



## **Design, Verification and Logical Synthesis for VLSI Circuits**

**תכן, אימות וסינתזה לוגית למעגלי VLSI**

**(5124711)**

# **Four-Port Switch Project**

**Implementation, Verification, and Synthesis**

**Majd Fadaos**

**22/11/2025**

# Four-Port Switch Project

## Table of Contents

<b>1. Introduction .....</b>	<b>3</b>
<b>2. Packet Specification .....</b>	<b>5</b>
<b>3. Packet Model Development .....</b>	<b>7</b>
<b>4. Constrained-Random Stimulus Generation .....</b>	<b>8</b>
<b>5. Stage A: Design (RTL Implementation) .....</b>	<b>10</b>
<b>6. Stage B: Verification Component Architecture .....</b>	<b>11</b>
6.1 Component Base Class .....	11
6.2 Sequencer .....	11
6.3 Driver .....	11
6.4 Monitor .....	11
6.5 Agent .....	12
6.6 Packet Verification Component (Packet_VC) .....	12
6.7 Checkers .....	12
6.8 Integration with the Switch Interface .....	13
6.9 Full DUT Verification Using Four Parallel Components .....	13
6.10 Expected Outcomes for Verification .....	14
<b>7. Stage C: Synthesis .....</b>	<b>15</b>
<b>8. Overall Deliverables .....</b>	<b>15</b>
<b>9. Submission Deadlines .....</b>	<b>16</b>
<b>10. Grading Criteria .....</b>	<b>16</b>
<b>11. Final Notes .....</b>	<b>17</b>

# Project: Four-Port Switch

## 1. Introduction

This project requires students to develop a complete RTL design, verification environment, and synthesis flow for a simplified four-port packet switch using SystemVerilog and Verilog principles learned in the course. The switch accepts packets on any of its four ports and forwards them to one or more destination ports according to fields encoded in each packet.

The project is divided into three stages:

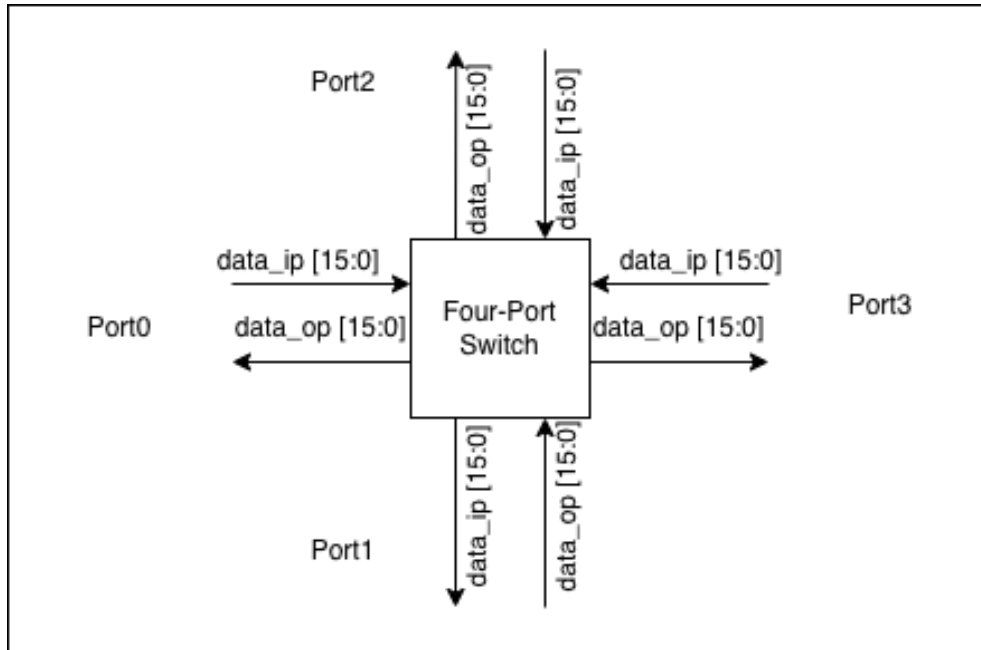
- **Stage A: Design (RTL Implementation)** – Design and implement the RTL for the switch, including basic QA tests to ensure compilation and basic functionality.
- **Stage B: Verification** – Build a SystemVerilog-based verification environment with constrained-random stimulus, modular components, and coverage metrics.
- **Stage C: Synthesis** – Perform synthesis, analyze area/power/frequency results, and validate at gate-level.

This assignment emphasizes RTL design best practices (e.g., FSM, clock/reset handling, modular architecture), verification methodologies (e.g., OOP, constraints, BFM, coverage), and synthesis optimization. It prepares students for full hardware development flows.

### Handling Concurrent Packet Arrivals

The switch must handle simultaneous packet arrivals on multiple inputs in the same clock cycle without drops, duplicates, or corruption. For output contention, use arbitration like Round-Robin for fair transmission; add buffering (e.g., FIFO per output) if needed. Verify via checkers to ensure no drops in multi-port mode.

**Note:** You may not use UVM or any external verification libraries. All implementations must be your own work. Do not modify provided files unless explicitly allowed.



A simplified packet switch design has four ports. The switch receives data packets on a port and transmits the packet to one or more of the four ports depending on the packet data.

## 2. Learning Objectives

By completing this project, you should be able to:

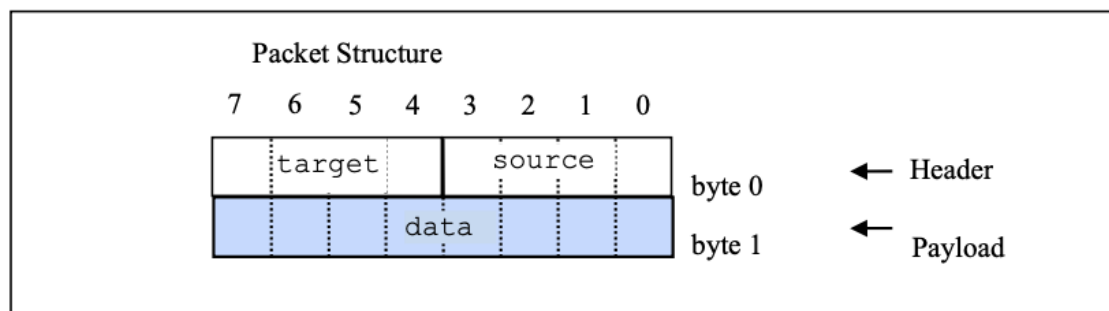
- Design modular RTL using Verilog/SystemVerilog, including FSMs, functions, loops, reg/wire distinctions, parametric coding, and DEFINE macros.
- Implement clock/reset synchronization, completion logic, and proper wiring between modules.
- Develop an object-oriented packet model with randomization and constraints.
- Build reusable verification components (sequencer, driver, monitor, agent, VC) using virtual interfaces.
- Implement checkers to validate DUT behavior.
- Achieve functional coverage and use Bus Functional Models (BFM) for verification.
- Perform synthesis, gate-level simulation, and analyze performance metrics (area, power, frequency, slack).
- Document designs with block diagrams, state diagrams, logical design descriptions, and efficiency analyses.

## 2. Packet Specification

Each packet processed by the switch contains the following fields:

- **source (4 bits, one-hot encoded):** Identifies the port on which the packet entered the switch.
- **target (4 bits, one-hot mask):** Specifies the destination port or ports to which the packet must be delivered.
- **data (8 bits):** The payload carried by the packet.

The switch data packet has the following format:



Where:

1. Source is the address of the port where the packet was sent into the switch. Source has a 4-bit, one-hot encoding as follows:

Port0	Port1	Port2	Port3
4'b0001	4'b0010	4'b0100	4'b1000

2. Target is the 4-bit address of the port(s) to which the packet is being sent (more information below).
3. Data is an 8-bit payload for the packet.

There are three types of packet, based on the value of target.

### 1. **Single-Destination Packet**

A packet forwarded to exactly one port. The destination port must not be identical to the source port.

Port0	Port1	Port2	Port3
4'b0001	4'b0010	4'b0100	4'b1000

### 2. **Multicast Packet**

A packet forwarded to two or three ports. The source port must not appear in the set of destination ports.

- a) Either 2 or 3 bits set in target, representing the multiple destination ports for the packet. For example, a target of 4'b1010 represents a packet to be sent to both port 1 and port 3.
- b) b. Multicast packets are not allowed to be transmitted to the same port from which they originated, i.e., the target cannot contain the same bit set as the source.

### 3. **Broadcast Packet**

A packet forwarded to all ports. All bits in the target field are set.

### 3. Packet Model Development

Students will implement an extensible object-oriented packet class in SystemVerilog. The class must include:

- A string identifying the instance name.
- The fields source, target, and data.
- An enumerated field representing the packet type.
- A constructor that accepts an instance name and a numeric port index, generating the appropriate one-hot source encoding.
- Methods for retrieving the packet type and instance name.
- A configurable print method capable of displaying packet fields in various numeric formats.

As the project progresses, additional requirements will be incorporated:

- Declaring selected fields as randomizable (rand).
- Defining validity constraints for legal packet formation.
- Implementing a static counter to track the number of packet objects created.
- Assigning unique tags for debugging, including optional automatic tag-to-data propagation.
- Creating derived classes representing Single, Multicast, and Broadcast packets, each enforcing its own structural constraints.

## 4. Constrained-Random Stimulus Generation

The packet class must support legal constrained-random generation of packet instances.

Students are required to:

- Define constraints that ensure non-zero targets and prevent illegal overlaps between source and target.
- Establish conditional constraints that enforce the number of destination bits according to the packet category.
- Override base-class constraints within derived classes where necessary.
- Demonstrate the effects of constraint failures and evaluate SystemVerilog randomization behavior.

This component ensures that the environment can generate meaningful and varied test scenarios for the switch.

The project must demonstrate:

- RTL managed by an FSM with clear states (cases) for packet reception, routing, and transmission.
- Use of functions as required.
- At least two types of loops (e.g., for, repeat/generate).
- Proper distinction and use of reg vs. wire.
- Clock/reset handling: Synchronization, initialization, and behavior at state start/end.
- Completion logic and wiring between modules.
- Modular design with separated, standard modules and correct inter-module communication.
- Parametric coding and use of DEFINE macros.

**Provided Files:**

- port\_if.sv – Interface for driving/collecting packets.
- vc\_test.sv – Testbench for single VC.
- switch\_test.sv – Testbench for full DUT.
- packet\_pkg.sv – Package for classes.
- Skeleton files: packet\_data.sv, component\_base.sv, sequencer.sv, driver.sv, monitor.sv, agent.sv, packet\_vc.sv.
- run.f – Simulator file list.

You must implement RTL files (e.g., switch\_port.sv, switch\_4port.sv) if not provided, or extend them as needed.

## 5. Stage A: Design (RTL Implementation)

Implement the RTL for the four-port switch:

- Design the switch to handle packet routing based on source/target.
- Use an FSM to manage packet flow: States for idle, receive, route, transmit, etc.
- Handle clock/reset properly: Edge-sensitive, synchronous reset, initialization of registers.
- Implement completion logic to signal end of processes (e.g., done flags).
- Divide into modules: e.g., input parser, router, output mux per port.
- Use parametric coding (e.g., parameter NUM\_PORTS=4) and DEFINE for constants.
- Include basic QA tests: Compile the code, run simple simulations to verify basic packet forwarding (e.g., single-destination, broadcast).

### Deliverables for Stage A:

- RTL files (Verilog/SystemVerilog).
- Basic testbenches for QA.
- Simulation logs showing compilation success and basic functionality.
- Block diagrams and state diagrams (FSM).
- Logical design description.

## 6. Stage B: Verification Component Architecture

Build a SystemVerilog verification environment:

Students will design a set of reusable verification components following a modular architecture. The hierarchy includes:

### 6.1 Component Base Class

A foundational class that provides:

- Component naming and hierarchy management.
- A parent pointer.
- A method to construct fully qualified hierarchical paths.
- A minimal print method.

### 6.2 Sequencer

A component responsible for producing packets.

It must:

- Select packet types pseudo-randomly.
- Construct and randomize appropriately-typed packet objects.
- Supply packets to the driver upon request.

### 6.3 Driver

A component responsible for transmitting packets to the design through a SystemVerilog virtual interface.

It must:

- Request the next packet from the sequencer.
- Display or log packet information.
- Call interface-level tasks to drive packets into the DUT.

### 6.4 Monitor

A component responsible for observing packets emerging from the switch.

It must:

- Use the interface's sampling tasks to collect received packets.
- Reconstruct and print the observed packet data.

### 6.5 Agent

An encapsulating unit that contains one sequencer, one driver, and one monitor.

It provides component construction and internal connectivity.

## 6.6 Packet Verification Component (Packet\_VC)

A top-level verification component for a single switch port.  
It must:

- Instantiate an agent.
- Provide configuration mechanisms for assigning the appropriate virtual interface and source-port number.
- Launch both driver and monitor processes.

## 6.7 Checkers

To perform verification, you must implement checkers that compare expected behavior against actual DUT outputs. Checkers are essential for validating correctness and cannot be omitted.

- Implement a checker component that:
  - Receives packets from the sequencer (expected packets) and monitors (actual outputs).
  - Compares packet fields (source, target, data) to ensure correct routing: Packets should arrive only at intended target ports, with no drops, duplications, or corruption.
  - Handles all packet types (single, multicast, broadcast).
  - Reports pass/fail for each packet and overall test.
  - Integrates into the agent or VC for automated checking during simulation.
- The checker should use assertions or comparison logic to flag errors, such as mismatched data or incorrect destinations.

Verify correct arbitration in contention scenarios, ensuring fairness (e.g., via Round-Robin) and no packet loss.

Incorporate:

- **BFM (Bus Functional Model):** Use the provided interface as a BFM for abstract transaction-level modeling.
- **Functional Coverage:** Implement functional coverage for packet types, FSM states, source/target combinations, and edge cases.
- **Code Coverage:** Collect line/branch/toggle coverage metrics (optional).
- Test all modules and FSM parts in the testbench.

- Handle clock/reset in verification: Synchronize stimuli, check reset behavior.
- Demonstrate concurrent operation on all four ports using `switch_test.sv`.
- Verify packet routing, including multicast/broadcast, using the checker for automated validation.

## 6.8 Integration with the Switch Interface

Students are provided with a SystemVerilog interface file that models the input and output signals of each switch port. The verification components must be connected to this interface using virtual interfaces. The `Packet_VC` configuration procedure must:

- Assign a virtual interface to the driver and monitor.
- Configure the sequencer with the appropriate source-port index.
- Manage parallel execution of driver and monitor activities.

A simple test module is provided to validate the environment on a single port before integrating with the full switch.

## 6.9 Full DUT Verification Using Four Parallel Components

The final phase of the project requires instantiating the RTL design of the four-port switch and verifying its behavior under concurrent multi-port operation. Students must:

- Instantiate four interface instances, one per port.
- Instantiate four `Packet_VC` components, each configured for a different port/interface pair.
- Start all verification components concurrently.
- Capture packet outputs from all ports through dedicated monitoring processes.
- Confirm that packets are distributed to their destination ports as dictated by their target fields, using the checker for validation.

Demonstrate concurrent multi-port operation with contention cases, confirming no drops using the checker.

The environment should demonstrate correct switching behavior, and students are expected to analyze the simulation outputs to validate end-to-end packet flow.

## 6.10 Expected Outcomes for Verification

Upon completing the verification stage, students will have constructed a fully operational SystemVerilog verification environment featuring:

- A parameterized, constrained-random packet generator.
- A set of reusable OOP-based verification components.
- A structured, multi-agent architecture capable of parallel operation.
- Checkers for automated correctness validation.
- A complete integration with a multi-port switch DUT.
- A thorough functional test demonstrating accurate packet routing behavior.

This stage provides applied experience with SystemVerilog classes, constraints, verification planning, interface-based design, and component-driven stimulus generation, preparing students for advanced verification methodologies such as UVM.

**Optional:** Explore formal verification aspects (e.g., basic assertions for properties like no packet loss). Code coverage analysis.

### **Deliverables for Stage B:**

- Completed class files (including checker implementation).
- Simulation transcripts: Packet generation, driver/monitor activity, DUT outputs, checker reports.
- Functional coverage reports (required); code coverage reports (optional).

## 7. Stage C: Synthesis

- Synthesize the RTL (e.g., using Synopsys DC or equivalent tool).
- Perform gate-level simulation and verification.
- Enable/disable clock gating and compare results.
- Analyze:
  - Maximum frequency (slack analysis).
  - Area/power consumption.
  - Resource utilization (logic elements, memory, interconnects).
  - Runtime performance.

### Deliverables for Stage C:

- Synthesis scripts and reports.
- Gate-level netlist and simulation logs.
- Analysis: Tables/graphs for area/power/frequency, comparisons (e.g., with/without clock gating).

## 8. Overall Deliverables

- All RTL, verification, and synthesis files.
- Simulation/synthesis logs and transcripts.
- Report including:
  - Design approach (RTL, verification, synthesis).
  - Key constraints and FSM details.
  - Architecture diagrams (class hierarchy, block diagrams, state diagrams, data flow).
  - Performance analysis: Efficiency, resource usage, runtime.
  - Issues encountered and resolutions.
  - Results: Graphs, tables, analysis.
- Code must be readable, documented (comments on modules, functions, special handling, cases).

## 9. Submission Deadlines

- **Stage A (Design):** 13/12/2025
- **Stage B (Verification):** 28/12/2025
- **Stage C (Synthesis):** 20/01/2026

## 10. Grading Criteria

The overall grade is divided by stages: Stage A (20%), Stage B (20%), Stage C (60%).

Within each stage, grading is based on the following criteria, aligned with course topics.

### Stage A Grading (20% of Total Grade)

- Correct FSM implementation, clock/reset handling, completion logic, wiring, modular design, case transitions (10%).
- Use of functions, loops (for/repeat/generate), reg/wire distinction, parametric coding with DEFINE (5%).
- Basic QA tests, simulation logs, block diagrams, state diagrams, logical design description (5%).

### Stage B Grading (20% of Total Grade)

- Verification components (sequencer, driver, monitor, agent, VC) and integration (8%).
- Packet model, randomization, constraints, BFM usage (4%).
- Checkers implementation and validation (4%).
- Functional coverage (required), testing all modules/FSM parts, clock/reset handling in TB (3%).
- Simulation transcripts, coverage reports (1%).

### Stage C Grading (60% of Total Grade)

- Synthesis execution, gate-level verification, clock gating comparison, max frequency (slack) analysis (20%).
- Resource utilization: Area/power calculation, memory usage, runtime analysis (15%).
- Correct coding practices across the project: Parametric writing, functions, loops, reg/wire, completion logic, wiring, modular design, full block diagrams, detailed logical design (15%).
- Submission quality: Organized submission, project explanation, results, code comments (modules/functions/special cases), FSM cases explanation, diagrams/logical design description, resource/runtime analysis (10%).

**Bonus (5% Extra Credit)**

Completing all optional features will earn up to 5% bonus points added to the final grade.

**11. Final Notes**

- Follow best practices: Incremental testing, clear documentation.
- Address all course topics (as listed in requirements).
- You can change Makefile.