/\*

\*/

#include <stdio.h>

#include <stdlib.h>

#include <mcs51reg.h>

#include <stdint.h>

#include <math.h>

#include <mcs51/8051.h>

#include <at89c51ed2.h>

#define SDA P1\_4 //Define SDA as Pin 1.4

#define SCL P1\_3 //Define SCL as Pin 1.3

// Define i2c speed

#define I2C\_SPEED\_FACTOR 1 // Low value means low i2c frequency

#define Crystal\_Value 12 // MHz

#define HalfBitDelay (500\*Crystal\_Value)/(12\*I2C\_SPEED\_FACTOR)

void delay(unsigned int a);

int getchar(void);

int putchar(int x);

void init\_i2c(void);

void start\_i2c();

void stop\_i2c();

void i2c\_NoAck();

void i2c\_ack();

unsigned char i2c\_read\_byte();

int i2c\_write\_byte(uint8\_t byte);

void i2c\_write();

void i2c\_read();

uint16\_t address\_in();

uint16\_t hex\_read(uint16\_t start\_addr);

void hex\_dump();

void eeprom\_reset();

//uint16\_t input\_num();

void delay(unsigned int a)

{

unsigned int i,limit;

limit = a/15;

for(i=0;i<limit;i++);

}

/\* Function to take input from keyboard.\*/

int getchar(void)

{

while(RI==0)

{

;

}

RI=0;

return SBUF;

}

/\* Function to print on terminal. \*/

int putchar(int x)

{

while(TI==0){

;

}

SBUF=x;

TI=0;

return x;

}

void start\_i2c(void)

{

SCL=1;

SDA=1;

delay(HalfBitDelay);

SDA=0;

delay(HalfBitDelay);

}

void init\_i2c(void)

{

// Make SDA and SCK pins input initially

SDA = 1;

SCL = 1;

}

void restart\_i2c()

{

SCL=0;

delay(HalfBitDelay/2);

SDA=1;

delay(HalfBitDelay/2);

SCL=1;

delay(HalfBitDelay/2);

SDA=0;

delay(HalfBitDelay/2);

}

void stop\_i2c(void)

{

SCL=0;

delay(HalfBitDelay/2);

SDA=0;

delay(HalfBitDelay/2);

SCL=1;

delay(HalfBitDelay/2);

SDA=1;

delay(HalfBitDelay/2);

}

void i2c\_NoAck()

{

SCL=0;

delay(HalfBitDelay/2);

SDA=1;

delay(HalfBitDelay/2);

SCL=1;

delay(HalfBitDelay/2);

}

void i2c\_ack()

{

SCL=0;

delay(HalfBitDelay/2);

SDA=0;

delay(HalfBitDelay/2);

SCL=1;

delay(HalfBitDelay/2);

}

int i2c\_write\_byte(unsigned char byte)

{

printf\_tiny("\n\rwrite byte=%x\n\r",byte);

unsigned char i;

for(i=0;i<8;i++)

{

SCL=0;

delay(HalfBitDelay/2);

if((byte<<i)&0x08)

SDA=1;

else

SDA=0;

delay(HalfBitDelay/2);

SCL=1;

delay(HalfBitDelay);

}

SCL=0;

SDA=1;

delay(HalfBitDelay);

SCL=1;

delay(HalfBitDelay);

return SDA;

}

unsigned char i2c\_read\_byte(void)

{

unsigned char i,d,rxdata=0;

for(i=0;i<8;i++)

{

SCL=0;

SDA=1;

delay(HalfBitDelay);

SCL=1;

delay(HalfBitDelay/2);

d=SDA;

rxdata=rxdata|(SDA<<(7-i));

delay(HalfBitDelay/2);

}

return rxdata;

}

void i2c\_write()

{

char in\_address[3]; //Get address from user

uint16\_t in\_address\_int[3];

uint16\_t address;

uint8\_t slave\_add;

uint8\_t word\_add;

uint8\_t add\_flag;

uint8\_t data\_flag;

char in\_data[2];

uint16\_t in\_data\_int[2];

uint16\_t data;

uint8\_t i;

address=0;

data=0;

do

{

add\_flag=0;

printf\_tiny("\n\rEnter the address between 0x00-0x7ff\n\r");

for(i=0;i<3;i++)

{

in\_address[i]=getchar();

putchar(in\_address[i]);

}

for(i=0;i<3;i++)

{

switch(i)

{

case 0:

if((in\_address[i]>=48) && (in\_address[i]<=55))

{

in\_address\_int[i]= in\_address[i]-48;

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

add\_flag=1;

}

break;

case 1:

if(((in\_address[i]>='0') && (in\_address[i]<='9')) || ((in\_address[i]>='A') && (in\_address[i]<='F')) || ((in\_address[i]>='a') && (in\_address[i]<='f')))

{

if((in\_address[i]>='0') && (in\_address[i]<='9'))

{

in\_address\_int[i]=in\_address[i]-48;

}

else if((in\_address[i]>='A') && (in\_address[i]<='B'))

{

in\_address\_int[i]=in\_address[i]-55;

}

else if((in\_address[i]>='a') && (in\_address[i]<='b'))

{

in\_address\_int[i]=in\_address[i]-87;

}

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

add\_flag=1;

}

break;

case 2:

if(((in\_address[i]>='0') && (in\_address[i]<='9')) || ((in\_address[i]>='A') && (in\_address[i]<='F')) || ((in\_address[i]>='a') && (in\_address[i]<='f')))

{

if((in\_address[i]>='0') && (in\_address[i]<='9'))

{

in\_address\_int[i]=in\_address[i]-48;

}

else if((in\_address[i]>='A') && (in\_address[i]<='B'))

{

in\_address\_int[i]=in\_address[i]-55;

}

else if((in\_address[i]>='a') && (in\_address[i]<='b'))

{

in\_address\_int[i]=in\_address[i]-87;

}

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

add\_flag=1;

}

break;

default:break;

}

}

}while(add\_flag==1);

/\*-------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*Convert address into a number which is an integer\*/

for(i=0;i<3;i++)

{

address = address+((in\_address\_int[i])\*powf(16,2-i));

}

printf\_tiny("\n\rEnter the address between 0x00-0x7ff\n\r");

// address=address\_in();

slave\_add = ((0xA0)|(address >> 7)&(0xfe));

printf\_tiny("\n\rslave address=%x\n\r",slave\_add);

word\_add= (uint8\_t)address;

printf\_tiny("\n\rword address=%x\n\r",word\_add);

do

{

data\_flag=0;

printf\_tiny("\n\rEnter the data to write into EEPROM(\*Note:DATA SHOULD BE BETWEEN 0x00 and 0xFF)\n\r");

for(i=0;i<2;i++)

{

in\_data[i]=getchar();

putchar(in\_data[i]);

}

for(i=0;i<2;i++)

{

switch(i)

{

case 0:

if(((in\_data[i]>='0') && (in\_data[i]<='9')) || ((in\_data[i]>='A') && (in\_data[i]<='F')) || ((in\_data[i]>='a') && (in\_address[i]<='f')))

{

if((in\_data[i]>='0') && (in\_data[i]<='9'))

{

in\_data\_int[i]=in\_data[i]-48;

}

else if((in\_data[i]>='A') && (in\_data[i]<='B'))

{

in\_data\_int[i]=in\_data[i]-55;

}

else if((in\_data[i]>='a') && (in\_data[i]<='b'))

{

in\_data\_int[i]=in\_data[i]-87;

}

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

data\_flag=1;

}

break;

case 1:

if(((in\_data[i]>='0') && (in\_data[i]<='9')) || ((in\_data[i]>='A') && (in\_data[i]<='F')) || ((in\_data[i]>='a') && (in\_address[i]<='f')))

{

if((in\_data[i]>='0') && (in\_data[i]<='9'))

{

in\_data\_int[i]=in\_data[i]-48;

}

else if((in\_data[i]>='A') && (in\_data[i]<='B'))

{

in\_data\_int[i]=in\_data[i]-55;

}

else if((in\_data[i]>='a') && (in\_data[i]<='b'))

{

in\_data\_int[i]=in\_data[i]-87;

}

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

data\_flag=1;

}

break;

default:

break;

}

}if(data\_flag==1)

{

printf("\n\r Invalid data input\n\r");

}

}while(data\_flag==1);

for(i=0;i<2;i++)

{

data = data +((in\_data\_int[i])\*powf(16,1-i));

}

printf\_tiny("\n\rdata=%x\n\r",data);

start\_i2c();

printf("\n\rafter start in write");

i2c\_write\_byte(slave\_add);

printf("\n\rafter slave in write");

i2c\_write\_byte(word\_add);

printf("\n\rafter word in write");

i2c\_write\_byte(data);

printf("\n\rafter data in write");

stop\_i2c();

printf("\n\rafter stop in write");

}

void i2c\_read()

{

char in\_address[3]; //Get address from user

uint16\_t in\_address\_int[3];

uint16\_t address;

uint16\_t byte\_received;

uint8\_t add\_flag;

int i;

uint8\_t slave\_add;

uint8\_t word\_add;

uint8\_t slave\_read;

address=0;

byte\_received=0x00;

do

{

add\_flag=0;

printf\_tiny("\n\rEnter the address between 0x00-0x7ff\n\r");

for(i=0;i<3;i++)

{

in\_address[i]=getchar();

putchar(in\_address[i]);

}

for(i=0;i<3;i++)

{

switch(i)

{

case 0:

if((in\_address[i]>=48) && (in\_address[i]<=55))

{

in\_address\_int[i]= in\_address[i]-48;

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

add\_flag=1;

}

break;

case 1:

if(((in\_address[i]>='0') && (in\_address[i]<='9')) || ((in\_address[i]>='A') && (in\_address[i]<='F')) || ((in\_address[i]>='a') && (in\_address[i]<='f')))

{

if((in\_address[i]>='0') && (in\_address[i]<='9'))

{

in\_address\_int[i]=in\_address[i]-48;

}

else if((in\_address[i]>='A') && (in\_address[i]<='B'))

{

in\_address\_int[i]=in\_address[i]-55;

}

else if((in\_address[i]>='a') && (in\_address[i]<='b'))

{

in\_address\_int[i]=in\_address[i]-87;

}

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

add\_flag=1;

}

break;

case 2:

if(((in\_address[i]>='0') && (in\_address[i]<='9')) || ((in\_address[i]>='A') && (in\_address[i]<='F')) || ((in\_address[i]>='a') && (in\_address[i]<='f')))

{

if((in\_address[i]>='0') && (in\_address[i]<='9'))

{

in\_address\_int[i]=in\_address[i]-48;

}

else if((in\_address[i]>='A') && (in\_address[i]<='B'))

{

in\_address\_int[i]=in\_address[i]-55;

}

else if((in\_address[i]>='a') && (in\_address[i]<='b'))

{

in\_address\_int[i]=in\_address[i]-87;

}

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

add\_flag=1;

}

break;

default:break;

}

}

}while(add\_flag==1);

/\*-------------------------------------------------------------------------------------------------------------------------------------------\*/

/\*Convert address into a number which is an integer\*/

for(i=0;i<3;i++)

{

address = address+((in\_address\_int[i])\*powf(16,2-i));

}

printf\_tiny("\n\rEnter the address between 0x00-0x7ff\n\r");

// address=address\_in();

slave\_add = ((0xA0)|(address >> 7)&(0xfe));

slave\_read=((0xA0)|(address>>7)|(0x01));

// printf\_tiny("\n\rslave address=%x\n\r",slave\_add);

word\_add= (uint8\_t)address;

start\_i2c();

i2c\_write\_byte(slave\_add);

i2c\_write\_byte(word\_add);

restart\_i2c();

i2c\_write\_byte(slave\_read);

byte\_received=i2c\_read\_byte();

printf\_tiny("\n\rReceived byte=%x\n\r",byte\_received);

i2c\_NoAck();

stop\_i2c();

}

uint16\_t hex\_read(uint16\_t start\_addr)

{

uint8\_t hex\_add;

uint8\_t hex\_r;

uint8\_t word\_add;

uint16\_t byte\_received;

byte\_received=0x00;

hex\_add = ((0xA0)|(start\_addr >> 7)&(0xfe));

hex\_r = ((0xA0)|(start\_addr >> 7)|(0x01));

word\_add = (uint8\_t)start\_addr;

printf\_tiny("\n\rhex address=%x\n\r",hex\_add);

printf\_tiny("\n\rword address=%x\n\r",word\_add);

printf\_tiny("\n\rhex read address=%x\n\r",hex\_r);

start\_i2c();

i2c\_write\_byte(hex\_add);

i2c\_write\_byte(word\_add);

restart\_i2c();

i2c\_write\_byte(hex\_r);

byte\_received=i2c\_read\_byte();

i2c\_NoAck();

stop\_i2c();

//check ack

//check ack

//check ack

//reading the data from the particular address

return byte\_received;

}

uint16\_t address\_in(){

uint8\_t i;

char hex\_address[3];

uint16\_t hex\_address\_int[3];

uint16\_t address\_hex;

int add\_flag;

address\_hex=0;

do

{

add\_flag=0;

for(i=0;i<3;i++)

{

hex\_address[i]=getchar();

putchar(hex\_address[i]);

}

for(i=0;i<3;i++)

{

switch(i)

{

case 0:

if((hex\_address[i]>='0') && (hex\_address[i]<='7'))

{

hex\_address\_int[i]= hex\_address[i]-48;

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

add\_flag=1;

}

break;

case 1:

if(((hex\_address[i]>='0') && (hex\_address[i]<='9')) || ((hex\_address[i]>='A') && (hex\_address[i]<='F')) || ((hex\_address[i]>='a') && (hex\_address[i]<='f')))

{

if((hex\_address[i]>='0') && (hex\_address[i]<='9'))

{

hex\_address\_int[i]=hex\_address[i]-48;

}

else if((hex\_address[i]>='A') && (hex\_address[i]<='B'))

{

hex\_address\_int[i]=hex\_address[i]-55;

}

else if((hex\_address[i]>='a') && (hex\_address[i]<='b'))

{

hex\_address\_int[i]=hex\_address[i]-87;

}

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

add\_flag=1;

}

break;

case 2:

if(((hex\_address[i]>='0') && (hex\_address[i]<='9')) || ((hex\_address[i]>='A') && (hex\_address[i]<='F')) || ((hex\_address[i]>='a') && (hex\_address[i]<='f')))

{

if((hex\_address[i]>='0') && (hex\_address[i]<='9'))

{

hex\_address\_int[i]=hex\_address[i]-48;

}

else if((hex\_address[i]>='A') && (hex\_address[i]<='B'))

{

hex\_address\_int[i]=hex\_address[i]-55;

}

else if((hex\_address[i]>='a') && (hex\_address[i]<='b'))

{

hex\_address\_int[i]=hex\_address[i]-87;

}

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

add\_flag=1;

}

break;

default:

break;

}

}

if(add\_flag==1)

{

printf\_tiny("\n\rInvalid address. Please enter a valid address\n\r");

}

}while(add\_flag==1);

for(i=0;i<3;i++)

{

address\_hex = address\_hex +((hex\_address\_int[i])\*powf(16,2-i));

}

return (address\_hex);

}

void hex\_dump()

{

uint16\_t start\_address;

uint16\_t stop\_address;

uint16\_t data;

int address;

int invalid\_add=0;

int count=0;

start\_address=0;

stop\_address=0;

printf\_tiny("\n\rEnter the start address between 0x00-0x7ff\n\r");

start\_address=address\_in();

printf\_tiny("\n\rEnter the stop address between 0x00-0x7ff\n\r");

stop\_address=address\_in();

if(start\_address <= stop\_address)

{

printf\_tiny("\n\r Valid scene\n\r");

}

else

{

invalid\_add=1;

printf\_tiny("\n\rInvalid Entries! Please enter hex dump again\n\r");

}

if(invalid\_add == 0)

{

for(address =(start\_address);address <= (stop\_address); (address)++)

{

if(count%16 == 0)

{

data=hex\_read(address);

printf\_tiny("\n\r%x\t:\t%x",address,data);

}

else if(count %16 != 0)

{

data=hex\_read(address);

printf\_tiny("\t%x",data);

}

count++;

}

printf\_tiny("\n\r finished printing of %d bytes as requested\n\r",count);

}

}

void eeprom\_reset()

{

uint8\_t i;

start\_i2c();

for(i=0;i<9;i++)

{

SDA=1;

SCL=1;

delay(HalfBitDelay);

SCL=0;

delay(HalfBitDelay);

}

SCL=1;

delay(HalfBitDelay);

start\_i2c();

stop\_i2c();

}

void main()

{

init\_i2c();

// int flag=0;

char in;

printf\_tiny("\n\rEnter the operation you want to perform\n\r");

while(1)

{

/\*Printf statements for Menu\*/

printf\_tiny("\n\rPress 1 for 'Write Byte'\n\r");

printf\_tiny("\n\rPress 2 for 'Read Byte'\n\r");

printf\_tiny("\n\rPress 3 for 'Hex Dump'\n\r");

printf\_tiny("\n\rPress 4 for 'Reset EEPROM'\n\r");

// printf\_tiny("\n\rPress E for 'Exit'\n\r");

/\*To display entered characters on the serial terminal \*/

in=getchar();

putchar(in);

/\*Check for input entered and perform the corresponding operation\*/

/\*Write byte operation\*/

if(in== '1')

{

printf\_tiny("\n\rWriting byte\n\r");

i2c\_write();

}

else if(in=='2')

{

printf\_tiny("\n\rReading byte\n\r");

i2c\_read();

}

else if(in =='3')

{

printf\_tiny("\n\rHex dump\n\r");

hex\_dump();

}

else if(in =='4')

{

printf\_tiny("\n\rReset EEPROM\n\r");

eeprom\_reset();

}

else

{

printf\_tiny("\n\rInvalid input. Please enter a valid input\n\r");

}

}

// printf\_tiny("\n\rEXIT!\n\rPress the RESET button to start again!\n\r");

}

--------

/\*

\*/

#include <stdio.h>

#include <stdlib.h>

#include <mcs51reg.h>

#include <stdint.h>

#include <stdint.h>

#include <math.h>

#include <mcs51/8051.h>

#include <at89c51ed2.h>

#define SDA P1\_4 //Define SDA as Pin 1.4

#define SCL P1\_3 //Define SCL as Pin 1.3

// Define i2c speed

#define I2C\_SPEED\_FACTOR 1 // Low value means low i2c frequency

#define Crystal\_Value 12 // MHz

#define HalfBitDelay (500\*Crystal\_Value)/(12\*I2C\_SPEED\_FACTOR)

void delay(unsigned int a);

int getchar(void);

int putchar(int x);

void init\_i2c(void);

void start\_i2c();

void stop\_i2c();

void i2c\_NoAck();

void i2c\_ack();

unsigned char i2c\_read\_byte();

int i2c\_write\_byte(unsigned char byte);

uint16\_t validity(char input);

void i2c\_write();

void i2c\_read();

uint16\_t address\_in();

uint16\_t hex\_read(uint16\_t start\_addr);

void hex\_dump();

void eeprom\_reset();

//uint16\_t input\_num();

void delay(unsigned int a)

{

unsigned int i,limit;

limit = a/15;

for(i=0;i<limit;i++);

}

/\* Function to take input from keyboard.\*/

int getchar(void)

{

while(!RI)

{

;

}

RI=0;

return SBUF;

}

/\* Function to print on terminal. \*/

int putchar(int x)

{

while(!TI){

;

}

SBUF=x;

TI=0;

return x;

}

void start\_i2c()

{

SCL=1;

SDA=1;

delay(HalfBitDelay);

SDA=0;

delay(HalfBitDelay);

}

void init\_i2c(void)

{

// Make SDA and SCK pins input initially

SDA = 1;

SCL = 1;

}

void restart\_i2c()

{

SCL=0;

delay(HalfBitDelay/2);

SDA=1;

delay(HalfBitDelay/2);

SCL=1;

delay(HalfBitDelay/2);

SDA=0;

delay(HalfBitDelay/2);

}

void stop\_i2c()

{

SCL=0;

delay(HalfBitDelay/2);

SDA=0;

delay(HalfBitDelay/2);

SCL=1;

delay(HalfBitDelay/2);

SDA=1;

delay(HalfBitDelay/2);

}

void i2c\_NoAck()

{

SCL=0;

delay(HalfBitDelay/2);

SDA=1;

delay(HalfBitDelay/2);

SCL=1;

delay(HalfBitDelay/2);

}

void i2c\_ack()

{

SCL=0;

delay(HalfBitDelay/2);

SDA=0;

delay(HalfBitDelay/2);

SCL=1;

delay(HalfBitDelay/2);

}

uint16\_t validity(char input)

{

uint16\_t input\_int;

input\_int=0;

if(((input>='0') && (input<='9')) || ((input>='A') && (input<='F')) || ((input>='a') && (input<='f')))

{

if((input>='0') && (input<='9'))

{

input\_int=input-'0';

}

else if((input>='A') && (input<='F'))

{

input\_int=input-'A'+10;

}

else if((input>='a') && (input<='f'))

{

input\_int=input-'a'+10;

}

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

input\_int='z';

}

return input\_int;

}

int i2c\_write\_byte(unsigned char byte)

{

unsigned char i;

for(i=0;i<8;i++)

{

SCL=0;

delay(HalfBitDelay/2);

if((byte<<i)&0x08)

{

SDA=1;

}

else{

SDA=0;

}

delay(HalfBitDelay/2);

SCL=1;

delay(HalfBitDelay);

}

SCL=0;

SDA=1;

delay(HalfBitDelay);

SCL=1;

delay(HalfBitDelay);

return SDA;

}

unsigned char i2c\_read\_byte(void)

{

unsigned char i,d,rxdata=0;

for(i=0;i<8;i++)

{

SCL=0;

SDA=1;

delay(HalfBitDelay);

SCL=1;

delay(HalfBitDelay/2);

d=SDA;

rxdata=rxdata|(d<<(7-i));

delay(HalfBitDelay/2);

}

return rxdata;

}

void i2c\_write()

{

uint16\_t address;

uint8\_t slave\_add;

uint8\_t word\_add;

uint8\_t data\_flag;

char in\_data[2];

char in\_data\_int[2];

uint16\_t valid;

uint16\_t data;

uint8\_t i;

address=0;

data=0;

printf\_tiny("\n\rEnter the address between 0x00-0x7ff\n\r");

address=address\_in();

slave\_add = ((0xA0)|(address >> 7)&(0xfe));

printf\_tiny("\n\rslave address=%x\n\r",slave\_add);

word\_add= (uint8\_t)address;

printf\_tiny("\n\rword address=%x\n\r",word\_add);

do

{

data\_flag=0;

printf\_tiny("\n\rEnter the data to write into EEPROM(\*Note:DATA SHOULD BE BETWEEN 0x00 and 0xFF)\n\r");

for(i=0;i<2;i++)

{

in\_data[i]=getchar();

putchar(in\_data[i]);

}

for(i=0;i<2;i++)

{

switch(i)

{

case 0:

valid=validity(in\_data[i]);

if(valid=='z')

{

data\_flag=1;

}

else{

in\_data\_int[i]=valid;

}

break;

case 1:

valid=validity(in\_data[i]);

if(valid=='z')

{

data\_flag=1;

}

else{

in\_data\_int[i]=valid;

}

break;

default:

break;

}

}if(data\_flag==1)

{

printf("\n\r Invalid data input\n\r");

}

}while(data\_flag==1);

for(i=0;i<2;i++)

{

data = data +((in\_data\_int[i])\*powf(16,1-i));

}

printf\_tiny("\n\rdata=%x\n\r",data);

start\_i2c();

printf("\n\rafter start in write");

i2c\_write\_byte(slave\_add);

printf("\n\rafter slave in write");

i2c\_write\_byte(word\_add);

printf("\n\rafter word in write");

i2c\_write\_byte(data);

printf("\n\rafter data in write");

stop\_i2c();

printf("\n\rafter stop in write");

}

void i2c\_read()

{

uint16\_t address;

uint16\_t byte\_received;

address=0;

byte\_received=0x00;

printf\_tiny("\n\rEnter the address between 0x00-0x7ff\n\r");

address=address\_in();

byte\_received=hex\_read(address);

printf\_tiny("\n\rReceived byte=%x\n\r",byte\_received);

// stop\_i2c();

}

uint16\_t hex\_read(uint16\_t start\_addr)

{

uint8\_t hex\_add;

uint8\_t hex\_r;

uint8\_t word\_add;

uint16\_t byte\_received;

byte\_received=0x00;

hex\_add = ((0xA0)|(start\_addr >> 7)&(0xfe));

hex\_r = ((0xA0)|(start\_addr >> 7)|(0x01));

word\_add = (uint8\_t)start\_addr;

printf\_tiny("\n\rhex address=%x\n\r",hex\_add);

printf\_tiny("\n\rword address=%x\n\r",word\_add);

printf\_tiny("\n\rhex read address=%x\n\r",hex\_r);

start\_i2c();

i2c\_write\_byte(hex\_add);

i2c\_write\_byte(word\_add);

restart\_i2c();

i2c\_write\_byte(hex\_r);

byte\_received=i2c\_read\_byte();

i2c\_NoAck();

stop\_i2c();

//check ack

//check ack

//check ack

//reading the data from the particular address

return byte\_received;

}

uint16\_t address\_in(){

uint8\_t i;

char hex\_address[3];

uint16\_t hex\_address\_int[3];

uint16\_t address\_hex;

int add\_flag;

uint16\_t valid;

address\_hex=0;

do

{

add\_flag=0;

for(i=0;i<3;i++)

{

hex\_address[i]=getchar();

putchar(hex\_address[i]);

}

for(i=0;i<3;i++)

{

switch(i)

{

case 0:

if((hex\_address[i]>='0') && (hex\_address[i]<='7'))

{

hex\_address\_int[i]= hex\_address[i]-48;

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

add\_flag=1;

}

break;

case 1:

valid=validity(hex\_address[i]);

if(valid=='z')

{

add\_flag=1;

// printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

}

else{

hex\_address\_int[i]=valid;

}

break;

case 2:

valid=validity(hex\_address[i]);

if(valid=='z')

{

add\_flag=1;

// printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

}

else{

hex\_address\_int[i]=valid;

}

break;

default:

break;

}

}

if(add\_flag==1)

{

printf\_tiny("\n\rInvalid address. Please enter a valid address\n\r");

}

}while(add\_flag==1);

for(i=0;i<3;i++)

{

address\_hex = address\_hex +((hex\_address\_int[i])\*powf(16,2-i));

}

return (address\_hex);

}

void hex\_dump()

{

uint16\_t start\_address;

uint16\_t stop\_address;

uint16\_t data;

int address;

int invalid\_add=0;

int count=0;

start\_address=0;

stop\_address=0;

printf\_tiny("\n\rEnter the start address between 0x00-0x7ff\n\r");

start\_address=address\_in();

printf\_tiny("\n\rEnter the stop address between 0x00-0x7ff\n\r");

stop\_address=address\_in();

if(start\_address <= stop\_address)

{

printf\_tiny("\n\r Valid scene\n\r");

}

else

{

invalid\_add=1;

printf\_tiny("\n\rInvalid Entries! Please enter hex dump again\n\r");

}

if(invalid\_add == 0)

{

for(address =(start\_address);address <= (stop\_address); (address)++)

{

if(count%16 == 0)

{

data=hex\_read(address);

printf\_tiny("\n\r%x\t:\t%x",address,data);

}

else if(count %16 != 0)

{

data=hex\_read(address);

printf\_tiny("\t%x",data);

}

count++;

}

printf\_tiny("\n\r finished printing of %d bytes as requested\n\r",count);

}

}

void eeprom\_reset()

{

uint8\_t i;

start\_i2c();

for(i=0;i<9;i++)

{

SDA=1;

SCL=1;

delay(HalfBitDelay);

SCL=0;

delay(HalfBitDelay);

}

SCL=1;

delay(HalfBitDelay);

start\_i2c();

stop\_i2c();

}

void main()

{

init\_i2c();

// int flag=0;

char in;

printf\_tiny("\n\rEnter the operation you want to perform\n\r");

while(1)

{

/\*Printf statements for Menu\*/

printf\_tiny("\n\rPress 1 for 'Write Byte'\n\r");

printf\_tiny("\n\rPress 2 for 'Read Byte'\n\r");

printf\_tiny("\n\rPress 3 for 'Hex Dump'\n\r");

printf\_tiny("\n\rPress 4 for 'Reset EEPROM'\n\r");

// printf\_tiny("\n\rPress E for 'Exit'\n\r");

/\*To display entered characters on the serial terminal \*/

in=getchar();

putchar(in);

/\*Check for input entered and perform the corresponding operation\*/

/\*Write byte operation\*/

if(in== '1')

{

printf\_tiny("\n\rWriting byte\n\r");

i2c\_write();

}

else if(in=='2')

{

printf\_tiny("\n\rReading byte\n\r");

i2c\_read();

}

else if(in =='3')

{

printf\_tiny("\n\rHex dump\n\r");

hex\_dump();

}

else if(in =='4')

{

printf\_tiny("\n\rReset EEPROM\n\r");

eeprom\_reset();

}

else

{

printf\_tiny("\n\rInvalid input. Please enter a valid input\n\r");

}

}

// printf\_tiny("\n\rEXIT!\n\rPress the RESET button to start again!\n\r");

}

-----

/\*

\*/

#include <stdio.h>

#include <stdlib.h>

#include <mcs51reg.h>

#include <stdint.h>

#include <stdint.h>

#include <math.h>

#include <mcs51/8051.h>

#include <at89c51ed2.h>

#define SDA P1\_4 //Define SDA as Pin 1.4

#define SCL P1\_3 //Define SCL as Pin 1.3

// Define i2c speed

#define I2C\_SPEED\_FACTOR 1 // Low value means low i2c frequency

#define Crystal\_Value 12 // MHz

#define HalfBitDelay (500\*Crystal\_Value)/(12\*I2C\_SPEED\_FACTOR)

void delay(unsigned int a);

int getchar(void);

int putchar(int x);

void init\_i2c(void);

void start\_i2c();

void stop\_i2c();

void i2c\_NoAck();

void i2c\_ack();

unsigned char i2c\_read\_byte();

int i2c\_write\_byte(unsigned char byte);

uint16\_t validity(char input);

void i2c\_write();

void i2c\_read();

uint16\_t address\_in();

uint16\_t hex\_read(uint16\_t start\_addr);

void hex\_dump();

void eeprom\_reset();

//uint16\_t input\_num();

void delay(unsigned int a)

{

unsigned int i,limit;

limit = a/15;

for(i=0;i<limit;i++);

}

/\* Function to take input from keyboard.\*/

int getchar(void)

{

while(!RI)

{

;

}

RI=0;

return SBUF;

}

/\* Function to print on terminal. \*/

int putchar(int x)

{

while(!TI){

;

}

SBUF=x;

TI=0;

return x;

}

void start\_i2c()

{

SCL=1;

SDA=1;

delay(HalfBitDelay);

SDA=0;

delay(HalfBitDelay);

}

void init\_i2c(void)

{

// Make SDA and SCK pins input initially

SDA = 1;

SCL = 1;

}

void restart\_i2c()

{

SCL=0;

delay(HalfBitDelay/2);

SDA=1;

delay(HalfBitDelay/2);

SCL=1;

delay(HalfBitDelay/2);

SDA=0;

delay(HalfBitDelay/2);

}

void stop\_i2c()

{

SCL=0;

delay(HalfBitDelay/2);

SDA=0;

delay(HalfBitDelay/2);

SCL=1;

delay(HalfBitDelay/2);

SDA=1;

delay(HalfBitDelay/2);

}

void i2c\_NoAck()

{

SCL=0;

delay(HalfBitDelay/2);

SDA=1;

delay(HalfBitDelay/2);

SCL=1;

delay(HalfBitDelay);

}

void i2c\_ack()

{

SCL=0;

delay(HalfBitDelay/2);

SDA=0;

delay(HalfBitDelay/2);

SCL=1;

delay(HalfBitDelay);

}

uint16\_t validity(char input)

{

uint16\_t input\_int;

input\_int=0;

if(((input>='0') && (input<='9')) || ((input>='A') && (input<='F')) || ((input>='a') && (input<='f')))

{

if((input>='0') && (input<='9'))

{

input\_int=input-'0';

}

else if((input>='A') && (input<='F'))

{

input\_int=input-'A'+10;

}

else if((input>='a') && (input<='f'))

{

input\_int=input-'a'+10;

}

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

input\_int='z';

}

return input\_int;

}

int i2c\_write\_byte(unsigned char byte)

{

unsigned char i;

for(i=0;i<8;i++)

{

SCL=0;

delay(HalfBitDelay/2);

if((byte<<i)&0x08)

{

SDA=1;

}

else{

SDA=0;

}

delay(HalfBitDelay/2);

SCL=1;

delay(HalfBitDelay);

}

SCL=0;

SDA=1;

delay(HalfBitDelay);

SCL=1;

delay(HalfBitDelay);

return SDA;

}

unsigned char i2c\_read\_byte()

{

unsigned char i,d,rxdata=0;

for(i=0;i<8;i++)

{

SCL=0;

SDA=1;

delay(HalfBitDelay);

SCL=1;

delay(HalfBitDelay/2);

d=SDA;

rxdata=rxdata|(d<<(7-i));

delay(HalfBitDelay/2);

}

return rxdata;

}

void i2c\_write()

{

char in\_address[3];

uint16\_t in\_address\_int[3];

uint16\_t address;

uint8\_t slave\_add;

uint8\_t word\_add;

uint8\_t add\_flag;

uint8\_t data\_flag;

char in\_data[2];

char in\_data\_int[2];

uint16\_t valid;

uint16\_t data;

uint8\_t i;

address=0;

data=0;

do

{

add\_flag=0;

printf\_tiny("\n\rEnter the address between 0x00-0x7ff\n\r");

for(i=0;i<3;i++)

{

in\_address[i]=getchar();

putchar(in\_address[i]);

}

for(i=0;i<3;i++)

{

switch(i)

{

case 0:

if((in\_address[i]>='0') && (in\_address[i]<='7'))

{

in\_address\_int[i]= in\_address[i]-'0';

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

add\_flag=1;

}

break;

case 1:

valid=validity(in\_address[i]);

if(valid=='z')

{

add\_flag=1;

// printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

}

else{

in\_address\_int[i]=valid;

}

break;

case 2:

valid=validity(in\_address[i]);

if(valid=='z')

{

add\_flag=1;

// printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

}

else{

in\_address\_int[i]=valid;

}

break;

default:

break;

}

}

}while(add\_flag==1);

for(i=0;i<3;i++)

{

address = address+((in\_address\_int[i])\*powf(16,2-i));

}

slave\_add = ((0xA0)|(address >> 7)&(0xfe));

word\_add= (uint8\_t)address;

do

{

data\_flag=0;

printf\_tiny("\n\rEnter the data to write into EEPROM(\*Note:DATA SHOULD BE BETWEEN 0x00 and 0xFF)\n\r");

for(i=0;i<2;i++)

{

in\_data[i]=getchar();

putchar(in\_data[i]);

}

for(i=0;i<2;i++)

{

switch(i)

{

case 0:

valid=validity(in\_data[i]);

if(valid=='z')

{

data\_flag=1;

}

else{

in\_data\_int[i]=valid;

}

break;

case 1:

valid=validity(in\_data[i]);

if(valid=='z')

{

data\_flag=1;

}

else{

in\_data\_int[i]=valid;

}

break;

default:

break;

}

}

}while(data\_flag==1);

for(i=0;i<2;i++)

{

data = data +((in\_data\_int[i])\*powf(16,1-i));

}

start\_i2c();

i2c\_write\_byte(slave\_add);

//check ack

i2c\_write\_byte(word\_add);

//check ack

i2c\_write\_byte(data);

//check ack

stop\_i2c();

}

void i2c\_read()

{

char in\_address[3];

uint16\_t in\_address\_int[3];

uint16\_t address;

uint8\_t add\_flag;

// uint8\_t data\_flag;

// uint8\_t slave\_add;

// uint8\_t slave\_read;

// uint8\_t word\_add;

uint16\_t valid;

// int z;

// uint16\_t dat;

uint16\_t byte\_received;

uint8\_t i;

address=0;

byte\_received=0x00;

do

{

add\_flag=0;

printf\_tiny("\n\rEnter the address between 0x00-0x7ff\n\r");

for( i=0;i<3;i++)

{

in\_address[i]=getchar();

putchar(in\_address[i]);

}

for(i=0;i<3;i++)

{

switch(i)

{

case 0:

if((in\_address[i]>='0') && (in\_address[i]<='7'))

{

in\_address\_int[i]= in\_address[i]-'0';

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

add\_flag=1;

}

break;

case 1:

valid=validity(in\_address[i]);

if(valid=='z')

{

add\_flag=1;

// printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

}

else{

in\_address\_int[i]=valid;

}

break;

case 2:

valid=validity(in\_address[i]);

if(valid=='z')

{

add\_flag=1;

// printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

}

else{

in\_address\_int[i]=valid;

}

break;

default:

break;

}

}

}while(add\_flag==1);

for(i=0;i<3;i++)

{

address = address+((in\_address\_int[i])\*powf(16,2-i));

}

slave\_add = ((0xA0)|(address >> 7)&(0xfe));

word\_add= (uint8\_t)address;

start\_i2c();

i2c\_write\_byte(slave\_add);

i2c\_write\_byte(word\_add);

restart\_i2c();

i2c\_write\_byte((0xA1)|(address >> 7));

byte\_received=i2c\_read\_byte();

printf\_tiny("\n\rReceived byte=%x\n\r",byte\_received);

i2c\_NoAck();

stop\_i2c();

}

uint16\_t hex\_read(uint16\_t start\_addr)

{

uint8\_t hex\_add;

uint8\_t hex\_r;

uint8\_t word\_add;

uint16\_t byte\_received;

byte\_received=0x00;

int i;

int z;

uint16\_t dat;

hex\_add = ((0xA0)|(start\_addr >> 7)&(0xfe));

hex\_r = ((0xA0)|(start\_addr >> 7)|(0x01));

word\_add = (uint8\_t)start\_addr;

start\_i2c();

i2c\_write\_byte(hex\_add);

i2c\_write\_byte(word\_add);

restart\_i2c();

i2c\_write\_byte(hex\_r);

//check ack

//check ack

//check ack

//reading the data from the particular address

return byte\_received;

}

void main()

{

unsigned char RxByte = 0;

init\_i2c(); // Initialize i2c pins

start\_i2c(); // Send start bit on i2c

i2c\_write\_byte(0xA0); // Send 0xA0 on i2c

RxByte = i2c\_read\_byte(); // Read value from i2c

i2c\_ack(); // Send ACK bit on i2c

stop\_i2c(); // Send stop bit on i2c

while(1)

{

}

}

------

void i2c\_write()

{

char in\_address[3];

uint16\_t in\_address\_int[3];

uint16\_t address;

uint8\_t slave\_add;

uint8\_t word\_add;

uint8\_t add\_flag;

uint8\_t data\_flag;

char in\_data[2];

char in\_data\_int[2];

uint16\_t valid;

uint16\_t data;

uint8\_t i;

address=0;

data=0;

do

{

add\_flag=0;

printf\_tiny("\n\rEnter the address between 0x00-0x7ff\n\r");

for(i=0;i<3;i++)

{

in\_address[i]=getchar();

putchar(in\_address[i]);

}

for(i=0;i<3;i++)

{

switch(i)

{

case 0:

if((in\_address[i]>='0') && (in\_address[i]<='7'))

{

in\_address\_int[i]= in\_address[i]-'0';

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

add\_flag=1;

}

break;

case 1:

valid=validity(in\_address[i]);

if(valid=='z')

{

add\_flag=1;

// printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

}

else{

in\_address\_int[i]=valid;

}

break;

case 2:

valid=validity(in\_address[i]);

if(valid=='z')

{

add\_flag=1;

// printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

}

else{

in\_address\_int[i]=valid;

}

break;

default:

break;

}

}

}while(add\_flag==1);

for(i=0;i<3;i++)

{

address = address+((in\_address\_int[i])\*powf(16,2-i));

}

slave\_add = ((0xA0)|(address >> 7)&(0xfe));

word\_add= (uint8\_t)address;

do

{

data\_flag=0;

printf\_tiny("\n\rEnter the data to write into EEPROM(\*Note:DATA SHOULD BE BETWEEN 0x00 and 0xFF)\n\r");

for(i=0;i<2;i++)

{

in\_data[i]=getchar();

putchar(in\_data[i]);

}

for(i=0;i<2;i++)

{

switch(i)

{

case 0:

valid=validity(in\_data[i]);

if(valid=='z')

{

data\_flag=1;

}

else{

in\_data\_int[i]=valid;

}

break;

case 1:

valid=validity(in\_data[i]);

if(valid=='z')

{

data\_flag=1;

}

else{

in\_data\_int[i]=valid;

}

break;

default:

break;

}

}

}while(data\_flag==1);

for(i=0;i<2;i++)

{

data = data +((in\_data\_int[i])\*powf(16,1-i));

}

start\_i2c();

// SCL=0;

for (int i=0;i<8;i++)

{

SCL=0;

delay(HalfBitDelay);

if((slave\_add & 0x80) == 0)

{

SDA = 0;

delay(HalfBitDelay);

}

else

{

SDA = 1;

delay(HalfBitDelay);

}

SCL = 1;

delay(HalfBitDelay);

// SCL=0;

slave\_add<<=1;

}

SCL=0;

SDA=1;

delay(HalfBitDelay);

SCL=1;

delay(HalfBitDelay);

//check ack

// SCL=0;

for (int i=0;i<8;i++)

{

SCL=0;

delay(HalfBitDelay);

if((word\_add & 0x80) == 0)

{

SDA = 0;

delay(HalfBitDelay);

}

else

{

SDA = 1;

delay(HalfBitDelay);

}

SCL = 1;

delay(HalfBitDelay);

SCL=0;

word\_add<<=1;

}

//check ack

SCL=0;

SDA=1;

delay(HalfBitDelay);

SCL=1;

delay(HalfBitDelay);

// SCL=0;

for (int i=0;i<8;i++)

{

if((data & 0x80) == 0)

{

SDA = 0;

delay(HalfBitDelay);

}

else

{

SDA = 1;

delay(HalfBitDelay);

}

SCL = 1;

delay(HalfBitDelay);

// SCL=0;

data<<=1;

}

//check ack

SCL=0;

SDA=1;

delay(HalfBitDelay);

SCL=1;

delay(HalfBitDelay);

stop\_i2c();

}

void i2c\_read()

{

char in\_address[3];

uint16\_t in\_address\_int[3];

uint16\_t address;

uint8\_t add\_flag;

// uint8\_t data\_flag;

// uint8\_t slave\_add;

// uint8\_t slave\_read;

// uint8\_t word\_add;

uint16\_t valid;

// int z;

// uint16\_t dat;

uint16\_t byte\_received;

uint8\_t i;

address=0;

byte\_received=0x00;

do

{

add\_flag=0;

printf\_tiny("\n\rEnter the address between 0x00-0x7ff\n\r");

for( i=0;i<3;i++)

{

in\_address[i]=getchar();

putchar(in\_address[i]);

}

for(i=0;i<3;i++)

{

switch(i)

{

case 0:

if((in\_address[i]>='0') && (in\_address[i]<='7'))

{

in\_address\_int[i]= in\_address[i]-'0';

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

add\_flag=1;

}

break;

case 1:

valid=validity(in\_address[i]);

if(valid=='z')

{

add\_flag=1;

// printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

}

else{

in\_address\_int[i]=valid;

}

break;

case 2:

valid=validity(in\_address[i]);

if(valid=='z')

{

add\_flag=1;

// printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

}

else{

in\_address\_int[i]=valid;

}

break;

default:

break;

}

}

}while(add\_flag==1);

for(i=0;i<3;i++)

{

address = address+((in\_address\_int[i])\*powf(16,2-i));

}

byte\_received=hex\_read(address);

printf\_tiny("\n\rReceived byte=%x\n\r",byte\_received);

i2c\_NoAck();

stop\_i2c();

}

uint16\_t address\_in(){

uint8\_t i;

char hex\_address[3];

uint16\_t hex\_address\_int[3];

uint16\_t address\_hex;

int add\_flag;

uint16\_t valid;

address\_hex=0;

do

{

add\_flag=0;

for(i=0;i<3;i++)

{

hex\_address[i]=getchar();

putchar(hex\_address[i]);

}

for(i=0;i<3;i++)

{

switch(i)

{

case 0:

if((hex\_address[i]>='0') && (hex\_address[i]<='7'))

{

hex\_address\_int[i]= hex\_address[i]-48;

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

add\_flag=1;

}

break;

case 1:

valid=validity(hex\_address[i]);

if(valid=='z')

{

add\_flag=1;

// printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

}

else{

hex\_address\_int[i]=valid;

}

break;

case 2:

valid=validity(hex\_address[i]);

if(valid=='z')

{

add\_flag=1;

// printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

}

else{

hex\_address\_int[i]=valid;

}

break;

default:

break;

}

}

}while(add\_flag==1);

for(i=0;i<3;i++)

{

address\_hex = address\_hex +((hex\_address\_int[i])\*powf(16,2-i));

}

return (address\_hex);

}

uint16\_t hex\_read(uint16\_t start\_addr)

{

uint8\_t hex\_add;

uint8\_t hex\_r;

uint8\_t word\_add;

uint16\_t byte\_received;

byte\_received=0x00;

int i;

int z;

uint16\_t dat;

hex\_add = ((0xA0)|(start\_addr >> 7)&(0xfe));

hex\_r = ((0xA0)|(start\_addr >> 7)|(0x01));

word\_add = (uint8\_t)start\_addr;

start\_i2c();

for (i=0;i<8;i++)

{

//SCL=0;

if((hex\_add & 0x80) == 0)

{

SDA = 0;

delay(HalfBitDelay);

}

else

{

SDA = 1;

delay(HalfBitDelay);

}

SCL = 1;

delay(HalfBitDelay);

SCL=0;

hex\_add<<=1;

}

//check ack

SCL=0;

for (i=0;i<8;i++)

{

if((word\_add & 0x80) == 0)

{

SDA = 0;

delay(HalfBitDelay);

}

else

{

SDA = 1;

delay(HalfBitDelay);

}

SCL = 1;

delay(HalfBitDelay);

SCL=0;

word\_add<<=1;

}

//check ack

SCL=0;

delay(HalfBitDelay);

start\_i2c();

for (i=0;i<8;i++)

{

if((hex\_r & 0x80) == 0)

{

SDA = 0;

delay(HalfBitDelay);

}

else

{

SDA = 1;

delay(HalfBitDelay);

}

SCL = 1;

delay(HalfBitDelay);

SCL=0;

hex\_r<<=1;

}

//check ack

//reading the data from the particular address

SDA=1; //Relase the bus/set SDA as input

SCL=1;

delay(HalfBitDelay);

SCL=0;

for(i=0;i<8;i++)

{

SDA=1;

delay(HalfBitDelay);

SCL=1;

delay(HalfBitDelay);

if(SDA==1)

{

z=1;

}

else

{

z=0;

}

dat=z;

// printf\_tiny("\n\r[%d]-->%d\n\r",7-i,dat);

byte\_received |= (dat<<(7-i)) ;

SCL=1;

delay(HalfBitDelay);

SCL=0;

delay(HalfBitDelay);

}

return byte\_received;

}

void hex\_dump()

{

uint16\_t start\_address;

uint16\_t stop\_address;

uint16\_t data;

int address;

int invalid\_add=0;

int count=0;

printf\_tiny("\n\rEnter the start address between 0x00-0x7ff\n\r");

start\_address=address\_in();

printf\_tiny("\n\rEnter the stop address between 0x00-0x7ff\n\r");

stop\_address=address\_in();

if(start\_address <= stop\_address)

{

printf\_tiny("\n\r Valid scene\n\r");

}

else

{

invalid\_add=1;

printf\_tiny("\n\rInvalid Entries! Please enter hex dump again\n\r");

}

if(invalid\_add == 0)

{

for(address =(start\_address);address <= (stop\_address); (address)++)

{

if(address==start\_address)

{

data=hex\_read(start\_address);

printf\_tiny("\n\r%x\t:\t%x",start\_address,data);

}

else if(count %15 != 0)

{

data=hex\_read(address);

printf\_tiny("\t%x",data);

}

else if(count%15 == 0)

{

data=hex\_read(address);

printf\_tiny("\n\r%x\t:\t%x",address,data);

}

count++;

}

printf\_tiny("\n\r finished printing of %d bytes as requested\n\r",count);

}

}

void eeprom\_reset()

{

uint8\_t i;

start\_i2c();

for(i=0;i<9;i++)

{

SDA=1;

delay(HalfBitDelay);

SCL=1;

delay(HalfBitDelay);

SCL=0;

delay(HalfBitDelay);

}

SCL=1;

delay(HalfBitDelay);

start\_i2c();

stop\_i2c();

}

void main()

{

// int flag=0;

char in;

printf\_tiny("\n\rEnter the operation you want to perform\n\r");

while(1)

{

/\*Printf statements for Menu\*/

printf\_tiny("\n\rPress 1 for 'Write Byte'\n\r");

printf\_tiny("\n\rPress 2 for 'Read Byte'\n\r");

printf\_tiny("\n\rPress 3 for 'Hex Dump'\n\r");

printf\_tiny("\n\rPress 4 for 'Reset EEPROM'\n\r");

// printf\_tiny("\n\rPress E for 'Exit'\n\r");

/\*To display entered characters on the serial terminal \*/

in=getchar();

putchar(in);

/\*Check for input entered and perform the corresponding operation\*/

/\*Write byte operation\*/

if(in== '1')

{

printf\_tiny("\n\rWriting byte\n\r");

i2c\_write();

}

else if(in=='2')

{

printf\_tiny("\n\rReading byte\n\r");

i2c\_read();

}

else if(in =='3')

{

printf\_tiny("\n\rHex dump\n\r");

hex\_dump();

}

else if(in =='4')

{

printf\_tiny("\n\rReset EEPROM\n\r");

eeprom\_reset();

}

else

{

printf\_tiny("\n\rInvalid input. Please enter a valid input\n\r");

}

}

// printf\_tiny("\n\rEXIT!\n\rPress the RESET button to start again!\n\r");

}

--------------------------------------------/\*

\*/

#include <stdio.h>

#include <stdlib.h>

#include <mcs51reg.h>

#include <stdint.h>

#include <stdint.h>

#include <math.h>

#include <mcs51/8051.h>

#include <at89c51ed2.h>

#define SDA P1\_4 //Define SDA as Pin 1.4

#define SCL P1\_3 //Define SCL as Pin 1.3

void delay();

int getchar(void);

int putchar(int x);

void start\_i2c();

void stop\_i2c();

void i2c\_NoAck();

int i2c\_ack();

uint16\_t validity(char input);

void i2c\_write();

void i2c\_read();

uint16\_t address\_in();

uint16\_t hex\_read(uint16\_t start\_addr);

void hex\_dump();

void eeprom\_reset();

//uint16\_t input\_num();

void delay()

{

int i;

for(i=0;i<500;i++);

}

/\* Function to take input from keyboard.\*/

int getchar(void)

{

while(!RI)

{

;

}

RI=0;

return SBUF;

}

/\* Function to print on terminal. \*/

int putchar(int x)

{

while(!TI){

;

}

SBUF=x;

TI=0;

return x;

}

void start\_i2c()

{

SDA=1;

delay();

SCL=1;

delay();

SDA=0;

delay();

SCL=0;

delay();

}

void stop\_i2c()

{

SCL=0;

SDA=0;

SCL=1;

SDA=1;

}

void i2c\_NoAck()

{

SDA=1;

delay();

SCL=1;

delay();

SCL=0;

delay();

SDA=0;

}

int i2c\_ack()

{

SDA=1;

delay();

SCL=0;

delay();

SCL=1;

delay();

if(SDA==0)

{

return 1;

}

else

{

return 0;

}

}

uint16\_t validity(char input)

{

uint16\_t input\_int;

input\_int=0;

if(((input>='0') && (input<='9')) || ((input>='A') && (input<='F')) || ((input>='a') && (input<='f')))

{

if((input>='0') && (input<='9'))

{

input\_int=input-'0';

}

else if((input>='A') && (input<='F'))

{

input\_int=input-'A'+10;

}

else if((input>='a') && (input<='f'))

{

input\_int=input-'a'+10;

}

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

input\_int='z';

}

return input\_int;

}

void i2c\_write()

{

char in\_address[3];

uint16\_t in\_address\_int[3];

uint16\_t address;

uint8\_t slave\_add;

uint8\_t word\_add;

uint8\_t add\_flag;

uint8\_t data\_flag;

char in\_data[2];

char in\_data\_int[2];

uint16\_t valid;

uint16\_t data;

uint8\_t i;

address=0;

data=0;

do

{

add\_flag=0;

printf\_tiny("\n\rEnter the address between 0x00-0x7ff\n\r");

for(i=0;i<3;i++)

{

in\_address[i]=getchar();

putchar(in\_address[i]);

}

for(i=0;i<3;i++)

{

switch(i)

{

case 0:

if((in\_address[i]>='0') && (in\_address[i]<='7'))

{

in\_address\_int[i]= in\_address[i]-'0';

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

add\_flag=1;

}

break;

case 1:

valid=validity(in\_address[i]);

if(valid=='z')

{

add\_flag=1;

// printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

}

else{

in\_address\_int[i]=valid;

}

break;

case 2:

valid=validity(in\_address[i]);

if(valid=='z')

{

add\_flag=1;

// printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

}

else{

in\_address\_int[i]=valid;

}

break;

default:

break;

}

}

}while(add\_flag==1);

for(i=0;i<3;i++)

{

address = address+((in\_address\_int[i])\*powf(16,2-i));

}

slave\_add = ((0xA0)|(address >> 7)&(0xfe));

word\_add= (uint8\_t)address;

do

{

data\_flag=0;

printf\_tiny("\n\rEnter the data to write into EEPROM(\*Note:DATA SHOULD BE BETWEEN 0x00 and 0xFF)\n\r");

for(i=0;i<2;i++)

{

in\_data[i]=getchar();

putchar(in\_data[i]);

}

for(i=0;i<2;i++)

{

switch(i)

{

case 0:

valid=validity(in\_data[i]);

if(valid=='z')

{

data\_flag=1;

}

else{

in\_data\_int[i]=valid;

}

break;

case 1:

valid=validity(in\_data[i]);

if(valid=='z')

{

data\_flag=1;

}

else{

in\_data\_int[i]=valid;

}

break;

default:

break;

}

}

}while(data\_flag==1);

for(i=0;i<2;i++)

{

data = data +((in\_data\_int[i])\*powf(16,1-i));

}

start\_i2c();

// SCL=0;

for (int i=0;i<8;i++)

{

SCL=0;

delay();

if((slave\_add & 0x80) == 0)

{

SDA = 0;

delay();

}

else

{

SDA = 1;

delay();

}

SCL = 1;

delay();

// SCL=0;

slave\_add<<=1;

}

SCL=0;

SDA=1;

delay();

SCL=1;

delay();

//check ack

// SCL=0;

for (int i=0;i<8;i++)

{

SCL=0;

delay();

if((word\_add & 0x80) == 0)

{

SDA = 0;

delay();

}

else

{

SDA = 1;

delay();

}

SCL = 1;

delay();

SCL=0;

word\_add<<=1;

}

//check ack

SCL=0;

SDA=1;

delay();

SCL=1;

delay();

// SCL=0;

for (int i=0;i<8;i++)

{

if((data & 0x80) == 0)

{

SDA = 0;

delay();

}

else

{

SDA = 1;

delay();

}

SCL = 1;

delay();

// SCL=0;

data<<=1;

}

//check ack

SCL=0;

SDA=1;

delay();

SCL=1;

delay();

stop\_i2c();

}

void i2c\_read()

{

char in\_address[3];

uint16\_t in\_address\_int[3];

uint16\_t address;

uint8\_t add\_flag;

// uint8\_t data\_flag;

// uint8\_t slave\_add;

// uint8\_t slave\_read;

// uint8\_t word\_add;

uint16\_t valid;

// int z;

// uint16\_t dat;

uint16\_t byte\_received;

uint8\_t i;

address=0;

byte\_received=0x00;

do

{

add\_flag=0;

printf\_tiny("\n\rEnter the address between 0x00-0x7ff\n\r");

for( i=0;i<3;i++)

{

in\_address[i]=getchar();

putchar(in\_address[i]);

}

for(i=0;i<3;i++)

{

switch(i)

{

case 0:

if((in\_address[i]>='0') && (in\_address[i]<='7'))

{

in\_address\_int[i]= in\_address[i]-'0';

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

add\_flag=1;

}

break;

case 1:

valid=validity(in\_address[i]);

if(valid=='z')

{

add\_flag=1;

// printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

}

else{

in\_address\_int[i]=valid;

}

break;

case 2:

valid=validity(in\_address[i]);

if(valid=='z')

{

add\_flag=1;

// printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

}

else{

in\_address\_int[i]=valid;

}

break;

default:

break;

}

}

}while(add\_flag==1);

for(i=0;i<3;i++)

{

address = address+((in\_address\_int[i])\*powf(16,2-i));

}

byte\_received=hex\_read(address);

printf\_tiny("\n\rReceived byte=%x\n\r",byte\_received);

i2c\_NoAck();

stop\_i2c();

}

uint16\_t address\_in(){

uint8\_t i;

char hex\_address[3];

uint16\_t hex\_address\_int[3];

uint16\_t address\_hex;

int add\_flag;

uint16\_t valid;

address\_hex=0;

do

{

add\_flag=0;

for(i=0;i<3;i++)

{

hex\_address[i]=getchar();

putchar(hex\_address[i]);

}

for(i=0;i<3;i++)

{

switch(i)

{

case 0:

if((hex\_address[i]>='0') && (hex\_address[i]<='7'))

{

hex\_address\_int[i]= hex\_address[i]-48;

}

else

{

printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

add\_flag=1;

}

break;

case 1:

valid=validity(hex\_address[i]);

if(valid=='z')

{

add\_flag=1;

// printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

}

else{

hex\_address\_int[i]=valid;

}

break;

case 2:

valid=validity(hex\_address[i]);

if(valid=='z')

{

add\_flag=1;

// printf\_tiny("\n\rInvalid Input! Please enter again\n\r");

}

else{

hex\_address\_int[i]=valid;

}

break;

default:

break;

}

}

}while(add\_flag==1);

for(i=0;i<3;i++)

{

address\_hex = address\_hex +((hex\_address\_int[i])\*powf(16,2-i));

}

return (address\_hex);

}

uint16\_t hex\_read(uint16\_t start\_addr)

{

uint8\_t hex\_add;

uint8\_t hex\_r;

uint8\_t word\_add;

uint16\_t byte\_received;

byte\_received=0x00;

int i;

int z;

uint16\_t dat;

hex\_add = ((0xA0)|(start\_addr >> 7)&(0xfe));

hex\_r = ((0xA0)|(start\_addr >> 7)|(0x01));

word\_add = (uint8\_t)start\_addr;

start\_i2c();

for (i=0;i<8;i++)

{

//SCL=0;

if((hex\_add & 0x80) == 0)

{

SDA = 0;

delay();

}

else

{

SDA = 1;

delay();

}

SCL = 1;

delay();

SCL=0;

hex\_add<<=1;

}

//check ack

SCL=0;

for (i=0;i<8;i++)

{

if((word\_add & 0x80) == 0)

{

SDA = 0;

delay();

}

else

{

SDA = 1;

delay();

}

SCL = 1;

delay();

SCL=0;

word\_add<<=1;

}

//check ack

SCL=0;

delay();

start\_i2c();

for (i=0;i<8;i++)

{

if((hex\_r & 0x80) == 0)

{

SDA = 0;

delay();

}

else

{

SDA = 1;

delay();

}

SCL = 1;

delay();

SCL=0;

hex\_r<<=1;

}

//check ack

//reading the data from the particular address

SDA=1; //Relase the bus/set SDA as input

SCL=1;

delay();

SCL=0;

for(i=0;i<8;i++)

{

SDA=1;

delay();

SCL=1;

delay();

if(SDA==1)

{

z=1;

}

else

{

z=0;

}

dat=z;

// printf\_tiny("\n\r[%d]-->%d\n\r",7-i,dat);

byte\_received |= (dat<<(7-i)) ;

SCL=1;

delay();

SCL=0;

delay();

}

return byte\_received;

}

void hex\_dump()

{

uint16\_t start\_address;

uint16\_t stop\_address;

uint16\_t data;

int address;

int invalid\_add=0;

int count=0;

printf\_tiny("\n\rEnter the start address between 0x00-0x7ff\n\r");

start\_address=address\_in();

printf\_tiny("\n\rEnter the stop address between 0x00-0x7ff\n\r");

stop\_address=address\_in();

if(start\_address <= stop\_address)

{

printf\_tiny("\n\r Valid scene\n\r");

}

else

{

invalid\_add=1;

printf\_tiny("\n\rInvalid Entries! Please enter hex dump again\n\r");

}

if(invalid\_add == 0)

{

for(address =(start\_address);address <= (stop\_address); (address)++)

{

if(address==start\_address)

{

data=hex\_read(start\_address);

printf\_tiny("\n\r%x\t:\t%x",start\_address,data);

}

else if(count %16 != 0)

{

data=hex\_read(address);

printf\_tiny("\t%x",data);

}

else if(count%16 == 0)

{

data=hex\_read(address);

printf\_tiny("\n\r%x\t:\t%x",address,data);

}

count++;

}

printf\_tiny("\n\r finished printing of %d bytes as requested\n\r",count);

}

}

void eeprom\_reset()

{

uint8\_t i;

start\_i2c();

for(i=0;i<9;i++)

{

SDA=1;

delay();

SCL=1;

delay();

SCL=0;

delay();

}

SCL=1;

delay();

start\_i2c();

stop\_i2c();

}

void main()

{

// int flag=0;

char in;

printf\_tiny("\n\rEnter the operation you want to perform\n\r");

while(1)

{

/\*Printf statements for Menu\*/

printf\_tiny("\n\rPress 1 for 'Write Byte'\n\r");

printf\_tiny("\n\rPress 2 for 'Read Byte'\n\r");

printf\_tiny("\n\rPress 3 for 'Hex Dump'\n\r");

printf\_tiny("\n\rPress 4 for 'Reset EEPROM'\n\r");

// printf\_tiny("\n\rPress E for 'Exit'\n\r");

/\*To display entered characters on the serial terminal \*/

in=getchar();

putchar(in);

/\*Check for input entered and perform the corresponding operation\*/

/\*Write byte operation\*/

if(in== '1')

{

printf\_tiny("\n\rWriting byte\n\r");

i2c\_write();

}

else if(in=='2')

{

printf\_tiny("\n\rReading byte\n\r");

i2c\_read();

}

else if(in =='3')

{

printf\_tiny("\n\rHex dump\n\r");

hex\_dump();

}

else if(in =='4')

{

printf\_tiny("\n\rReset EEPROM\n\r");

eeprom\_reset();

}

else

{

printf\_tiny("\n\rInvalid input. Please enter a valid input\n\r");

}

}

// printf\_tiny("\n\rEXIT!\n\rPress the RESET button to start again!\n\r");

}