**// MAIN.C**

//include Files

#include <hidef.h>

#include "derivative.h"

#include "init.h"

#include "global.h"

#include "sensorAnalyser.h"

#include "TRIGGER\_CAR.h"

void main(void)

{

bool start\_car;

Bool sensor\_calibrated = FALSE;

initialisation(); //Initialisation

// Wait until the calibration is done and Exit when the sensor is calibrated

while(sensor\_calibrated == FALSE){

sensor\_calibrated = Calibrate\_Sensors(); //Sensor Calibration

}

//Enable Interrupts

start\_car=sensor\_calibrated;

for(;;){

TRIGGER\_CAR(start\_car);

}

}

**// file: init.c**

// Initializes the hardware and software and calibrate the sensors

#include "mc9s12xdt512.h"

#include "init.h"

#include "global.h"

void initialisation(void){

A\_D\_init(); // Sensor Initialisation

Servo\_Motor\_init(); //Servo Motor Initialisation

DC\_Motor\_init(); // DC Motor Initialisation

Switch\_n\_LED\_init();

}

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

// Name: A\_D\_init

// Function: Initialising the Sensors for Analog to Digital conversion on ATD `Channel 0

// Input: NULL

// Output: NULL

void A\_D\_init(void){

ATD0CTL1=EXT\_TRIG\_OFF; // no external trigger

ATD0CTL2\_ADPU=ENABLE; // ATD Enable

ATD0CTL2\_ETRIGE=DISABLE; // no external trigger

ATD0CTL3\_S8C=ENABLE; // 8 adc channels sequence

ATD0CTL4\_SRES8=ENABLE; // 8-bit Resolution

ATD0CTL5=MUL\_SEQ; // multisequence

}

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

// Name: Servo\_Motor\_init

// Function: Initialising the Servo motor, defining the clock period ()

// Input: NULL

// Output: NULL

void Servo\_Motor\_init(void) {

PWMCTL\_CON23=ENABLE;

PWME\_PWME2=ENABLE; //PWM channel 2 Enable

PWME\_PWME3=ENABLE; //PWM channel 3 Enable

PWMPOL\_PPOL2=HIGH; //PWM pulse High at begining of Period

PWMPOL\_PPOL3=HIGH; //PWM pulse High at begining of Period

PWMCLK\_PCLK2=ENABLE; // clock SA as clock source for PWM

PWMCLK\_PCLK3=ENABLE; // clock SB as clock source for PWM

PWMPRCLK =RESET; //clock A = 2MHz clockB = 2MHz

PWMSCLA =SA\_SERVO; //clock SA = clock A / (2 \* 100) = 10KHz //Issue is here

PWMSCLB =SB\_SERVO; //clock SB = clock B / (2 \* 100) = 10KHz

PWMPER2 = PER\_SERVO1 ; // PWM Period 20ms 50Hz

PWMPER3 = PER\_SERVO2 ; // PWM Period 20ms 50Hz

}

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

// Name: DC\_Motor\_init

// Function: Initialising the DC motor, defining the clock period ()

// Input: NULL

// Output: NULL

void DC\_Motor\_init(void) {

DDRE\_DDRE2=OUTPUT; // Port E pin 2 & 3 set to output

DDRE\_DDRE3=OUTPUT;

DDRP\_DDRP0=OUTPUT;

DDRP\_DDRP1=OUTPUT;

PORTE\_PE2=DISABLE;

PORTE\_PE3=ENABLE;

PWMPOL\_PPOL0=HIGH; //PWM pulse High at begining of Period

PWMPOL\_PPOL1=HIGH; //PWM pulse High at begining of Period

PWMCLK\_PCLK0=ENABLE; // clock SA as clock source for PWM

PWMCLK\_PCLK1=ENABLE; // clock SB as clock source for PWM

PWMPRCLK =RESET; //clock A = 2MHz clockB = 2MHz

PWMSCLA =SA\_SERVO; //clock SA = clock A / (2 \* 5) = 200KHz

PWMSCLB =SB\_SERVO; //clock SB = clock B / (2 \* 5) = 200KHz

PWMPER0 = PER\_DC1; // PWM Period 1KHz

PWMPER1 = PER\_DC2; // PWM Period 1KHz

}

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

// Name: Switch\_n\_LED\_init

// Function: Initialise the Switch SW2 and SW3 and LEDs.

// Enable Port P as i/p for switch and Port T as o/p for LEDs

// Input: NULL

// Output: NULL

void Switch\_n\_LED\_init(void) {

DDRP\_DDRP5=INPUT; //Port P Pin 5 & 7 set to input

DDRP\_DDRP7=INPUT;

PERP\_PERP5=ENABLE; //Port P Pin 5 & 7 Pullup Enable

PERP\_PERP7=ENABLE;

DDRT=PORT\_T\_AS\_OUTPUT; // Setting Port T pin 4,5,6,7 as output

PTT=ALLLED\_OFF; //Turning Off all LEDs

}

**// file: init.h**

#ifndef \_INIT\_H

#define \_INIT\_H

/\* Local Function Prototypes \*/

void initialisation(void);

void A\_D\_init(void);

void DC\_Motor\_init(void);

void Servo\_Motor\_init(void);

void Switch\_n\_LED\_init(void);

#endif

**// file: sensor.c**

// Initializes the hardware and software and calibrate the sensors

#include "mc9s12xdt512.h"

#include "sensor.h"

#include "global.h"

//Global variables

int ATD\_arr[BYTES];

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

// Name: Switch\_ON\_LED

// Function: switch on the desired LED

// Input: LED number

// Output: NULL

void Switch\_ON\_LED(int LED){

switch(LED) {

case LED1: PTT\_PTT4=ON; //LED 1 On

break;

case LED2: PTT\_PTT5=ON; //LED 2 On

break;

case LED3: PTT\_PTT6=ON; //LED 3 On

break;

case LED4: PTT\_PTT7=ON; //LED 4 On

break;

}

}

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

// Name: Switch\_ON\_LED

// Function: switch on the desired LED

// Input: LED number

// Output: NULL

bool Switch\_Pressed(int SW){

int result = FALSE;

switch(SW){

case SW2: if(PTP\_PTP5 == ON){ //Check for if SW2 is pressed

result = TRUE;

}

break;

case SW3: if(PTP\_PTP7 == ON){ //Check for if SW3 is pressed

result = TRUE;

}

break;

default: break;

}

return result;

}

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

// Name: Uncomp\_Sensor\_Value

// Function: Read the sensors value from the ATD pins

// Input: NULL

// Output: Pointer to the ATD sensor values

void Uncomp\_Sensor\_Value(int \*ATD\_ptr) {

ATD0CTL5=SENSOR\_RESET; // Reset the sensor values

Delay(ENABLE);

\*ATD\_ptr = ATD0DR0H; //Reads the current value available at the pin: ATD0

\*(ATD\_ptr+1)= ATD0DR1H;

\*(ATD\_ptr+2)= ATD0DR2H;

\*(ATD\_ptr+3)= ATD0DR3H;

\*(ATD\_ptr+4)= ATD0DR4H;

\*(ATD\_ptr+5)= ATD0DR5H;

\*(ATD\_ptr+6)= ATD0DR6H;

\*(ATD\_ptr+7)= ATD0DR7H;

}

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

// Name: Delay

// Function:

// Input: Delay in milliseconds

// Output: NULL

void Delay(unsigned char time) {

unsigned int count1,count2;

for(count2=1;count2<=time;count2++){

//Delay

for(count1=0;count1<=1000;count1++){

}

}

}

**// file: sensor.h**

#ifndef \_SENSOR\_H

#define \_SENSOR\_H

#include "global.h"

/\* Local Function Prototypes \*/

void Uncomp\_Sensor\_Value(int \*ATD\_ptr);

void Delay(unsigned char a);

void Switch\_ON\_LED(int LED);

bool Switch\_Pressed(int SW);

#endif

**// TRIGGER\_CAR.c**

#include "TRIGGER\_CAR.h"

#include "sensorAnalyser.h"

#include "mc9s12xdt512.h"

#include "global.h"

#include "pid.h"

int ATD\_cmp\_arr[BYTES];

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

// Name: TRIGGER\_CAR

// Function: Runs the car after completion of initialization and calibration

// Input: start\_IRQ

// Output: Error

void TRIGGER\_CAR(bool start\_car){

if(start\_car== TRUE){

run();

}

}

**// TRIGGER\_CAR.h**

#ifndef \_TRIGGERCAR\_H

#define \_TRIGGERCAR\_H

#include "global.h"

/\* Local Function Prototypes \*/

void TRIGGER\_CAR(bool start\_car);

#endif

**// file: sensorAnalyser.c**

// Performs the task of calculating compensated ratio and error. Then it provides the desred

// sensor values to Main.c

#include "sensorAnalyser.h"

#include "sensor.h"

#include "global.h"

//Global Variables

int condition\_var = LED1;

int index[BYTES];

int black[BYTES], white[BYTES];

int comp\_ratio[BYTES];

int ATD\_comp\_arr[BYTES];

int clb\_sns\_black = FALSE;

int clb\_sns\_white = FALSE;

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

// Name: Read\_Sensor\_Values

// Function: Read the sensors value from the ATD pins and guides the whole process to get error

// Input: NULL

// Output: Error

int Read\_Sensor\_Values(void) {

int error= NONE;

int ATD\_comp\_ptr[BYTES];

Comp\_Sensor\_Value(ATD\_comp\_ptr);

Bubble\_sort();

error = Position\_center();

return error;

}

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

// Name: Calibrate\_Sensors

// Function: Calibrating the Sensors against the black and white region of the track before racing.

// to compensate for the sensitivity difference of the sensors. A compensation ratio can be

// calculated from these 'black' and 'white' values.It is scalling factor to be applied to future

// readings so that '0' would be obtained when each sensor is on white region and 255(for 8-bit ATD)

// when each sensor is on black region.

//

// Compensation Ratio = 256 / (Black value - White Value)

// if Reading > 'Black value", then Reading = Black

// if Reading < 'White Value', then Reading = White

// Compensated Reading = (Reading - White) \* Compensation Ratio

//

// Input: NULL

// Output: NULL

Bool Calibrate\_Sensors(void) {

int count, clb\_result;

int finish\_process = FALSE;

int ATD\_arr[BYTES];

switch(condition\_var){

case LED1: Switch\_ON\_LED(LED1);

++condition\_var; //Forward the process to next case

break;

case LED2: if(Switch\_Pressed(SW2)){ //Check for if SW2 is pressed

Uncomp\_Sensor\_Value(ATD\_arr); //Read Sensor values for black

count = LED1;

for (;count<LED8;count++) {

black[count]=ATD\_arr[count]; //Get the black values

}

clb\_sns\_black = TRUE;

Switch\_ON\_LED(LED2);

++condition\_var; //Forward the process to next case

}

break;

case LED3: if(Switch\_Pressed(SW3)){ //Check for if SW3 is pressed

Uncomp\_Sensor\_Value(ATD\_arr); //Read Sensor values for white

count = LED1;

for (;count<LED8;count++) {

white[count]=ATD\_arr[count]; //get the White Values

}

clb\_sns\_white = TRUE;

Switch\_ON\_LED(LED3);

++condition\_var; //Forward the process to next case

}

break;

case LED4: if(Switch\_Pressed(SW2)){ //Check for if SW2 is pressed

finish\_process = TRUE;

Switch\_ON\_LED(LED4);

}

break;

}

clb\_result = clb\_sns\_black & clb\_sns\_white;

if(clb\_result == TRUE){

for( count=LED1;count<=LED7;count++)

{

comp\_ratio[count]=(MAX\_CAL\*Scaling\_factor)/(black[count]-white[count]); //Compensation Ratio of black and white values

}

clb\_sns\_black = FALSE; //Only to store the compensation value during actual calibration process and

clb\_sns\_white = FALSE; //avoid over writing of the compensation ratio value

}

clb\_result = finish\_process;

if(clb\_result == TRUE){

return clb\_result;

}

}

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

// Name: Comp\_Sensor\_Value

// Function: Read the sensors value from the ATD pins and perform compensation

// Input: NULL

// Output: Pointer to the ATD sensor values

void Comp\_Sensor\_Value(int \*ATD\_comp\_ptr) {

int ATD\_arr[BYTES];

int count = NONE;

Uncomp\_Sensor\_Value(ATD\_arr);

\*ATD\_comp\_ptr = ((ATD\_arr[LED1]-white[LED1])\*comp\_ratio[LED1])/Scaling\_factor; //Do compensation for current value available at the pin: ATD0

\*(ATD\_comp\_ptr+1)= ((ATD\_arr[LED2]-white[LED2])\*comp\_ratio[LED2])/Scaling\_factor; //Do compensation for current value available at the pin: ATD1

\*(ATD\_comp\_ptr+2)= ((ATD\_arr[LED3]-white[LED3])\*comp\_ratio[LED3])/Scaling\_factor; //Do compensation for current value available at the pin: ATD2

\*(ATD\_comp\_ptr+3)= ((ATD\_arr[LED4]-white[LED4])\*comp\_ratio[LED4])/Scaling\_factor; //Do compensation for current value available at the pin: ATD3

\*(ATD\_comp\_ptr+4)= ((ATD\_arr[LED5]-white[LED5])\*comp\_ratio[LED5])/Scaling\_factor; //Do compensation for current value available at the pin: ATD4

\*(ATD\_comp\_ptr+5)= ((ATD\_arr[LED6]-white[LED6])\*comp\_ratio[LED6])/Scaling\_factor; //Do compensation for current value available at the pin: ATD5

\*(ATD\_comp\_ptr+6)= ((ATD\_arr[LED7]-white[LED7])\*comp\_ratio[LED7])/Scaling\_factor; //Do compensation for current value available at the pin: ATD6

for(count=LED1;count<TOTAL\_LEDS;count++){

ATD\_comp\_arr[count] = ATD\_comp\_ptr[count];

}

}

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

// Name: Bubble\_sort

// Function: Bubble sort the compensated values

// Input: NULL

// Output: NULL

void Bubble\_sort(void){

//Variable Declaration

int count, count1, count2,temp;

int BIN\_SORT[BYTES];

for (count=LED1;count<LED7;count++) {

BIN\_SORT[count]=ATD\_comp\_arr[count];

}

for (count1=LED1;count1<LED8;count1++) {

for (count2=count1+LED1;count2<LED8;count2++ ) {

if ( BIN\_SORT[count1]<BIN\_SORT[count2]) { // Compare the reading of sensors

temp=BIN\_SORT[count1];

BIN\_SORT[count1]=BIN\_SORT[count2];

BIN\_SORT[count2]=temp; // Bubble sort the sensor readings

}

}

}

for (count=LED1;count<LED7;count++) // Initialize the outer loop

{

if (BIN\_SORT[LED1]==ATD\_comp\_arr[count])

index[LED1]=count;

if (BIN\_SORT[LED2]==ATD\_comp\_arr[count])

index[LED2]=count;

if (BIN\_SORT[LED3]==ATD\_comp\_arr[count] )

index[LED3]=count;

}

for (count1=LED1;count1<LED4;count1++) {

for (count2=count1+LED1;count2<LED4;count2++ ) {

if ( index[count1]<index[count2]) { // Compare the reading of sensors

temp=index[count1];

index[count1]=index[count2];

index[count2]=temp; // Bubble sort the sensor readings with indeces

}

}

}

}

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

// Name: Postion\_center

// Function: To find the final position of center of track according to sensors

// Input: NULL

// Output: Error (integer)

int Position\_center(void) {

int pos\_data[BYTES],pos\_center,count,error;

pos\_center= index[LED2]\*ASST\_SENSORS\*MAX\_CAL +

(ATD\_comp\_arr[index[LED1]]- ATD\_comp\_arr[index[LED3]]);

count = LED1;

for (;count<LED8;count++) {

pos\_data[count]= MAX\_CAL\_SNS\_VALUE \* count;

}

error = 1536 - pos\_center ;

return error;

}

**// file: sensorAnalyser.h**

#ifndef \_SENSORANALYSER\_H

#define \_SENSORANALYSER\_H

#include "global.h"

/\* Local Function Prototypes \*/

int Read\_Sensor\_Values(void);

void Comp\_Sensor\_Value(int \*ATD\_comp\_ptr);

void Bubble\_sort(void);

int Position\_center(void);

bool Calibrate\_Sensors(void);

#endif

**// pid.c**

#include "pid.h"

#include "sensorAnalyser.h"

#include "mc9s12xdt512.h"

#include "motor.h"

#include "global.h"

float Kp,Ki;

int error,p\_ctrl,i\_ctrl;

int output;

int error1;

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

// Name: PID

// Function: Performs the desired PI control to run motor

// Input: None

// Output: None

int PID(void){

Kp=0.11;

Ki=0.005;

error = Read\_Sensor\_Values();

p\_ctrl=(int)(error\*Kp);

i\_ctrl=(int)((error + error1) \* Ki) ;

error1=error+error1;

if (error1>32000){

error1=32000 ;

}

if (error1<-32000){

error1=-32000 ;

}

output= 1500 - i\_ctrl - p\_ctrl;

if (output>1800) {

output = 1800;

}

if (output<1200){

output = 1200;

}

return output;

}

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

// Name: RUN

// Function: Run the motor according to PI control

// Input: None

// Output: None

void run(void){

Servo\_Motor();

DC\_Motor();

}

**// pid.h**

#ifndef \_INIT\_H

#define \_INIT\_H

/\* Local Function Prototypes \*/

int PID(void);

void run(void);

#endif

**// file: motor.c**

// Initializes the hardware and software and calibrate the sensors

#include "mc9s12xdt512.h"

#include "motor.h"

#include "global.h"

#include "pid.h"

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

// Name: Servo\_Motor

// Function: Align the Servo motor

// Input: NULL

// Output: NULL

void Servo\_Motor(void) {

int dty2,dty3,servo;

servo= PID();

dty2= servo>>8;

servo= servo & 0x00FF;

dty3=servo;

PWMDTY2 = dty2; // Pulse Width 1.5ms: Center Positionign

PWMDTY3 = dty3; //1.415ms

}

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

// Name: DC\_Motor

// Function: Run the DC motor straight

// Input: NULL

// Output: NULL

void DC\_Motor(void) {

PORTE\_PE3=ENABLE;

PWMDTY0 = DTY0\_FORW ; // Setting Duty Cycle-0 to 30 fo DC Motor

PWMDTY1 = DTY1\_FORW ; // Setting Duty Cycle-1 to 0

PWME\_PWME0=ENABLE; //PWM channel 0 Enable

PWME\_PWME1=ENABLE; //PWM channel 1 Enable

}

**// file: motor.h**

#ifndef \_MOTOR\_H

#define \_MOTOR\_H

/\* Local Function Prototypes \*/

void Servo\_Motor(void);

void DC\_Motor(void);

#endif

**// file: global.h**

#ifndef \_GLOBAL\_H

#define \_GLOBAL\_H

#include <mc9s12xdt512.h>

#include "stdtypes.h"

#define bool Bool

#define SINT8 sint

//General definition

#define ON 0

#define OUTPUT 1

#define ENABLE 1

#define OFF 1

#define INPUT 0

#define DISABLE 0

#define HIGH 1

#define LOW 0

#define RESET 0x00

#define NONE 0

//hardware based defined values

#define SA\_DC 5

#define SB\_DC 5

#define PER\_DC1 200

#define PER\_DC2 200

#define PER\_SERVO1 0x4E

#define PER\_SERVO2 0x20

#define EXT\_TRIG\_OFF 0x87

#define EXT\_TRG\_ON 0x03

#define MUL\_SEQ 0x10

#define SA\_SERVO 1

#define SB\_SERVO 1

#define BYTES 8

#define DTY2\_CENTER 5

#define DTY3\_CENTER 135

#define DTY0\_FORW 60

#define DTY1\_FORW 0

#define PORT\_T\_AS\_OUTPUT 0xF0

#define LED1\_ON 0xEF

#define LED2\_ON 0xDF

#define LED3\_ON 0xBF

#define ALLLED\_OFF 0xFF

//Sensor related

#define LED1 0

#define LED2 1

#define LED3 2

#define LED4 3

#define LED5 4

#define LED6 5

#define LED7 6

#define LED8 7

#define TOTAL\_LEDS 7

#define SW2 2

#define SW3 3

#define VALUE\_DIFF 50

#define SENSOR\_RESET 0x10

#define ASST\_SENSORS 2

#define MAX\_CAL 256

#define MAX\_CAL\_SNS\_VALUE 512

#define Scaling\_factor 100

#define HEX 16

#endif