ALU   Verification Plan

CONTENTS **Page Number**

CHAPTER 1 – DESIGN OVERVIEW

1.1.ALU Introduction:

The arithmetic-logical unit (ALU) is a fundamental combinational unit in a digital circuits which performs different arithmetic and logical operations. The given design focuses on parameterizing the inputs and outputs. The design involves various operations including addition, subtraction, multiplication, bitwise logic, shift of the input operands. The design ensures 2 clock cycle delay for multiplication and 1 clock cycle delay for the rest of the operations.

Verification is a phase in hardware development, which ensures that the design meets its specification and functionalities with all possible combinations. The verification has systemverilog based approach, where concepts like constraints, coverage are used.

1.2 Advantages of ALU:

- **Scalable Design:**
  The ALU is designed for easy scalability, allowing you to begin with a 16-bit version and effortlessly expand to 32, 64, or even 128 bits. This flexibility reduces development time and helps minimize the introduction of new errors.

- **Wide Range of Operations:**
  With support for 11 arithmetic and 14 logic operations, ranging from simple addition to complex rotation functions.

- **Status Flags:**
  Each executed operation generates valuable status flags, including indicators like carry, overflow, greater, less, equal, and error. These flags empower the system to make informed decisions and enhance fault detection.

- **Power Saving:**
  The presence of a clock enable (CE) signal allows you to deactivate the ALU when it is not required, leading to significant power savings, especially in battery-powered devices.

- **Error Detection:**
  Catches invalid operations, especially in rotate cases where bad input patterns can cause problems.

1.3 Disadvantages of ALU:

- **Inflexible Timeout Duration**
  The ALU enforces a fixed waiting period of 16 clock cycles for input data. This rigidity can be a limitation, as it may be too slow for high-speed systems or too fast for slower ones, and cannot be adjusted as needed.

- **Commands Can Be Confusing**
  The use of the same command codes for different functions depending on the mode (e.g., CMD = 0 representing ADD in one mode and AND in another) can lead to confusion and an increased risk of incorrect usage.

- **One Error Signal for Everything**
  When an error occurs, the ALU provides only a general error signal, which does not specify whether the problem was due to a timeout, an invalid command, or incorrect input. This lack of detail complicates troubleshooting efforts.

- **Rotate Operation Is Tricky**
  Rotate left and rotate right instructions require operand B to meet strict formatting. It's easy to mess this up in software, leading to errors.

1.4 Use cases of ALU:

- **Arithmetic Operations**
  ▪ Executes fundamental mathematical tasks such as addition, subtraction, and multiplication.

- **Logical Operations**
  ▪ Executes logical operations like AND, OR, XOR, NO.
  ▪ Useful for comparing values, masking bits, and making decisions in circuits.

- **Bit Shifting and Rotation**
  ▪ Shifting of bits left or right, rotates bits for specific applications.
  ▪ Commonly in encryption, encoding-decoding, and data manipulation.

- **Comparisons**
  ▪ Compares two numbers to check greater than, less than, or equal.
  ▪ Useful for loops and conditional branching in software and hardware.

- **Checksum and CRC Computation**
  ▪ Helps compute checksums and cyclic redundancy checks for error detection in communication systems.

- **Control Signal Generation**
  ▪ Generates control signals in finite state machines (FSMs) based on comparison results.

- **Overflow and Carry Detection**
  ▪ Detects arithmetic overflow or carry during operations.

- **Executing CPU Instructions**
  ▪ The ALU acts as the main part of the execution stage of a CPU.It carries out actual operations for instructions.

1.5 Project Overview of ALU:

This project is about designing a flexible and powerful Arithmetic Logic Unit (ALU) that can be used in many digital systems like processors and embedded controllers. The ALU is parameterized, which means it can work with different data sizes like 16, 32, 64, or 128 bits. This makes it useful for both simple and high-performance systems.

The ALU takes two inputs (OPA and OPB) and works in two modes: arithmetic mode (MODE=1) and logical mode (MODE=0). Each mode supports 11 and 14 operations respectively, such as addition, subtraction, multiplication, and bitwise logic and rotate operations. A 4-bit command signal selects the operation.

To handle real system needs, the ALU has some advanced features. The INP_VALID signal is used to manage inputs that may not come at the same time. There is also a timeout mechanism so the ALU doesn’t wait forever for missing inputs. This helps improve reliability.

The ALU provides status outputs like overflow, carry, and comparison results (greater, less, equal). These help the control unit or CPU make decisions like branching or error handling.

The design also includes error detection, especially for tricky operations like variable rotation. If something goes wrong, the ALU sets an error flag instead of continuing silently. This makes debugging easier.

1.6 Design Features:

- **Uses Asynchronous Reset**
  The ALU resets instantly using an asynchronous reset signal. It works on the rising edge of the clock but can be reset at any time. This is helpful for system startup or emergency reset conditions.

- **Flexible Bit-Width**
  Uses parameters to set data width (16, 32, 64, 128 bits, etc.), so it can be easily adapted without changing the source code.

- **Smart Operand Control**
  A 2-bit INP_VALID signal shows whether zero, one, or both inputs are available. This is useful when inputs arrive at different times, such as in pipelined or asynchronous systems.

- **Advanced Rotate Operations**
  The ALU supports rotating bits left or right based on operand B. It also includes error checking for invalid rotate values, making the system more robust.

- **Rich Comparison Output**
  The ALU provides greater (G), less (L), and equal (E) signals all at once. This allows the control logic to make faster decisions, such as branching or comparisons.

- **Built-in Overflow Detection**
  Automatically detects overflow in arithmetic operations, which helps to avoid errors in calculations.

- **Power Saving Option**
  A clock enable(CE) signal is included to disable the ALU when not in use. This saves power, which is important in battery-powered or low-energy systems.

1.7 Design Limitation:

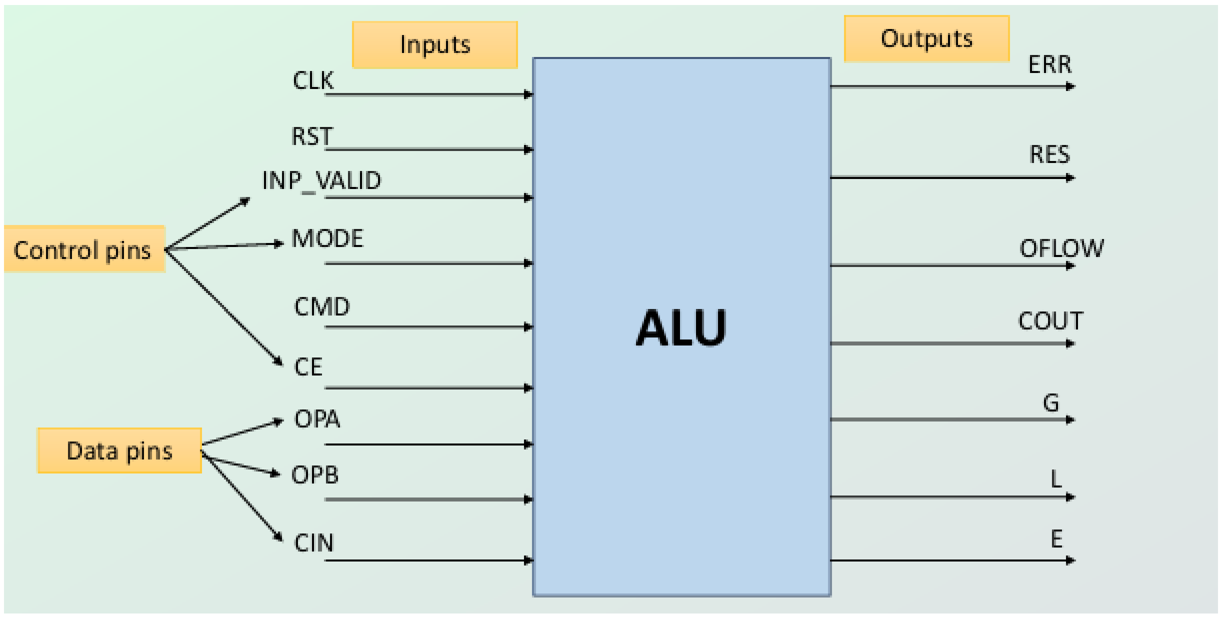
- **Fixed Timeout**
  The timeout is hardcoded to 16 clock cycles. If a system needs a different wait time, the ALU code must be modified.

- **Mode-Based Command Confusion**
  The same command value does different things in arithmetic and logical modes. This can cause confusion or bugs in control software.

- **Single Error Signal**
  Only one ERR signal is used for all errors, so you can't tell if it was a timeout, invalid command, or input problem.

- **Fixed Operand Priority**
  When a timeout happens, the ALU always uses the most recent input operand. This may not match the expected behavior in some systems.

1.8 Design diagram with interface signals:



Pin description: The design has several input and output pins. Below table gives the description of the pins.

Pin name	Description
INPUT PORTS	
CLK	The clock signal on which the design performs function during the positive edge.
RESET	ALU is designed with the asynchronous active high reset signal
CE	Clock enable is active high signal
MODE	It is a 1 bit signal. Arithmetic (MODE=1) or Logical (MODE=0) operations are performed, based on this signal
INP_VALID	It is a 2-bit input valid signal to check the validity of the input operands
CMD	CMD is the command input, which tells which operation has to be performed, depending on, in which MODE it is present
OPA	Parameterized input operand
OPB	Parameterized input operand
CIN	1 bit input signal
OUTPUT PORTS	
RESULT	Parameterized output
COUT	1 bit carry-out signal used in addition/subtraction
OFLOW	1 bit overflow flag used in addition/subtraction
ERR	1 bit error flag is raised to indicate errors
G	1 bit output raised if OPA is greater than OPB
L	1 bit output raised if OPA is lesser than OPB
E	1 bit output raised if OPA and OPB are equal

CHAPTER 2 - Verification Architecture

2.1 Verification Architecture

Fig.1 is the general testbench architecture architecture that is followed while writing systemverilog tesbench.

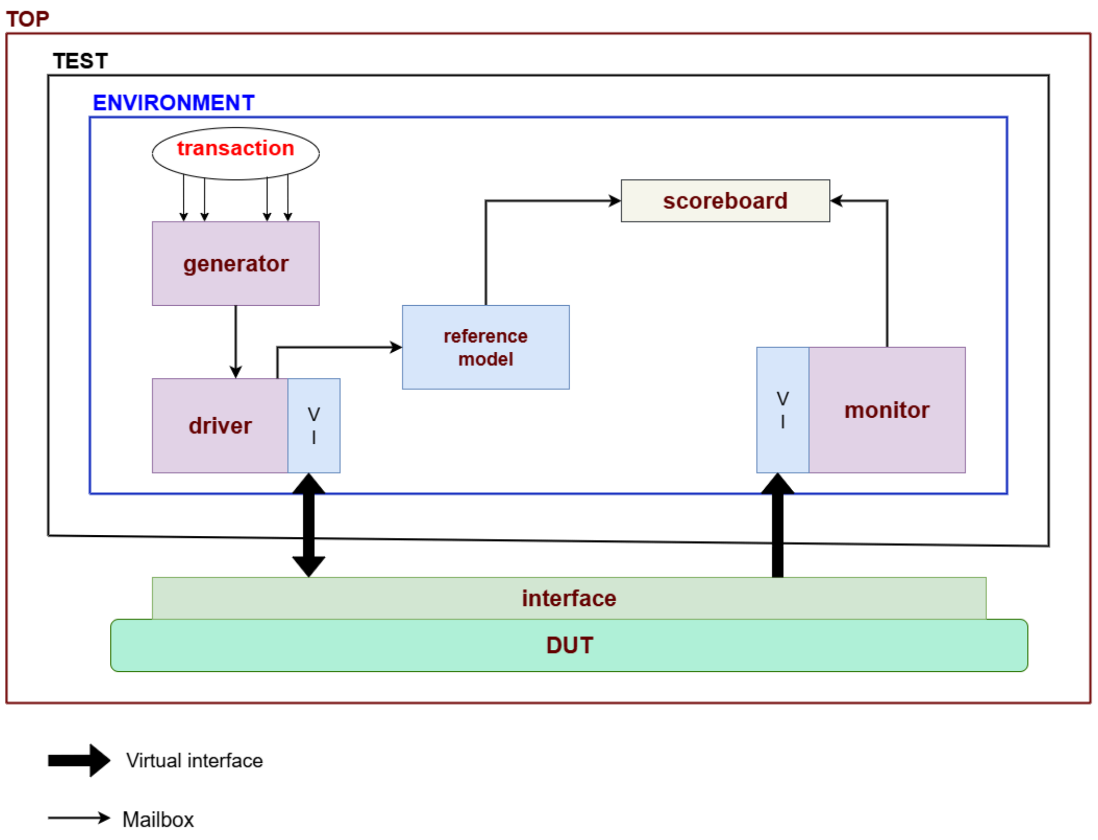


Fig.1 : General testbench architecture

2.2 Verification Architecture for ALU

- Fig.2 is the testbench architecture of ALU.
- The upper most module is TOP(alu_top), which has interface, DUT(design under test/verification) and TEST.
- TEST(alu_test) has ENVIRONMENT(alu_env).
- ENVIRONMENT has transaction, generator(alu_generator), driver, monitor, reference model, scoreboard.
- Driver and monitor uses virtual interface to interact with the design.

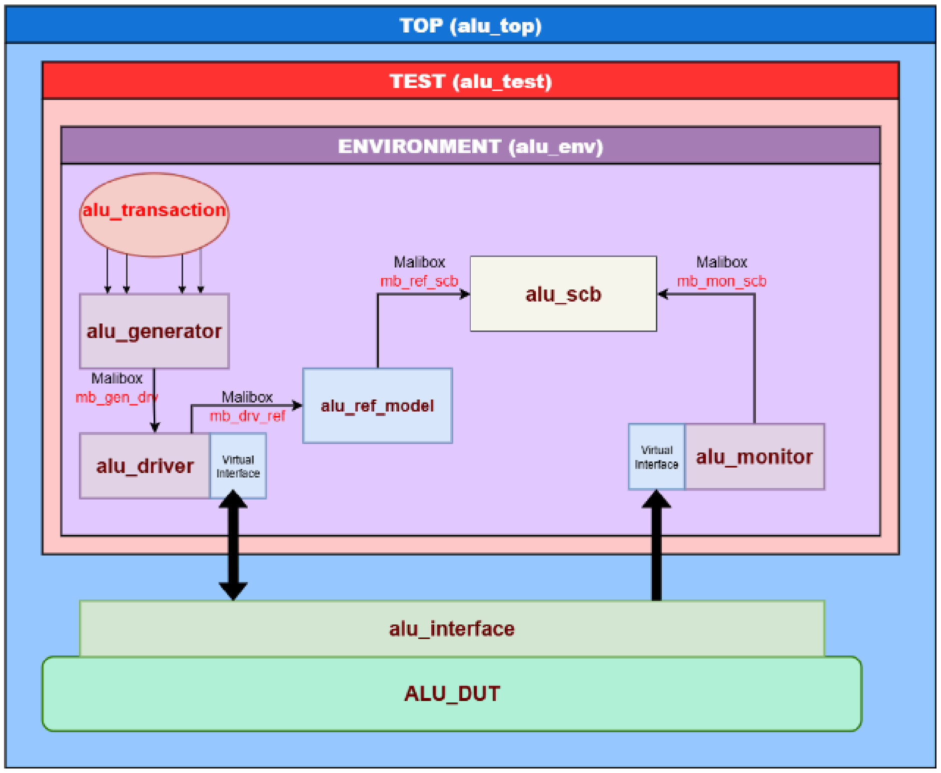


Fig.2 : ALU testbench architecture

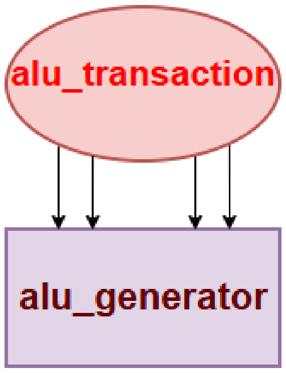
- Mailboxes were used for the transfer of data between the different blocks.
  mb_gen_drv : mailbox between generator and driver
  mb_drv_ref : mailbox between driver and mailbox between
  mb_ref_scb : mailbox between reference model and scoreboard
  mb_mon_scb : mailbox between monitor and scoreboard

2.3 FLOW CHART OF SV COMPONENTS :

The below figure shows the flow chart that the testbench follows.
Generator generates the random test cases. Driver drives the test cases to both DUT and reference model. Monitor captures the response from DUT and send it to scoreboard. Reference model sends expected result to scoreboard. Scoreboard compares both the data from reference model and monitor, and emirates a report.

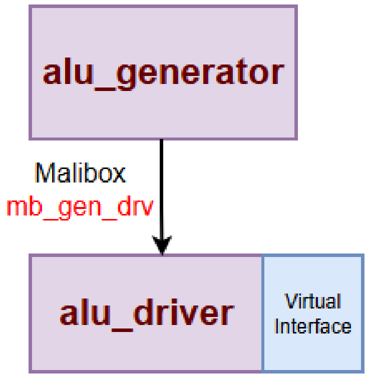
2.3.1 TRANSACTION COMPONENT

The transaction class serves as the fundamental data container that standardizes communication across all testbench components,which contains all the input and output ports. This contains constraints.



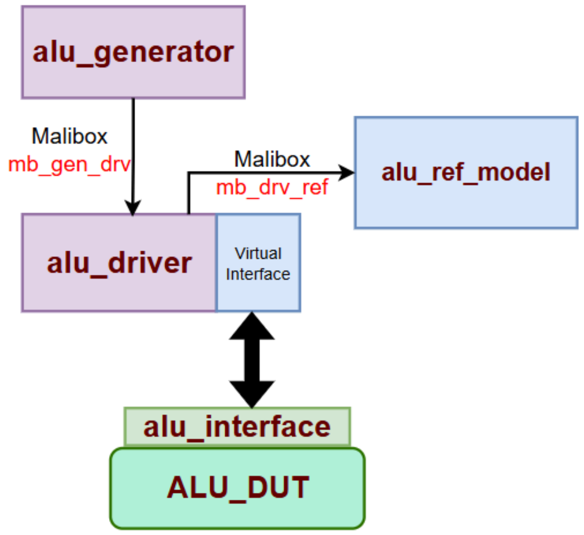
2.3.2 GENERATOR COMPONENT

The generator contains transaction class handle, which helps to generate randomized data. Generated randomized data is send it to the driver through mailbox.



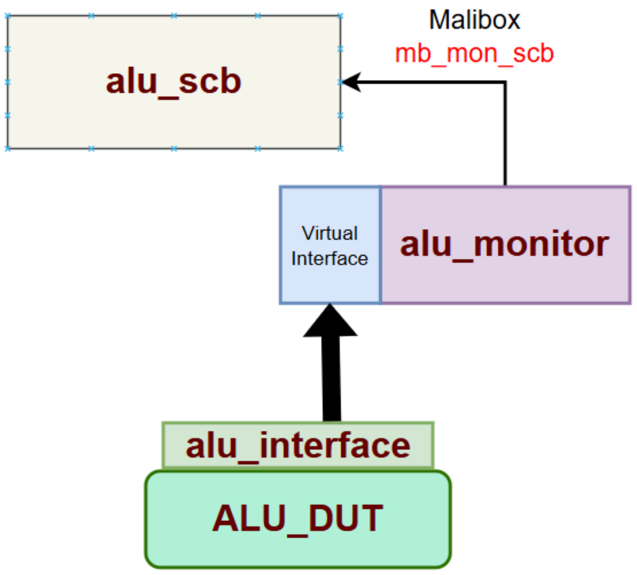
2.3.3 DRIVER COMPONENT

The driver collects the data from the generator and drives the data to DUT via virtual interface and sends the same to the reference model through the mailbox.



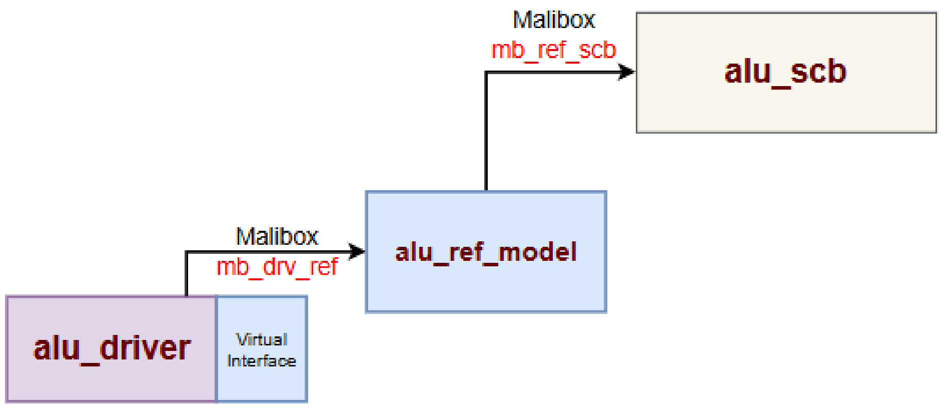
2.3.4 MONITOR COMPONENT

The monitor captures the response immediately from the DUT via virtual interface, and send it to the scoreboard through the mailbox.



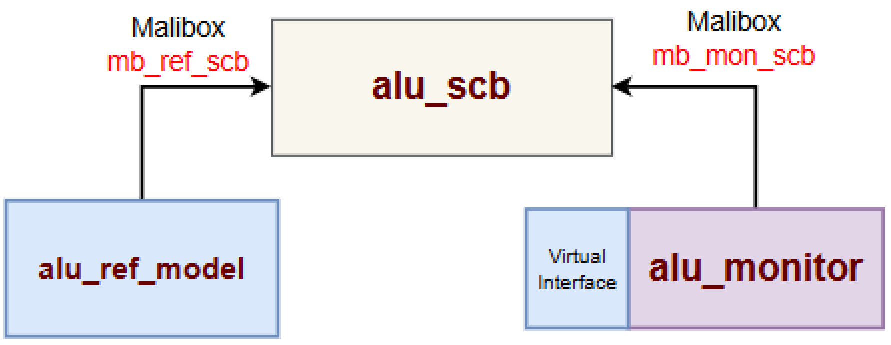
2.3.5 REFERENCE MODEL COMPONENT

The reference model has similar functionalities of the DUT, which predicts the output of the DUT. It receives the data from the driver through mailbox. The reference model gives expected result and send it to scoreboard through another mailbox.



2.3.4 SCOREBOARD COMPONENT

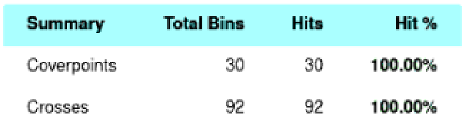
The scoreboard receives the expected data from reference model and the actual data from the monitor. It compares the exact data and expected data. If both are matched, then test case is passed, else test case is failed.



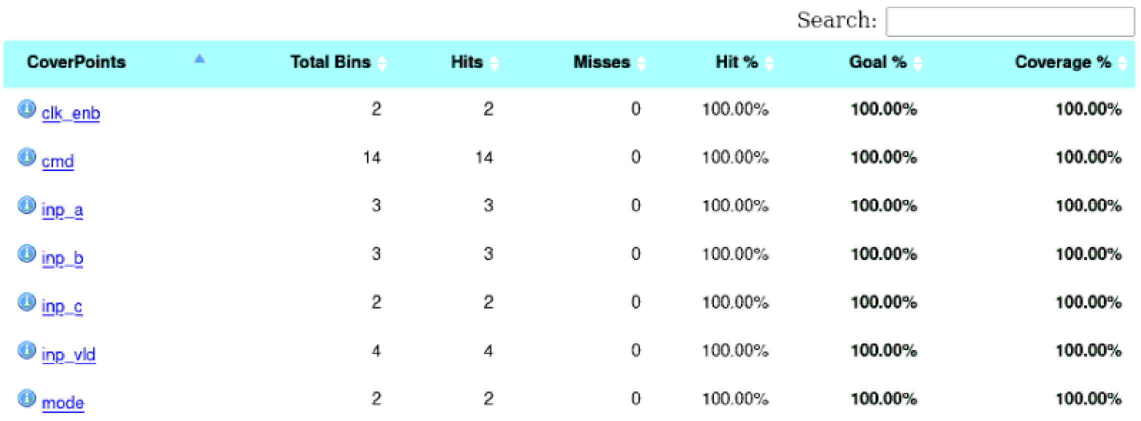
CHAPTER 3 - COVERAGE REPORT

3.1 Input coverage:

drv_cg

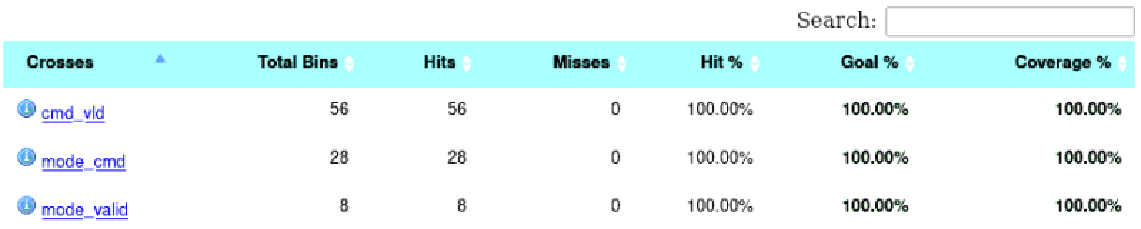


Summary	Total Bins	Hits	Hit %
Coverpoints	30	30	100.00%
Crosses	92	92	100.00%



Search:

CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
clk_enb	2	2	0	100.00%	100.00%	100.00%
cmd	14	14	0	100.00%	100.00%	100.00%
inp_a	3	3	0	100.00%	100.00%	100.00%
inp_b	3	3	0	100.00%	100.00%	100.00%
inp_c	2	2	0	100.00%	100.00%	100.00%
inp_vld	4	4	0	100.00%	100.00%	100.00%
mode	2	2	0	100.00%	100.00%	100.00%

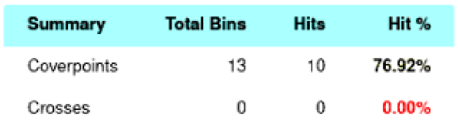


Search:

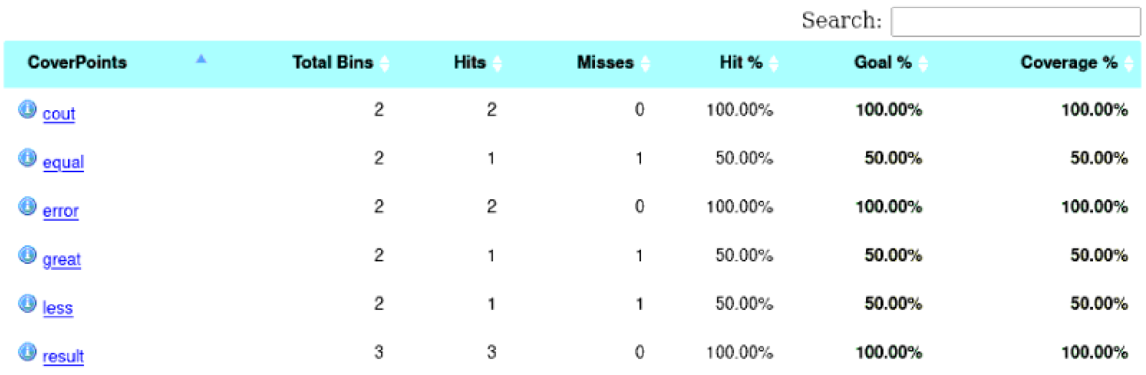
Crosses	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
cmd_vld	56	56	0	100.00%	100.00%	100.00%
mode_cmd	28	28	0	100.00%	100.00%	100.00%
mode_valid	8	8	0	100.00%	100.00%	100.00%

3.2 Output coverage:

mon_cg



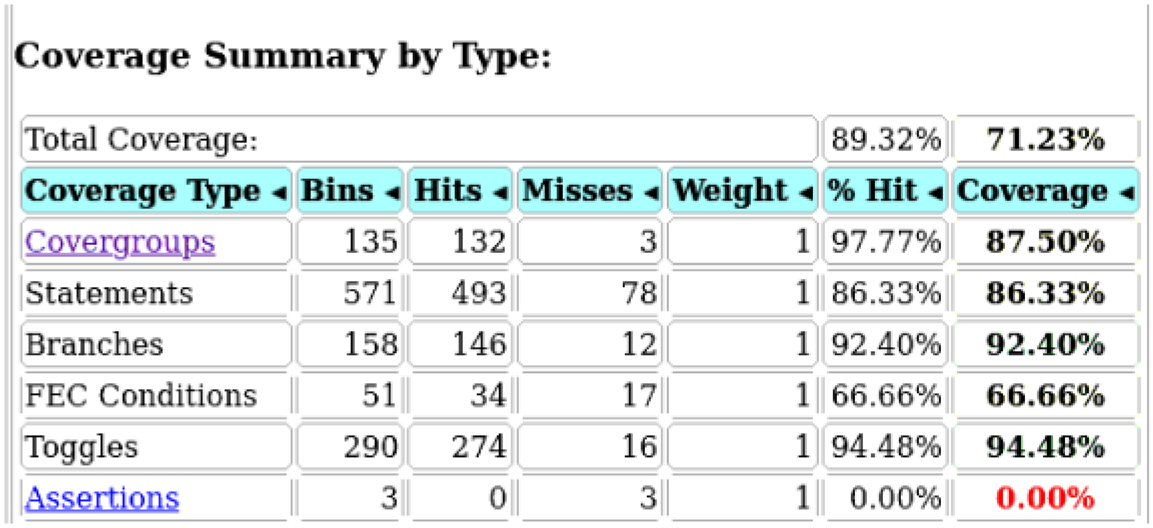
Summary	Total Bins	Hits	Hit %
Coverpoints	13	10	76.92%
Crosses	0	0	0.00%



Search:

CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
cout	2	2	0	100.00%	100.00%	100.00%
equal	2	1	1	50.00%	50.00%	50.00%
error	2	2	0	100.00%	100.00%	100.00%
great	2	1	1	50.00%	50.00%	50.00%
less	2	1	1	50.00%	50.00%	50.00%
result	3	3	0	100.00%	100.00%	100.00%

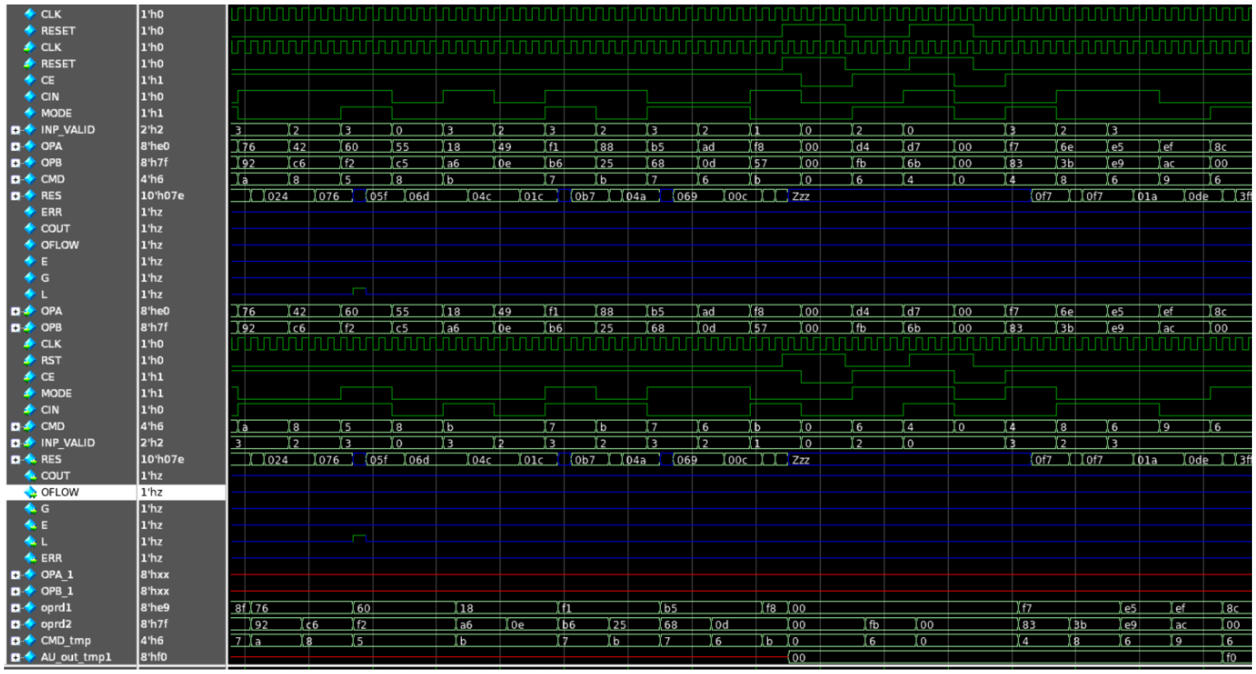
3.3 Overall coverage



Coverage Summary by Type:

Total Coverage:					89.32%	**71.23%**
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Covergroups	135	132	3	1	97.77%	**87.50%**
Statements	571	493	78	1	86.33%	**86.33%**
Branches	158	146	12	1	92.40%	**92.40%**
FEC Conditions	51	34	17	1	66.66%	**66.66%**
Toggles	290	274	16	1	94.48%	**94.48%**
Assertions	3	0	3	1	0.00%	**0.00%**

Waveform


CLK 1'h0
RESET 1'h0
CLK 1'h0
RESET 1'h0
CE 1'h1
CIN 1'h0
MODE 1'h1
INP_VALID 2'h2
OPA 8'he0
OPB 8'h7f
CMD 4'h6
RES 10'h07e
ERR 1'hz
COUT 1'hz
OFLOW 1'hz
E 1'hz
G 1'hz
L 1'hz
OPA 8'he0
OPB 8'h7f
CLK 1'h0
RST 1'h0
CE 1'h1
MODE 1'h1
CIN 1'h0
CMD 4'h6
INP_VALID 2'h2
RES 10'h07e
COUT 1'hz
OFLOW 1'hz
G 1'hz
E 1'hz
L 1'hz
ERR 1'hz
OPA_1 8'hxx
OPB_1 8'hxx
oprd1 8'he9
oprd2 8'h7f
CMD_tmp 4'h6
AU_out_tmp1 8'hf0