Embedded_session_8

Embedded System Session 8 - Communication Protocols & UART

Created: 2025-07-22
Author: Fares Hesham Mahmoud
Tags: AVR, Embedded system, Communication Protocol, UART
Status: #Formatted

اللهمَّ علِّمنا ما ينفعنا، وانفعنا بما علمتنا، وزدنا علمًا. وافتح علينا فتحًا عظيمًا.

Communication Protocols Overview

Communication protocols define the way of connection between any two entities (micro-micro, micro-server, etc.).

Communication Medium

Wired Communication

Both devices are connected with physical wires (e.g., UART, SPI, I2C).

Parallel Connection

- **Data Width:** 8-bit data needs 8 wires
- **Speed:** Fast transmission (all bits sent simultaneously)
- **Example:** Sending value 7 (11100000) in 1ms

Serial Connection

- **Data Width:** Only one wire connects the devices
- **Speed:** Slower transmission (bits sent sequentially)
- **Example:** Same value 7 takes 8ms (8× longer)

Wireless Communication

Data transmitted through radio waves, infrared, or other wireless methods.

Types of Communication Architecture

1. Peer-to-Peer (UART)

- **Participants:** Only two devices
- **Authority:** Both devices have equal communication rights
- **Connections:**
  - $TX_1 \rightarrow RX_2$
  - $RX_1 \leftarrow TX_2$
- **Characteristics:**
  - Typical baud rates: 9600, 115200 bps
  - No clock signal required

2. Master-Slave Architecture

Master has authority to initiate communication; slaves respond only when requested.

Configuration	Example Protocols	Description
Single Master, Single Slave	Radio communication	Simple point-to-point
Single Master, Multi Slave	SPI, LIN	Master controls multiple slaves
Multi Master, Multi Slave	I2C	Multiple masters can control slaves

Data Direction Modes

Mode	Description	Example
Simplex	One-way communication only	Radio broadcast
Half Duplex	Two-way, but not simultaneous	Walkie-talkie
Full Duplex	Two-way simultaneous communication	Phone call, UART, SPI

Synchronization Methods

Synchronous Communication

- **Clock Signal:** Shared clock between devices

- Dynamic Data Rate: Can change baud rate during runtime
- Examples: SPI, I2C

Asynchronous Communication (UART)

- No Clock Signal: Each device uses its own clock
- Fixed Baud Rate: Must be preset and identical on all devices
- Challenge: Clock synchronization drift over time

UART (Universal Asynchronous Receiver Transmitter)

UART Frame Structure

Frame Components

Each UART transmission consists of a frame with the following structure:

Component	Size	Purpose	Notes
Start Bit	1 bit	Signals beginning of data	Always low (0)
Data Bits	5-9 bits	Actual data payload	Usually 8 bits
Parity Bit	0-1 bit	Error detection	Optional: even/odd/none
Stop Bits	1-2 bits	Signals end of frame	Always high (1)

Frame Format Visualization

```
Idle → Start → Data[0] → Data[1] → ... → Data[7] → Parity → Stop → Idle
 1   →   0   →    ?    →    ?    → ... →    ?    →   ?    →  1   →  1
```

Throughput Calculation

Throughput = Data Bits / Total Frame Bits

Throughput Examples

Minimum Configuration (5 data bits):

```
Total bits = 1 (start) + 5 (data) + 1 (stop) = 7 bits
Throughput = 5/7 ≈ 71.4%
```

Maximum Configuration (9 data bits):

```
Total bits = 1 (start) + 9 (data) + 1 (stop) = 11 bits
Throughput = 9/11 ≈ 81.8%
```

Common Configuration (8 data bits, parity, 2 stop bits):

```
Total bits = 1 + 8 + 1 + 2 = 12 bits
Throughput = 8/12 ≈ 66.7%
```

UART Data Registers

In microcontrollers, UART typically uses shared registers:

```c
// Transmission and Reception often share the same address
#define UDR_ADDRESS 0xFF    // Example address

// Writing to UDR sends data
UDR = 'A';  // Transmit character 'A'

// Reading from UDR receives data
char received = UDR;  // Receive character
```

UART Configuration Parameters

Baud Rate

Common baud rates and their applications:

Baud Rate	Use Case	Notes
9600	Basic communication, debugging	Most common, reliable
19200	Moderate speed applications	Good balance
38400	Faster data transfer	Still widely supported
115200	High-speed applications	Modern standard

Parity Options

Type	Description	When to Use
None	No error checking	Clean environments, higher throughput
Even	Even number of 1s	Basic error detection
Odd	Odd number of 1s	Basic error detection

UART Implementation Example

```
// UART Initialization
void UART_Init(uint32_t baud_rate) {
    uint16_t ubrr = F_CPU/16/baud_rate - 1;

    // Set baud rate
    UBRRH = (uint8_t)(ubrr >> 8);
    UBRRL = (uint8_t)ubrr;

    // Enable receiver and transmitter
    UCSRB = (1<<RXEN) | (1<<TXEN);

    // Set frame format: 8 data bits, 1 stop bit, no parity
    UCSRC = (1<<URSEL) | (1<<UCSZ1) | (1<<UCSZ0);
}

// Transmit single character
void UART_Transmit(uint8_t data) {
    // Wait for empty transmit buffer
    while (!(UCSRA & (1<<UDRE)));

    // Put data into buffer, sends the data
    UDR = data;
}

// Receive single character
uint8_t UART_Receive(void) {
    // Wait for data to be received
    while (!(UCSRA & (1<<RXC)));

    // Get and return received data from buffer
    return UDR;
}

// Transmit string
```

```c
void UART_SendString(const char* str) {
    while (*str) {
        UART_Transmit(*str++);
    }
}
```

UART Advantages and Limitations

Advantages

- **Simple Implementation:** Easy to understand and implement
- **Widespread Support:** Available on most microcontrollers
- **Full Duplex:** Can send and receive simultaneously
- **No Clock Required:** Asynchronous operation
- **Point-to-Point:** Direct communication between two devices

Limitations

- **Speed Limitations:** Generally slower than synchronous protocols
- **Distance Limitations:** Signal integrity degrades over long distances
- **Two-Device Limit:** Cannot easily expand to multiple devices
- **Clock Synchronization:** Both devices must agree on timing
- **No Built-in Addressing:** Cannot distinguish between multiple senders

Error Detection and Handling

⚠ Common UART Errors ›

- **Framing Error:** Stop bit not detected at expected time
- **Overrun Error:** New data received before previous data was read
- **Parity Error:** Calculated parity doesn't match received parity
- **Break Condition:** Extended low signal (simulates very long start bit)

```c
// Error checking example
uint8_t UART_ReceiveWithErrorCheck(void) {
    // Wait for data
    while (!(UCSRA & (1<<RXC)));

    // Check for errors
    if (UCSRA & ((1<<FE) | (1<<DOR) | (1<<PE))) {
```

```c
        // Handle error condition
        return 0xFF;  // Error indicator
    }

    return UDR;  // Return valid data
}
```

References

- ATmega32 USART Documentation
- RS-232 Standard Specification
- Serial Communication Best Practices