A Comparative Analysis of Synchronous USART And Asynchronous UART Communication Protocols

Meiting Dong *

School of Information and Control Engineering, Qingdao University of Technology, Qingdao, 266520, China

* Corresponding Author Email: 201805650110@smail.xtu.edu.cn

Abstract. In contemporary applications, the integration of the STM32F103ZET6 microcontroller with various peripherals is indispensable, emphasizing the critical role of serial communication in data transmission between the microcontroller and these peripherals. This study delineates the distinctions between Universal Synchronous Asynchronous Receiver-Transmitter (USART) and Universal Asynchronous Receiver-Transmitter (UART) within the context of USART synchronous communication. The superiority of USART is underscored by its utilization of a uniform clock and identical baud rate for simultaneous and synchronous transmission, enhancing data transfer efficiency significantly. This methodology facilitates the transmission of larger data quantities concurrently and mitigates data loss during prolonged transmission, a prevalent issue in UART asynchronous communication. Additionally, USART preserves essential functionalities such as the detection of data boundaries, automatically discerning the initiation and termination of individual data frames, which guarantees the uninterrupted reception and transmission of data. This analysis is pivotal for professionals seeking optimized communication protocols in microcontroller-based systems.

Keywords: Stm32f103zet6 microcontroller, USART, UART, Serial communication.

1. Introduction

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) functions as a versatile serial interface facilitating both synchronous and asynchronous data communication within the STM32F103ZET6 microcontroller, essential for interaction between various peripherals. Specifically, this microcontroller incorporates three distinct USART interfaces, designated as USART1, USART2, and USART3 [1]. Conversely, the Universal Asynchronous Receiver and Transmitter (UART), another integral interface, supports asynchronous communication specifically. Within the same microcontroller environment, two UART interfaces are identifiable, referred to as UART4 and UART5 [2, 3].

In contemporary avionic flight control systems, there exists a limitation pertaining to connectivity, with a maximum of five UART interfaces for peripheral linkage. This constraint poses challenges in an ecosystem requiring extensive peripheral interactions. To navigate this, microcontrollers like the STM32F103ZET6 can interface with any available UART port on the flight control, facilitating real-time data transactions between the control system and the microcontroller, thereby achieving comprehensive control over connected peripherals.

This study accentuates the efficaciousness of the STM32F103ZET6's USART interface in synchronous serial communication, enabling simultaneous data reception and transmission between the flight control system and its peripherals. This duality not only propels efficiency in data throughput but also circumvents potential data loss typically associated with asynchronous communication, attributable to disparate clock configurations. Thus, USART emerges as a robust solution for high-stakes applications such as avionic systems, necessitating reliable, high-speed data exchanges.

2. Basic Theory Analysis

The schematic diagram of single-chip microcomputer stm32f103zet6 is as follows in figure 1 [4].

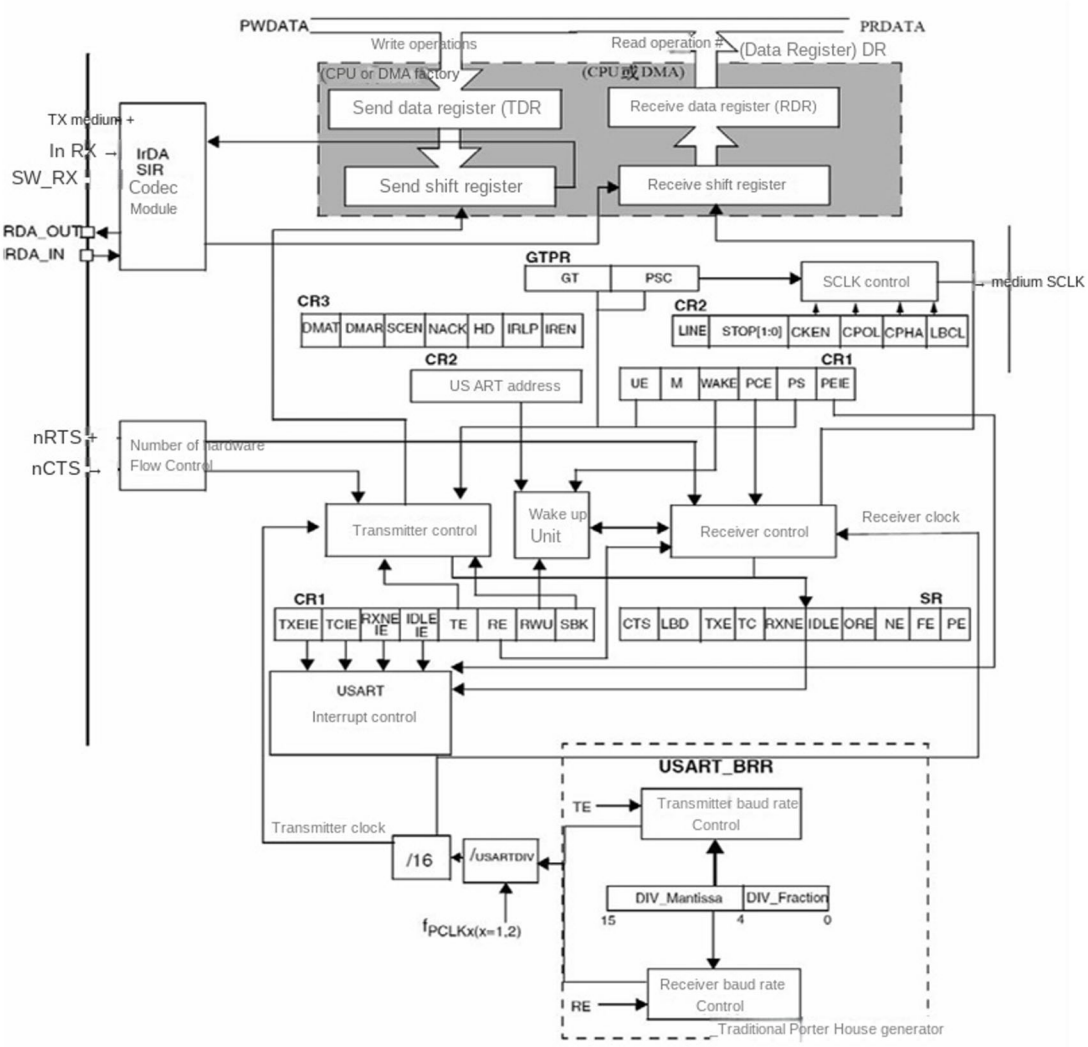


Fig. 1 MCU schematic diagram [4]

2.1. Pin

The serial port and pins of stm32f103zet6 are as follows in table 1.

Table 1. USART pin of STM32F103ZET6 chip

PIN	APB1	APB2			
	USART1	USART2	USART3	UART4	UART5
TX	PA9	PA2	PB10	PC10	PC12
RX	PA10	PA3	PB11	PC11	PD2
SCLK	PA8	PA4	PB12		
nCTS	PA11	PA0	PB13		
nRTS	PA12	PA1	PB14		

TX is used for sending data, and RX is used for receiving data. SCLK is a Clock. And it is used only for synchronous communication. nRTS is used as the pin for requesting to send. nCTS is used as the pin for clearing to send. It is indicating that the low level is valid.

2.2. Data Register

Send data register TDR and receive data register RDR are both included in the data register. A single address translates to two physical memory blocks.

Data that has been sent or received is contained in USART_DR. In reality, USART_DR is made up of two registers: a readable RDR for receiving and a writable TDR for sending. Data written to

the USART_DR during transmission is automatically stored in the TDR. Reading data to USART_DR automatically pulls RDR data during read operations [5].

2.3. Controller

A wake-up unit, interrupt control, a transmitter that regulates sending, a receiver that controls receiving, and other components make up USART. You must enable the USART in UE position 1 of the USART_CR1 register before utilizing it. The serial port clock is turned on via the UE bit.

When sending data, several important flag bits: TE indicates the enable sending flag bit. The TXE flag bit indicates that the send register is empty and can be used when sending a single byte. The TC flag bit occurs when sending multiple bytes of data is complete. TXIE signals send complete interrupt enabled.

2.4. Baud Rate

USART_BRR is baud rate register [6]. USART_CR1 is OVER8.

Decimal baud rate generation: The rate at which the signal modifies the carrier is measured by the baud rate index. The number of carrier modulation state changes per unit of time is used to express it. It is a baud unit. In terms of bit/s(bps), bit rate is the quantity of bits transmitted per unit of time. For USART baud rate and bit rate equal, the two concepts will not be distinguished in the future. The larger the baud rate, the faster the transfer rate.

$$\frac{\text{Tx}}{\text{Rx}} baud\ rate = \frac{f}{16*USARTDIV} \quad (1)$$

2.5. Interrupt

USART has multiple interrupt request events, as shown in Table 2 [7].

Table 2. Interrupt flag and control bit

Interrupt event	Time stamp	Enable control
The send data register is empty	TXE	TXEIE
CTS mark	CTS	CTSIE
Send complete	TC	TCIE
Be ready to read the received data	RXNE	RXNEIE
An overflow error was detected	ORE	

3. Research Method

Empirical research method, according to the need for high-speed and effective data transmission between the single chip microcomputer and various peripherals, put forward the research on the synchronous communication mode of USART, using STM32F103ZET6 for data transmission, and using Jlink to test and debug the program during the research process.

4. Results and Discussion

USART adds synchronization function on the basis of UART, that is, USART is an enhancement of UART. And it is indeed so.

First, it needs to configure serial port USART2 to enable channel 2 and clock A [8]. In the process of this research, only the RX and TX modes of serial ports are involved, and the USART2 sending data port PA2 and receiving data port PA3, which have been designed by the single chip microcomputer itself, need to initialize the ports PA2 and PA3. Common baud rates of serial port are 9600, 39200, 115200, etc. In this study, the baud rate of 115200 is adopted. Since there is no mandatory requirement for serial port USART2 priority in this experiment, the sub-priority and preemption priority can be set to the minimum 3 for the time being.

In the function that performs serial port initialization, the serial port initialization needs to be turned on [9]. When writing the send function, it should be noted that the setting of the frame header and frame tail is very important, and the sum check, odd check, even check can be set if necessary. Use and check more in-flight control. In digital logic circuits, odd check and even check are often used. However, if in the process of data transmission, there is an error in the reception of multi-bit data, then the odd check and even check are used, and the conclusion will be no longer accurate.

Here 0xAA and 0xAF are used as the frame header of the required data string. And the flag bits are added up after the frame header is detected by the serial port interrupt at the receiving end. 0xcd as the frame end of the data, indicating the completion of data sending or receiving; 0x08 is the number of bits that represent the actual data transferred.

The program stores the data that needs to be sent to the array. When the send interrupt is turned off, it will reopen the send interrupt to send the data in the array. The serial port of USART2 is interrupted. Receiving interruption means that the serial port enters an interruption every time a bit of data is received. And the received data is stored in the variable com_data. The send interrupt is to send the data stored in the array BUFF [10]. And every time the data is sent, it enters the interrupt once. It should be noted that avoid adding printf function to the interrupt of receiving data, which will easily cause the send interrupt of receiving data to be added, and the data will conflict.

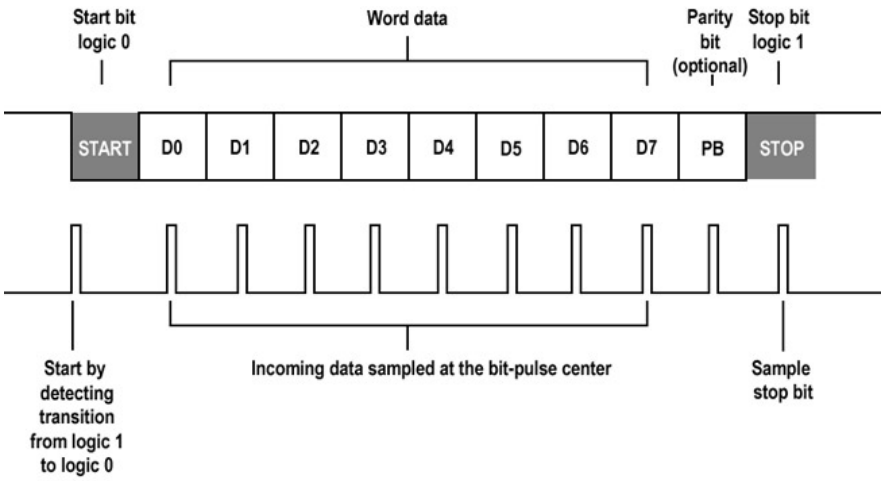


Fig. 2 Serial communication configures byte receiving process timing relationships [10]

Data processing function idea as shown in Figure 2, by the program easy to receive each frame header of a data frame, the flag bit will be added one; When the maximum number of data bits that can be stored in the array is not reached, the flag bit is not increased. When the maximum number of bits that can be stored in the array is reached, the next flag bit is entered. And the judgment at the end of the frame is entered, that is, all the data sent has been received and the next data frame is also received. During the process of enabling the serial port, after measuring with a voltage pen, it is found that the voltage has been maintained at about 3.3V.

Both parties in synchronous communication use the same clock. It is the key characteristic that sets synchronous communication apart from asynchronous communication. Each character in asynchronous communication must employ the start bit and the stop bit as a symbol for the beginning and end of the character, which adds time to the process. As a result, synchronous communication is frequently used instead of these indicators in data block transmission to increase communication speed. The sending end and the receiving end are synchronized by the clock in synchronous communication, which means that after the designated synchronization characters are recognized, data is constantly delivered in sequence until a piece of data is transmitted. During synchronous transmission, there is no gap between characters, no start bit and stop bit, and synchronous characters can be omitted, as shown in Figure 3 [5].

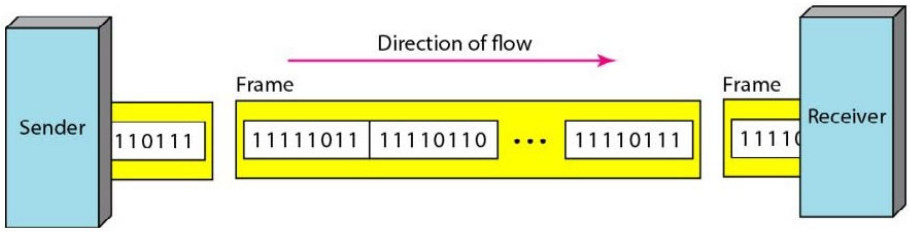


Fig 3. Data format for synchronous transmission [5]

However, when the serial port uses synchronous serial communication, just change the if (USART_GetITStatus (USART2, USART_IT_RXNE)) and if (USART_GetITStatus (USART2, USART_IT_TXE)) of the if statement. That is, the receiving and sending interrupts are always open.

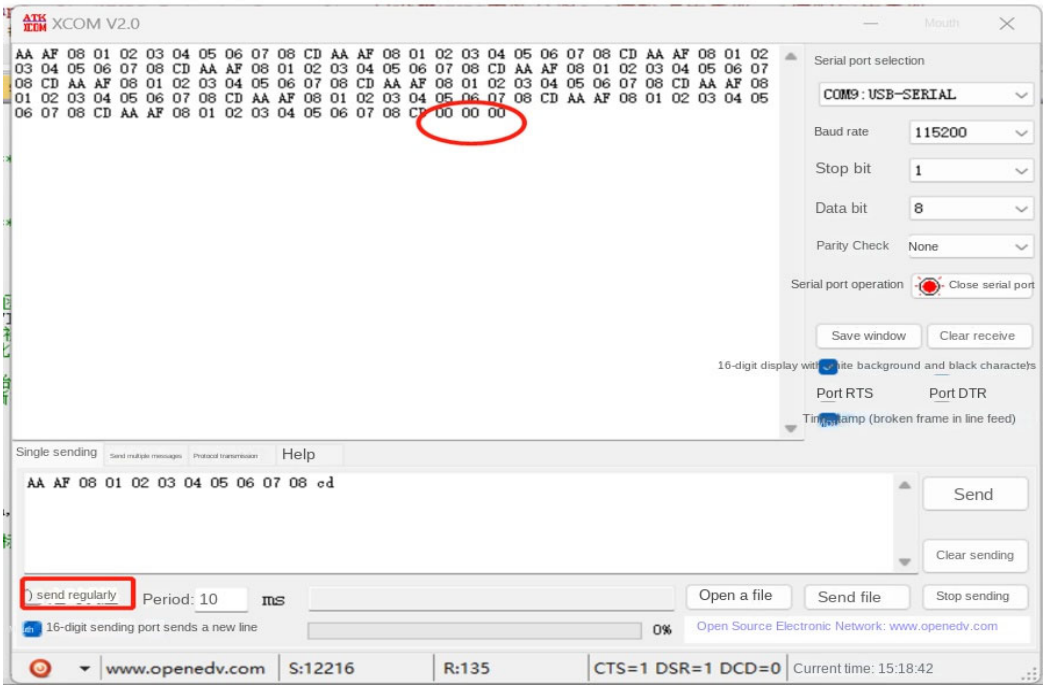


Fig. 4 Initial results show (Photo/Picture credit: Original)

As shown in Figure 4, when the data sending end sends data to the MCU, the function of sending data is triggered. However, when no data is sent, the MCU has been sending useless data 0.

5. Expansion and Optimization

5.1. Optimization 1: Implementing Synchronous Loopback for Efficient Data Transmission

Since USART is in the state of synchronous sending and receiving, 0 will be continuously sent without sending data. To avoid this, add the following code to the main.c file in figure 5.

```
while(1)
{
  if(Tx_flag)
  {
    DrvUart2SendBuf(Tx_data, data_len_);
    Tx_flag = 0;// clear the flag bit after sending.
  }
```

Fig. 5 Optimization process (Photo/Picture credit: Original)

The optimized data situation is shown in figure 6.

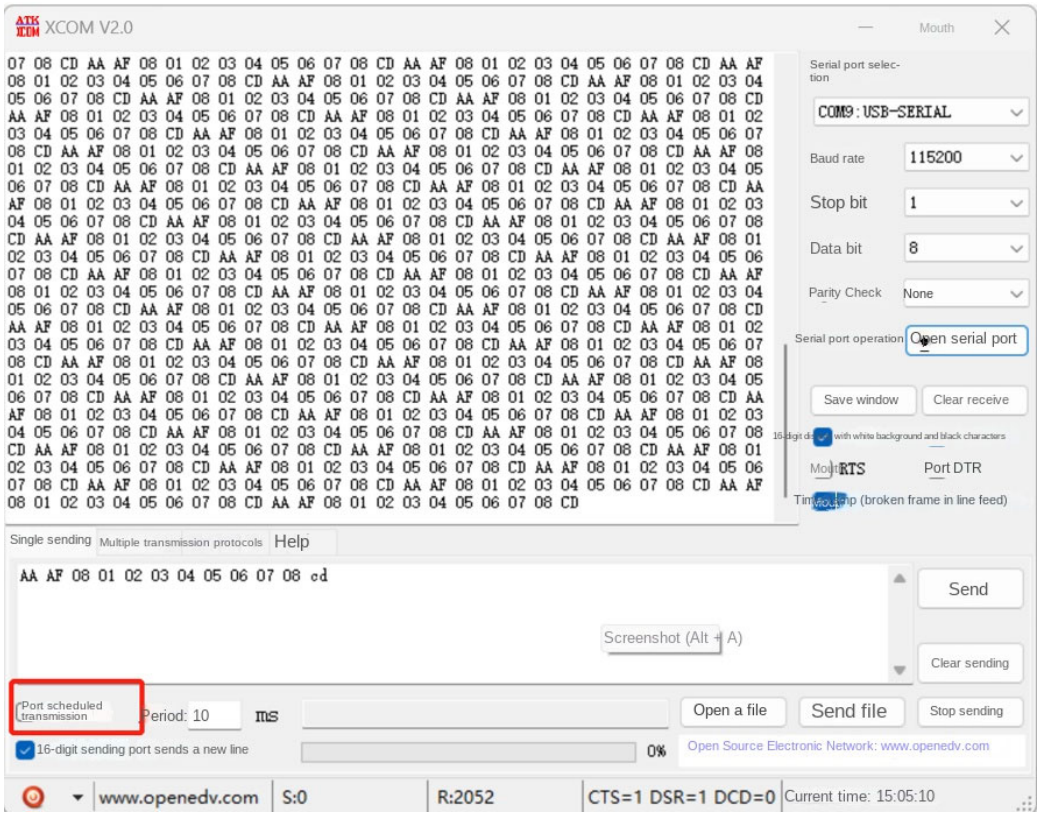


Fig. 6 Optimization 1 result (Photo/Picture credit: Original)

That is, it can send the data at the same time when the data is received, and it can close the data transmission when the data is not received. It can realize synchronous loopback transmission of data between two single chip computers.

5.2. Optimization 2: Utilizing Timer 8 for Precise Data Transfer Intervals

In order to utilizing timer 8 for precise data transfer intervals, here's the flow:

1. Initialization: Begin by initializing a structure TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure; designated for the timer's configuration. The clock for TIM8 is enabled using RCC_APB2PeriphClockCmd (RCC_APB2Periph_TIM8, ENABLE);.

2. Timer Configuration: The clock prescaler is set to 360, meaning that the timer clock is divided by this value. The timer's counter will automatically reload at a count of 9999, thereby setting the timing period to approximately one second when combined with the prescaler. The counter mode is configured to count upwards with TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;.

3. Interrupt Configuration: An interrupt for TIM8 is configured using the Nested Vectored Interrupt Controller (NVIC) structure. The preemption priority and subpriority for the interrupt are set to 3. The interrupt is then enabled with NVIC_InitStructure. NVIC_IRQChannelCmd = ENABLE;.

4. Interrupt Flag Clearing and Timer Start: Any existing interrupt flags for TIM8 updates are cleared. The timer's interrupt for updates is enabled, allowing the timer to generate interrupts. The timer (TIM8) is then started, commencing the count.

5. Interrupt Service Routine (ISR): The function void TIM8_UP_IRQHandler(void) acts as the interrupt service routine for the TIM8 update event. Inside the ISR: It checks if the update interrupt flag is set, indicating that the timer has reached its predefined count. If set, the flag is cleared, resetting the interrupt. A data sequence 1,2,3,4,5,6,7,8 is sent, presumably to another function or hardware, each time the ISR is triggered.

The program can control the serial port to send data every second. It should be noted that timer 8 is a clock with a pre-division coefficient of 360, rather than a timer similar to timer 3 that uses the main frequency division coefficient of 720 [8]. The results are as follows in figure 7.

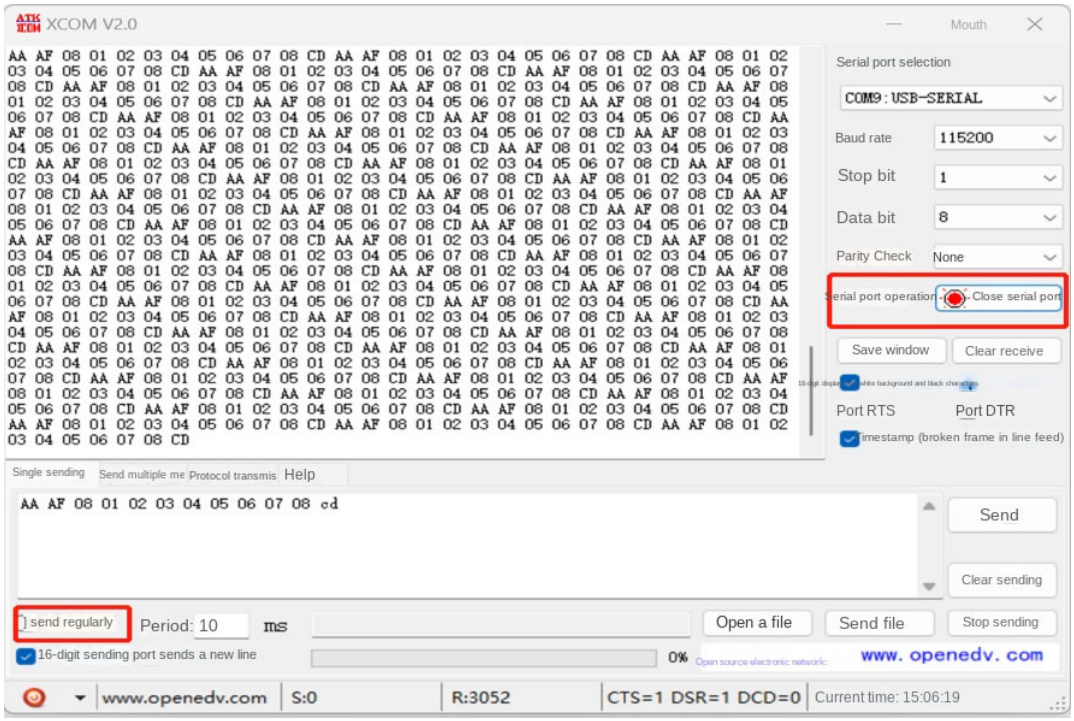


Fig. 7 Results of optimization 2 (Photo/Picture credit: Original)

6. Conclution

In a broader context, the USART's synchronous communication capability, when set up correctly with appropriate pin configurations, often necessitates the utilization of arrays. These arrays are pivotal in holding data designated for both sending and receiving operations. The actuation of data transmission and reception is meticulously managed through the judicious toggling of interrupts. Furthermore, to enhance this management, one can employ either a signal bit or perhaps a timer, subsequent to both data receipt and dispatch, to regulate the activation and deactivation of data communication channels.

Despite these mechanisms, the synchronous communication methodologies associated with USART encounter significant hurdles, especially in scenarios demanding rapid and voluminous data transmission. In such situations, the simultaneous occurrence of transmission errors across multiple data sets is not an uncommon phenomenon. Consequently, conventional error-checking strategies, such as parity checks (odd or even), may fall short of addressing these challenges effectively. Opting for a checksum-based approach introduces its own set of complications; for instance, continuously accumulating checksum bits over multiple iterations can dramatically impede the overall data transfer rate, introducing considerable latencies.

Given these complexities, it becomes evident that there is a pressing need for more advanced research in this domain. Future studies should pivot towards not only enhancing the fidelity of data transfers but also ensuring impeccable data transmission precision.

References

[1] Di Giusto C, Guizouarn L G, Lozes E. Multiparty half-duplex systems and synchronous communications. Journal of Logical and Algebraic Methods in Programming, 2023, 131: 100843.

[2] Akhoury A, Birla K, Sarkar R, et al. Design and analysis of RTOS and interrupt-based data handling system for nanosatellites. 2019 IEEE Aerospace Conference. IEEE, 2019: 1-9.

[3] Duesterberg D. Circuit for generating a sampling signal for a UART interface, and by comparing values stored in peripheral clock memory: U.S. Patent 10,705,991. 2020-7-7.

[4] Jin H, Sun X. Design of the Brushless DC Motor Driving System Based on STM32. Open Access Library Journal, 2017, 4(6): 1-9.

[5] La X. Research on Key Technologies and Innovative Development of Electronic Communication System Based on Serial Communication. Journal of Physics: Conference Series. IOP Publishing, 2020, 1549(5): 052068.

[6] Chang S, Yang J, Yang J, et al. The experimental research of UV communication. Wireless Communications and Networks. SPIE, 2004, 5284: 344-348.

[7] Mejia-Alvarez P, Leyva-del-Foyo L E, Diaz-Ramirez A. Interrupt Handling Schemes in Operating Systems. Springer International Publishing, 2018.

[8] Oh, S W, Seo Y S, Da In I M. Semiconductor apparatus capable of synchronizing command signal and clock signal, and operation method thereof: U.S. Patent 10,796,737. 2020-10-6.

[9] Shi G. Method, client device and data forwarding device for transmitting data through serial port: U.S. Patent 10,346,344. 2019-7-9.

[10] Son S C, Kim N W, Lee B T, et al. A time synchronization technique for coap-based home automation systems. IEEE Transactions on Consumer Electronics, 2016, 62(1): 10-16.