```
/*
Author: Group 2 (Hang Xu, Wen Wu, Wenjun Ma)
Date complete: 13/3/2018
Filename: EE2A Experiment5 Wire-Following Sensor and Associated Signal
Processing-Improved version with side judgement
Target device: PIC18F27K40
Fuse settings:NOMCLR, NOWDT,NOPROTECT,NOCPD
Program function: To determine the direction of the vehicle by implementing the
signal processing algorithm.
*/

#include <18F27K40.h>
#device adc=8
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/***************main frequency setting***************/
#use delay(internal=64Mhz,clock_out)

/***************rs232 setting***************/
#pin_select U1TX=PIN_C0 // transmit data
#pin_select U1RX=PIN_C1 // receive data
#use rs232(uart1, baud=9600, ERRORS)

/***************spi setting***************/
#use spi(MASTER,DO=PIN_A2,MODE=0,CLK=PIN_A3,BITS=8) //set SPI

/***************pwm setting***************/
#pin_select PWM4=PIN_A0 //select PIN_A0 as output of PWM

/***************structure***************/
struct IO_Port_Definition
{
    int1 PWM;//PIN_A0(LDAC)
    int1 cs; //PIN_A1
    int1 SDO;//PIN_A2
    int1 SCK; //PIN_A3
    int unusedA:3;//PIN_A4..6
    int1 ADC;//PIN_A7
    int unusedB:8;//PIN_B0..7
    int1 ts;//PIN_C0
    int1 rc;//PIN_C1
    int unusedC:6; //PIN_C2..7
};
struct IO_Port_Definition Port;
struct IO_Port_Definition PortDirection;
#byte Port = 0xF8D
#byte PortDirection = 0xF88

/***************variables***************/
//RDA//
char CommandString[32];
int    in=0;
//Command//
char        collectdata[]="COLLECT DATA";
//ADC//
signed int8       adctable[128];
long        count=0;
//Main//
int         pause=1;//1 when main function is interrupted; 0 for resume
//Look up table//
unsigned int16 SinTable[32] =
{2048,2640,3164,3563,3796,3845,3718,3444,3072,2660,2270,1953,1748,1671,17
16,1856,2048,2240,2380,2425,2348,2143,1826,1436,1024,652,378,251,300,533,9
32,1456};//LUT for combined 1 kHz + 2 kHz signal
signed int32 Multi_1coscos;
signed int32 Multi_1cossin;
signed int32 Multi_2coscos;
signed int32 Multi_2cossin;
int Look_Up_Table_Index=0;
int i;// indext

/***************RDA_interrupt***************/
#INT_RDA
void rda_isr(void)
{
    pause=1;
    CommandString[0]=0;//reset c
    in=0;
    do
    {
        CommandString[in]=getc();
        putc(CommandString[in]);
        if(CommandString[in]==127) in=in-2;//backspace check
        in=in+1;
    }
    while((in<31)&&(CommandString[in-1]!=13));
    CommandString[in-1]=0;
    putc(13);//enter
    putc(10);//back to first column
    //ERROR JUDGEMENT//
    if(STRICMP(CommandString,collectdata)!=0)    puts("ERROR");
}

/***************Timer2_interrupt***************/
#int_timer2
void Timer2_Service_Routine(void)
{
    Port.cs = 0b0;//SPI Chip select signal low
    spi_xfer((SinTable[Look_Up_Table_Index])>>8); //High byte(+4096(2^12) for
SHDN=1)
    spi_xfer((SinTable[Look_Up_Table_Index])&0x00FF);// Low byte
    Port.cs = 0b1;//SPI Chip select signal high
    Look_Up_Table_Index=++Look_Up_Table_Index % 32;//if already count to 32,
then reset to 0
}

/***************ADC_interrupt***************/
signed int8 adcmin;
signed int8 adcmax;
signed int16 sum;
#INT_AD
void adc_isr(void)
{
    if(count<128)
    {
        adctable[count] = read_adc(ADC_READ_ONLY)-128; //remove offeset
        sum = sum + (signed int16)adctable[count];// extend word length to meet
the need for sum operation
        // find out the maximun and minimum value of sensed data
        if(count ==0)
        {
            adcmin = adctable[count];
            adcmax = adctable[count];
        }
        else
        {
            if (adcmin>adctable[count]) adcmin = adctable[count];
            if (adcmax<adctable[count]) adcmax = adctable[count];
        }
     }
        count++;
}

/***************main_function***************/
void main()
{
    //Port Setting//
    int BWPU;//weak pull up PIN_B
    #byte BWPU = 0x0F18;
    BWPU = 0b11111111;
        //A//
    PortDirection.PWM=0b0;
    PortDirection.ADC=0b1;
        //C//
    PortDirection.ts=0b0;
    PortDirection.rc=0b1;
    PortDirection.cs=0b0;
    PortDirection.SDO=0b0;
    PortDirection.SCK=0b0;
    //RDA//
```

```
    enable_interrupts(INT_RDA);
    //TIMER2//
    setup_timer_2(T2_CLK_INTERNAL|T2_DIV_BY_2,249,1);
    enable_interrupts(INT_TIMER2);// Timer 2 interrupt enabled
    //PWM//
    setup_ccp2(CCP_PWM|CCP_USE_TIMER1_AND_TIMER2);
    setup_pwm4(PWM_ENABLED|PWM_ACTIVE_LOW|PWM_TIMER2);
    set_pwm4_duty(64);//active low for 1us
    //ADC//
    setup_adc_ports(sAN7,VSS_FVR);
    setup_adc(ADC_LEGACY_MODE|ADC_CLOCK_DIV_128);
    setup_vref(VREF_ON|VREF_ADC_1v024);
    set_adc_channel(7);
    set_adc_trigger(ADC_TRIGGER_TIMER2);
    enable_interrupts(INT_AD);
    //GLOBAL//
    enable_interrupts(GLOBAL);

    for(i=0;i<32;i++)
    {
        SinTable[i]=SinTable[i]+4096;
    }

    while(1)
    {
/**************ADC CONTROL*****************/
        if (STRICMP(CommandString,collectdata)==0)//if collect data command is
received
        {
            puts("OK");
            pause=0;
            count=0;
            sum=0;
            while(pause==0)
            {
                if(count==128)
                {
                    long ii;//index
                    printf("[");
                    for(ii=0;ii<128;ii++)
                    {
                        if(ii<128) printf("%d ",adctable[ii]);
                    }
                    printf("];");
                    putc(13);putc(10);
                    float avg = ((float)sum)/128;//average value
                    printf("%d ,%d ,%f",adcmin,adcmax,avg);
                    putc(13);putc(10);
                    // judge the direction
                    if ((avg - adcmin)>(adcmax - avg))printf("left");
                    if ((avg - adcmin)<(adcmax - avg))printf("right");
                    if ((avg - adcmin)==(adcmax - avg))printf("middle");
                    putc(13); putc(10);
                    pause=1;
                }
              }
        CommandString[0]=0;//reset c
        }
    }
}
```