Daniel Komac

CPE 185-01

Wednesday Lab Section

Lab 3: PICkit3 Microchip

Professor Moyer

Lab Partners:

Anastasiya Herasimava

Slavik Chiley

Thomas Bock

**Part 1 – Intro to PICkit3 PIC18 Microcontroller**

Our package came with the PICkit 3, a 44 Pin Demo Board (with built in microprocessor, resistors, LEDs, etc.), and a USB cord for connection to the computer. For the introduction we need to complete Lessons 1 through 5 from the PICkit 3 Lessons Book provided online. Once we had these programs written and working we needed to demonstrate them to the professor.

Lesson 1- The first lesson showed how to create a simple MPLAB C compiler project in the IDE and use the demo board. Within the first lesson we made a single LED turn on

Lesson 2 – This lesson consisted of taking the Hello LED code and putting a delay on it to make it blink. The delay determined how fast the LED would blink, dependent on how short or long our delay was

Lesson 3 – This lesson consisted of taking all the LEDs on the board and having it rotate singularly between all of them starting from port 0 and then proceeding to port 7, LED 1 to 8

Lesson 4 – The fourth lesson had us use LED rotating from lesson 3 and implement a switch to swap the direction that the LEDs were rotating in. This switch was located on the demo board and was integrated through the PICkit microprocessor.

Lesson 5 – This lesson used an internal clock to control the rotation of the LED lights, still holding the same function from the fourth lesson to switch the direction of the LEDs on a button press. Doing this made it so that we could control the speed of the rotating LEDs by incrementing the clock/timer by factors of x and observing the results rather than a delay.

Lesson 1: Hello LED code

/\*\* C O N F I G U R A T I O N B I T S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma config FOSC = INTIO67

#pragma config WDTEN = OFF, LVP = OFF, MCLRE = ON

/\*\* I N C L U D E S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <xc.h>

/\*\* D E C L A R A T I O N S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main (void)

{

TRISD = 0b01111111; // PORTD bit 7 to output (0); bits 6:0 are inputs (1)

LATDbits.LATD7 = 1; // Set LAT register bit 7 to turn on LED

//LATDbits.LATD0 = 1;

//LATDbits.LATD4 = 1;

while (1) ; //infinite loop

}

Lesson 2: Blinking LED code

/\*\* C O N F I G U R A T I O N B I T S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma config WDTEN = OFF, LVP = OFF, MCLRE = ON

#pragma config FOSC = INTIO67

/\*\* D E C L A R A T I O N S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// none

/\*\* I N C L U D E S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <xc.h>

#include "delays.h"

void main (void)

{

TRISD = 0x7F; // PORTD bit 7 to output (0) ; bits 6:0 are inputs (1)

LATD = 0x80 ;

while (1)

{

LATD ^= 0x80 ; // toggle LATD

delay(10); // Delay loop

}

}

Lesson 3: Rotating LED code

/\*\* C O N F I G U R A T I O N B I T S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma config FOSC = INTIO67

#pragma config WDTEN = OFF, LVP = OFF, MCLRE = ON

/\*\* I N C L U D E S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <xc.h> // compiler header

#include "delays.h" // contains function prototypes for functions is delay.c

/\*\* V A R I A B L E S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned char LED\_Number; // 8-bit variable

/\*\* D E C L A R A T I O N S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned char LED\_LookupTable[8] = {0x01, 0x02, 0x04, 0x08,

0x10, 0x20, 0x40, 0x80};

/\* Code \*/

void main (void)

{

LED\_Number = 0; // initialize

TRISD = 0b00000000; // PORTD bits 7:0 are all outputs (0)

while (1)

{

unsigned char data ;

// use lookup table to output one LED on based on LED\_Number value

data = LED\_LookupTable[LED\_Number];

LATD = data ;

LED\_Number++; // rotate display by 1

if (LED\_Number == 8)

{

LED\_Number = 0; // go back to LED 0.

}

delay(20); // Delay 50 x 1000 = 50,000 cycles; 200ms @ 1MHz

}

}

Lesson 4: Switch Input code

/\*\* C O N F I G U R A T I O N B I T S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma config WDTEN = OFF, LVP = OFF, MCLRE = ON

#pragma config FOSC = INTIO67

/\*\* I N C L U D E S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <xc.h>

#include "delays.h"

/\*\* V A R I A B L E S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned char LED\_Display; // 8-bit variable

unsigned char Switch\_Count ;

/\*\* D E C L A R A T I O N S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void wait\_for\_SW1\_press(void);

void main (void)

{

LED\_Display = 1; // initialize variable

TRISD = 0b00000000; // Turn PORTD output port to drive

// .. the LEDs

ANSELH = 0x00; // ensure all pins are digital Pins

TRISBbits.TRISB0 = 1; // Turn RB0 into an input pin for SW1

while (1)

{

//LED\_Display = ~LED\_Display;

LATD = LED\_Display; // output LED\_Display value to PORTD LEDs

LED\_Display <<= 1; // rotate display by 1

if (LED\_Display == 0)

LED\_Display = 1; // rotated bit out, so set bit 0

wait\_for\_SW1\_press(); // wait until SW1 is pressed and released

}

}

void wait\_for\_SW1\_press(void)

{ while( !PORTBbits.RB0) ; // wait until SW1 is pushed down

delay(2) ; // delay for de-bounce

while( PORTBbits.RB0) ; // wait for release

}

Lesson 5: Using Timer code

/\*\* C O N F I G U R A T I O N B I T S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma config FOSC = INTIO67

#pragma config WDTEN = OFF, LVP = OFF, MCLRE = ON

/\*\* I N C L U D E S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <xc.h>

//#include "delays.h" // no longer being used.

/\*\* V A R I A B L E S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// declare statically allocated uinitialized variables

unsigned char LED\_Display; // 8-bit variable

unsigned char Direction ;

#define LEFT2RIGHT 1

#define RIGHT2LEFT 0

unsigned char SwitchPressed ;

#define TRUE 1

#define FALSE 0

/\*\* D E C L A R A T I O N S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main (void)

{

Direction = LEFT2RIGHT;

SwitchPressed = FALSE;

LED\_Display = 1; // initialize variable

// Init I/O

TRISD = 0b00000000; // PORTD bits 7:0 are all outputs (0)

ANSELH = 0x00; // AN8-12 are digital inputs (AN12 on RB0)

TRISBbits.TRISB0 = 1; // PORTB bit 0 (connected to switch) is input (1)

// Init Timer

INTCONbits.TMR0IF = 0; // clear roll-over interrupt flag

T0CON = 0b00001000; // no prescale - increments every instruction clock

//T0CON = 0b00000001; // prescale 1:4 - four times the delay.

TMR0H = 0; // clear timer - always write upper byte first

TMR0L = 0;

T0CONbits.TMR0ON = 1; // start timer

while (1)

{

if (Direction == LEFT2RIGHT)

{

LED\_Display <<= 1; // rotate display by 1 from 0 to 7

if (LED\_Display == 0)

LED\_Display = 1; // rotated bit out, so set bit 0

}

if (Direction == RIGHT2LEFT)

{

LED\_Display >>= 1; // rotate display by 1 from 7 to 0

if (LED\_Display == 0)

LED\_Display = 0x80; // rotated bit out, so set bit 7

}

LATD = LED\_Display; // output LED\_Display value to PORTD LEDs

do

{ // poll the switch while waiting for the timer to roll over.

if (PORTBbits.RB0 == 1)

{ // look for switch released.

SwitchPressed = FALSE;

}

else if (SwitchPressed == FALSE) // && (Switch\_Pin == 0) due to if-else

{ // switch was just pressed

SwitchPressed = TRUE;

// change direction

if (Direction == LEFT2RIGHT)

Direction = RIGHT2LEFT;

else

Direction = LEFT2RIGHT;

}

} while (INTCONbits.TMR0IF == 0);

// Timer expired

INTCONbits.TMR0IF = 0; // Reset Timer flag

}

}

**Part 2 – PICkit 3 Debug**

The debug lesson was paired with Lesson 5(Lesson 5/6) in order to help us understand a fundamental feature for understanding how our hardware will interact with the code that we have written.

Lesson 6: Using PICkit3 Debug Express

The debugger allows us to step into code and analysis specific bits or registers being used by the user so that we can see the outcome of our code. This is obtainable through breakpoints and using the viewer window to observe each variable that we had declared and see how they change throughout the process.

**Part 3 – ADC, Interrupts, and PWM**

Part 3 consisted of completing two more advanced lessons, Lessons 7 and 12.

Lesson 7 – This lesson used Analog-to-Digital Convertor located in the chip. We used it to determine the demo board’s potentiometer voltage. We used this to determine the LED rotation speed. An ADC is used to convert analog voltage into a digital number to represent voltage. Do so allowed us to control the rotation of the LEDs with the potentiometer and fluctuate faster or slower corresponding to position.

Lesson 8 – Interrupts are hardware based functions that intervene when the hardware needs to execute a specific set of instructions. Doing this allows for the hardware to handle situations of importance by giving priority to the event. In Lesson 8 this was used instead of pooling for the switch press or Timer0 event.

Lesson 12 – This lesson introduced us to Pulse Width Modulation (PWM). The PWM is a square wave of a given frequency where the duty cycle of the period is varied. This duty cycle is what made the LED light up and pulse as the frequency changed.

**Lesson 7 ADC code:**

/\*\* C O N F I G U R A T I O N B I T S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma config FOSC = INTIO67

#pragma config WDTEN = OFF, LVP = OFF, MCLRE = ON

/\*\* I N C L U D E S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <xc.h>

#include "07 ADC.h" // header file

/\*\* V A R I A B L E S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned char LED\_Display; // 8-bit variable

/\*\* D E C L A R A T I O N S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main (void)

{

Direction = LEFT2RIGHT;

SwitchPressed = FALSE;

LED\_Display = 1; // initialize

// Init I/O

TRISD = 0b00000000; // PORTD bits 7:0 are all outputs (0)

TRISAbits.TRISA0 = 1; // TRISA0 input

TRISBbits.TRISB0 = 1; // PORTB bit 0 (connected to switch) is input (1)

// Init Timer0

Timer0\_Init(); // set up timer 0

// Init ADC

ADC\_Init();

while (1)

{

if (Direction == LEFT2RIGHT)

{

LED\_Display <<= 1; // rotate display by 1 from 0 to 7

if (LED\_Display == 0)

LED\_Display = 1; // rotated bit out, so set bit 0

}

if (Direction == RIGHT2LEFT)

{

LED\_Display >>= 1; // rotate display by 1 from 7 to 0

if (LED\_Display == 0)

LED\_Display = 0x80; // rotated bit out, so set bit 7

}

LATD = LED\_Display; // output LED\_Display value to PORTD LEDs

do

{ // poll the switch while waiting for the timer to roll over.

if (Switch\_Pin == 1)

{ // look for switch released.

SwitchPressed = FALSE;

}

else if (SwitchPressed == FALSE) // && (Switch\_Pin == 0) due to if-else

{ // switch was just pressed

SwitchPressed = TRUE;

// change direction

if (Direction == LEFT2RIGHT)

Direction = RIGHT2LEFT;

else

Direction = LEFT2RIGHT;

}

} while (INTCONbits.TMR0IF == 0);

// Timer expired

INTCONbits.TMR0IF = 0; // Reset Timer flag

// Take an ADC conversion and use it to set Timer0

TMR0H = ADC\_Convert(); // MSB from ADC

TMR0L = 0; // LSB = 0

}

}

void Timer0\_Init(void)

{

INTCONbits.TMR0IF = 0; // clear roll-over interrupt flag

T0CON = 0b00000001; // prescale 1:4 - about 1 second maximum delay.

TMR0H = 0; // clear timer - always write upper byte first

TMR0L = 0;

T0CONbits.TMR0ON = 1; // start timer

}

void ADC\_Init(void)

{ // initialize the Analog-To-Digital converter.

// First, we need to make sure the AN0 pin is enabled as an analog input

// as the demo board potentiometer is connected to RA0/AN0

// Don't forget that RB0/AN12 must be digital!

ANSEL = 0; //turn off all other analog inputs

ANSELH = 0;

ANSELbits.ANS0 = 1; // turn on RA0 analog

// Sets bits VCFG1 and VCFG0 in ADCON1 so the ADC voltage reference is VSS to VDD

ADCON1 = 0;

// The ADC clock must as short as possible but still greater than the

// minimum TAD time, datasheet parameter 130. At the time this lesson was

// written TAD minimum for the PIC18F45K20 is 1.4us.

// At 1MHz clock, selecting ADCS = FOSC/2 = 500kHz. One clock period

// 1 / 500kHz = 2us, which greater than minimum required 1.4us.

// So ADCON2 bits ADCS2-0 = 000

//

// The ACQT aquisition time should take into accound the internal aquisition

// time TACQ of the ADC, datasheet paramter 130, and the settling time of

// of the application circuit connected to the ADC pin. Since the actual

// settling time of the RC circuit with the demo board potentiometer is very

// long but accuracy is not very important to this demo, we'll set ACQT2-0 to

// 20TAD = 111

//

// ADFM = 0 so we can easily read the 8 Most Significant bits from the ADRESH

// Special Function Register

ADCON2 = 0b00111000;

// Select channel 0 (AN0) to read the potentiometer voltage and turn on ADC

ADCON0 = 0b00000001;

}

unsigned char ADC\_Convert(void)

{ // start an ADC conversion and return the 8 most-significant bits of the result

ADCON0bits.GO\_DONE = 1; // start conversion

while (ADCON0bits.GO\_DONE == 1); // wait for it to complete

return ADRESH; // return high byte of result

}

**Lesson 8 Interrupts code:**

/\*\* C O N F I G U R A T I O N B I T S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma config FOSC = INTIO67

#pragma config WDTEN = OFF, LVP = OFF, MCLRE = ON

/\*\* I N C L U D E S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <xc.h>

#include "08 Interrupts.h" // header file

/\*\* V A R I A B L E S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned char Direction = LEFT2RIGHT; // global now to access in Interrupts

unsigned char LED\_Display ;

/\*\* D E C L A R A T I O N S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//#pragma code // declare executable instructions

void main (void)

{

LED\_Display = 1; // initialize

// Init I/O

TRISD = 0b00000000; // PORTD bits 7:0 are all outputs (0)

TRISAbits.TRISA0 = 1; // TRISA0 input

INTCON2bits.RBPU = 0; // enable PORTB internal pullups

WPUBbits.WPUB0 = 1; // enable pull up on RB0

// ADCON1 is now set up in the InitADC() function.

TRISBbits.TRISB0 = 1; // PORTB bit 0 (connected to switch) is input (1)

// Init Timer0

Timer0\_Init(); // now enables interrupt.

// Init ADC

ADC\_Init();

// Set up switch interrupt on INT0

INTCON2bits.INTEDG0 = 0; // interrupt on falling edge of INT0 (switch pressed)

INTCONbits.INT0IF = 0; // ensure flag is cleared

INTCONbits.INT0IE = 1; // enable INT0 interrupt

// NOTE: INT0 is ALWAYS a high priority interrupt

// Set up global interrupts

RCONbits.IPEN = 1; // Enable priority levels on interrupts

INTCONbits.GIEL = 1; // Low priority interrupts allowed

INTCONbits.GIEH = 1; // Interrupting enabled.

while (1)

{ // we update the port pins in our "background" loop while the interrupts

// handle the switch and timer.

LATD = LED\_Display; // output LED\_Display value to PORTD LEDs

}

}

void Timer0\_Init(void)

{

// Set up Interrupts for timer

INTCONbits.TMR0IF = 0; // clear roll-over interrupt flag

INTCON2bits.TMR0IP = 0; // Timer0 is low priority interrupt

INTCONbits.TMR0IE = 1; // enable the Timer0 interrupt.

// Set up timer itself

T0CON = 0b00000001; // prescale 1:4 - about 1 second maximum delay.

TMR0H = 0; // clear timer - always write upper byte first

TMR0L = 0;

T0CONbits.TMR0ON = 1; // start timer

}

void ADC\_Init(void)

{ // initialize the Analog-To-Digital converter.

ANSEL = 0; //turn off all other analog inputs

ANSELH = 0;

ANSELbits.ANS0 = 1; // turn on RA0 analog

ADCON2 = 0b00111000;

ADCON0 = 0b00000001;

}

unsigned char ADC\_Convert(void)

{ // start an ADC conversion and return the 8 most-significant bits of the result

ADCON0bits.GO\_DONE = 1; // start conversion

while (ADCON0bits.GO\_DONE == 1); // wait for it to complete

return ADRESH; // return high byte of result

}

// -------------------- Interrupt Service Routines --------------------------

//#pragma interrupt InterruptServiceHigh // "interrupt" pragma also for high priority

void interrupt InterruptServiceHigh(void)

{

// Check to see what caused the interrupt

// (Necessary when more than 1 interrupt at a priority level)

// Check for INT0 interrupt

if (INTCONbits.INT0IF)

{

// clear (reset) flag

INTCONbits.INT0IF = 0;

// change directions

if (Direction == LEFT2RIGHT)

{

Direction = RIGHT2LEFT; // change direction

}

else // (Direction == RIGHT2LEFT)

{

Direction = LEFT2RIGHT; // change direction

}

}

// Check for another interrupt, examples:

// if (PIR1bits.TMR1IF) // Timer 1

// if (PIR1bits.ADIF) // ADC

return ;

} // return from high-priority interrupt

void interrupt low\_priority InterruptServiceLow(void)

{

// Check to see what caused the interrupt

// (Necessary when more than 1 interrupt at a priority level)

// Check for Timer0 Interrupt

if (INTCONbits.TMR0IF)

{

INTCONbits.TMR0IF = 0; // clear (reset) flag

// Take an ADC conversion and use it to set Timer0

TMR0H = ADC\_Convert(); // MSB from ADC

TMR0L = 0; // LSB = 0

// update display variable

if (Direction == LEFT2RIGHT)

{

LED\_Display <<= 1; // rotate display by 1 from 0 to 7

if (LED\_Display == 0)

LED\_Display = 1; // rotated bit out, so set bit 0

}

else // (Direction == RIGHT2LEFT)

{

LED\_Display >>= 1; // rotate display by 1 from 7 to 0

if (LED\_Display == 0)

LED\_Display = 0x80; // rotated bit out, so set bit 7

}

}

}

**Lesson 12 PWM code:**

/\*\* C O N F I G U R A T I O N B I T S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma config FOSC = INTIO67

#pragma config WDTEN = OFF, LVP = OFF, MCLRE = ON

/\*\* I N C L U D E S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <xc.h>

#include "delays.h"

/\*\* V A R I A B L E S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned char brightness = 125; // = 0x7D

/\*\* D E C L A R A T I O N S \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main (void)

{

// Set RD7/P1D pin output so P1D PWM output drives LED7

TRISDbits.TRISD7 = 0;

// Set up 8-bit Timer2 to generate the PWM period (frequency)

T2CON = 0b00000111;// Prescale = 1:16, timer on, postscale not used with CCP module

PR2 = 249; // Timer 2 Period Register = 250 counts

// Thus, the PWM frequency is:

// 1MHz clock / 4 = 250kHz instruction rate.

// (250kHz / 16 prescale) / 250) = 62.5Hz, a period of 16ms.

// The Duty Cycle is controlled by the ten-bit CCPR1L<7,0>:DC1B1:DC1B0

// 50% Duty Cycle = 0.5 \* (250 \* 4) = 500

CCPR1L = 0x7D; // The 8 most significant bits of the value 500 = 0x1F4 are 0x7D

// The 2 least significant bits of the value (0b00) are written

// to the DC1B1 and DC1B0 bits in CCP1CON

CCP1CON = 0b01001100;

// P1Mx = 01 Full-Bridge output forward, so we get the PWM

// signal on P1D to LED7. Only Single Output (00) is needed,

// but the P1A pin does not connect to a demo board LED

// CCP1Mx = 1100, PWM mode with P1D active-high.

// The LED brightness is affected by by the Duty Cycle, which determines how much

// of each 16ms period it is on and how much it is off. As the duty cycle gets

// less than 50%, it is off more than it is on so the LED becomes dimmer. As the

//duty cycle increases above 50%, it is on more than off, so it gets brighter.

//

// This increases the brightness over 2 seconds, then decreases it over the next 2 seconds

// Updating the CCPR1L value more than once per 16ms period has no benefit, so we'll update

// it a total of 125 times, once per period, which works out to 2 seconds.

//

// Although we have nearly ten bits of resolution in the duty cycle (1000 counts)

// we'll increment the duty cycle by 8 each time as we only have 125 levels over the

// 2 second period.

while(1)

{

while (brightness < 250)

{

brightness += 2;

CCPR1L = brightness; // + 8 including 2 bits DC1Bx in CCP1CON

PIR1bits.TMR2IF = 0; // clear interrupt flag; set on every TMR2 = PR2 match

while (PIR1bits.TMR2IF == 0); // watch for match, which is end of period.

}

delay(13); // delay about 250ms at peak brightness, just for effect!

// decrease brightness over 2 seconds.

while (brightness > 1)

{

brightness -= 2;

CCPR1L = brightness; // - 8 including 2 bits DC1Bx in CCP1CON

PIR1bits.TMR2IF = 0; // clear interrupt flag; set on every TMR2 = PR2 match

while (PIR1bits.TMR2IF == 0); // watch for match, which is end of period.

}

delay(13); // delay about 250ms at dimmest, it gives a better effect!

};

}

**Conclusion**

The PICkit3 lab gave us a general overview of the microchip processor showing us that it can utilize functions of the demo board and also provides and on-board debugger for revising and making sure our code is applied properly. The labs were fairly simple but the integration of the software was rather complicated as it took many hours to get it running. Once we got Lab 1-5 working everything else came fairly easily. Hello world LED and rotating LEDS was just a matter of turning on specific bits on the demo board. Incorporating the chips timers, ADC, and PWM was fairly complicated but it became easier with a few examples and help from the online lessons from Microchip Technology.