USART2 Transmitter with Modular Programming on STM32F401RE

1. Introduction

In this project, I implemented a **USART2 transmitter** on the STM32F401RE and restructured the code using **modular programming**. This approach enhances **code organization, reusability, and debugging efficiency**.

2. Code Structure

USART.h (Header File)

```
/*
 * usart.h
 *
 *  Created on: Feb 16, 2025
 *      Author: PY-DOM
 */

#ifndef USART_H_
#define USART_H_

void usart_init(void);
void usart_write(int ch);

#endif /* USART_H_ */
```

USART.c (Implementation File)

```
/*USAT TX is available at PA2 and it connect to AHB
 * SO AHB1ENR FOR CLOCK PLUSE Because it is connect to Port
A
 * Set PA2 to alternate function(MODER)
 * Set PA2 as USART TX (AFR[0])
 * Enable Clock for USART2 using ABH1ENR
 * set Baud Rate (BRR)
 * Enable transmitter (CR1)
 * Enable USART2 (CR1)
 * WAIT TILL THE LAST BYTE IS TRANSMITTED(SR)
 * send data byte*/
#include "stm32f4xx.h"
#define GPIOAEN (1U<<0)
#define USART2EN (1U<<17)
```

```c
#define PERIPHCLK 16000000
#define BAUDRATE 115200

#define TE (1U<<3)
#define UE (1U<<13)
#define TXE (1U<<7) //it ensure the last bit data
transfered

void usart_init(void){
    /*Enable clock for PORT A*/
    RCC->AHB1ENR |= GPIOAEN;
    //Set PA2(USART_TX) to alternate function
    GPIOA->MODER &=~(1U<<4); // bit4 =0
    GPIOA->MODER |= (1U<<5); // bit5 =1
    /*we need to set PA2 TO USART_TX Using AFR[0]
     * which is AFRLy: Alternate function register low
     * for port x bit y(y=0..7)
     * IF need to configure more than bit 7 , we can use
     * AFR[1] WHICH AFRHy for PORT x Bit y (y=8..15)
     * reference to alternate function mapping we have
USART_TX
     * at column AF07 to set this Bit11 ->0 , Bit10,9,8->1
     */
    GPIOA->AFR[0]&=~(1U<<11); //BIT11->0
    GPIOA->AFR[0]|= (1U<<10);
    GPIOA->AFR[0]|= (1U<<9);
    GPIOA->AFR[0]|= (1U<<8);
    // clock for USART2_TX
    RCC->APB1ENR |= USART2EN;
    /*Set baudrate*/
    USART2->BRR = (PERIPHCLK + (BAUDRATE/2))/BAUDRATE;
    //ENABLE transmitter CR1 bit3->1
    USART2->CR1 = TE;
    //enable USART CR1 bit13->1
    USART2->CR1 = UE;
}

void usart_write(int ch){
    // to wait until the last BYTE of data transfered
    while((USART2->SR & TXE)==0);
```

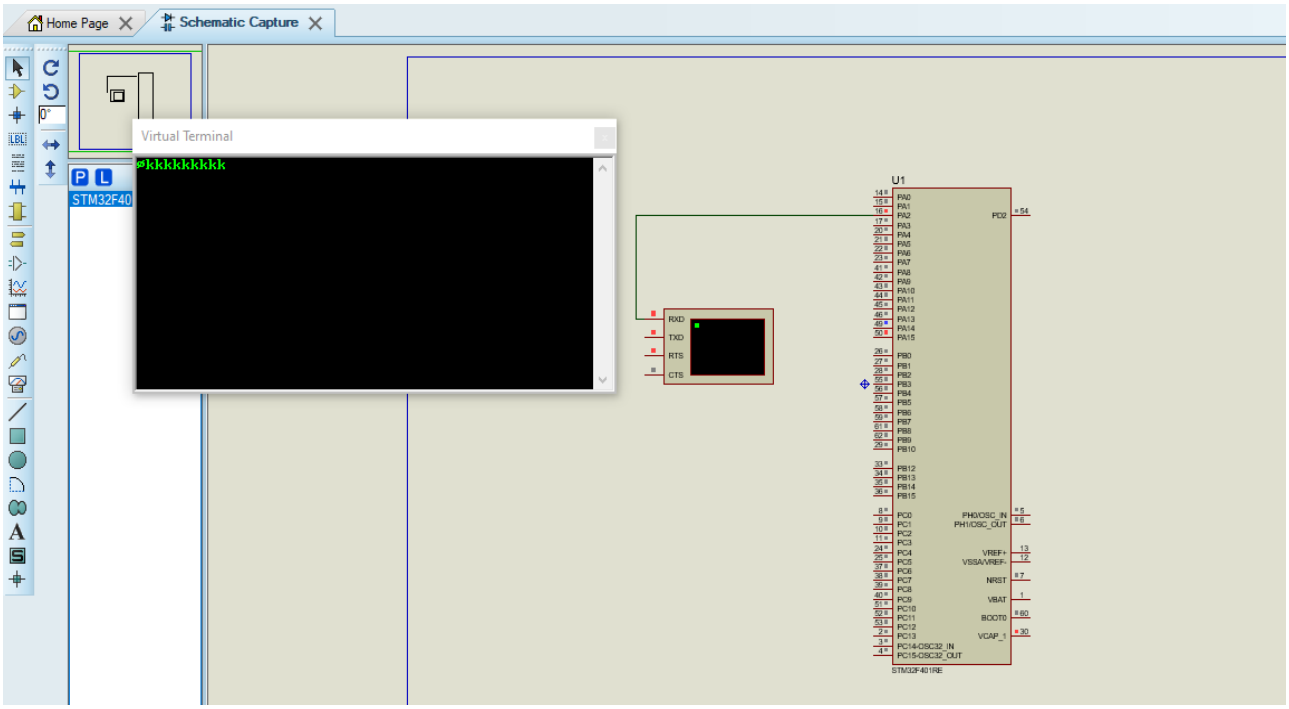
```
    // if '&' operation give 1 then it show the data is
transfered
    /*send the data*/
    USART2->DR = (ch & 0xff);
}
```

main.c (Main Program)

```
/*USART Transmit a single character Using Modular code
 * the actual function written in "USART.C"
 * Created a header for calling the function
 * by using this type of method we can reduce the repeated
code
 * for the same function for different operation
 *also for simple and structured Main.c file
 */
#include "stm32f4xx.h"
#include "usart.h"

int main(void){
    usart_init(); // calling the function Initialization
    // it get all the initial steps from Usart.c by Usart.h
    while(1){
    usart_write('k');
    for(int i=0;i<1000000;i++);
    }

}
```



4. Conclusion

- Modular programming has **simplified** the main function, making code easier to read and maintain.
- The USART2 transmitter **successfully** sends messages at **115200 baud rate**.
- Next steps: Implementing **string transmission enhancements** and **USART reception**! 🚀