Department of Computer Science and Engineering
University of Dhaka

CSE3116: Microcontroller Lab
Year - 3, Semester - 1
Assignment 2:

"Traffic Signal Customization"

Submitted to:

Dr. Mosaddek Tushar
Redwan Ahmed Rizvee

SUBMITTED BY

Shahanaz Sharmin
Roll No: 07
Anika Tabassum
Roll No: 61

April 5, 2023

Contents

Chapter 1

INTRODUCTION

The objectives of the lab assignment are to understand and have hands-on training in features of the STM32F446RE microcontroller: UART (Universal Asynchronous Receiver Transmitter) and TIMER configuration. The primary goal is to learn about the configuration and operation of UART pins and ports as buffered input and output using interrupt subroutines. The project aims to enhance the understanding of students on how to use these features to build real-time applications that require reliable communication and timekeeping

1.1 General Objectives

1. To provide hands-on training in configuring and using UART and TIMER features of the STM32F446RE microcontroller.
2. To enhance the understanding of students on the operation and applications of UART and TIMER features in real-time systems.
3. To develop the skills of students in implementing real-time applications using UART and TIMER features.
4. To provide practical experience in working with interrupt subroutines for buffered input and output using UART pins and ports.

Chapter 2

PROBLEM STATEMENT

2.1 Problem Statement

We have been tasked with developing a traffic monitoring system that allows for custom configuration commands. The system will enable traffic control room operators to send commands to the traffic light controller to direct the flow of traffic. To meet the project requirements, we need to implement the following features:

1. Establish communication between USART/UART or microcontrollers for transmitting data.
2. Provide real-time traffic state information to the control center through the terminal, both periodically and on-demand.
3. Allow for configuration of reporting delay, traffic light intervals, and extra time for imbalanced traffic flow via the control center.
4. Display current and previous two reporting states after a reporting interval.

2.2 Explanation

1. **config traffic light x G Y R u**: Set the new intervals for traffic light x by specifying the duration in seconds for each light (G for green, Y for yellow, and R for red), as well as the extra interval (u) for imbalanced traffic.
2. **config traffic delay X**: Set the number of seconds (X) after which the traffic light will automatically report its state.
3. **read config**: Display the current configuration of the entire system, including traffic light intervals and reporting delay.
4. **read traffic light x**: Display the current configuration for traffic light x, including its current state and time remaining for each light.
5. **read report delay**: Display the current auto-report delay.

Chapter 3

Theory

In order to implement the traffic monitoring system with custom configuration commands, we need to make use of two key hardware components: the Basic Timers and the Universal Asynchronous/Synchronous Receiver Transmitter UART/USART. These components will allow us to control the timing of the traffic light intervals and transmit data between the traffic control room and the traffic light controller, respectively.

3.1 General Purpose Timers

STM32F446RE has 10 synchronized general purpose timers. We are working with TIM2 and TIM5. They are based on 32-bit auto-reload up/down counter and a 16 bit prescaler. Both are connected through the APB1 peripheral bus. The registers and functionality of the timers are given below. Where x in TIMx stands for the number of timer.

3.1.1 TIMx_CR1

The zeroth bit in the control register CR1 of the timers simply enables the timer. We can enable a Timer by simply setting the 0th bit to high

3.1.2 TIMx_SR

This bit is set by hardware on an update event. It is cleared by software.
0: No update occurred.
1: Update interrupt pending. This bit is set by hardware when registers are updated.

3.1.3 TIMx_PSC

The lower 16 bits of this register holds the prescaler value for the timer. We simply load a desired prescaler value to this timer according to the timer frequency calculation

Calculating the timer frequency

The clock frequency CK_CNT is equal to

$$CK\_CNT = \frac{f_{CK\_CNT}}{PSC + 1}$$

So to set the prescaler we must reduce Prescaler.

3.1.4 TIMx_ARR

The lower 16 bits of this register holds the auto-reload value of the timer. The timer resets when it is done counting up to this value or counting done from this value.

3.2 UART/USART Universal Synchronous/Asynchronous Receiver Transmitter

STM32F446RE has four embedded universal synchronous/asynchronous receiver transmitters(USART1, USART2, USART3 and USART6) and four universal asynchronous receiver transmitters (UART4, and UART5). For communication of the console we have used USART2 and for communication between the microcontrollers we have used UART4 and UART 5. Since the communication of these are full duplex we must utilize the defined TX and RX pins corresponding to each UART. These pins are exposed to us via GPIO pin alternate functions. The registers which help us configure the GPIO are explained as follows.

3.2.1 GPIOx_MODER

The GPIOx_MODER register is used to set the mode for each individual GPIO pin of the microcontroller. The register is a 32-bit wide register, where each pair of bits corresponds to a single GPIO pin. The possible configurations for the GPIOx_MODER register are:

Function	Bit pattern to load
Input	00
General purpose output	01
Alternate Function Mode	10
Analog mode	11
Default	00

Table 3.1: GPIO_MODER configurations

We configure it as alternate function for our application. As USART/UART is an alternate function of the GPIO pins. To configure a GPIO pin as a particular mode, the corresponding two bits in the GPIOx_MODER register must be set to the appropriate value. For example, to set a GPIO pin as an alternate function mode, the corresponding two bits in the GPIOx_MODER register must be set to "10".

It is important to note that the GPIOx_MODER register is not the only register used to configure the GPIO pins. There are several other registers, such as GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR, and GPIOx_AFRL/AFRH, that are used to further configure the pins depending on the specific requirements of the application.

3.2.2 GPIOx_OSPEEDR

The GPIOx_OSPEEDR register is used to set the output speed for each individual GPIO pin of the microcontroller. The register is a 32-bit wide register, where each pair of bits corresponds to a single GPIO pin. The possible configurations for the GPIOx_OSPEEDR register are:

Function	Bit pattern to load
Low Speed	00
Medium Speed	01
Fast Speed	10
High Speed	01

Table 3.2: GPIO_OSPEEDR configurations

To configure a GPIO pin as a particular output speed, the corresponding two bits in the
GPIOx_OSPEEDR register must be set to the appropriate value. For example, to set a GPIO pin as high speed, the corresponding two bits in the GPIOx_OSPEEDR register must be set to "11".

It is important to note that the GPIOx_OSPEEDR register is only applicable for output modes (General purpose output or Alternate function modes). For input modes, the GPIOx_OSPEEDR register has no effect.

The output speed of a GPIO pin is an important consideration for applications where the GPIO pins are being used to drive external devices such as LEDs or motors. A higher output speed will result in faster switching times and can improve the overall performance of the application. However, it is important to ensure that the output speed does not exceed the maximum capabilities of the external devices being driven, as this can lead to damage or malfunction.

3.2.3 GPIOx_AFR(L&H)

The GPIOx_AFRL and GPIOx_AFRH registers are used to configure the alternate function of the GPIO pins. Each GPIO pin has a corresponding 4-bit field in the registers, with GPIOx_AFRL used for pins 0-7 and GPIOx_AFRH used for pins 8-15. The possible values for these fields depend on the specific alternate function being used.

To configure the alternate function of a GPIO pin, the corresponding 4-bit field in the appropriate register must be set to the appropriate value. For example, if USART2 is being used on pins PA2 and PA3, the GPIOx_AFRL fields for pins 2 and 3 would need to be set to the appropriate USART2 alternate function value.

It is important to note that the GPIOx_AFRL and GPIOx_AFRH registers are not the only registers used to configure the alternate function of the GPIO pins. Depending on the specific requirements of the application, other registers such as GPIOx_OTYPER, GPIOx_OSPEEDR, and GPIOx_PUPDR may also need to be configured.
Here is a table showing some example alternate function values for USART2 on pins PA2 and PA3:

GPIO Pin	Field Number	USART2 Alternate Function Value
PA2	8-11	7
PA3	12-15	7

Table 3.3: Example USART2 Alternate Function Values for PA2 and PA3

Here is a table showing some example alternate function values for UART1 on pins PB6 and PB7:

GPIO Pin	Field Number	UART1 Alternate Function Value
PB6	24-27	7
PB7	28-31	7

Table 3.4: Example UART1 Alternate Function Values for PB6 and PB7

Here is a table showing some example alternate function values for UART6 on pins PC6 and PC7:

GPIO Pin	Field Number	UART6 Alternate Function Value
PC6	24-27	8
PC7	28-31	8

Table 3.5: Example UART6 Alternate Function Values for PC6 and PC7

3.2.4 UARTx_CR1

We are concerned with a few bits of this register which are

1. **UE** The bit 13 of the Control register, it enables the UART/USART when set to high. Else the prescaler and outputs are disabled.
2. **RE** The bit 2 which when set to high enables the Receiver.
3. **TE** The bit 3 which when set to high enables the Transmitter.
4. **M** when set to high sets word length to 9. when low sets word length to 8.
5. **RXNEIE** The bit 5 which enables an interrupt whenever the Rx buffer is not empty when this bit is set.

The UARTx CR1 register is used to configure the various parameters of the UART, such as enabling the receiver and transmitter, setting the word length and parity, and enabling interrupts. It is a 16-bit register where each bit has a specific function. The table above provides a brief description of each bit and its function.

3.2.5 UARTx_BRR

The baud rate register holds the values for the Mantissa (DIV_Mantissa) and Fraction (DIV_Fraction) used to set the baud rate of the USART/UART interface. The first 4 bits are reserved for the fraction of USARTDIV. The next 12 bits are for the mantissa of USARTDIV. The upper 16 bits are reserved and kept at reset value.

Calculating the Baud rate

$$Tx/Rx\ baud = \frac{f_{CK}}{8 \times (2 - OVER8) \times USARTDIV}$$

where,

$OVER8$ is the oversampling by 8 bit in the CR1 register.

$$USARTDIV = DIV\_Mantissa + \frac{DIV\_Fraction}{8 \times (2 - OVER8)}$$

3.2.6 UARTx_SR

The status register (UARTx_SR) is a register used in UART communication to indicate the status of the communication channel. It provides information about various events and errors that can occur during data transmission.
The UARTx_SR register is a 16-bit register with various bits dedicated to specific functions. Some of the key bits of the register are:

- **RXNE (Read Data Register Not Empty)**: This bit is set when the receive buffer is not empty, indicating that there is data available to be read.
- **TXE (Transmit Data Register Empty)**: This bit is set when the transmit buffer is empty, indicating that data can be written to the transmit buffer.
- **TC (Transmission Complete)**: This bit is set when the transmission of all data in the transmit buffer is complete.
- **PE (Parity Error)**: This bit is set when a parity error occurs during communication, indicating that the parity of the received data does not match the expected parity.
- **FE (Framing Error)**: This bit is set when a framing error occurs during communication, indicating that the received data is not framed correctly.
- **NE (Noise Error)**: This bit is set when a noise error occurs during communication, indicating that the received data contains unwanted noise.
- **ORE (Overrun Error)**: This bit is set when an overrun error occurs during communication, indicating that data was not read from the receive buffer in time, and new data overwrote the old data.

By checking the status of the UARTx_SR register, the microcontroller can determine the status of the communication channel and take appropriate action. For example, if the RXNE bit is set, the microcontroller can read the data from the receive buffer. If the TXE bit is set, the microcontroller can write data to the transmit buffer. If the PE, FE, NE, or ORE bits are set, the microcontroller can take corrective action to resolve the error.

3.2.7 UARTx_DR

The UARTx_DR register is used to transmit and receive data through the USART/UART interface. This register is 8 bits wide and allows for full duplex communication. When data is written to the UARTx_DR register, it is transmitted through the TX line. Similarly, when data is received through the RX line, it is stored in the UARTx_DR register.

The UARTx_DR register is a read-write register, which means that data can be both written to and read from this register. Writing data to the UARTx_DR register is achieved by simply loading the data to be transmitted into the register. Reading data from the UARTx_DR register is done by reading the register contents after the reception of a complete character.

Chapter 4

Physical Design

4.1 Control and Communication

Since we had to do it with only one microcontroller, We have considered UART4 and USART2 to be the available interface at the control room. Where USART2 is the interface for the terminal and UART4 is the interface for the traffic control system. UART5 is the interface through which the traffic control system talks to the control room

4.2 Traffic system physical design

We have set one LED to show the congestion. Where on indicates high traffic. We have also observed that we don't explicitly need 4 traffic lights since there are two independent axes. Also we minimized further by noticing the fact that the light yellow can be abstracted by the state of having both lights off. Thus we minimized the circuit even further.

Chapter 5

Implementation

5.1 UART Configuration

We are using USART 2 to communicate with the user and UART 4 as control center and UART 5 as traffic center. So we need to configure these three at first. To configure the UARTs, we turn on necessary bits. We set 1 at necessary position to enable the UARTs. We also enable necessary GPIO bus and UART bus. Then we set the mode, which is alternate function for the UARTs. Then we, we save the alternate function number in the alternate function register for the respective UARTs. We also set the speed very high for the UARTs. Then we set the data word length as 8 by setting the M bit 0 in UART CR1 register. Then we set the baud rate 115200 by setting the fraction part 7 and the mantissa part 24. Lastly, we enable the recieve and transmit bit for the UARTs and enable the receive interrupt as the Uarts might need to receive data always.

5.2 Designing Interrupts

Interrupts are called for transmitting and receiving data. When the transmission interrupt bit TXNEIE or the receive interrupt bit RXNEIE is on, an interrupt is generated and the written interrupt function is called. Interrupt helps the program to do the work of communication in the background while all the other work is being done. The receive interrupt bit is always on as data might come at any moment. But the transmission interrupt bit is enabled only when the data is being transimitted.

5.2.1 USART2 Interrupt Handler

Listing 5.1: USART2 Interrupt Handler

```
1 void USART2_IRQHandler(void)
2 {
3     USART2->CR1 &= (uint32_t)~(1<<5);
4     UART_GetString(USART2,input);
5     USART2->CR1|=(1<<5);
6 }
```

In the interrupt handler, at first the receive interrupt bit is disabled (otherwise, the interrupt will be called again and again) then the get string function is called to get the data from the DR register of the USART2. The data is saved in the input array. After

that, the receive interrupt bit is enabled again. USART2 only communicates with the user it does not communicate with the other UARTs. This is why, there is no transmission part in this interrupt handler.

5.2.2 UART4 and UART5 Interrupt Handler

Listing 5.2: UART4 Interrupt Handler

```
7
8  void UART4_IRQHandler(void)
9  {
10     if(UART4->SR &(1<<5))
11     {
12          UART4->CR1 &= (uint32_t)~(1<<5);
13          output[k]=(uint8_t)UART4->DR;
14          UART4->SR&=(uint32_t)~(1<<5);
15          UART4->CR1|=(1<<5);
16
17     }
18
19     if(UART4->SR & (1<<7))
20     {
21          UART4->DR=input[k];
22          UART4->SR &= (uint32_t)~(1<<7);
23          UART4->CR1 &=(uint32_t) ~(1<<7);
24
25     }
26
27 }
```

Here, we need to handle the transmission and reception both part. If the RXNE bit is enabled (which indicated received data is ready to be read), then data is read from the Data Register in the output buffer. If the TXNE bit is enabled, then data is transferred to the Data register from the input buffer. Both are done character by character. UART4 and UART5 interrupt handlers are both same. They are enabled when needed by turning the TXNEIE and RXNEIE bit on of the respective UARTs.

5.3 Processing the Command

The command is sent in the form of a string. A condition is set in the while loop to check if there is any input. If there is, the input is sent from UART4 to UART5 that is from control center to traffic center and the command is processed by reading the string character by character. The delays, extra time and monitor time are set by Configure command. If the command is "read", current configuration is shown. Before showing the configuration in USART2, the configuration is sent at first from UART5 to UART4.

5.4 Turning on the LEDs with appropriate Delays

We turn on and off the LEDs by enabling necessary GPIO ports and GPIO Write Pin function used in the previous labs. There are initial delays and extra time set in the codes which are later on changed according to the input. But the usual delay function cannot be used. Because, in the usual delay functions, the program is halted and wait for the duration of the delay. But in this project, we need to process the command continuously and show output like status of the lights. This is why we need to design a delay that do this both while keeping on the LEDs for the certain delays.

Listing 5.3: Delay function

```
29
30 void delay(uint16_t ms)
31 {
32     ms = (uint32_t)2 * ms;
33     TIM5->CNT = 0;
34     while(TIM5->CNT < ms)
35     {
36
37         if(strlen(input) != 0)
38         {
39                 processString();
40         }
41
42         if(TIM2->CNT > (uint16_t)interval*1000*2)
43         {
44                 time_count += interval;
45                 traffic_status();
46                 TIM2->CNT = 0;
47         }
48     }
49 }
```

In the delay function two timers are used. One is used to count the delay time and another is used to count the monitor time or the time after which traffic status has to be shown. In the delay time, if a input comes, it has to be processed. And if the timer reaches the monitoring time, the traffic status is shown and the TIM2 is again set to 0. Thus, the function runs the delay and process the commands at the same time.

5.5 Showing traffic status

The traffic monitor time is set initially to a time. It can be changed later on by the Configuration command. After the traffic monitor time interval, traffic status of the lights have to be shown. To show this, a timer is set in the delay function so that the program does not stop there. In the status function, at first the traffic lights are read by using the read pin function. A global timer is used so that the interval with the time can be shown. Then the status is sent from UART5 to UART4, that is from traffic center to control center and then in shown in USART2.

Chapter 6

Output

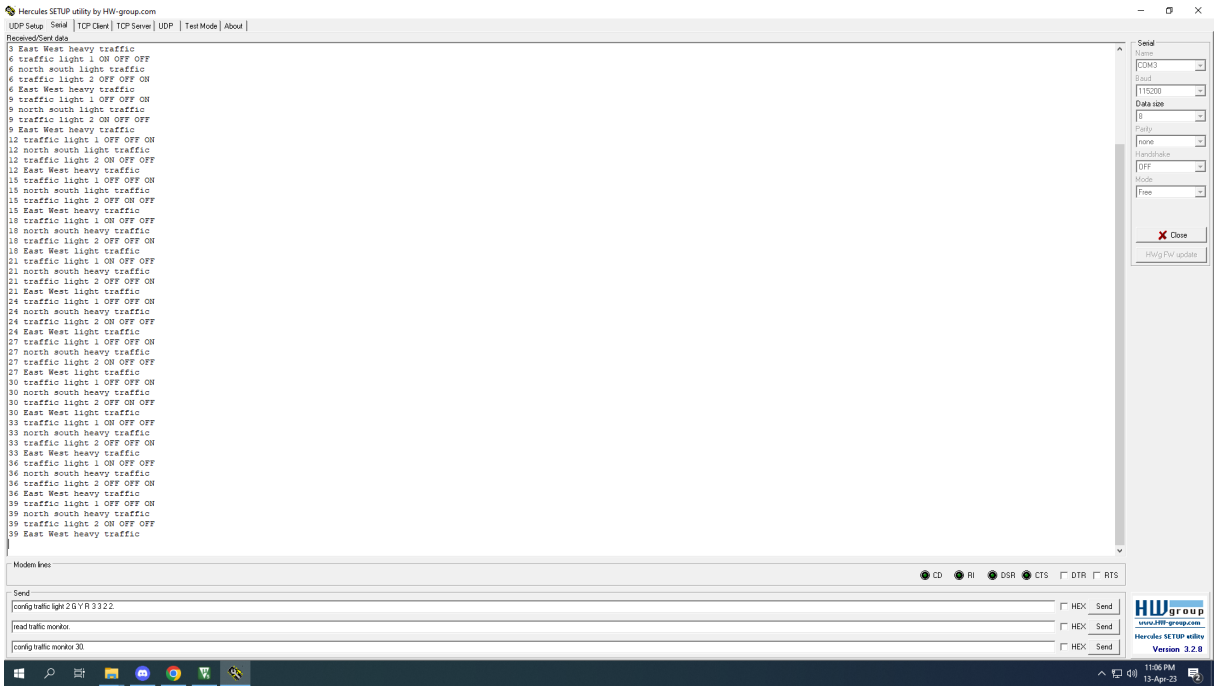

Figure 6.1: Showing traffic status after a certain delay

```
51 north south light traffic
51 traffic light 2 OFF OFF ON
51 East West heavy traffic
read.traffic light 1 G Y R 5 2 5 2
traffic light 2 G Y R 5 2 5 2
traffic monitor 3
54 traffic light 1 OFF OFF ON
54 north south light traffic
54 traffic light 2 ON OFF OFF
54 East West heavy traffic
```

Figure 6.2: Showing the current configuration of the traffic with the read command

```
config traffic light 2 G Y R 3 3 2 2.66 traffic light 1 ON OFF OFF
66 north south heavy traffic
66 traffic light 2 OFF OFF ON
66 East West light traffic
69 traffic light 1 OFF OFF ON
69 north south heavy traffic
69 traffic light 2 ON OFF OFF
69 East West light traffic
72 traffic light 1 OFF OFF ON
72 north south heavy traffic
72 traffic light 2 ON OFF OFF
72 East West light traffic
75 traffic light 1 OFF OFF ON
75 north south heavy traffic
75 traffic light 2 OFF ON OFF
75 East West light traffic
read traffic light 2.traffic light 2 G Y R 3 3 2 2
78 traffic light 1 ON OFF OFF
78 north south light traffic
78 traffic light 2 OFF OFF ON
78 East West heavy traffic
```

Figure 6.3: Changing the configuration of a light and reading that configuration

```
102 north south light traffic
102 traffic light 2 OFF ON OFF
102 East West heavy traffic
105 traffic light 1 ON OFF OFF
105 north south heavy traffic
105 traffic light 2 OFF OFF ON
105 East West heavy traffic
config traffic monitor 10.115 traffic light 1 OFF OFF ON
115 north south heavy traffic
115 traffic light 2 ON OFF OFF
115 East West heavy traffic
125 traffic light 1 ON OFF OFF
125 north south light traffic
125 traffic light 2 OFF OFF ON
125 East West light traffic
```

Figure 6.4: Changing the traffic monitor time