**EGRE 364 – Microcomputer System**

**Laboratory Number 6**

**Serial Communication**

**Lab Section: Tuesday 1PM**

***Lab conducted on: October 29th , 2019***

***Report Submitted on: November 5th , 2018***

**Papa Beye & Omar Amr**

**Major: CPE & CPE**

**Introduction**

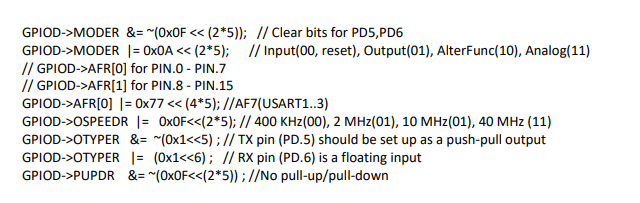
The purpose of this lab was to build on the concepts of the use of multiple GPIO input and output configurations to detect inputs and transmit data across Serial communications using Assembly in the Keil µVision software development environment. To achieve this objective, we used input/output registers of push pull, open drain, pull up/down, ODR, IDR and the USART registers. The objective was to program the microcontroller to correctly parse the inputs from the keypad to a host computer via serial communication and view the correct character. We used a virtual serial port to perform the view the data transferred from microcontroller USART. We used branches and mnemonics such as LDR, STR and BIC. To scan for the button that would be pressed on the keypad, we used two ports for the keypad, GPIO Port A for inputs and Port E for outputs. The rows were set as output and the columns were set as inputs to allow us to setup the appropriate algorithm to determine which value was pressed. We were able to initialize each GPIO port and manipulate the appropriate ODR and IDR values to determine which button, thus allowing us to send the correct ASCII value to the Transfer Data Register (TDR) and showing up correctly on the virtual serial port on the host computer. In order to perform USART communication, we must configuration the TX and RX as AF mode. Our program configures USART2, PD5 as TX and PD6 as RX. The configuration used for this lab can be see below in the C code. We implemented a delay function by utilizing a loop counts down from a given number and we used one delay that would loop back to the start of the algorithm and one that cause a delay and return to return address with BX LR. In this lab, we used the practiced methods and material that we learned in class about Serial communication and overall Assembly processes from Lab 4.

**Functionality and Correctness**

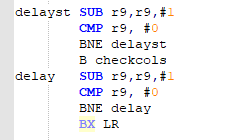
The STM32L4 discovery board has the choice for serial communication with the USART. In this lab, our circuit consisted of 4 pins that

we used the up and down push buttons, PA3 and PA5 and the left and right push buttons, PA1 and PA2. Since the inputs’ buttons don’t have an internal pull-down resistor, we must initialize the pins as pull down. The assembly code below shows how we were able to delay and return to back to previous line of instruction using BX LR. The left and right buttons, PA3 and PA5, were implemented in our Assembly program by checking if they were pressed and if they were, they would either increase the delay value, thus slow down the motor speed, or decrease the delay value, thus speed up the motor. We set upper and lower bounds for the delay value because if the delay is too low or too high, the motor will not turn at all. Using the sequence table, we created two subroutines that looped that set the appropriate ODR bit(s) of GPIO port E. We checked if any button was pressed, and if they were pressed, it would branch to correct subroutine, otherwise it would continue to loop. We controlled the stepper motor via the H-Bridge because we cannot directly a motor by using the GPIO pins due to the low current. In our code, we had to initialize the clock and each GPIO port. We needed to implement two sequences of full stepping and half stepping while also polling/checking for input from the joystick (Up, Down, Left, Right).

**C Code to configure USART (PD5 as TX and PD6 as RX)**



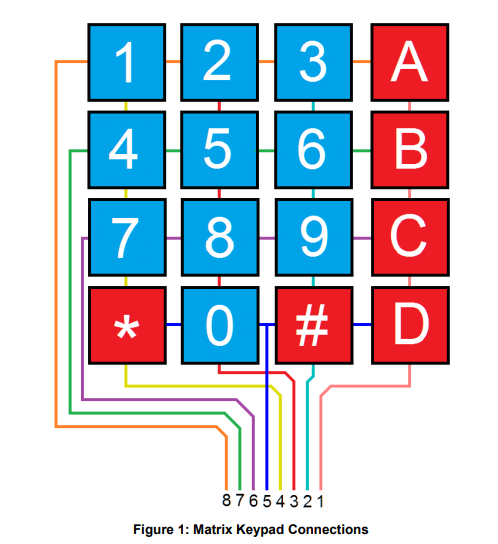
**Delay Code used in Assembly program**

****

**Connection configuration used in circuit built for programmed Algorithm**

**Pins 1- 4 : Columns (Input)**

**Pins 5 - 8 : Rows (Output)**

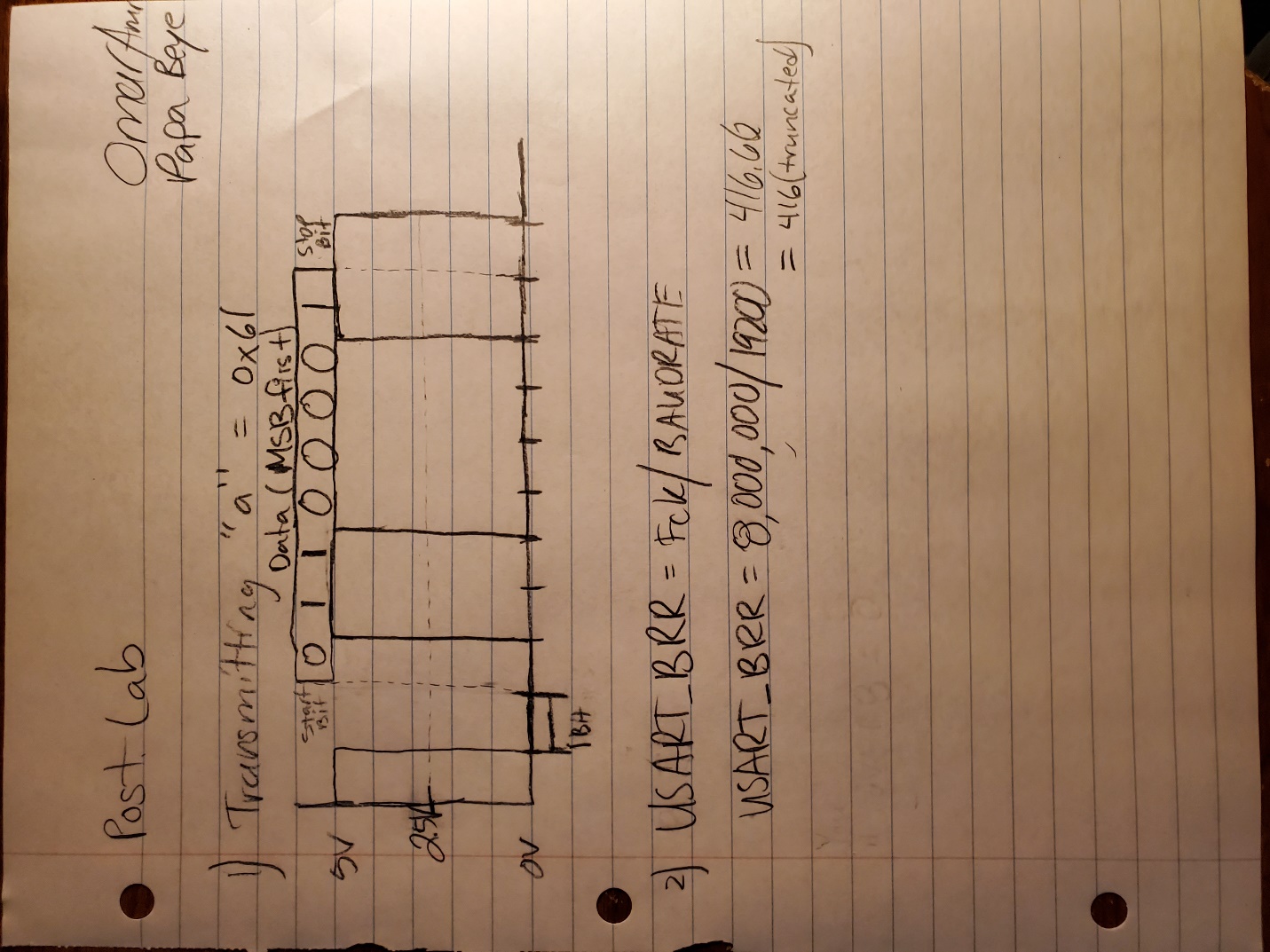
****

**Joystick circuit for each button of input used in Assembly program**

**Pre-Lab included below**

Omar Amr Papa Beye

**Post-Lab**



**Conclusion**

In this lab, we became more comfortable with interfacing a motor, implementing multiple inputs, and creating an algorithm of processes and inputs to complete our lab objective. We learned about checking the IDR value of a multiples pin and correct initializing pins for our objective. We were able to achieve the objective of the stepper motor with speed control and half-step and full-step implementation control using the joystick buttons. We were able to achieve the objective of the up-down counter using the joystick and 7-segment display with proper software debouncing. We used branches and CMP statements to be able to test and run routines to receive inputs from the joystick buttons and alter the sequence stepping or the speed of the motor. We learned a lot about how to use branches (BL and BX LR) and Assembly mnemonics and labels to initialize ports, access registers and reading inputs to complete a task. We didn’t face any issues but there was a learning curve that allowed us to learn how to create and use Assembly branches to properly access and manipulate bit values of interest without altering other bits and reading inputs through the IDR and properly polling and checking for inputs from the joystick.