**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**

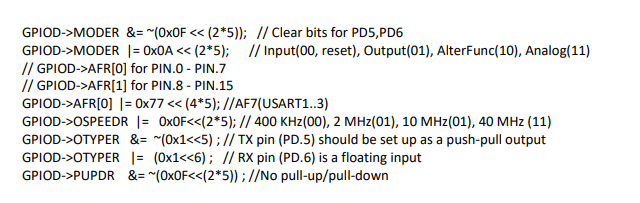
The purpose of this lab was to build on the concepts of the use of multiple GPIO input and output configurations to detect inputs and transmit data across Serial communications using Assembly in the Keil µVision software development environment. To achieve this objective, we used input/output registers 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**

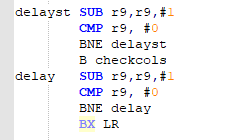
The STM32L4 discovery board has the choice for serial communication with the USART. In this lab, our circuit consisted of 4 pins for input from the keypad connected to Port A and 4 pins for output from the keyboard connected to Port E. The inputs pins were also connected to pull – down resistor to allow us to detect a change in the value of the input of Port A pins when a button is pressed. Our algorithm can detect what character is pressed. We started with detecting if a button was pressed by setting the rows (outputs) to 1 and checks the inputs for a change from 0 to 1, showing that the button pressed is on that column.

After the keypad determines which button is pressed, we are to load the ASCII byte equivalent of the specific character that has been pressed and write to the TDR and send the byte of data. In order to send the byte of data, we had to first check that the USART\_ISR is equal to the value of USART\_ISE\_TXE to make sure it is empty, allowing us to set the TDR with the value of the ASCII value, otherwise it waits until it is empty and satisfies the condition of being empty.

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



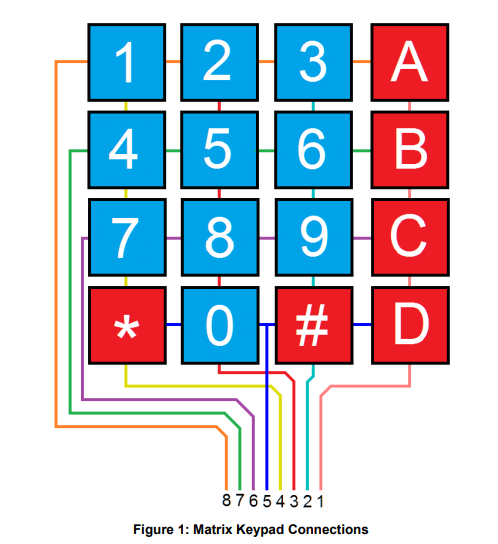
**Delay Code used in Assembly program**

****

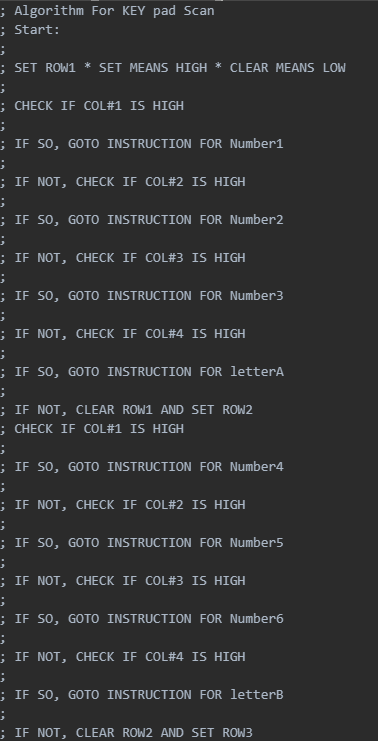
**Connection configuration used in circuit built for programmed Algorithm**

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

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

****

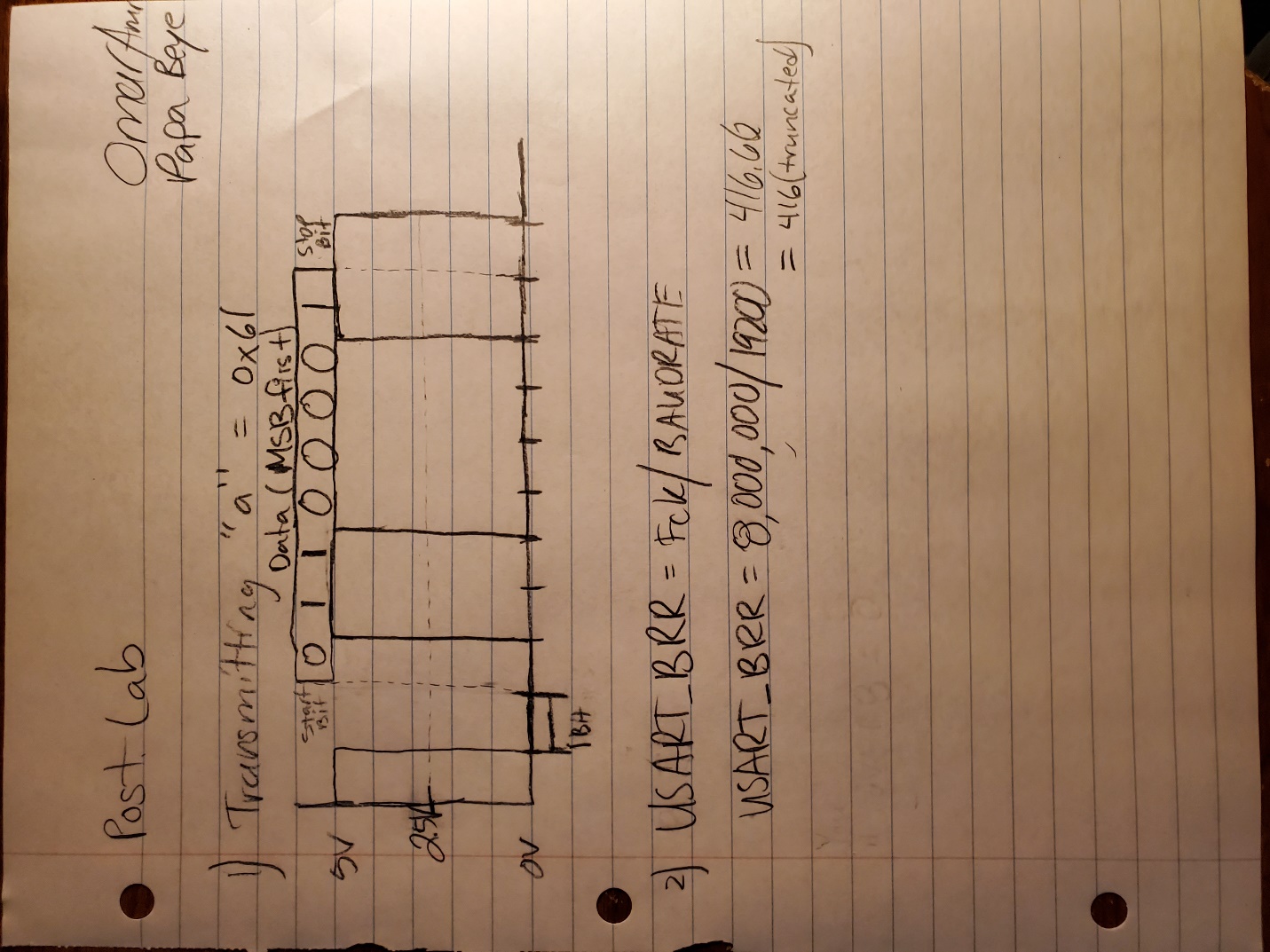
**Keypad scan 4x4 matrix algorithm scan**

****

**Pre-Lab included below**

Omar Amr Papa Beye

**Post-Lab**



**Conclusion**

In this lab, we became more comfortable with interfacing with the USART protocol on the STM32L4 micros

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.