Faculty of Engineering and Technology

Electrical and Computer Engineering Department

HARDWARE DESIGN LAB

ENCS5131

Project#1 – Design Verification (APB FIFO Verification using UVM)

Prepared by: Enas Qutit

Student NO: 1210236

Instructor: Ayman Hroub

Section: 3

Date: 1-1-2026

1 Abstract

This project presents the verification of an APB-based synchronous FIFO using the UVM methodology a complete UVM testbench was developed including an APB agent sequences scoreboard and coverage collection directed and random tests were applied to validate normal and boundary operations the verification environment successfully detected multiple intentional bugs in the RTL design demonstrating the effectiveness of the implemented testbench.

Table of Contents

2 Table of figures

3 Theory

3.1 APB Interface

APB interface that is used to connect the FIFO to the processor this interface allows the processor to send commands and data to the FIFO using read and write operations standard APB signals are used such as PSEL to select the FIFO PENABLE to control the transfer PWRITE to choose between read and write operations PADDR to select the register address and PWDATA and PRDATA to transfer data the APB interface provides a simple and reliable way for the processor to access the FIFO registers[1].

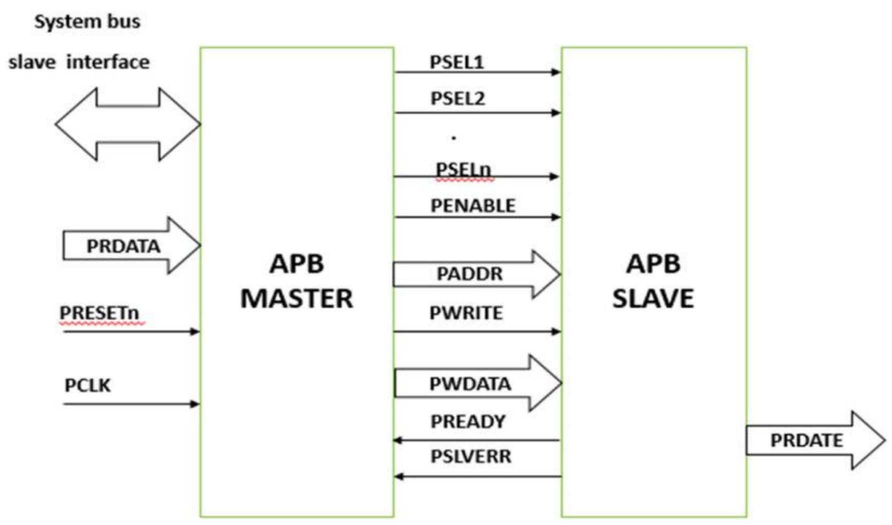


Figure 1 : APB master–slave interface signals[2].

3.2 FIFO Memory and Control Logic

FIFO memory that is used to store data temporarily data is written into the FIFO when a write operation occurs and read from the FIFO when a read operation occurs the FIFO uses control logic to manage the write pointer read pointer and the number of stored elements this control logic ensures that data is written and read in the correct order it also checks whether the FIFO is full or empty before allowing write or read operations this helps prevent incorrect data access and supports proper FIFO operation[3].

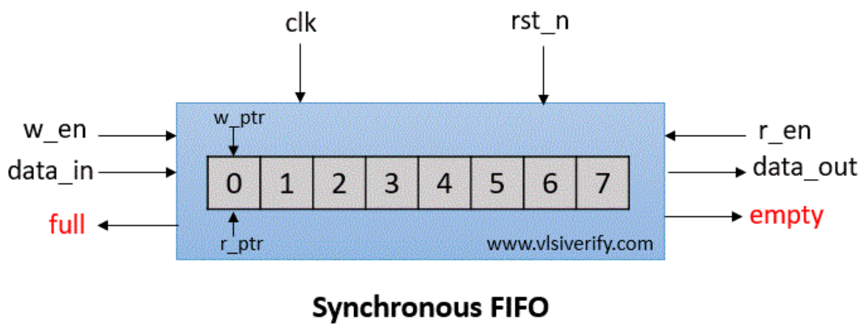


Figure 2 : Synchronous FIFO [4].

3.3 Control and Status Registers

control and status registers that are used to manage and observe the FIFO operation the control register allows the processor to enable or disable the FIFO clear its contents and define how the FIFO behaves when it becomes full these control bits help the user configure the FIFO behavior according to system needs the status register provides information about the current state of the FIFO it shows whether the FIFO is empty or full and indicates if overflow or underflow has occurred during operation these status signals help detect error conditions and monitor the FIFO state in addition the design includes threshold registers that are used to set almost full and almost empty levels these registers allow early warning before the FIFO becomes completely full or empty[1].

3.4 Status Flags and Error Handling

status flags that indicate the current state of the FIFO the empty flag shows when no data is stored while the full flag indicates that the FIFO cannot accept more data almost empty and almost full flags provide early warnings based on programmable threshold values the design also includes overflow and underflow flags which occur when writing to a full FIFO or reading from an empty FIFO these flags help detect incorrect FIFO usage and allow the verification environment to identify functional bugs during simulation[5].

3.5 UVM Testbench Architecture

The UVM Testbench Architecture is a structured verification environment used to test and validate a design. Its main purpose is to apply inputs to the design, observe the outputs, and check that the design behaves correctly. It consists of basic components such as the driver, which sends signals to the design, the monitor, which observes the design behavior, and the scoreboard, which compares the actual results with the expected ones. These components are organized together in an agent and an environment to simplify testing. Overall, the UVM testbench helps verify the design in an organized and reliable way[6].

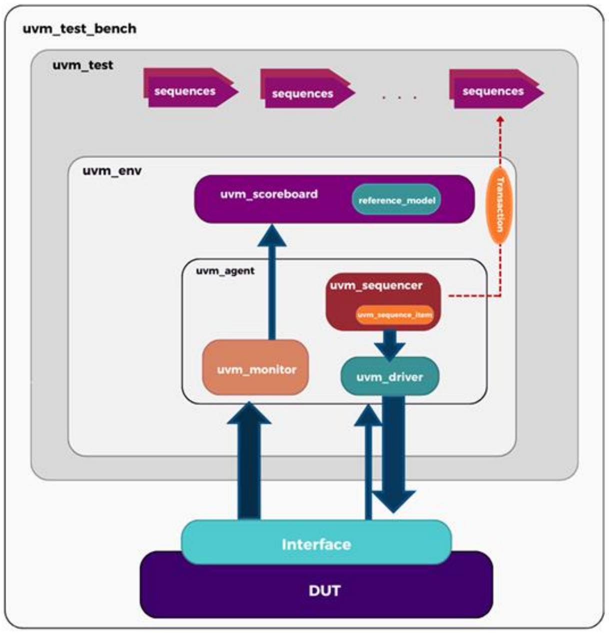


Figure 3 : UVM Environment[6].

4 Stimulus Description

4.1 Directed Sequence

The directed sequence follows a predefined order of operations it first enables the FIFO by writing to the CTRL register then it performs consecutive write transactions to the DATA register to fill the FIFO up to its maximum depth after that an additional write transaction is issued to the DATA register to trigger an overflow condition next consecutive read transactions from the DATA register are performed to empty the FIFO finally one extra read transaction is applied to trigger an underflow condition.

4.2 Random Sequence

The random sequence generates a fixed number of transactions with randomized read/write operations each transaction randomly selects between accessing the CTRL or DATA register and randomly chooses whether the operation is a read or a write this sequence provides random traffic to the FIFO interface and helps exercise different operation combinations beyond the directed tests.

5 Coverage

Functional coverage was collected using a cover group in the monitor the coverage tracks read and write operations on the APB interface to ensure that both types of transactions were exercised it also covers accesses to the CTRL and DATA registers to verify that control operations and data transfers were both tested a cross coverage between the operation type (read/write) and the accessed register (CTRL/DATA) was included to confirm that all combinations were observed during simulation if available a coverage screenshot can be attached to demonstrate coverage results.

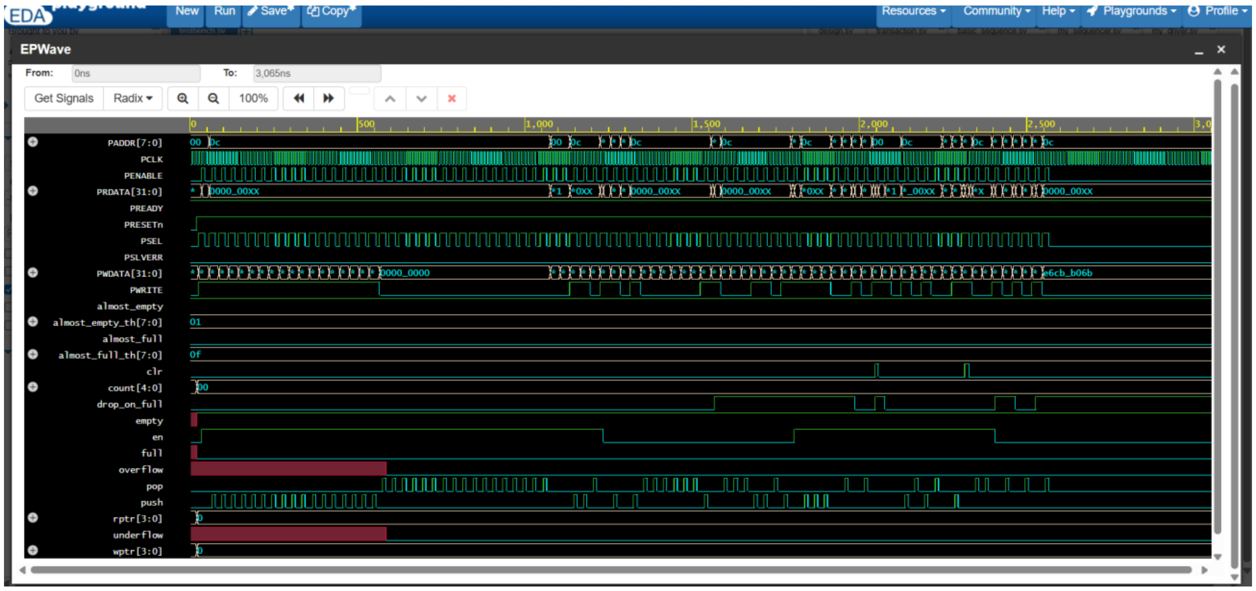
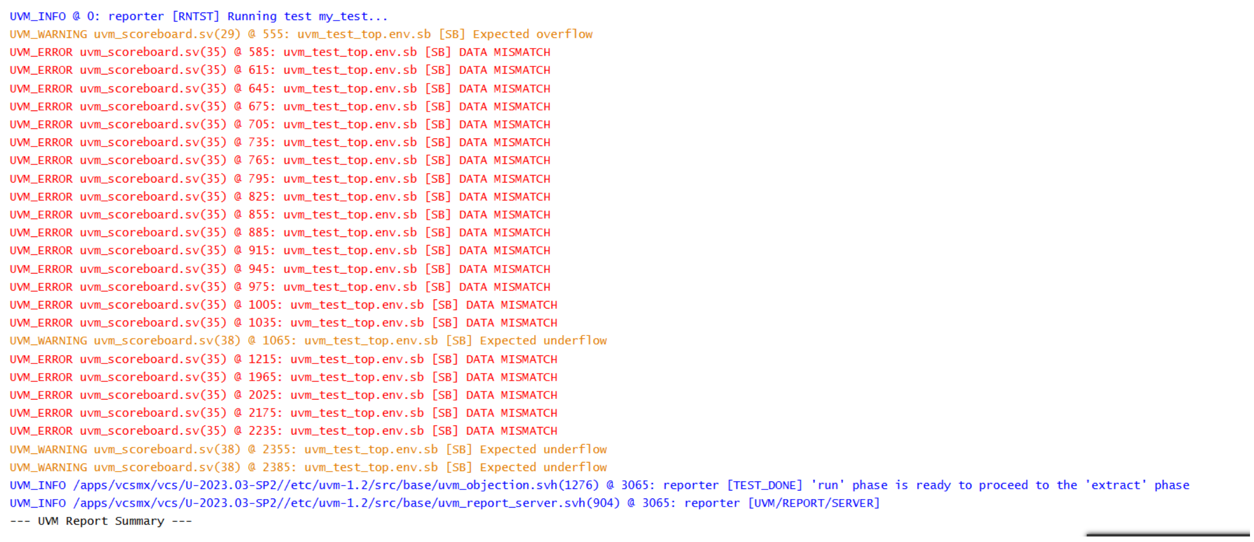


Figure 4 : EPWave waveform

The waveform demonstrates read/write accesses to the FIFO and the corresponding changes in FIFO status signals.

6 Bug Report

During simulation, multiple functional bugs were detected in the RTL design using the UVM scoreboard and reference FIFO model. These bugs are intentionally present in the design and were successfully identified by the verification environment.



```
UVM_INFO @ 0: reporter [RNTST] Running test my_test...
UVM_WARNING uvm_scoreboard.sv(29) @ 555: uvm_test_top.env.sb [SB] Expected overflow
UVM_ERROR uvm_scoreboard.sv(35) @ 585: uvm_test_top.env.sb [SB] DATA MISMATCH
UVM_ERROR uvm_scoreboard.sv(35) @ 615: uvm_test_top.env.sb [SB] DATA MISMATCH
UVM_ERROR uvm_scoreboard.sv(35) @ 645: uvm_test_top.env.sb [SB] DATA MISMATCH
UVM_ERROR uvm_scoreboard.sv(35) @ 675: uvm_test_top.env.sb [SB] DATA MISMATCH
UVM_ERROR uvm_scoreboard.sv(35) @ 705: uvm_test_top.env.sb [SB] DATA MISMATCH
UVM_ERROR uvm_scoreboard.sv(35) @ 735: uvm_test_top.env.sb [SB] DATA MISMATCH
UVM_ERROR uvm_scoreboard.sv(35) @ 765: uvm_test_top.env.sb [SB] DATA MISMATCH
UVM_ERROR uvm_scoreboard.sv(35) @ 795: uvm_test_top.env.sb [SB] DATA MISMATCH
UVM_ERROR uvm_scoreboard.sv(35) @ 825: uvm_test_top.env.sb [SB] DATA MISMATCH
UVM_ERROR uvm_scoreboard.sv(35) @ 855: uvm_test_top.env.sb [SB] DATA MISMATCH
UVM_ERROR uvm_scoreboard.sv(35) @ 885: uvm_test_top.env.sb [SB] DATA MISMATCH
UVM_ERROR uvm_scoreboard.sv(35) @ 915: uvm_test_top.env.sb [SB] DATA MISMATCH
UVM_ERROR uvm_scoreboard.sv(35) @ 945: uvm_test_top.env.sb [SB] DATA MISMATCH
UVM_ERROR uvm_scoreboard.sv(35) @ 975: uvm_test_top.env.sb [SB] DATA MISMATCH
UVM_ERROR uvm_scoreboard.sv(35) @ 1005: uvm_test_top.env.sb [SB] DATA MISMATCH
UVM_ERROR uvm_scoreboard.sv(35) @ 1035: uvm_test_top.env.sb [SB] DATA MISMATCH
UVM_WARNING uvm_scoreboard.sv(38) @ 1065: uvm_test_top.env.sb [SB] Expected underflow
UVM_ERROR uvm_scoreboard.sv(35) @ 1215: uvm_test_top.env.sb [SB] DATA MISMATCH
UVM_ERROR uvm_scoreboard.sv(35) @ 1965: uvm_test_top.env.sb [SB] DATA MISMATCH
UVM_ERROR uvm_scoreboard.sv(35) @ 2025: uvm_test_top.env.sb [SB] DATA MISMATCH
UVM_ERROR uvm_scoreboard.sv(35) @ 2175: uvm_test_top.env.sb [SB] DATA MISMATCH
UVM_ERROR uvm_scoreboard.sv(35) @ 2235: uvm_test_top.env.sb [SB] DATA MISMATCH
UVM_WARNING uvm_scoreboard.sv(38) @ 2355: uvm_test_top.env.sb [SB] Expected underflow
UVM_WARNING uvm_scoreboard.sv(38) @ 2385: uvm_test_top.env.sb [SB] Expected underflow
UVM_INFO /apps/vcsmx/vcs/U-2023.03-SP2//etc/uvm-1.2/src/base/uvm_objection.svh(1276) @ 3065: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
UVM_INFO /apps/vcsmx/vcs/U-2023.03-SP2//etc/uvm-1.2/src/base/uvm_report_server.svh(904) @ 3065: reporter [UVM/REPORT/SERVER]
--- UVM Report Summary ---
```

Figure 5 : UVM simulation log showing detected DATA MISMATCH errors and expected overflow and underflow warnings.

6.1 Wrong FULL condition

The FIFO asserts the `full` flag when the count reaches `DEPTH-1` instead of `DEPTH`. This causes the FIFO to indicate a full condition earlier than expected, leading to incorrect handling of write operations near the FIFO capacity.

6.2 DATA read without valid pop

The DATA register returns the value of `mem[rptr]` regardless of whether a valid pop operation has occurred. This results in incorrect data being read from the FIFO and causes multiple `DATA MISMATCH` errors reported by the scoreboard.

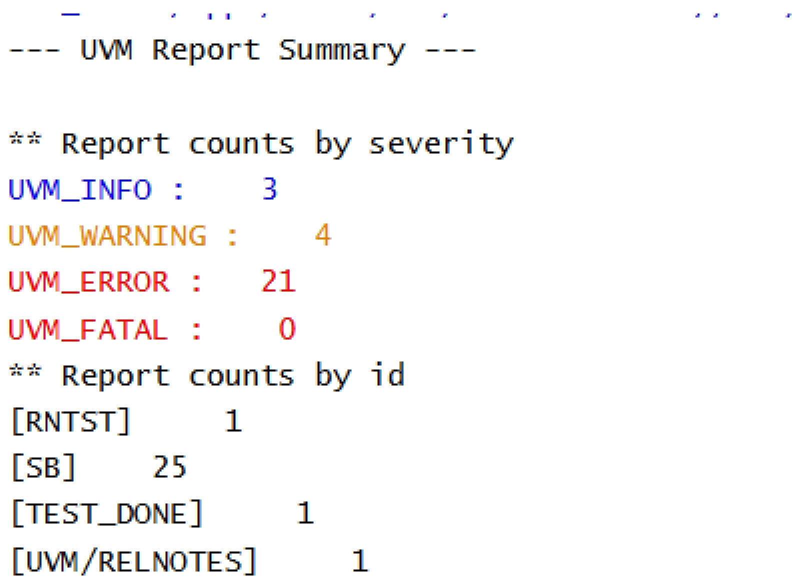
6.3 Overflow and Underflow flags not sticky

The overflow and underflow flags are cleared on any read operation instead of remaining asserted until explicitly cleared. This behavior violates the specification of sticky error flags and was observed as repeated "Expected overflow" and "Expected underflow" warnings in the simulation log.

6.4 Count width mismatch

The count signal is declared with a width of `[4:0]`, but the STATUS register reads `count[7:0]`. This mismatch causes an out-of-bounds access warning during compilation and may lead to incorrect count reporting.

7 Coverage & Report Summary



```
--- UVM Report Summary ---

** Report counts by severity
UVM_INFO :    3
UVM_WARNING :    4
UVM_ERROR :   21
UVM_FATAL :    0
** Report counts by id
[RNTST]     1
[SB]    25
[TEST_DONE]     1
[UVM/RELNOTES]     1
```

Figure 6 : Report Summary

8 Conclusion

The UVM verification environment successfully detected multiple functional bugs in the APB FIFO design, confirming the correctness and completeness of the testbench.

9 References

[1] ARM Ltd., *AMBA APB Protocol Specification*, ARM Architecture Reference Manual.

[2] https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcS600Iwrtp96uOPC_9UDeWnf64etR-fkTzcNHUX0HXy73sTQwjgZlsMc8nKLGtZug6zxCA&usqp=CAU

[3] Patterson, D. A., and Hennessy, J. L., *Computer Organization and Design*, Morgan Kaufmann.

[4]https://statics.mylandingpages.co/static/aaabwehecm7g6u5k/image/722fc1db148a465ab9c5aef2dac81f55.png

[5] Patterson, D. A., and Hennessy, J. L., *Computer Organization and Design*, Morgan Kaufmann.

[6] lab manual