CUSTOM CHARACRWE IS PRINTING TWICE

#include <stdint.h>

#include <stdio.h>

#include <mcs51/8051.h>

#include <mcs51/at89c51ed2.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

#include "LCD.h"

// LCD control macros

#define RS\_CMD P1\_6 = 0

#define RS\_DATA P1\_6 = 1

#define LCD\_WRITE P1\_7 = 0

#define LCD\_READ P1\_7 = 1

// LCD row base addresses

#define LCD\_R0\_BASE (0x80)

#define LCD\_R1\_BASE (0xC0)

#define LCD\_R2\_BASE (0x90)

#define LCD\_R3\_BASE (0xD0)

// LCD memory addresses

#define LCD\_CMD\_READ\_ADD ((\_\_xdata uint8\_t\*)0x8002)

#define LCD\_CMD\_WRITE\_ADD ((\_\_xdata uint8\_t\*)0x8004)

#define LCD\_DATA\_READ\_ADD ((\_\_xdata uint8\_t\*)0x8000)

#define LCD\_DATA\_WRITE\_ADD ((\_\_xdata uint8\_t\*)0x8000)

// LCD constants

#define LCD\_WIDTH (16)

#define LCD\_HEIGHT (4)

#define LCD\_CLRSCR (0x01)

#define LCD\_CURSOR\_HOME (0x02)

#define LCD\_WAIT\_MASK (0x80)

#define LCD\_DDRAM\_ADD\_MASK (0x80)

#define LCD\_ENTRY\_MODE (0x06)

#define LCD\_DISPLAY\_CURSOR (0x0F)

#define LCD\_DISPLAY\_SHIFT (0x14)

#define LCD\_FXN\_SET (0x38)

#define LCD\_UNLOCK (0x30)

#define LCD\_OFF (0x08)

#define LCD\_ON (0x0F)

#define LCD\_CURSOR\_OFF (0x0C)

#define lcd\_ddram\_address\_mask (0x80)

#define lcd\_cgram\_address\_mask (0x40)

#define lcd\_cgram\_address\_mask\_2 (0x7F)

#define TIMER0\_LOW (0xEF)

#define TIMER0\_HIGH (0x73)

// Function prototypes

void timer\_init(void);

void timer\_display(void);

void timer\_stop(void);

void timer\_start(void);

void timer\_reset(void);

void lcd\_row\_change(void);

void lcd\_init(void);

void lcd\_busy\_wait(void);

void lcd\_goto\_xy(\_\_xdata uint8\_t row,\_\_xdata uint8\_t column);

void lcd\_putchar(\_\_xdata uint8\_t data\_byte);

void lcd\_putstr(\_\_xdata uint8\_t\* text\_ptr);

void lcd\_cmd\_write(\_\_xdata uint8\_t data\_byte);

void lcd\_goto\_add(\_\_xdata uint16\_t address);

int putchar (int ch);

int getchar (void);

void delay\_ms(uint32\_t time);

uint16\_t fetch\_number(uint8\_t base);

uint8\_t char\_to\_int(uint8\_t temp);

void lcd\_clear(void);

void lcd\_ddram\_flush(void);

void lcd\_cgram\_flush(void);

void print\_number\_hex(\_\_xdata uint32\_t number,\_\_xdata uint8\_t display\_width);

int8\_t int\_to\_char(int temp);

void lcd\_print\_number(\_\_xdata uint32\_t number,\_\_xdata uint8\_t display\_width);

void lcdcreatechar(unsigned char ccode, unsigned char row\_vals[]);

// Global variables

\_\_xdata uint8\_t lcd\_current\_column,lcd\_current\_row,timer\_on\_off\_flag;

\_\_xdata uint32\_t time;

/\*\*

\* @brief External startup function for SDCC.

\*

\* This function initializes specific control registers for the microcontroller.

\*

\* @return 0 indicating successful startup.

\*/

\_sdcc\_external\_startup()

{

AUXR |= 0X0C; // Set some control register (specific to the microcontroller)

return 0;

}

void main(void)

{

\_\_xdata uint8\_t key\_pressed=0,exit\_code=0,data\_byte=0, x=0, y=0, i=0;

\_\_xdata uint16\_t address=0;

\_\_xdata uint8\_t Welcome\_lcd\_txt[] = "WELCOME";

printf\_tiny("\n\rWELCOME");

lcd\_init();

lcd\_putstr(Welcome\_lcd\_txt);

timer\_on\_off\_flag = 1;

timer\_init();

while(1)

{

printf\_tiny("\n\rMenu:");

printf\_tiny("\n\r 1. LCD Jump address");

printf\_tiny("\n\r 2. clear LCD");

printf\_tiny("\n\r 3. LCD Jump co-ordinates");

printf\_tiny("\n\r 4. Put string");

printf\_tiny("\n\r 5. Start or stop time");

printf\_tiny("\n\r 6. Reset time ");

printf\_tiny("\n\r 7. LCD DDRAM Dump ");

printf\_tiny("\n\r 8. LCD CGRAM Dump ");

printf\_tiny("\n\r 9. Create custom character \n\r");

printf\_tiny("Entered input: ");

key\_pressed= getchar();

putchar(key\_pressed);

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

if(timer\_on\_off\_flag == 1)

{

TR0 = 0;

}

switch(key\_pressed)

{

case '1':

{

printf\_tiny("\n\r LCD Jump address");

printf\_tiny("\n\r Enter the address:\t");

address = fetch\_number(16);

lcd\_goto\_add(address);

break;

}

case '2':

{

lcd\_clear();

break;

}

case '3':

{

if(timer\_on\_off\_flag == 1)

{

TR0 = 1;

}

printf\_tiny("\n\rLCD Jump co-ordinates");

printf\_tiny("\n\rEnter X coordinate(row):");

x = fetch\_number(10);

printf\_tiny("\n\rEnter Y coordinate(row):");

y = fetch\_number(10);

lcd\_goto\_xy(x,y);

break;

}

case '4':

{

printf("Enter the string ");

\_\_xdata uint8\_t input\_string[50];

uint8\_t i = 0;

// Read characters until Enter is pressed (13)

while ((input\_string[i] = getchar()) != 13)

{

putchar(input\_string[i]);

i++;

}

input\_string[i] = '\0'; // Null-terminate the string

lcd\_putstr(input\_string);

break;

}

case '5':

{

if(timer\_on\_off\_flag == 1)

{

timer\_stop();

timer\_on\_off\_flag = 0;

}

else

{

timer\_start();

timer\_on\_off\_flag = 1;

}

break;

}

case '6':

{

if(timer\_on\_off\_flag == 1)

{

TR0 = 1;

}

timer\_reset();

break;

}

case '7':

{

lcd\_ddram\_flush();

break;

}

case '8':

{

lcd\_cgram\_flush();

break;

}

// Inside the while loop in the main function

case '9':

{

printf\_tiny("\n\rCreate Custom Character");

printf\_tiny("\n\rEnter character code (0-7): ");

uint8\_t ccode = fetch\_number(10);

printf\_tiny("\n\rEnter pixel pattern for each row (8 rows): ");

unsigned char row\_vals[8];

for (uint8\_t i = 0; i < 8; ++i) {

printf\_tiny("\n\rRow %d: ", i);

row\_vals[i] = fetch\_number(2); // Assuming user enters binary values, you can modify as needed

lcd\_putchar(row\_vals[i]); // Display the current state on the LCD

}

lcdcreatechar(ccode, row\_vals);

break;

}

default:

{

printf\_tiny("\n\r Invalid input!!\n\r");

break;

}

}

if(timer\_on\_off\_flag == 1)

{

TR0 = 1;

}

}

}

/\*\*

\* @brief Interrupt handler for Timer 0.

\*

\* This function is called when Timer 0 overflows to generate interrupts.

\* It updates the time variable and calls the timer\_display function.

\*/

void rtc\_interrupt\_handler(void) \_\_interrupt 1

{

\_\_xdata static uint16\_t counter\_flag = 0;

TF0 = 0;

TL0 = TIMER0\_LOW;

TH0 = TIMER0\_HIGH;

if(counter\_flag == 0)

{

counter\_flag = 1;

}

else

{

time++;

timer\_display();

counter\_flag=0;

}

}

/\*\*

\* @brief Initializes the LCD.

\*/

void lcd\_init(void)

{

delay\_ms(15);

RS\_CMD;

LCD\_WRITE;

\*(LCD\_CMD\_WRITE\_ADD) = LCD\_UNLOCK;

delay\_ms(5);

RS\_CMD;

LCD\_WRITE;

\*(LCD\_CMD\_WRITE\_ADD) = LCD\_UNLOCK;

RS\_CMD;

LCD\_WRITE;

delay\_ms(1);

RS\_CMD;

LCD\_WRITE;

\*(LCD\_CMD\_WRITE\_ADD) = LCD\_UNLOCK;

lcd\_cmd\_write(LCD\_FXN\_SET);

lcd\_cmd\_write(LCD\_OFF);

lcd\_cmd\_write(LCD\_ON);

lcd\_cmd\_write(LCD\_ENTRY\_MODE);

lcd\_cmd\_write(LCD\_CLRSCR);

return;

}

/\*\*

\* @brief Changes the LCD row.

\*/

void lcd\_row\_change(void)

{

\_\_xdata uint8\_t lcd\_row\_base\_address[]={LCD\_R0\_BASE,LCD\_R1\_BASE,LCD\_R2\_BASE,LCD\_R3\_BASE};

if(lcd\_current\_row >= 3)

{

lcd\_current\_row = 0;

}

else

{

lcd\_current\_row++;

}

lcd\_goto\_add(lcd\_row\_base\_address[lcd\_current\_row]);

return;

}

/\*\*

\* @brief Waits until the LCD is not busy.

\*/

void lcd\_busy\_wait(void)

{

RS\_CMD;

LCD\_READ;

while(\*LCD\_CMD\_READ\_ADD & LCD\_WAIT\_MASK);

return;

}

/\*\*

\* @brief Sets the LCD to a specific address.

\*

\* @param address The address to set in the LCD.

\*/

void lcd\_goto\_add(\_\_xdata uint16\_t address)

{

if((address>=0x90&&address<=0x9F)||(address>=0xD0&&address<=0xDF)||(address>=0xC0&&address<=0xCF)||(address>=0x80&&address<=0x8F))

{

lcd\_busy\_wait();

address |= LCD\_DDRAM\_ADD\_MASK;

lcd\_cmd\_write(address);

lcd\_current\_column = address % LCD\_WIDTH;

if(address>=0xD0&&address<=0xDF)

{

lcd\_current\_row = 3;

}

if(address>=0x90&&address<=0x9F)

{

lcd\_current\_row = 2;

}

if(address>=0xC0&&address<=0xCF)

{

lcd\_current\_row = 1;

}

if(address>=0x80&&address<=0x8F)

{

lcd\_current\_row = 0;

}

}

else

{

printf\_tiny("\n\rInvalid address");

}

return;

}

/\*\*

\* @brief Sets the cursor position on the LCD.

\*

\* @param row The row where the cursor should be placed.

\* @param column The column where the cursor should be placed.

\*/

void lcd\_goto\_xy(\_\_xdata uint8\_t row,\_\_xdata uint8\_t column)

{

\_\_xdata uint8\_t lcd\_row\_base\_address[]={LCD\_R0\_BASE,LCD\_R1\_BASE,LCD\_R2\_BASE,LCD\_R3\_BASE};

\_\_xdata uint8\_t address = 0;

if(row >= LCD\_HEIGHT)

{

printf\_tiny("\n\rInvalid Row number");

return;

}

if(column >= LCD\_WIDTH)

{

printf\_tiny("\n\rInvalid Column number");

return;

}

address = lcd\_row\_base\_address[row] + column;

lcd\_goto\_add(address);

return;

}

/\*\*

\* @brief Writes a command to the LCD.

\*

\* @param data\_byte The command to write to the LCD.

\*/

void lcd\_cmd\_write(\_\_xdata uint8\_t data\_byte)

{

lcd\_busy\_wait();

RS\_CMD;

LCD\_WRITE;

\*(LCD\_CMD\_WRITE\_ADD) = data\_byte;

return;

}

/\*\*

\* @brief Writes a character to the LCD.

\*

\* @param data\_byte The character to write to the LCD.

\*/

void lcd\_putchar(\_\_xdata uint8\_t data\_byte)

{

lcd\_busy\_wait();

RS\_DATA;

LCD\_WRITE;

\*(LCD\_DATA\_WRITE\_ADD) = data\_byte;

lcd\_current\_column++;

if(lcd\_current\_column>=16)

{

lcd\_row\_change();

}

return;

}

/\*\*

\* @brief Writes a string to the LCD.

\*

\* @param text\_ptr Pointer to the string to be written to the LCD.

\*/

void lcd\_putstr(\_\_xdata uint8\_t\* text\_ptr)

{

while(\*text\_ptr!='\0')

{

lcd\_putchar(\*text\_ptr);

text\_ptr++;

}

return;

}

/\*\*

\* @brief Clears the LCD screen.

\*/

void lcd\_clear(void)

{

lcd\_cmd\_write(LCD\_CLRSCR);

lcd\_current\_row=0;

lcd\_current\_column=0;

return;

}

/\*\*

\* @brief Writes a character to the UART.

\*

\* @param ch The character to be written to the UART.

\* @return The written character.

\*/

int putchar (int ch)

{

while (TI==0)

{

;

}

SBUF = ch;

TI = 0;

return ch;

}

/\*\*

\* @brief Reads a character from the UART.

\*

\* @return The character read from the UART.

\*/

int getchar (void)

{

while (RI==0)

{

;

}

RI = 0;

return SBUF;

}

/\*\*

\* @brief Delays for a specified time in milliseconds.

\*

\* @param time The time to delay in milliseconds.

\*/

void delay\_ms(uint32\_t time)

{

uint32\_t i=0,j=0;

for(j=0;j<time;j++)

{

for(i=0;i<1120;i++); //1 ms

}

return;

}

/\*\*

\* @brief Fetches a number from the user.

\*

\* @param base The base of the number (e.g., 10 for decimal, 16 for hexadecimal).

\* @return The fetched number.

\*/

uint16\_t fetch\_number(uint8\_t base)

{

\_\_xdata uint8\_t scanned\_digit=0,digit\_array[20],digit\_counter=0,i=0;

\_\_xdata uint16\_t number=0;

while(scanned\_digit!=13)

{

scanned\_digit=getchar();

if(((scanned\_digit >= '0') && (scanned\_digit <= '9'))||

((scanned\_digit >= 'a') && (scanned\_digit <= 'f'))||

((scanned\_digit >= 'A') && (scanned\_digit <= 'F')))

{

putchar(scanned\_digit);

digit\_array[digit\_counter]=char\_to\_int(scanned\_digit);

digit\_counter++;

}

else if(scanned\_digit==8) //check for backspace

{

putchar(8); //print backspace

putchar(32); //print space

putchar(8); //print backspace

digit\_counter--;

}

}

for(i=0;i<digit\_counter;i++)

{

number\*= base;

number+= digit\_array[i];

}

return number;

}

/\*\*

\* @brief Converts a character to an integer.

\*

\* @param temp The character to be converted.

\* @return The integer value of the character.

\*/

uint8\_t char\_to\_int(uint8\_t temp)

{

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

{

temp-='0';

}

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

{

temp-='a';

temp+=10;

}

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

{

temp-='A';

temp+=10;

}

return temp;

}

/\*\*

\* @brief Initializes Timer 0 for timekeeping.

\*/

void timer\_init(void)

{

time = 0;

IE |= 0x82;

TMOD |= 0x01;

TMOD &= 0xF1;

TL0 = TIMER0\_LOW;

TH0 = TIMER0\_HIGH;

TR0 = 1;

return;

}

/\*\*

\* @brief Displays the current time on the LCD.

\*/

void timer\_display(void)

{

\_\_xdata uint8\_t tenth\_of\_second = 0, seconds = 0, minutes = 0, temp\_lcd\_row=0,temp\_lcd\_column=0;

temp\_lcd\_column = lcd\_current\_column;

temp\_lcd\_row = lcd\_current\_row;

tenth\_of\_second = time % 10;

seconds = (time/10) % 60;

minutes = time/600;

lcd\_goto\_xy(3,8);

lcd\_print\_number(minutes,2);

lcd\_putchar(':');

lcd\_print\_number(seconds,2);

lcd\_putchar('.');

lcd\_print\_number(tenth\_of\_second,1);

lcd\_goto\_xy(temp\_lcd\_row,temp\_lcd\_column);

return;

}

/\*\*

\* @brief Stops Timer 0.

\*/

void timer\_stop(void)

{

TR0 = 0;

}

/\*\*

\* @brief Starts Timer 0.

\*/

void timer\_start(void)

{

TR0 = 1;

}

/\*\*

\* @brief Resets the time variable.

\*/

void timer\_reset(void)

{

time = 0;

}

/\*\*

\* @brief Prints a number on the LCD with a specified width.

\*

\* @param number The number to be printed.

\* @param display\_width The width to display the number.

\*/

void lcd\_print\_number(\_\_xdata uint32\_t number,\_\_xdata uint8\_t display\_width)

{

\_\_xdata uint8\_t temp\_ascii\_store[10],temp\_data=0;

\_\_xdata int8\_t counter=0;

\_\_xdata uint32\_t value\_check=0;

for(counter=display\_width;counter>1;counter--)

{

lcd\_busy\_wait();

switch(counter)

{

case 2:

{

value\_check = 9;

if(number<=value\_check)

{

RS\_DATA;

LCD\_WRITE;

\*(LCD\_DATA\_WRITE\_ADD) = '0';

}

break;

}

case 3:

{

value\_check = 99;

if(number<=value\_check)

{

RS\_DATA;

LCD\_WRITE;

\*(LCD\_DATA\_WRITE\_ADD) = '0';

}

break;

}

case 4:

{

value\_check = 999;

if(number<=value\_check)

{

RS\_DATA;

LCD\_WRITE;

\*(LCD\_DATA\_WRITE\_ADD) = '0';

}

break;

}

}

}

do

{

temp\_ascii\_store[counter]='0'+ number%10;

number/=10;

counter++;

}while(number>0);

for(counter-=1;counter>0;counter--)

{

lcd\_busy\_wait();

RS\_DATA;

LCD\_WRITE;

\*(LCD\_DATA\_WRITE\_ADD) = temp\_ascii\_store[counter];

}

return;

}

void lcd\_ddram\_flush(void)

{

\_\_xdata uint8\_t count=0,i=0, temp\_storage=0, data\_byte=0;

\_\_xdata uint16\_t address=0x80;

temp\_storage = lcd\_ddram\_address\_mask;

RS\_CMD;

LCD\_WRITE;

lcd\_cmd\_write(temp\_storage);

printf\_tiny("\n\rDDRAM Hexdump");

for(count=0;count<5;count++)

{

putchar('\n');

putchar('\r');

print\_number\_hex(address,2);

putchar(':');

for(i=0;i<16;i++)

{

putchar(32);//space

RS\_DATA;

LCD\_READ;

data\_byte = \*(LCD\_DATA\_READ\_ADD);

print\_number\_hex(data\_byte,2);

}

address+=16;

}

lcd\_goto\_xy(lcd\_current\_row,lcd\_current\_column);

return;

}

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

\* @brief lcd\_cgram\_flush()

\* Function to hex dump the contents of CGRAM on UART

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

void lcd\_cgram\_flush(void)

{

\_\_xdata uint8\_t count=0, i=0, temp\_storage=0, data\_byte=0;

\_\_xdata uint16\_t address=0;

temp\_storage = lcd\_cgram\_address\_mask;

temp\_storage &= lcd\_cgram\_address\_mask\_2;

RS\_CMD;

LCD\_WRITE;

lcd\_cmd\_write(temp\_storage);

printf\_tiny("\n\rCGRAM Hexdump");

for(count=0; count<4; count++)

{

putchar('\n');

putchar('\r');

print\_number\_hex(address,2);

putchar(':');

for(i=0;i<16;i++)

{

putchar(32);//space

RS\_DATA;

LCD\_READ;

data\_byte = \*(LCD\_DATA\_READ\_ADD);

print\_number\_hex(data\_byte,2);

}

address+=16;

}

lcd\_goto\_xy(lcd\_current\_row,lcd\_current\_column);

return;

}

void print\_number\_hex(\_\_xdata uint32\_t number,\_\_xdata uint8\_t display\_width)

{

\_\_xdata uint8\_t temp\_ascii\_store[10],temp\_value=0;

\_\_xdata int8\_t counter=0;

\_\_xdata uint32\_t value\_check=0;

for(counter=display\_width;counter>1;counter--)

{

switch(counter)

{

case 4:

{

value\_check = 0xFFF;

if(number<=value\_check)

{

putchar('0');

}

break;

}

case 3:

{

value\_check = 0xFF;

if(number<=value\_check)

{

putchar('0');

}

break;

}

case 2:

{

value\_check = 0xF;

if(number<=value\_check)

{

putchar('0');

}

break;

}

}

}

do

{

temp\_ascii\_store[counter]=int\_to\_char(number%16);

number/=16;

counter++;

}while(number>0);

for(counter-=1;counter>=0;counter--)

{

putchar(temp\_ascii\_store[counter]);

}

return;

}

int8\_t int\_to\_char(int temp) /\*Function that includes switch statement for hex definition\*/

{

switch(temp) /\*switch case for defining hex characters above 9 i.e. 'A' to 'F'\*/

{ case 1:

return '1';

case 0:

return '0';

case 2:

return '2';

case 3:

return '3';

case 4:

return '4';

case 5:

return '5';

case 6:

return '6';

case 7:

return '7';

case 8:

return '8';

case 9:

return '9';

case 10:

return 'A';

case 11:

return 'B';

case 12:

return 'C';

case 13:

return 'D';

case 14:

return 'E';

case 15:

return 'F';

/\*To convert int to char\*/

}

return '0';

}

/\*\*

\* @brief Creates a custom character in CGRAM.

\*

\* @param ccode The character code (0 <= ccode <= 7).

\* @param row\_vals An 8-byte array containing the pixel pattern for each row.

\*/

void lcdcreatechar(unsigned char ccode, unsigned char row\_vals[]) {

if (ccode >= 8) {

printf\_tiny("\n\rInvalid character code. It must be between 0 and 7.");

return;

}

// Set CGRAM address for the custom character

lcd\_cmd\_write(lcd\_cgram\_address\_mask | (ccode << 3));

// Write the pixel pattern for each row to CGRAM

for (uint8\_t i = 0; i < 8; ++i) {

lcd\_putchar(row\_vals[i]);

}

printf\_tiny("\n\rCustom character created with code %d.", ccode);

lcd\_goto\_xy(lcd\_current\_row, lcd\_current\_column); // Return to the original position

}

#include <stdint.h>

#include <stdio.h>

#include <mcs51/8051.h>

#include <mcs51/at89c51ed2.h>

#include <stdlib.h>

#include <math.h>

#include<string.h>

// LCD control macros

#define RS\_CMD P1\_6 = 0

#define RS\_DATA P1\_6 = 1

#define LCD\_WRITE P1\_7 = 0

#define LCD\_READ P1\_7 = 1

// LCD row base addresses

#define LCD\_R0\_BASE (0x80)

#define LCD\_R1\_BASE (0xC0)

#define LCD\_R2\_BASE (0x90)

#define LCD\_R3\_BASE (0xD0)

// LCD memory addresses

#define LCD\_CMD\_READ\_ADD ((\_\_xdata uint8\_t\*)0x8002)

#define LCD\_CMD\_WRITE\_ADD ((\_\_xdata uint8\_t\*)0x8004)

#define LCD\_DATA\_READ\_ADD ((\_\_xdata uint8\_t\*)0x8000)

#define LCD\_DATA\_WRITE\_ADD ((\_\_xdata uint8\_t\*)0x8000)

// LCD constants

#define LCD\_WIDTH (16)

#define LCD\_HEIGHT (4)

#define LCD\_CLRSCR (0x01)

#define LCD\_CURSOR\_HOME (0x02)

#define LCD\_WAIT\_MASK (0x80)

#define LCD\_DDRAM\_ADD\_MASK (0x80)

#define LCD\_ENTRY\_MODE (0x06)

#define LCD\_DISPLAY\_CURSOR (0x0F)

#define LCD\_DISPLAY\_SHIFT (0x14)

#define LCD\_FXN\_SET (0x38)

#define LCD\_UNLOCK (0x30)

#define LCD\_OFF (0x08)

#define LCD\_ON (0x0F)

#define LCD\_CURSOR\_OFF (0x0C)

#define TIMER0\_LOW (0xEF)

#define TIMER0\_HIGH (0x73)

#define lcd\_ddram\_address\_mask (0x80)

#define lcd\_cgram\_address\_mask (0x40)

#define lcd\_cgram\_address\_mask\_2 (0x7F)

// Function prototypes

void lcd\_print\_number(\_\_xdata int32\_t number,\_\_xdata uint8\_t display\_width);

void timer\_init(void);

void timer\_display(void);

void timer\_stop(void);

void timer\_start(void);

void timer\_reset(void);

void lcd\_row\_change(void);

void lcd\_init(void);

void lcd\_busy\_wait(void);

void lcd\_goto\_add(\_\_xdata uint8\_t address);

void lcd\_goto\_xy(\_\_xdata uint8\_t row,\_\_xdata uint8\_t column);

void lcd\_putchar(\_\_xdata uint8\_t data\_byte);

void lcd\_putstr(\_\_xdata uint8\_t\* text\_ptr);

void lcd\_cmd\_write(\_\_xdata uint8\_t data\_byte);

int putchar (int ch);

int getchar (void);

void delay\_ms(uint32\_t time);

uint16\_t fetch\_number(uint8\_t base);

uint8\_t char\_to\_int(uint8\_t temp);

void lcd\_clear(void);

void lcd\_ddram\_flush(void);

void lcd\_cgram\_flush(void);

void print\_number\_hex(\_\_xdata uint32\_t number,\_\_xdata uint8\_t display\_width);

int8\_t int\_to\_char(int temp);

// Global variables

\_\_xdata uint8\_t lcd\_current\_column,lcd\_current\_row,timer\_on\_off\_flag;

\_\_xdata uint32\_t time;

/\*\*

\* @brief External startup function for SDCC.

\*

\* This function initializes specific control registers for the microcontroller.

\*

\* @return 0 indicating successful startup.

\*/

\_sdcc\_external\_startup()

{

AUXR |= 0X0C; // Set some control register (specific to the microcontroller)

return 0;

}

void main(void)

{

\_\_xdata uint8\_t key\_pressed=0,exit\_code=0,data\_byte=0, x=0, y=0, i=0;

\_\_xdata uint16\_t address=0;

\_\_xdata uint8\_t Welcome\_lcd\_txt[] = "WELCOME";

printf\_tiny("\n\rWELCOME");

lcd\_init();

lcd\_putstr(Welcome\_lcd\_txt);

timer\_on\_off\_flag = 1;

timer\_init();

while(1)

{

printf\_tiny("\n\rMenu:");

printf\_tiny("\n\r 1. LCD Jump address");

printf\_tiny("\n\r 2. clear LCD");

printf\_tiny("\n\r 3. LCD Jump co-ordinates");

printf\_tiny("\n\r 4. Put string");

printf\_tiny("\n\r 5. Start or stop time");

printf\_tiny("\n\r 6. Reset time \n\r");

printf\_tiny("\n\r 7. LCD DDRAM Dump \n\r");

printf\_tiny("\n\r 8. LCD CGRAM Dump \n\r");

printf\_tiny("Entered input: ");

key\_pressed= getchar();

putchar(key\_pressed);

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

if(timer\_on\_off\_flag == 1)

{

TR0 = 0;

}

switch(key\_pressed)

{

case '1':

{

printf\_tiny("\n\r LCD Jump address");

printf\_tiny("\n\r Enter the address:\t");

address = fetch\_number(16);

lcd\_goto\_add(address);

break;

}

case '2':

{

lcd\_clear();

break;

}

case '3':

{

if(timer\_on\_off\_flag == 1)

{

TR0 = 1;

}

printf\_tiny("\n\rLCD Jump co-ordinates");

printf\_tiny("\n\rEnter X coordinate(row):");

x = fetch\_number(10);

printf\_tiny("\n\rEnter Y coordinate(row):");

y = fetch\_number(10);

lcd\_goto\_xy(x,y);

break;

}

case '4':

{

printf("Enter the string ");

\_\_xdata uint8\_t input\_string[50];

uint8\_t i = 0;

// Read characters until Enter is pressed (13)

while ((input\_string[i] = getchar()) != 13)

{

putchar(input\_string[i]);

i++;

}

input\_string[i] = '\0'; // Null-terminate the string

lcd\_putstr(input\_string);

break;

}

case '5':

{

if(timer\_on\_off\_flag == 1)

{

timer\_stop();

timer\_on\_off\_flag = 0;

}

else

{

timer\_start();

timer\_on\_off\_flag = 1;

}

break;

}

case '6':

{

if(timer\_on\_off\_flag == 1)

{

TR0 = 1;

}

timer\_reset();

break;

}

case '7':

{

lcd\_ddram\_flush();

break;

}

case '8':

{

lcd\_cgram\_flush();

break;

}

default:

{

printf\_tiny("\n\r Invalid input!!\n\r");

break;

}

}

if(timer\_on\_off\_flag == 1)

{

TR0 = 1;

}

}

}

/\*\*

\* @brief Initializes the LCD.

\*/

void lcd\_init(void)

{

delay\_ms(15);

RS\_CMD;

LCD\_WRITE;

\*(LCD\_CMD\_WRITE\_ADD) = LCD\_UNLOCK;

delay\_ms(5);

RS\_CMD;

LCD\_WRITE;

\*(LCD\_CMD\_WRITE\_ADD) = LCD\_UNLOCK;

RS\_CMD;

LCD\_WRITE;

delay\_ms(1);

RS\_CMD;

LCD\_WRITE;

\*(LCD\_CMD\_WRITE\_ADD) = LCD\_UNLOCK;

lcd\_cmd\_write(LCD\_FXN\_SET);

lcd\_cmd\_write(LCD\_OFF);

lcd\_cmd\_write(LCD\_ON);

lcd\_cmd\_write(LCD\_ENTRY\_MODE);

lcd\_cmd\_write(LCD\_CLRSCR);

return;

}

/\*\*

\* @brief Changes the LCD row.

\*/

void lcd\_row\_change(void)

{

\_\_xdata uint8\_t lcd\_row\_base\_address[]={LCD\_R0\_BASE,LCD\_R1\_BASE,LCD\_R2\_BASE,LCD\_R3\_BASE};

if(lcd\_current\_row >= 3)

{

lcd\_current\_row = 0;

}

else

{

lcd\_current\_row++;

}

lcd\_goto\_add(lcd\_row\_base\_address[lcd\_current\_row]);

return;

}

/\*\*

\* @brief Waits until the LCD is not busy.

\*/

void lcd\_busy\_wait(void)

{

RS\_CMD;

LCD\_READ;

while(\*LCD\_CMD\_READ\_ADD & LCD\_WAIT\_MASK);

return;

}

/\*\*

\* @brief Sets the LCD to a specific address.

\*

\* @param address The address to set in the LCD.

\*/

void lcd\_goto\_add(\_\_xdata uint8\_t address)

{

if((address>=0x90&&address<=0x9F)||(address>=0xD0&&address<=0xDF)||(address>=0xC0&&address<=0xCF)||(address>=0x80&&address<=0x8F))

{

lcd\_busy\_wait();

address |= LCD\_DDRAM\_ADD\_MASK;

lcd\_cmd\_write(address);

lcd\_current\_column = address % LCD\_WIDTH;

if(address>=0xD0&&address<=0xDF)

{

lcd\_current\_row = 3;

}

if(address>=0x90&&address<=0x9F)

{

lcd\_current\_row = 2;

}

if(address>=0xC0&&address<=0xCF)

{

lcd\_current\_row = 1;

}

if(address>=0x80&&address<=0x8F)

{

lcd\_current\_row = 0;

}

}

else

{

printf\_tiny("\n\rInvalid address");

}

return;

}

/\*\*

\* @brief Sets the cursor position on the LCD.

\*

\* @param row The row where the cursor should be placed.

\* @param column The column where the cursor should be placed.

\*/

void lcd\_goto\_xy(\_\_xdata uint8\_t row,\_\_xdata uint8\_t column)

{

\_\_xdata uint8\_t lcd\_row\_base\_address[]={LCD\_R0\_BASE,LCD\_R1\_BASE,LCD\_R2\_BASE,LCD\_R3\_BASE};

\_\_xdata uint8\_t address = 0;

if(row >= LCD\_HEIGHT)

{

printf\_tiny("\n\rInvalid Row number");

return;

}

if(column >= LCD\_WIDTH)

{

printf\_tiny("\n\rInvalid Column number");

return;

}

address = lcd\_row\_base\_address[row] + column;

lcd\_goto\_add(address);

return;

}

/\*\*

\* @brief Writes a command to the LCD.

\*

\* @param data\_byte The command to write to the LCD.

\*/

void lcd\_cmd\_write(\_\_xdata uint8\_t data\_byte)

{

lcd\_busy\_wait();

RS\_CMD;

LCD\_WRITE;

\*(LCD\_CMD\_WRITE\_ADD) = data\_byte;

return;

}

/\*\*

\* @brief Writes a character to the LCD.

\*

\* @param data\_byte The character to write to the LCD.

\*/

void lcd\_putchar(\_\_xdata uint8\_t data\_byte)

{

lcd\_busy\_wait();

RS\_DATA;

LCD\_WRITE;

\*(LCD\_DATA\_WRITE\_ADD) = data\_byte;

lcd\_current\_column++;

if(lcd\_current\_column>=16)

{

lcd\_row\_change();

}

return;

}

/\*\*

\* @brief Writes a string to the LCD.

\*

\* @param text\_ptr Pointer to the string to be written to the LCD.

\*/

void lcd\_putstr(\_\_xdata uint8\_t\* text\_ptr)

{

while(\*text\_ptr!='\0')

{

lcd\_putchar(\*text\_ptr);

text\_ptr++;

}

return;

}

/\*\*

\* @brief Clears the LCD screen.

\*/

void lcd\_clear(void)

{

lcd\_cmd\_write(LCD\_CLRSCR);

lcd\_current\_row=0;

lcd\_current\_column=0;

return;

}

/\*\*

\* @brief Writes a character to the UART.

\*

\* @param ch The character to be written to the UART.

\* @return The written character.

\*/

int putchar (int ch)

{

while (TI==0)

{

;

}

SBUF = ch;

TI = 0;

return ch;

}

/\*\*

\* @brief Reads a character from the UART.

\*

\* @return The character read from the UART.

\*/

int getchar (void)

{

while (RI==0)

{

;

}

RI = 0;

return SBUF;

}

/\*\*

\* @brief Delays for a specified time in milliseconds.

\*

\* @param time The time to delay in milliseconds.

\*/

void delay\_ms(uint32\_t time)

{

uint32\_t i=0,j=0;

for(j=0;j<time;j++)

{

for(i=0;i<1120;i++); //1 ms

}

return;

}

/\*\*

\* @brief Fetches a number from the user.

\*

\* @param base The base of the number (e.g., 10 for decimal, 16 for hexadecimal).

\* @return The fetched number.

\*/

uint16\_t fetch\_number(uint8\_t base)

{

\_\_xdata uint8\_t scanned\_digit=0,digit\_array[20],digit\_counter=0,i=0;

\_\_xdata uint16\_t number=0;

while(scanned\_digit!=13)

{

scanned\_digit=getchar();

if(((scanned\_digit >= '0') && (scanned\_digit <= '9'))||

((scanned\_digit >= 'a') && (scanned\_digit <= 'f'))||

((scanned\_digit >= 'A') && (scanned\_digit <= 'F')))

{

putchar(scanned\_digit);

digit\_array[digit\_counter]=char\_to\_int(scanned\_digit);

digit\_counter++;

}

else if(scanned\_digit==8) //check for backspace

{

putchar(8); //print backspace

putchar(32); //print space

putchar(8); //print backspace

digit\_counter--;

}

}

for(i=0;i<digit\_counter;i++)

{

number\*= base;

number+= digit\_array[i];

}

return number;

}

/\*\*

\* @brief Converts a character to an integer.

\*

\* @param temp The character to be converted.

\* @return The integer value of the character.

\*/

uint8\_t char\_to\_int(uint8\_t temp)

{

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

{

temp-='0';

}

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

{

temp-='a';

temp+=10;

}

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

{

temp-='A';

temp+=10;

}

return temp;

}

/\*\*

\* @brief Interrupt handler for Timer 0.

\*

\* This function is called when Timer 0 overflows to generate interrupts.

\* It updates the time variable and calls the timer\_display function.

\*/

void rtc\_interrupt\_handler(void) \_\_interrupt 1

{

\_\_xdata static uint16\_t counter\_flag = 0;

TF0 = 0;

TL0 = TIMER0\_LOW;

TH0 = TIMER0\_HIGH;

if(counter\_flag == 0)

{

counter\_flag = 1;

}

else

{

time++;

timer\_display();

counter\_flag=0;

}

}

/\*\*

\* @brief Initializes Timer 0 for timekeeping.

\*/

void timer\_init(void)

{

time = 0;

IE |= 0x82;

TMOD |= 0x01;

TMOD &= 0xF1;

TL0 = TIMER0\_LOW;

TH0 = TIMER0\_HIGH;

TR0 = 1;

return;

}

/\*\*

\* @brief Displays the current time on the LCD.

\*/

void timer\_display(void)

{

\_\_xdata uint8\_t tenth\_of\_second = 0, seconds = 0, minutes = 0, temp\_lcd\_row=0,temp\_lcd\_column=0;

temp\_lcd\_column = lcd\_current\_column;

temp\_lcd\_row = lcd\_current\_row;

tenth\_of\_second = time % 10;

seconds = (time/10) % 60;

minutes = time/600;

lcd\_goto\_xy(3,8);

lcd\_print\_number(minutes,2);

lcd\_putchar(':');

lcd\_print\_number(seconds,2);

lcd\_putchar('.');

lcd\_print\_number(tenth\_of\_second,1);

lcd\_goto\_xy(temp\_lcd\_row,temp\_lcd\_column);

return;

}

/\*\*

\* @brief Stops Timer 0.

\*/

void timer\_stop(void)

{

TR0 = 0;

}

/\*\*

\* @brief Starts Timer 0.

\*/

void timer\_start(void)

{

TR0 = 1;

}

/\*\*

\* @brief Resets the time variable.

\*/

void timer\_reset(void)

{

time = 0;

}

/\*\*

\* @brief Prints a number on the LCD with a specified width.

\*

\* @param number The number to be printed.

\* @param display\_width The width to display the number.

\*/

void lcd\_print\_number(\_\_xdata int32\_t number,\_\_xdata uint8\_t display\_width)

{

\_\_xdata uint8\_t temp\_ascii\_store[10],temp\_data=0;

\_\_xdata int8\_t counter=0;

\_\_xdata uint32\_t value\_check=0;

for(counter=display\_width;counter>1;counter--)

{

lcd\_busy\_wait();

switch(counter)

{

case 2:

{

value\_check = 9;

if(number<=value\_check)

{

RS\_DATA;

LCD\_WRITE;

\*(LCD\_DATA\_WRITE\_ADD) = '0';

}

break;

}

case 3:

{

value\_check = 99;

if(number<=value\_check)

{

RS\_DATA;

LCD\_WRITE;

\*(LCD\_DATA\_WRITE\_ADD) = '0';

}

break;

}

case 4:

{

value\_check = 999;

if(number<=value\_check)

{

RS\_DATA;

LCD\_WRITE;

\*(LCD\_DATA\_WRITE\_ADD) = '0';

}

break;

}

}

}

do

{

temp\_ascii\_store[counter]='0'+ number%10;

number/=10;

counter++;

}while(number>0);

for(counter-=1;counter>0;counter--)

{

lcd\_busy\_wait();

RS\_DATA;

LCD\_WRITE;

\*(LCD\_DATA\_WRITE\_ADD) = temp\_ascii\_store[counter];

}

return;

}

void lcd\_ddram\_flush(void)

{

\_\_xdata uint8\_t count=0,i=0, temp\_storage=0, data\_byte=0;

\_\_xdata uint16\_t address=0x80;

temp\_storage = lcd\_ddram\_address\_mask;

RS\_CMD;

LCD\_WRITE;

lcd\_cmd\_write(temp\_storage);

printf\_tiny("\n\rDDRAM Hexdump");

for(count=0;count<5;count++)

{

putchar('\n');

putchar('\r');

print\_number\_hex(address,2);

putchar(':');

for(i=0;i<16;i++)

{

putchar(32);//space

RS\_DATA;

LCD\_READ;

data\_byte = \*(LCD\_DATA\_READ\_ADD);

print\_number\_hex(data\_byte,2);

}

address+=16;

}

lcd\_goto\_xy(lcd\_current\_row,lcd\_current\_column);

return;

}

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

\* @brief lcd\_cgram\_flush()

\* Function to hex dump the contents of CGRAM on UART

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

void lcd\_cgram\_flush(void)

{

\_\_xdata uint8\_t count=0, i=0, temp\_storage=0, data\_byte=0;

\_\_xdata uint16\_t address=0;

temp\_storage = lcd\_cgram\_address\_mask;

temp\_storage &= lcd\_cgram\_address\_mask\_2;

RS\_CMD;

LCD\_WRITE;

lcd\_cmd\_write(temp\_storage);

printf\_tiny("\n\rCGRAM Hexdump");

for(count=0; count<4; count++)

{

putchar('\n');

putchar('\r');

print\_number\_hex(address,2);

putchar(':');

for(i=0;i<16;i++)

{

putchar(32);//space

RS\_DATA;

LCD\_READ;

data\_byte = \*(LCD\_DATA\_READ\_ADD);

print\_number\_hex(data\_byte,2);

}

address+=16;

}

lcd\_goto\_xy(lcd\_current\_row,lcd\_current\_column);

return;

}

void print\_number\_hex(\_\_xdata uint32\_t number,\_\_xdata uint8\_t display\_width)

{

\_\_xdata uint8\_t temp\_ascii\_store[10],temp\_value=0;

\_\_xdata int8\_t counter=0;

\_\_xdata uint32\_t value\_check=0;

for(counter=display\_width;counter>1;counter--)

{

switch(counter)

{

case 4:

{

value\_check = 0xFFF;

if(number<=value\_check)

{

putchar('0');

}

break;

}

case 3:

{

value\_check = 0xFF;

if(number<=value\_check)

{

putchar('0');

}

break;

}

case 2:

{

value\_check = 0xF;

if(number<=value\_check)

{

putchar('0');

}

break;

}

}

}

do

{

temp\_ascii\_store[counter]=int\_to\_char(number%16);

number/=16;

counter++;

}while(number>0);

for(counter-=1;counter>=0;counter--)

{

putchar(temp\_ascii\_store[counter]);

}

return;

}

int8\_t int\_to\_char(int temp) /\*Function that includes switch statement for hex definition\*/

{

switch(temp) /\*switch case for defining hex characters above 9 i.e. 'A' to 'F'\*/

{ case 1:

return '1';

case 0:

return '0';

case 2:

return '2';

case 3:

return '3';

case 4:

return '4';

case 5:

return '5';

case 6:

return '6';

case 7:

return '7';

case 8:

return '8';

case 9:

return '9';

case 10:

return 'A';

case 11:

return 'B';

case 12:

return 'C';

case 13:

return 'D';

case 14:

return 'E';

case 15:

return 'F';

/\*To convert int to char\*/

}

return '0';

}