Inputs, Outputs, and Time

EE/CSE 474 Lab 2

Daisy Xu, Yunie Yi, Abigail Santos



October 21, 2016

**Table of Contents:**

1 [Introduction](#_1fob9te)

2 [Discussion Of the Lab](#_3znysh7)

[2.1 Talking to the Oscilloscope](#_2et92p0)

[2.2 Designing and Building the LCD Interface](#_2et92p0)

[2.3 Implementing the Pipe](#_2et92p0)

[2.4 Designing and Building the Memorization Game](#_2et92p0)

3 [Test Procedure & Cases](#_lnxbz9)

3[.1](#_35nkun2) [Talking to the Oscilloscope](#_2et92p0)

3[.2 Initialization](#_1ksv4uv)

3[.3 Pipe](#_44sinio)

3[.4 Memorization Game](#_44sinio)

4 [Results](#_z337ya)

4[.1](#_35nkun2) [Talking to the Oscilloscope](#_2et92p0)

4[.2 Initialization](#_1ksv4uv)

4[.3 Pipe](#_44sinio)

4[.4 Memorization Game](#_44sinio)

5 [Error Analysis](#_1ci93xb)

6 [Summary and Conclusion](#_qsh70q)

# **1. Introduction**

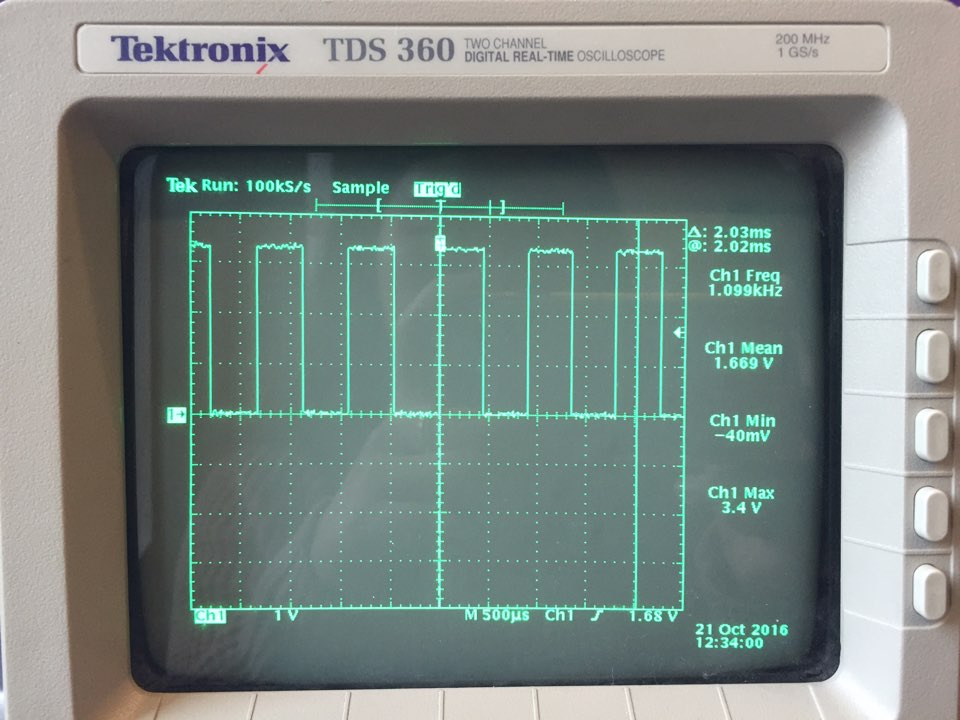
# The purpose of this laboratory was to learn/familiarize ourselves with the beaglebone, how it interacts with an LCD display, and how the internal scheduler operates. There were three core parts to this lab. In the first part, we had to correctly execute a particular initialization sequence to allow the LCD to read and display data. In the second part, we had to integrate a pipe with our LCD Interface code, which would accept text and push it onto the display. Finally, after establishing this basic functionality, we built a memorization game.

The game starts by lighting up three unique spaces on the LCD in succession before disappearing. The user has to memorize the position of the three spaces for the brief moments that they appear. Then, controlling the cursor, the user must find and select the positions in the correct order to win the game. The cursor is controlled by four keys: ‘a’ (left), ‘s’ (down), ‘d’ (right), ‘w’ (up). User selection is confirmed with the ‘y’ key. If the player manages to correctly guess all three locations, a ‘hard’ mode will begin, in which the player must play again, but this time memorize five locations. If this hard mode is beaten, the player wins the game.

# **2. Discussion Of the lab & Specifications**

## **2.1 Talking to the Oscilloscope**

This part is to check if our gpio ports to control LCD functions works properly in C. The oscilloscope displays pulse waveform when we execute the program osc.c. Within the infinite while loop, we generate a signal and off it with a usleep(300) as its time delay between on and off function. **Figure 1** belows shows on and off of our gipo49 which is Enable pin of the LCD.

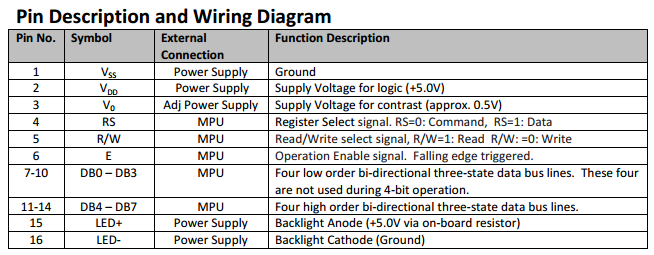


**Figure 1: Enable Pin Output Waveforms**

When you run it, you can observe some variation or “jitter” in the pulse width and that the peak of the square pulse waveform moves slightly back and forth. This is because running the Linux OS on the board means other programs are running in the background. Even if the timing is perfect in our C code, timing will not be perfect because it is dependant on the execution timing imperfections of the background programs.

## **2.2 Designing and Building the LCD Interface**

Before we could initialize the LCD, we first needed to connect the LCD with the beaglebone GPIO pins. The LCD has the following pins:



**Figure 2: Pin Description of NHD LCD Display**

We then connected each of these pins to the ports on the Beaglebone. The connections are shown in Table 1 below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **LED Pin** | **GPIO Pin** |  | **LED Pin** | **GPIO Pin** |
| 1: Ground | Digital Ground |  | 9: DB2 | GPIO 112 |
| 2: Power Supply | SYS 5V |  | 10: DB3 | GPIO 20 |
| 3: Contrast Power Supply | Digital Ground w/1kOhm Resistor |  | 11: DB4 | GPIO 66 |
| 4: Register Select (RS) | GPIO 61 |  | 12: DB5 | GPIO 69 |
| 5: Read/Write (RW) | GPIO 65 |  | 13: DB6 | GPIO 45 |
| 6: Enable (E) | GPIO 49 |  | 14: DB7 | GPIO 47 |
| 7: DB0 | GPIO 48 |  | 15: Backlight Power Supply | SYS 5V |
| 8: DB1 | GPIO 60 |  | 16: Backlight Ground | Digital Ground |

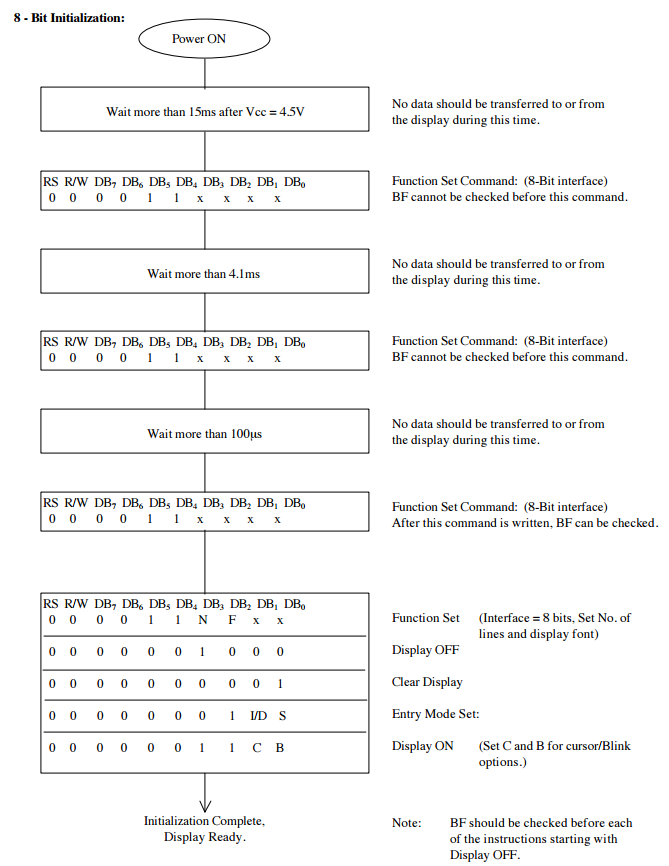
**Table 1: LCD and GPIO Connections**

After wiring the LCD and GPIO ports together we began to code a C file for interfacing. The code was structured to follow the 8-bit initialization procedure as outlined in the LCD data sheet, as seen on the next page in **Figure 2**. Delays between steps are necessary to ensure that the system only has to execute one command at a time. We used much larger delays than necessary within our code to account for any possible timing mishaps.

The ports themselves (RS, RW, E, DB0-7) were controlled by directly opening and writing to the appropriate files of the GPIO pins of which they were connected to. Each GPIO Direction was set to Output, and we controlled the Values were controlled to turn the pins on and off as necessary.  
   
 Each of the instruction calls, e.g. Function Set, Display Off, were executed in one method within our code. This method was designed to set the data registers as commanded, turn on the Enable signal, delay to avoid timing conflicts, then turn off the Enable signal. A complete summary of methods used in this code is found below in **Table 2**.

|  |  |
| --- | --- |
| **Method Name** | **Method Description** |
| initializeAll() | The main initialization method in which other smaller methods are called. |
| initGPIO() | Turns on GPIO ports, setting the Direction to “Out” and creating pointers to control GPIO Values. |
| checkGPIO() | Verifies GPIOs were all initialized correctly. |
| zeroOut() | Sets all GPIO ports to 0. |
| setRSRW(int x, int y) | Sets RS and R/W to desired Value. |
| functionSet(char c) | Sets DB7-DB0 to the given command. |

**Table 2: Summary of Initialization Methods**



**Figure 3: 8-Bit Initialization Procedure of NHD LCD Display**

**2.3 Implementing the Pipe**

To communicate with a user interface, the program needs a method to allow the user to read and write data from the LCD. In Linux, the mkfifo() function allows us to do this. The C code file which contains mkfifo write function, called pipe.c in our project, can write data into the 8 bit data bus of our LCD instruction (DB0-DB7). Since the 8-bits data bus indicates one character, this allows us to write and read a single character at a time. This is accomplished with the scan() function and the write() function writing to a character buffer.

The pipe.c file takes input from Linux terminal when the user enters in characters, and writes and stores the characters to the LCD. Thus two programs, our pipe.c and interfacing code, can work independently yet still be connected to allow the user to send whatever input the user desires in real time.

## **2.4 Designing and Building the Memorization Game**

After establishing core functionality in our LCD interface and pipe program, we were able to develop a memorization game. The game works by lighting up a randomized 3 spots anywhere on the screen in brief succession. The goal of the game is to memorize, in order, the position in which those places lit up. The user must then use the cursor, controlled by keys, to select each spot. If all three spots were correctly found, a “hard mode” is started in which the player must instead memorize 5 spots.

The cursor position is controlled by four keys which control four methods as indicated below in **Table 2**. The configuration is likely familiar to gamers.

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Method Description** | **Corresponding Key Input** |
| moveUp(int place) | Changes user’s place (current location on the LCD). Will move cursor up if it does not exceed LCD boundaries. | ‘w’ |
| moveDown(int place) | Changes user’s place (current location on the LCD). Will move cursor down if it does not exceed LCD boundaries. | ‘s’ |
| moveRight(int place) | Changes user’s place (current location on the LCD). Will move cursor right if it does not exceed LCD boundaries. | ‘d’ |
| moveLeft(int place) | Changes user’s place (current location on the LCD). Will move cursor left if it does not exceed LCD boundaries. | ‘a’ |

**Table 3: Cursor Method Configuration**

Each key must be entered into the terminal one at a time, pressing enter each time. Once the player has moved the cursor to the desired location, the user must enter ‘y’ to confirm the location guess.

The actual C code was based on the LCD Interface code, so it worked in tandem with our pipe.c code. Thus the methods used in the Interface were also used for the game. However, additional methods were built on top of this. These methods are summarized below in **Table 4.**

|  |  |
| --- | --- |
| **Method Name** | **Method Description** |
| showSpots(int difficulty) | Generates and displays random spots on the LCD which the player must memorize. If difficulty = 0, generates 3 spots. If difficulty = 1, generates 5 spots. |
| closePort() | Closes GPIO ports. Called at the end of the game. |

**Table 4: Summary of Game Methods**

Most of the code that controlled the game was written in the main method. The main method also contains the logic for reading the user commands, checking if the user successfully guessed or not, and initializing easy/hard modes.

# **3 Test Procedure & Cases**

## **3.1 Talking to the Oscilloscope**

To display on/off signal of one GPIO, a probe for the oscilloscope should be connected into the ground and GPIO 49 port(indicated as lcdPort[2] based on our gpio declaration in its header file). The GIPO 49 is activated in the infinite while loop to keep generating the on/off signals and observe any characteristics. The time delay used between on/off function is usleep(300) which is 300ns in real time.

## **3.2 Initialization**

Three tests were performed in initialization:

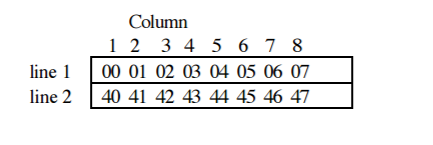
**Case 1: Signal Verification w/ LED Connected to each GPIO Port.**

We first tried to check if our code works fine through the beaglebone. Based on the Figure 2 diagram in 2.2, the LCD instruction ended functionSet(0x0e) indicating DB 3 = 1, DB 2 = 1, DB 1 = 1, and the rest of the ports are all equal to zero. After connecting LED anode with GPIO connected to DB3, DB2, and DB1 pins of the LCD, LED lid up at GPIO20, GPIO112, and GPIO60.

**Case 2: Checking Cat Value in Linux Terminal**

In our lcd.h header, lcdPort[11] array contains each unique pin of the LCD; therefore, the cat value of each GPIO from a certain instruction should be the same as what funcionSet() indicates if the array pointer works correctly.

**Case 3: LCD Display Blinking**

**Figure4: LCD spots of 8-bits display** 

The first 0 spot of line 1 and Column 1 should have a cursor after we execute the program completely.

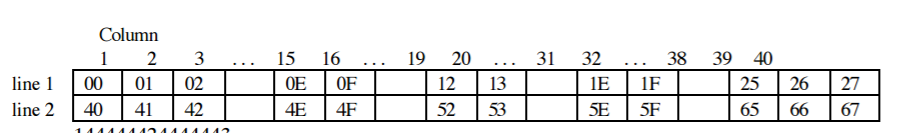
## **3.3 Pipe**

**Case 1: Checking FIFO after W/R Operation**

Since we use a pipe to access a stored data to read and display in LCD, we printed the character sent after the read command of mkfifo. Therefore, every time we type a character in the Linux terminal of pipe.c, the lcd.c terminal should print out the corresponding character stored in the character buffer.

## **3.4 Memorization Game**

**Case 1: Moving Cursors Out of the LCD Boundary**



**Figure 5 : LCD matrix indicating 40 characters in each row**

Since our setup displays only 16 character in each row, we have total 32 spots in our game. However direction methods needs a lot of shifting to get to the next spot, we make sure if the cursor spot should be in a range of 0 - 15 and 40 to 55 of the Figure. If users try to move the cursor beyond the boundary; for example, moveUP() method is used when the cursor spot is 00, it should stay in the current spot. All direction methods(moveUp(), moveDown(), moveLeft(), and moveRight() ) have boundary conditions described in **Table 5.**

|  |  |
| --- | --- |
| **Method Name** | **Execution Condition** |
| moveUp(int place) | place - 40 >= 0 |
| moveDown(int place) | place + 80 < 0 |
| moveRight(int place) | place != 15 && place != 55 |
| moveLeft(int place) | place != 40 && place != 0 |

**Table 5: Methods and Conditions Used for Cursor Direction in C**

**Case 2: Invalid Inputs**

We also checked how the game would perform when more than one character was entered at a time. Because we designed our code to only read the first character in the buffer, anything beyond one character was ignored.

In the instance where a wrong key (i.e. a key that was not ‘a’, ‘s’, ‘d’, ‘w’, or ‘y’) was entered, the program did nothing, just as designed.

# **4. Results**

## **4.1 Talking to the Oscilloscope**

The testing of the Enable pin was successful. The waveforms on the oscilloscope clearly reflected the pulses of Enable as it was turned on and off within our C code.

## **4.2 Initialization**

The initialization process was executed as intended. The LCD correctly cleared the display and turned on the blinking cursor at the head of the first line.

## **4.3 Pipe**

The incorporation of the pipe with our LCD interface was successful. Whatever was typed and entered into the terminal correctly was written onto the LCD display. Overflow text on the first line continued onto the second, and vice versa, such that no text was ever lost.

## **4.4 Memorization Game**

The memorization game was also a success. The random spots at the start of easy and hard mode were correctly displayed. The cursor was fully functional. If the user chose a wrong spot at any time, the game ended with a display message, “YOU SUCK”. If easy mode was successfully cleared, “LVL DONE” displayed. If, after easy mode, hard mode was cleared, “GG BRUH” was displayed as an indication of victory.

# **5. Error Analysis**

## Several issues arose throughout the development of this project. Initialization was particularly difficult. Our code seemed correct, a new LCD monitor didn’t change anything, and the GPIO values were being correctly written according to our LED testing. Even after the TAs and Professor evaluations, the source of the problem was still unknown. However, after changing out the Beaglebone for a new one, initialization finally ran successfully. Why exactly the board was malfunctioning is unknown, but at least we were able to fix the problem.

Creating the game also had problems. Moving the cursor from the end of line 1 to the beginning of line 2 was not working, despite the fact that we had this function working in our interface code. After debugging, we discovered that we had changed the font size to make it smaller - this meant that the gap in the address locations of line 1 and 2 had changed. So, after shifting the cursor over a few more spaces during the transition, we eventually made the cursor fully functional.

# **6. Summary and Conclusion**

Through this project, we designed a game application based on the basic LCD functions with beaglebone GPIOs and pipe communication user interface. We tested our C programs in physical boards and all of our execution displayed correctly to run the game. Our results showed that our pipe.c can fulfill use interface which is required to communicate between user and LCD in beaglebone. Memorization game and initialization of LCD programs are all written in C and could be executed in Linux terminal. With this project, we got to familiarize ourselves with C in Linux OS, basic commands to run the beaglebone specifications.