Faculty of Engineering, Ain-Shams University
Communication Systems Engineering Program
National Telecommunication Institute
FPGA Digital Design Training
Summer 25

Final Project
"A Custom APB UART IP"

Submitted to:

Eng. Mohammed Salah

Submitted by:

Ahmed Youssef Abdelfattah Youssef Semary

Date of submission: 4/9/2025

Table of Contents

1.0 INTRODUCTION

The Universal Asynchronous Receiver/Transmitter (UART) is a widely used hardware block for serial communication. Unlike parallel interfaces, UART transmits and receives data sequentially, one bit at a time, over two dedicated lines: one for transmit (TX) and one for receive (RX). This simplicity makes UART highly popular in embedded systems, microcontrollers, and FPGA-based.

A UART communication link is organized into frames, which define how data is structured on the line. The most common configuration is the 8-N-1 frame format: one start bit (logic 0), eight data bits transmitted least significant bit first, no parity bit, and one stop bit (logic 1). Between frames, the line remains idle at logic 1. This standard frame structure ensures compatibility across a wide range of devices and applications.

Another key parameter in UART communication is the baud rate, which defines the number of bits transmitted per second (e.g., 9600, 115200). Both the transmitter and receiver must be configured with the same baud rate to maintain synchronization. For example, at 9600 baud, each bit is transmitted over approximately 104 µs. To achieve correct sampling of the data, the receiver must generate timing signals based on the agreed baud rate, typically using a baud rate counter driven by a higher system clock.

In modern SoC design, peripherals such as UART are integrated as memory-mapped devices connected to a system bus. The AMBA APB (Advanced Peripheral Bus) provides a simple and low-power interface for such peripherals. By wrapping the UART core with an APB slave interface, control, status, data, and configuration registers become accessible to software through standard read and write transactions.

The goal of this project is to design, implement, and verify a custom UART IP with an APB wrapper. This includes implementing transmit and receive FSMs, register mapping, baud rate generation, and an APB protocol-compliant slave interface. In addition to functional design, the project reinforces concepts of RTL development in Verilog, memory-mapped peripheral design, and FPGA-based implementation using Quartus Prime.

2.0 DESIGN ANALYSIS

Our design consists mainly of three modules and in this section, we are goind to analyze each one of them in some detail:

2.1 UART Receiver:

The UART Receiver module is responsible for sampling incoming serial data one bit at a time and reconstructing complete data words. It is composed of three key submodules: the Baud Counter, the FSM Controller, and the Shift Register.

2.1.1 Baud Counter

This counter takes a load value (Port named: Load_Value) and generates a counter output (count) which counts from 0 to 15 indicating a bit period according to the required baud rate. The value of (Load_Value) is calculated according to the following formula:

$$\text{Load\_Value} = \frac{1}{\text{Baud Rate} * 16 * \text{System Clk Period}} - 1$$

So, for a system that runs on a 100MHz clock frequency, to achieve 9600 baud rate, Load_Value should approximately be 650.

2.1.2 FSM Controller

The finite state machine controls the behaviour of the other modules, i.e. generates control signals such as: shift_en, count_en and count_rst. The fsm takes the counter value of the baud counter as an input to recognize bit timings. The fsm has four states described as follows:

- Idle: No activity occurs in this state. The receiver waits for the line to go low, which indicates the arrival of a start bit.
- Start: Triggered when the RX line transitions low (indicating the start bit). In this state, busy signal is asserted and counter is enabled till it reaches the value 8 (which indicates mid-bit time). Immediately afterwards, the counter is reset and state changes to data.
- Data: this is the state when data gets sampled and shifted into the shift register. Shift enable is asserted for one clock cycle only after a bit period has passed. This repeats for 8 cycles in total (indicating a full frame) before transition to stop state.

- Stop: This state checks for the stop bit. If a one is present at this point, no error occurred, otherwise err signal is asserted. After one bit period, done signal is asserted and state returns to idle again.

2.1.3 Shift Register

The Shift Register is a Serial-In Parallel-Out (SIPO) structure that reconstructs the received data word from serial input bits. It captures each bit when the shift enable signal is asserted by the FSM and outputs the complete word once the frame is received.

2.2 UART Transmitter

The UART Transmitter module is responsible for converting parallel data into serial frames and transmitting them over the tx line. Similar to the receiver, it relies on a Baud Counter, an FSM Controller, and a Shift Register.

2.2.1 Baud Counter

The transmitter's baud counter is just the same as the receiver's. It generates the timing reference for each transmitted bit. Based on the configured baud rate and system clock, it ensures that each bit (start, data, and stop) is held on the tx line for the correct duration.

2.2.2 FSM Controller

Similar to the FSM Controller of the receiver, this FSM Controller has the following states:

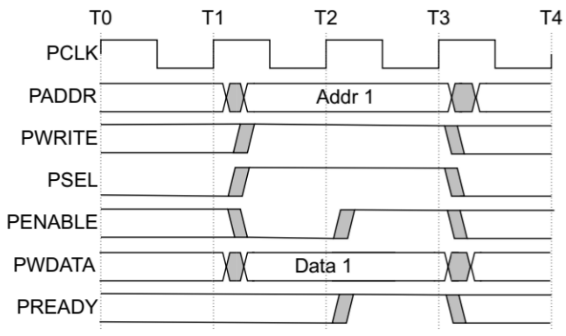
- Idle: The line remains high while no data is being transmitted. The busy signal is deasserted.
- Start: Once data is loaded and transmission is enabled, the FSM drives the line low to send the start bit. The busy signal is asserted.
- Data: The FSM sequentially shifts out the 8 data bits, least significant bit first. Each bit is held on the line for one bit period.
- Stop: After all data bits are transmitted, the FSM drives the line high to send the stop bit. At the end of this period, the done signal is asserted and the FSM returns to Idle.

2.2.3 Shift Register

The parallel-to-serial (PISO) shift register loads the byte to be transmitted and shifts one bit onto the line at each baud period, under the control of the FSM. This mechanism ensures correct serialization of data.

2.3 APB UART Wrapper

The APB UART Wrapper integrates the UART transmitter and receiver into a memory-mapped peripheral accessible via the AMBA APB protocol. It provides an interface between the system bus and the UART core through a set of control and status registers. Writing to or reading from these registers takes two clock cycles as the following figure illustrates:



2.3.1 Register File

The design exposes the following 32-bit registers:

- CTRL_REG (0x0000): Contains control bits for tx_en, rx_en, tx_rst, and rx_rst.
- STATUS_REG (0x0001): Reports rx_busy, tx_busy, rx_done, tx_done, and rx_error.
- TX_DATA (0x0002): Holds the byte to be transmitted.
- RX_DATA (0x0003): Stores the most recently received byte.
- BAUDDIV (0x0004): Configures the baud rate divisor (bonus feature).

2.3.2 APB FSM Controller

The wrapper implements an APB-compliant FSM with three states:

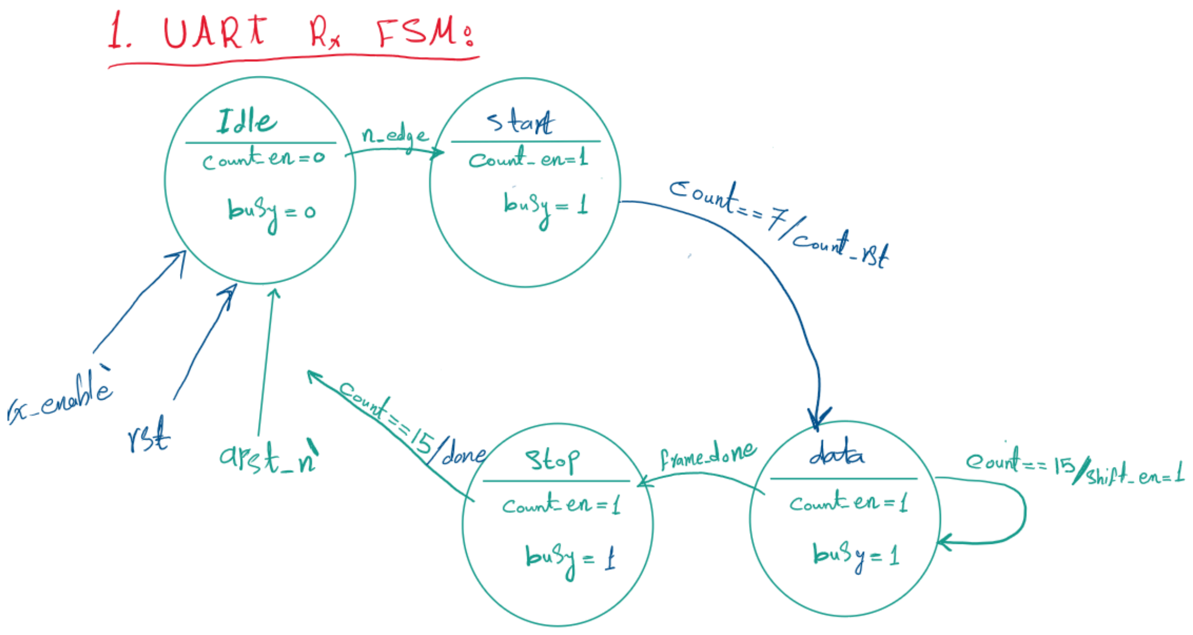
- Idle: Waits for PSEL to be asserted.
- Write: On PWRITE=1 and PENABLE=1, writes data from PWDATA to the addressed register and asserts PREADY for one cycle.
- Read: On PWRITE=0 and PENABLE=1, places the addressed register contents on PRDATA and asserts PREADY.

Between transactions, the FSM always returns to Idle.

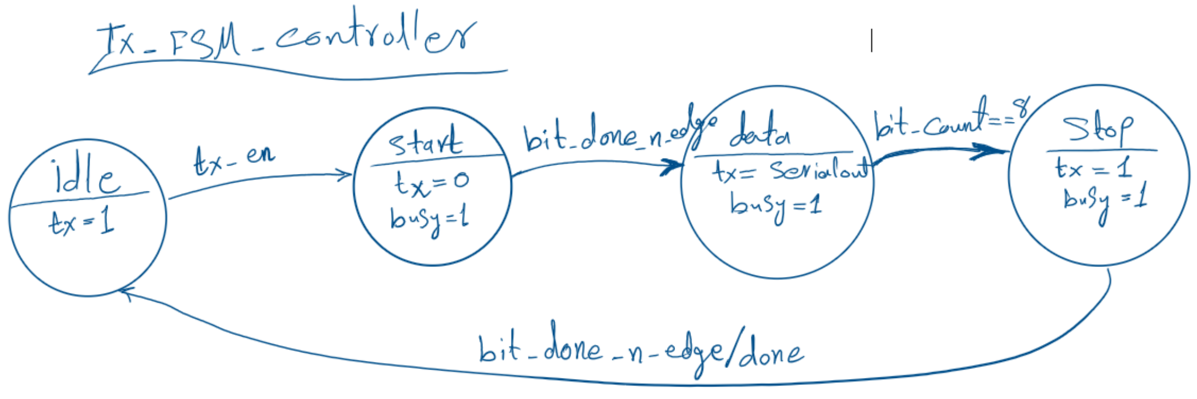
3.0 STATE DIAGRAMS

Here are the state diagrams for each module in the design:

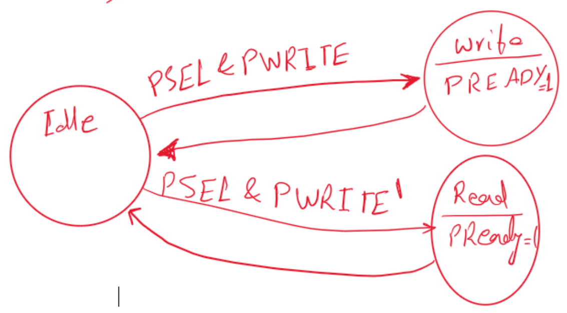
3.1 Receiver FSM Controller



3.2 Transmitter FSM Controller



3.3 APB FSM Controller



4.0 DESIGN DECISIONS

These are some of the decisions made for this design:

4.1 Oversampling in Baud Counter

The receiver uses a 16× oversampling technique to improve reliability. By sampling each bit multiple times, the FSM can align sampling to the middle of the bit period, reducing errors due to clock mismatch, jitter, or noise. This makes data recovery more robust compared to single-sample approaches.

4.2 Error Detection in Receiver

A framing error flag (rx_error) was included to improve robustness. This ensures that invalid stop bits can be flagged to software, allowing higher-level protocols to react appropriately.

4.3 APB FSM Simplification

The APB wrapper FSM was intentionally limited to three states (Idle, Write, Read) to keep the bus interface straightforward while still meeting the APB protocol requirements(Not Including the wait states).

PREADY was asserted for a single cycle after each transfer to mimic typical APB peripheral behavior.

5.0 VERIFICATION STRATEGY

The verification strategy is kept as simple as possible; only three testbenches for UART Receiver, UART Transmitter and the APB Wrapped UART respectively.

5.1 UART Receiver

The strategy is to receive a sequence of incoming bits and observing the output at the end of the frame.

5.2 UART Transmitter

The same strategy applies; this time by loading a word into the transmitter's register and observing the output Tx signal.

5.3 APB Wrapped UART

The approach includes:

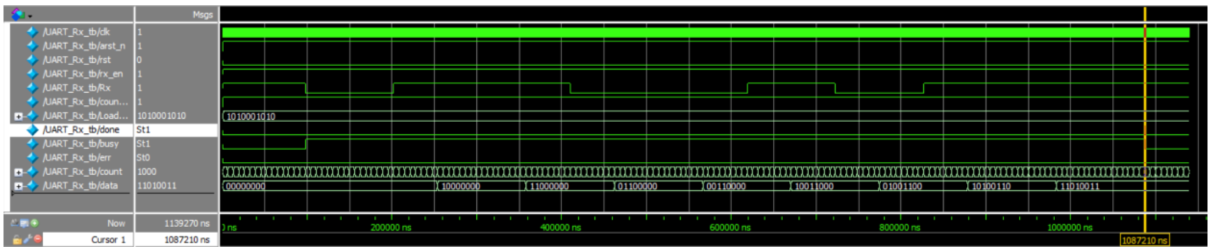
- Writing data to register 0x02 (The tx_data to be transmitted).
- Writing data to register 0x00 (The values tx_en, rx_en, tx_rst, rx_rst).
- Loading a UART Transmitter external instance by an 8-bit word.
- After writing completion, we wait for a complete data frame and check the received data in each device.

6.0 SIMULATION RESULTS

The outputs of each testbench is the following:

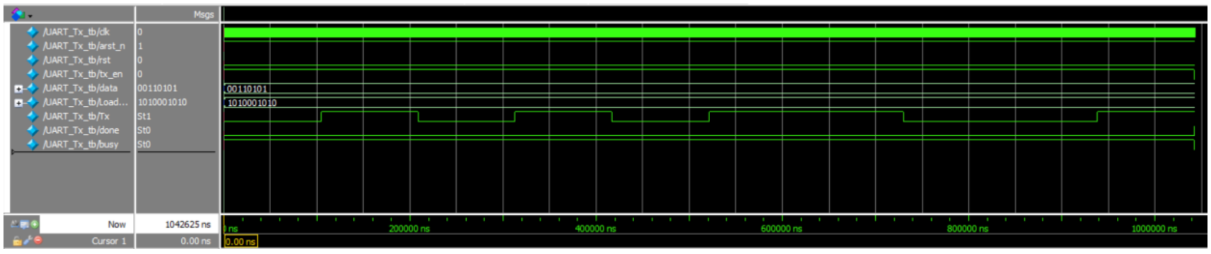
6.1 UART Receiver Results

The waveform of a complete frame reception



6.2 UART Transmitter Results

The waveform of a complete frame Transmition

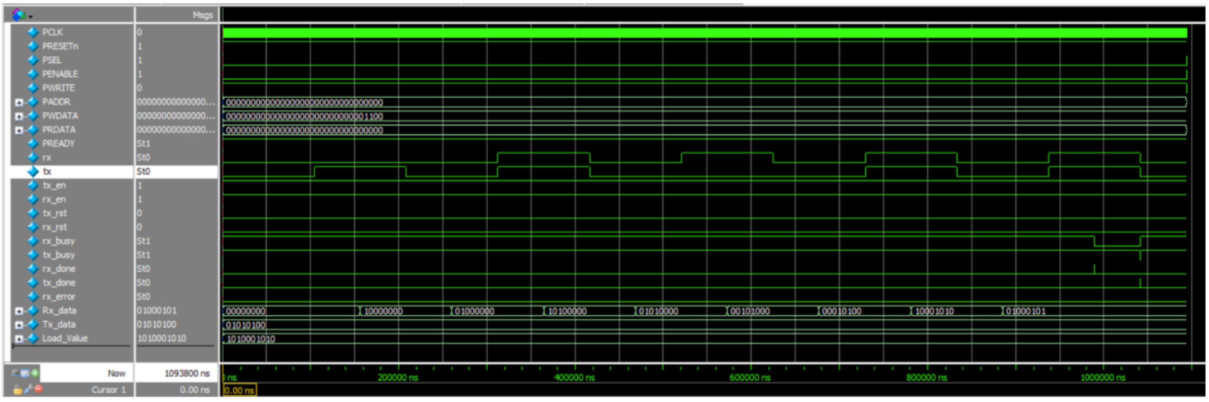


6.3 APB Wrapped UART

Log output:

```
# IP Reception Test Passed
# IP Transmission Test Passed
# Rx_data: 01010100
# Tx_data: 01000101
```

Waveform:



	Msgs
PCLK	0
PRESETn	1
PSEL	1
PENABLE	1
PWRITE	0
PADDR	00000000000000...
PWDATA	00000000000000...
PRDATA	00000000000000...
PREADY	St1
rx	St0
tx	St0
tx_en	1
rx_en	1
tx_rst	0
rx_rst	0
rx_busy	St1
tx_busy	St1
rx_done	St0
tx_done	St0
rx_error	St0
Rx_data	01000101
Tx_data	01010100
Load_Value	1010001010
Now	1093800 ns
Cursor 1	0.00 ns

7.0 CONCLUSION

In this project, we successfully designed and verified a custom UART peripheral wrapped with an AMBA APB slave interface. The design included a transmitter, a receiver with oversampling for reliable data recovery, and an APB-compliant wrapper providing register-based control and status visibility. Through this work, we gained practical experience in RTL design, finite state machine implementation, register mapping, and memory-mapped bus integration. Simulation and verification confirmed the correctness of the design. Overall, the project provided valuable insight into peripheral design within an SoC context and laid the groundwork for future extensions such as parity support.