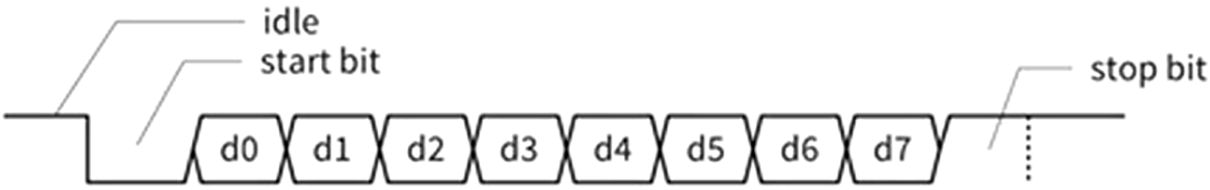
UART (Universal Asynchronous Receiver-Transmitter)

Introduction

UART is a hardware communication protocol used for serial communication between devices. Unlike synchronous communication protocols that use a shared clock signal, UART operates asynchronously, allowing devices to communicate without sharing a clock signal. Let me provide a comprehensive explanation of how UART works.

Core UART Concepts



Bit Timing and Definition

In UART, a "bit" refers to the time duration for which the communication line is held at a specific voltage level (typically high or low). This time duration is determined by the baud rate and must be agreed upon by both the transmitter (Tx) and receiver (Rx).

Idle State and Communication Initiation

- **Idle State**: When no communication is occurring, the line is kept at logic high (1).
- **Start Bit**: To begin communication, the transmitter pulls the line low (0) for one bit duration. This serves as a clear signal to the receiver that data transmission is about to begin.

Data Transmission

- After the start bit, the actual data bits are transmitted sequentially.
- Typically, 8 data bits (one byte) are sent, though UART can be configured for 6, 7, or 8 bits.
- Each bit is maintained for the same duration determined by the baud rate.
- Data is usually sent least significant bit (LSB) first.

Parity Bit (Optional)

- A parity bit may be included after the data bits as a simple error detection mechanism.
- **Even Parity**: The parity bit is set so that the total number of 1s (including the parity bit) is even.
- **Odd Parity**: The parity bit is set so that the total number of 1s is odd.
- **No Parity**: Some UART configurations omit the parity bit entirely.

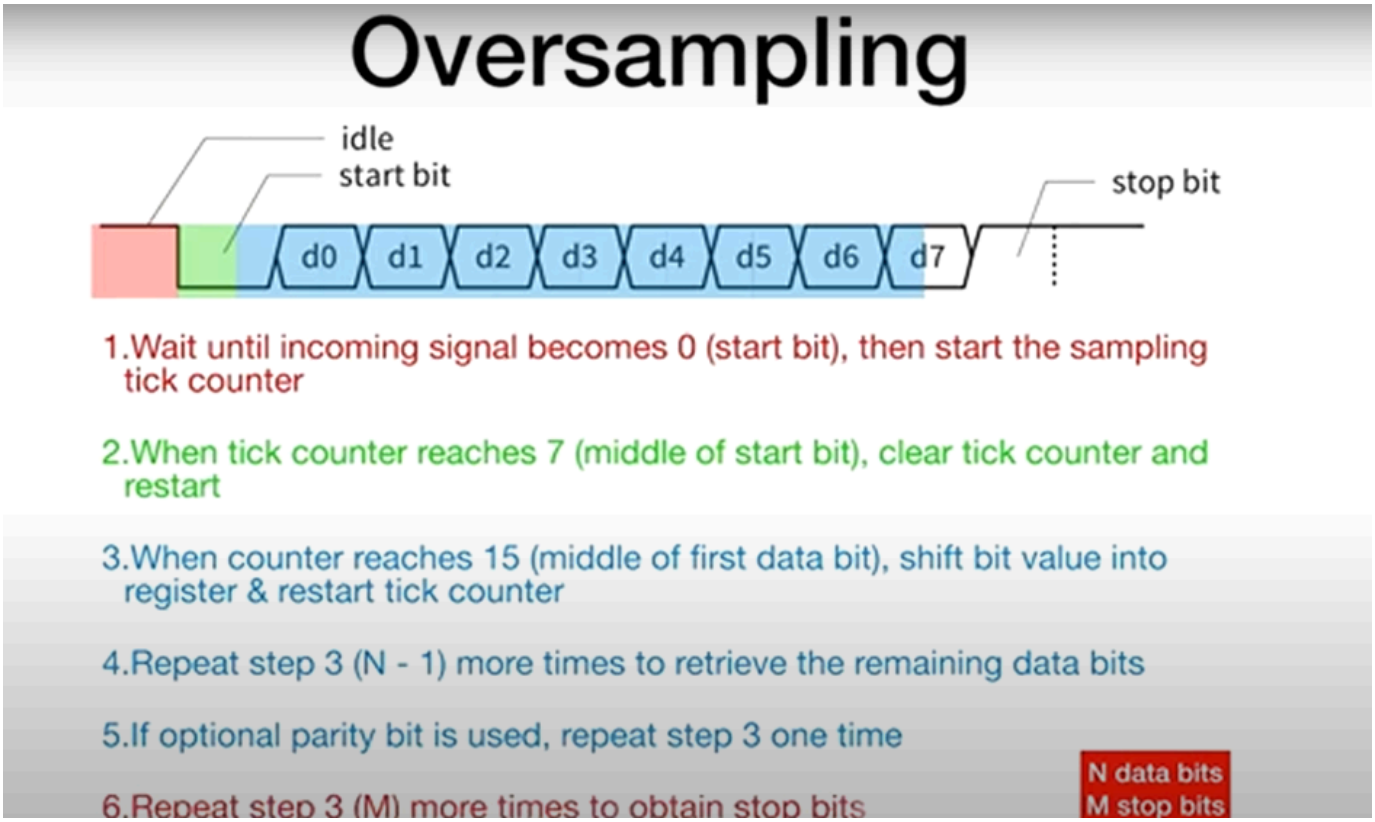
Stop Bit(s)

- After all data bits (and the optional parity bit) are sent, one or more stop bits are transmitted.
- Stop bits are always logic high (1).
- UART can be configured for 1, 1.5, or 2 stop bits.
- These bits signal the end of the current data frame and return the line to the idle state.

Critical Parameters Both Sides Must Agree On

1. **Baud Rate**: The speed of communication measured in bits per second (bps). Common baud rates include 9600, 19200, 38400, 57600, and 115200 bps.
2. **Data Bits**: The number of bits used to represent actual data (6, 7, or 8 bits).
3. **Parity**: Whether error detection is used, and if so, whether it's even or odd parity.
4. **Stop Bits**: The number of bits indicating the end of a frame (1, 1.5, or 2 bits).
5. **Flow Control**: Optional mechanism to regulate data flow (hardware flow control with RTS/CTS or software flow control with XON/XOFF).

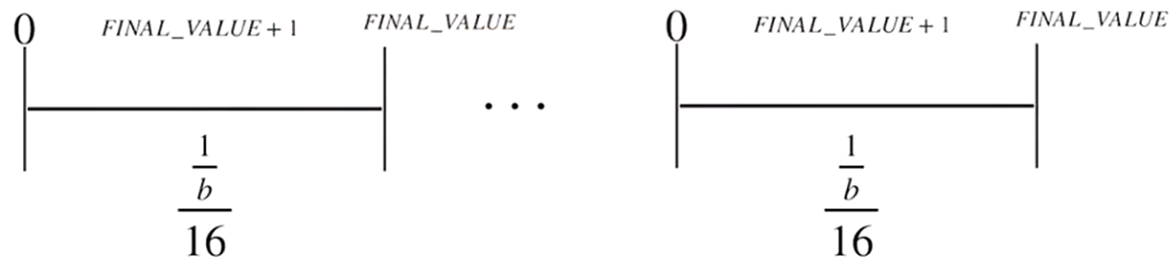
Oversampling in UART



Oversampling is a crucial technique that enables reliable asynchronous communication:

- The receiver samples the incoming signal at a much higher rate than the baud rate, typically 16 times per bit duration.
- This oversampling allows the receiver to:
  - Accurately detect the falling edge of the start bit
  - Sample each bit near its center for the most reliable reading
  - Maintain synchronization throughout the entire frame without a shared clock

Baud Rate Generator



$$(FINAL\ VALUE + 1) \times T = \frac{1}{16 \times b}$$

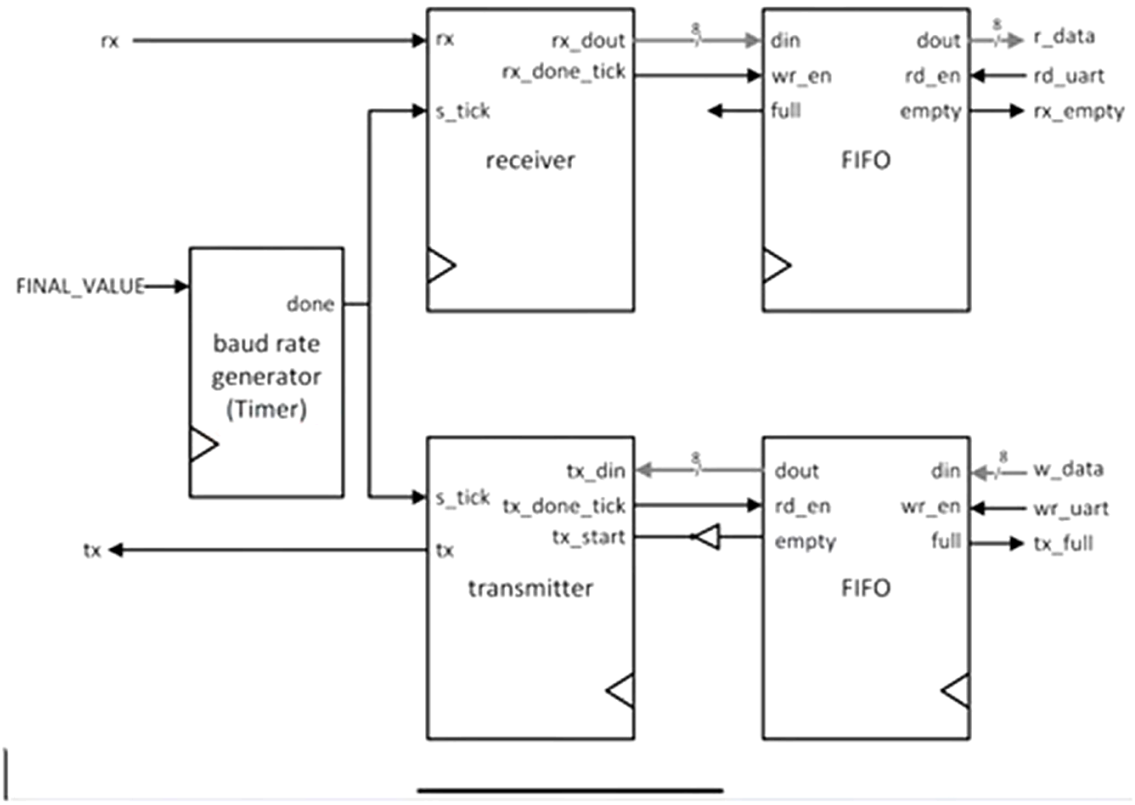
Find the final value for
f = 100 MHz, b = 9600 bits/second

$$FINAL\ VALUE = \frac{1}{16 \times b \times T} - 1$$

For example, with 16x oversampling at 9600 baud:

- The receiver samples at 153,600 Hz (16 × 9600)
- It can detect the middle of each bit by counting samples after the start bit

UART Hardware Construction



A UART module typically consists of:

1. **Transmitter Section**:
   - Parallel-to-serial converter (takes parallel data from CPU and converts it to serial)
   - Shift register to clock out bits sequentially
   - Baud rate generator to control timing
   - Control logic to add start, parity, and stop bits
2. **Receiver Section**:
   - Serial-to-parallel converter
   - Shift register to accumulate incoming bits
   - Oversampling clock (typically 16x baud rate)
   - Start bit detector
   - Data sampling logic
   - Error detection (framing, parity, overrun)
   - Buffer registers to hold received data
3. **Buffer Registers**:
   - Transmit and receive data registers
   - Status registers to indicate errors or completion
   - Control registers for configuration

UART Communication Example

Here's an example of transmitting the ASCII character 'A' (binary 01000001) with 8 data bits, no parity, and 1 stop bit:

1. **Idle State**: Line is high (1)
2. **Start Bit**: Line goes low (0) for one bit duration
3. **Data Bits**: The 8 bits of 'A' are sent LSB first: 10000010 (reversed because LSB first)
4. **Stop Bit**: Line goes high (1) for one bit duration
5. **Return to Idle**: Line remains high (1) until the next transmission

Advantages and Limitations

Advantages:

- Simple hardware implementation
- Only requires two wires (Tx and Rx)
- No shared clock required
- Widely supported in microcontrollers and peripherals

Limitations:

- Lower maximum speed compared to synchronous protocols
- Limited error detection capabilities
- Inefficient (requires start/stop bits for every byte)
- Susceptible to timing errors when using very long cables or high baud rates

Common Applications

UART is used in numerous applications:

- Serial communication between microcontrollers
- Debug console interfaces
- Communication with serial peripherals
- Legacy RS-232, RS-422, and RS-485 interfaces
- Bluetooth serial port profiles
- GPS modules
- Many IoT devices

UART remains one of the most fundamental and widely used communication protocols in embedded systems due to its simplicity and reliability for moderate-speed applications.

Screenshots credited to: https://youtu.be/_JW9wVEBE3c?si=yk0kM_y7FsiORlm0