**EGRE 364 – Microcomputer System**

**Laboratory Number 7**

**Light up a LED via EXTI**

**Lab Section: Tuesday 1PM**

***Lab conducted on: November 12th , 2019***

***Report Submitted on: November 22th , 2018***

**Papa Beye & Omar Amr**

**Major: CPE & CPE**

**Introduction**

The purpose of this lab was to utilize the GPIO Output Data Register for the two LEDs from the first lab, red and green, and interrupts to be able to repeatedly blink the red LED and use an external interrupt to toggle the green LED simultaneously using C language in the Keil µVision software development environment. To achieve this objective, we used GPIO registers, internal interrupt SysTick, and initialized external interrupts to be able to set up the interrupt handlers to complete the objective of using interrupts to read push buttons to complete task simultaneously with the blinking red LED.

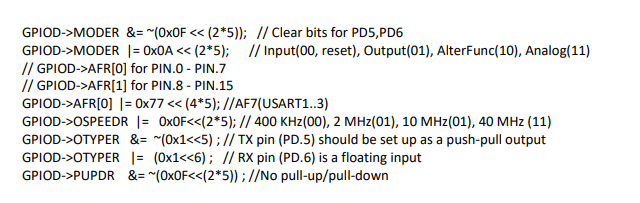
build on the concepts of the use of multiple GPIO input and output configurations to detect inputs and transmit data across Serial communications using Assembly in the Keil µVision software development environment. To achieve this objective, we used input/output registers of push pull, open drain, pull 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 PuTTY was used in lab to listen in any data that received from the USB connection to the board. 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**

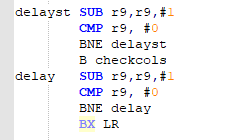
The STM32L4 discovery board has the choice for serial communication with the USART. In this lab, our circuit consisted of 4 pins for input from the keypad connected to Port A and 4 pins for output from the keyboard connected to Port E. The inputs pins were also connected to pull – down resistor to allow 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)**



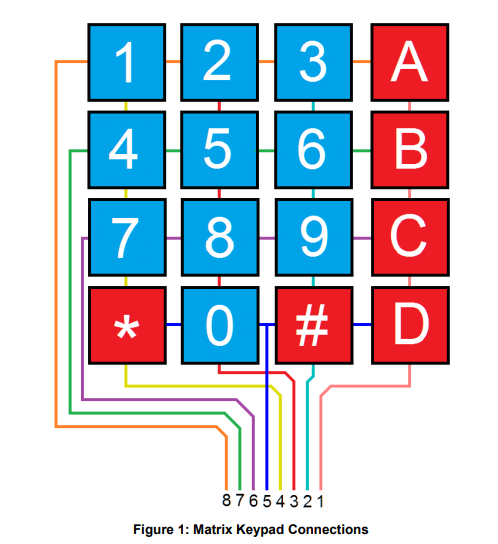
**Delay Code used in Assembly program**

****

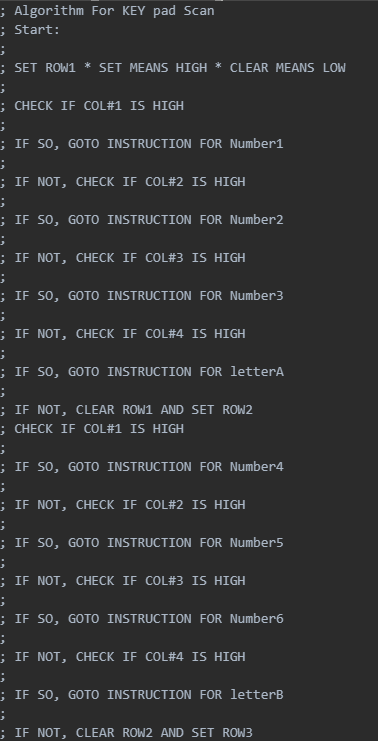
**Connection configuration used in circuit built for programmed Algorithm**

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

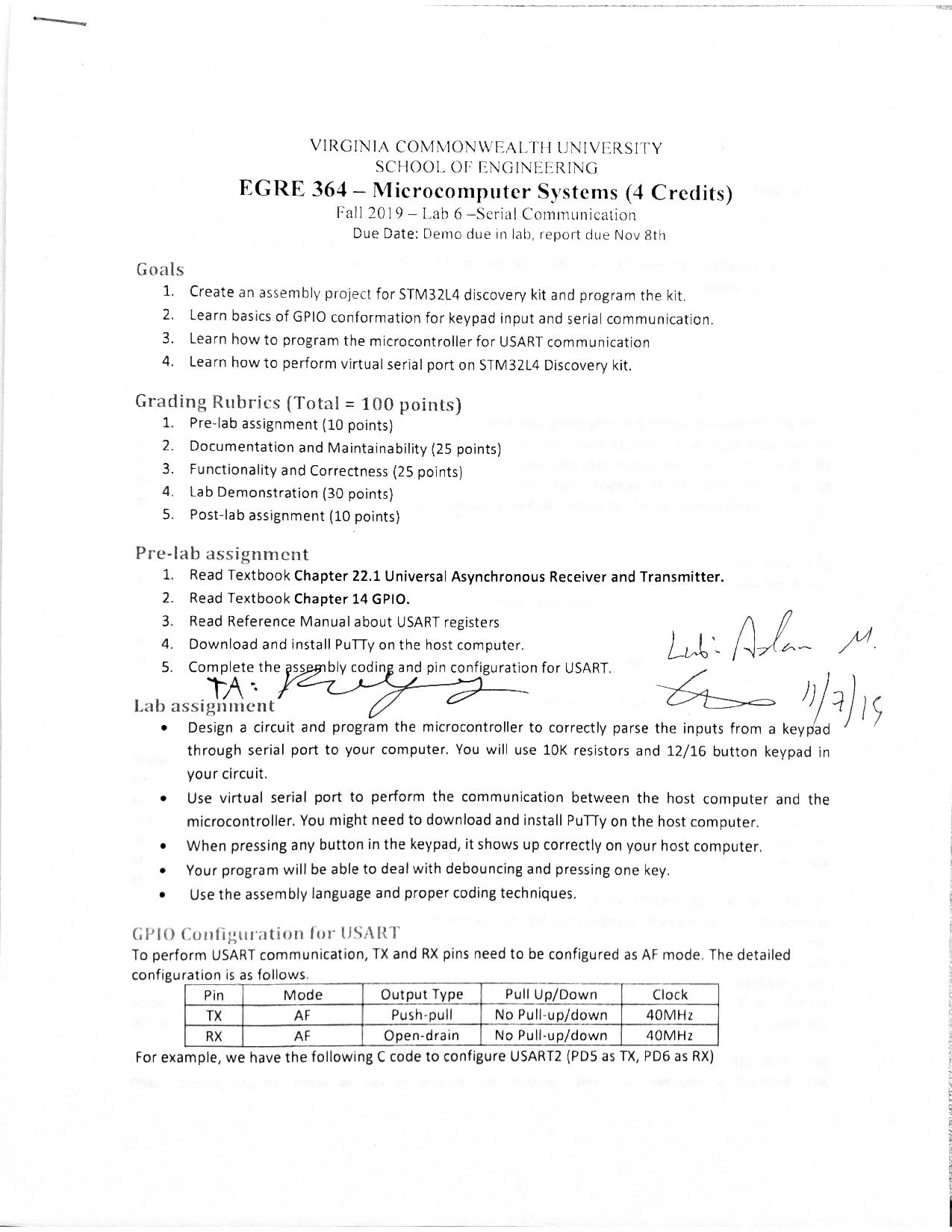
**Pins 5 - 8 : Rows (Output)**

****

**Keypad 4x4 matrix algorithm**

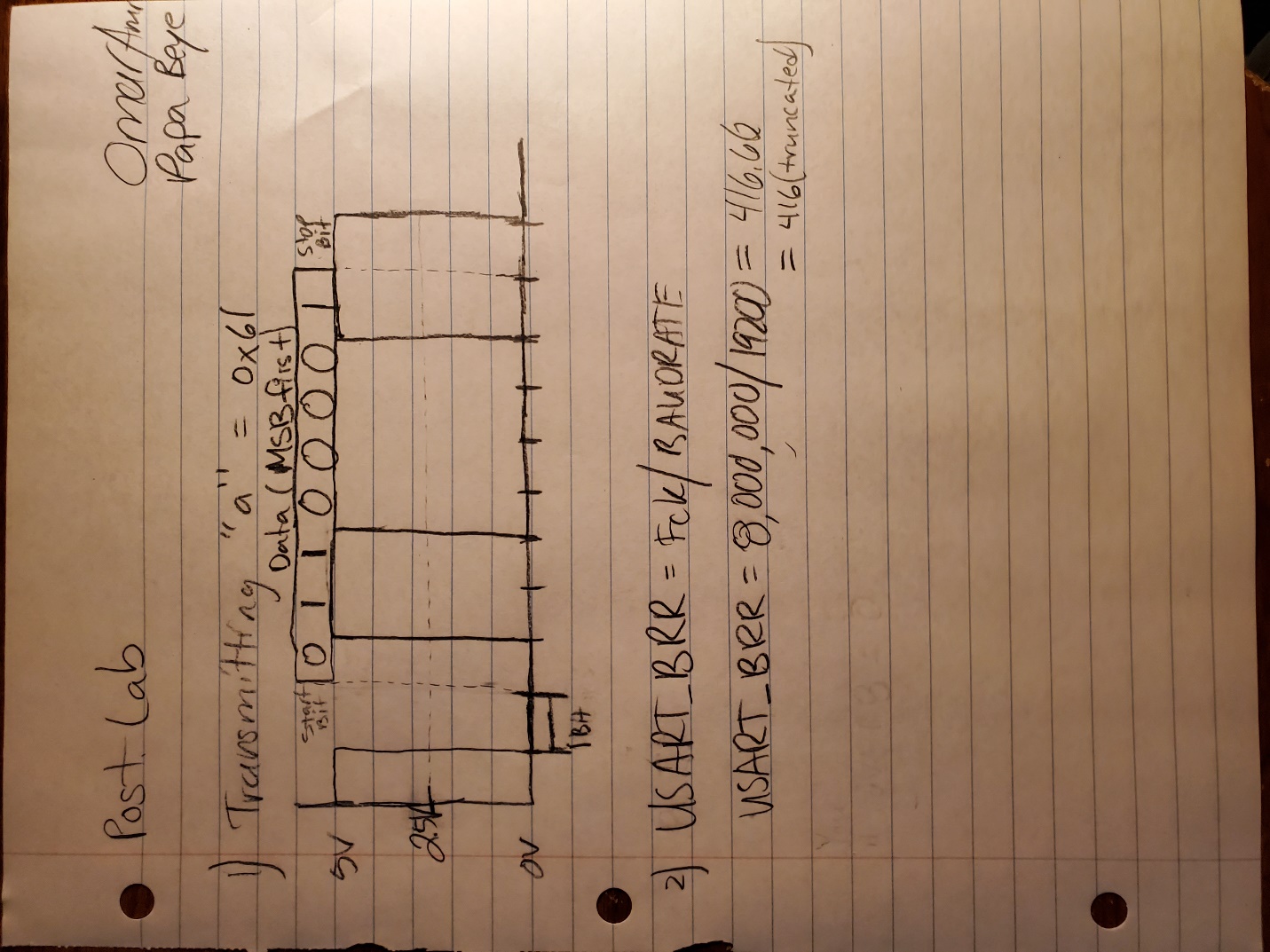
****

**Pre-Lab and Demo**

****

Omar Amr Papa Beye

**Post-Lab**



There are two oversampling settings. One is Oversampling by 8 or 16, which allows the sample the frame at 8 or 16 times the baud rate to mitigate the side effects of noise or clock deviation. If the OVER8 bit is 0, then the signal is oversampled by 16 and the BRR is equal to the USARTDIV, just like in the lab or the equation used in in the picture above. If the OVER8 bit is 1, then the signal is oversampled 8 and the BRR[15:4] \*16 + BRR[2:0]\*2, where BRR[3] is cleared.

**Conclusion**

In this lab, we became more comfortable with interfacing with the USART protocol on the STM32L4 microcontroller and creating an algorithm to determine what character was pressed on the keypad to complete the objective. We learned about checking the IDR value of multiple values iteratively based on outputs set. We were able to achieve the objective of sending the ASCII value of the pressed button to the host computer via serial communication by using the USART transmit data register to transmit data out based on the decided character from our keypad scanning algorithm. We used branches, conditionals, GPIO registers, and USART register to complete the objective. We used delays when we change outputs or waiting for release of the button. We learned a lot and gain more experience about how to use branches (BL and BX LR) and Assembly mnemonics and labels to initialize ports, access registers and reading inputs to design an algorithm to output a value that would set to the USART TDR. We didn’t face any issues but there was a learning curve that allowed us to learn how to create and use Assembly branches and loops to properly access and manipulate bit values of interest and set bit value without altering other bits and reading inputs through the IDR and properly polling and checking for inputs from the keypad and sending the value to the host computer via serial communication.