Lab Report #06

Mina Ibrahim Henien 49-1743

This report includes:

- Introduction for Verification and System Verilog
- ALU Task
  - Methodology and Implementation
  - Output
- Conclusion
- Appendix (codes)

Introduction

Verification is the process of assuring correct functionality for the design against all possible test cases.

It is essential to assure that the design will meet all the specifications and will not lead to unexpected behaviour.

Types of verification:

1] Functional Verification: It is the process of demonstrating the functional correctness of a design with respect to the design specifications. [Static Verification – Functional Stimulation – FPGA Prototyping – Emulation - UVM]

2] Timing Verification consists of validating the path delays and checking the clock pulses to meet the design goals [Static Timing Analysis – Timing Simulation] + [Hold and setup time]

Dynamic – driving the inputs of the DUT (design under test) and checking if the value on the outputs and selected points is as expected. Done from unit level to full system level, from RTL phase through gate-level to post silicon.

Formal – mathematical way to analyse the design and check if the behaviour is as expected.

Static – tools that verify the coding style.

Timing – STA (Static timing analysis):

Verifying that all paths meet the timing requirements (setup/hold) at all relevant corners (PVT) and modes (test modes).

Setup: Verify that the data at the input of each flip-flop is stable at least for "setup time" before the rising edge of the clock.

If the data cannot be stable before that time, wrong data might be sampled by the flip-flop and lead to logic failure.

Hold: Verify that the data at the input of each flip-flop is stable at least for "hold time" after the rising edge of the clock (the data from the cones is not propagating too fast).

If expansion time from source flop to destination flop is less than the hold time, the value might affect the next flop in the logic cone one cycle earlier.

PVT: The expansion time of the data is different in each PVT mode. STA must be checked in each mode.

For example: At FF corner, 125C degree, 10% more than nominal voltage path A is the longest in flip-flop X cone, but in SS corner, -40c, 10% less than nominal voltage, path B is the longest.

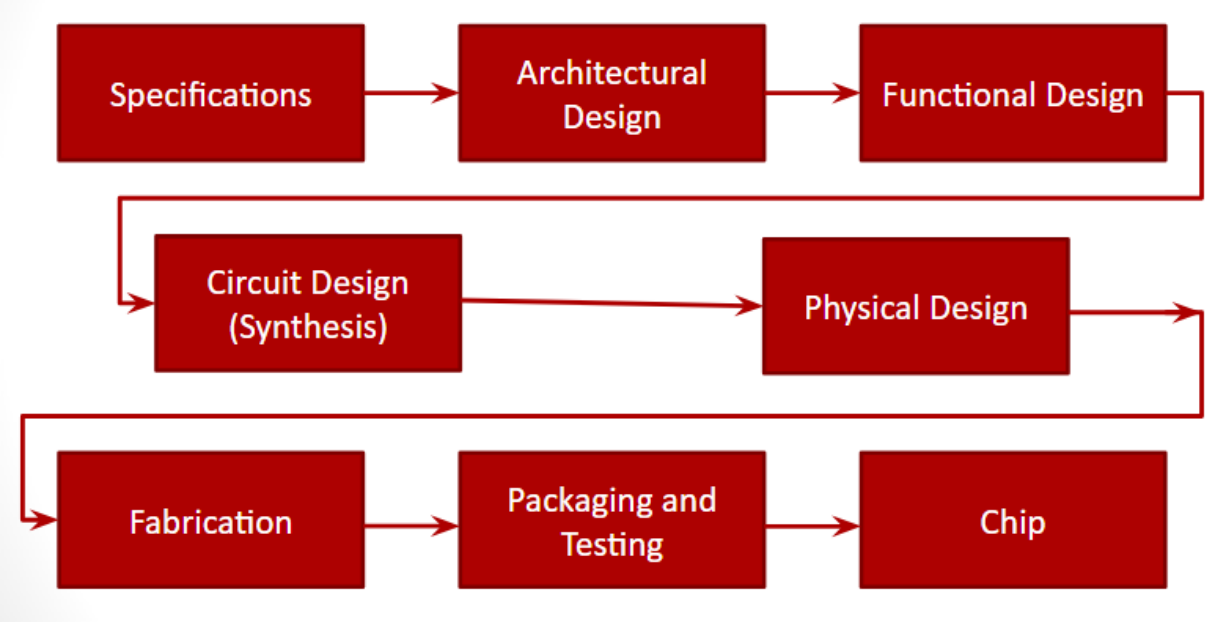

Figure 1: VLSI design flow.

- HVL: Hardware Verification Language (i.e. System Verilog)
- Distinct features for HVL:
  - Constrained random stimulus generation
  - Functional coverage
  - Object oriented programming
  - Support for HDL types (i.e. four state values 0, 1, Z, X)

Testing techniques:

- Direct Testing
  - An incremental approach that tests related features at a time
- Constrained random testing
  - Random test scenarios until full coverage is achieved

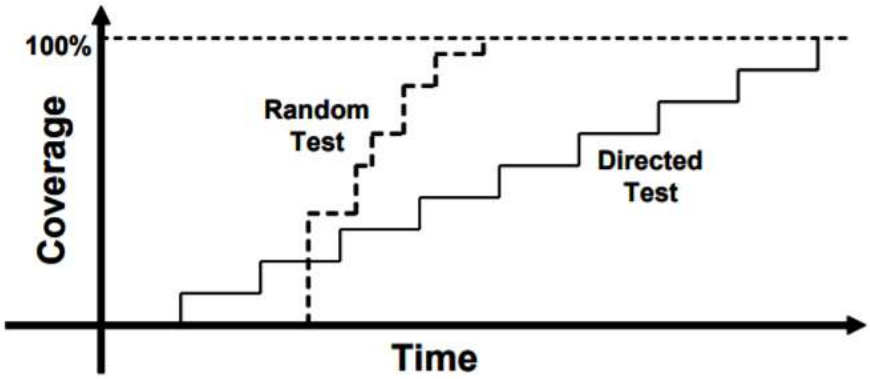

Figure 2: Testing techniques.

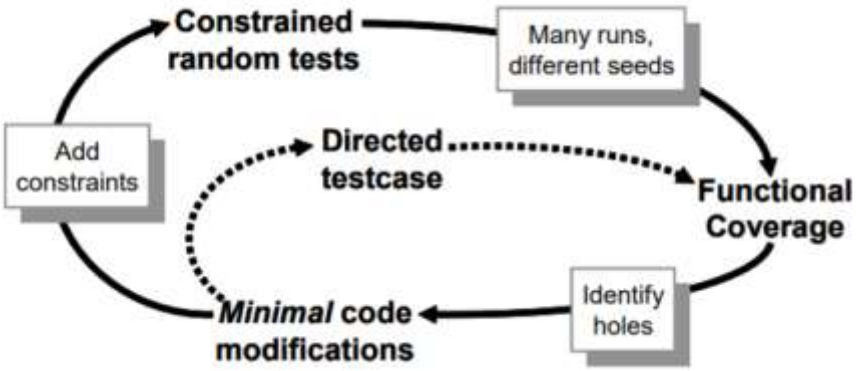

Figure 3: Testing cycle.

Arithmetic and Logic Unit (ALU):-

A. Methodology and Implementation

The ALU is implemented using a SystemVerilog tasks. For each operation, a task is implemented, and there is another task calculates the result depending on two input integers “32 bits” operands and 2 bits opcode. Case statement is added to check the opcode in which “00” means addition, “01” means subtraction, “10” means multiplication and “11” means division. Inside an initial begin block various test cases are used to check the functionality of the ALU tasks. The output is a longint variable “64 bits”. Corners, such as maximum positive, minimum negative and zero are tested as well.

B. Output

```
VSIM 4> run -all
# Opcode = 0,A = 10 , B = 5 , output = 15
# Opcode = 1,A = 10 , B = 5 , output = 5
# Opcode = 2,A = 10 , B = 5 , output = 50
# Opcode = 3,A = 10 , B = 5 , output = 2
# Opcode = 0,A = 180 , B = 15 , output = 195
# Opcode = 3,A = 200 , B = 5 , output = 40
# Opcode = 2,A = 30 , B = 6 , output = 180
# Opcode = 1,A = 80 , B = 50 , output = 30
# Opcode = 2,A = 25 , B = 2 , output = 50
# Opcode = 0,A = 40 , B = 60 , output = 100
# Opcode = 1,A = 90 , B = 40 , output = 50
# Opcode = 0,A = -2147483648 , B = 20000 , output = -2147463648
# Opcode = 1,A = 2147483647 , B = 12 , output = 2147483635
# Opcode = 2,A = 0 , B = 40 , output = 0

VSIM 5>
```

Conclusion

In conclusion, this report includes an ALU written using SystemVerilog tasks and the output is simulated for various testcases including corners using Questasim.

Appendix

A. Task #01: System Verilog code

```
module ALU;


  int A;

  int B;

  logic[1:0] opcode;

  longint C;

  int maxpos = 2147483647;

  int minneg = -2147483648;

  int zero = 0;

  localparam Add = 2'b00;

  localparam Sub = 2'b01;

  localparam Mult = 2'b10;

  localparam Div  = 2'b11;
```

```
task addition;
  C = A + B;
endtask

task subtraction;
  C = A - B;
endtask

task multiplication;
  C = A * B;
endtask

task division;
  if(B != 0)
    C = A / B;
  else
    C = 0;
endtask

task calculate_result;
  case (opcode)
```

```
    Add: addition;
    Sub: subtraction;
     Mult: multiplication;
    Div: division;
    default: addition;
  endcase
endtask

initial begin

  A = 10;
  B = 5;
  opcode = 0;
  calculate_result;
  $display("Opcode = %0d,A = %0d , B = %0d , output = %0d", opcode,A , B ,C);

  #2;
  A = 10;
  B = 5;
  opcode = 1;
  calculate_result;
  $display("Opcode = %0d,A = %0d , B = %0d , output = %0d", opcode,A , B ,C);
```

```
#2;
A = 10;
B = 5;
opcode = 2;
calculate_result;
$display("Opcode = %0d,A = %0d , B = %0d , output = %0d", opcode,A , B ,C);

#2;
A = 10;
B = 5;
opcode = 3;
calculate_result;
$display("Opcode = %0d,A = %0d , B = %0d , output = %0d", opcode,A , B ,C);

#2;
A = 180;
B = 15;
opcode = 0;
calculate_result;
$display("Opcode = %0d,A = %0d , B = %0d , output = %0d", opcode,A , B ,C);
```

```
#2;

A = 200;

B = 5;

opcode = 3;

calculate_result;

$display("Opcode = %0d,A = %0d , B = %0d , output = %0d", opcode,A , B ,C);


#2;

A = 30;

B = 6;

opcode = 2;

calculate_result;

$display("Opcode = %0d,A = %0d , B = %0d , output = %0d", opcode,A , B ,C);


#2;

A = 80;

B = 50;

opcode = 1;

calculate_result;

$display("Opcode = %0d,A = %0d , B = %0d , output = %0d", opcode,A , B ,C);


#2;
```

```
A = 25;

B = 2;

opcode = 2;

calculate_result;

$display("Opcode = %0d,A = %0d , B = %0d , output = %0d", opcode,A , B ,C);


#2;

A = 40;

B = 60;

opcode = 0;

calculate_result;

$display("Opcode = %0d,A = %0d , B = %0d , output = %0d", opcode,A , B ,C);


#2;

A = 90;

B = 40;

opcode = 1;

calculate_result;

$display("Opcode = %0d,A = %0d , B = %0d , output = %0d", opcode,A , B ,C);


// Testing corners
```

```
    #2;
    A = minneg;
    B = 20000;
    opcode = 0;
    calculate_result;
    $display("Opcode = %0d,A = %0d , B = %0d , output = %0d", opcode,A , B ,C);

    #2;
    A = maxpos;
    B = 12;
    opcode = 1;
    calculate_result;
    $display("Opcode = %0d,A = %0d , B = %0d , output = %0d", opcode,A , B ,C);

    #2;
    A = zero;
    B = 40;
    opcode = 2;
    calculate_result;
    $display("Opcode = %0d,A = %0d , B = %0d , output = %0d", opcode,A , B ,C);
  end
endmodule
```