/\* ========================================

\*

\* Copyright YOUR COMPANY, THE YEAR

\* All Rights Reserved

\* UNPUBLISHED, LICENSED SOFTWARE.

\*

\* CONFIDENTIAL AND PROPRIETARY INFORMATION

\* WHICH IS THE PROPERTY OF your company.

\*

\* ========================================

\*/

/\*TRC3500 Ultrasonic Distance Meter Project Source Code.\*/

#include "project.h"

#define MAX\_READINGS 10 //Maximum number of readings per recording.

#define USED\_EEPROM\_SECTOR 1u //Indicates the sector that the program will read the EEPROM from.

#define GROUP\_BASE ((USED\_EEPROM\_SECTOR \* CYDEV\_EEPROM\_SECTOR\_SIZE) + 0x00) //Read and Write address of the Group number information on the EEPROM

#define UNIT\_BASE ((USED\_EEPROM\_SECTOR \* CYDEV\_EEPROM\_SECTOR\_SIZE) + 0x01) //Read and Write address of the Unit information on the EEPROM.

//N.B. Disambiguation of the Term Reading and Recording.

//A Reading in the context of this code is a single measurement of the distance of the object.

//A Recording is the average of all the readings that were obtained.

//Global Variables

int start=0; //Indicates the start of a single recording.

int ready=0; //Indicates the start of a single reading.

int program=0; //Indicates whether device is in program mode or not.

float final\_value=0; //The distance obtained after one recording.

int unit; //Used to store and change whether the device is measuring in centimeters or inches.

//N.B. 1 = cm 0 = inches

int unit2=1; //Used to store and change whether the device is measuring in centimeters or millimeters.

//N.B 1 = cm 0 = mm

//Functions Prototypes

void transmitter();

void reset();

void sortMeasurements();

void detectOutliers();

void boot();

//Timer Interrupt Handler

uint16 count=0; //Stores the value of the count after a reading.

float measurements[MAX\_READINGS]; //The array that stores all the readings for the current recording.

int readings=0; //The number of readings taken for the current recording.

CY\_ISR(Timer\_Handler){

Timer\_ReadStatusRegister(); //Resets the interrupt.

if(readings<MAX\_READINGS){

//If there are more readings to be done, store the reading in the array.

count=Timer\_ReadCounter();

measurements[readings]=(65536-count)/58.0;

readings++;

ready=1;

}

else{

//Else, begin to sort and filter the measurements

sortMeasurements();

detectOutliers();

float sum = 0;

int count=0;

for (int x = 0; x < MAX\_READINGS; x++) {

sum += measurements[x];

if (measurements[x] > 0) {

count++;

}

}

final\_value = sum / count; //Value is in Cm

if(unit==0){

final\_value = final\_value\*(0.3937);

}

//The following is Calibration code that was used during the demonstration.

if(final\_value>6){

final\_value = final\_value-0.7;

}

else if(final\_value<4&&final\_value>1.5){

final\_value=final\_value+0.5;

}

int val; //This variable is set by whether the unit is in cm or mm.

//It is set so that the final\_value the decimals is preserved when displaying

if(unit2==1){

val=10;

}

if(unit2==0){

val=100;

}

//The display function only takes in integer values. Without the 'val' variable, the final\_value variable

//will truncate and the decimals will not display. So to preserve the decimals, the value is multiplied by the 'val' variable.

LED\_Display\_Write7SegNumberDec(final\_value\*val,0,4,LED\_Display\_RIGHT\_ALIGN);

if(unit2==1){

LED\_Display\_PutDecimalPoint(1,2);

}

CyDelay(2000);

LED\_Display\_ClearDisplayAll();

start=0;

final\_value=0;

readings=0;

}

reset();

}

int main(void)

{

CyGlobalIntEnable;

//Initialization of Components

Comp\_Start();

PGA\_Start();

EEPROM\_Start();

LED\_Display\_Start();

//Initialization of Variables.

int sleep\_count=0; //Used to have the decimal point blink during sleep mode.

//The following are setting variables that are used to ensure that certain functions are executed fully

int set\_sw2=0; //Used to debounce SW2

int set\_hard=0; //Used to set and maintain hard reset.

int set\_program=0; //Used to set and maintain Program mode

int set\_sw3=0; //Used to debounce SW3

//Ensuring Group name is recorded. Will only run when PSoC is programmed. Will not run when powered by the PSoC.

if(EEPROM\_ReadByte(GROUP\_BASE)==255){

EEPROM\_WriteByte(4,GROUP\_BASE);

}

uint8 group=EEPROM\_ReadByte(GROUP\_BASE);

//Timer Interrupt Service Request.

timer\_isr\_StartEx(Timer\_Handler);

//Ensuring reading isn't taken prematurely in the beginning.

Timer\_Stop();

Transmitter\_Stop();

//Start Up

boot();

for(;;)

{

//Recording Code.

//The following code is used to begin and repeat the readings for one recording

if(!SW1\_Read()&&start==0&&program==0){

CyDelay(100);

if(!SW1\_Read()&&start==0){

//This block represents the starting of the recording procedure.

start=1;

ready=1;

LED\_Display\_ClearDisplayAll();

}

}

if(start&&ready){

//This block is used to reset the required components to begin another reading.

CyDelay(100);

reset();

transmitter();

Timer\_Start();

ready=0;

}

//Sleep Mode Code

//When no measuring operation or if program mode is not enabled, then the device goes to sleep mode.

if(!start&&!ready){

if(sleep\_count<1000){

//Having the code wait only for 1 millisecond and increment it until it reaches a second allows

//the measuring code to run at any given point.

sleep\_count++;

LED\_Display\_PutDecimalPoint(0,3);

CyDelay(1);

}

else if(sleep\_count>=1000&&sleep\_count<2000){

sleep\_count++;

LED\_Display\_PutDecimalPoint(1,3);

CyDelay(1);

}

else{

sleep\_count=0;

}

}

//Code for Switching unit (cm and inches)

if(!SW2\_Read()&&set\_sw2==0&&program==0){

CyDelay(100);

if(!SW2\_Read()&&set\_sw2==0){

set\_sw2=1;

if(unit==0){

unit=1;

EEPROM\_WriteByte(1,UNIT\_BASE);

}

else{

unit=0;

EEPROM\_WriteByte(0,UNIT\_BASE);

}

}

}

if(SW2\_Read()){

set\_sw2=0;

}

//Code for Switching unit (cm and mm)

if(!SW3\_Read()&&set\_sw3==0){

CyDelay(100);

if(!SW3\_Read()&&set\_sw3==0){

set\_sw3=1;

if(unit2==0){

unit2=1;

}

else{

unit2=0;

}

}

}

if(SW3\_Read()){

set\_sw3=0;

}

//Code for Program mode.

//To enable program mode, SW3 and SW4 must be both pushed.

if(!SW4\_Read()&&!SW3\_Read()&&set\_program==0){

CyDelay(1000);

if(!SW4\_Read()&&!SW3\_Read()&&set\_program==0){

program++;

}

}

//N.B. program=1 :: program mode enabled and setting group number.

// program=2 :: program mode enabled and setting initial unit.

// program=0 :: terminate program mode.

if(program){

//This block controls the flow and display of the program mode.

if(program==1){

LED\_Display\_Write7SegNumberDec(group,0,4,LED\_Display\_RIGHT\_ALIGN);

CyDelay(1000);

LED\_Display\_ClearDisplayAll();

CyDelay(500);

}

else if(program==2){

if(unit){

LED\_Display\_WriteString7Seg(" C",0);

}

else{

LED\_Display\_WriteString7Seg(" I",0);

}

CyDelay(1000);

LED\_Display\_ClearDisplayAll();

CyDelay(500);

}

}

if((program==1||program==2)&&!SW3\_Read()){

CyDelay(100);

//detects push button 3 and increments group number or unit.

if((program==1||program==2)&&!SW3\_Read()){

if(program==1){

group++;

EEPROM\_WriteByte(group,GROUP\_BASE);

}

else{

if(unit){

unit=0;

}

else{

unit=1;

}

}

EEPROM\_WriteByte(unit,UNIT\_BASE);

}

}

if((program==1||program==2)&&!SW4\_Read()){

CyDelay(100);

//detects push button 4 and decrements group number or unit.

if((program==1||program==2)&&!SW4\_Read()){

if(program==1){

group--;

EEPROM\_WriteByte(group,GROUP\_BASE);

}

else{

if(unit){

unit=0;

}

else{

unit=1;

}

}

}

}

if((program==1||program==2)&&!SW2\_Read()&&set\_sw2==0){

CyDelay(100);

//detects push button 2 which changes the flow of program mode or terminate it.

if((program==1||program==2)&&!SW2\_Read()&&set\_sw2==0){

if(program){

EEPROM\_WriteByte(group,GROUP\_BASE);

program++;

}

else if(program==2){

EEPROM\_WriteByte(unit,UNIT\_BASE);

program=0;

}

}

}

LED\_Write(unit); //Displays whether cm or inches is active.

//LED ON :: cm LED OFF :: inches.

//Hard Reset Code

//It is possible that the timer code runs out before a pulse has been detected by the Receiver.

//This maybe the case if the object isn't aligned properly or if there is no object infront of the receiver or

//if the echoed signal is too weak to produce a pulse.

//This block of code terminates the recording and returns the program to sleep mode.

if(!SW4\_Read()&&SW3\_Read()&&set\_hard==0){

CyDelay(100);

if(!SW4\_Read()&&SW3\_Read()&&set\_hard==0){

reset();

start=0;

ready=0;

Timer\_Stop();

Reset\_Write(1);

Reset\_Write(0);

final\_value=0;

for(int x=0;x<MAX\_READINGS;x++){

measurements[x]=0;

}

readings=0;

}

}

}

}

void transmitter(){

//This function is called whenever a reading is taken.

//It activates the clock signal that transmits the 40 kHz pulses.

//10 pulses are sent per reading.

Transmitter\_Start();

CyDelayUs(250);

Transmitter\_Stop();

}

void reset(){

//A Function to reset the Timer component

//This function is called after a reading has been taken or by a button that soft resets the entire process.

Timer\_Stop();

Reset\_Write(1);

Reset\_Write(0);

}

void sortMeasurements() {

//This function is used to sort the array of readings that were obtained.

//A simple bubble sort is used to ensure the values get sorted.

int set = 1;

while (set) {

set = 0;

for (int x = 0; x<MAX\_READINGS; x++) {

if (x == 0) {

continue;

}

else if (measurements[x]<measurements[x - 1]) {

float a = measurements[x];

measurements[x] = measurements[x - 1];

measurements[x - 1] = a;

set = 1;

}

}

}

}

void detectOutliers() {

//This function work with the SortMeasurements function to ensure that

//measurements that are identified as an outlier (determined according to data)

//is eliminated.

float Q1 = measurements[((MAX\_READINGS + 1) / 4)-1];

float Q3 = measurements[MAX\_READINGS - ((MAX\_READINGS + 1) / 4)];

float IQR = Q3 - Q1;

float Range\_Min = Q1 - (1.5\*IQR);

float Range\_Max = Q3 + (1.5\*IQR);

for (int x = 0; x < MAX\_READINGS; x++) {

if (measurements[x] < Range\_Min||measurements[x]>Range\_Max) {

measurements[x] = 0;

}

}

}

void boot(){

//This function runs only once. It is called when the PSOC is powered up.

LED\_Display\_WriteString7Seg(" 8",0);

CyDelay(1000);

LED\_Display\_WriteString7Seg(" 8 ",0);

CyDelay(1000);

LED\_Display\_WriteString7Seg(" 8 ",0);

CyDelay(1000);

LED\_Display\_WriteString7Seg("8 ",0);

CyDelay(1000);

int Group\_init=EEPROM\_ReadByte(GROUP\_BASE);

LED\_Display\_Write7SegNumberDec(Group\_init,0,4,LED\_Display\_RIGHT\_ALIGN);

CyDelay(1000);

unit=EEPROM\_ReadByte(UNIT\_BASE);

if(unit){

LED\_Display\_WriteString7Seg(" C",0);

}

else{

LED\_Display\_WriteString7Seg(" I",0);

}

CyDelay(1000);

LED\_Display\_ClearDisplayAll();

}

/\* [] END OF FILE \*/