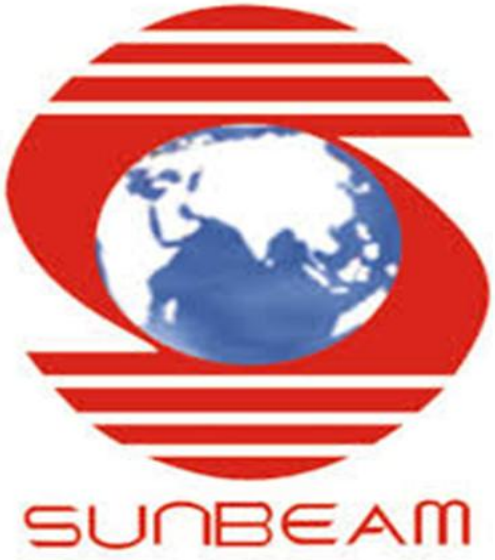


ARM Microcontroller Internship Programme

Trainer: *Kiran Jaybhave*

Sunbeam – Industrial Training Programme

Sunbeam Pune (Hinjawadi)

Communication Protocols in Embedded Systems

- **Understanding Serial vs Parallel Communication**
  - Communication between devices can happen in two fundamental ways, each with distinct characteristics and use cases.
- **Serial Communication**
  - **Definition**: One bit is transferred at a time over a single data line
  - **Advantages**:
    - **Requires fewer wires (typically 2–3 lines)**
      - Uses only TX, RX, and GND, which simplifies wiring and reduces pin usage.
    - **Smaller circuit footprint**
      - Fewer external components and connections result in compact hardware design.
    - **Better for long-distance communication**
      - Can reliably transmit data over longer cables compared to parallel communication.
    - **Less susceptible to electromagnetic interference (EMI)**
      - Serial transmission and fewer signal lines reduce noise pickup and cross-talk.

- **Parallel Communication**
  - **Definition**: Multiple bits transferred simultaneously over multiple data lines
  - **Historical context**: Initially parallel communication was faster than serial communication
  - **Limitations**:
    - **Parallel communication needs more wires and hence bigger circuit**
      - Compared to UART, parallel interfaces increase wiring and PCB size.
    - **Signal skew issues at high frequencies**
      - Multiple data lines in parallel can arrive at different times, causing errors.
    - **More complex routing and higher cost**
      - Extra traces increase PCB complexity, design effort, and overall cost.

- **Note:** While parallel was historically faster, modern high-speed serial protocols like USB 3.0, PCIe, and SATA have largely replaced parallel interfaces due to their superior performance and reliability

Popular Serial Communication Protocols

- **Several serial communication protocols have been developed for different applications:**
  - **General-Purpose Protocols**
    - **RS232**
      - Legacy serial communication standard still used for debugging and industrial devices.
    - **USB (Universal Serial Bus)**
      - High-speed serial interface for computers and peripherals.
    - **PS/2**
      - Older serial interface used for keyboards and mouse devices.
  - **Embedded System Protocols**
    - **I²C (Inter-Integrated Circuit)**
      - Two-wire protocol for connecting multiple low-speed peripherals.
    - **SPI (Serial Peripheral Interface)**
      - High-speed, full-duplex protocol for sensors and displays.
    - **CAN (Controller Area Network)**
      - Robust protocol used in automotive and industrial systems.
    - **JTAG**
      - Used for debugging, testing, and programming embedded devices.

Classification by Distance

- **Short-Distance Communication**
  - **I²C (Inter-Integrated Circuit)**
    - Works up to ~1 meter; ideal for communication between ICs on the same board using minimal wiring.
  - **SPI (Serial Peripheral Interface)**
    - Used for very short distances, typically within the same PCB, offering high-speed data transfer.

- **Long-Distance Communication**
  - **RS232**
    - Suitable for communication over several meters using proper cables, commonly used for PC-to-device communication.

- **Noisy Environment Communication**
  - **CAN (Controller Area Network)**
    - Designed to operate reliably in high electromagnetic interference (EMI) environments such as automobiles and industrial systems.

Classification by Communication Direction

- **Simplex Communication**
  - **One-way communication only**
    - Data moves strictly from sender to receiver with no return path.
  - **Data flows in a single direction without feedback**
    - Receiver cannot acknowledge or respond to the sender.
  - **Example: Simple sensor data transmission**
    - Sensors send readings to a controller without expecting any reply.
- **Half Duplex Communication**
  - **Bidirectional but not simultaneous**
    - Data can travel in both directions, but only one direction at a time.
  - **I2C: Uses shared data line with addressing**
    - Master and slave devices take turns using the same data line.
  - **CAN: Bus-based system with collision detection**
    - Multiple nodes share the bus and communicate one at a time using arbitration.
- **Full Duplex Communication**
  - **Simultaneous bidirectional communication**
    - Data transmission and reception happen at the same time.
  - **RS232: Separate transmit and receive lines**
    - Dedicated TX and RX lines allow continuous two-way data flow.
  - **SPI: Independent MOSI and MISO lines**
    - Master and slave exchange data simultaneously using separate data paths.

Classification by Network Topology

- **Peer-to-Peer**
  - **Peer-to-Peer Topology**: Direct communication between exactly two devices with no sharing
    - **RS232**: Direct one-to-one serial communication between two devices
    - **PS2**: Dedicated point-to-point link between a keyboard/mouse and the host system
- **Bus Topology**
  - **Bus Topology**: Multiple devices share a common communication line
    - **SPI**: One master communicates with multiple slaves using shared data lines and separate chip-selects
    - **I2C**: Multiple masters and slaves share two wires using device addressing
    - **CAN**: Many nodes share a common bus with priority-based arbitration to avoid collisions
- **Tree Topology**
  - **Tree Topology**: Hierarchical structure where devices connect through hubs or branches
    - **USB :** Devices are connected in a hierarchical tree structure using hubs for expansion

UART vs USART

- **UART (Universal Asynchronous Receiver Transmitter)**
  - **Clock Generation:** Each device generates its own clock internally
  - **Synchronization:** Both devices must be configured for the same baud rate
  - **Limitation:** Clock drift can cause communication errors over time
  - **Simplicity:** Easier to implement and configure
- **USART (Universal Synchronous Asynchronous Receiver Transmitter)**
  - **Enhanced Capability:** Supports both synchronous and asynchronous modes
  - **Clock Sharing:** In synchronous mode, one device generates and shares clock with the other
  - **Data Frame:** Single byte communication in a data frame
  - **Performance:** Can work at higher speeds than UART due to shared clock
  - **Advanced Features:** Supports complex protocols like:
    - **RS-485:** Multi-point differential signaling
    - **IrDA:** Infrared communication
    - **LIN:** Local Interconnect Network for automotive
    - **Smart Card:** ISO 7816 compliant communication
    - **ModBus:** Industrial communication protoco

RS-232

- RS-232 remains one of the most important serial communication standards, especially in industrial and legacy applications.
- **Physical Characteristics**
  - **Communication Type**
    - **Full-duplex:** Simultaneous bidirectional communication
    - **Peer-to-peer:** Direct device-to-device connection
  - **Connection Requirements**
    - Minimum Connection (3-wire):
      - **Tx (Transmit):** Data output line
      - **Rx (Receive):** Data input line
      - **GND (Ground):** Common reference voltage
  - **Full Connection (with Handshaking):**
    - **CTS (Clear To Send):** Hardware flow control from receiver
    - **RTS (Request To Send):** Hardware flow control from transmitter
    - These handshaking signals prevent data loss during transmission

- **Connector Standards**
  - **Legacy: DB-25 (25-pin connector)** - largely obsolete
  - **Standard: DB-9 (9-pin connector)** - most common today
    - **Half Serial Cable:** Only 3 essential wires connected (Rx, Tx, Ground)
    - **Full Serial Cable:** All 9 wires connected including handshaking signals
- **Voltage Levels**
  - RS-232 uses inverted logic with specific voltage ranges:
    - **Logic 0 (Space):** +3V to +25V
    - **Logic 1 (Mark):** -3V to -25V
    - **Encoding: NRZ (Non-Return-to-Zero)** - signal doesn't return to zero between bits
  - **Voltage Conversion**:
    - **TTL Logic (0V/5V) ←→ MAX-232 IC ←→ RS-232 Levels (±3V to ±25V)**
    - The **MAX-232 IC** is commonly used to convert between TTL and RS-232 voltage level.
- **Standard Baud Rates**
  - **Common rates include: 9600, 19200, 38400, 57600, 115200 bps .**
  - Higher baud rates require better quality cables and shorter distances

- **Logical Characteristics**
  - **Data Frame Structure**
    - Every RS-232 transmission follows this frame format:
      1. **Start Bit:** Always 0 (space) - signals beginning of data
      2. **Data Bits:** 5 to 9 bits (typically 8 bits) - LSB transmitted first
      3. **Parity Bit (optional):** Error detection mechanism
      4. **Stop Bit(s):** Always 1 (mark) - signals end of data frame
  - **Parity Options**
    - **Even Parity:** Total number of 1-bits (including parity) is even
    - **Odd Parity:** Total number of 1-bits (including parity) is odd .
    - **Sticky 1/0:** Parity bit always set to 1 or 0 regardless of data
    - **No Parity:** Parity bit omitted to increase data throughput

- **Stop Bit Configuration**
  - **1 Stop Bit:** Sufficient for lower baud rates
  - **2 Stop Bits:** Recommended for higher baud rates to ensure reliable frame detection

- **Error Detection and Handling**
  - Common Error Conditions:

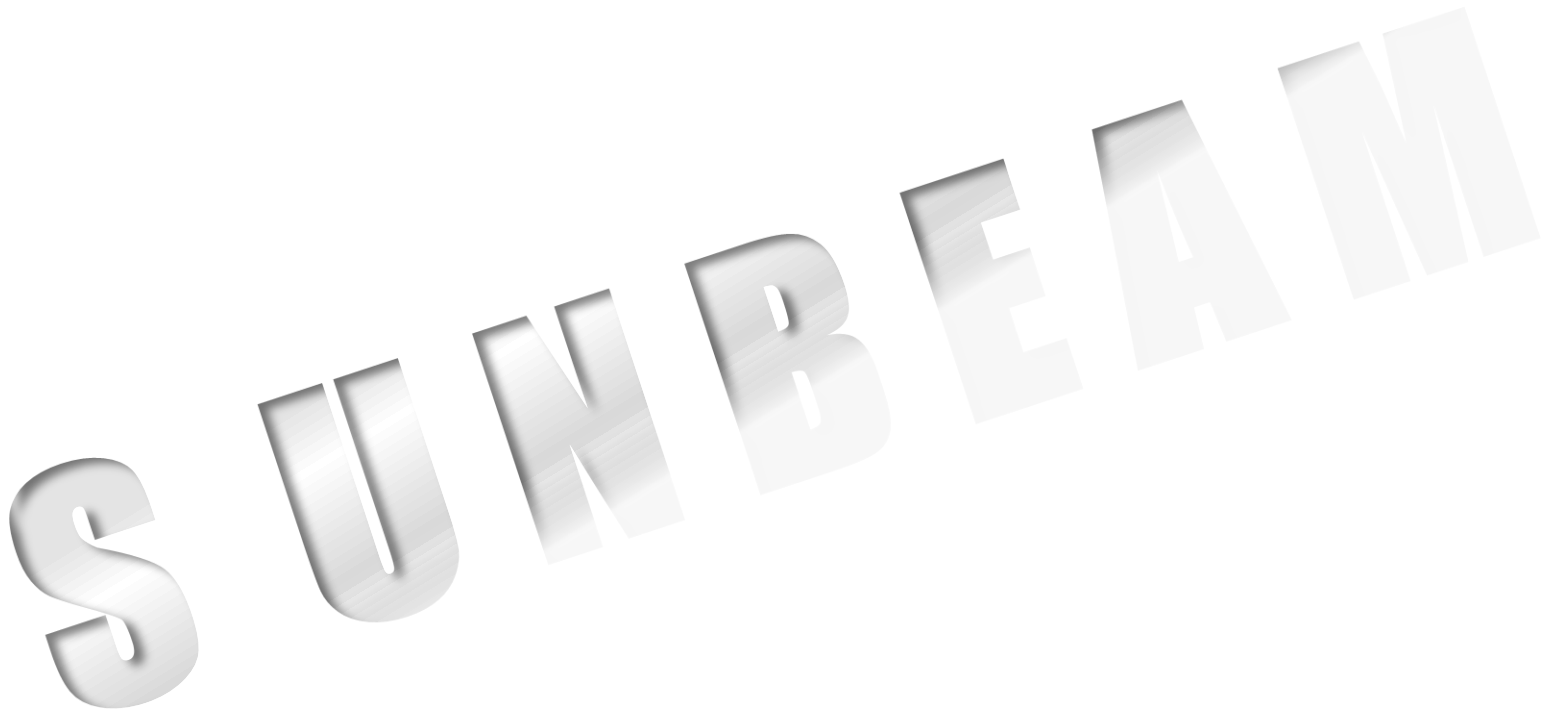
1. **Parity Error:**
   - Received parity doesn't match expected parity based on configuration
   - Indicates possible bit corruption during transmission
2. **Frame Error:**
   - Stop bit received as 0 instead of expected 1
   - Usually indicates timing synchronization issues

3. **Read Overrun**
   - Microcontroller fails to read received data before next byte arrives
   - Previous data gets overwritten, causing data loss
   - Solution: Use hardware/software flow control or increase processing speed
4. **Noise Error:**
   - Detected through oversampling techniques (especially in STM32 microcontrollers) I
   - ndicates electrical noise interference on communication lines

STM32 USART Overview

- **STM32F407xx** microcontroller provides **4 USARTs and 2 UARTs** for serial communication.
- **USART** supports both **synchronous and asynchronous** communication, while **UART** supports only asynchronous.
- STM32 USART Features
  - **Full-duplex communication**
    - Data can be transmitted and received simultaneously.
  - **Synchronous & asynchronous modes**
    - Supports clocked (USART) and clock-less (UART-like) communication.
  - **NRZ standard format**
    - Uses Non-Return-to-Zero encoding for serial data bits.
  - **Flexible configuration**
    - Baud rate, word length, parity, and stop bits can be configured as required.
    - **Accurate baud rate generation**
      - Ensures reliable communication with minimal timing error.
    - **Fractional baud rate setting**
      - Allows fine-grained baud rate adjustment for better accuracy.
    - **Programmable word length, stop bits, parity**
      - Supports different data formats like 8N1, 9E1, etc.
    - **Single buffer or DMA multi-buffer communication**
      - Data can be transferred using polling, interrupts, or DMA.

- **Transfer detection flags**
  - Hardware flags indicate transmit complete, receive ready, etc.
- **Error detection flags**
  - Detects parity error, framing error, noise error, and overrun error.
- **Transfer and error interrupts**
  - Interrupts allow CPU to respond immediately to data or errors.
- **Programmable hardware flow control**
  - Supports RTS/CTS for controlling data flow automatically.


SUNBEAM

- **STM32 Oversampling**
  - **Purpose:** Improves data reliability by detecting noise during UART reception.
  - **Working Principle:** The RX signal is sampled multiple times per bit to confirm valid data.
  - **Control:** Oversampling rate is selected using the **OVER8 bit** in the USART control register.
  - **Benefit:** Helps distinguish real data from glitches and noise on the communication line.
- **Oversampling Modes (STM32)**
  - **Oversampling by 16:**
    - Samples each bit 16 times for better noise immunity (default mode).
  - **Oversampling by 8:**
    - Samples each bit 8 times, allowing higher baud rates with slightly reduced noise tolerance.

STM32 UART baud rate calculation

- UART clock is APB1 or APB2 clock.
- For default setting on STM32F407VG
  - APB1 = APB2 = Fcclk = 16 MHz
- $baud = \frac{PCLK}{8 * (2-OVER8) * USARTDIV}$
  - If OVER8 = 1, $baud = \frac{PCLK}{8 * USARTDIV}$
  - If OVER8 = 0, $baud = \frac{PCLK}{16 * USARTDIV}$
- $USARTDIV = \frac{PCLK}{8 * (2-OVER8) * baud}$
- USARTDIV is set into USART_BRR in two components i.e. Mantissa and Fractional.

- Example:
  - Desired baud rate = 9600
  - PCLK = 16 MHz
  - OVER8 = 0
  - USART_BRR = ?
- $USARTDIV = \frac{16000000}{16 * 9600} = 104.166667$
  - Mantissa = INT(USARTDIV) = 104 = 0x68
  - Fractional = ROUND(0.166667 * 16) = 0x3
- BRR (16 bits)
  - Mantissa (12-bits) + Fractional (4-bits)
  - 0x68 (12-bits) + 0x3 (4-bits) = 0x683

- **USART Registers**
  - **USART_DR (Data Register)**
    - Used for **data transfer** between CPU and USART.
    - Internally divided into:
      - **TDR (Transmit Data Register)** → holds data to be sent.
      - **RDR (Receive Data Register)** → holds received data.
    - Writing to DR sends data, reading from DR receives data.
  - **USART_BRR (Baud Rate Register)**
    - Used to **set the baud rate** for USART communication.
    - Contains two parts:
      - **Mantissa** (integer part)
      - **Fractional** (fraction part)
    - Value is calculated from **PCLK and desired baud rate**.

- **USART_CR1, CR2, CR3 (Control Registers)**
  - Used to **configure USART operation**.
  - Control settings such as:
    - Enable/disable USART
    - Transmitter and Receiver enable
    - Word length, parity
    - Interrupt enable
    - Hardware flow control (RTS/CTS)
- **USART_SR (Status Register)**
  - Shows **USART status and error information**.
  - Used to:
    - Check if data is transmitted or received
    - Detect errors like:
      - Parity error
      - Overrun error
      - Frame error
      - Noise error
- Also provides **interrupt status flags**.

USART Tx and Rx with Polling

- **USART Initialization**
  - **Configure GPIO**
  - Enable GPIO clock so pins can be configured.
  - Set TX and RX pins to **Alternate Function (AF) mode**.
  - Select the correct AF number for USART (e.g., AF7 for USART2).
  - Configure output type and pull-up/pull-down as required.
- **Configure USART**
  - Enable USART peripheral clock.
  - Configure communication parameters using control registers (CR1, CR2, CR3):
    - Data length
    - Stop bits
    - Parity
    - Enable transmitter and receiver
  - Set baud rate using the **BRR (Baud Rate Register)**.

- **Enable USART**
  - Set the **UE (USART Enable)** bit to activate the USART peripheral.
- **USART Transmit (Polling Method)**
  - Continuously check the **TXE (Transmit Data Register Empty)** flag in the **Status Register (SR)**.
  - TXE = 1 indicates the previous data has been transferred to the shift register.
  - Write the next character into the **Data Register (DR / TDR)**.
  - Hardware automatically transmits the data serially.
- **USART Receive (Polling Method)**
  - Continuously check the **RXNE (Receive Data Register Not Empty)** flag in the **Status Register (SR)**.
  - RXNE = 1 indicates a new character has been received.
  - Read the received data from the **Data Register (DR / RDR)**.
  - Reading DR automatically clears the RXNE flag
- **Example: USART2 Pin Mapping (STM32F407)**
  - **TX → PA3 (AF7)**
  - **RX → PA2 (AF7)**
  - Alternate function selection is done using **GPIOA_AFRL**.

Thank you!

Kiran Jaybhave

email – kiran.jaybhave@sunbeaminfo.com