# CSE 312: Electronic Design Automation
# Project 2: Task 3

# CESS
# Fall 2022


**Submitted by:**

| | |
|---|---|
| Mariam Sameh Elkhatib | 20P1599 |
| Laila Ihab Mohamed | 20P1005 |
| Seif allah Mohamed Elalfy | 20P5640 |
| Omar Ali Sobhy | 20P2084 |
| Habiba Yasser Abdel Razek | 20P9595 |
| Adham Yasser Kassab | 20P9705 |
| Khaled Mohamed Hassan | 20P9978 |
| Sarah Medhat Sayed | 20P8579 |

# Table of Contents

# Task 3: Automatic Test Bench Generator

## Introduction:

RTL code needs to frequently be tested using a testbench by hardware engineers using Verilog. Writing a testbench skeleton using an entity declaration is a common text manipulation technique.

There must be a testbench for each design unit in a project. Hours can be saved on each project by automatically creating testbench skeletons. In the future, though, that time may be cut down to seconds with a little Perl programming.

**This Test Bench Generator should be constructed from:**

o Verilog Parser, that also extract control flow of your design (if/else, branches, cases,…)

o Stimulus Generator (Sequencer, Driver)

o TB Writer: TB Module, Initial Block for Initialization, Clock Generation, Instantiation , Tests Generator, Monitor

Three Verilog designs have been implemented and tested through the test bench generator.

## Design 1:

```
module fulladder (

input wire X,
input wire Y,
input wire Ci,
output wire S,
output wire Co

);

assign S = X ^ Y ^ Ci;
assign Co = (X&Y) | (Ci&X) | (Ci&Y);

endmodule
```

## Test Bench generated:

```
module fulladder_tb ();
reg X;
reg Y;
reg Ci;

wire S;
wire Co;



initial
    begin
    $monitor("Ci = %1b", Ci);
    end

initial
    begin
    $dumpfile("fulladder.vcd");
    $dumpvars ;
```

```
    //testing condtional statements


    //directed test cases
    X = 1'b1;
    Y = 1'b0;
    Ci = 1'b0;

    #20

    X = 1'b1;
    Y = 1'b1;
    Ci = 1'b0;

    #20

    X = 1'b1;
    Y = 1'b0;
    Ci = 1'b1;
```

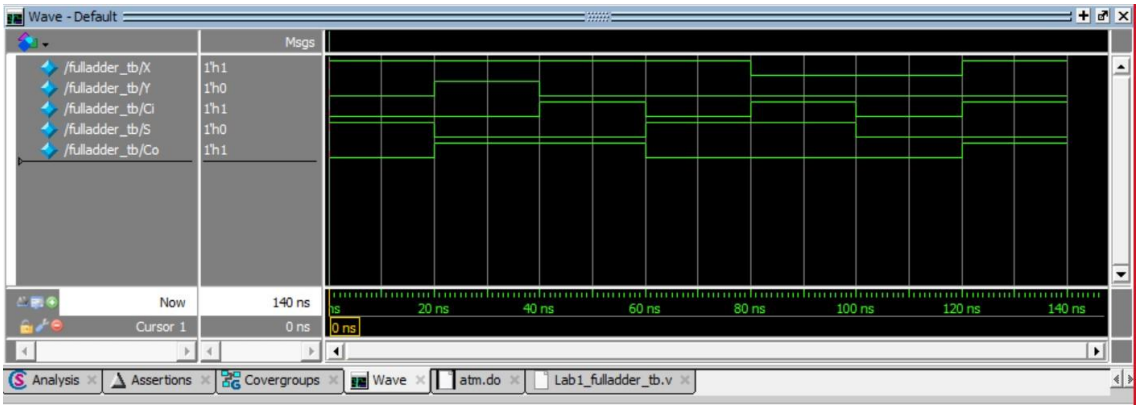
```
    #20

    //randomized test cases
    X = $urandom%2;
    Y = $urandom%2;
    Ci = $urandom%2;

    #20

    X = $urandom%2;
    Y = $urandom%2;
    Ci = $urandom%2;

    #20

    X = $urandom%2;
    Y = $urandom%2;
    Ci = $urandom%2;

    #20
```

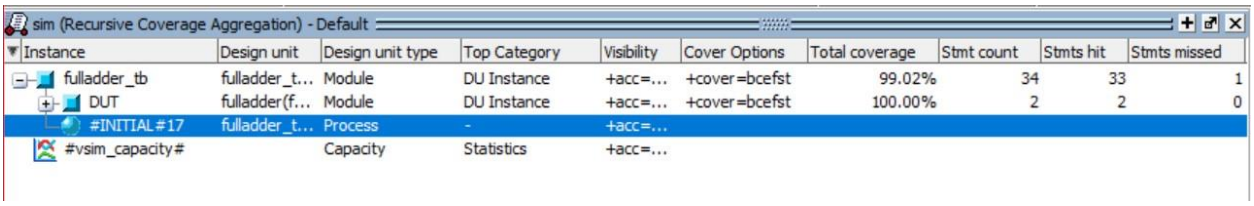
```
    X = $urandom%2;
    Y = $urandom%2;
    Ci = $urandom%2;

    #20



    $finish;
    end

    fulladder DUT (
        .X(X),
        .Y(Y),
        .Ci(Ci),
        .S(S),
        .Co(Co)
    );
endmodule
```

**The Test bench generator divides the design into:**

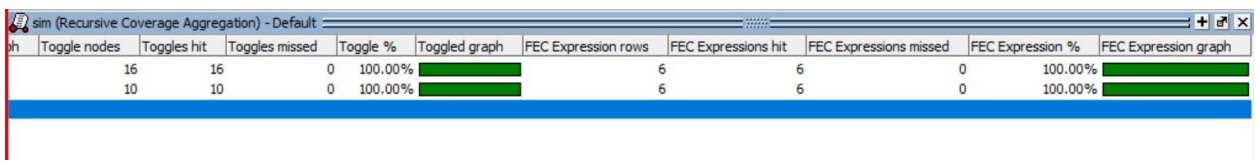
1) **Verilog parser:** identifies the module name of the design (fulladder), input output ports of the module, registers and wires: input wires X,Y and Ci as registers to hold values, outputs are defined as wires(S, Co)
2) **Stimulus generator:** generates the randomize test cases by using $urandom which must know the max number the register can hold to generate number between 0 and this number, it also generates corner test cases to target certain conditions.
3) **TB writer:** identifies the initial blocks It also generates monitor commands and instantiation of input output ports.

# Coverage report and waveforms:




| Instance | Design unit | Design unit type | Top Category | Visibility | Cover Options | Total coverage | Stmt count | Stmts hit | Stmts missed |
|---|---|---|---|---|---|---|---|---|---|
| fulladder_tb | fulladder_t... | Module | DU Instance | +acc=... | +cover=bcefst | 99.02% | 34 | 33 | 1 |
| DUT | fulladder(f... | Module | DU Instance | +acc=... | +cover=bcefst | 100.00% | 2 | 2 | 0 |
| #INITIAL#17 | fulladder_t... | Process | - | +acc=... | | | | | |
| #vsim_capacity# | | Capacity | Statistics | +acc=... | | | | | |




| Toggle nodes | Toggles hit | Toggles missed | Toggle % | Toggled graph | FEC Expression rows | FEC Expressions hit | FEC Expressions missed | FEC Expression % | FEC Expression graph |
|---|---|---|---|---|---|---|---|---|---|
| 16 | 16 | 0 | 100.00% | | 6 | 6 | 0 | 100.00% | |
| 10 | 10 | 0 | 100.00% | | 6 | 6 | 0 | 100.00% | |

## Design 2:

```
module shifter (
        input                           clk,
        input                           load,
        input                           right,
        input                           left,
        input                   [4:0]   in_value,
        output  reg             [4:0]   value
);
reg [4:0] internal_reg ;
always @ (posedge clk)
 begin
        if (load)
         begin
                internal_reg <= in_value ;
         value <= internal_reg;
         end
        else if (right)
         begin
                internal_reg <= internal_reg >> 1 ;
                value <= internal_reg ;
         end
        else if (left)
         begin
                internal_reg <= internal_reg << 1 ;
    value <= internal_reg ;
         end
    else value <= internal_reg;
 end



endmodule
```

## Test Bench generated:

```
module shifter_tb ();
reg clk;
reg load;
reg right;
reg left;
reg [4:0] in_value;

wire [4:0] value;


initial
    begin
    clk = 1'b1;
    repeat(1000)
        begin
        #10
        clk = ~clk;
        end
    end
```

```
initial
    begin
    $monitor("in_value = %5b", in_value);
    end

initial
    begin
    $dumpfile("shifter.vcd");
    $dumpvars ;

    //testing condtional statements


    //directed test cases
    load = 1'b1;
    right = 1'b1;
    left = 1'b0;
    in_value = 5'b00111;
```

```
    #20
    load = 1'b0;
    right = 1'b0;
    left = 1'b0;
    in_value = 5'b10100;

    #20

    load = 1'b0;
    right = 1'b0;
    left = 1'b1;
    in_value = 5'b11001;

    #20

    //randomized test cases
    load = $urandom%2;
    right = $urandom%2;
    left = $urandom%2;
    in_value = $urandom%33;
```

```
    #20

    load = $urandom%2;
    right = $urandom%2;
    left = $urandom%2;
    in_value = $urandom%33;

    #20

    load = $urandom%2;
    right = $urandom%2;
    left = $urandom%2;
    in_value = $urandom%33;

    #20

    load = $urandom%2;
    right = $urandom%2;
    left = $urandom%2;
    in_value = $urandom%33;
```

```
    #20



    $finish;
    end

    shifter DUT (
        .clk(clk),
        .load(load),
        .right(right),
        .left(left),
        .in_value(in_value),
        .value(value)
    );
endmodule
```

**The Test bench generator divides the design into:**

1) **Verilog parser:** since the design is made up of multiple pf if statements, the parser extracts and controls the flow of these conditions, it also identifies the module name shifter(), input outputs ports and their registers and wires.
2) **Stimulus generator:** generates the randomize test cases by using $urandom which must know the max number the register can hold to generate number between 0 and this number, it also generates corner test cases to target certain conditions to ensure that it will be totally covered.
3) **TB writer:** identifies the initial blocks It also generates monitor commands and instantiation of input output ports. In this design the clock is used so the TB writer is responsible for generating the clock and making the appropriate delays.

## Coverage report and waveforms:

sim (Recursive Coverage Aggregation) - Default

| Total coverage | Stmt count | Stmts hit | Stmts missed | Stmt % | Stmt graph | Branch co |
|---|---|---|---|---|---|---|
| 95.45% | 51 | 51 | 0 | 100.00% | | |
| 95.61% | 8 | 8 | 0 | 100.00% | | |

Library | Files | Project | Instance | sim

sim (Recursive Coverage Aggregation) - Default

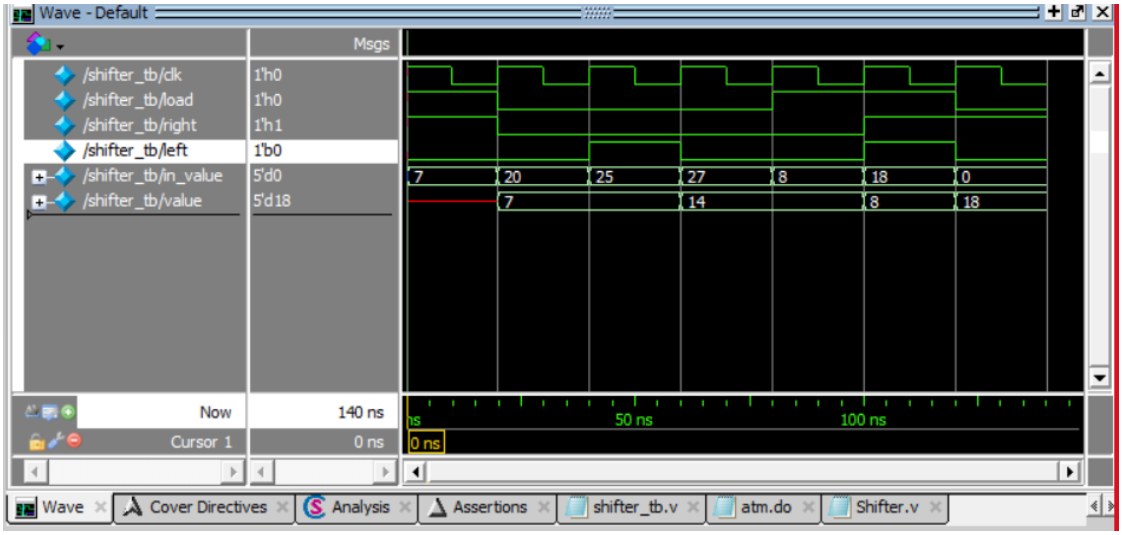
| Branch count | Branches hit | Branches missed | Branch % | Branch graph | Toggle nodes |
|---|---|---|---|---|---|
| 4 | 4 | 0 | 100.00% | | |
| 4 | 4 | 0 | 100.00% | | |

Library | Files | Project | Instance | sim

sim (Recursive Coverage Aggregation) - Default

| aph | Toggle nodes | Toggles hit | Toggles missed | Toggle % | Toggled graph |
|---|---|---|---|---|---|
| | 66 | 57 | 9 | 86.36% | |
| | 38 | 33 | 5 | 86.84% | |

Library | Files | Project | Instance | sim



## Design 3

```
module  case3
(
input wire  [3:0] x,
output reg  [1:0]  pcode
);

always  @(x)

if (x == 4'b1010)
        pcode = 2'b01;
else if (x > 4'b1010)
        pcode = 2'b10;

else pcode = 2'b11;

endmodule
```

## Test Bench generated:

```
module case3_tb ();
reg [3:0] x;

wire [1:0] pcode;



initial
    begin
    $monitor("x = %4b", x);
    end

initial
    begin
    $dumpfile("case3.vcd");
    $dumpvars ;
```

```
    //testing condtional statements


    //directed test cases
    x = 4'b1010;

    #20

    x = 4'b1110;

    #20

    x = 4'b0101;

    #20

    //randomized test cases
    x = $urandom%17;
```

```
    #20

    x = $urandom%17;

    #20

    x = $urandom%17;

    #20

    x = $urandom%17;

    #20
```

```
    $finish;
    end

    case3 DUT (
        .x(x),
        .pcode(pcode)
    );
endmodule
```

**The Test bench generator divides the design into:**

1) **Verilog parser:** since the design is made up of multiple pf if statements, the parser extracts and controls the flow of these conditions, it also identifies the module name case3(), input outputs ports and their registers and wires. In this design logical operators are used in the condition part of the if statements, parser can extract those operators and perform the comparison to produce either 0 (false) and 1(true) to be able to identify either to continue in this if statement or proceed to the other one.
2) **Stimulus generator:** generates the randomize test cases by using $urandom which must know the max number the register can hold to generate number between 0 and this number, it also generates corner test cases to target certain conditions to ensure that it will be totally covered.
3) **TB writer:** identifies the initial blocks It also generates monitor commands and instantiation of input output ports. In this design the clock is used so the TB writer is responsible for generating the clock and generating the appropriate delays.

# Coverage report and waveforms:

sim (Recursive Coverage Aggregation) - Default

| Instance | Design unit | Design unit type | Top Category | Visibility | Cover Options | Total coverage | Stmt count | Stmts hit | Stmts missed |
|---|---|---|---|---|---|---|---|---|---|
| case3_tb | case3_tb(f... | Module | DU Instance | +acc=... | +cover=bcefst | 94.69% | 22 | 21 | 1 |
| DUT | case3(fast) | Module | DU Instance | +acc=... | +cover=bcefst | 95.83% | 4 | 4 | 0 |
| #INITIAL#14 | case3_tb(f... | Process | - | +acc=... | | | | | |
| #vsim_capacity# | | Capacity | Statistics | +acc=... | | | | | |

sim (Recursive Coverage Aggregation) - Default

| Stmt % | Stmt graph | Branch count | Branches hit | Branches missed | Branch % | Branch graph | UDP Condition rows | UDP Conditions hit | UDP Conditions misse |
|---|---|---|---|---|---|---|---|---|---|
| 95.45% | | 3 | 3 | 0 | 100.00% | | | | |
| 100.00% | | 3 | 3 | 0 | 100.00% | | | | |

sim (Recursive Coverage Aggregation) - Default

| Toggle nodes | Toggles hit | Toggles missed | Toggle % | Toggled graph | FEC Condition rows | FEC Conditions hit | FEC Conditions missed | FEC Condition % | FE |
|---|---|---|---|---|---|---|---|---|---|
| 24 | 20 | 4 | 83.33% | | 2 | 2 | 0 | 100.00% | |
| 12 | 10 | 2 | 83.33% | | 2 | 2 | 0 | 100.00% | |