ECE 212 – Digital Circuits II

Lab 4 – C Programming

Revised February 24, 2019

1. Introduction

In Lab 3 you developed a scrolling display given messages that were encoded as pixels and programmed in BRAM on the FPGA using the `INIT_xx` parameters of the BRAM macro instantiation. In this lab you will write a C program to generate these parameters from user input.

In this lab you will learn to:

- Write a C program in the Linux environment
- Work with arrays and character pointers in C
- Use I/O and utility functions in the C standard libraries

2. The Design Task

The goal of this lab is to create a C program that takes a character string as a command-line argument and outputs a sequence of `INIT_xx` parameters for a Xilinx BRAM that contains pixel data for the dot-matrix characters that correspond to the input string.

Specific requirements:

- The completed program will accept up to two command-line arguments:
    - a character string of an arbitrary length that is the message to be displayed on the LED matrix.
    - an optional "color mask" that specifies the color of each row of the generated characters. This mask is expressed as a 32-bit hexadecimal number where each digit corresponds to the color of one row of the generated characters. If no color mask is given, all pixels should be white.
- The program will generate a sequence of `INIT_XX` parameter assignments on standard output that can be copied and pasted into a SystemVerilog Block RAM instantiation. This sequence must not exceed the capacity of the BRAM (i.e., 1024 columns) and start in column 32 as in Lab 3.
- The output of the program must be successfully integrated into your Lab 3 project and synthesized into a working display.
- The program should check for invalid inputs (e.g., excessively long input string, improper input for color mask) and print error messages when encountered.

3. Background

3.1 Converting Characters to Pixel Data

Your program must translate ASCII characters into 8 row by 6 column dot-matrix characters using the patterns provided in Appendix A. It should then generate output in the form of `INIT_XX` parameter statements for a BRAM instantiation.

To aid in speedy completion of your program, we will provide you with a two-dimensional array that specifies the pixel values of each column of each ASCII character. However, this code may contain errors that you will need to fix.

This array is declared as follows:

```
unsigned int char_map [128][6] = {
  {0, 0, 0, 0, 0, 0 }, // 0x00 is not a printable character
  {0, 0, 0, 0, 0, 0 }, // 0x01 is not a printable character
  …
  { {0, 0, 0xf0ffffff, 0, 0, 0}, // 0x21 ('!')
  …
};
```

For a given character value c, you can look up individual column values as follows:

```
char c;
unsigned int col0, col1, …, col5;

col0 = char_map[c][0];
col1 = char_map[c][1];
…
col5 = char_map[c][5];
```

The complete array declaration with initializers for all 128 possible ASCII characters is available on the Moodle page in file `char_map.c`. A companion header file `char_map.h` contains the declaration of this data structure so that you can use it in your program by including this declaration with

```
#include "char_map.h"
```

You can then compile your program with the `char_map.c` source code with the command:

```
gcc char_map.c bram_init.c -o bram_init
```

3.2 Using Pixel Data to Generate `INIT_XX` Statements

Given the `char_map` array, the main task of your program will be to scan through the characters of a given message, covert each character to its column values, and generate `INIT_xx` messages to configure these characters in the BRAM.

Note that each `INIT_XX` statement specifies the values of 256 bits; since we are working with 32 bits in each column this corresponds to 8 columns of pixel data. You may find it useful to declare an array to store the data for one of these statements as follows:

```
unsigned int init_buf[8];
```

You can then partition your program into two pieces – one which reads characters and places column data into `init_buf`, and another which writes a complete `INIT_XX` statement when `init_buf` has been completely filled.

4. Prelab

There is no formal prelab for this week. However, you should prepare for lab reading this handout, the "Linux Tutorial" handout distributed in class, and Appendix C of Harris & Harris, especially Section C.9 (Standard Libraries) and C.10 (Compiler and Command Line Options).

4. In the Lab

1. If the PC at your lab station is not already running Linux, reboot it into Linux.
2. Log in to Linux (use your standard ID and password) and open a shell window.
3. Clone your ECE 212 Git Repository into your main directory (note that the commands are the same as you use in the "Git Bash" shell on Windows.
4. Create a new subdirectory within your repository named `Lab04`. You will not need to create any subdirectories.
5. Download the files `char_map.c` and `char_map.h` from the Moodle page into your `Lab04` directory.
6. Create a new file named `bram_init.c` using a text editor of your choice. This file should include your main function and any other functions that you may need to write in order to complete your program. Note that you will need to use the `#include` compiler directive to include declarations for the `char_map` data structure and several different library files. These directives will look something like this:
   ```
   #include <stdlib.h>
   #include <stdio.h>
   #include <string.h> // you may not need this
   #include "char_map.h"
   ```
7. After you have completed coding, compile your program with the command:
   ```
   gcc char_map.c bram_init.c -o bram_init
   ```
8. Once you have worked through the syntax errors, this will create an executable file for the program. Run the program as follows:
   ```
   ./bram_init "test message" 0x44662211
   ```
   Test your program with a variety of inputs and make sure that it generates a plausible output. Demonstrate you successfully operating program to the instructor before proceeding.
9. Copy and paste the output of your program into your Lab 3 project, replacing the previously displayed messages with messages of your own choice. Synthesize your system using Vivado and verify that it correctly displays the desired text messages. Demonstrate your successfully operating LED Matrix to the instructor.
10. Commit the C files (i.e. "`.c`" and "`.h`" extensions) to your Git repository and push your repository back to the GitLab server.

5. Report

You must submit an electronic copy of your lab report in PDF format on Moodle. The name of your PDF file should be of the form

"`Lab04_Report_LastName1_LastName2.pdf`"

Be sure to label each section and organize them in the order given. Messy or disorganized labs will lose points.

Each lab report should have a "scribe" (i.e., the person who writes most of the report). The scribe should alternate every lab. Please clearly indicate in the title block of your report the names of the lab group and the name of the scribe.

Your report should include the following sections:

1. **Introduction** – write a brief paragraph describing what you did in the lab. This should be a summary written in your own words; it is not acceptable to copy text for this from the lab handout. For this lab, describe the general strategy that your program takes to successfully met the lab requirements.
2. **Results** – Include the results specified in the "In the Lab" section including the boldfaced label (if any). Results in this lab include a listing of your final C program formatted single-space in a fixed-width font such as `Courier New`.
3. **Conclusion** – Briefly describe any difficulties that you encountered and how you resolved them. Also, if you have any suggestions for how to improve this lab, please include them here.
4. **Time Spent on Lab** – Please indicate how many hours you spent on this lab (including time during the lab period). This will not affect your grade, but will be helpful for calibrating the workload for future offerings of ECE 211.
5. **Suggestions for Improvement** – if you have any feedback about how this lab might be improved, please include it here.

Each section should begin with a section heading that includes the section number and title in a boldface font as in the report for Lab 1. Be sure to label each section and organize them in the order given. Messy or disorganized labs will lose points.

Appendix – Dot-Matrix Character Patterns