1. INRODUCTION:

The objective of this lab is to get you communicate with MCU from PC, and utilize bidirectional data transmission and parsing. You will use C language for the problems unless some parts require inline assembly.

2. PROBLEMS:

2.1. Problem 1:

In this problem, you will connect your board to the PC using UART protocol. For this you will need to create an initialization routine for UART, and create send and receive functions. You should not use interrupts for this assignment. Basically create two functions as given below

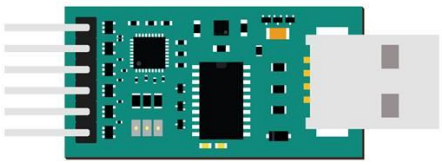
```
void uart_tx(uint8_t c);
uint8_t uart_rx(void);
```

and in your main loop, call them like

```
while(1) {
uart_tx(uart_rx());
}
```

2.1.1. Theoretical Research:

What is UART Protocol?



Universal asynchronous receiver-transmitter (UART) is one of the simplest and oldest forms of device-to-device digital communication. You can find **UART devices** as a part of integrated circuits (ICs) or as individual components. UARTs communicate between two separate nodes using a pair of wires and a common ground.

UART Protocol Guide

As a “universal” setup, we can configure UART to work with many different types of serial protocols. UART was adapted into single-chip units in the early 1970s, starting with Western Digital’s WD1402A.

In a UART communication scheme:

1. One chip’s Tx (transmit) pin connects directly to the other’s Rx (receive) pin and vice versa. Commonly, the transmission will take place at 3.3 or 5V. UART is a single-master, single-slave protocol, where one device is set up to communicate with only one partner.

2. Data travels to and from a UART in parallel to the controlling device (e.g., a CPU).

3. When sending on the Tx pin, the first UART translates this parallel information into serial and transmits it to the receiving counterpart.

4. The second UART receives this data on its Rx pin and transforms it back into parallel to communicate with its controlling device.

UARTs transmit data serially, in one of three modes:

- **Full duplex**: Simultaneous communication to and from each master and slave

- **Half duplex**: Data flows in one direction at a time

- **Simplex**: One-way communication only

Data transmission takes place in the form of data packets, beginning with a start bit, where the ordinarily high line is pulled to ground. After the start bit, five to nine data bits transmit in what is known as the packet’s data frame, followed by an optional parity bit to verify proper data transmission. Finally, one or more stop bits are transmitted where the line is set to high. This ends a packet.

As an asynchronous protocol— no clock line regulates data transmission speed— users must set both devices to communicate at the same speed. This speed is known as the baud rate, expressed in bits per second, or bps. Transmission speeds vary dramatically, from the typical 9600 baud setting to 115200 and beyond.

While something of an “ancient” protocol, and one that can only communicate between a single master and slave, UART is well known, easy to set up, and extremely versatile. As such, you’re likely to encounter this system when working with microcontroller projects. UARTs may be a part of systems that you use every day, whether you realize it or not.

Duty cycle

A **duty cycle** or **power cycle** is the fraction of one period in which a signal or system is active. Duty cycle is commonly expressed as a percentage or a ratio. A period is the time it takes for a signal to complete an on-and-off cycle. As a formula, a duty cycle (%) may be expressed as:

$$D = \frac{PW}{T} \times 100\%$$[2]

Equally, a duty cycle (ratio) may be expressed as:

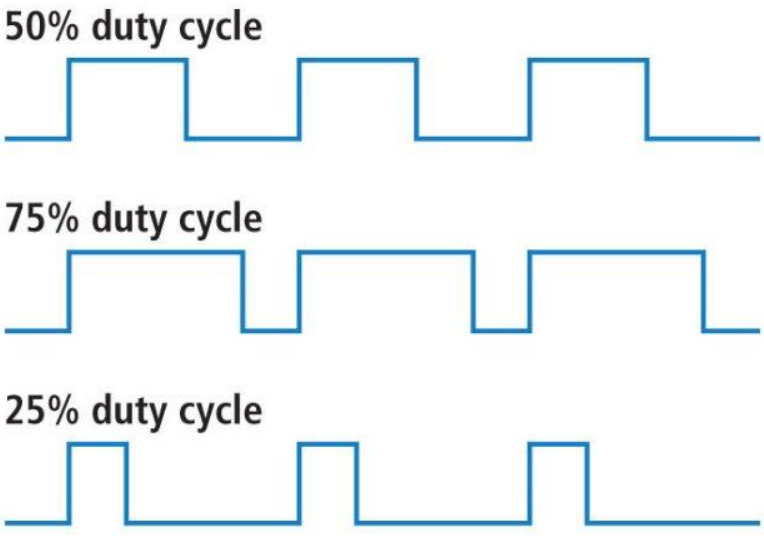
$$D = \frac{PW}{T}$$

where is the duty cycle, is the pulse width (pulse active time), and is the total period of the signal. Thus, a 60% duty cycle means the signal is on 60% of the time but off 40% of the time. The "on time" for a 60% duty cycle could be a fraction of a second, a day, or even a week, depending on the length of the period.

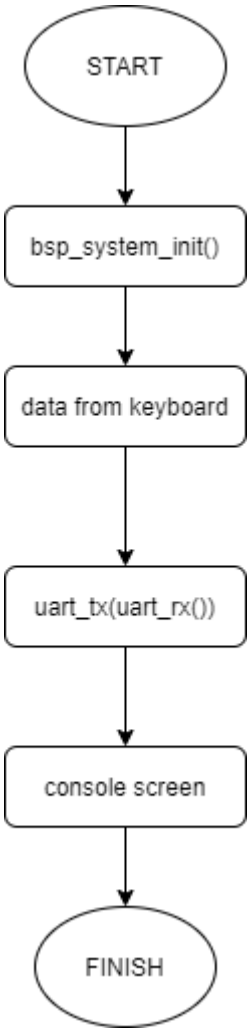
Duty cycles can be used to describe the percent time of an active signal in an electrical device such as the power switch in a switching power supply or the firing of action potentials by a living system such as a neuron.

The **duty factor** for periodic signal expresses the same notion, but is usually scaled to a maximum of one rather than 100%.[

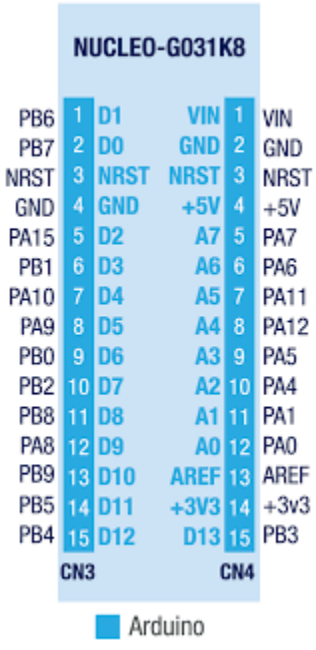
The duty cycle can also be notated as



2.1.2 Flowchart:



2.1.3 Block diagram:



2.1.4 Code:

Bsp.h:

```
#ifndef     BSP_H_
#define BSP_H_

#include "stm32g0xx.h"

void delay(volatile unsigned int);

void BSP_system_init();
void BSP_UART_init(uint32_t);
void uart_tx(uint8_t);

uint8_t uart_rx();

void BSP_led_init();
void BSP_led_set();
void BSP_led_clear();
void BSP_led_toggle();

//Button related function
void BSP_button_init();
int BSP_button_read();

#endif
```

Bsp.c:

```
#include "bsp.h"

void delay(volatile unsigned int s) {
    for(; s>0; s--);
}

void BSP_system_init(void){

      __disable_irq();
      BSP_led_init();
      BSP_button_init();
      BSP_UART_init(9600);
      __enable_irq();

}

void uart_tx(uint8_t c){

      USART2->TDR = (uint16_t)c;
```

```
      while (!(USART2->ISR & (1 << 6)));

}

uint8_t uart_rx(void){

      while (!(USART2->ISR & (1 << 5))); //RXFNE bit of ISR, receive
data ready to read
      return (uint8_t)USART2->RDR;
      //RXNE is automatically cleared when read
}


/*
 * PA2 PA3 are connected to the Virtual COM port
 * USART2 module
 */


void BSP_UART_init(uint32_t baud ){

      RCC->IOPENR |= (1U << 0);
      RCC->APBENR1 |= (1U << 17);

      //setup PA2 as AF1
    GPIOA->MODER &= ~(3U << 2*2);
      GPIOA->MODER |= (2U << 2*2);
      // choose AF1 from mux
      GPIOA->AFR[0] &= ~(0xFU << 4*2);
      GPIOA->AFR[0] |= (1U << 4*2);


      //setup PA3 as AF1
      GPIOA->MODER &= ~(3U << 2*3);
      GPIOA->MODER |= (2U << 2*3);
      // choose AF1 from mux
      GPIOA->AFR[0] &= ~(0xFU << 4*3);
      GPIOA->AFR[0] |= (1U << 4*3);

      //setup uart2
      //reset uart cr1
      USART2->CR1 = 0;
      USART2->CR1 |= (1 << 3);  // TE
      USART2->CR1 |= (1 << 2);  // RE
      USART2->CR1 |= (1 << 5);  // RXNEIE

      USART2->BRR = (uint16_t)(SystemCoreClock / baud);


      USART2->CR1 |= (1 << 0);  // UE
```

```
}


void BSP_led_init(void){


    /* Enable GPIOC clock */
    RCC->IOPENR |= (1U << 2);


    /* Setup PC6 as output */
    GPIOC->MODER &= ~(3U << 2*6);
    GPIOC->MODER |= (1U << 2*6);

    /*Clear led*/
    GPIOC->BRR |= (1U << 6);

}
void BSP_led_set(void){

      /*trun-on led*/
    GPIOC->ODR |= (1U << 6);
}
void BSP_led_clear(void){

    /*Clear led*/
    GPIOC->BRR |= (1U << 6);
}
void BSP_led_toggle(void){

    /* Toggle LED */
    GPIOC->ODR ^= (1U << 6);
}

void BSP_button_init(void){
       /* Enable GPIOA clock */
    RCC->IOPENR |= (1U << 0);

          /* Setup PA1 as input */
          GPIOA->MODER &= ~(3U << 2*1);
          GPIOA->PUPDR |= (2U << 2*1); // Pull-down mode

          /*setup interrrupts for inputs*/
          EXTI->EXTICR[0] |=(0U << 8*1);//PA1    EXTI1 mux ta PA1
iÃ§in EXTICR0'Ä±n 9.biti 0 yapÄ±ldÄ±

          /* MASK*/
          EXTI->IMR1 |= (1U << 1);

          /*rising edge*/
          EXTI->RTSR1 |= (1U << 1);
```

```
          /*NVIC*/
          NVIC_SetPriority(EXTI0_1_IRQn,0);     // buton interruptÄ± PA1
iÃ§in EXTI1 in iÃ§erisinde olduÄŸundan EXTI0_1_IRQn kullanÄ±ldÄ±.
          NVIC_EnableIRQ(EXTI0_1_IRQn);   //buton interrupt Ä± nvic
iÃ§erisinde enable edildi
}
// returns 1 if button is pressed
int BSP_button_read(void){
      int b = ((GPIOA->IDR >> 1) & 0x0001);

      if (b) return 1;
      else return 0;
}
```

main.c:

```
/*
 * main.c
 *
 * author: Mehmet Akif GÃœMÃœÅ

 */

#include "bsp.h"

#define DELAY    16000U



int main(void) {

      BSP_system_init();

      while(1){

            uart_tx(uart_rx());

}
    return 0;
}
```

3. CONCLUSION

The objective of this lab is to get you communicate with MCU from PC, and utilize bidirectional data transmission and parsing. Problem 1 connect your board to the PC using UART protocol. In this problem I created an initialization routine for UART, and created send and receive functions. I did not use interrupts for this assignment. The applications made in the previous lesson were used and a uart protocol was created. The data received by the rx function was read and then this data was used in the tx function. So briefly the function representation is uart_tx (uart_rx ()). In problem 2 I implement a PWM signal and drive an external LED using varying duty cycles. The LED should display a triangular pattern, meaning the brightness should linearly increase and decrease in 1 second. I tried to write the code by reading the rm444 datasheet, but I couldn't succeed. I have failed to implement AFR (alternate function register) correctly. Many problems were successfully concluded, but other problems could not be completed because there was not enough time and the self-teach technique was used in this lesson.

4. References

- https://www.arrow.com/en/research-and-events/articles/what-is-uart-protocol-uart-communication-explained
- https://en.wikipedia.org/wiki/Duty_cycle#:~:text=Electrical%20motors%20typically%20use%20less,power%20delivery%20and%20voltage%20regulation.