

# Microprocessor Systems Lab 1

## Checkoff and Grade Sheet

Partner 1 Name: Qianqian Che

Partner 2 Name: Wen Fan

Grade Component	Max.	Points Awarded		TA Init.s	Date
		Partner 1	Partner 2		
Performance Verification: Task 1	5 %	5		XT	9/20
	Task 2	10		XT	9/20
	Task 3	10	10	XT	9/20
	Task 4	5		XT	9/20
TA Questioning	20 %	20	20	XT	9/20
Documentation and Appearance	50 %				
Aide Deduction	-				
Total:					

Grader's signature: \_\_\_\_\_

Date: \_\_\_\_\_

## **Introduction and Background Information**

This lab is designed to introduce the use of System Workbench IDE which was Eclipse based, designed for STM32 microcontrollers. During this lab, one should familiarize oneself with STM32F769I Discovery board and should learn how to program the STM32F769NI using the GNU ARM C Compiler.

More specifically, one achieves communication between the STM32 microcontroller and a computer through a virtual serial connection provided by the USB connection, where the serial connection uses the peripheral USART1 to generate a UART interface operating at 115200 baud. Task 1 and Task 2 are designed to familiarize oneself with VT100/ANSI terminal standards. Task 3 and Task 4 are designed for one to learn how to configure the 769 with external electronics and to learn how to implement the manipulation of LEDs and switches using only registers or using the HAL driver.

## **Task 1 Introduction to User Interface**

Task 1 uses a simple program to get characters on the keyboard and print out on the terminal. The program should be halt when the user presses ESC.

## **Materials and Methods/Procedure**

In this task, team members need to clean the screen at the beginning of the main function. There is a simple while loop for getting characters continuously from the keyboard. Also, some Escape sequences are used in the code from Table of escape code. For instance, "\033[2J\033[;H" is used to erase screen and move cursor to the home position. Furthermore, team members need to flush stdout after each escape sequence. Within the while loop, program gets the character from getchar() and output certain text. There is also an if statement in the while loop to halt the program.

## **Results and Analysis**

From figure 1.1 shown below, we can observe that there are two lines on the terminal. The notation of how to terminate the program is at top middle of the screen. If we do not press the ESC button, the screen will be updated every time we get a character. The terminal result is exactly what we except. The program can be halted when we press ESC.



Figure 1.1 Task 1 terminal result

### Discussion and Observations

What we learn in task 1 is how to use escape sequence in different situations. The biggest problem for us is to familiar with C programming language we learned before.

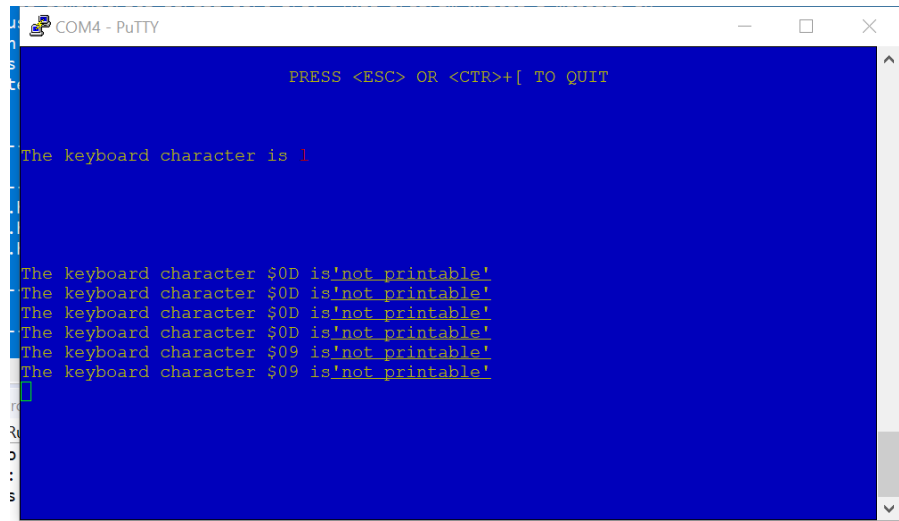
## Task 2

There are several requirements in Task 2. First is to change the background to blue and display yellow characters. Program needs to center the termination notation on line 2 and response of the printable character should be fixed at line 6 with pressed character in red color. If the keyboard character is not printable, a notation should be added from line 12 to line 24 and delete the oldest response if the screen without empty line.

### Materials and Methods/Procedure

In this task, most methods team members used is ANSI terminal sequence. The program needs to wait for a character at the beginning of the while statement. Same as the task 1 there is an IF for halting the program. If the character is printable, terminal should move the cursor to line 6 and erase the previous characters in this line. The response of new character is printed in this line and uses escape sequence \033[31m to change a single letter to red and \033[33m to change the rest sentence back to yellow. If the character is not printable it should be added to an array which will be printed from line 12. Team member also uses escape sequences to draw underlines for certain words and beep of the computer. In order to remove the oldest information in array, a new array is created to copy the the non-printable characters without the first one already in the old array.

## Results and Analysis

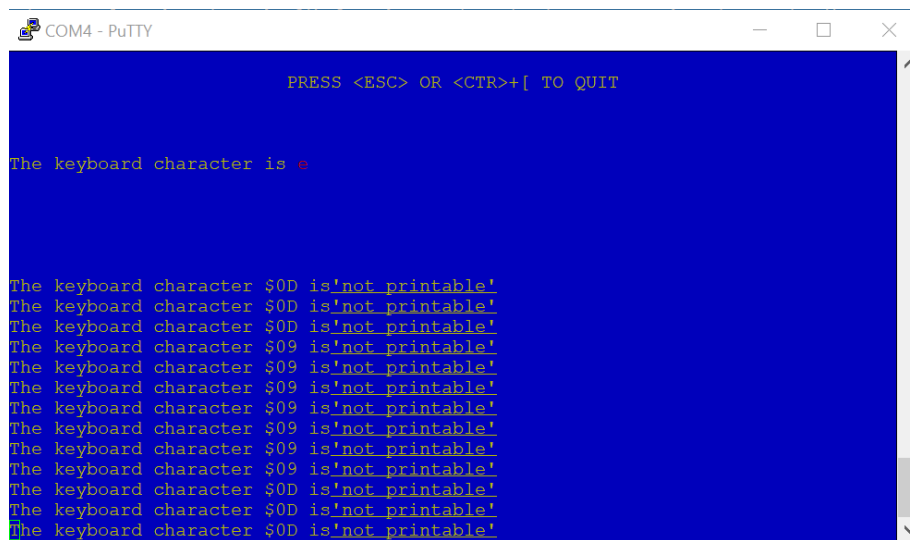


```
COM4 - PuTTY
PRESS <ESC> OR <CTR>+[ TO QUIT

The keyboard character is |

The keyboard character $0D is 'not printable'
The keyboard character $0D is 'not printable'
The keyboard character $0D is 'not printable'
The keyboard character $0D is 'not printable'
The keyboard character $09 is 'not printable'
The keyboard character $09 is 'not printable'
```

Figure 2.1 Task 2 terminal result



```
COM4 - PuTTY
PRESS <ESC> OR <CTR>+[ TO QUIT

The keyboard character is e

The keyboard character $0D is 'not printable'
The keyboard character $0D is 'not printable'
The keyboard character $0D is 'not printable'
The keyboard character $09 is 'not printable'
The keyboard character $09 is 'not printable'
The keyboard character $09 is 'not printable'
The keyboard character $09 is 'not printable'
The keyboard character $09 is 'not printable'
The keyboard character $09 is 'not printable'
The keyboard character $09 is 'not printable'
The keyboard character $0D is 'not printable'
The keyboard character $0D is 'not printable'
The keyboard character $0D is 'not printable'
```

Figure 2.2 Task 2 terminal result

From the figure 2.1 and 2.2 above, we can observe that there is a termination information at middle of line 2 and a response of getting character on line 6. Non printable messages are beginning from line 12 to 24. More messages will replace the previous one without scrolling the screen.

## Discussion and Observations

The hardest problem in this task is to remove previous character and keep the screen without scroll down. We spent a lot time on creating a new line with “\n” that will scroll the screen and line number of all

messages already printed will be changed. Therefore we change the \n in each line with an escape sequence "\033[1B" that just moves the cursor down 1 row.

### **Task 3**

In task 3, team members generally work together to achieve proper manipulation of four on-board LEDs dependent upon digital inputs from 4 switches. When wiring, the switches are wired on the protoboard to D0-D3 which are Arduino Header pins. The switches are wired such that when not toggled, the pin associated receives a logic LOW (0v) and a logic HIGH(+5V) when toggled. Accordingly, the state of the switches (D0-D3) is continuously read to control LEDs (LED1-LED4).

#### **Materials and Methods/Procedure**

Task 3 requires each team member to configure the 769 to execute manipulation of LEDs and switches using different methods. One team member implements the functionality using only registers while the other team member uses the HAL driver which are two separate programs.

#### **Member 1 (Registers):**

Configuration and access to the GPIO (General Purpose Input and Output) are enabled through the registers (GPIOx\_MODER, GPIOx\_OTYPER, GPIOx\_OSPEEDR, and GPIOx\_PUPDR). The team member looked through reference manual and got familiar with the functionalities of each register of GPIO.

First, in order for the GPIO functionality to work properly, the team member enabled peripheral clock of GPIO port through the register AHB1ENR. Then the team member followed the sequences of setting output mode, setting input mode, enabling pull-up resistors and then implementing while loop to continuously read inputs and write to output pins. More specifically, to set the pins relating to LEDs to output mode and to set the pins relating to toggle switches to input mode, the team member uses GPIOx\_MODER register, where x is the port number. To set the input mode pins to enable the internal pull-up resistor, the team member uses the register GPIOx\_PUPDR.

After the GPIO configuration (bit-masking operations), the team member starts a while loop to achieve getting continuous readings from input pins. To read from the input pins, the team member uses the register GPIOx\_IDR where it captures the data present on the Input pin at every AHB clock cycle. Then based on the input read from the switched, use the register GPIOx\_BSRR to write to the pins associated to the LEDs to achieve manipulation.

## **Member 2 (HAL):**

First statement in the main function is to enable clock for GPIO port A,C,D,J and F. The initial configuration for each pin consists 4 values that can be set. Team member needs to select which pin should be initialised using `GPIO_InitStructure.Pin` function. Also, team member chooses the mode for this selected pin which can be output push-pull or floating input. Resistors for the specified pins need to be set with pull-up and pull down. If the selected pin is digital output, we also need to specifies the speed for this pin. After all initial configuration, all input pin should be read using `HAL_GPIO_ReadPin` and turn on the light with `HAL_GPIO_WritePin`. The if statement for fourth light uses negation logic. Within a while loop, there are 4 if statements to figure out which switch is in logic high to turn on the corresponding light on the board. Also, we add a 0.1s delay in each loop.

## **Results and Analysis**

After the configuration, when one of the four switches is not toggled, the corresponding LED lights up, when toggled, the corresponding LED turns off. They can be lit at the same time or off the same time, in other words, each set of switch and LED is not affected by other sets.

The switches paired with pull-up resistors are wired (see Appendix 1.1) such that when the switch is on (not toggled), a logic LOW is read into the input pin associated, when the switch is off (toggled), a logic HIGH is read, which met the expectation.

Looking deeper into the two output registers `GPIOx_ODR` and `GPIOx_BSRR`. They are both output registers, but when writing to the LEDs in this case, `GPIOx_BSRR` is used instead of `GPIOx_ODR`. The team realized the difference between these two registers and the way of using them. For `GPIOx_ODR`, the bits 31:16 must be kept at reset value, meaning they cannot be changed or modified. So when using this register, the only used bits are 15:0 and they can be read or written. While the `GPIOx_BSRR` register utilizes all 31 bits but are write-only. A read to these bits returns the value 0x0000.

## **Discussion and Observations**

When using registers to implement the task, the team struggled a little with bit-masking operations. After picking up previous absorbed knowledge, the team was able to do bit-masking accurately and thoroughly.

Another issue was with the hardware implementation. In one occasion, the team assembled the STM32 board and the protecting shield board in the opposite way, resulting no power to the protoboard system. The team was able to fix the problem using multimeter to test the circuit. The pins connecting the

STM32 board and the protecting board are mapped in a symmetric way, which causes a problem when negligence occurs. The team will be more cautious on hardware implementation in the future.

## **Task 4**

### **Materials and Methods/Procedure**

Task 4 is built on Task 3 and focuses on circuit analysis. One of the switches is replaced with a potentiometer connected between +5V and GND with the wiper connected to the digital input as shown in figure 1.2 (in the Appendices). Using a multimeter, it is expected to measure the voltage of the potentiometer (between the wiper and ground) when the input reaches logic LOW from logic HIGH, and when the input reaches logic HIGH from logic LOW to determine whether the input logic is a simple digital buffer or Schmitt trigger.

### **Results and Analysis**

As shown in the figure 4.1, 4.2 below, the voltage when it reaches logic LOW is 1.23 V, and the voltage when it reaches logic HIGH is 1.71.

As there is a difference in voltage between when it reaches logic LOW from logic HIGH and when it reaches logic HIGH from logic LOW, the input logic uses Schmitt trigger. Schmitt trigger is a comparator circuit with hysteresis implemented by applying positive feedback to the noninverting input of a comparator or differential amplifier. It is an active circuit which converts an analog input signal to a digital output signal. The circuit is named a "trigger" because the output retains its value until the input changes sufficiently to trigger a change.

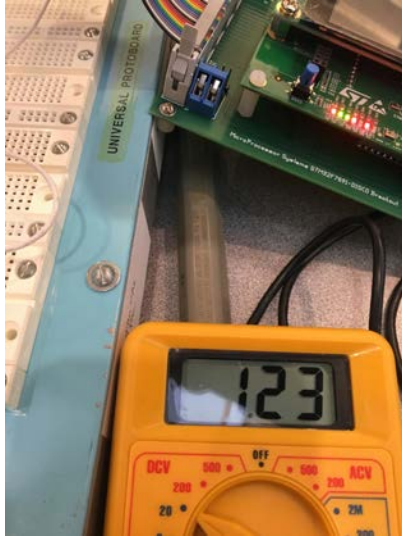


Figure 4.1 Input voltage when first logic LOW



Figure 4.2 input voltage when first logic HIGH

Our results show that  $<1.23\text{V}$  yield a logic LOW, and  $>1.71\text{V}$  yield a logic HIGH, which means that the values between them are neither logic LOW or logic HIGH. The dual threshold action is the hysteresis (as shown in figure 1.3 in Appendices).

### Discussion and Observations

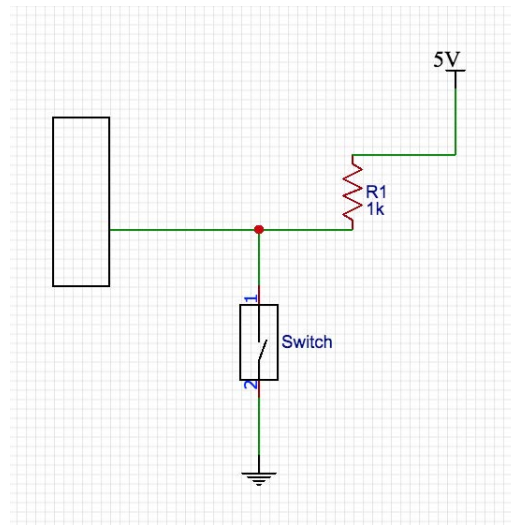
When adjusting the potentiometer, the adjustment by hand is not accurate enough for reading the threshold voltage of logic LOW and logic HIGH. A more automatically measured device would be more accurate.

### Conclusions

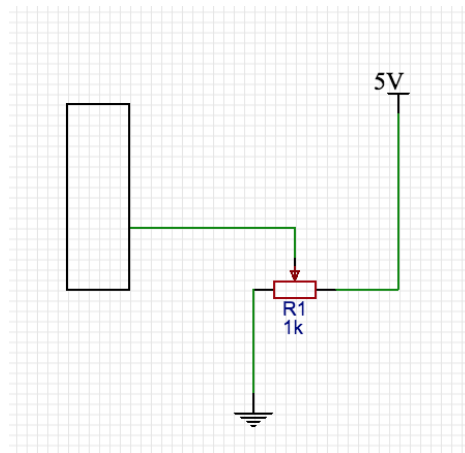
From this lab we learned how to use the eclipse-based STM32 microcontroller system. We are familiar with how to program STM32f769 with the language we learned before. New material for us is the using of ANSI terminal escape sequence. We can use this knowledge to show more diverse representation on the terminal screen in the future. ANSI terminal standards enable us to change the style of text with the background, font color and more controllable of the cursor. Furthermore, we learned how to access pins on the microcontroller using register and HAL methods. That allows us to do more action using the STM32 microcontroller not limited to light control.

### Appendices





1.1 Circuit diagram of one switch paired with a pulled up resistor



1.2 Circuit diagram of one potentiometer replacing one of the switches

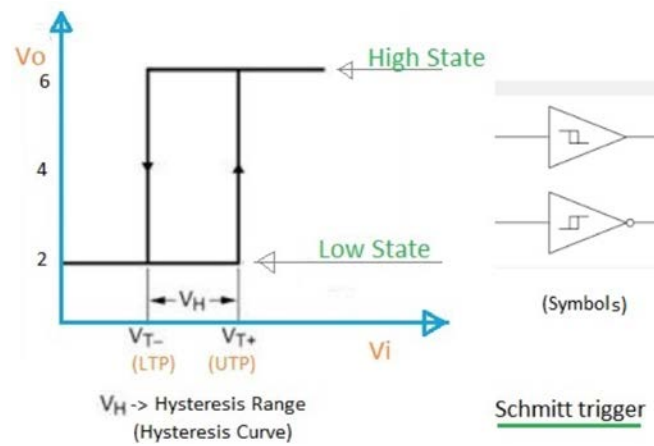


Figure 1.3 Schmitt trigger

## References

1. RM0410-stm32f7\_Reference\_Manual-2.pdf
2. <http://www.rfwireless-world.com/Terminology/UTP-vs-LTP-Schmitt-Trigger.html>