APB-UART PROJECT

By:

Omar Ahmed Fouad

Supervisor:

Eng/ Mohammed Salah

Contents

1. Introduction

In this project, a Universal Asynchronous Receiver and Transmitter (UART) was implemented and connected to an Advanced Peripheral Bus (APB) wrapper. The design was created from the ground up using Finite State Machines (FSMs) to control each component.

The project is divided into three main modules:
- Transmitter (TX): Responsible for serializing parallel data and transmitting it bit by bit.
- Receiver (RX): Responsible for deserializing incoming serial data and reconstructing the parallel word.
- APB Wrapper: Provides a memory-mapped interface for configuring and monitoring the UART using standard APB protocol signals.

This modular approach allows separation of concerns and simplifies testing, since each module can be verified individually before integration.

2. Design Analysis

The system architecture is based entirely on FSM-driven control. Each FSM was carefully designed to ensure correct UART protocol behavior and reliable communication through APB.

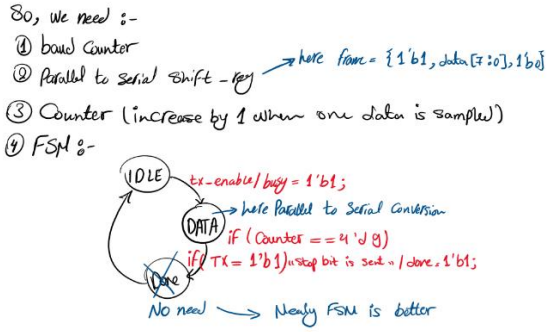
- Transmitter (TX): Uses a PISO (Parallel-In Serial-Out) shift register and a baud-rate counter. It transmits start bit, 8 data bits (LSB first), and a stop bit.
- Receiver (RX): Uses an edge detector to detect the start bit, a baud-rate counter for sampling at mid-bit, and a SIPO (Serial-In Parallel-Out) shift register to reconstruct the received word.
- APB Wrapper: Exposes four memory-mapped registers:
  - CTRL_REG → enable/reset signals for TX and RX
  - TX_DATA → holds the character to be transmitted
  - RX_DATA → holds the received character
  - STATS_REG → contains status flags (tx_busy, tx_done, rx_busy, rx_done, rx_err)

This hierarchical design provides both flexibility and scalability.

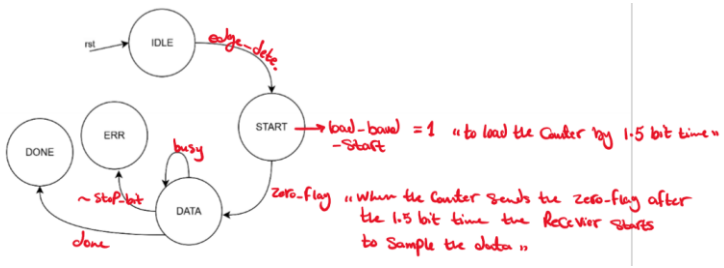
3. State Diagrams

Each module is based on a dedicated FSM. The following diagrams illustrate the FSM operation for each part of the design:

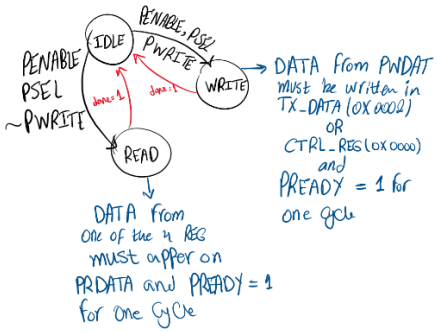
- Transmitter FSM :



- Receiver FSM :



- APB FSM :



4. Design Decisions

Several important design choices were made during development:
- FSM-based approach: Ensures deterministic behavior, easier debugging, and clear modular separation.
- Fixed Baud Rate (9600 bps): For simplicity, the baud counter values were derived directly from a 100 MHz system clock.
- APB Wrapper: Added as an interface layer to allow memory-mapped register access, making the UART reusable in SoC designs.
- Status Register: A dedicated status register was included to simplify verification and provide real-time feedback on the state of TX and RX.

5. Verification Methodology

The verification strategy was carried out through a Verilog testbench simulating APB read/write transactions. The following sequence was performed:

1. TX Data Preparation: APB writes the character 'A' into TX_DATA register.
2. Enabling the UART: APB writes to CTRL_REG to enable both tx_enable and rx_enable.
3. Check Enables: APB reads back CTRL_REG to confirm the enables are set.
4. Disabling the UART: APB writes again to CTRL_REG to clear both enables.
5. Check Disables: APB reads back CTRL_REG to confirm the enables are cleared.
6. Busy State Check: APB reads STATS_REG to verify that both tx_busy = 1 and rx_busy = 1.
7. Transmission Delay: The testbench waits 10 bit-times to ensure the entire frame has been transmitted and received.
8. RX Data Check: APB reads RX_DATA to confirm that the received character matches 'A'.
9. Completion Check: APB reads STATS_REG to confirm that tx_done = 1 and rx_done = 1.

6. Simulation Results

Screenshots from the simulation are provided below to demonstrate functional correctness:

- Transcript:

```
    $display(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
    $display(" ");
    $display("THE APB MASTER TOLD THE SLAVE TO LOAD LETTER A IN THE TX IN ORDER TO SEND IT"); // display TX have been loaded
    $display(" ");
    $display(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

```

    $display("TURNING ON THE ENABLES");

```

```
    $display(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
    $display(" ");
    $display("tx_enable is %b and rx_enable is %b", PRDATA[0], PRDATA[2]);
    $display(" ");
    $display(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

```

    $display("TURNING OFF THE ENABLES");

```

```
    $display(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
    $display(" ");
    $display("tx_enable is %b and rx_enable is %b", PRDATA[0], PRDATA[2]);
    $display(" ");
    $display(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

    $display("UART SHOULD BE WORKING");
```

```
    $display(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
    $display(" ");
    $display("THE STATE OF tx_busy is %b and rx_busy is %b", PRDATA[0], PRDATA[2]);
    $display(" ");
    $display(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

```
    repeat(10*10417) @(negedge PCLK); // to make sure that the data has been sent and received
    $display("10 BIT CYCLES HAVE FINISHED");
```

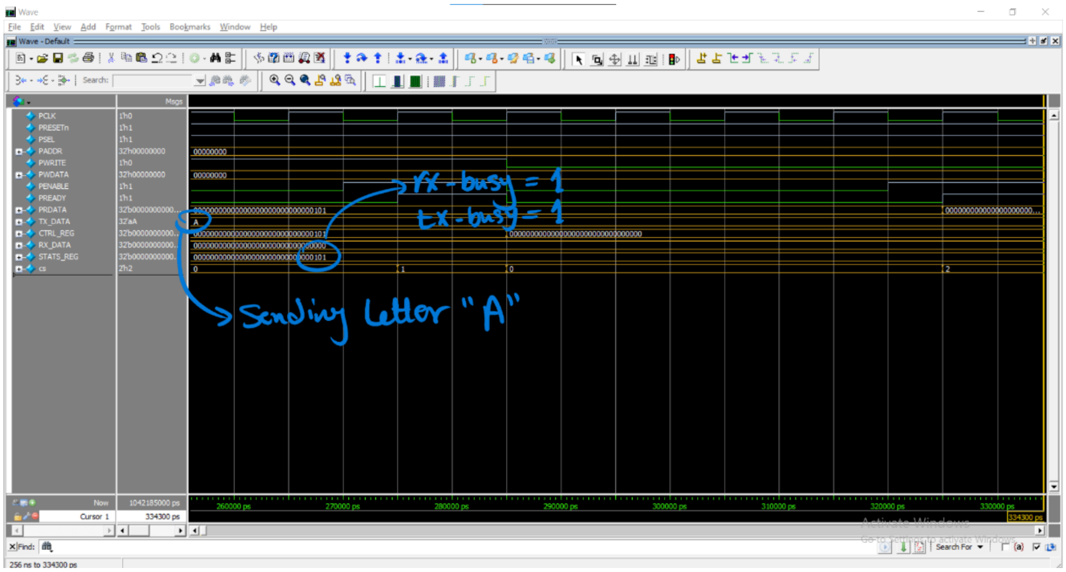
```
    $display(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
    $display(" ");
    $display("THE STATE OF tx_done is %b and rx_done is %b", PRDATA[1], PRDATA[3]);
    $display(" ");
    $display(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

    $display("CHECKING THE RECEIVED INFO");
```

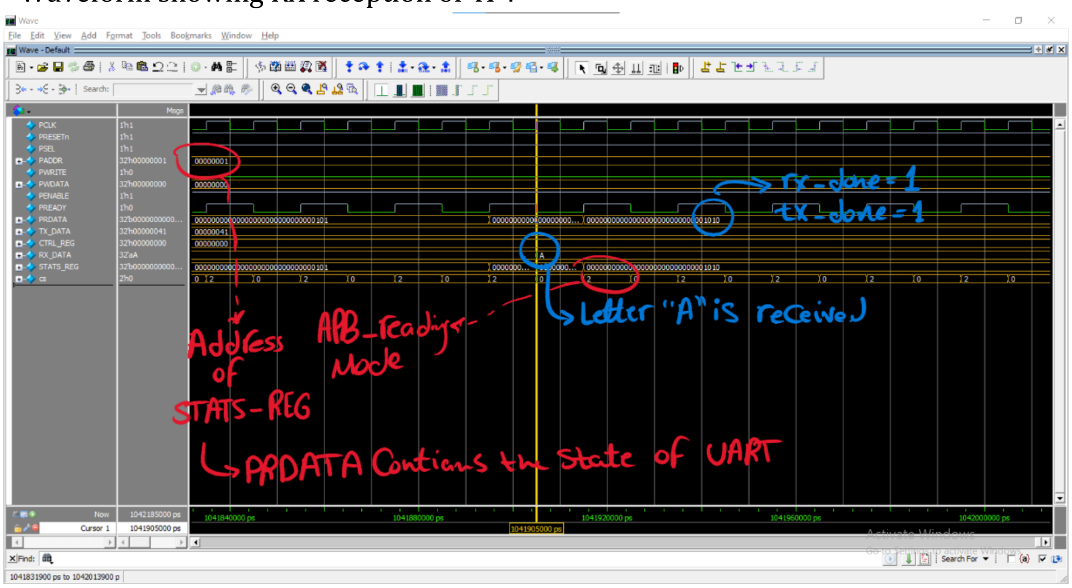
```
    if(PRDATA != "A") begin
        $display(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
        $display(" ");
        $display("There is an error");
        $display(" ");
        $display(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
    end
    else begin
        $display(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
        $display(" ");
        $display("THE LETTER THAT HAS BEEN RECEIVED IS %s", PRDATA[7:0]);
        $display(" ");
        $display(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
    end
```

```
VSIM 4> run -all
# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
#
# THE APB MASTER TOLD THE SLAVE TO LOAD LETTER A IN THE TX IN ORDER TO SEND IT
#
# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
# TURNING ON THE ENABLES
# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
#
# tx_enable is 1 and rx_enable is 1
#
# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
# TURNING OFF THE ENABLES
# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
#
# tx_enable is 0 and rx_enable is 0
#
# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
# UART SHOULD BE WORKING
# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
#
# THE STATE OF tx_busy is 1 and rx_busy is 1
#
# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
# 10 BIT CYCLES HAVE FINISHED
# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
#
# THE STATE OF tx_done is 1 and rx_done is 1
#
# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
# CHECKING THE RECEIVED INFO
# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
#
# THE LETTER THAT HAS BEEN RECEIVED IS A
#
# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
# ** Note: $stop    : D:/Engineering/ASU/3_Summer2/DIGITAL_UNI/Project/7_Q_SIM/APB_UART_tb.v(279)
#    Time: 1042185 ns  Iteration: 1  Instance: /APB_UART
# Break in Module APB_UART at D:/Engineering/ASU/3_Summer2/DIGITAL_UNI/Project/7_Q_SIM/APB_UART_tb.v line 279
```

- Waveform showing TX transmission of ‘A’ :



- Waveform showing RX reception of ‘A’ :



7. Conclusion

The project successfully implemented a UART system with APB integration, entirely controlled by FSMs.

Key achievements include:
- Modular design (separated TX, RX, and APB wrapper).
- Fully functional data transmission and reception at 9600 baud.
- Correct APB read/write protocol handling with PRDATA and PWDATA.
- Verification through a structured testbench that confirmed UART functionality and synchronization.

This design provides a solid foundation for further improvements such as adding parity support, implementing a programmable baud rate, and integrating with FPGA hardware. Overall, the design demonstrates a complete APB–UART communication system that is robust, verifiable, and extendable.

Additional Resources

All RTL codes, testbench code, constraint files, and complete source codes along with the results obtained from running Vivado are available on my GitHub repository. In addition, my handwriting analysis, a detailed description of the errors I faced during development, and the corresponding solutions can also be found in the following link:

1. ***https://github.com/omar3363/APB_UART_Project***