Key()
{
    keyOut = chk_Keypad();
    return keyOut;
}

int get_Flag()
{
    return keyFlag;
}

void clear_Flag()
{
    keyFlag = 0;
}
```

```
//Sets P4.0-4.3 as rows and P5.0-5.2 as columns
void KEYPAD_INIT()
{
    P4 -> DIR |= (ROWA | ROWB | ROWC | ROWD);   //Sets DIR reg of the outputs
    P5 -> DIR &= ~(COL1 | COL2 | COL3);         //Clears DIR reg of the inputs
    P5 -> REN |=  (COL1 | COL2 | COL3);         //Enables PU or PD resistor
    P5 -> OUT |=  (COL1 | COL2 | COL3);         //PU or PD is set through PxOUT reg
}

//Decodes row & column of key pressed into ASCII value
uint8_t chk_Keypad()
{
    uint8_t RC_Info = KEY_LOCATE();
    switch(RC_Info)
    {
    case A1:
        return K_1;
    case A2:
        return K_2;
    case A3:
        return K_3;

    case B1:
        return K_4;
    case B2:
        return K_5;
    case B3:
        return K_6;

    case C1:
        return K_7;
    case C2:
        return K_8;
    case C3:
        return K_9;

    case D1:
        return K_Ast;
    case D2:
        return K_0;
    case D3:
        return K_Pnd;
    default:
        return K_NP; //Returns no press; empty value in LCD table
    }
}

//Finds and returns row and column of uppermost key pressed
uint8_t KEY_LOCATE()
{
```

```
    uint8_t i;
    uint8_t col = 0;
    for (i = 1; i <= 0x08; i = i << 1) {
        P4 -> OUT |= (ROWA | ROWB | ROWC | ROWD); //Sets all rows high
        P4 -> OUT &= ~i & (ROWA | ROWB | ROWC | ROWD); //Sets selected row low
        col = P5->IN & 0x07;   //Reads column states into lower nibble
        if (col == 0x07) {
            //Do nothing
        }
        else {
            i = i << 4; //Shift row info into upper nibble
            col |= i;   //Add row info to column info
            return col; //Return column and row info
        }
    }
    return 0;
}

void keypad_ISR()
{
    keyOut = chk_Keypad;
}
```

Delays Library

```
/*
 * delays.h
 * Header file for both the ms and us delay functions.
 *
 * Created on: Apr 9, 2018
 * Author: Zach Bunce, Garrett Maxon
 */

#ifndef DELAYS_H_
#define DELAYS_H_

#define F_1p5_MeHz  15  //Defines various frequency values in almost MHz (10^5)
#define F_3_MeHz    30  //MeHz labels are used to indicate this
#define F_6_MeHz    60  //Blame data type truncation
#define F_12_MeHz   120
#define F_24_MeHz   240
#define F_48_MeHz   480

void delay_ms(int, int);
void delay_us(int, int);

#endif /* DELAYS_H_ */
```

```
/*
 * delays.c
 * Code file for both the ms and us delay functions.
 * Utilizes __delay__cycles
 * Divide us by 2 cause ?
 *
 * Date: April 9, 2018
 * Authors: Zach Bunce, Garret Maxon
 */

#include "msp.h"
#include "delays.h"

//Takes in desired time delay in ms and clock frequency in MeHz
//Accurate to less than 1%
void delay_ms(int time_ms, int freq_MeHz)
{
    int i;
    //Unit Conversion: MeHz * ms = 10^5 * 10^-3 = 10^2 = 100
    switch (freq_MeHz)
    {
    case F_1p5_MeHz:
        //Accurate to 0.1%
        for (i = time_ms; i > 0; i--) {
            __delay_cycles(1500);
        }
        break;
    case F_3_MeHz:
        //Accurate to 0.4%
        for (i = time_ms; i > 0; i--) {
            __delay_cycles(3000);
        }
        break;
    case F_6_MeHz:
        //Accurate to 0.6%
        for (i = time_ms; i > 0; i--) {
            __delay_cycles(6000);
        }
        break;
    case F_12_MeHz:
        //Accurate to 0.7%
        for (i = time_ms; i > 0; i--) {
            __delay_cycles(12000);
        }
        break;
    case F_24_MeHz:
        //Accurate to 0.6%
        for (i = time_ms; i > 0; i--) {
            __delay_cycles(24000);
        }
```

```
        break;
    case F_48_MeHz:
        //Accurate to 0.4%
        //Div by 2 cause ?
        for (i = time_ms; i > 0; i--) {
            __delay_cycles(24000);
        }
        break;
    default:
        break;
    }
}

//Takes in desired time delay in us and clock frequency in MeHz
//1.5, 3, 6, 12 MHz NOT TUNED
void delay_us(int time_us, int freq_MeHz)
{
    int i;
    int j;
    int z;
    int time_fix;
    //Unit Conversion: MeHz * us = 10^5 * 10^-6 = 10^-1 = 0.1; Accounted for in decrement
    switch (freq_MeHz)
    {
    case F_1p5_MeHz:
        time_fix = time_us / 2;
        for (i = time_fix; i > 0; i--) {
            for (j = 10; j > 0; j--) {
                __delay_cycles(15);
            }
        }
        break;
    case F_3_MeHz:
        time_fix = time_us / 2;
        for (i = time_fix; i > 0; i--) {
            __delay_cycles(3);
        }
        break;
    case F_6_MeHz:
        time_fix = time_us / 2;
        for (i = time_fix; i > 0; i--) {
            __delay_cycles(6);
        }
        break;
    case F_12_MeHz:
        time_fix = time_us / 2;
        for (i = time_fix; i > 0; i--) {
            __delay_cycles(12);
        }
        break;
```

```
    case F_24_MeHz:
        //Within +1us 26 < t < 100; Accurate within 1% up to 1000 us
        time_fix = (time_us - 1) / 2;
        for (i = time_fix; i > 0; i--) {
            __delay_cycles(24);
            z++;
            z++;
            z++;
        }
        break;
    case F_48_MeHz:
        //Within +1us 25 < t < 100; Accurate within 2% up to 1000 us
        time_fix = time_us / 2;
        for (i = time_fix; i > 0; i--) {
            __delay_cycles(48);
            z++;
            z++;
            z++;
            z++;
        }
        break;
    default:
        break;
    }
}
```

Set_DCO Library

```
/*
 *  set_DCO.h
 *  Header file for the DCO frequency change function.
 *
 *  Date: April 6, 2018
 *  Author: Zach Bunce, Garret Maxon
 */
#ifndef SET_DCO_H_
#define SET_DCO_H_

#define F_1p5_MeHz  15  //Defines various frequency values in almost MHz (10^5)
#define F_3_MeHz    30  //MeHz labels are used to indicate this
#define F_6_MeHz    60  //Blame data type truncation
#define F_12_MeHz   120
#define F_24_MeHz   240
#define F_48_MeHz   480

void set_DCO(int);

#endif /* SET_DCO_H_ */
```

```
/*
 *  set_DCO.c
 *  Code file for the DCO frequency change function.
 *
 *  Date: April 6, 2018
 *  Authors: Zach Bunce, Garret Maxon
 */

#include "msp.h"
#include "set_DCO.h"

void set_DCO(int freq)
{
    CS->KEY = CS_KEY_VAL;                                           //Unlocks CS regs
    CS->CTL0 = 0;                                                   //Clears CTL0 reg

    switch (freq)
    {
    case F_1p5_MeHz:
        CS->CTL0 = CS_CTL0_DCORSEL_0;                               //Sets DC0 to 1.5 MHz
        CS->CTL1 = CS_CTL1_SELA_2 | CS_CTL1_SELS_3 | CS_CTL1_SELM_3;    //Sets the clock refs
        break;
    case F_3_MeHz:
        CS->CTL0 = CS_CTL0_DCORSEL_1;                               //Sets DC0 to 3 MHz
        CS->CTL1 = CS_CTL1_SELA_2 | CS_CTL1_SELS_3 | CS_CTL1_SELM_3;    //Sets the clock refs
        break;
    case F_6_MeHz:
        CS->CTL0 = CS_CTL0_DCORSEL_2;                               //Sets DC0 to 6 MHz
        CS->CTL1 = CS_CTL1_SELA_2 | CS_CTL1_SELS_3 | CS_CTL1_SELM_3;    //Sets the clock refs
        break;
    case F_12_MeHz:
        CS->CTL0 = CS_CTL0_DCORSEL_3;                               //Sets DC0 to 12 MHz
        CS->CTL1 = CS_CTL1_SELA_2 | CS_CTL1_SELS_3 | CS_CTL1_SELM_3;    //Sets the clock refs
        break;
    case F_24_MeHz:
        CS->CTL0 = CS_CTL0_DCORSEL_4;                               //Sets DC0 to 24 MHz
        CS->CTL1 = CS_CTL1_SELA_2 | CS_CTL1_SELS_3 | CS_CTL1_SELM_3;    //Sets the clock refs
        break;
    case F_48_MeHz:
        // Transition to VCORE Level 1: AM0_LDO --> AM1_LDO
        while ((PCM->CTL1 & PCM_CTL1_PMR_BUSY));
        PCM->CTL0 = PCM_CTL0_KEY_VAL | PCM_CTL0_AMR_1;
        while ((PCM->CTL1 & PCM_CTL1_PMR_BUSY));

        // Configure Flash wait-state to 1 for both banks 0 & 1
        FLCTL->BANK0_RDCTL = (FLCTL->BANK0_RDCTL
            & ~(FLCTL_BANK0_RDCTL_WAIT_MASK)) | FLCTL_BANK0_RDCTL_WAIT_1;
        FLCTL->BANK1_RDCTL = (FLCTL->BANK0_RDCTL
            & ~(FLCTL_BANK1_RDCTL_WAIT_MASK)) | FLCTL_BANK1_RDCTL_WAIT_1;
```

```
        CS->CTL0 = CS_CTL0_DCORSEL_5;                                   //Sets DC0 to 48 MHz
        CS->CTL1 = CS->CTL1
            & ~(CS_CTL1_SELM_MASK | CS_CTL1_DIVM_MASK)| CS_CTL1_SELM_3  //Sets MCLK to DCO
            | CS_CTL1_SELA_2 | CS_CTL1_SELS_3;  //Sets the clock refs
        break;
    default:
        break;
    }
    CS->KEY = 0;                                                        //Locks CS regs
}
```