Lab 1 - Introducing the Lab and Development Environment Report

Authors: **Long Nguyen and Chase Arline**
ECE/CSE 474, Embedded Systems
University of Washington – Dept. of Electrical and Computer Engineering

Date: 18th January 2020

TABLE OF CONTENTS

1.0 ABSTRACT

The goal of this project is to create an application that detects the user input and flashes text on a display.

2.0 INTRODUCTION

The purpose of this lab is to get familiar with the C programming language and the Arduino IDE by making simple applications in C, porting them into the Arduino IDE and running the application using the Arduino Mega 2860 which displays the results to the TFT screen.

3.0 DESIGN SPECIFICATION

This Arduino application will perform the flashing "A B C D" string with the refresh rate being input by the user. This delay is able to be changed at any time through the user input. This application will only support user input via the Arduino Serial Port.
The hardware of this application consists of 2 main parts : the Arduino Mega R3 microcontroller which is the computational core of the project and the TFT screen which will be displaying output which is the flashing string "A B C D" with black background. There is no need for wiring and using the breadboard to connect those parts together since the TFT display shield will be stacked right on the Arduino Mega R3. The TFT display used is the Elegoo 2.8-inch TFT screen with the ILI9341 driver.
The software will be written in C language using CodeBlock IDE then it will be ported to Arduino Language using Arduino IDE.

4.0 SOFTWARE IMPLEMENTATION

1) Troubleshooting the snippet of codes in project1b-2021.c file:

The error from the old version is that the array of characters is declared as length of 5. However, we want to print 6 characters for A to F. So, when we start to fill the arrays with 6 characters (A to F), it will not report any errors since C doesn't have out of bound checking, this allows you to write to the index that is out of bound. However, the out of bound index that you write will not guarantee to hold the value you put in since it is not registered at the

declaration of the array. Thus, later it might be used to store different values by computer. That is the reason why when you try to print the value at index 5 of arrays (which is out of bound), it will print a seemingly random character that is different from the one that is stored by the program. That is the bug in the snippet of codes above.
In order to fix it, we can just simply declare the array as length of 6, which means now the index 5 will be registered for holding a character and will not be used for other purposes by the computer.

2) Troubleshooting the snippet of codes in project1c-2021.c file:

The error in the snippet code in the project1c is that we assigned the valuePtr to a local variable stored on the stack, i.e., tempValue. Since we don't backup the address stored in valuePtr before reassigning it to a new address, when we exit the getData function, the local variable (tempValue) will disappear, and we will never be able to retrieve the value from that variable anymore. Thus, when we return to the main function, and we want to print it one more, we cannot print the value stored at the address stored in valuePtr anymore. In order to fix it, we can just simply delete the local variable and the line that we reassign the value in the valuePtr.

3) Application of flashing "A B C D" characters with delay amount specified by users input from Arduino Serial Port:

The first procedure is to obtain the user specified delay measured in milliseconds. This is done through the serial monitor. This value is also used to create a "good delay" which is the user-specified delay minus the delay that occurs when parsing the user input for new delay. This parsing delay is hard coded as a constant literal and can be set when programming. The software prints the characters on the screen, then delays for the user specified delay. It then clears the screen and delays for the user specified delay minus the parsing delay (otherwise it would be realistically delaying for user specified delay + parsing delay). It then parses the user input for a new delay and moves on if one is not entered in the serial monitor. This then loops back to the step of printing the characters on the display and continues from then.

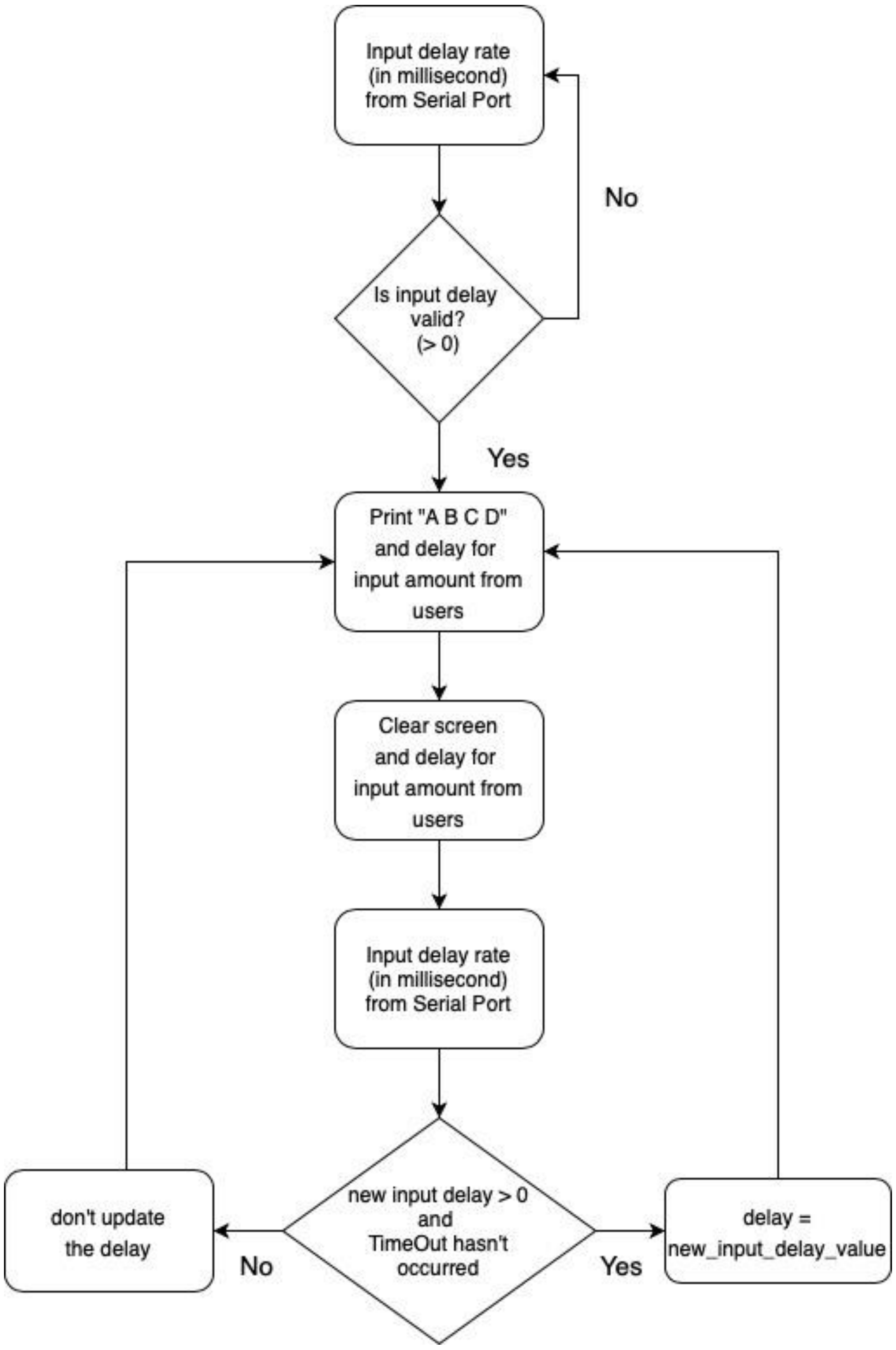

Input delay rate (in millisecond) from Serial Port
Is input delay valid? (> 0)
No
Yes
Print "A B C D" and delay for input amount from users
Clear screen and delay for input amount from users
Input delay rate (in millisecond) from Serial Port
new input delay > 0 and TimeOut hasn't occurred
No
don't update the delay
Yes
delay = new_input_delay_value

Figure 4.1 Diagram represents how the program works

The delay is slightly added on by the time spent clearing the display and printing characters in the printDelay function. This was effectively removed by implementing this pseudocode:

```
printDelay:
    start_timer
    clear screen, print characters, move cursor
    end_timer
    newDelay ← start_timer - end_timer
    delay for the new number of milliseconds
```

5.0 TEST PLAN (20PTS)

The test plan includes three parts:

1. Requirements
2. Test coverage
3. Test cases

5.1 Requirements

The Arduino will print characters onto the display and they will stay for a user specified delay. It will then clear the screen and stay cleared for the user specified delay. It will continue this cycle. The user specified delay is set at any point through the serial monitor. The display is connected as a PCB shield into the Arduino Mega R3 pins. The input is fed through the serial monitor. The display should flash the characters as fast as the hardware allows.

5.2 Test Coverage

The display must be able to flash the characters at specified delay. This test limit is until the delay is as small as the parsing delay, at which point the delay is not able to keep up (default is 20 milliseconds). The high limit is ~2,147,483 seconds as the delay in milliseconds is kept as a signed long variable. After this they will be turned to a minimum delay. Negative delay will not be accepted and will not change the current delay.

5.3 Test Cases

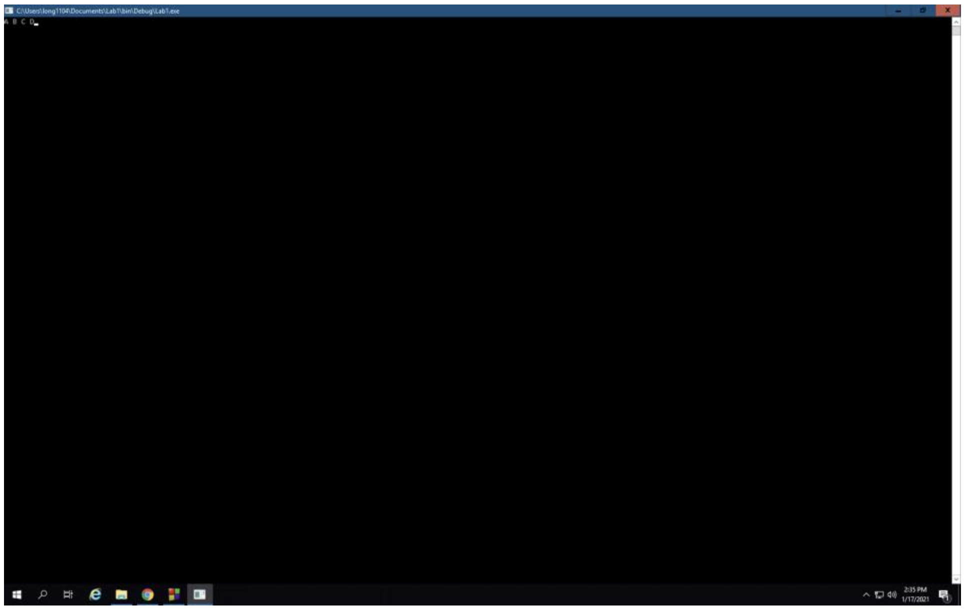
The Arduino will print out the time taken for each action of printing and clearing the characters (including parsing delay as it adds to the clearing time). A negative delay will not be accepted. A delay above the maximum of 2,147,483 seconds will turn to the minimum delay.
This will be measured using the millis() function in Arduino and printed to the serial monitor.

6.0 PRESENTATION, DISCUSSION, AND ANALYSIS OF THE RESULTS (10PTS)

The minimum achievable delay was 240 milliseconds due to how long the functions took to run. The specific function call that takes this long is to fill the screen black. Specifically this was measured to be about 218 milliseconds for the single function call. This was the only part of the firmware that had a very significant delay other than the expected delays, and it was not removable. The other was a general function delay from printing the characters that takes about 20 milliseconds which added onto the printing of the characters. When inputting a delay longer than 2,147,483 seconds it turned to the minimum delay. Also negative values were not accepted. The serial output of the delay is tracked in the serial monitor for each important action.
Overall, when the user-delay is in a reasonable range (>220+timeout_delay & >220 +time it takes to print characters; lower than the maximum number of seconds) it is accurate within 1-2 milliseconds for each period of flashing the characters.

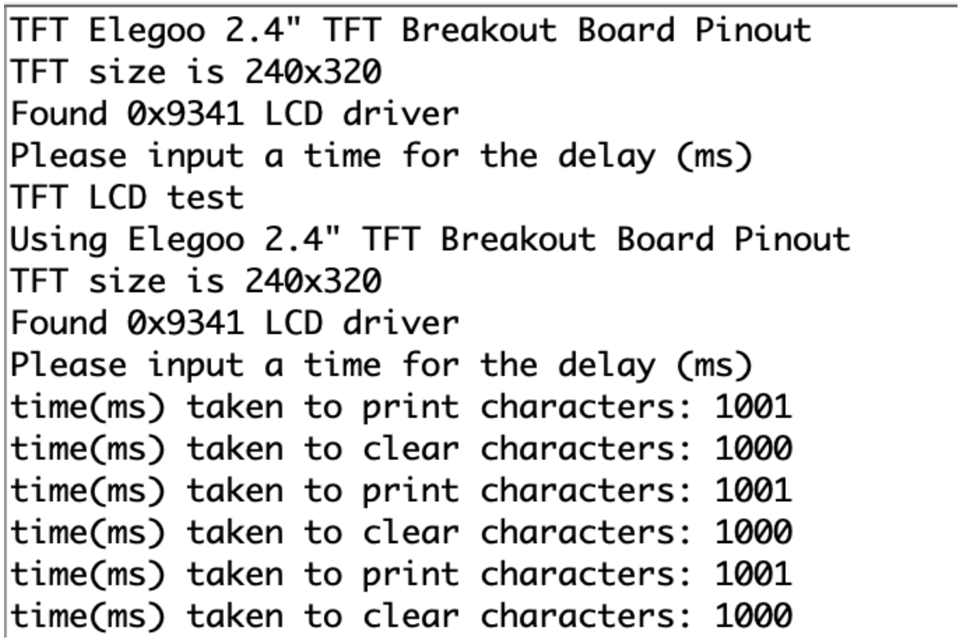
Our design did not entirely work but is mostly due to the time it takes to print characters and how long it takes to clear the display's screen.

The screenshots below are the result of running application on CodeBlock and Arduino:



```
C:\Users\long1104\Documents\Lab1\bin\Debug\Lab1.exe
A B C D
```

Figure 6.1 The result of string A B C D flashing on PC



```
TFT Elegoo 2.4" TFT Breakout Board Pinout
TFT size is 240x320
Found 0x9341 LCD driver
Please input a time for the delay (ms)
TFT LCD test
Using Elegoo 2.4" TFT Breakout Board Pinout
TFT size is 240x320
Found 0x9341 LCD driver
Please input a time for the delay (ms)
time(ms) taken to print characters: 1001
time(ms) taken to clear characters: 1000
time(ms) taken to print characters: 1001
time(ms) taken to clear characters: 1000
time(ms) taken to print characters: 1001
time(ms) taken to clear characters: 1000
```

Figure 6.2 The Arduino Serial Port Prompting an Input for delay

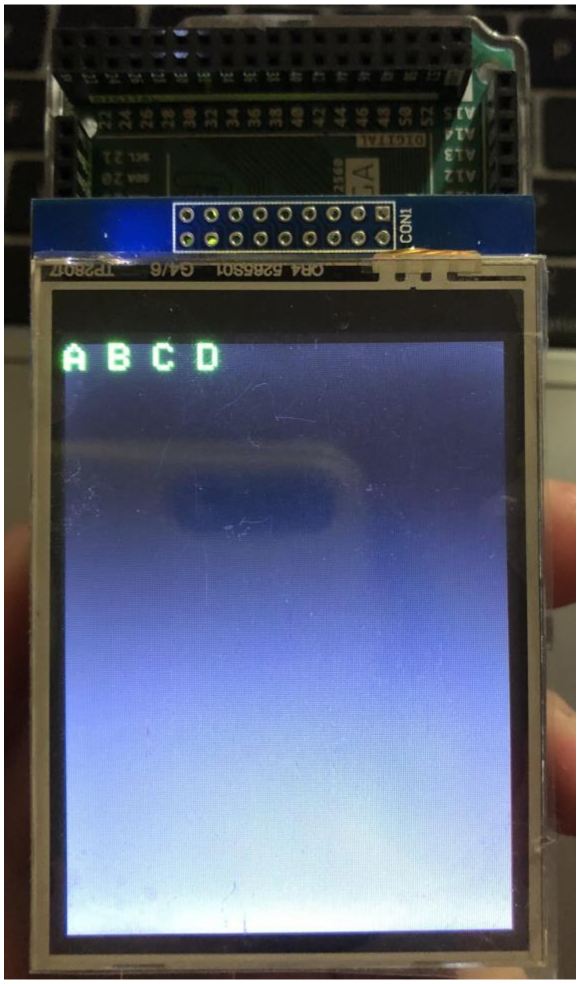


Figure 6.3 The characters “A B C D” flashing on TFT screen

6.1 Analysis of Any Resolved Errors

There were delays inside the printDelay function we made such as clearing the screen or printing characters. These were being added onto the delay given by the user as a parameter. This was fixed by subtracting the amount of time from delays we can’t control from the user parameter before delaying. This effectively delayed the screen output for the time given by the user.
This does not account for the delay given by the Serial.parseInt() function and so we have to subtract the timeout delay from the printDelay call that it follows. This effectively negates the timeout delay for the user.

This looks like:

```
        userDelay = input()
```

printDelay(userDelay)
printDelay(userDelay-timeoutDelay)
{timeoutDelay}
which effectively cancels out the timeoutDelay so that the time for printDelay is equal

6.2 Analysis of Any Unresolved Errors

The biggest issue is how slow the ILI9341 clears the display output. It takes ~ 220 ms every time it is called which limits the rate at which the text can flash on the screen and seems very slow for an embedded application. This would not be resolved unless you replaced the ILI9341 driver with a different LCD driver or, somehow, ran the driver in parallel so that you could interface with the display quicker.

The minimum period for the screen to be flashing characters and checking user interface is ~220ms + the timeout delay from Serial.parseInt() or 220ms + how long it takes to print characters on the display
With a 20 millisecond timeout this allows for a minimum user delay of ~240 milliseconds. Any user input below that will have noticeable artifacts as the display hardware cannot keep up.

7.0 QUESTIONS (20PTS)

No

8.0 CONCLUSION (5PTS)

In conclusion the display is able to flash the characters to a user specified rate with good accuracy. This is true as long as the delay is at a large enough value that the display hardware can keep up (about 250-300 milliseconds). This delay could most likely be lowered if the display used different hardware, the microcontroller ran faster, or the display was somehow run with a more parallel process so it could receive more data each cycle. It would be possible to, instead of clearing the screen entirely, to write the text over itself with the background color set to the text color. This would speed up the process but only works in a very particular problem like this where you are not changing the text being written ever.

9.0 CONTRIBUTIONS (5PTS)

We are both equally working on this project. However, Long works more on the C program, while Chase works more on Arduino and tests on Arduino since Long had problems with connecting the Arduino to the MacBook as first. However, later Long's still able to test and run the program on the Arduino. For the lab report, we both equally work on writing the lab report.

10.0 APPENDICES (5PTS)

Pseudo code for the overall project running on Arduino:

```
loop:
    prompt user input
    wait for valid user input:
        valid_delay (in milliseconds) ← user's input
    while (true)
        print "A B C D"
        delay(valid_delay)
        clear screen
        delay(input_delay)
        new_user_input ← user's input
        if new_user_input > 0:
            valid_delay = new_user_input
```