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

\*

\* Copyright YOUR COMPANY, THE YEAR

\* All Rights Reserved

\* UNPUBLISHED, LICENSED SOFTWARE.

\*

\* CONFIDENTIAL AND PROPRIETARY INFORMATION

\* WHICH IS THE PROPERTY OF your company.

\*

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

\*/

#include "project.h"

#define SLEEP\_MAX 43

#define MAX\_READ 15

#define MAX\_CALC 10

#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.

#define MAX\_BRIGHT 1000

#define MIN\_BRIGHT 100

#define FAULT\_VALUE 200

#define FAULT\_LIMIT 3

int mode=0;

//0 : Boot mode

//1 : Sleep mode

//2 : Calc mode

//3 : Read mode

//4 : Disp mode

//5 : Program mode

//6 : return to sleep mode

int time\_on=MIN\_BRIGHT;

int time\_off=1000-MIN\_BRIGHT;

double readings[MAX\_READ];

double calc[MAX\_CALC];

void loadRead(double value);

double bpm;

double final;

int calc\_count=0;

int read\_count=0;

int disp\_count=0;

int output\_count=0;

CY\_ISR(Timer\_Handler){

Timer\_ReadStatusRegister();

//Read Timer to get the amount of counts that has passed

int32 count=Timer\_ReadCounter();

//The time is the difference between the counts divided by the clock frequency.

double time = (Timer\_ReadPeriod()-count)/1e6;

double bpm\_measured = (1/time)\*60;

bpm = bpm\_measured;

if(calc\_count<MAX\_CALC&&mode==2){

calc[calc\_count]=bpm;

calc\_count++;

}

else if(mode==2){

int faults=0;

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

if(calc[x]>FAULT\_VALUE||calc[x]<70){

faults++;

}

}

if(faults>FAULT\_LIMIT){

//The brightness needs to increase

if(time\_on<MAX\_BRIGHT){

time\_on+=200;

time\_off-=200;

}

else{

//Unable to adjust brightness, return to sleep mode.

time\_on=MIN\_BRIGHT;

time\_off=1000-MIN\_BRIGHT;

LED\_Display\_ClearDisplayAll();

mode=1;

}

calc\_count=0;

}

else{

mode=3;

calc\_count=0;

}

}

if(mode==3){

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

if(read\_count<MAX\_READ){

readings[read\_count]=bpm;

read\_count++;

}

else{

mode=4;

read\_count=0;

}

}

if(mode==4){

loadRead(bpm);

int count=0;

int sum=0;

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

if(readings[x]<200||readings[x]>60){

count++;

sum+=readings[x];

}

}

final=sum/count;

}

if(final>200&&mode==4){

//Return to Sleep mode

if(disp\_count<3){

disp\_count++;

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

}

else{

disp\_count=0;

mode=6;

}

}

else if(mode==4){

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

disp\_count=0;

}

Reset\_Write(1);

CyDelay(10);

Reset\_Write(0);

}

int main(void)

{

CyGlobalIntEnable;

Opamp\_Start();

Comp\_Start();

LED\_Display\_Start();

Timer\_Start();

ISR\_Timer\_StartEx(Timer\_Handler);

EEPROM\_Start();

//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);

int sleep\_count=0;

int boot\_count=0;

int boot\_section=0;

int program\_count=0;

int program\_check=0;

int return\_count=0;

int tran=0;

for(;;)

{

switch(mode){

case 0:

if(boot\_count<1000){

CyDelayUs(500);

boot\_count++;

}

else{

boot\_count=0;

boot\_section++;

}

switch(boot\_section){

case 0:

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

break;

case 1:

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

break;

case 2:

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

break;

case 3:

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

break;

case 4:

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

break;

default:

LED\_Display\_ClearDisplayAll();

mode=1;

break;

}

case 1:

if(sleep\_count<500){

LED\_Write(1);

CyDelayUs(MIN\_BRIGHT);

LED\_Write(0);

CyDelayUs(1000-MIN\_BRIGHT);

if(mode){

LED\_Display\_PutDecimalPoint(1,3);

CyDelay(1);

}

else{

CyDelayUs(500);

}

sleep\_count++;

}

else if(sleep\_count>=500&&sleep\_count<1000){

LED\_Write(0);

CyDelayUs(500);

if(mode){

LED\_Display\_PutDecimalPoint(0,3);

CyDelay(1);

}

else{

CyDelayUs(500);

}

sleep\_count++;

}

else{

sleep\_count=0;

}

if(bpm>SLEEP\_MAX&&mode!=0){

mode=2;

sleep\_count=0;

}

break;

case 2:

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

LED\_Write(1);

CyDelayUs(time\_on);

LED\_Write(0);

CyDelayUs(time\_off);

break;

case 3:

case 4:

LED\_Write(1);

CyDelayUs(time\_on+100);

LED\_Write(0);

CyDelayUs(time\_off-100);

//Reloading measurements in recordings

break;

case 5:

if(program\_count<500){

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

program\_count++;

CyDelay(1);

}

else if(program\_count>=500&&program\_count<1000){

LED\_Display\_ClearDisplayAll();

program\_count++;

CyDelay(1);

}

else{

program\_count=0;

}

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

CyDelay(100);

if(SW1\_Read()&&program\_check==0){

program\_check=1;

group++;

}

}

else if(SW1\_Read()&&program\_check){

program\_check=0;

}

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

CyDelay(100);

if(SW2\_Read()&&program\_check==0){

program\_check=1;

group--;

}

}

else if(SW2\_Read()&&program\_check){

program\_check=0;

}

if(!SW3\_Read()&&program\_check==0){

CyDelay(100);

if(!SW3\_Read()&&program\_check==0){

sleep\_count=0;

mode=6;

}

}

break;

case 6:

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

if(return\_count<500){

LED\_Write(1);

CyDelayUs(MIN\_BRIGHT);

LED\_Write(0);

CyDelayUs(1000-MIN\_BRIGHT);

return\_count++;

CyDelay(1);

}

else if(return\_count>=500&&return\_count<1000){

LED\_Write(0);

return\_count++;

CyDelay(1);

}

else{

tran++;

return\_count=0;

}

if(tran>=3){

mode=1;

tran=0;

LED\_Display\_ClearDisplayAll();

}

break;

}

if(!SW1\_Read()&&!SW2\_Read()&&program\_check==0){

CyDelay(1000);

if(!SW1\_Read()&&!SW2\_Read()&&program\_check==0){

mode=5;

program\_check=1;

}

}

if(SW1\_Read()&&SW2\_Read()){

program\_check=0;

}

}

}

void loadRead(double value){

for(int x=(MAX\_READ-1);x>=0;x--){

if(x!=(MAX\_READ-1)){

readings[x+1]=readings[x];

}

}

readings[0]=value;

}

/\* [] END OF FILE \*/