

Universal Serial Library 1.1

- Ready to give up on **STM32 UART DMA**?
- Here's the simple method that actually works
- No need to learn NOW, just repeat

UART

STM32U575

STM32F407

STM32H743

STM32G474

STM32F103

STM32L476

The Problem

- Why can't it just be this simple?

```
COM_Read(); // Read data  
COM_Write(); // Write data
```

The Problem

- Why can't it just be this simple?

```
COM_Read(); // Read data  
COM_Write(); // Write data
```

It can.

But what you usually face is:

- Should you use DMA or interrupts?
- How to handle buffer overflows?
- Your UART code still fails

Then how do you implement **COM_Read()** and **COM_Write()** in a way that is both simple and robust?

The Solution

- Don't reinvent the wheel
- Proven approach used in real devices
- Use it now – learn how it's made later

Start by including universal serial library:

```
#include "uart.h" // Universal serial library
```

After this, you will be able to work with UART using simple functions:

```
COM_Init();    // Initialize  
COM_Read();    // Read  
COM_Write();   // Write
```

The Solution

It's deeper than you might expect:

```
#include "uart.h" // Universal serial library

// Basic functions
COM_Init();      // Initialize
COM_Read();      // Read
COM_Write();     // Write

// Additional functions
COM_ReadFast();  // Read (no timeout)
COM_SetCallbacks485(); // RS-485 callbacks
COM_Select();    // Select port
COM_SetTimeout(); // Custom timeout
COM_RxFail();    // RX failure check
COM_CleanRxFail(); // Clear error flag
```

And if you need to **printf()** to the serial, include:

```
#include "uart_stdio.h" // printf() helper
```

The Solution

For now, these are sufficient:

```
COM_Init();    // Initialize
COM_Read();    // Read
COM_Write();   // Write
```

These functions are DMA-based:

- Circular DMA is essential for reliable reception
- DMA frees MCU resources during transmission

Forget the common but misleading belief that UART should use polling, interrupts, or DMA depending on the task. **It does not.**

Trust me — I learned this the hard way, building systems where UART failures cost real time and money.

The Only “Hard” Part

Before everything runs smoothly, here is how you should configure UART in STM32CubeMX:

- Connectivity ⇒ USART ⇒ DMA Settings tab
- Add USART_TX and **USART_RX** DMA Requests
- Set **USART_RX** DMA Mode to **Circular DMA**
- NVIC Settings tab ⇒ Enable Global Interrupt
- Parameter Settings tab ⇒ Set your Baud Rate

And finally, add universal library initialization:

```
// Circular DMA receive buffer
#define UART_BUFFER_SIZE 64
__IO uint8_t uartBuffer[UART_BUFFER_SIZE];

// Initialize universal library
COM_Init(0,           // Port number
         &huart1,     // HAL UART
         uartBuffer,   // RX buffer
         UART_BUFFER_SIZE); // Buffer size
```

LED Blink Control

Upgrade any LED blink code with serial control:

```
if (COM_ReadFast(&cmdByte, 1))
switch(cmdByte)
{
    case 0x01:
    {
        // Turn LED Off
        HAL_GPIO_WritePin(GPIOC,
                           GPIO_PIN_13,
                           GPIO_PIN_SET);

        break;
    }

    case 0x02:
    {
        // Turn LED On
        HAL_GPIO_WritePin(GPIOC,
                           GPIO_PIN_13,
                           GPIO_PIN_RESET);

        break;
    }
}
```


Send Data to PC

Respond to command 0x03 with an array:

```
case 0x03:
{
    uint8_t array[5] = {1, 2, 3, 4, 5};
    COM_Write(array, sizeof(array));
    break;
}
```

Data Manipulation

Sum received numbers and send the result:

```
case 0x04:
{
    uint8_t a, b;
    COM_Read(&a, 1);
    COM_Read(&b, 1);

    uint8_t c = a + b;
    COM_Write(&c, 1);
    break;
}
```

Thank You!

- Get the Code & Subscribe
- Comment what you'd like to see in the next video