HOLY SPIRIT UNIVERSITY OF KASLIK

Faculty of Engineering

Department of Computer Engineering

Digital Fuel Gauge

GEL 575

Presented to:

Dr. Bechara NEHME

Prepared By:

Adib RACHID and Charbel ZALAKET

USEK-Fall 2017-2018

Problem:

Using the PIC24FJ256DA210 microcontroller, we would like to display the percentage of fuel left in a car reservoir. For security reasons, we will use water reservoirs already developed in the lab.

- The display will be made on **two seven segment** displays (max 99%).
- Use the available pins on the PICtail plus of **PORTB** to measure the level.
- The microcontroller must generate an alarm (**Lighting a LED**) if the fuel level is less than a certain threshold.
- The fuel level must be updated each **2.5 seconds**.
- Make sure that the fuel fluctuations in the reservoir **does not affect** your readings.

Solution:

```
/*
 * File:   newmain.c
 * Author: adibrachid- charbelzalaket
 * Created on December 6, 2017, 4:07 PM
 */
// PIC24FJ256DA210 Configuration Bit Settings
// 'C' source line config statements
// CONFIG4
// CONFIG3
#pragma config WPFP = WPFP255          // Write Protection Flash Page Segment Boundary
(Highest Page (same as page 170))
#pragma config SOSCSEL = SOSC          // Secondary Oscillator Power Mode Select
(Secondary oscillator is in Default (high drive strength) Oscillator mode)
#pragma config WUTSEL = LEG            // Voltage Regulator Wake-up Time Select (Default
regulator start-up time is used)
#pragma config ALTPMP = ALPMPDIS       // Alternate PMP Pin Mapping (EPMP pins are in
default location mode)
#pragma config WPDIS = WPDIS           // Segment Write Protection Disable (Segmented code
protection is disabled)
#pragma config WPCFG = WPCFGDIS        // Write Protect Configuration Page Select (Last
page (at the top of program memory) and Flash Configuration Words are not write-protected)
#pragma config WPEND = WPENDMEM        // Segment Write Protection End Page Select
(Protected code segment upper boundary is at the last page of program memory; the lower
boundary is the code page specified by WPFP)
// CONFIG2
#pragma config POSCMOD = HS            // Primary Oscillator Select (HS Oscillator mode is
selected)
#pragma config IOL1WAY = ON            // IOLOCK One-Way Set Enable (The IOLOCK bit
(OSCCON<6>) can be set once, provided the unlock sequence has been completed. Once set, the
Peripheral Pin Select registers cannot be written to a second time.)
#pragma config OSCIOFNC = OFF          // OSCO Pin Configuration (OSCO/CLKO/RC15
functions as CLKO (FOSC/2))
#pragma config FCKSM = CSDCMD          // Clock Switching and Fail-Safe Clock Monitor
(Clock switching and Fail-Safe Clock Monitor are disabled)
#pragma config FNOSC = PRI             // Initial Oscillator Select (Primary Oscillator (XT, HS,
EC))
#pragma config PLL96MHZ = ON           // 96MHz PLL Startup Select (96 MHz PLL is enabled
automatically on start-up)
#pragma config PLLDIV = DIV12          // 96 MHz PLL Prescaler Select (Oscillator input is
divided by 12 (48 MHz input))
#pragma config IESO = ON               // Internal External Switchover (IESO mode (Two-Speed
Start-up) is enabled)
// CONFIG1
#pragma config WDTPS = PS32768         // Watchdog Timer Postscaler (1:32,768)
```

```
#pragma config FWPSA = PR128            // WDT Prescaler (Prescaler ratio of 1:128)
#pragma config ALTVREF = ALTVREDIS      // Alternate VREF location Enable (VREF is on
a default pin (VREF+ on RA9 and VREF- on RA10))
#pragma config WINDIS = OFF             // Windowed WDT (Standard Watchdog Timer
enabled,(Windowed-mode is disabled))
#pragma config FWDTEN = ON              // Watchdog Timer (Watchdog Timer is enabled)
#pragma config ICS = PGx1               // Emulator Pin Placement Select bits (Emulator functions
are shared with PGEC1/PGED1)
#pragma config GWRP = OFF               // General Segment Write Protect (Writes to program
memory are allowed)
#pragma config GCP = OFF                // General Segment Code Protect (Code protection is
disabled)
#pragma config JTAGEN = ON              // JTAG Port Enable (JTAG port is enabled)
// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

#include <xc.h>
#define FCY 4000000UL
#include <libpic30.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <p24FJ256DA210.h>

int value;
int value_tmp;
int array_values[25];
int i=0;

int threshold=20;
int main(int argc, char** argv) {
   //Initialisation

   //PortB
   //7 segments output on portB  15:10 - 5
   TRISB=0;
   //Port D
   //12:10-8:3 sensors

   TRISD=1;
   //Inputs
   //A:15 output LED  - 14-10 balaillage
   TRISA=0;
   TRISG=1;
   TRISE=1;
   TRISAbits.TRISA4=1;
```

```
    //Timer2 of 0.1sec for inserting in table for the average
    /*Fosc/2=4MHZ
    period=2.5x10^-7 sec
    Prescale: 256 => periode: 6.4x10^-5
    PR2=0x061A H = 1562 =>0.1sec
     */
    T2CON = 0x0030;
    TMR2 = 0x00;
    PR2 = 0x061A;
    IPC1bits.T2IP=0x01;
    IFS0bits.T2IF = 0;
    IEC0bits.T2IE = 1;
    T2CONbits.TON = 1;
    //read initial value from sensor
    value=read_from_sensor();
    while(1){
        display(value);
    }
    return 0;
}

int read_from_sensor(){
    //read and get maximum bit.
    if(PORTDbits.RD11==1)//t
        value_tmp=99;
    else if(PORTDbits.RD13==1)//t
        value_tmp=90;
    else if(PORTGbits.RG14==1)//t
        value_tmp=80;
    else if(PORTDbits.RD8==1)//t
        value_tmp=60;
    else if(PORTEbits.RE1==1)
        value_tmp=50;
    else if(PORTEbits.RE4==1)
        value_tmp=40;
    else if(PORTDbits.RD5==1)//t
        value_tmp=20;
    else if(PORTDbits.RD4==1)//t
        value_tmp=10;
    else if(PORTDbits.RD3==1)//t
        value_tmp=5;
    else value_tmp=0;
    return value_tmp ;
}

void display(int dec){
    int d1,d2;
    d1=dec%10;
    d2=(dec/10)%10;
    int ii=0;
    while(ii<20){
        show(d2);
        PORTAbits.RA14=1;
        __delay_ms(10);
        PORTAbits.RA14=0;
        show(d1);
        PORTAbits.RA10=1;
        __delay_ms(10);
        PORTAbits.RA10=0;
        ii++;
    }
}

void show(int dec){
    switch(dec){
    case 0:
        //PORTD=0xFE07;
        //xxxx x1x0 0 0 0 0 0xxx  0 turn on
        PORTB=0x8000;
        break;
    case 1:
        //PORTD=0xFFCF;
        //xxxx x1x1 1 1 0 0 1xxx   1 turn on
        PORTB=0xF330;
        break;
    case 2:
        //PORTD=0xFB27;
        //xxxx x0x1 0 0 1 0 0xxx   2 turn on
        PORTB=0x4BDF;
        break;
    case 3:
        //PORTD=0xFB87;
        //xxxx x0x1 1 0 0 0 0xxx   3 turn on
        PORTB=0x63DF;
        break;
    case 4:
        //PORTD=0xFACF;
        //xxxx x0x0 1 1 0 0 1xxx   4 turn on
        PORTB=0x33FF;
        break;
    case 5:
        //PORTD=0xFA97;
        //xxxx x0x0 1 0 0 1 0xxx   5 turn on
        PORTB=0x26DF;
        break;
    case 6:
        //PORTD=0xFA17;
        //xxxx x0x0 0 0 0 1 0xxx   6 turn on
        PORTB=0x07DF;
        break;
    case 7:
        //PORTD=0xFFC7;
        //xxxx x1x1 1 1 0 0 0xxx   7 turn on
        PORTB=0xF3DF;
        break;
    case 8:
        //PORTD=0xFA07;
        //xxxx x0x0 0 0 0 0 0xxx   8 turn on
        PORTB=0x03DF;
        break;
    case 9:
        //PORTD=0xFA87;
        //xxxx x0x0 1 0 0 0 0xxx   9 turn on
        PORTB=0x23DF;
        break;
    }
}

void __attribute__((__interrupt__, __shadow__)) _T2Interrupt(void) {
    //interrupt of 0.1sec

    if(i<25){
        array_values[i++]=read_from_sensor();
    }
    else{
       value=get_average();
    if(value<=threshold)//threshold
        PORTAbits.RA15=1;
    else
        PORTAbits.RA15=0;
    i=0; //re index the array from the beginning
    }
    IFS0bits.T2IF = 0; //Reset Timer2 interrupt flag and Return from ISR
    TMR2 = 0x00;
}

int get_average(){
    int sum=0;
    int j=0;
    for(j=0;j<25;j++){
        sum=sum+array_values[j];
    }
    return (int)(sum/25);
}
```

In brief, we used an interrupt function using timer 2 each 0.1sec to get the sensor value, save in the array and on each 2.5sec get the average and update the value showing on the 7 segments.

Each sensor value was connected to an input and each segment on the 7 segments is connected to an output port and same for the led to turn on when the average calculated is less or equal to the threshold value initialized in the top of the code.

Hardware picture:

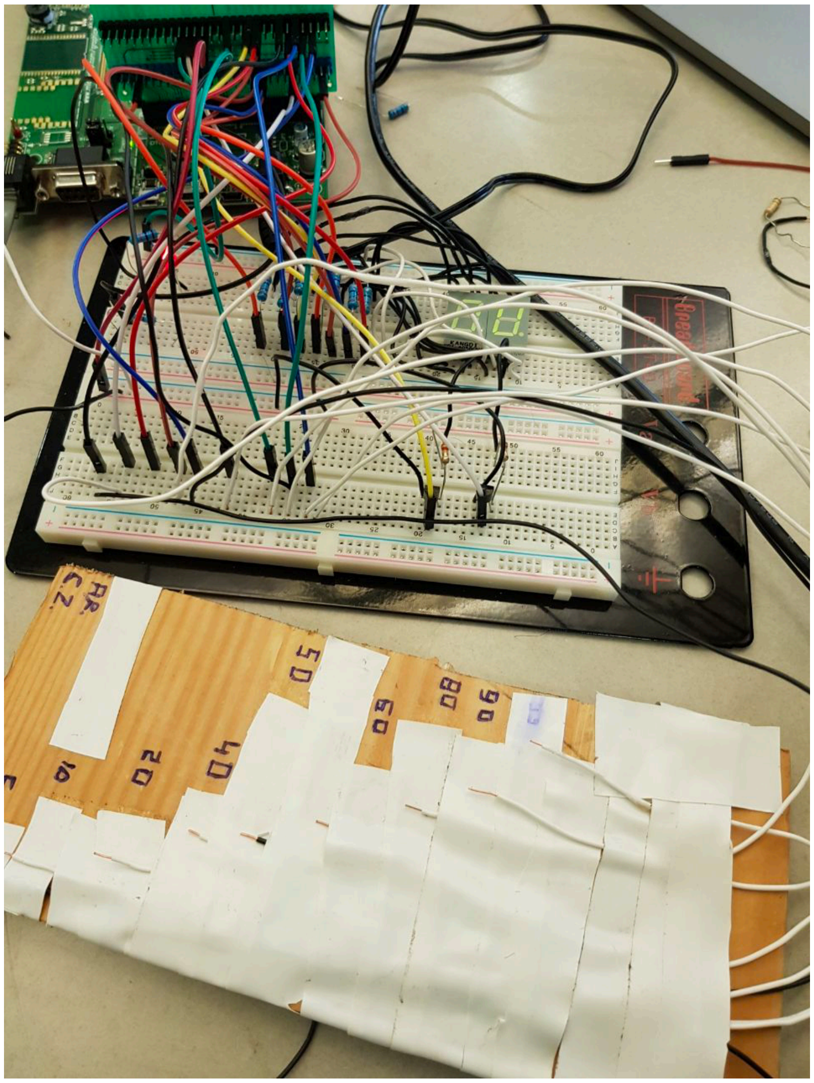

Figure 1 - Circuit implementation