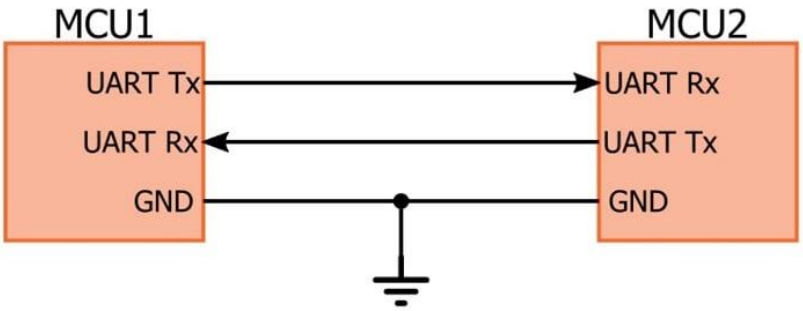


UART

ABSTRACT

UART (Universal Asynchronous Receiver/Transmitter) is a simple serial communication protocol used to transmit and receive data between two devices. It works without a shared clock by framing data with start, data, optional parity, and stop bits at an agreed baud rate.

Seif Hassan

Github repo:

Universal Asynchronous Receiver/Transmitter (UART)

1. Introduction

The Universal Asynchronous Receiver/Transmitter (UART) is one of the oldest and most widely used methods for serial communication. It provides a simple, reliable way for two digital devices to exchange data over just two main wires: a **transmit (TX)** line and a **receive (RX)** line. Unlike synchronous communication protocols (e.g., SPI or I^2C), UART does not require a shared clock signal. Instead, the transmitter and receiver agree on a **baud rate** (bits per second), which determines the timing of data transmission.

UART has been present in computer systems since the early days of RS-232 communication in the 1960s. Today, it is still embedded in microcontrollers, FPGAs, SoCs, and various embedded platforms for debugging, configuration, and data exchange.

2. Basic Principle of UART

UART works by converting parallel data (inside a CPU or microcontroller) into a serial bitstream for transmission, and then back into parallel form at the receiver.

- **Transmitter (TX):** Converts data bytes into serial frames and shifts them out bit by bit.
- **Receiver (RX):** Monitors the line for a start bit, samples incoming bits at the right time, and reconstructs the original data.

Unlike network protocols, UART is **point-to-point**, meaning it connects exactly two devices.

3. UART Data Frame Structure

Each transmission in UART is wrapped in a **frame**. A frame consists of:

1. **Start Bit** – A single low-level bit (0) that signals the beginning of a new character.
2. **Data Bits** – Typically 5 to 9 bits, most commonly 8 bits.
3. **Parity Bit (optional)** – Provides simple error detection (even or odd parity).
4. **Stop Bit(s)** – One or two high-level bits (1) that mark the end of the frame.

Example for 8N1 format (8 data bits, no parity, 1 stop bit):

| Start | D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 | Stop |

4. Baud Rate and Clock Synchronization

The baud rate defines how many symbols (bits) are transmitted per second. Common values include **9600, 19200, 115200 bps**.

Since UART has no clock line, both transmitter and receiver must be **configured with the same baud rate**. The receiver usually oversamples the input (e.g., 16x oversampling) to accurately detect edges.

Example: At 9600 baud, each bit lasts about 104 µs.

5. Error Handling in UART

UART includes several error detection mechanisms:

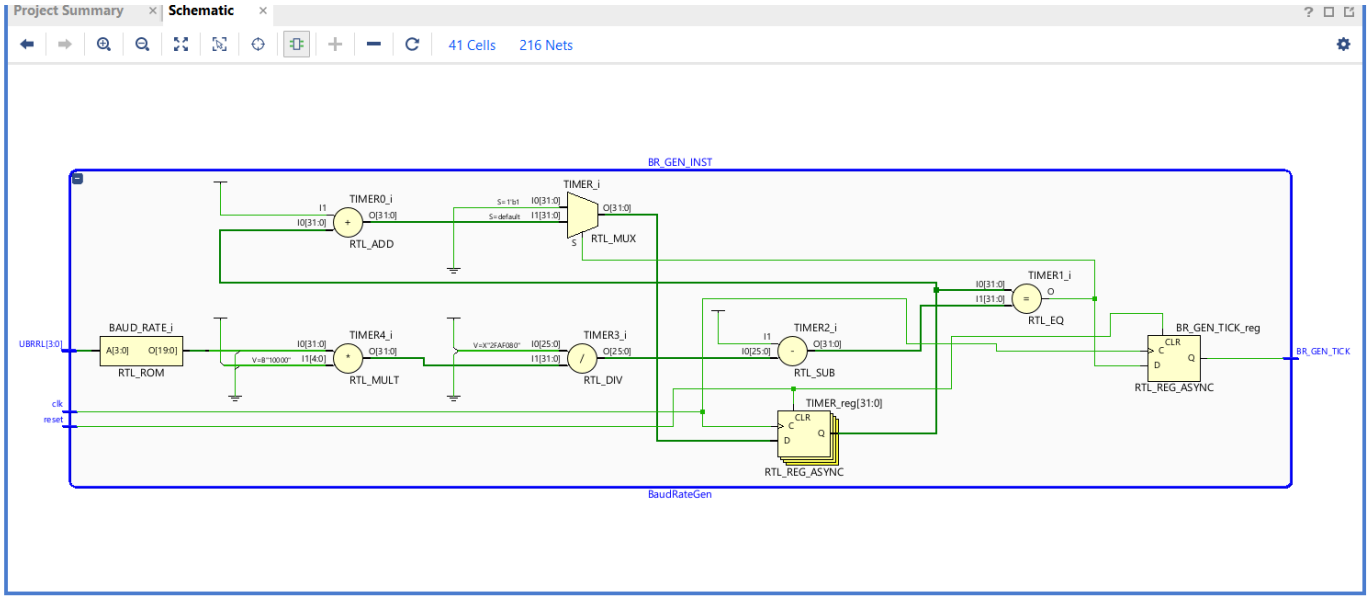
- **Parity Error:** Detected when parity bit does not match expected even/odd count.
- **Framing Error:** Occurs when stop bit is not detected at the expected position.
- **Overrun Error:** Happens when the receiver buffer is full, and a new byte arrives before the old one is read.

These errors help ensure data integrity but cannot correct corrupted data.

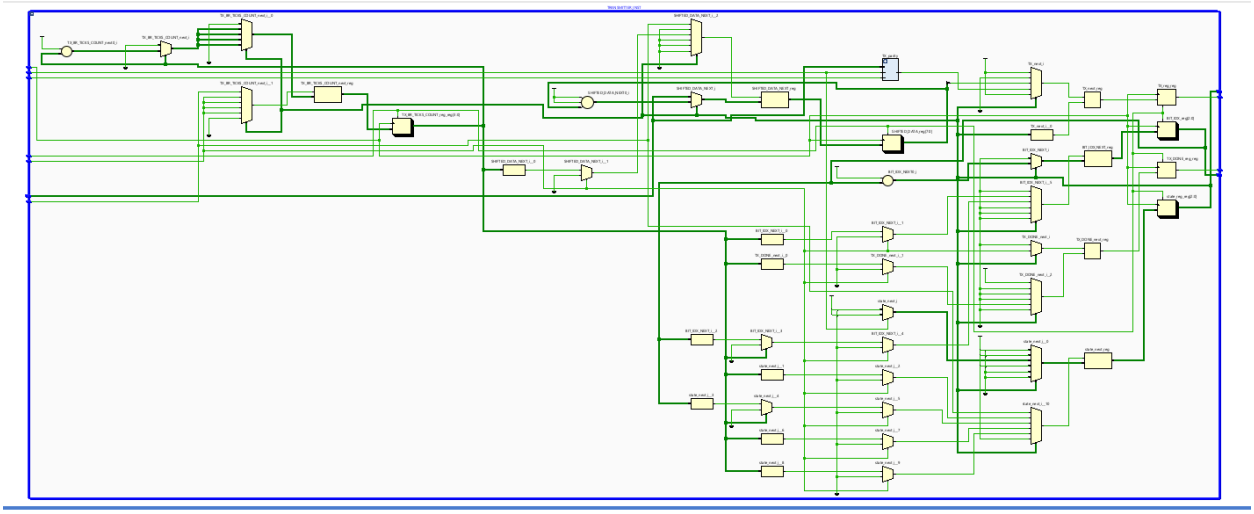
6. Hardware Architecture of UART

A typical UART module includes the following components:

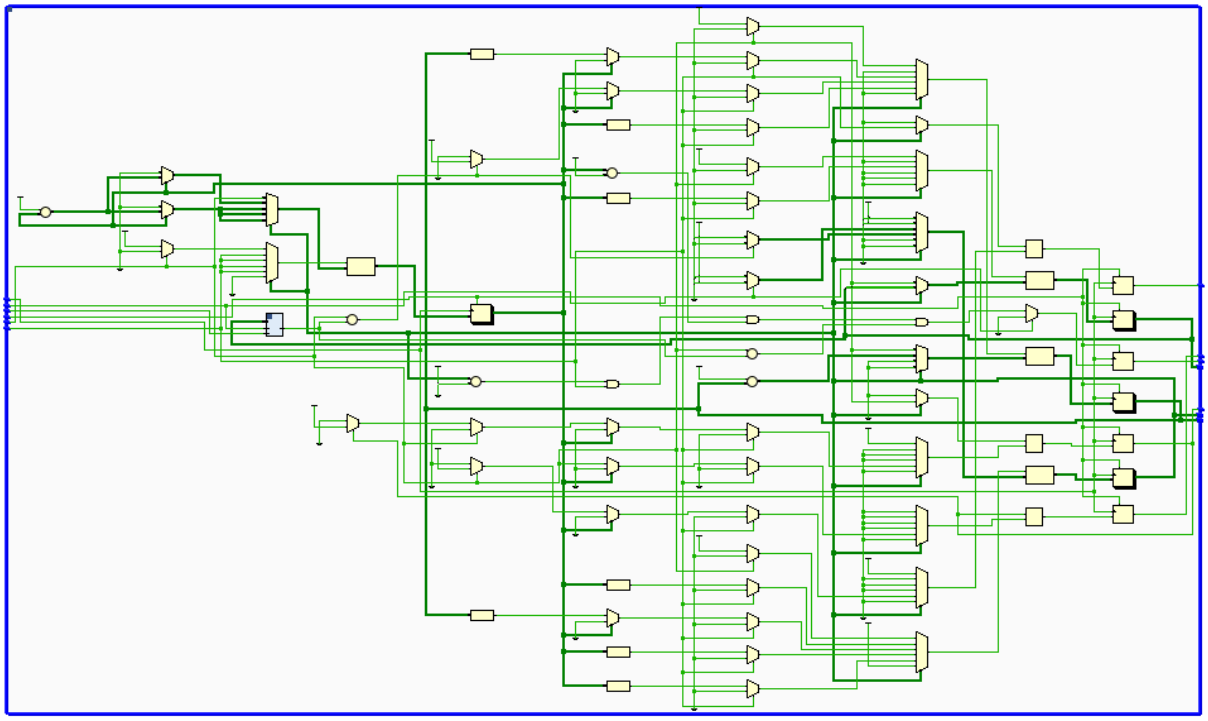
1. **Baud Rate Generator (BRG):** Divides system clock to match desired baud rate.



2. **Transmitter Section:**
    - serial shift register.
    - State machine for start, data, parity, stop bits.



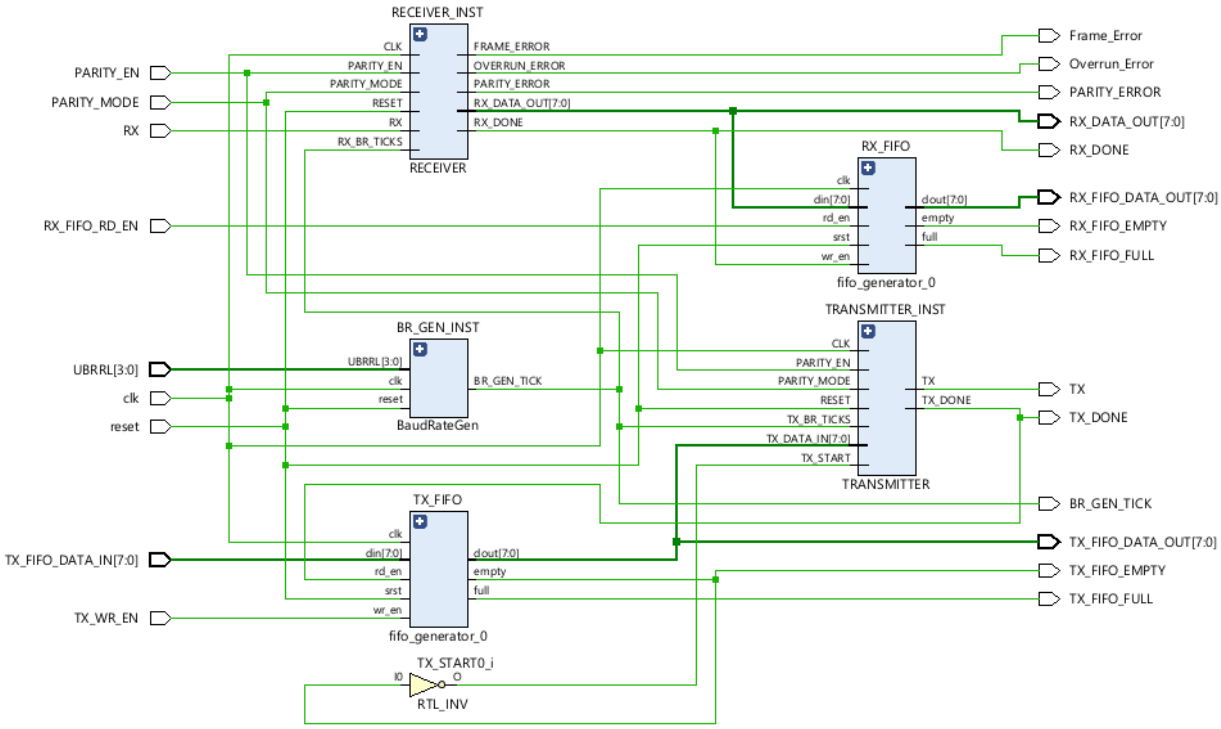
3. **Receiver Section:**
    - Serial-to-parallel shift register.
    - Oversampling and edge detection logic.
    - Error detection logic.



4. **FIFOs:**
    - TX FIFO stores data bytes before transmission.
    - RX FIFO buffers received bytes until CPU reads them.

- Used Vivado’s FIFO GEN from the ip catalog

5. FULL UART DESIGN:



7. UART in Microcontrollers and FPGAs

- **Microcontrollers (MCUs):** Almost all MCUs (e.g., STM32, PIC, AVR, ARM Cortex-M) have UART/USART peripherals. They are often used for debugging, data logging, or wireless modules (Bluetooth, GSM).
- **FPGAs:** In FPGA-based designs, UART can be implemented as an IP core or custom RTL module. Designers add FIFOs, parity logic, and baud rate generators. The FPGA UART can interface with PCs through USB-to-UART bridges like FT232 or CP2102.

8. Advantages and Limitations

Advantages:

- Simple and reliable.
- Only requires two data wires.
- Supported by almost all MCUs and PCs (via USB adapters).

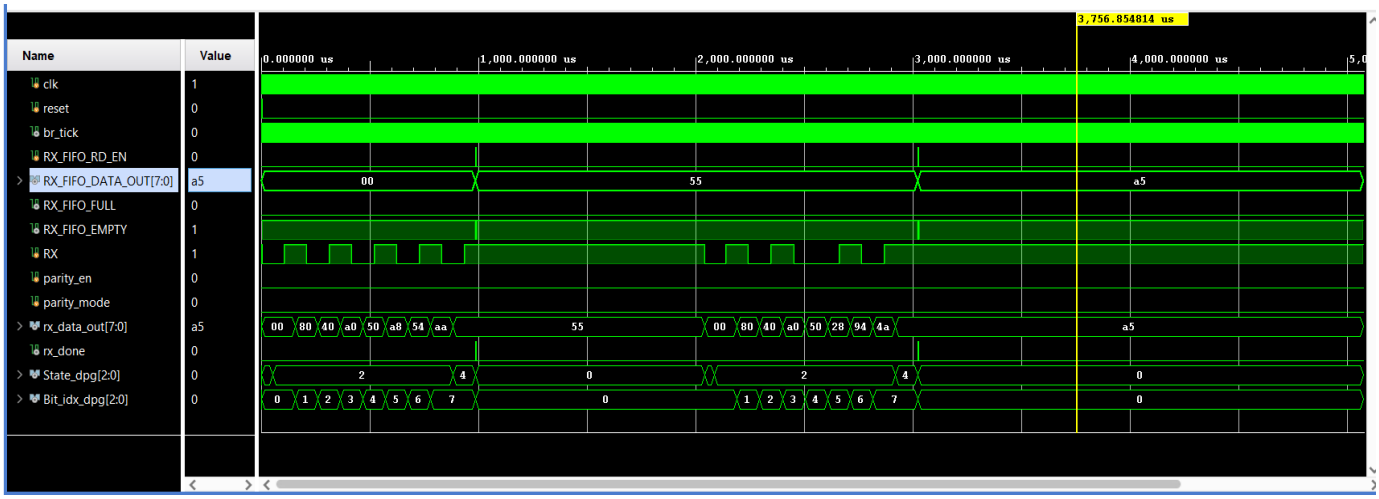
Limitations:

- Point-to-point only (no multi-device bus like I^2C).
- Limited data rate compared to modern interfaces.
- No built-in addressing (always one-to-one communication).

Test Bench

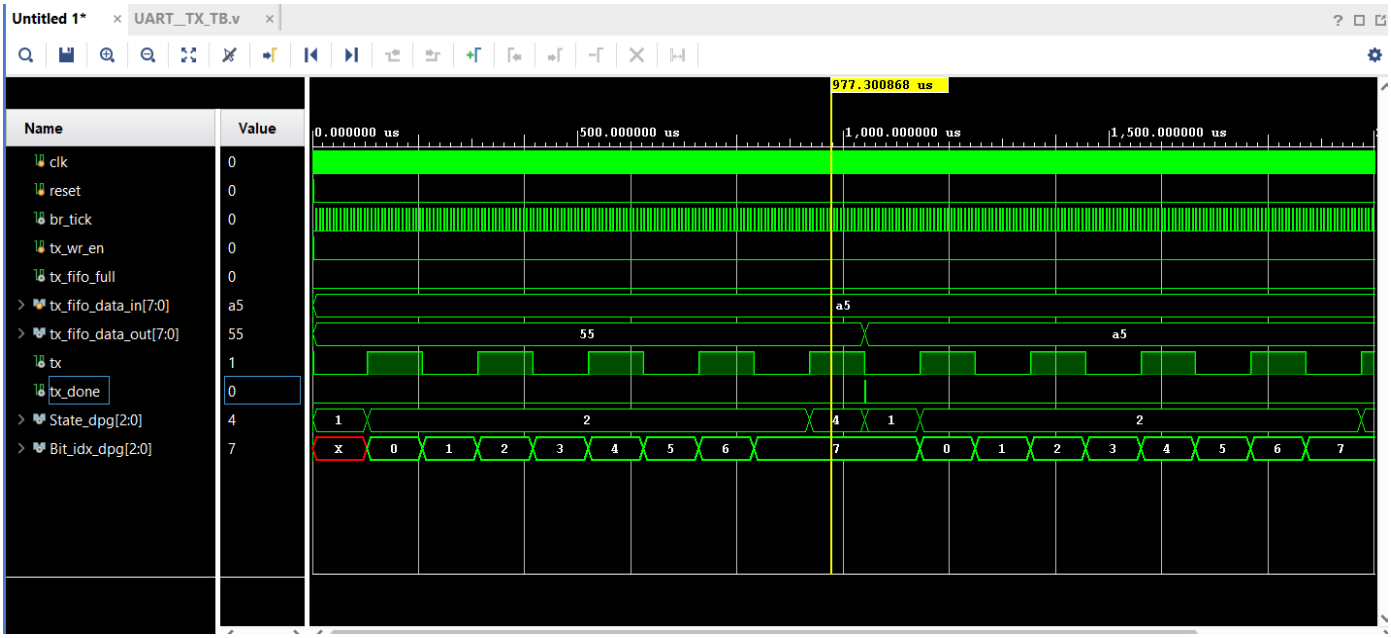
1. Receiver (RX) Testbench

- **Stimulus:**
  - Applied serial bitstreams emulating UART input (RX line).
  - Tested different configurations:
    - With and without parity checking.
    - Valid and invalid stop bits (to trigger frame errors).
    - Back-to-back frames to test overrun conditions.
- **Monitored Outputs:**
  - RX_DATA_OUT checked against expected parallel data.
  - RX_DONE observed for proper frame completion.
  - Error signals (PARITY_ERROR, FRAME_ERROR, OVERRUN_ERROR) validated against intentional faults.
- **Result:**
  - Correct parallel data was received.
  - Error flags activated in the correct scenarios, proving the FSM's robustness.
  - Debug signals (State_dpg, bit_idx_dpg) provided visibility into FSM transitions during waveform analysis.

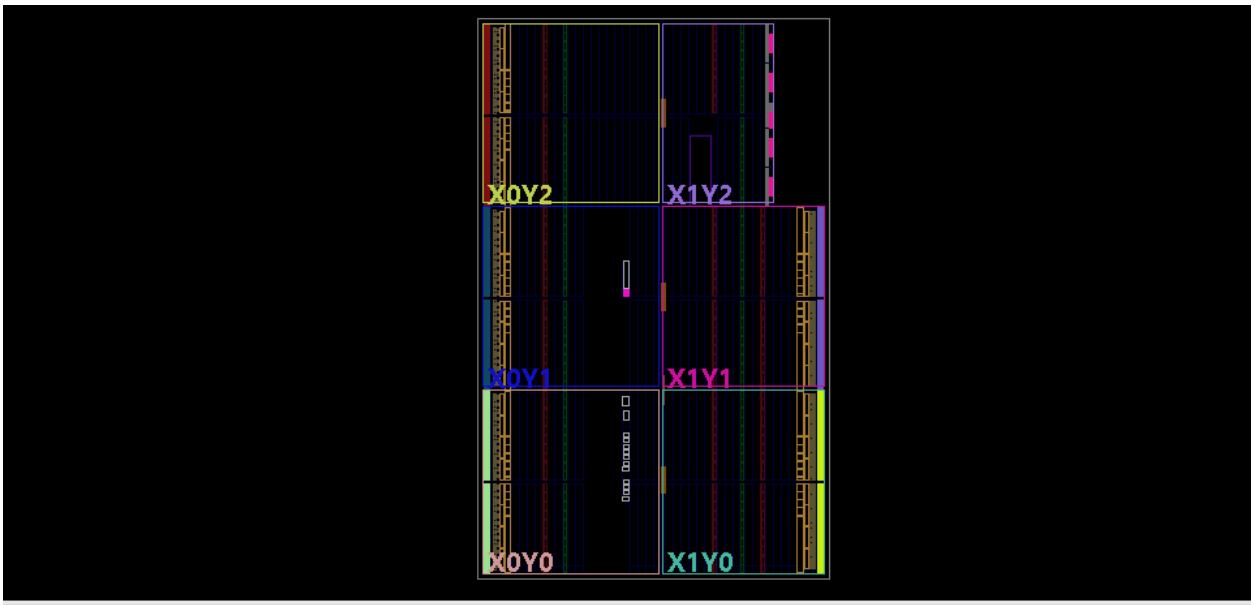


2. Transmitter (TX) Testbench

- **Stimulus:**
  - Applied parallel data inputs (TX_DATA_IN) with TX_START pulse to trigger transmission.
  - Tested multiple frames sequentially.
  - Enabled/disabling parity modes for validation.
- **Monitored Outputs:**
  - TX line checked for correct UART serial waveform:
    - Start bit (0),
    - Data bits in correct order,
    - Optional parity bit,
    - Stop bit (1).
  - TX_DONE observed to ensure transmission completion.



Synthesis



uns ×

stions	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Run Strategy
	487	**75**	**0**	**0**	**0**	**9/17/25, 1:26 PM**	**00:00:30**	**Vivado Synthesis Defaults (Vivado Synthesis 2024)**
								Vivado Implementation Defaults (Vivado Implementation 2024)
	48	60	0.5	0	0	9/13/25, 9:15 AM	00:00:46	Vivado Synthesis Defaults (Vivado Synthesis 2024)

Utilization Report:

Name	Slice LUTs (20800)	Slice Registers (41600)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
N UART	583	236	1	54	1
BR_GEN_INST (BaudRateGen)	439	33	0	0	0
RECEIVER_INST (RECEIVER)	27	43	0	0	0
RX_FIFO (fifo_generator_0)	48	60	0.5	0	0
TRANSMITTER_INST (TRANSMITTER)	21	40	0	0	0
TX_FIFO (fifo_generator_0)	48	60	0.5	0	0

Project Summary – UART Hardware Implementation in Vivado:

In this project, a Universal Asynchronous Receiver-Transmitter (UART) was implemented and verified in Xilinx Vivado. The design provides reliable serial communication with integrated error detection mechanisms.

Key Features of the Design:

1. Configurable Baud Rate Generator (BRG)
    - A parameterized baud rate generator was developed to support variable data rates.
    - The BRG derives timing ticks from a 50 MHz system clock using a programmable divider (UBRRL input).
2. Receiver (RX) Module
    - Handles serial-to-parallel data conversion.
    - Supports parity checking (even/odd, enabled via PARITY_EN and PARITY_MODE).
    - Implements error detection mechanisms:
        - Parity Error: Detected if received parity does not match expected parity.
        - Frame Error: Flags invalid stop bit reception.
        - Overrun Error: Raised when a new frame arrives before the previous data is consumed.
    - Internal FSM controls start-bit sampling, data shifting, optional parity checking, and stop-bit validation.
3. Transmitter (TX) Module
    - Performs parallel-to-serial data conversion.
    - Supports optional parity bit generation.
    - Ensures proper start, data, parity, and stop-bit sequencing.
4. FIFO Buffers (Implemented using Xilinx fifo_generator)
    - Separate TX FIFO and RX FIFO were integrated.
    - Provide asynchronous buffering to decouple processor and UART timing.
    - Prevent data loss under bursty traffic conditions.

5. Error Detection and Debugging Support
    - Real-time error flags (PARITY_ERROR, FRAME_ERROR, OVERRUN_ERROR) are exposed as top-level outputs.
    - Debug signals (State_dpg, bit_idx_dpg) allow FSM monitoring during simulation.

Achievements:

- Successfully built a modular UART system with transmit, receive, and error detection functionality.
- Implemented robust error handling (parity, frame, overrun) ensuring reliable communication.
- Added FIFO buffering for both directions, enhancing throughput and system flexibility.
- Verified synthesizability in Vivado with reusable parametrized modules.
- Created a scalable and testbench-friendly design with debug outputs.