GEBZE TECHNICAL UNIVERSITY

ELECTRONICS ENGINEERING

ELEC 335

Microprocessors Lab

LAB #5

Prepared by
1) 1801022037 Ömer Emre POLAT

1. Introduction

In this lab we will look into USART serial communication and pulse width modulation. Pulse width modulation will be used both alone and with a keypad input.

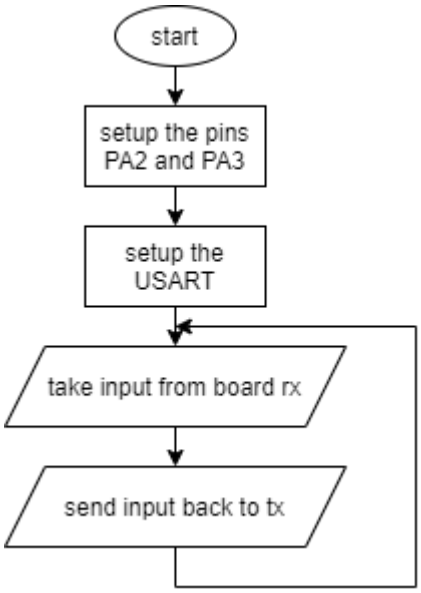
2. Problem 1

In this problem we will create a serial connection using USART functionality of the board. First, we can choose the pins for USART communication and draw a flowchart for the code.

Table 13. Port A alternate function mapping

Port	AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7
PA0	SPI2_SCK	USART2_CTS	TIM2_CH1_ETR	-	-	LPTIM1_OUT	-	-
PA1	SPI1_SCK/ I2S1_CK	USART2_RTS _DE_CK	TIM2_CH2	-	-	-	I2C1_SMBA	EVENTOUT
PA2	SPI1_MOSI/ I2S1_SD	USART2_TX	TIM2_CH3	-	-	-	LPUART1_TX	-
PA3	SPI2_MISO	USART2_RX	TIM2_CH4	-	-	-	LPUART1_RX	EVENTOUT
PA4	SPI1_NSS/ I2S1_WS	SPI2_MOSI	-	-	TIM14_CH1	LPTIM2_OUT	-	EVENTOUT
PA5	SPI1_SCK/ I2S1_CK	-	TIM2_CH1_ETR	-	-	LPTIM2_ETR	-	EVENTOUT
PA6	SPI1_MISO/ I2S1_MCK	TIM3_CH1	TIM1_BKIN	-	-	TIM16_CH1	LPUART1_CTS	-
PA7	SPI1_MOSI/ I2S1_SD	TIM3_CH2	TIM1_CH1N	-	TIM14_CH1	TIM17_CH1	-	-
PA8	MCO	SPI2_NSS	TIM1_CH1	-	-	LPTIM2_OUT	-	EVENTOUT
PA9	MCO	USART1_TX	TIM1_CH2	-	SPI2_MISO	-	I2C1_SCL	EVENTOUT
PA10	SPI2_MOSI	USART1_RX	TIM1_CH3	-	-	TIM17_BKIN	I2C1_SDA	EVENTOUT
PA11	SPI1_MISO/ I2S1_MCK	USART1_CTS	TIM1_CH4	-	-	TIM1_BKIN2	I2C2_SCL	-
PA12	SPI1_MOSI/ I2S1_SD	USART1_RTS _DE_CK	TIM1_ETR	-	-	I2S_CKIN	I2C2_SDA	-
PA13	SWDIO	IR_OUT	-	-	-	-	-	EVENTOUT
PA14	SWCLK	USART2_TX	-	-	-	-	-	EVENTOUT
PA15	SPI1_NSS/ I2S1_WS	USART2_RX	TIM2_CH1_ETR	-	-	-	-	EVENTOUT

As we can see from the table above we can use the PA2 and PA3 as the USART2 pins.



With the flowchart done we can write the actual code using the flowchart as a template.

bsp.h

```
#ifndef BSP_H_
#define BSP_H_

#include "stm32g031xx.h"

void BSP_system_init(void);

void BSP_IWDG_init(void);
void BSP_IWDG_refresh(void);

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

#endif
```

bsp.c

```
#include "bsp.h"

static volatile uint32_t tick = 0;

void BSP_IWDG_init(void)
{
    IWDG->KR = 0x5555;
    IWDG->PR = 1; // prescaler
    while(IWDG->SR & 0x1); // wait while status update
    IWDG->KR = 0xCCCC;
}

void BSP_IWDG_refresh(void)
```

```
{
    IWDG->KR = 0xAAAA;
}

void SysTick_Handler(void)
{
      if(tick > 0)
      {
            --tick;
      }
}

uint8_t uart_rx(void)
{
    volatile uint8_t data = (uint8_t) USART2->RDR;
    return data;
}

void uart_tx(uint8_t c)
{
    USART2->TDR = (uint16_t) c;
    while(!((USART2->ISR) & (1U << 6)));
}

void BSP_UART_init(uint32_t baud_rate)
{
    //Enable Port A clock
    RCC->IOPENR |= (1U << 0);

    //USART2 Enable
    RCC->APBENR1 |= (1U << 17);

    //PA2 as AF mode for USART
    GPIOA->MODER &= ~(3U << 2*2);
    GPIOA->MODER |= (2U << 2*2);

    //Choose AF1 from alternate functions
    GPIOA->AFR[0] &= ~(0xFU << 4*2);
    GPIOA->AFR[0] |= (0x1 << 4*2);


    //PA3 as AF mode for USART
    GPIOA->MODER &= ~(3U << 2*3);
    GPIOA->MODER |= (2U << 2*3);

    //Choose AF1 from alternate functions
    GPIOA->AFR[0] &= ~(0xFU << 4*3);
    GPIOA->AFR[0] |= (0x1 << 4*3);

    //UART2 Setup
    USART2->CR1 = 0; // reset control register

    USART2->CR1 |= (1U << 2); // Read enable
    USART2->CR1 |= (1U << 3); // Transfer enable


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

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

```
}

void BSP_system_init()
{
    __disable_irq();
    SysTick_Config(SystemCoreClock / 1000);
    //BSP_IWDG_init(); // Watchdog init
    //__enable_irq();
}
```

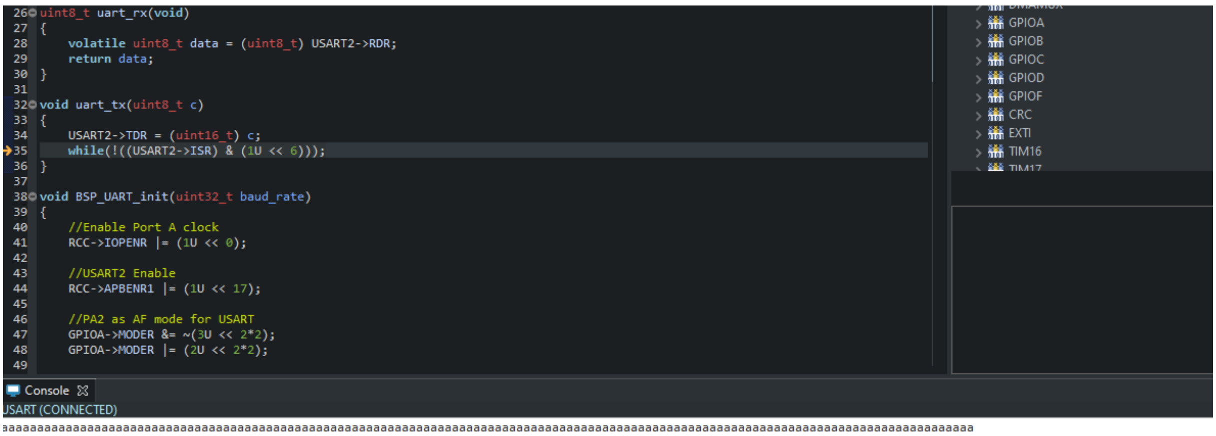
main.c

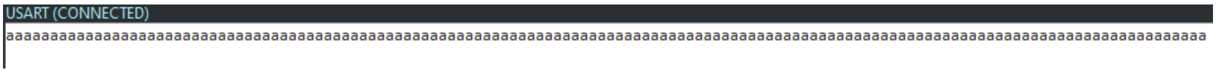
```
#include "bsp.h"
#include "stdlib.h"

int main(void) {

      BSP_system_init();
      BSP_UART_init(9600);

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

    return 0;
}
```

The code works by implementing and setting up a USART protocol. This is done by choosing the pins from the table above. The chosen PA2 and PA3 as the USART2 pins and they are set from MODER as Alternate function mode. After that we setup the USART2 from its CR1 register.

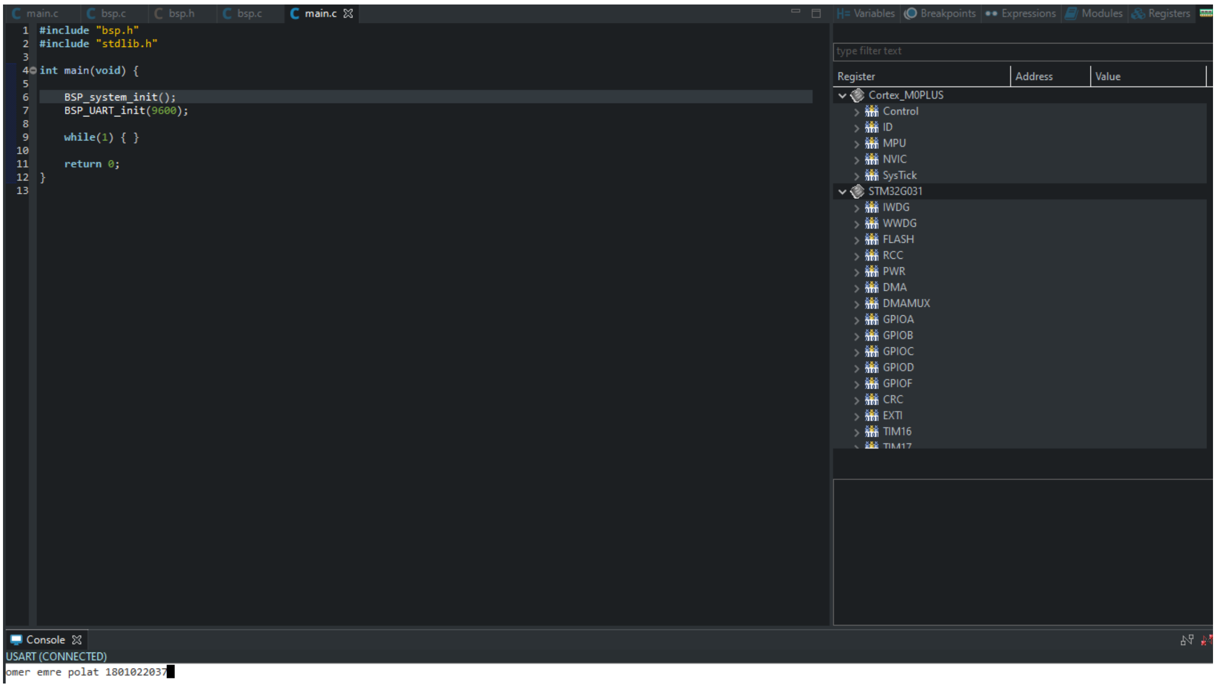
After all the setup is done, we use the implemented uart_rx and uart_tx functions to receive from pc and transmit information to pc. The watch dog timers are not used in the current problem.

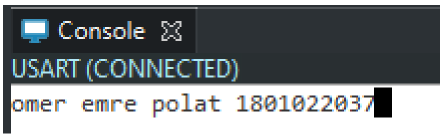
With the code done we can test the code using debug mode in STM32 IDE.





As we can see the console prints the sent character which is “a”. This continuous sending to pc occurs because we use polling to receive and transmit the characters. This event does not happen on interrupt-based code as its demonstrated in the console below.





As we can see on the console using interrupts fix the continuous transmission between the pc and board.

3. Results and General Comments

In this lab we have used USART serial communication to communicate between the board and the pc. This communication does not use an interrupt and instead uses polling, so it creates an infinite loop with the same input being sent to the pc. This does not happen when we try using interrupt routines to print the items because it only runs once when the interrupt is triggered while polling runs and prints constantly.

To fix the continuous transmission and getting stuck we can use interrupts on the code to fix this stuck on a single character problem. As we can see from the debug with the interrupt code, we can see that stuck one the same character problem is fixed with interrupts.

4. References

[1]. ELEC 335 Lecture Slides (5-6-7).