UART PROJECT

Made by:
Mohamed Khaled Abd el-sattar

NTI
Digital Design using FPGA

SEPTEMBER 10, 2025

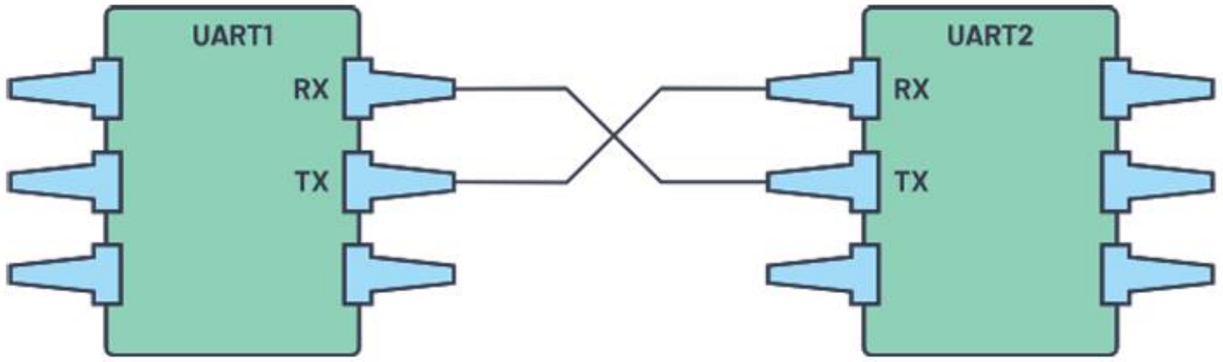
Introduction:

What is UART:

UART stands for Universal Asynchronous Receiver/Transmitter.

It is a simple serial communication method where data is sent one bit at a time over a single wire in one direction, plus another wire for the reverse direction.

It's point-to-point, meaning one transmitter talks directly to one receiver. Many microcontrollers, FPGAs, and PCs have built-in UART hardware blocks because it's so widely used.



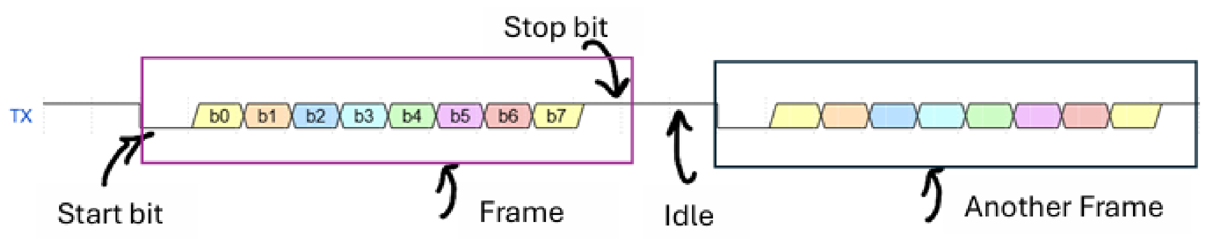
UART uses 2 physical connections, one to transmit data & another to receive data. Communication happens in frames. Each frame has a very specific structure, so the receiver knows where it starts, where it ends, and what data it contains. The most common frame format is 8-N-1, which means:

8 data bits (sent least significant bit first)

N → No parity bit

1 stop bit

Before the data bits, there's a start bit (logic 0), and after them there's one or more stop bits (logic 1). The line stays idle (logic 1) between frames.



Before sending or receiving data, receiver and transmitter are configured to use the same baud rate (bits

per second), e.g. 9600, 115200, etc.

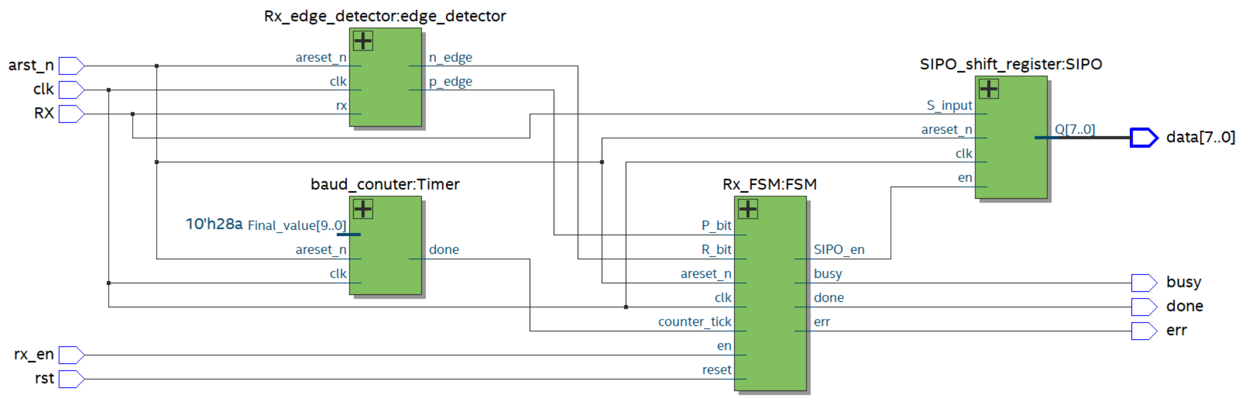
At 9600 baud → 1 bit period = 1/9600 ≈ 104.167 µs.

UART architecture

The receiver:

Only receives one byte at a time

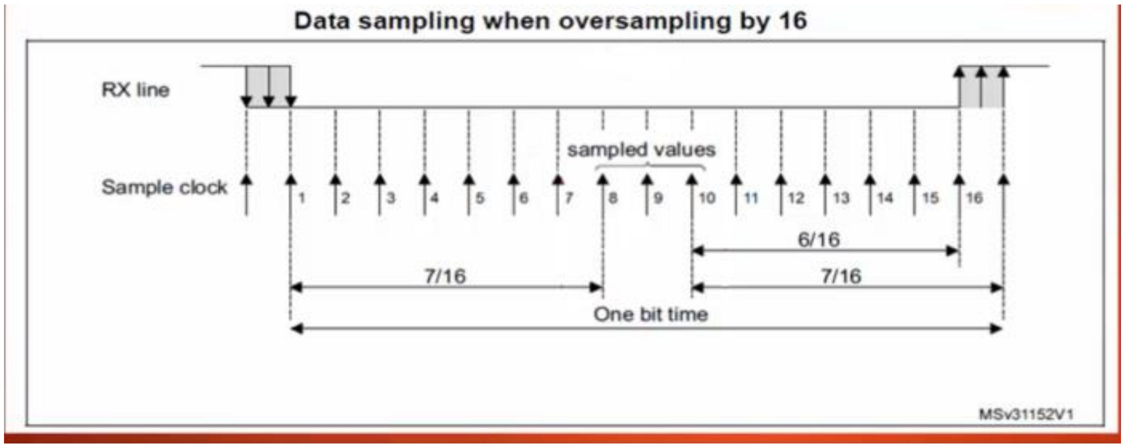
- Waits for a falling edge (start bit).
- Samples the data bits in the middle of each bit period.
- Validates stop bit(s)
- Asserts busy when a byte is being received
- Asserts done when a byte is received



How to sample in the middle of the bit?

The technique I used is oversampling:

Oversampling in UART is a receiving technique where the receiver samples the incoming data at a much higher rate than the baud rate (often 16 times the baud rate), allowing it to accurately locate the middle of each data bit to counteract variations in the sender's and receiver's clock frequencies.



1. Rx_edge_detector

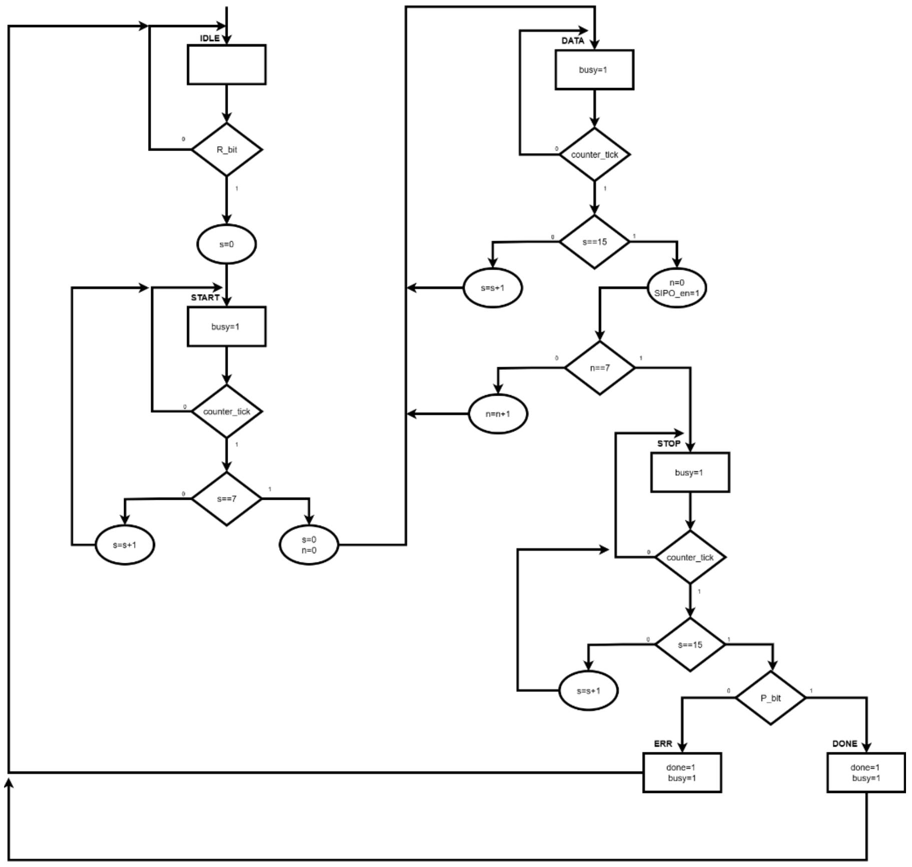
- Function: Detects edges on the serial input line RX.
- Outputs:
    - n_edge: Asserted on a falling edge (1 → 0), which usually indicates the start bit.
    - p_edge: Asserted when rx is high, which can be used to check the stop bit.
- Role: Provides synchronization by identifying when a new UART frame begins.

2. baud_counter (Timer)

- Generates periodic ticks used to sample incoming bits at the correct baud rate.
- Operation:
  - Counts clock cycles up to Final_value (which corresponds to one bit duration).
  - When the counter reaches this value, it asserts done and resets.
- Ensures correct timing for sampling each received bit in the middle of its time slot.

3. Rx_FSM (Finite State Machine)

- Controls the entire UART receive process.
- States:
  - IDLE: Waits for a start bit.
  - START: Confirms the presence of the start bit.
  - DATA: Samples and shifts in 8 data bits.
  - DONE: Indicates that a full frame has been received successfully.
  - ERR: Signals a framing error if the stop bit is invalid.
- Outputs:
  - SIPO_en: Enables shifting into the SIPO register.
  - done: Asserted when a full byte has been received.
  - busy: Indicates that the receiver is in progress.
  - err: Raised when a stop-bit error occurs.
- Coordinates sampling, error detection, and data readiness.
- State diagram:

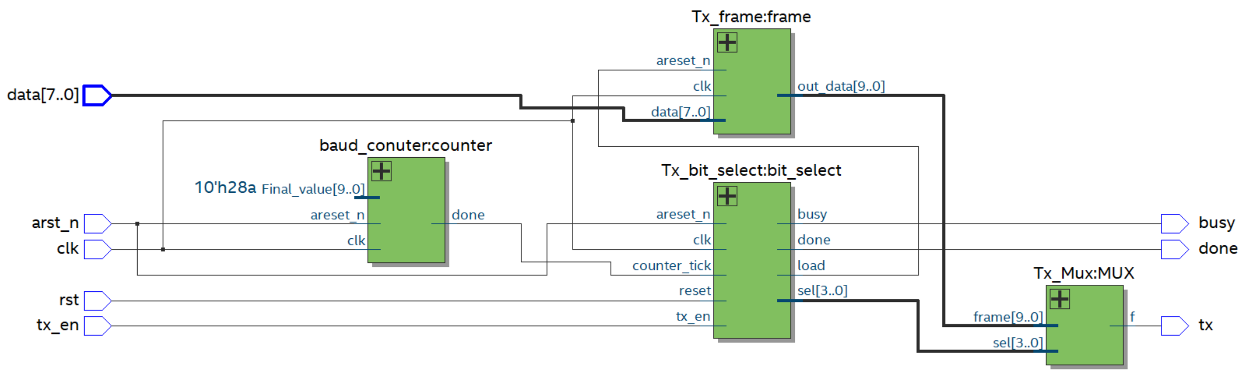


4. SIPO_shift_register (Serial-In Parallel-Out)

- Converts the serial data stream from the RX line into an 8-bit parallel word.
- Operation:
  - On each enable (SIPO_en), the incoming bit is shifted into the register.
  - After 8 shifts, the complete byte is available on output Q[7:0].
- Buffers the received serial bits into a parallel format for further processing.

The Transmitter

- Sends only one byte at a time
- Appends start and stop bits
- Handles sending data with right bit times based on baud rate
- Asserts done after finishing sending the loaded byte
- Asserts busy while sending the loaded byte



1. Baud Counter

- Generates periodic ticks used to sample incoming bits at the correct baud rate.
- Operation:
  - Counts clock cycles up to Final_value (which corresponds to one bit duration).
  - When the counter reaches this value, it asserts done and resets.
- Ensures correct timing for sampling each received bit in the middle of its time slot.

2. Tx Frame

This block prepares the UART frame. It takes the 8-bit input data and appends the start bit (0) at the beginning and the stop bit (1) at the end, producing a 10-bit frame ready for transmission.

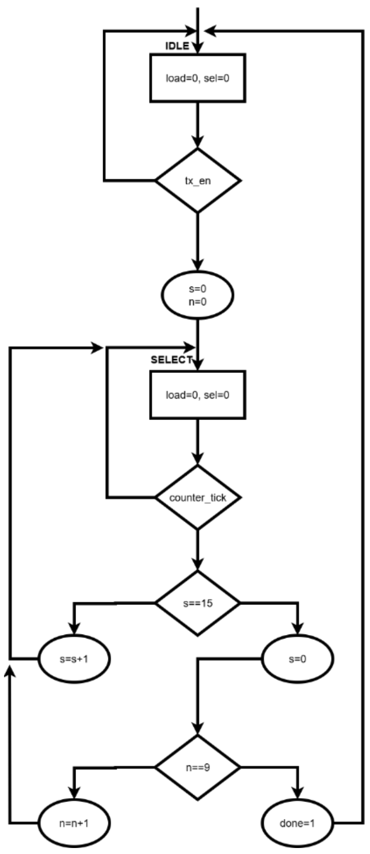
It is always high until load from bit select gets active.

3. Tx Bit Select

This block controls the transmission process. It uses the baud counter pulses to select which bit of the frame should be sent. It also generates status signals:

- busy: transmission is in progress,
- done: transmission has finished,
- load: indicates when a new frame can be loaded.

State diagram:



4. Tx Mux

This block outputs the actual serial data. It selects one bit of the frame (based on the bit index from Tx Bit Select) and drives it to the UART transmit line (tx).

Test Bench and Verification

The receiver test bench is used to verify the functionality of the UART_RX module.

The test bench uses a

localparam for clock frequency (100 MHz) and baud rate (9600) to calculate the BIT_PERIOD in terms of clock cycles. This ensures the timing is accurate for simulation.

The core of the test bench is the send_byte task. This task simulates the serial transmission of a single byte:

1. It drives the

RX line low to simulate the start bit for one bit period.

2. It iterates through the 8 data bits of the input byte, driving the

RX line to the correct value for each bit, and waiting one bit period after each bit. It's crucial to send the data bits

least significant bit (LSB) first.

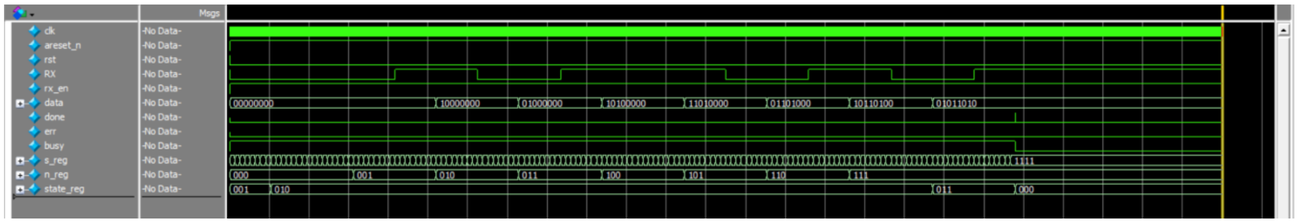
3. Finally, it drives the

RX line high for the stop bit and waits one bit period.

The initial block sets up the simulation environment. It initializes the signals, applies a reset, and then calls the

send_byte task to transmit the hex value 0x5A (8'b01011010). After the transmission, the test bench waits a moment to observe the

done signal from the receiver and then $stops the simulation.



Transmitter testbench:

This test bench verifies the transmitter module by providing data and enabling the transmission. It includes a continuous clock generation block. The

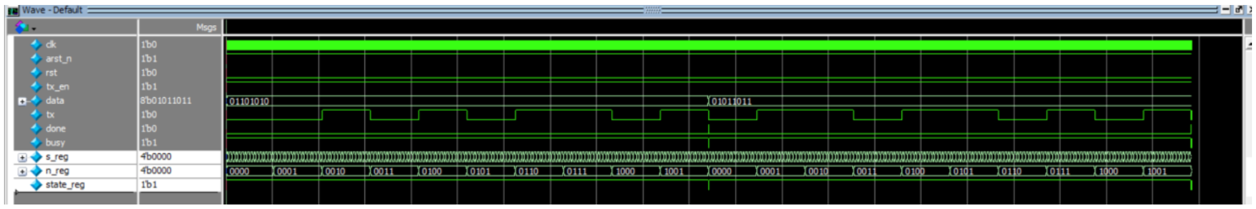
initial block initializes the signals and applies a brief asynchronous reset before the main test sequence begins.

The test bench then enables the transmitter (

tx_en = 1'b1) and sets the data to 8'b01101010. It uses

wait(done == 1) to pause the simulation until the first transmission is complete. After a brief delay, it loads a second byte (

8'b01011011) and waits for the completion of the second transmission before stopping the simulation.


Wave - Default
Msgs
clk 1'b0
arst_n 1'b1
rst 1'b0
tx_en 1'b1
data 8'b01011011
tx 1'b0
done 1'b0
busy 1'b1
s_reg 4'b0000
n_reg 4'b0000
state_reg 1'b1
01101010
01011011
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001