Serial Terminal

Design Documentation

William Hatch and Scott Sorensen

April 2013

Table of Contents

1. Introduction	2
2. Scope	2
3. Design Overview	2
3.1 Requirements	2
3.2 Theory of Operation	3
4. Design Detail	3
4.1 Hardware Design	3
4.2 Software Design	4
4.2.1 PS/2 Keyboard Communication	4
4.2.2 USART Communication	5
4.2.3 LCD	5
4.2.4 Control Codes and Escape Sequences	6
4.2.5 Visual Effects	6
4.2.6 Onboard DAC	6
5. Verification	6
6. Conclusion	7
Appendix A: Hardware Schematic	8
Appendix B: Software Flow Charts	g
Appendix C: Verification Pictures	11
Appendix D: Source Code	12

1 Introduction

This document describes the design of a serial computer terminal. The microcontroller reads input from a keyboard connected to the PS/2 port, outputs the input received via the RS-232 to the host computer, and then receives data from the host computer to output to the LCD screen.

2 Scope

In this document is shown how to connect a microcontroller and its components to a personal computer and the software design necessary for the communication between all of the devices.

This document does not cover the host computer's software nor the mechanical design of the components.

3 Design Overview

3.1 Requirements

- 1. The system will receive data from a PS/2 keyboard.
- 2. The system will be able to translate the key codes from the PS/2 keyboard to usable ASCII characters for the host computer.
- 3. The system will transmit and receive data to the host computer via USART
- 4. The system will be able to handle color codes and ANSI escape sequences.
- 5. The system will print the correct characters and colors to the LCD screen

6. The system will print a cursor to the LCD screen

3.2 Theory of Operation

Serial terminals are used to transfer data between a computer system and its users. The user types at the terminal's keyboard, and the ascii codes are sent to the computer to be interpreted. The computer outputs ascii letters and control sequences to the terminal, which are then displayed, sounded as terminal beep alerts, or used to alter the state of the display.

There are many different terminal designs used on various systems ranging from UNIX mainframes to DOS PCs. The capabilities and features of serial terminals vary widely, and the control code standards are complicated. The terminal in this design has a large subset of capabilities of the virtual Linux console, and is largely compatible with vt100 series terminals. Control codes supported include cursor manipulation, visual effects, terminal state saving, and basic line editing and output codes.

4. Design Details

4.1 Hardware

The following hardware components are used for this design:

- 1. STM32F103RC Micro-controller
- 2. SSD 1289 LCD Screen
- 3. RS-232 Module
- 4. Micro-controller Onboard DAC
- 5. Speaker
- 6. PS/2 Keyboard
- 7. Serial Cable
- 8. Host Computing System

A schematic of the design is included in Appendix A.

4.2 Software Design

The software for the design consists of the configuration of the devices used, as well as the handling of receiving, transmitting, and interpreting data between all of the devices.

The main function calls various initialization functions, along with a function to clear the screen. It then goes into an infinite while loop and functions for handling the usart data, handling the ps2 data, and refreshing the screen are continuously called.

A software flow chart is included in Appendix B.

4.2.1 PS/2 Keyboard Communication

The PS/2 port on the micro-controller shares pins with PC3 and PC4, which are the clock and data for the PS/2 Keyboard, respectively. The pins are configured as pull-up/pull-down input as the micro-controller will not be sending data to the keyboard. The external interrupt is enabled for the pins as well as the AFIO clock.

The PS/2 interrupt has various static variables to handle the start bit, the stop bit, and the data bits. The interrupt has an infinite while loop which checks which bit the has been received, and exits out of the loop depending on which bits have been received so far. Once a start bit and 8 data bits have been received, the data received is interpreted to be a special or regular byte. If it isn't a special byte, the code is translated to an ASCII code via a map so the host computer can read the data properly. It is then put into a data buffer. There are various special bytes that need to be handled differently: the shift key code, the ctrl key code, the alt key code, and the code signifying an up keystroke. None of these are put into the data buffer. For the shift and ctrl codes, modified maps are used to pass different ASCII values. For the alt key code, the ASCII escape code is put into the buffer to precede whatever key is being pressed down with it. If the byte is 0xF0, it signifies an upstroke of a key, and a variable is set to true to handle the next

byte read. If the byte following an upstroke is shift or ctrl, then the data being translated goes back to using the default map, and for alt the escape code is no longer sent.

4.2.2 USART2 Communication

The data from the PS/2 keyboard is put into a buffer which is then sent transmitted via the USART2. In the USART2 initialize function the AFIO clock and alternate function for GPIOA are set, along with an interrupt enable to be triggered when the rx (receive) buffer is no longer empty. The tx function transmits the data from the PS/2 buffer to the host computer. The USART2 interrupt handles the data received from the host computer and puts it into a buffer which is processed to handle control codes and write the necessary data to the LCD screen.

4.2.3 LCD

The LCD screen is configured to use the correct pins from GPIOA and GPIOC, and it is initialized to have a horizontal configuration. Functions are written for clearing the screen, writing to the screen, setting the position of the cursor, writing the commands and data for the LCD, and drawing lines of characters to the screen.

An internal buffer of screen data is kept, with the characters to be drawn at each row and column, along with their individual visual effects. When they are printed their row and column positions are translated into x and y coordinates on the screen, and a bitmap is retrieved and drawn for each ascii character. The printing function displays the bitmap with appropriate foreground and background colors, as well as other effects such as underlining, one line of pixels at a time to the 8x16 character space. As the screen is periodically refreshed, characters set to blink are toggled on and off to produce the effect.

4.2.4 Control Codes and Escape Sequences

While most ascii characters received by the terminal are drawn to the screen, some must be handled as control codes or escape sequences that perform special functions. There is a state machine function that handles all the characters

received from the host computer. Basic control codes are a single character and have simple functions, such as a carriage return, line feed, or bell (alert) character. Simple escape sequences start with an escape (ESC) character, then a character indicating a function, usually slightly more complex than the single character control codes. As terminal capabilities rapidly expanded at the time of creating these standards, some of them became somewhat complex, and to handle the more advanced features of graphical terminals, the CSI, or Control Sequence Inductor code standards were made. CSI codes begin with either a single CSI control character or an escape character followed by a left bracket (ESC[). They then have a sequence of ascii decimal numbers separated with semicolons. Finally they are terminated by a single character. The ending character determines how the numbers are interpreted and handled. This provides complicated functions for adding color and other visual effects, arbitrary cursor movement, and other advanced functions.

4.2.5 Visual Effects

The visual effects used for the terminal include adding background color, foreground color, underline for the text, bold (brightly colored) text, and blinking text. The various CSI codes are passed from the host computer and each code is interpreted to display correctly to the screen.

4.2.6 Onboard DAC

The DAC uses a timer interrupt to play sound according to a 40 entry wave table whenever a bell character is received from the host.

5. Verification

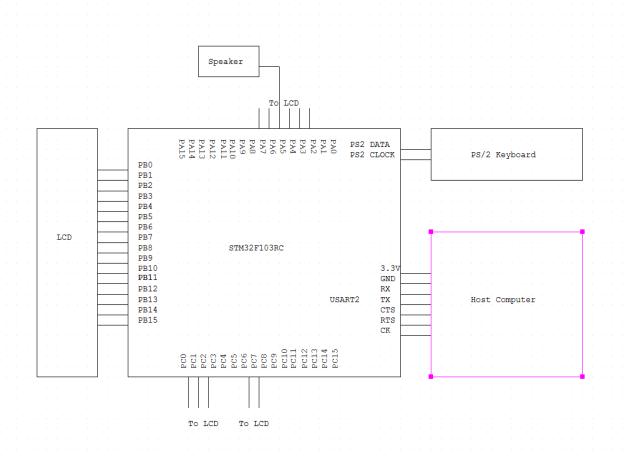
The system was hand tested to see that it received, transferred, and displayed data properly. A color comparison script was used to display all possible color combinations, and an effects script was written to test blinking, underlining, and reverse-video effects. Cursor manipulation was tested by viewing compatibility with programs such as Vim which use Curses libraries to manipulate the cursor

and modify arbitrary sections of the display. The design is verified to work properly, although many programs are not designed to handle a display size as small as 15 rows x 40 columns, and such programs display poorly. Programs written to scale to very small displays display perfectly, however.

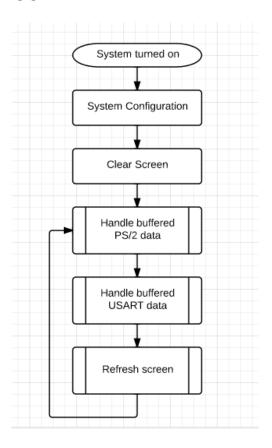
6. Conclusion

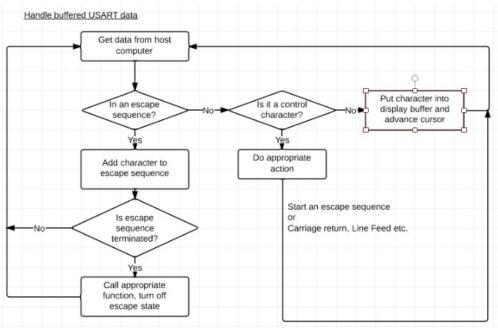
Serial terminals are an important part of computer history, and remain in use even today for various server administration tasks. The terminal outlined in this document is a functional, small, and light design which would be useful. The main drawback is the restrictive screen size. If a larger LCD screen were used to increase the display area it would be a useful terminal for actual use by people who use serial terminals today. For most practical applications, however, a software terminal emulator is recommended.

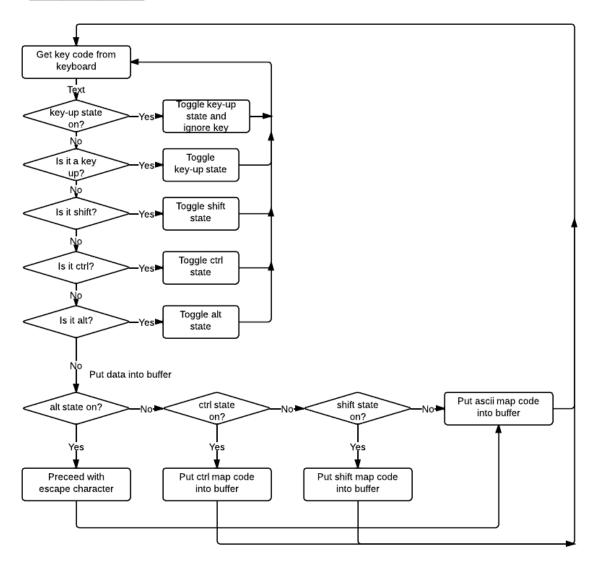
Appendix A: Hardware Schematic



Appendix B: Software Flow Charts

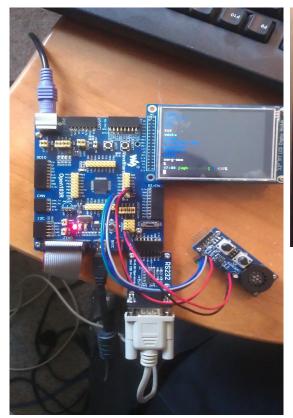






Appendix C: Verification Photographs

View of hardware setup, connected to Linux computer, and color test script output.





Connected to online multiplayer text-based game.

```
12631/12237 183 7783/6722 115 4520 188 9
40000 74317386 who

FUL ]

Turtle Hermit 329 Pal Baathran Faeinth
, Sun's Paladin \|/PHOENIX\|/

Vou can currently see 1 player.
Max number this reboot 3 players.

12631/12237 183 7783/6722 115 4528 186 9
40000 74317386
```

Appendix D: Source Code

```
2
        ECE 3710 Lab 6: ascii.h
 3
 4
 5
 6
 7
    #ifndef __ASCII_H
 8
    #define __ASCII_H
 9
10
    #include <string.h>
11
12
    void get ascii( unsigned char *, unsigned char );
13
14
    #endif
15
    /* END OF FILE */
16
17
    //code_to_ascii.h
18
19
    #ifndef CODE_TO_ASCII
20
    #define CODE_TO_ASCII
21
22
    void scan_code_init(void);
23
24
    #endif
25
    // dac.h
26
27
    void DAC_init(void);
28
    void DAC_beep(void);
29
    //
30
    //
        lcd.h
31
        ECE 3710 Microcontroller H&S
32
        Utah State University
33
34
    #ifndef __LCD_H
#define __LCD_H
35
36
37
    #include "stm32f10x.h"
38
39
    #define DISP_ORIENTATION 90
40
41
42
    #if ( DISP_ORIENTATION == 90 ) || ( DISP_ORIENTATION == 270 )
43
    #define MAX_X 320
44
    #define MAX_Y 240
45
    //#define CHARS_HORIZ_ON_Y 30
46
    #define CHARS_HORIZ_ON_Y 28
47
    #define CHARS_VERT_ON_Y 20
48
    #define CHARS HORIZ ON X 40
49
50
    #define CHARS_VERT_ON_X 15
51
    #elif ( DISP_ORIENTATION == 0 ) || ( DISP_ORIENTATION == 180 )
52
53
    #define MAX X 240
54
    #define MAX Y 320
55
56
    #define CHARS HORIZ ON Y 40
57
    #define CHARS_VERT_ON_Y 15
58
59
    #endif
60
61
62
    /* some LCD colors */
63
    #define White
                            0xFFFF
64
    #define Black
                            0x0000
65
    #define Grey
                            0xF7DE
66
    #define Blue
                            0x001F
67
    #define Blue2
                            0x051F
68
    #define Red
                            0xF800
69
    #define Magenta
                            0xF81F
70
    #define Green
                            0×07E0
```

```
#define Cyan
                             0x7FFF
     #define Yellow
72
                             0 \times FFF0
73
74
75
     #define TermBlack Black
     #define TermBlackBright 0x52aa
76
     #define TermRed 0xa800
77
     #define TermRedBright Oxfaaa
78
     #define TermGreen 0x540
79
     #define TermGreenBright 0x57ea
80
     #define TermBrown 0xaaa0
81
82
     #define TermBrownBright Oxffea
     #define TermBlue 0x15
83
     #define TermBlueBright 0x52bf
84
85
     #define TermMagenta 0xa815
     #define TermMagentaBright 0xfabf
86
87
     #define TermCyan 0x555
     #define TermCyanBright 0x57ff
88
     #define TermWhite 0xa554
89
     #define TermWhiteBright White
90
     #define TermDefault TermWhite
91
92
     #define TermDefaultBright TermWhiteBright
93
94
95
     void LCD_Config(void);
     void LCD_Initialization(void);
96
97
     void LCD_Clear( unsigned short Color );
98
99
     void LCD WriteIndex( unsigned short index );
100
     void LCD WriteData( unsigned short data );
101
     void LCD Write Generic(unsigned short toWrite, unsigned short dataBool);
102
     void LCD WriteReg( unsigned short LCD Reg, unsigned short LCD RegValue );
103
     void LCD_SetCursor( unsigned short x, unsigned int y );
104
     void delay_ms( unsigned int ms );
105
106
107
     void LCD_DrawSquare( unsigned short x, unsigned short y, unsigned short h, unsigned short w,
     unsigned short color );
108
109
     void LCD DrawCharacterOnY (unsigned short x, unsigned short y, unsigned short fgColor, unsigned
110
     short bgColor, unsigned char symbol);
     void LCD WriteCharactersOnY (unsigned short x, unsigned short y, unsigned short fgColor, unsigned
111
     short bgColor, char* words, int maxLength);
     void LCD_WriteLinesOnY(unsigned short x, unsigned short fgColor, unsigned short bgColor, char*
112
     words, char drawToLineEnd);
113
114
     void LCD DrawChar rc (unsigned int row, unsigned int col, unsigned short fgColor, unsigned short
     bgColor, unsigned char symbol, unsigned char underline);
115
116
     #endif
117
118
     // END OF FILE
119
     //ps2 over gpioc.h
     #ifndef PS2 OVER GPIOC
120
121
     #define PS2_OVER_GPIOC
122
     #define PS2_DATA_SIZE 1000
123
124
125
126
127
     void ps2 over gpioc init(void);
     //void ps2 dump data over usart2(void);
128
129
     int ps2 memcpy(unsigned char * dst);
130
     void ps2 insert to buffer(char *insert, int size);
131
132
133
     #endif
134
```

135

```
136
         // terminal.h
137
         #ifndef __TERMINAL
138
         #define __TERMINAL
139
         #include "lcd.h"
140
141
142
         #define ROWS CHARS_VERT_ON_X // Num lines
143
         #define COLS CHARS HORIZ ON X // Num columns
144
145
146
         void flushScreen(void);
147
148
         void bufClear(void);
         void handleAscii(unsigned char *buf, int bytes);
149
150
151
         #endif
152
153
         // usart2.h
154
155
         #ifndef __USART2
         #define _
156
                          USART2
157
         #include "stm32f10x.h"
158
159
         #define USART2_DATA_SIZE 1000
160
161
162
         void usart2_init(void);
163
164
         void usart2 tx(unsigned char byte);
165
         int usart2 memcpy(unsigned char * dst);
166
167
         #endif
168
                ECE 3710 Lab 6: ascii.c
169
170
171
172
         #include "ascii.h"
173
174
         static const unsigned char ascii[95][16] =
175
              176
                                                                                                                                                                      0.10
                                                                                                                                                                 /*
                177
                                                                                                                                                                      0.00
178
                {0x00,0x00,0x00,0x36,0x36,0x7F,0x36,0x36,0x36,0x7F,0x36,0x36,0x00,0x00,0x00,0x00},
179
                                                                                                                                                                      "$"
                {0x00,0x18,0x18,0x3C,0x66,0x60,0x30,0x18,0x0C,0x06,0x66,0x3C,0x18,0x18,0x00,0x00},
180
                                                                                                                                                                      11%11
                {0x00,0x00,0x70,0xD8,0xDA,0x76,0x0C,0x18,0x30,0x6E,0x5B,0x1B,0x0E,0x00,0x00,0x00},
181
                                                                                                                                                                      "&"
182
                {0x00,0x00,0x00,0x38,0x6C,0x6C,0x38,0x60,0x6F,0x66,0x66,0x3B,0x00,0x00,0x00,0x00},
                                                                                                                                                                      0.10
183
                "("
184
                {0x00,0x00,0x00,0x0C,0x18,0x18,0x30,0x30,0x30,0x30,0x30,0x18,0x18,0x0C,0x00,0x00},
                                                                                                                                                                      " j "
185
                {0x00,0x00,0x00,0x30,0x18,0x18,0x0C,0x0C,0x0C,0x0C,0x0C,0x18,0x18,0x30,0x00,0x00},
                                                                                                                                                                      "*"
186
                {0x00,0x00,0x00,0x00,0x00,0x36,0x1C,0x7F,0x1C,0x36,0x00,0x00,0x00,0x00,0x00,0x00},
                                                                                                                                                                      ^{\rm n}+^{\rm n}
187
                /* ","
188
                -0 \stackrel{?}{=} 0
189
                0.0
                190
                                                                                                                                                                     "/" */
191
                {0x00,0x00,0x00,0x06,0x06,0x0C,0x0C,0x18,0x18,0x30,0x30,0x60,0x60,0x00,0x00,0x00},
                                                                                                                                                                      "0" */
                {0x00,0x00,0x00,0x1E,0x33,0x37,0x37,0x33,0x3B,0x3B,0x33,0x1E,0x00,0x00,0x00,0x00},
192
                 \{0 \times 00, 0 \times 00, 0 \times 00, 0 \times 00, 0 \times 10, 0 \times 70, 0 \times 00, 0 
                                                                                                                                                                      "1" */
193
                                                                                                                                                                      "2" */
194
                {0x00,0x00,0x00,0x3C,0x66,0x66,0x06,0x0C,0x18,0x30,0x60,0x7E,0x00,0x00,0x00,0x00},
                                                                                                                                                                     "3"
195
                {0x00,0x00,0x00,0x3C,0x66,0x66,0x66,0x1C,0x06,0x66,0x66,0x3C,0x00,0x00,0x00,0x00},
                                                                                                                                                                     "4"
                196
                                                                                                                                                                      "5"
197
                {0x00,0x00,0x00,0x7E,0x60,0x60,0x60,0x7C,0x06,0x06,0x0C,0x78,0x00,0x00,0x00,0x00},
                                                                                                                                                                      "6"
                {0x00,0x00,0x00,0x1C,0x18,0x30,0x7C,0x66,0x66,0x66,0x66,0x3C,0x00,0x00,0x00,0x00},
198
                                                                                                                                                                      "7"
                {0x00,0x00,0x00,0x7E,0x06,0x0C,0x0C,0x18,0x18,0x30,0x30,0x30,0x00,0x00,0x00,0x00},
199
                                                                                                                                                                      "8"
200
                {0x00,0x00,0x00,0x3C,0x66,0x66,0x76,0x3C,0x6E,0x66,0x66,0x3C,0x00,0x00,0x00,0x00},
                                                                                                                                                                 /*
                                                                                                                                                                     "9" */
201
                {0x00,0x00,0x00,0x3C,0x66,0x66,0x66,0x66,0x3E,0x0C,0x18,0x38,0x00,0x00,0x00,0x00},
                                                                                                                                                                 /* ":" */
202
                /* ";" */
203
                /* "<"
                \{0\times00,0\times00,0\times00,0\times06,0\times00,0\times18,0\times30,0\times60,0\times30,0\times18,0\times00,0\times06,0\times00,0\times00,0\times00,0\times00\}
204
                                                                                                                                                                      0-0
205
```

```
206
      {0x00,0x00,0x00,0x60,0x30,0x18,0x0C,0x06,0x0C,0x18,0x30,0x60,0x00,0x00,0x00,0x00},
                                                             /*
                                                               "?"
207
      /*
                                                               "@"
208
      {0x00,0x00,0x00,0x7E,0xC3,0xC3,0xCF,0xDB,0xDB,0xCF,0xC0,0x7F,0x00,0x00,0x00,0x00},
                                                             /*
                                                               "Ā"
209
      /*
                                                               "B"
210
      {0x00,0x00,0x00,0x7C,0x66,0x66,0x66,0x7C,0x66,0x66,0x66,0x7C,0x00,0x00,0x00,0x00},
                                                             /*
      \{0\times00,0\times00,0\times00,0\times3C,0\times66,0\times66,0\times60,0\times60,0\times60,0\times66,0\times66,0\times3C,0\times00,0\times00,0\times00,0\times00\}
                                                               "C"
211
                                                             /*
                                                               "D"
      {0x00,0x00,0x00,0x78,0x6C,0x66,0x66,0x66,0x66,0x66,0x6C,0x78,0x00,0x00,0x00,0x00},
212
                                                             /*
                                                               "E"
213
      {0x00,0x00,0x00,0x7E,0x60,0x60,0x60,0x7C,0x60,0x60,0x60,0x7E,0x00,0x00,0x00,0x00},
                                                             /*
                                                               "F"
      214
                                                             /*
                                                               "G"
      {0x00,0x00,0x00,0x3C,0x66,0x66,0x60,0x60,0x6E,0x66,0x66,0x3E,0x00,0x00,0x00,0x00},
215
                                                             /*
      "H"
216
                                                             /*
                                                               "I"
      217
                                                             /*
                                                               "J"
218
      /*
219
      /*
      "["
220
                                                             /*
      {0x00,0x00,0x00,0x63,0x63,0x77,0x6B,0x6B,0x6B,0x63,0x63,0x63,0x00,0x00,0x00,0x00},
                                                               "M"
221
                                                             /*
222
      {0x00,0x00,0x00,0x63,0x63,0x73,0x7B,0x6F,0x67,0x63,0x63,0x63,0x00,0x00,0x00,0x00},
                                                               "N"
                                                             /*
                                                               "0"
223
      {0x00,0x00,0x00,0x3C,0x66,0x66,0x66,0x66,0x66,0x66,0x66,0x3C,0x00,0x00,0x00,0x00},
                                                             /*
                                                               "P"
      224
                                                             /*
                                                               "0"
225
      {0x00,0x00,0x00,0x3C,0x66,0x66,0x66,0x66,0x66,0x66,0x66,0x3C,0x0C,0x06,0x00,0x00},
                                                             /*
                                                               "R"
226
      227
      {0x00,0x00,0x00,0x3C,0x66,0x60,0x30,0x18,0x0C,0x06,0x66,0x3C,0x00,0x00,0x00,0x00},
                                                             /*
                                                               "S"
                                                             /*
228
      /*
229
      "V"
                                                             /*
230
      "W"
                                                             /*
231
      232
      \{0\times00,0\times00,0\times00,0\times66,0\times66,0\times34,0\times18,0\times18,0\times2C,0\times66,0\times66,0\times66,0\times00,0\times00,0\times00,0\times00\}
                                                             /*
                                                               "Y"
                                                             /*
233
      "Z"
                                                             /*
234
      {0x00,0x00,0x00,0x7E,0x06,0x06,0x0C,0x18,0x30,0x60,0x60,0x7E,0x00,0x00,0x00,0x00},
235
      236
      {0x00,0x00,0x00,0x60,0x60,0x30,0x30,0x18,0x18,0x0C,0x0C,0x06,0x06,0x00,0x00,0x00},
237
      238
      0 = 0
239
      \pi T \pi
240
      "a"
241
      {0x00,0x00,0x00,0x00,0x00,0x3C,0x06,0x06,0x3E,0x66,0x66,0x3E,0x00,0x00,0x00,0x00},
                                                               "b"
242
      {0x00,0x00,0x00,0x60,0x60,0x7C,0x66,0x66,0x66,0x66,0x66,0x7C,0x00,0x00,0x00,0x00},
                                                               "c"
243
      {0x00,0x00,0x00,0x00,0x00,0x3C,0x66,0x60,0x60,0x60,0x66,0x3C,0x00,0x00,0x00,0x00},
                                                               "d"
244
      {0x00,0x00,0x00,0x06,0x06,0x3E,0x66,0x66,0x66,0x66,0x66,0x3E,0x00,0x00,0x00,0x00},
                                                               "e"
      245
                                                               "f"
246
      "g"
247
      {0x00,0x00,0x00,0x00,0x00,0x3E,0x66,0x66,0x66,0x66,0x66,0x66,0x3E,0x06,0x06,0x7C,0x00},
                                                               "ĥ"
248
      {0x00,0x00,0x00,0x60,0x60,0x7C,0x66,0x66,0x66,0x66,0x66,0x66,0x00,0x00,0x00,0x00},
                                                               "i"
249
      {0x00,0x00,0x18,0x18,0x00,0x78,0x18,0x18,0x18,0x18,0x18,0x7E,0x00,0x00,0x00,0x00},
                                                               "i"
250
      "Ř"
251
      {0x00,0x00,0x00,0x60,0x60,0x66,0x66,0x6C,0x78,0x6C,0x66,0x66,0x00,0x00,0x00,0x00},
                                                               "1"
252
      "m"
253
      {0x00,0x00,0x00,0x00,0x00,0x7E,0x6B,0x6B,0x6B,0x6B,0x6B,0x63,0x00,0x00,0x00,0x00},
      {0x00,0x00,0x00,0x00,0x00,0x7C,0x66,0x66,0x66,0x66,0x66,0x66,0x66,0x00,0x00,0x00,0x00},
                                                               "n"
254
255
                                                               "0"
      256
      "p"
                                                               "q"
257
      {0x00,0x00,0x00,0x00,0x00,0x3E,0x66,0x66,0x66,0x66,0x66,0x3E,0x06,0x06,0x06,0x00},
                                                               "r"
258
      "s"
259
      {0x00,0x00,0x00,0x00,0x00,0x3E,0x60,0x60,0x3C,0x06,0x06,0x7C,0x00,0x00,0x00,0x00},
                                                               "t"
260
      {0x00,0x00,0x00,0x30,0x30,0x7E,0x30,0x30,0x30,0x30,0x30,0x1E,0x00,0x00,0x00,0x00},
                                                               "u"
261
      "v"
262
      "w"
263
      \{0\times00,0\times00,0\times00,0\times00,0\times00,0\times63,0\times6B,0\times6B,0\times6B,0\times6B,0\times36,0\times36,0\times36,0\times00,0\times00,0\times00,0\times00\}
                                                               "x"
264
      "y"
265
      "Z"
       \{0 \times 00, 0 \times 00, 0 \times 00, 0 \times 00, 0 \times 7E, 0 \times 06, 0 \times 0C, 0 \times 18, 0 \times 30, 0 \times 60, 0 \times 7E, 0 \times 00, 0 \times 00, 0 \times 00, 0 \times 00\}, 
266
                                                             /*
                                                               "{" */
267
      \{0\times00,0\times00,0\times00,0\times00,0\times18,0\times18,0\times18,0\times30,0\times60,0\times30,0\times18,0\times18,0\times18,0\times00,0\times00,0\times00\}
                                                             /*
                                                               "|"
268
      /*
                                                               "}"
269
      \{0\times00,0\times00,0\times00,0\times30,0\times18,0\times18,0\times18,0\times00,0\times06,0\times00,0\times18,0\times18,0\times18,0\times30,0\times00,0\times00\}
270
      271
   void get_ascii( unsigned char* buffer, unsigned char chr )
272
273
   {
      memcpy( buffer, ascii[chr-32], 16 );
274
   }
275
```

```
276
      /* END OF FILE */
277
278
     //code_to_ascii.c
279
     #include "code_to_ascii.h"
280
281
282
     unsigned char map[0 \times 100];
283
     unsigned char shift_map[0x100];
284
     unsigned char ctl_map[0x100];
285
286
     void scan_code_init(void)
287
288
              int i = 0;
289
              for (i=0; i<0\times100; i++)
290
               {
291
                       map[i] = 0;
292
                       shift_map[i] = 0;
293
                       ctl_map[i] = 0;
294
              }
295
296
              map[0x1C]='a';
297
              map[0x32]='b';
298
              map [0x21] = 'c';
299
              map[0x23]='d';
              map[0x24]='e';
300
              map[0x2B]='f';
301
              map[0x34]='g';
302
              map[0x33]='h';
303
              map[0x43]='i';
304
              map[0x3B]='j
305
306
              map[0x42]='k';
307
              map[0x4B]='l';
308
              map[0x3A]='m';
309
              map [0x31] = 'n';
310
              map [0x44] = 'o';
311
              map[0x4D]='p';
312
              map[0x15] = 'q';
              map[0x2D]='
313
314
              map[0x1B]='
              map[0x2C]='t
315
              map[0x3C]='u';
316
              map[0x2A]='v'
317
318
              map[0x1D]='w';
              map[0x22]='
319
              map[0x35] = '
320
              map[0x1A]='
321
              map[0x45]='0';
322
323
              map[0x16]='1';
324
              map[0x1E]='2';
              map[0x26]='3';
325
326
              map[0x25]='4';
              map[0x2E]='5';
327
              map[0x36]='6';
328
              map[0x3D]='7';
329
              map[0x3E]='8';
330
331
              map[0x46]='9';
332
333
              map[0x54]='[';
334
              map[0x0E]='
              map[0x4E] = ' - ';
335
336
              map[0x55] = '=';
              map[0x5D]='\\';
337
              map[0x52]='\';
338
              map [0x49] = '.';
339
              map[0x41]=',';
340
341
              map[0x4C]=';';
342
              map[0x5B]=']';
343
              map[0x4A]='/';
344
              map[0x29]=0x20;//SPACE
345
```

```
346
               map[0x5A]=0x0D;//ENTER
347
               map[0x76]=0x1B;//ESC
348
               map[0x66] = 0x08; //BKSP
349
               map[0x0D]=0x09;//TAB
350
               map[0x7C]='*';//'KP *';
351
               map[0x79]='+';//'KP +';
352
              map[0x7B] = ' - '; // 'KP
353
              map[0x71]='.';//'KP
354
              map[0 \times 70] = '0'; // 'KP 0';
355
              map[0x69]='1';//'KP 1';
356
357
              map[0x72]='2';//'KP 2';
              map[0x7A]='3';//'KP 3';
358
              map[0x6B]='4';//'KP 4';
359
360
               map[0x73]='5';//'KP 5';
361
               map[0x74]='6';//'KP 6';
               map[0x6C]='7';//'KP 7';
362
               map[0x75]='8';//'KP 8';
363
               map[0x7D] = '9'; // 'KP 9';
364
365
366
367
               shift map[0 \times 1C]='A';
368
               shift_map[0x32]='B';
369
               shift_map[0x21]='C';
370
               shift_map[0x23]='D';
371
               shift_map[0x24]='E';
372
               shift_map[0x2B]='F';
373
               shift_map[0x34]='G';
374
               shift map[0x33]='H';
375
               shift map[0x43]='I
376
               shift map[0x3B]='J
377
               shift map[0x42]='K';
378
               shift_map[0x4B]='L
379
               shift_map[0x3A]='M'
380
               shift_map[0x31]='N
381
               shift_map[0x44]='0
382
               shift_map[0x4D]='P
               shift_map[0x15]='0
383
               shift_map[0x2D] = 'R
384
               shift_map[0x1B]='
385
386
               shift_map[0x2C]='
387
               shift_map[0x3C]='U';
               shift_map[0x2A]='V
388
               shift map[0\times1D]='W'
389
390
               shift_map[0x22]='X
391
               shift_map[0x35]=
               shift_map[0x1A]='Z'
392
393
               shift_map[0x45]=
394
               shift_map[0x16]='!'
395
               shift_map[0x1E]='0';
396
               shift map[0 \times 26]='#';
397
               shift map[0 \times 25]='$';
398
               shift map[0 \times 2E]='%';
               shift_map[0x36]='^';
399
               shift_map[0x3D]='&';
400
401
               shift_map[0x3E]='*';
               shift_map[0\times46]='(';
402
403
404
               shift map[0x54]='\{';
               shift_map[0\times0E]='\sim';
405
               shift_map[0x4E]='_';
406
407
               shift map[0 \times 55]='+';
               shift_map[0x5D]='|';
408
               shift_map[0x52]='"'
409
410
               shift map[0x49]='>';
411
               shift map[0\times41]='<';
               shift_{map}[0x4C]=':';
412
               shift_{map}[0x5B]='\}';
413
414
               shift_map[0x4A]='?';
415
```

```
416
              shift map[0x29]=0x20;//SPACE
417
              shift_map[0x5A]='\n';//ENTER - set to newline for debugging purposes
418
              shift_map[0x76]=0x1B;//ESC
419
              shift_map[0x66] = 0x08; //BKSP
420
              shift_map[0x0D]=0x09;//TAB
421
              shift map[0x7C]='*';//'KP *';
422
423
              shift map[0x79]='+';//'KP +';
              shift map[0 \times 7B]='-';//'KP -'
424
              shift_{map}[0x71]='.';//'KP
425
              shift map[0 \times 70]='0';//'KP 0';
426
              shift map [0x69] = '1'; // 'KP 1';
427
              shift_map[0x72]='2';//'KP 2';
428
              shift_map[0x7A]='3';//'KP 3';
429
              shift_map[0x6B] = '4'; // 'KP 4';
430
              shift_map[0x73]='5';//'KP 5';
431
432
              shift_map[0x74]='6';//'KP 6';
433
              shift_map[0x6C]='7';//'KP 7';
434
              shift_map[0x75]='8';//'KP 8';
435
              shift_map[0x7D] = '9'; // 'KP 9';
436
437
              ctl_map[0x1e] = 0x00; //^@
438
              ctl_map[0x36] = 0x1e; //^^
              ctl_map[0x3e] = 0x7f; //^?
439
440
              ctl_map[0x4e] = 0x1f; //^
              ctl_map[0x15] = 0x11; //^{\overline{Q}}
441
442
              ctl_map[0x1d] = 0x17; //^W
              ctl_map[0x24] = 0x05; //^E
443
              ctl_map[0x2d] = 0x12; //^R
444
445
              ctl map[0x2c] = 0x14; //^T
446
              ctl_map[0x35] = 0x19; //^Y
447
              ctl_map[0x3c] = 0x15; //^U
              ctl_map[0x44] = 0x0f; //^0
448
449
              ctl_map[0x4d] = 0x10; //^P
450
              ctl_map[0x54] = 0x1b; //^[
              ctl_map[0x5b] = 0x1d; //^]
451
452
              ctl_map[0x5d] = 0x1c; //^\
              ctl_map[0x1c] = 0x01; //^A
453
              ctl_map[0x1b] = 0x13; //^S
454
              ctl_map[0x2b] = 0x06; //^F
455
              ctl_map[0x34] = 0x07; //^G
456
              ctl map[0x33] = 0x08; //^H
457
              ctl_map[0x3b] = 0x0a; //^J
458
              ctl_map[0x42] = 0x0b; //^K
459
              ctl_map[0x4b] = 0x0c; //^L
460
461
              ctl_map[0x1a] = 0x1a; //^Z
              ctl_map[0x22] = 0x18; //^X
462
              ctl_map[0x21] = 0x03; //^C
463
              ctl_map[0x2a] = 0x16; //^V
464
              ctl_map[0x32] = 0x02; //^B
465
466
              ctl_map[0x31] = 0x0e; //^N
              ctl_map[0x3a] = 0x0d; //^M
467
              ctl map[0x23] = 0x04; //^D
468
              ctl_map[0x43] = 0x09; //^I
469
470
471
     // dac.c
472
473
474
     #include "stm32f10x.h"
475
     #include "dac.h"
476
477
478
      static int wave[40] = {
479
              2047, 2367, 2679, 2976, 3250, 3494, 3703, 3870, 3993, 4068, 4094, 4068, 3993, 3870, 3703, 3494, 3250, 2976,
480
      2679,2367,2047,1726,1414,1117,843,599,390,223,100,25,0,25,100,223,390,599,843,1117,1414,1726
481
482
     int beepCount = 0;
483
484
485
     void DAC_beep(void)
```

```
486
487
              beepCount = 100;
488
     }
489
490
     void Tim3_init()
491
492
     {
493
              RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;
              NVIC->ISER[0] = NVIC_ISER_SETENA_29;
494
              NVIC->IP[7] = 0; // Highest priority!
495
              TIM3->CR1 = 0 \times 94; // Count down, restart automatically, only update on under/overflow
496
              TIM3->DIER = 1; // enable interrupt on timer finish
497
498
              TIM3->ARR = 0xFFFF; // 8000 is 1ms on 8Mhz clock
499
500
              TIM3->CR1 \mid = 1; // enable timer.
501
502
     }
503
504
     void TIM3 IRQHandler()
505
     // Interrupt on ISER[0]0x20000000
506
     // Output new DAC value along the wave form
507
              if (beepCount)
508
509
              {
510
                      static int count = 0;
                      DAC->DHR12R2 = wave[count++];
511
512
                      //DAC->SWTRIGR = 2;
513
                      if (count == 40)
514
                               count = 0;
515
                      beepCount - - ;
516
              }
517
518
              // reset interrupt pending in NVIC
519
              TIM3->SR &= 0xFFFFFFFE;
520
              NVIC->ICPR[0] = NVIC_ICPR_CLRPEND_29;
521
522
     }
523
     void DAC_init()
524
525
              // Enable gpio clock
526
              RCC->APB1ENR |= RCC_APB1ENR_DACEN;
527
528
              // Setup GPIOS
              // DAC OUT2 is PA5
529
              RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;
530
              GPIOA->CRL = (GPIOA->CRL & 0xFF0FFFFF) | 0x00B00000; // PC5 set to output AF push-pull
531
532
533
              // Write configs
534
              DAC->CR = 0\times010000; // enable DAC channel 2, turn off buffering
535
              //DAC->CR |= 0x3C0000; // enable triggers on software trigger
536
537
              Tim3_init();
538
     }
539
540
541
     //
542
         lcd.c
543
         ECE 3710 Microcontroller H&S
544
         Utah State University
545
     //
         Written by Kelly Hathaway
546
         And William Hatch and Scott Sorensen
     //
547
548
549
     #include "stm32f10x.h"
     #include "lcd.h"
550
551
     #include "ascii.h"
552
                                  // Pin 1
553
     #define WR low Pin
                          0×0002
     #define RD_low_Pin 0x0004
                                  // Pin 2
554
555
     #define CS_low_Pin 0x0040 // Pin 6
```

```
556
      #define DC Pin
                             0 \times 0080 // Pin 7
557
558
559
          configuration of the LCD port pins
560
      void LCD_Config(void)
561
562
          unsigned int config_temp;
563
          RCC->APB2ENR \mid = 0 \times 1D;
564
                                          // Enable port A, B, and C
565
          config temp = GPIOA->CRL; // Pin A.3 for Back light
566
          config_temp &= \sim 0 \times 00000 F0000;
567
          config_temp \mid = 0 \times 00003000;
568
          GPIOA->CRL
569
                         = config_temp;
570
571
          GPIOB->CRL
                         = 0x33333333; // Port B for Data[15:0] pins
572
          GPIOB->CRH
                         = 0x333333333;
573
          config_temp = GPIOC->CRL; // PC.0(LCD RST), PC.1(WR), PC.2(RD) , PC.6(CS), PC.7(DC)
574
575
          config_temp &= ~0xFF000FFF;
576
          config_temp \mid = 0 \times 33000333;
577
          GPIOC->CRL
                         = config temp;
578
      }
579
580
      void LCD_Initialization(void)
581
582
          unsigned int config temp;
583
584
          LCD_Config();
585
586
          config temp = AFIO->MAPR; // enable SW Disable JTAG
587
          config_temp &= \sim 0 \times 070000000;
          config_temp \mid = 0 \times 020000000;
588
589
          AFIO->MAPR
                        = config_temp;
590
          GPIOC -> BRR = 0 \times 0001;
591
                                     // LCD reset
592
          delay_ms(100);
          GPIOC -> BSRR = 0 \times 00001;
593
594
          GPIOA -> BSRR = 0 \times 00008;
                                      // back light
595
          LCD_WriteReg(0 \times 0000, 0 \times 0001);
                                               delay_ms(50);
                                                                 /* Enable LCD Oscillator */
596
          LCD_WriteReg(0x0003,0xA8A4);
                                               delay_ms(50);
                                                                 // Power control(1)
597
                                               delay_ms(50);
          LCD_WriteReg(0x000C,0x0000);
                                                                 // Power control(2)
598
                                               delay_ms(50);
          LCD_WriteReg(0x000D,0x080C);
                                                                 // Power control(3)
599
          LCD_WriteReg(0x000E,0x2B00);
                                               delay_ms(50);
                                                                 // Power control(4)
600
          LCD_WriteReg(0x001E,0x00B0);
                                               delay_ms(50);
                                                                 // Power control(5)
601
          LCD_WriteReg(0x0001,0x2B3F);
                                               delay_ms(50);
602
                                                                  // Driver Output Control /* 320*240 0x2B3F */
603
          LCD_WriteReg(0\times0002,0\times0600);
                                               delay_ms(50);
                                                                  // LCD Drive AC Control
604
          LCD_WriteReg(0\times0010,0\times0000);
                                               delay_ms(50);
                                                                  // Sleep Mode off
                                                                    // Entry Mode
         // LCD_WriteReg(0x0011,0x6070);
                                                 delay_ms(50);
                                                                                                          ## flip bit
605
      3 to switch horiz/vert auto-update on write
606
                        LCD_WriteReg(0\times0011,0\times6078);
                                                             delay_ms(50);
                                                                              // Entry Mode
      ## flip bit 3 to switch horiz/vert auto-update on write
607
          LCD_WriteReg(0 \times 0005, 0 \times 0000);
                                               delay ms(50);
                                                                 // Compare register(1)
608
          LCD_WriteReg(0 \times 0006, 0 \times 0000);
                                               delay_ms(50);
                                                                 // Compare register(2)
609
          LCD_WriteReg(0x0016,0xEF1C);
                                               delay_ms(50);
                                                                 // Horizontal Porch
          LCD_WriteReg(0x0017,0x0003);
                                               delay_ms(50);
                                                                 // Vertical Porch
610
          LCD_WriteReg(0x0007,0x0133);
                                               delay_ms(50);
                                                                 // Display Control
611
          LCD_WriteReg(0x000B,0x0000);
                                                                 // Frame Cycle control
612
                                               delay ms(50);
                                                                  // Gate scan start position
613
          LCD_WriteReg(0 \times 000F, 0 \times 0000);
                                               delay ms(50);
          LCD WriteReg(0 \times 0.041, 0 \times 0.000);
                                               delay ms(50);
                                                                 // Vertical scroll control(1)
614
                                                                  // Vertical scroll control(2)
615
          LCD_WriteReg(0\times0042,0\times0000);
                                               delay ms(50);
          LCD WriteReg(0 \times 0048, 0 \times 0000);
                                               delay ms(50);
                                                                  // First window start
616
          LCD WriteReg(0 \times 0049, 0 \times 013F);
                                               delay ms(50);
                                                                  // First window end
617
          LCD WriteReg(0 \times 0.04A, 0 \times 0.000);
                                               delay ms(50);
                                                                 // Second window start
618
619
          LCD WriteReg(0 \times 0.04B, 0 \times 0.000);
                                               delay_ms(50);
                                                                 // Second window end
620
          LCD WriteReg(0x0044,0xEF00);
                                               delay ms(50);
                                                                 // Horizontal RAM address position
          LCD_WriteReg(0 \times 0045, 0 \times 0000);
                                               delay ms(50);
                                                                  // Vertical RAM address start position
621
                                               delay_ms(50);
                                                                  // Vertical RAM address end position
622
          LCD_WriteReg(0\times0046,0\times013F);
623
          LCD_WriteReg(0 \times 0030, 0 \times 0707);
                                               delay_ms(50);
                                                                  // gamma control(1)
```

```
624
          LCD WriteReg(0 \times 0031, 0 \times 0204);
                                               delay ms(50);
                                                                 // gamma control(2)
625
          LCD_WriteReg(0 \times 0032, 0 \times 0204);
                                               delay_ms(50);
                                                                 // gamma control(3)
626
          LCD_WriteReg(0 \times 0033, 0 \times 0502);
                                               delay_ms(50);
                                                                // gamma control(4)
627
          LCD_WriteReg(0\times0034,0\times0507);
                                               delay_ms(50);
                                                                 // gamma control(5)
628
          LCD_WriteReg(0 \times 0035, 0 \times 0204);
                                               delay_ms(50);
                                                                 // gamma control(6)
          LCD_WriteReg(0 \times 0036, 0 \times 0204);
                                               delay_ms(50);
629
                                                                 // gamma control(7)
630
          LCD_WriteReg(0 \times 0037, 0 \times 0502);
                                               delay_ms(50);
                                                                 // gamma control(8)
631
          LCD_WriteReg(0 \times 003A, 0 \times 0302);
                                               delay_ms(50);
                                                                 // gamma control(9)
          LCD WriteReg(0 \times 003B, 0 \times 0302);
                                               delay_ms(50);
632
                                                                 // gamma control(10)
                                                                 // RAM write data mask(1)
          LCD WriteReg(0 \times 0.023, 0 \times 0.000);
                                               delay_ms(50);
633
          LCD WriteReg(0 \times 0.024, 0 \times 0.000);
                                               delay_ms(50);
                                                                 // RAM write data mask(2)
634
          LCD WriteReg(0 \times 0025, 0 \times 8000);
                                               delay_ms(50);
                                                                 // Frame Frequency
635
                                                                 // Set GDDRAM Y address counter
636
          LCD_WriteReg(0 \times 0.04 f, 0);
          LCD_WriteReg(0x004e,0);
                                                                 // Set GDDRAM X address counter
637
638
639
          delay_ms(50);
640
      }
641
      // Paints the LCD with Color
642
     void LCD_Clear( unsigned short Color )
643
644
      {
          unsigned int i;
645
646
647
          LCD_SetCursor(0,0);
648
          GPIOC->BRR = CS_low_Pin;
649
650
651
          LCD WriteIndex( 0x0022 );
          for( i=0; i < MAX X*MAX Y; i++ )</pre>
652
653
               LCD WriteData( Color );
654
655
          GPIOC->BSRR = CS_low_Pin;
656
      }
657
658
      // Write a command
      void LCD_WriteIndex( unsigned short index )
659
660
661
        LCD_Write_Generic(index, 0);
662
663
664
      // Write data
      void LCD_WriteData( unsigned short data )
665
666
667
        LCD_Write_Generic(data, 1);
      }
668
669
670
      // Write generic...
671
      void LCD_Write_Generic(unsigned short toWrite, unsigned short dataBool)
672
673
               unsigned short pc_ops = GPIOC->ODR;
674
675
               // Configure Ports - done in LCD init function
               // Set control bits (RD, WR, D/C, CS)
676
677
               // PC.0(LCD RST = ?), PC.1(WR = 1, then 0, then 1), PC.2(RD = 1), PC.6(CS = unset then
      set), PC.7(DC = dataBool)
678
               pc_ops &= 0xFF38; // unset RST,WR,RD,CS,DC
679
               pc_{ops} = 0x0007; // set RD = 1, and WR = 1, and RST = 1
680
681
               if (dataBool) pc_{ops} = 0x80; // set DC = 1 if we want to write data
682
               GPIOC->ODR = pc_ops;
683
684
               delay_ms(0);
685
               // Write data bits
686
               GPIOB->ODR = toWrite;
687
               delay_ms(0);
688
689
690
               GPIOC->ODR = (pc_ops & 0xFFFD); // unset WR
691
               GPIOC->ODR = pc_ops; // set WR
692
```

```
693
     }
694
695
     void LCD_WriteReg( unsigned short LCD_Reg, unsigned short LCD_RegValue )
696
697
     {
          GPIOC->BRR = CS_low_Pin;
698
699
700
          LCD WriteIndex( LCD Reg );
701
          LCD_WriteData( LCD_RegValue );
702
703
         GPIOC->BSRR = CS low Pin;
704
     }
705
706
     // Set cursor to x y address
707
     void LCD_SetCursor( unsigned short x, unsigned int y )
708
709
          #if
              ( DISP_ORIENTATION == 90 ) || ( DISP_ORIENTATION == 270 )
710
              unsigned short swap_temp;
711
712
              y = (MAX_Y-1) - y;
713
714
              swap\_temp = y;
715
              y = x;
              x = swap\_temp;
716
717
          #elif ( DISP_ORIENTATION == 0 ) || ( DISP_ORIENTATION == 180 )
718
719
720
              y = (MAX_Y-1) - y;
721
722
          #endif
723
724
          LCD WriteReg(0 \times 0.04E, x);
725
          LCD_WriteReg(0\times004F, y);
726
     }
727
     void delay_ms( unsigned int ms )
728
729
730
              int i;
731
              while (ms--)
732
              {
                      //for(i = 0; i < 1669; ++i); // 1 ms delay loop
733
                      for(i = 0; i < 8676; ++i); // 1 ms delay loop
734
735
              }
736
     }
737
738
     void LCD_DrawSquareY( unsigned short x, unsigned short y, unsigned short w, unsigned short h,
739
     unsigned short color )
740
     {
741
              unsigned int i,j;
742
743
          LCD_SetCursor(x,y);
744
          GPIOC->BRR = CS low Pin;
745
746
747
                      for (j=0; j < w; j++)
748
                {
749
                               LCD_SetCursor(x+j, y);
750
751
                               LCD WriteIndex( 0x0022 );
                               for( i=0; i < h; i++ )
752
753
                                                LCD WriteData( color );
754
          }
755
756
          GPIOC->BSRR = CS low Pin;
757
     }
758
     void LCD_DrawSquare( unsigned short x, unsigned short y, unsigned short w, unsigned short h,
     unsigned short color )
759
     {
760
              unsigned int i,j;
```

```
761
          LCD_SetCursor(x,y);
762
763
          GPIOC->BRR = CS_low_Pin;
764
765
766
767
                      for (j=0; j < w; j++)
768
                {
                               LCD_SetCursor(x, y+j);
769
770
                               LCD WriteIndex( 0x0022 );
                               for( i=0; i < h; i++ )
771
                                                LCD WriteData( color );
772
773
          }
774
775
          GPIOC->BSRR = CS_low_Pin;
776
     }
777
     void LCD DrawCharacterOnY (unsigned short x, unsigned short y, unsigned short fgColor, unsigned
778
     short bgColor, unsigned char symbol)
779
     // Draws a character oriented so that left to right goes along the positive y axis
780
781
              unsigned char ascii_buf[16];
782
              unsigned char line;
783
              int i, j;
784
              LCD SetCursor(x,y);
785
786
              GPIOC->BRR = CS_low_Pin;
787
788
              get ascii(ascii buf, symbol);
789
              for (i = 0; i < 16; ++i)
790
791
792
                      line = ascii buf[i];
793
                      LCD_SetCursor(x+i, y);
                      LCD_WriteIndex( 0x0022 );
794
795
                      for (j = 0; j < 8; ++j)
796
                      {
                               if (line & (0\times80 \gg j))
797
798
                                       LCD_WriteData( fgColor );
                               else
799
                                       LCD WriteData( bgColor );
800
                               //delay_ms(1);
801
                      }
802
803
              GPIOC->BSRR = CS_low_Pin;
804
805
     void LCD_DrawCharacterOnX (unsigned short x, unsigned short y, unsigned short fgColor, unsigned
806
     short bgColor, unsigned char symbol, unsigned char underline)
807
     // Draws a character oriented so that left to right goes along the positive X axis
808
809
              unsigned char ascii buf[16];
810
              unsigned char line;
              int i, j;
811
812
813
              LCD SetCursor(x,y);
814
              GPIOC->BRR = CS_low_Pin;
815
              get_ascii(ascii_buf, symbol);
816
817
818
              for (i = 0; i < 16; ++i)
819
              {
820
                      line = ascii buf[i];
                      if (i == 15 && underline)
821
                               line = 0xFF;
822
                      LCD SetCursor(x, y+i);
823
824
                      LCD WriteIndex( 0x0022 );
825
                      for (j = 0; j < 8; ++j)
826
                      {
                               if (line & (0\times80 >> j))
827
                                       LCD_WriteData( fgColor );
828
```

```
829
                              else
830
                                       LCD_WriteData( bgColor );
831
                              //delay_ms(1);
                      }
832
833
              GPIOC->BSRR = CS_low_Pin;
834
835
     void LCD DrawChar rc (unsigned int row, unsigned int col, unsigned short fgColor, unsigned short
836
     bgColor, unsigned char symbol, unsigned char underline)
     // Draws a character on the givel row and column
837
838
              //LCD_DrawCharacterOnY(row * 16, MAX_Y - 12 - (col*8), fgColor, bgColor, symbol);
839
              LCD_DrawCharacterOnX(col*8, row*16, fgColor, bgColor, symbol, underline);
840
841
842
     void LCD_WriteCharactersOnY (unsigned short x, unsigned short y, unsigned short fgColor, unsigned
843
     short bgColor, char* words, int maxLength)
844
     // Draws a line of characters increasing on Y axis
845
              int i;
846
              for (i = 0; i < maxLength; ++i)
847
848
849
                      if(words[i] == 0)
850
                              break:
                      LCD_DrawCharacterOnY(x, y - (8*i), fgColor, bgColor, words[i]);
851
852
853
     void LCD WriteLinesOnY(unsigned short lineNumber, unsigned short fgColor, unsigned short bgColor,
854
     char* words, char drawToLineEnd)
855
              int len, numLines, lastLineLength, i, curLine;
856
              char spaces[CHARS HORIZ ON Y];
857
858
              len = strlen(words);
              numLines = len / CHARS_HORIZ_ON_Y;
859
860
              lastLineLength = len % CHARS_HORIZ_ON_Y;
861
              if(lastLineLength != 0)
862
                      numLines++;
863
864
              for(curLine = 0; curLine < numLines; ++curLine)</pre>
865
                      LCD_WriteCharactersOnY((lineNumber + curLine) * 16, MAX_Y-12, fgColor, bgColor,
866
     words+((CHARS_HORIZ_ON_Y) * curLine), CHARS_HORIZ_ON_Y);
867
              // TODO: fix draw to end of last line
868
              if(lastLineLength != 0)
869
870
                      for(i = 0; i < CHARS_HORIZ_ON_Y; ++i)</pre>
871
872
                              spaces[i] = ' ';
873
874
875
                      LCD WriteCharactersOnY((lineNumber + curLine - 1) * 16, MAX Y-12-
     (lastLineLength*8), fgColor, bgColor, spaces, CHARS_HORIZ_ON_Y-lastLineLength);
876
              }
877
     }
878
879
880
881
882
     // END OF FILE
883
     // main.c
     // ECE 3710
884
885
     // Final Project
     // By William Hatch and Scott Sorensen
886
887
888
889
     #include "stm32f10x.h"
     #include "lcd.h"
890
     #include "usart2.h"
891
     #include "ps2_over_gpioc.h"
892
     #include "code_to_ascii.h"
893
```

```
#include "terminal.h"
894
     #include "dac.h"
895
896
897
898
     void SystemInit(void)
899
900
     {
901
902
     void handlePs2Data(void);
903
     void handleUsartData(void);
904
905
906
907
908
     extern int cursorCol, cursorRow;
909
     extern char screenChars[ROWS][COLS];
910
     extern unsigned short screenFgColor[ROWS][COLS];
911
     extern unsigned short screenBgColor[ROWS][COLS];
912
913
914
915
     int main()
916
              // Enable external oscillator
917
        //RCC->CFGR = 0x0418000A;
                                                          // Mult PLL by 8 = 32 \text{ MHz}
918
              RCC->CFGR = 0 \times 0428000A;
919
                                                              // 48MHz
920
              /// 418->428 makes it 48Mhz
921
        RCC->CR = 0 \times 03004583; //USe PLL Clock for SW and MC
922
923
924
              // Initialize everything
925
              LCD Initialization();
926
              usart2_init();
927
              ps2_over_gpioc_init();
928
              LCD_Clear(Black);
929
930
              scan_code_init();
              DAC_init();
931
              bufClear();
932
933
              while(1)
934
935
              {
936
                       handlePs2Data();
                       handleUsartData();
937
938
                       flushScreen();
              }
939
940
941
     }
942
943
     void handlePs2Data()
944
      {
945
              int bytes;
              int i;
946
947
              unsigned char buf[PS2_DATA_SIZE];
948
949
              // copy buffer
950
              bytes = ps2_memcpy(buf);
951
952
              // tx buffer
953
              for (i = 0; i < bytes; ++i)</pre>
954
              {
955
                       usart2_tx(buf[i]);
956
              }
957
     }
958
     void handleUsartData()
959
960
      {
961
              int bytes;
              unsigned char buf[USART2_DATA_SIZE];
962
963
              // copy buffer
```

```
964
               bytes = usart2 memcpy(buf);
 965
 966
               // push buffered data to display
 967
               handleAscii(buf, bytes);
 968
      }
 969
 970
 971
      void HardFault Handler()
 972
      {
               LCD Clear(Red);
 973
               //Reset Handler();
 974
 975
 976
977
      //ps2_over_gpioc.c
 978
      #include "stm32f10x.h"
 979
 980
      #include "ps2_over_gpioc.h"
      #include "usart2.h
 981
      #include "code_to_ascii.h"
 982
      #include "lcd.\overline{h}"
 983
 984
 985
      unsigned char ps2_data[PS2_DATA_SIZE];
 986
      int ps2_bytes_rec = 0;
 987
 988
      extern unsigned char map [0 \times 100];
 989
      extern unsigned char shift_map[0x100];
 990
      extern unsigned char ctl_map[0x100];
 991
 992
      void ps2_insert_to_buffer(char *insert, int size)
 993
 994
               int i:
 995
 996
               if (ps2_bytes_rec + size > PS2_DATA_SIZE)
 997
                       return;
998
               NVIC->ICER[0] = NVIC_ICER_CLRENA_10;
999
1000
               for (i = 0; i < size; ++i)
1001
               {
1002
                       ps2_data[ps2_bytes_rec++] = insert[i];
1003
               NVIC->ISER[0] = NVIC_ISER_SETENA_10;
1004
1005
1006
      // Switch PC0 to PC4, and PC1 to PC3
1007
      void ps2_over_gpioc_init(void)
1008
1009
1010
               // Setup EXTIO
1011
               EXTI->IMR |= EXTI_IMR_MR4;
1012
               //EXTI->EMR |= EXTI EMR MR4;
               EXTI->RTSR |= EXTI_RTSR_TR4;
1013
1014
          RCC->APB2ENR |= RCC_APB2ENR_IOPCEN;
1015
1016
1017
               // Setup AFIO events
1018
               RCC->APB2ENR |= 1; // enable AFIO clock
1019
               //AFIO->EVCR = 0b10100000; // Events enabled on PC0
               //AFIO->EVCR = AFIO_EVCR_EVOE | AFIO_EVCR_PORT_PC | AFIO_EVCR_PIN_PX0;
1020
               AFIO->EXTICR[1] = (AFIO->EXTICR[2] & 0xFFFFFFF0) | 0x2;
1021
1022
               GPIOC->CRL = (GPIOC->CRL & 0xFFF00FFF) | 0x88000; // Configure PC[3-4] for pull-up/pull-
      down input
               GPIOC->ODR |= 0x18; // set bits of ODR so we can read the input.
1023
1024
               // Setup NVIC
1025
               // EXT4 is interrupt number 10
1026
               NVIC->ISER[0] = NVIC ISER SETENA 10;
1027
1028
1029
               // Set priority - this should be our high priority interrupt
1030
               //NVIC->IP[1] = (NVIC->IP[1] & 0xFF00FFFF) | 0x00FF0000;
1031
1032
      }
```

```
void EXTI4_IRQHandler(void)
1034
1035
       {
               static unsigned int calls = 0;
1036
               static unsigned char rx_state = 0; // 0 not receiving, 1 receiving
1037
               static unsigned char data_bits_rec = 0;
1038
1039
               static unsigned char data = 0;
1040
               static unsigned char stop_bit_rec = 0;
               static unsigned char last_byte_was_escape = 0;
1041
               static unsigned char shift_on = 0;
1042
               static unsigned char ctl_on = 0;
1043
               static unsigned char alt on = 0;
1044
1045
               unsigned char bit, ascii;
1046
1047
               calls++;
1048
1049
               bit = GPIOC->IDR & 0x8; // read pc3 (data line)
               bit = bit << 4;
1050
1051
               while(1)
1052
         {
1053
               // Handle start bit
1054
               if (!rx_state)
1055
               {
1056
                        if(bit)
1057
                                 break;
1058
                        rx state = 1;
1059
                        data_bits_rec = 0;
1060
                        data = 0;
1061
                        stop_bit_rec = 0;
1062
1063
                        break;
1064
               }
1065
               // Receive data
1066
               if (data_bits_rec < 8)</pre>
1067
1068
               {
1069
                        data = data >> 1;
                        data |= bit;
1070
1071
                        data_bits_rec++;
1072
                        break;
               }
1073
1074
1075
               // Handle stop/acknowledge bits
               if (! stop_bit_rec)
1076
1077
               {
1078
                        stop_bit_rec = 1;
1079
1080
1081
                        // Map scancodes to ascii, throwing away everything but lowercase
                        // alphanumeric characters and spaces.
1082
1083
1084
                        if (last_byte_was_escape)
1085
                        {
1086
                                 last byte was escape = 0;
                                 if(shift_on \& \& (data == 0x12 | data == 0x59))
1087
1088
                                          shift on = 0;
                                 if(ctl_on && \overline{\text{(data == 0x14)}})
1089
1090
                                          ctl on = 0;
                                 if(alt_on && (data == 0 \times 11))
1091
1092
                                          alt_on = 0;
                                 break:
1093
1094
                        if (data == 0xF0) // key up escape
1095
1096
1097
                                 last byte was escape = 1;
1098
                                 break;
1099
                        }
1100
                        if (data == 0 \times 12 | data == 0 \times 59) // key down shift
1101
1102
```

1033

```
1103
                                 shift on = 1;
1104
                                 break;
1105
                         if (data == 0 \times 14) // key down ctl
1106
1107
                         {
1108
                                 ctl_on = 1;
1109
                                 break;
1110
                        if (data == 0 \times 11) // key down alt
1111
1112
1113
                                 alt on = 1;
                                 break;
1114
1115
                         }
1116
                         if (ctl_on)
1117
1118
                                 ascii = ctl_map[data];
1119
                         else if (shift_on)
1120
                                 ascii = shift_map[data];
                         else
1121
1122
                                 ascii = map[data];
1123
1124
1125
1126
1127
                         // put data into buffer
1128
                         if (ps2_bytes_rec < PS2_DATA_SIZE-1)</pre>
1129
                                 if(alt_on)
1130
1131
                                 {
1132
                                          ps2_data[ps2_bytes_rec] = 0x1b; // escape
1133
                                          ps2_bytes_rec++;
1134
1135
                                 ps2_data[ps2_bytes_rec] = ascii;
1136
                                 //ps2_data[ps2_bytes_rec] = data;
1137
                                 ps2_bytes_rec++;
1138
1139
                         }
1140
1141
                        break;
                }
1142
1143
1144
1145
                // Here we will let one clock cycle pass, because there may
                // be an acknowledgement bit, but we're not sure how to handle it.
1146
1147
1148
                rx_state = 0;
1149
                break;
1150
           }
1151
                  //NVIC -> ICPR[0] = 0 \times 40;
1152
1153
                         EXTI->PR = 0 \times 10; // Clear EXTI pending
                         NVIC->ICPR[0] = NVIC_ICPR_CLRPEND_10;
1154
1155
           return;
1156
       }
1157
1158
       void ps2_dump_data_over_usart2()
1159
       // Dumps data in ps2_data over usart2
1160
1161
                int i;
1162
                for (i = 0; i < ps2_bytes_rec; ++i)</pre>
1163
1164
1165
                         usart2 tx(ps2 data[i]);
                        LCD_DrawCharacterOnY(40,40,Blue, Black, ps2_data[i]);
1166
1167
1168
                ps2_bytes_rec =0;
1169
       }
1170
       int ps2_memcpy(unsigned char * dst)
1171
1172
```

```
1173
              int ret, i;
1174
              // turn off interrupts
1175
              NVIC->ICER[0] = NVIC_ICER_CLRENA_10;
1176
              // copy buffer
1177
              for(i = 0; i < ps2_bytes_rec; ++i)</pre>
1178
              {
1179
                      dst[i] = ps2_data[i];
1180
1181
              ret = ps2 bytes rec;
1182
1183
              ps2\_bytes\_rec = 0;
1184
              // enable interrupts
1185
              NVIC->ISER[0] = NVIC_ISER_SETENA_10;
1186
              return ret;
1187
      }
1188
1189
1190
      // terminal.c
1191
1192
1193
      #include "terminal.h"
1194
      #include "lcd.h
      #include "ps2_over_gpioc.h"
1195
      #include "dac.h"
1196
1197
1198
      #define CURSOR_STACK_SIZE 200
1199
      int cursorStack[2][2][CURSOR_STACK_SIZE]; // one for CSI stack, one for ESC (non-CSI stack)
1200
      int CSIcursorStackPointer=-1;
1201
      int ESCcursorStackPointer=-1;
1202
      int cursorCol = 0; // cursor column number
1203
      int cursorRow = 0; // cursor line number
1204
1205
      unsigned short currFqColor = TermDefault;
1206
      unsigned short currBgColor = TermBlack;
1207
      unsigned char currDisplayOps = 0; // for flags for underline, blink, etc
1208
1209
      char screenChars[ROWS][COLS];
1210
1211
      char oldScreenChars[ROWS][COLS];
      unsigned short screenFgColor[ROWS][COLS];
1212
      unsigned short screenBgColor[ROWS][COLS];
1213
      unsigned short oldScreenFgColor[ROWS][COLS];
1214
1215
      unsigned short oldScreenBgColor[ROWS][COLS];
1216
      int screenTop = 0;
1217
      int oldScreenTop = 0;
1218
1219
      unsigned char screenDisplayOps[ROWS][COLS]
1220
      unsigned char oldScreenDisplayOps[ROWS][COLS];
      unsigned char drawCursor = 1;
1221
      #define CURSOR CHAR 1 // attribute for character that the cursor is on
1222
1223
      #define UNDERLINE 2
1224
      #define BOLD 4
1225
      #define BLINK 8
1226
      #define REVERSE VIDEO 16
1227
      #define HALF_BRIGHT 32
1228
1229
1230
1231
      1232
1233
      void addCursorAttr()
1234
      { // add cursor attribute to screenChar at current cursor location
1235
              int rowTr; // translated row
1236
              rowTr = (cursorRow + screenTop) % ROWS;
              screenDisplayOps[rowTr][cursorCol] |= CURSOR CHAR;
1237
1238
1239
      void remCursorAttr()
1240
      { // remove cursor attribute to screenChar at current cursor location
1241
              int rowTr; // translated row
1242
              rowTr = (cursorRow + screenTop) % ROWS;
```

```
screenDisplayOps[rowTr][cursorCol] &= ~CURSOR CHAR;
1243
1244
1245
      void clearExtraneousCursors()
      { // since we have extra cursors, let's add this cludge to clear them.
1246
1247
               int i,j;
               for(i = 0; i < ROWS; ++i)
1248
1249
               {
                        for(j = 0; j < COLS; ++j)
1250
1251
                                screenDisplayOps[i][j] &= ~CURSOR CHAR;
1252
                        }
1253
1254
               addCursorAttr();
1255
1256
      }
1257
      void do_LF()
1258
1259
      // Do a standard line feed
1260
               int i, rowTr;
1261
1262
               remCursorAttr();
1263
1264
               if (cursorRow < ROWS -1)</pre>
1265
                        cursorRow = (cursorRow + 1);
1266
               else
1267
               {
                        screenTop = (screenTop+1) % ROWS;
1268
1269
                        rowTr = (ROWS - 1 + screenTop) % ROWS;
1270
                        for(i = 0; i < COLS; ++i)
1271
                        {
1272
                                screenChars[rowTr][i] = ' ';
1273
                        }
1274
1275
               addCursorAttr();
1276
      }
1277
      void do_CR()
1278
1279
      // Do a standard carriage return
1280
1281
               remCursorAttr();
1282
               cursorCol = 0;
1283
               addCursorAttr();
      }
1284
1285
      void handle_LF()
1286
      // Handle the \n character
1287
      // In newline mode this gives both LF and CR
1288
1289
      {
1290
               do LF();
1291
               // TODO - check somehow whether newline mode is on (it probably always will be
1292
               // if (newLineMode)
1293
                        do_CR();
1294
      }
1295
1296
      void handle CR()
1297
       // handle CR character
1298
               // I think this may be the place to do this line blackout...
1299
               int j;
1300
1301
               for(j = cursorCol; j < COLS; ++j)</pre>
1302
               {
                        screenChars[(cursorRow + screenTop)%ROWS][j] = ' ';
1303
1304
               }
               do_CR();
1305
1306
      }
1307
1308
      void advance cursor()
1309
      // advances the cursor, duh.
1310
1311
               int i;
1312
               remCursorAttr();
```

```
1313
               if(cursorCol >= COLS - 1)
                  // It seems many terminals don't automatically scroll, and it breaks some
1314
               {
       functionality.
                         // Unless I make a termcap entry for this terminal or figure out how it works,
1315
      it breaks functionality of
                         // programs like vim in screen... odd...
1316
1317
                       cursorCol = 0;
1318
                       if (cursorRow < ROWS -1)</pre>
                                cursorRow = (cursorRow + 1);
1319
                       else
1320
                       { // Scroll screen
1321
1322
                                screenTop = (screenTop +1) % ROWS;
                                for (i = 0; i < COLS; ++i)
1323
                                { // clear the new bottom line to be blank as it scrolls in.
1324
                                         screenChars[(screenTop+ROWS-1)%ROWS][i] = ' ';
1325
                                         screenDisplayOps[(screenTop+ROWS-1)%ROWS][i] = 0;
1326
1327
                                         screenFgColor[(screenTop+ROWS-1)%ROWS][i] = TermDefault;
                                         screenBgColor[(screenTop+ROWS-1)%ROWS][i] = TermBlack;
1328
1329
                                }
1330
                       }
1331
1332
               else
1333
                       cursorCol++;
1334
               addCursorAttr();
1335
      }
1336
1337
      void handle normal(char ascii)
1338
      // handles normal characters for printing to the screen
1339
               int rowTr; // translated row
1340
1341
               rowTr = (cursorRow + screenTop) % ROWS;
1342
               screenChars[rowTr][cursorCol] = ascii;
               screenFgColor[rowTr][cursorCol] = currFgColor;
1343
               screenBgColor[rowTr][cursorCol] = currBgColor;
1344
               screenDisplayOps[rowTr][cursorCol] = currDisplayOps;
1345
1346
               advance_cursor();
1347
      }
1348
      void moveCursor(unsigned int num, unsigned int dir)
1349
1350
               #define UP 1
1351
               #define DOWN 2
1352
               #define LEFT 3
1353
               #define RIGHT 4
1354
               remCursorAttr();
1355
               if (dir == UP)
1356
1357
               {
1358
                       cursorRow -= num;
1359
                       if (cursorRow < 0)
                                cursorRow = 0;
1360
1361
               else if (dir == DOWN)
1362
1363
                       cursorRow += num;
1364
                       if (cursorRow > ROWS -1)
1365
1366
                                cursorRow = ROWS - 1;
1367
               else if (dir == RIGHT)
1368
1369
               {
1370
                       cursorCol += num;
                       if (cursorCol > COLS - 1)
1371
1372
                                cursorCol = COLS -1;
1373
               else if (dir == LEFT)
1374
1375
               {
1376
                       cursorCol -= num;
1377
                       if (cursorCol < 0)</pre>
1378
                                cursorCol = 0;
1379
1380
               addCursorAttr();
```

```
1381
      }
1382
1383
      void handle_colorCodes(unsigned int *csi_numbers, unsigned int csi_numbers_rec)
1384
1385
               int i;
               for (i = 0; i < csi_numbers_rec; ++i)</pre>
1386
1387
               {
1388
                        switch (csi numbers[i])
1389
                        {
1390
                                 case 0:
                                         currBgColor = TermBlack;
1391
1392
                                         currFgColor = TermDefault;
1393
                                         currDisplayOps = 0;
1394
                                         break;
1395
                                 case 1:
1396
                                         currDisplayOps |= BOLD;
1397
                                         break;
1398
                                 case 2:
1399
                                         // set half-bright
1400
                                         currDisplayOps |= HALF BRIGHT;
1401
                                         break:
1402
                                 case 4:
                                         // set underscore
1403
                                         currDisplayOps |= UNDERLINE;
1404
1405
                                         break:
1406
                                 case 5:
1407
                                         // set blink
1408
                                         currDisplayOps |= BLINK;
1409
                                         break;
1410
                                 case 7:
1411
                                         // set reverse video (whatever that is)
                                         currDisplayOps |= REVERSE_VIDEO;
1412
1413
                                         break;
                                 case 21:
1414
                                         // set normal intensity (bold off?)
1415
                                         currDisplayOps &= ~BOLD;
1416
1417
                                         break;
                                 case 22:
1418
                                         // set normal intensity (half-bright off?)
1419
                                         currDisplayOps &= ~HALF_BRIGHT;
1420
                                         break;
1421
1422
                                 case 24:
                                         // set underline off
1423
                                         currDisplayOps &= ~UNDERLINE;
1424
                                         break;
1425
                                 case 25:
1426
1427
                                         // set blink off
1428
                                         currDisplayOps &= ~BLINK;
1429
                                         break:
1430
                                 case 27:
1431
                                         // set reverse video off
1432
                                         currDisplayOps &= ~REVERSE VIDEO;
1433
                                         break:
1434
                                // 30-49 are mostly basic color options
1435
1436
                                 case 30:
                                         currFgColor = TermBlack;
1437
                                         if(currDisplayOps & BOLD)
1438
1439
                                                  currFgColor = TermBlackBright;
1440
                                         break:
                                 case 31:
1441
1442
                                         currFqColor = TermRed;
                                         if(currDisplayOps & BOLD)
1443
1444
                                                  currFqColor = TermRedBright;
1445
                                         break;
1446
                                 case 32:
1447
                                         currFgColor = TermGreen;
1448
                                         if(currDisplayOps & BOLD)
                                                  currFgColor = TermGreenBright;
1449
1450
                                         break;
```

```
1451
                                 case 33:
1452
                                         currFgColor = TermBrown;
1453
                                         if(currDisplayOps & BOLD)
                                                           currFgColor = TermBrownBright;
1454
1455
                                         break;
                                 case 34:
1456
                                         currFgColor = TermBlue;
1457
                                         if(currDisplayOps & BOLD)
1458
                                                  currFgColor = TermBlueBright;
1459
1460
                                         break:
                                 case 35:
1461
1462
                                         currFgColor = TermMagenta;
                                         if(currDisplayOps & BOLD)
1463
1464
                                                  currFgColor = TermMagentaBright;
1465
                                         break:
1466
                                 case 36:
1467
                                         currFgColor = TermCyan;
                                         if(currDisplayOps & BOLD)
1468
                                                  currFgColor = TermCyanBright;
1469
1470
                                         break:
1471
                                 case 37:
1472
                                         currFgColor = TermWhite;
                                         if(currDisplayOps & BOLD)
1473
                                                  currFgColor = TermWhiteBright;
1474
1475
                                         break:
                                 case 38:
1476
1477
                                         currFgColor = TermDefault;
                                         currDisplayOps |= UNDERLINE;
1478
1479
                                         break;
                                 case 39:
1480
                                         currFqColor = TermDefault;
1481
1482
                                         currDisplayOps &= ~UNDERLINE;
1483
                                         break;
                                 case 40:
1484
                                          currBgColor = TermBlack;
1485
1486
                                         break;
1487
                                 case 41:
                                          currBgColor = TermRed;
1488
1489
                                         break;
                                 case 42:
1490
                                         currBgColor = TermGreen;
1491
1492
                                         break;
                                 case 43:
1493
1494
                                         currBgColor = TermBrown;
1495
                                         break;
1496
                                 case 44:
1497
                                         currBgColor = TermBlue;
1498
                                         break;
1499
                                 case 45:
                                         currBgColor = TermMagenta;
1500
1501
                                         break:
1502
                                 case 46:
                                         currBgColor = TermCyan;
1503
1504
                                         break;
1505
                                 case 47:
1506
                                         currBgColor = TermWhite;
1507
                                         break:
1508
                                 case 48:
1509
                                          // ???
1510
                                         break:
                                 case 49:
1511
1512
                                         currBqColor = TermBlack;
1513
                                         break:
1514
1515
                                 default:
1516
                                         break;
1517
                        }
1518
               }
1519
      }
1520
```

```
1521
1522
1523
      void keepCursorInBounds()
1524
               if (cursorRow >= ROWS)
1525
1526
               {
1527
                        cursorRow = ROWS-1;
1528
               else if (cursorRow < 0)</pre>
1529
                       cursorRow = 0;
1530
               if (cursorCol >= COLS-1)
1531
                       cursorCol = COLS-1;
1532
               else if (cursorCol < 0)</pre>
1533
                       cursorCol = 0;
1534
1535
      }
1536
1537
      void reportCursorToHost()
1538
1539
               char rowColResponse[10];
               char rowNum[3];
1540
1541
               char colNum[3];
               int divisor, i,j;
1542
1543
               char firstSeen = 0;
1544
               int rowsSent=0, colsSent=0;
               // echo ESC[<ROW>;<COL>R
1545
1546
1547
               for (i = 0, divisor = 100; divisor; ++i, divisor/=10)
1548
1549
                        rowNum[i] = ((cursorRow+1)/divisor) % 10;
1550
                        colNum[i] = ((cursorCol+1)/divisor) % 10;
1551
1552
               rowColResponse[0] = 033; // ESC
               rowColResponse[1]='[';
1553
               for(i = 2, j = 0; j < 4; ++j) // i is response position, j is loop position
1554
1555
               {
                        if(rowNum[j] || j == 3) // only start putting out characters if we've seen the
1556
      first character or it's all 0
                                firstSeen = 1;
1557
1558
                        if(firstSeen)
1559
                                rowColResponse[i] = rowNum[j];
1560
1561
                                ++i;
1562
                                ++rowsSent;
                        }
1563
1564
               rowColResponse[i++] = ';';
1565
1566
               firstSeen = 0;
1567
               for(j = 0; j < 4; ++j) // i is response position and keeps its previous value, j is loop
      position
1568
                        if(colNum[j])
1569
1570
                                firstSeen = 1;
                        if(firstSeen)
1571
1572
                        {
                                rowColResponse[i] = colNum[j];
1573
1574
                                ++i;
                                ++colsSent;
1575
1576
                        }
1577
1578
               rowColResponse[i] = 'R';
1579
1580
               ps2 insert to buffer(rowColResponse, rowsSent+colsSent+4);
      }
1581
1582
      void CSIpushCursor()
1583
1584
1585
               if(CSIcursorStackPointer < CURSOR STACK SIZE-1)</pre>
1586
                        CSIcursorStackPointer++;
1587
1588
                        cursorStack[0][0][CSIcursorStackPointer] = cursorRow;
```

```
cursorStack[0][1][CSIcursorStackPointer] = cursorCol;
1589
1590
1591
      void CSIpopCursor()
1592
1593
      {
              remCursorAttr();
1594
              if(CSIcursorStackPointer >= 0)
1595
1596
                       cursorRow = cursorStack[0][0][CSIcursorStackPointer];
1597
                       cursorCol = cursorStack[0][1][CSIcursorStackPointer];
1598
                       CSIcursorStackPointer--;
1599
1600
              addCursorAttr();
1601
1602
      void ESCpushCursor()
1603
1604
1605
              if(ESCcursorStackPointer < CURSOR_STACK_SIZE-1)</pre>
1606
                       ESCcursorStackPointer++;
1607
                       cursorStack[0][0][ESCcursorStackPointer] = cursorRow;
1608
                       cursorStack[0][1][ESCcursorStackPointer] = cursorCol;
1609
1610
1611
      void ESCpopCursor()
1612
1613
      {
1614
              remCursorAttr();
1615
              if(ESCcursorStackPointer >= 0)
1616
              {
1617
                       cursorRow = cursorStack[0][0][ESCcursorStackPointer];
1618
                       cursorCol = cursorStack[0][1][ESCcursorStackPointer];
1619
                       ESCcursorStackPointer--;
1620
1621
              addCursorAttr();
1622
      }
1623
      1624
1625
      void bufClear(void)
1626
1627
              int i,j;
1628
              for(i = 0; i < ROWS; ++i)
1629
1630
                       for(j = 0; j < COLS; ++j)
1631
1632
                               screenChars[i][j] = ' ';
1633
                               screenFgColor[i][j] = Grey;
1634
                               screenBgColor[i][j] = Black;
1635
1636
                               screenDisplayOps[i][j] = 0;
1637
                       }
              }
1638
      }
1639
1640
1641
      void flushScreen()
1642
1643
      {
              int i,j;
1644
              int Io, In; // Translated i for screen top differences due to scrolling
1645
              char letter;
1646
1647
              static int count = 0;
1648
              static unsigned int refreshes = 0;
              #define REFRESH COUNT 20
1649
1650
              clearExtraneousCursors();
              count = (count+1) % (REFRESH COUNT +1);
1651
              if (count == REFRESH COUNT)
1652
                       refreshes++; // count number of refreshes, for blinking
1653
1654
1655
              for(i = ROWS-1; i \geq 0; --i) // Start from the bottom of the screen
1656
                       Io = (i + oldScreenTop) % ROWS;
1657
                       In = (i + screenTop) % ROWS;
1658
```

```
1659
                      for(j = 0; j < COLS; ++j)
1660
1661
                              if (oldScreenChars[Io][j] != screenChars[In][j]
                                       || oldScreenFgColor[Io][j] != screenFgColor[In][j]
1662
1663
                                       || oldScreenBgColor[Io][j] != screenBgColor[In][j]
1664
                                 || oldScreenDisplayOps[Io][j] != screenDisplayOps[In][j]
1665
                                       || count == REFRESH COUNT) // redraw everything every so often to
      clear out glitches
                              {
1666
                                       letter = screenChars[In][j];
1667
                                       if(screenDisplayOps[In][j] & BLINK && refreshes%2)
1668
                                       { // blink every other whole screen refresh cycle
1669
                                               letter = ' ';
1670
1671
                                       if ((screenDisplayOps[In][j] & CURSOR_CHAR) && drawCursor ||
1672
      screenDisplayOps[In][j] & REVERSE_VIDEO)
1673
1674
                                               LCD_DrawChar_rc (i, j, screenBgColor[In][j], screenFgColor
      [In][j], letter, screenDisplayOps[In][j] & UNDERLINE);
1675
1676
                                       else
1677
                                               LCD_DrawChar_rc (i, j, screenFgColor[In][j], screenBgColor
      [In][j], letter, screenDisplayOps[In][j] & UNDERLINE);
1678
1679
                      }
1680
              }
1681
1682
              for(i = 0; i < ROWS; ++i)
1683
              {
                      for (j = 0; j < COLS; ++j)
1684
1685
                      1
1686
                              oldScreenChars[i][j] = screenChars[i][j];
                              oldScreenFgColor[i][j] = screenFgColor[i][j];
1687
1688
                              oldScreenBgColor[i][j] = screenBgColor[i][j];
1689
                      }
1690
1691
              oldScreenTop = screenTop;
1692
1693
1694
      }
1695
      1696
1697
      void handleAscii(unsigned char *buf, int bytes)
1698
      // Outputs ascii to screen or handles escape codes
1699
      // Much of the escape code handling is in "man console_codes" in in the Linux Programmer's Manual
1700
1701
      {
              int i,j,k, rowTr;
1702
1703
              static unsigned int escStat = 0; // variable to hold current escape sequence status
1704
              #define ESC ON 0x00000001
              #define CSI ON 0x00000002
1705
              #define CSI QUESTION ON 0x00000004
1706
              // TODO - add flags for multi-stage escapes
1707
1708
              #define CSI NUM MAX 25
1709
              static unsigned int csi_nums[CSI_NUM_MAX];
1710
              static unsigned int csi nums rec = 0;
              static unsigned int csi_digits = 0; // to hold digits until a separator is hit
1711
1712
1713
              keepCursorInBounds();
1714
              for(i = 0; i < bytes; ++i)
1715
              {
1716
                      if(escStat & CSI ON)
1717
                      {
                              // handle csi codes
1718
1719
                              // if you get:
1720
                              // <CSI escape><number>m
1721
                              // you change colors. You can add multiple color options like this:
1722
                              // <CSI esc><number>;<number>m
1723
                              // Other CSI sequences behave similarly, a semicolon separated list of
      decimal numbers
```

```
// The function is determined by the terminating character (m for color
1724
      options, others for other things)
1725
                                // Note that the decimal number can be multiple bytes long... but probably
      not more than 2.
                                // My testing on the Linux console shows that numbers with more than 2
1726
      digits are to be simply ignored.
1727
1728
                                if (buf[i] >= 48 && buf[i] <= 57)
1729
                                // numbers in ascii are 48-57 (0-9)
                                {
1730
                                         csi digits *= 10; // decimal left shift
1731
                                         csi digits += buf[i] - 48; // add in the new number
1732
1733
                                else if (buf[i] == ';')
1734
1735
                                { // separator for another number
1736
                                         if (csi_nums_rec < CSI_NUM_MAX)</pre>
1737
1738
                                                 csi_nums[csi_nums_rec++] = csi_digits;
1739
                                         csi_digits = 0;
1740
1741
1742
                                else if (buf[i] == '?')
1743
                                {
1744
                                         escStat |= CSI_QUESTION_ON;
1745
                                }
1746
                                else
1747
                                { // Here we handle the numbers received based on the terminating
      character.
1748
                                         // if no numbers have been received, treat it as if a 0 were
       received
1749
                                         csi nums[csi nums rec++] = csi digits;
1750
                                         csi digits = 0;
1751
                                         switch (buf[i])
1752
1753
                                                 case 'm': // Colors!!!
1754
1755
                                                          handle_colorCodes(csi_nums, csi_nums_rec);
1756
                                                          break;
1757
                                                 case 'A': // move cursor up n rows
                                                          for(j = 0; j < csi_nums_rec; ++j)</pre>
1758
1759
                                                          {
                                                                  if(csi_nums[j] == 0)
1760
1761
                                                                           csi_nums[j] = 1;
                                                                  moveCursor(csi_nums[j], UP);
1762
                                                          }
1763
1764
                                                          break;
                                                 case 'B': // move down n rows
1765
                                                 case 'e': // same as B
1766
1767
                                                          for(j = 0; j < csi_nums_rec; ++j)</pre>
1768
1769
                                                                  if(csi nums[j] == 0)
1770
                                                                           csi_nums[j] = 1;
1771
                                                                  moveCursor(csi_nums[j], DOWN);
1772
                                                          }
1773
                                                          break;
1774
                                                 case 'C': // move right n rows
                                                       'a': // same as C
1775
1776
                                                          for(j = 0; j < csi_nums_rec; ++j)</pre>
1777
                                                          {
1778
                                                                  if(csi_nums[j] == 0)
1779
                                                                           csi_nums[j] = 1;
1780
                                                                  moveCursor(csi_nums[j], RIGHT);
                                                          }
1781
                                                          break;
1782
1783
                                                 case 'D': // move left n rows
1784
                                                          for(j = 0; j < csi_nums_rec; ++j)</pre>
1785
                                                          {
1786
                                                                  if(csi_nums[j] == 0)
1787
                                                                           csi_nums[j] = 1;
1788
                                                                  moveCursor(csi_nums[j], LEFT);
```

```
1789
1790
                                                          break;
1791
                                                  case 'E': // move cursor down n rows, to column 1
1792
                                                          for(j = 0; j < csi_nums_rec; ++j)</pre>
1793
                                                                   if(csi_nums[j] == 0)
1794
1795
                                                                            csi_nums[j] = 1;
1796
                                                                   moveCursor(csi_nums[j], DOWN);
1797
1798
                                                          cursorCol = 0;
1799
                                                          break;
1800
                                                  case 'F': // move cursor up n rows, to column 1
1801
                                                          for(j = 0; j < csi_nums_rec; ++j)</pre>
1802
                                                                   if(csi_nums[j] == 0)
1803
1804
                                                                           csi_nums[j] = 1;
1805
                                                                   moveCursor(csi_nums[j], UP);
1806
1807
                                                          cursorCol = 0;
1808
                                                          break:
1809
                                                  case 'G': // move cursor to indicated column, current row
1810
                                                          remCursorAttr();
1811
                                                          cursorCol = csi_nums[csi_nums_rec-1] - 1;
1812
                                                          keepCursorInBounds();
1813
                                                          addCursorAttr();
1814
                                                          break;
                                                  case 'H': // move cursor to indicated row,column
1815
                                                  case 'f': // same as H
1816
1817
                                                          remCursorAttr();
1818
                                                          if (csi_nums_rec <2)</pre>
1819
                                                                   cursorCol = 0;
1820
                                                          else
1821
                                                                   cursorCol = csi_nums[1] -1;
                                                          cursorRow = csi_nums[0] -1;
1822
1823
                                                          keepCursorInBounds();
1824
1825
                                                          addCursorAttr();
1826
                                                          break;
                                                  case 'd': // move to indicated row, current column
1827
                                                          remCursorAttr();
1828
                                                          cursorRow = csi_nums[csi_nums_rec-1] -1;
1829
                                                          if (cursorRow < 0)
1830
1831
                                                                   cursorRow = 0;
                                                          else if (cursorRow > ROWS - 1)
1832
                                                                   cursorRow = ROWS-1;
1833
                                                          addCursorAttr();
1834
1835
                                                          break;
                                                  case 'J':
1836
1837
                                                          // Erase display
1838
                                                          if (csi_nums[csi_nums_rec-1] == 1)
1839
                                                          { // erase from start to cursor
1840
                                                                   for(j = 0; j < cursorRow; ++j)
1841
                                                                            rowTr = (j + screenTop) % ROWS;
1842
                                                                            for(k = 0; k < COLS; ++k)
1843
1844
                                                                                    screenChars[rowTr][k] = '
1845
1846
                                                                            }
1847
1848
                                                                   for(j = 0; j < cursorCol; ++j)
1849
1850
                                                                            screenChars[rowTr+1][j] = ' ';
1851
1852
                                                          if (csi nums[csi_nums_rec-1] == 0)
1853
1854
                                                          { // erase from cursor to end
1855
                                                                   for(j = cursorRow; j < ROWS; ++j)</pre>
                                                                   {
1856
                                                                            for(k = 0; k < COLS; ++k)
1857
```

```
{
1858
                                                                                    rowTr = (j + screenTop) %
1859
      ROWS;
                                                                                    screenChars[rowTr][k] = '
1860
                                                                           }
1861
1862
                                                                   rowTr = (cursorRow + screenTop) % ROWS;
1863
1864
                                                                   for(j = 0; j < cursorCol; ++j)
1865
                                                                           screenChars[cursorRow][j] = ' ';
1866
1867
1868
1869
                                                          else if(csi_nums[csi_nums_rec-1] == 2 || csi_nums
       [csi_nums_rec-1] == 3)
1870
                                                          { // erase whole display
1871
                                                                   for(j = 0; j < ROWS; ++j)
1872
                                                                           for (k = 0; k < COLS; ++k)
1873
1874
                                                                                    screenChars[j][k] = ' ';
1875
1876
                                                                           }
                                                                  }
1877
1878
                                                          }
                                                          break:
1879
                                                 case 'K':
1880
1881
                                                          // Erase line
1882
                                                          rowTr = (cursorRow + screenTop) % ROWS;
1883
                                                          if (csi_nums[csi_nums_rec-1] == 1)
1884
                                                          { // erase from start to cursor
1885
                                                                   for(j = 0; j < cursorCol; ++j)
1886
                                                                           screenChars[rowTr][j] = ' ';
1887
1888
1889
                                                          if (csi_nums[csi_nums_rec-1] == 0)
1890
1891
                                                          { // erase from cursor to end
                                                                  for(j = cursorCol; j < COLS; ++j)</pre>
1892
1893
                                                                           screenChars[rowTr][j] = ' ';
1894
1895
1896
                                                          }
                                                          else if(csi_nums[csi_nums_rec-1] == 2 || csi_nums
1897
       [csi_nums_rec-1] == 3)
                                                          { // erase whole line
1898
1899
                                                                   for(j = 0; j < COLS; ++j)
1900
1901
                                                                           screenChars[rowTr][j] = ' ';
1902
1903
                                                          }
1904
                                                          break:
1905
                                                 case 's': // push cursor position
                                                          CSIpushCursor();
1906
1907
                                                          break;
1908
                                                 case 'u': // pop cursor position
1909
                                                          CSIpopCursor();
1910
                                                          break;
                                                 case 'n': // if we get 6, it reports cursor position
1911
1912
                                                          if (csi nums[0] == 6)
1913
                                                          {
                                                                   reportCursorToHost();
1914
1915
                                                          }
1916
1917
                                                 case 'l': // sometimes hides the cursor (ESC[?25l)
1918
                                                          if (csi nums[0] == 25 && (escStat &
      CSI QUESTION ON))
1919
                                                          {
1920
                                                                   drawCursor = 0;
1921
                                                          break;
1922
```

```
1923
                                                 case 'h': // sometimes shows the cursor (ESC[?25h)
1924
                                                          if (csi_nums[0] == 25 \&\& (escStat \&
      CSI_QUESTION_ON))
1925
                                                          {
                                                                  drawCursor = 1;
1926
                                                          }
1927
1928
                                                          break;
                                                 default:
1929
1930
                                                          break;
1931
                                        escStat &= ~(CSI ON | CSI QUESTION ON);
1932
1933
                                        csi nums rec = 0;
1934
                       } ///////// END HANDLING CSI CODES
1935
1936
                       else if(escStat & ESC_ON)
1937
1938
                                // handle escape codes
1939
                                The following is an exerpt from the "console_codes" man page.
1940
1941
                                We're not implementing all of them (Operating system command?
                                                                                                   Obviously
      that's Linux only.)
1942
1943
1944
                                ESC- but not CSI-sequences
1945
              ESC c
                         RIS
1946
                                  Reset.
1947
              ESC D
                         IND
                                  Linefeed.
1948
              ESC E
                         NEL
                                  Newline.
1949
              ESC H
                        HTS
                                  Set tab stop at current column.
1950
              ESC M
                         RI
                                  Reverse linefeed.
              ESC Z
1951
                        DECID
                                  DEC private identification. The kernel returns the
1952
                                  string ESC [ ? 6 c, claiming that it is a VT102.
              ESC 7
                        DECSC
1953
                                          current
                                                     state
                                                               (cursor
                                                                           coordinates,
1954
                                  attributes, character sets pointed at by G0, G1).
              ESC 8
                        DECRC
                                  Restore state most recently saved by ESC 7.
1955
              ESC [
                                  Control sequence introducer
1956
                         CSI
              ESC %
1957
                                  Start sequence selecting character set
              ESC % @
                                     Select default (ISO 646 / ISO 8859-1)
1958
              ESC % G
                                     Select UTF-8
1959
              ESC % 8
                                     Select UTF-8 (obsolete)
1960
              ESC # 8
                        DECALN
                                  DEC screen alignment test - fill screen with E's.
1961
                                  Start sequence defining G0 character set
              ESC (
1962
                                     Select default (ISO 8859-1 mapping)
              ESC (B
1963
              ESC ( 0
                                     Select VT100 graphics mapping
1964
              ESC ( U
                                     Select null mapping - straight to character ROM Select user mapping - the map that is loaded by
1965
              ESC (K
1966
                                     the utility mapscrn(8).
1967
1968
              ESC )
                                  Start sequence defining G1
1969
                                  (followed by one of B, 0, U, K, as above).
1970
              ESC >
                         DECPNM
                                  Set numeric keypad mode
1971
              ESC =
                         DECPAM
                                  Set application keypad mode
              ESC 1
                         0SC
1972
                                  (Should be: Operating system command) ESC ] P
                                  nrrggbb: set palette, with parameter given in 7
1973
                                  hexadecimal digits after the final P:-(. Here n
1974
                                  is the color (0-15), and rrggbb indicates the
1975
1976
                                  red/green/blue values (0-255).
                                                                      ESC ] R: reset
1977
                                  palette
1978
1979
                                switch(buf[i])
1980
1981
                                         case '[':
1982
                                                 escStat |= CSI ON;
                                                 break;
1983
                                         case 'c':
1984
                                                 // reset
1985
1986
                                                 bufClear();
1987
                                                 break;
1988
                                         case '7'
                                                 ESCpushCursor();
1989
1990
                                                 break;
```

```
1991
                                         case '8':
1992
                                                 ESCpopCursor();
1993
                                                 break;
                                         case 'D':
1994
                                         case 'E':
1995
                                                 handle_LF();
1996
1997
                                                 break;
                                         case 'M':
1998
                                                  // reverse line feed...
1999
                                                 moveCursor(1,UP);
2000
2001
                                                 break;
                                         case 'H':
2002
2003
                                                  // set tab stop at current column...
2004
2005
                                         case 'Z':
2006
                                                  // report DECID
2007
                                                 break;
2008
                                         default:
2009
                                                 break;
                                }
2010
2011
2012
                                escStat &= ~ESC ON;
2013
                        }
                       else
2014
2015
                        {
2016
                                switch(buf[i])
2017
                                {
2018
                                         // Handle control characters
                                         //00 (NUL), 07 (BEL), 08 (BS), 09 (HT), 0a (LF), 0b (VT), 0c (FF),
2019
                                              Od (CR), Oe (SO), Of (SI), 18 (CAN), 1a (SUB), 1b (ESC), 7f
2020
       (DEL)
2021
                                         // 0x9B (CSI)
2022
                                         case 0x00: // NUL
                                         break; // it's ignored.
case 0x07: // BEL
2023
2024
                                                 DAC_beep();
2025
2026
                                                 break;
                                         case 0x08: // BS
2027
                                                  // This may be tricky... we may need so implement some
2028
       sort of line discipline stuff to know how many characters the user's input since the last <return>
                                                  // It works properly if user input doesn't make it go on
2029
      to a second line... the same problem that plagues many
2030
                                                  // X terminal emulators...
                                                  if (cursorCol > 0)
2031
2032
                                                          --cursorCol;
                                                  keepCursorInBounds();
2033
2034
                                                 break;
2035
                                         case 0\times09: // HT - goes to the next tab stop or to the end of the
      line if there is no earlier tab stop
2036
                                                  break;
                                         case 0 \times 0 A: // LF - down a line.
2037
                                         case 0x0B: // VT, same as LF
2038
                                         case 0x0C: // FF, same as VT and LF
2039
                                                 handle_LF();
2040
2041
                                                  break:
2042
                                         case 0x0D: // CR - push the carriage to the left! On old
       typewriters.
                                                 handle_CR();
2043
2044
                                                 break;
2045
                                         case 0x0E: // SO - activate G1 character set
2046
                                                 break;
2047
                                         case 0x0F: // SI - activate GO character set
2048
                                                 break;
2049
                                         case 0x18: // CAN - interrupt escape sequence (not sure what it
      does)
2050
                                                  break;
2051
                                         case 0x1A: // SUB - interrupt escape sequence (not sure what it
      does)
2052
                                         case 0x1B: // ESC - start escape sequence
2053
```

```
2054
                                                    escStat |= ESC ON;
2055
                                                    break;
2056
                                           case 0x7F: // DEL - ignored
2057
                                                    break;
                                           case 0x9B: // CSI - start CSI sequence
2058
2059
                                                    escStat |= CSI_ON;
2060
                                                    break;
2061
                                           // Normal character handling!
                                           default:
2062
                                                    handle normal(buf[i]);
2063
2064
                                                    break;
2065
                                  }
                        }
2066
2067
               }
2068
2069
2070
       // usart2.c
2071
2072
      #include "usart2.h"
2073
2074
2075
2076
                ; baud rate, 0x08, 12bit mantissa plus 4 bits fractional
2077
       ; BRD = 8e6/(16*9600) = 52.083
         ; integer portion: int(52.083)=52 = 0b00110100
2078
                                                                    (0xD0 \text{ if } 32Mhz, 0x138 \text{ for } 48MHz)
         ; fractional portion: int(0.083*2^6+0.5)=6 = 0b0110
2079
                                                                    (0x32 \text{ if } 32Mhz, 0x5 \text{ for } 48MHz)
2080
          to put in register: 0b001101000110 = 0x0341
                                                                      (0 \times D03 \text{ if } 32 \text{Mhz}, 0 \times 1385 \text{ for } 48 \text{MHz})
2081
       BAUD_RATE DCD 0x0341
2082
       ; control register 1, 0x0C, bits 31-14 reserved (0), 0b10000000001100 = 0x200C
2083
                                                                                                  no interrupts, no
       break, active receiver, RX and TX enabled
2084
       USART_CTRL1_SETTINGS DCD 0x200C
2085
                SETUPA DCD 0x000A8AA8 ; Usart alt function settings -- see usart comments below
2086
2087
                ; alternate functionality pins
       ;USART2_CTS PA0 PD3
2088
                              - input pull-up (8)
2089
       ;USART2_RTS PA1 PD4
                             - alt push-pull
       ;USART2_TX PA2 PD5
;USART2_RX PA3 PD6
2090

    alt push-pull

                             - input pull up
2091
       ;USART2_CK PA4 PD7

    alt push-pull

2092
                */
2093
2094
2095
       unsigned char usart2_data[USART2_DATA_SIZE];
       int usart2_bytes_rec = 0;
2096
2097
       void usart2_init(void)
2098
2099
2100
                RCC->APB1ENR |= RCC APB1ENR USART2EN;
2101
                //GPIOA->CRL = (GPIOA->CRL & 0xFFF00000) | 0x000A8AA8;
                GPIOA \rightarrow CRL = (GPIOA \rightarrow CRL \& 0xFFF00000) | 0x000B8BB8;
2102
2103
2104
                //USART2->BRR = 0x0341;
                USART2->BRR = 0\times1385;
2105
2106
                USART2->CR1 = 0x202C; //Enable RXNEIE(bit 5) bit for receive interrupt
                NVIC->ISER[1] = NVIC_ISER_SETENA_6;
2107
2108
2109
       }
2110
2111
       void usart2_tx(unsigned char byte)
2112
               while(!(USART2->SR & USART_SR_TC));
2113
2114
                USART2->DR = byte;
2115
       }
2116
2117
2118
2119
       void USART2_IRQHandler()
2120
2121
                if (usart2_bytes_rec < USART2_DATA_SIZE)</pre>
2122
```

```
2123
                       usart2_data[usart2_bytes_rec] = USART2->DR;
                       usart2_bytes_rec++;
2124
2125
               }
2126
               NVIC->ISER[1] = NVIC_ICPR_CLRPEND_6;
2127
2128
      }
2129
      int usart2_memcpy(unsigned char * dst)
2130
      // Copies the usart2 buffer, then resets the usart2 buffer
2131
2132
2133
               int ret, i;
2134
               // turn off interrupts
2135
               NVIC->ICER[1] = NVIC_ICER_CLRENA_6;
2136
2137
2138
               // copy buffer
2139
               for(i = 0; i < usart2_bytes_rec; ++i)</pre>
2140
               {
                       dst[i] = usart2_data[i];
2141
               }
2142
               ret = usart2_bytes_rec;
2143
2144
               usart2_bytes_rec = 0;
2145
2146
               // enable interrupts
               NVIC->ISER[1] = NVIC_ISER_SETENA_6;
2147
2148
2149
               return ret;
2150
      }
```