Compe490 Mini Project

Yusuf Shaikh(816921177)

Thursday, March 23, 2017

Lab session: Monday 9:00a.m.

For the Mini-Project we were exploring the capabilities of a PIC Microcontroller to alter a fans speed based on the temperature of the heater. To do this, the Mini-Project was broken into various parts. I began by creating a Wheatstone bridge that incorporated a Thermometer on one of the legs and an interweaved an Op-Amp. I recorded the output voltage from the Op-Amp between the temperature readings of 78°F - 130°F, receiving a voltage range of 0.8V – 4.60V.

The next step of the process was to use the output of the Op-Amp as the input for the PIC microcontroller. The input that was coming in would be an analog signal and had to be converted to a Digital Signal. The best way to do this process was to have the input analog signal come enter though an ADC register and be converted by the PIC Microcontroller. To do this I had to analyze the microcontroller data sheet and see what pins could be allocated to the ADC register. I saw ADCON0 could be allocated to multiple pins; so, I decided to set ADCON0 to ANA1 (RA1/Pin12). Do this I set the last 6 bits of ADCON0 to 000001, enabling Pin12 as an ADC input, and set the First two bits to 11, signifying that ADC is enabled and ready to convert.

Once the input analog signal had been converted it was sent to a holding register (ADRESH) where it could be stored and used for comparisons. The values stored in ADRESH were not the same as the voltage values originally inputted, they had been converted to a digital value. The conversion formula from Analog to Digital was *255\*(Voltage/5).* The 255 signify the maximum number in an 8 bit binary representation and the 5 signifies the 5 volts being provided to the chip. The Mini-Lab instructions asked for 16 increments from 80°F - 130°F. Knowing at 80°F my voltage was 1.0V and at 130°F my voltage was 4.60V I simply created a MATALB script that that divided *(4.60V-1.0V)/16* and incremented from 1.00V to 4.60V. The script then multiplied each value by *255/5* to convert it to a digital value.

For the actual development of the code, I have an *if…else if…else* statements that compared the converted digital voltage to a hardcoded value(started at 41 and incremented in 16 steps of *(255-41)/16*. A global variable ‘DigitalSignal’ was created, and was set in the interrupt as ADRESH. Each time ADRESH got a new value, the Main() was interrupted and the value of ADRESH was set into ‘DigitalSignal’. The Last thing that had to be set was the output. To do, I once again analyzed the data sheet for PWM, I set PWM to RC5/pin 5 by labeling PWM5DCH = 0. Based on what *if…else if…else* statement was satisfied the pulse width would be set correspondingly.

Ex: if(DigitalSignal < 61)

PWM5DCH = 127.5;

After the maximum Digital Signal value the PWM5DCH would be set to 255. The last thing that was left was the burst mode that would occur during the initialization of the system. The Burst mode is setup using a global ‘Burst’ flag variable. Initially the ‘Burst’ variable is set to zero, and in the main while(1) loop if the ‘Burst’ variable is 0, a series of configurations occur.

* PWM5DCH is set to 255(The Maximum Duty Cycle)
* A Delay is set for 3000ms (3seconds)
* Burst Mode is set to 1.

MATLAB Conversion code:

x = 1.0;

xx = 63.75;

for i = 1:16

y = (x/5) \* 255;

fprintf([num2str(i) '-->' num2str(x) '-->' num2str(y) '-->' num2str(xx) '\n'])

x = x + .24;

xx = xx + 12.75;

end

Microcontroller code:

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Filename: FanMiniProject.c

//Author: Yusuf Shaikh

//Date: 9-21-16

//Version: 1.0

//Device: 16f18324

//Description: Blink an LED using a time wasting loop 5Hz

//Complier: XC8

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Configuration

// Window -> PIC Memory Views -> Configuration Bits

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// CONFIG1

#pragma config FEXTOSC = OFF // FEXTOSC External Oscillator mode Selection bits->Oscillator not enabled

#pragma config RSTOSC = HFINT1 // Power-up default value for COSC bits->HFINTOSC

#pragma config CLKOUTEN = OFF // Clock Out Enable bit->CLKOUT function is disabled; I/O or oscillator function on OSC2

#pragma config CSWEN = ON // Clock Switch Enable bit->Writing to NOSC and NDIV is allowed

#pragma config FCMEN = ON // Fail-Safe Clock Monitor Enable->Fail-Safe Clock Monitor is enabled

// CONFIG2

#pragma config MCLRE = ON // Master Clear Enable bit->MCLR/VPP pin function is MCLR; Weak pull-up enabled

#pragma config PWRTE = OFF // Power-up Timer Enable bit->PWRT disabled

#pragma config WDTE = OFF // Watchdog Timer Enable bits->WDT disabled; SWDTEN is ignored

#pragma config LPBOREN = OFF // Low-power BOR enable bit->ULPBOR disabled

#pragma config BOREN = ON // Brown-out Reset Enable bits->Brown-out Reset enabled, SBOREN bit ignored

#pragma config BORV = LOW // Brown-out Reset Voltage selection bit->Brown-out voltage (Vbor) set to 2.45V

#pragma config PPS1WAY = ON // PPSLOCK bit One-Way Set Enable bit->The PPSLOCK bit can be cleared and set only once; PPS registers remain locked after one clear/set cycle

#pragma config STVREN = ON // Stack Overflow/Underflow Reset Enable bit->Stack Overflow or Underflow will cause a Reset

#pragma config DEBUG = OFF // Debugger enable bit->Background debugger disabled

// CONFIG3

#pragma config WRT = OFF // User NVM self-write protection bits->Write protection off

#pragma config LVP = ON // Low Voltage Programming Enable bit->Low Voltage programming enabled. MCLR/VPP pin function is MCLR. MCLRE configuration bit is ignored.

// CONFIG4

#pragma config CP = OFF // User NVM Program Memory Code Protection bit->User NVM code protection disabled

#pragma config CPD = OFF // Data NVM Memory Code Protection bit->Data NVM code protection disabled

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Includes

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <xc.h>

#include <stdint.h>

#include <stdbool.h>

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Defines

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define \_XTAL\_FREQ 4000000

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Global Variables

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

uint16\_t count = 0;

int Burst = 0; // Enable Change PWM Mode

uint16\_t DigitalValue = 255;

void interrupt my\_isr(void)

{

if(TMR0IE && TMR0IE)

{

TMR0IF = 0; //clear flag

DigitalValue = ADRESH;

ADGO = 1;

Burst = 1;

return;

}

}

void main(void)

{

OSCCON1 = 0x60; // HFINTOSC

OSCFRQ = 0x03; // HFFRQ 4\_MHz;

TRISC = 0b11111111; // Disable Output Drivers

RC5PPS = 0b00010; // PWM5 on RC5

PPSLOCK = 1;

TMR0H = 156;

T0CON0 = 0b10000000;

T0CON1 = 0b01011010;

ADCON0 = 0b00000111;

ADCON1 = 0b01010000;

PWM5CON = 0;

PR2 = 0b11111111;

TMR2ON = 1;

PWM5DCH = 0;

PWM5DCL = 0;

PWM5CON = 0b10000000;

TRISC5 = 0;

TMR0IE = 1;

PEIE = 1;

GIE = 1;

ADGO = 1;

while (1)

{

if (Burst == 1) // if flag is enabled

{

if (DigitalValue < 41)

PWM5DCH = 41;

else if (DigitalValue < 51)

PWM5DCH = 53.75;

else if (DigitalValue < 63)

PWM5DCH = 66.5;

else if (DigitalValue < 75)

PWM5DCH = 79.25;

else if (DigitalValue < 88)

PWM5DCH = 92;

else if (DigitalValue < 100)

PWM5DCH = 104.75;

else if (DigitalValue < 112)

PWM5DCH = 117.5;

else if (DigitalValue < 124)

PWM5DCH = 130.25;

else if (DigitalValue < 137)

PWM5DCH = 143;

else if (DigitalValue < 149)

PWM5DCH = 155.75;

else if (DigitalValue < 161)

PWM5DCH = 168.5;

else if (DigitalValue < 173)

PWM5DCH = 181.25;

else if (DigitalValue < 186)

PWM5DCH = 194;

else if (DigitalValue < 198)

PWM5DCH = 206.75;

else if (DigitalValue < 210)

PWM5DCH = 219.5;

else if (DigitalValue < 222)

PWM5DCH = 232.25;

else if (DigitalValue < 235)

PWM5DCH = 245; // Possibly Change

else

PWM5DCH = 255;

}

else

{

PWM5DCH = 255; //Full burst mode

\_\_delay\_ms(3000);

Burst = 1;

}

//Then leave burst mode

}

}