Verification of 16 bit Pipelined Adder using SystemVerilog

By: Prashis Raghuwanshi
Ameya Chhatre

Submitted to -:
Dr. Hamid Mahmoodi
(Assistant Professor , Engineering)

Table of Elements

Acknowledgement

We want to sincerely thank Dr. Hamid Mahmoodi for his support and sincere guidance without which this project was not possible. We sincerely thank the professor for giving the guidance and the time during his office hours and appreciate the quick response received from emails.

Prashis Raghuwanshi

Ameya Chhatre

System/Architecture Description

Pipelined Adder circuits are essential inside microprocessors as part of the ALU,□or arithmetic logic unit, where the processing and manipulation of binary numbers takes place .How does binary addition work? The simplest addition you can do is to add together two 1-bit binary numbers. Suppose the numbers are called A and B. Each of these numbers can take the values 0 or 1. There are four possible additions. In this project we propose a method of addition of Two 16-bit Inputs. We propose a method of using Pipelining Registers for addition of two 16 bit Adder.

Description of Design under test

Number of Input Ports -: 5

Internal Register -: 33 bits

Pipelined Registers -: 25 bits

4 8- bit Input Ports for Input of Data

2 16–bit Output Ports for Output of Data

1 Port for Input of Carry

2 Internal Ports c_out and c_in 1 port for Output of Carry

BLOCK DIAGRAM OF DUT

. ***In our Design ,□Number of parallel single-bit input ports + ∑serial port × packet size >= 36 -- 4 + 4 * 8 >= 36□□***

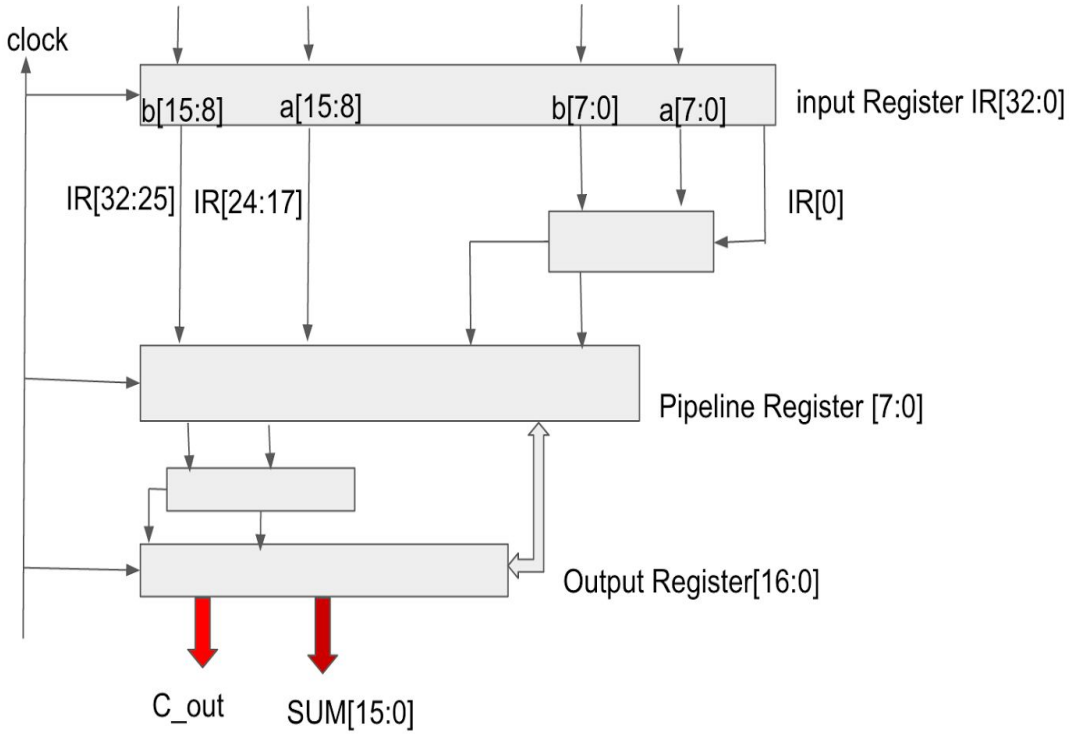


Figure 1 (Block Diagram of DUT)

DESIGN FLOW

- In our Design , Carry Signal becomes high (1) if the addition of integers a and b exceeds 16'hFFFF or 65535.
- No carry is Generated if sum of a and b is less than 65535.
- The Functionality Cover points are the Inputs that is a and b.
- The Constraints are such that we get a fairly large number such that the addition overflows & generates a carry

PIPELINING APPROACH FOR OUR DESIGN

- Pipelining has been done to operate at higher throughput by distributing the processing over multiple cycles of the clock
- Trade - off between speed and physical resources warrant this approach
- More registers needs to be used
- The architecture contains an additional register (PR) between the data input register (IR) and the data output registers.

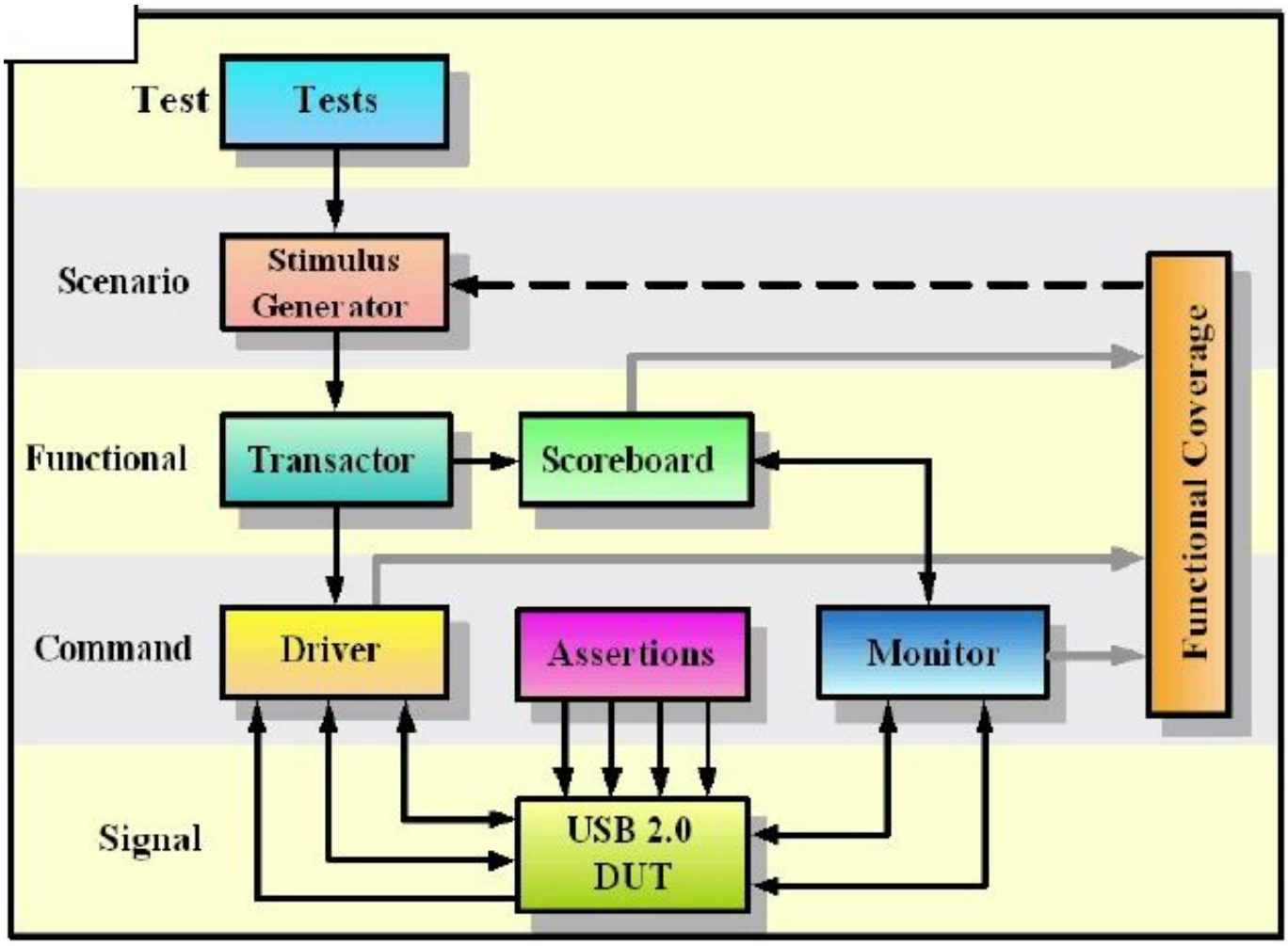
VERILOG CODE FOR DESIGN OF PIPELINED ADDER

```
module adder #(parameter

size    = 16,

half    = size/2,          //8
double  = 2*size,          //32
triple  = 3*half,          //24

size1 = half - 1,          //7
size2 = size - 1,          //15
size3 = half + 1,          //9
R1 = 1,                    //R1 = 1

L1 = half,                 //L1 = 8
R2 = size3,                //R2 = 9
L2 = size,                 //L2 = 16
R3 = size + 1,             //R3 = 17
L3 = size + half,          //L3 = 24
R4 = double - half + 1,    //R4 = 25
L4 = double                //L4 = 32

                    //L1 = 8; L2 = 16; L3 = 24; L4 = 32
                    //R1 = 1; R2 = 9;  R3 = 17; R4 = 25
)

(

input [15:0]    a, b,      //16 bit inputs
input c_in, clk,
output [size2:0]    sum,    // 16 bit output
```

```
output c_out

);

reg [double:0]       IR;   //33 bit IR
reg [triple:0]       PR;   //25 bit PR
reg [size:0]         OR;   //17 bit OR

assign {c_out, sum} = OR;   // 1 bit carry out + 16 bit sum = 17 bit OR

always @ (posedge clk) begin

//Load Input Register

IR[0] <= c_in;              // We are loading the 16 bit input data in a & b +
carry in signal in the Input Register = 33 bit data
IR[L1:R1] <= a[size1:0];     //IR[8:1]   <= a[7:0]
IR[L2:R2] <= b[size1:0];     //IR[16:9]  <= b[7:0]
IR[L3:R3] <= a[size2: half]; //IR[24:17] <= a[15:8]
IR[L4:R4] <= b[size2: half]; //IR[32:25] <= b[15:8]

//Load Pipeline Register     //Piperline Register is 25 bit. We are loading the
upper (upper 8 bit a, upper 8 bit b) bits of IR +
                        //8 bit a + 8 bit b + Cin bit = 9 bit data

PR[L3:R3] <= IR[L4:R4];      //PR[24:17] <= IR[32:25]
PR[L2:R2] <= IR[L3:R3];      //PR[16:9]  <= IR[24:17]
PR[half:0] <= IR[L2:R2] + IR[L1:R1] + IR[0];              //PR[8:0]  <= IR[16:9]
+ IR [8:1] + IR[0]

// Load Output Register      //Output Register is 17 bit. addition of upper 8
bits of a, upper 8 bits of b + 9 bits from lower addition
```

```
OR <= {{1'b0,PR[L3:R3]} + {1'b0,PR[L2:R2]} + PR[half], PR[size1:0]};
 //OR <= (0, 8) + (0, 8) + 9

end
endmodule
```

SYSTEM VERILOG MODEL



INTERFACE CODE

```
interface adder_io(input logic clk);
  parameter simulation_cycle = 100;


  logic [15:0]  a;
  logic [15:0]  b;
  logic         c_in;

  logic [size2:0]       sum;
  logic         c_out;

  clocking cb @ (posedge clk);
        default input #1 output #1;
        output a;
        output b;
        output cin;
        input sum;
        input c_out;
  endclocking

  modport TB(output a,output b,output c_in,clocking cb,input sum,input c_out);

endinterface
```

TOP LEVEL CODE

```
module adder_test_top;

bit clk;

adder_io top_if(clk);
adder_tb t(top_if);

adder dut(
.a(top_if.a),
.b(top_if.b),
.c_in(top_if.c_in),
.clk(top_if.clk),
.sum(top_if.sum),
.c_out(top_if.c_out)
);

initial begin

forever begin
    #5
    clk = ~clk;
    end
  end
endmodule
```

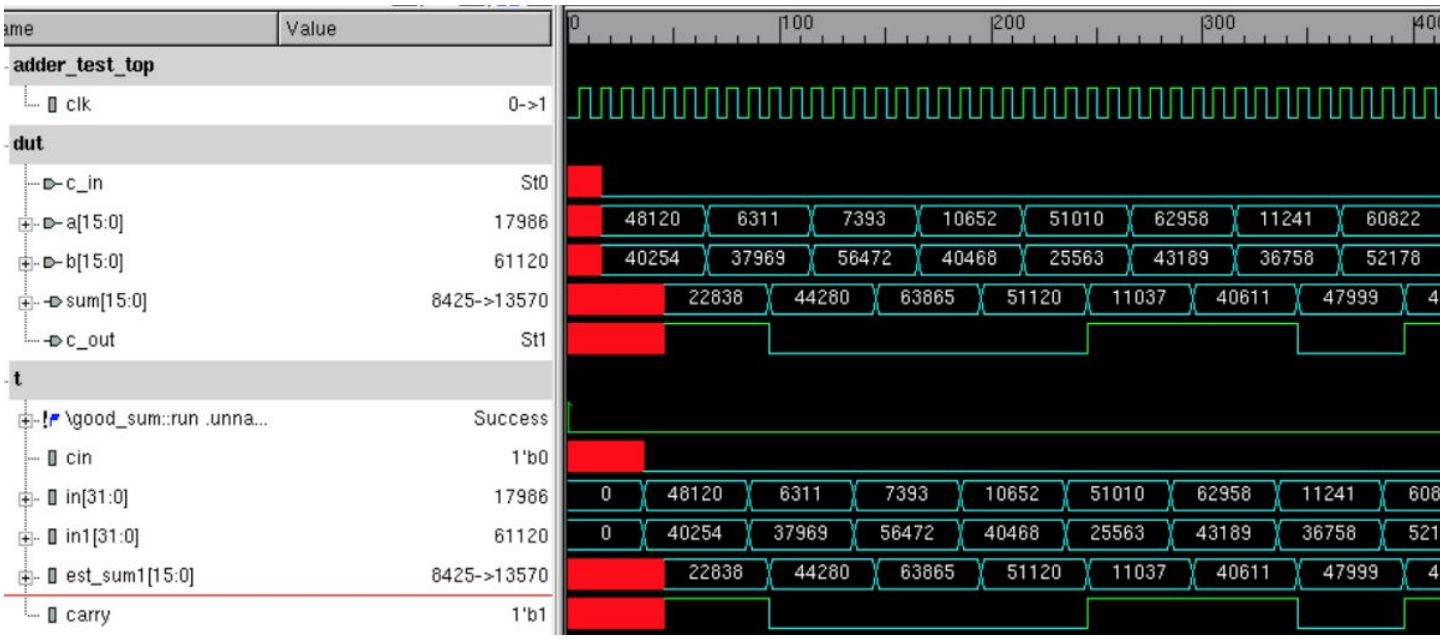
TESTBENCH CODE

```
program automatic adder_tb(adder_io.TB adder_io);

int in=0;
logic i;
logic k;
int j = 0;
logic [16:0]est_sum;
logic [15:0]est_sum1;
logic carry;
int in1 ;
logic cin;
int est_sum1;

covergroup funct;                          //Functional Coverage for input a
    coverpoint adder_io.a
       {option.auto_bin_max = 4;}
endgroup

covergroup funct2;                          //Functional Coverage for input b
   coverpoint adder_io.b
       {option.auto_bin_max = 4;}
endgroup

class good_sum;
    rand int len[];                        // Define Constraints for input a
    constraint c_len {foreach (len[i])
    len[i] inside {[1:65535]};
    len.sum() > 65535;
    len.size() inside {[10:1000]};}  ;
```

```
   rand int len1[];                        // Define Constraints for input b
   constraint c_len1 {foreach (len1[k])
   len1[k] inside {[1:65535]};
   len1.sum() > 65535;
   len1.size() inside {[10:1000]};}

task run();                               //Generate Random Values based on Constraints.
   assert(randomize());
endtask

endclass

task driveinput();                        // Drive the input signal @ clock edge of Clocking
block
   begin
   @adder_io.cb;
   begin
   @adder_io.cb;
   adder_io.a = ua.len[j];
   adder_io.b = ua.len1[j];
   adder_io.c_in = 1'b0;

   functa.sample();                        // Functional Covererage for input a
   functb.sample();                        // Functional Covererage for input b

   @adder_io.cb;
   @adder_io.cb;
   in = adder_io.a;
   in1 = adder_io.b;
   cin = adder_io.c_in;

   @adder_io.cb;
   est_sum = in+in1+cin;                   // Estimate the sum
   est_sum1 = est_sum[15:0];               //Seperate the sum & carry
   carry = est_sum[16];

   end
   end
```

```
endtask

task overflow();                        // Check for overflow condition
  begin
    if (est_sum1 > 65535)
    est_sum1 = 65535;
  end
endtask

task selftest();                        // Self Check the Correctness of the design
                                    // Compare the estimated sum with the Acquired Sum
    begin

    if (adder_io.sum==est_sum1)
        $display("design correct");
    else $display("design incorrect");

    end
endtask

good_sum ua = new();                    // new instance of class
funct functa = new();                   // new instance of functional coverage for a
funct2 functb = new();                  // new instance of functional coverage for b

initial begin

ua.run();                               // display the random values
repeat(50)                              // repeat 50 times to get 100% functional Coverage
begin
driveinput();                           // Drive the inputs with the constrained random
stimulus & estimate the sum
selftest();
overflow();                             // Check for overflow condition

j = j+1;                                // increment the counter for repeating 50 times
end
end
endprogram
```

FINAL WAVEFORM



- In the Waveform , Estimated output is equal to the actual output at every clock cycle.
- Carry is generated when addition to two integers a and b is more than 65535
- a and b are being Randomized

SELF CHECKING REPORT

```
Running simv -V
Chronologic VCS simulator copyright 1991-2012
Contains Synopsys proprietary information.
Compiler version G-2012.09-SP1; Runtime version G-2012.09-SP1;  Dec 13 10:08 2015
VCS Build Date = Feb 25 2013 20:36:30
Start run at Dec 13 10:08 2015

design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
```

- Self Checking is Successfully Completed

Functional Coverage Report

dashboard | hierarchy | modlist | **groups** | **tests** | asserts

Total Groups Coverage Summary

SCORE	WEIGHT
100.00	1

Total groups in report: 2

SCORE	WEIGHT	GOAL	NAME
100.00	1	100	adder_test_top.t::funct
100.00	1	100	adder_test_top.t::funct2

dashboard | hierarchy | modlist | **groups** | **tests** | asserts

0%	10%	20%	30%	40%	50%	60%	70%	80%	90%

Functional Coverage Report
User Defined Bins Report

Summary for Group adder_test_top.t::funct

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
Variables	4	0	4	100.00

Variables for Group adder_test_top.t::funct

VARIABLE	EXPECTED	UNCOVERED	COVERED	PERCENT	GOAL	WEIGHT
adder_io.a	4	0	4	100.00	100	1

Go to top

Summary for Variable adder_io.a

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
User Defined Bins	4	0	4	100.00

User Defined Bins for adder_io.a

Bins

NAME	COUNT	AT LEAST
d	15	1
c	11	1
b	11	1
a	13	1

- Functionality Coverage is done by defining Manual bins as well as Autobins.
- We have got 100 % Functionality Coverage using Autobins and user defines bins.

Functional Coverage Report Using AutoBins

Summary for Group adder_test_top.t::funct2

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
Variables	4	0	4	100.00

Variables for Group adder_test_top.t::funct2

VARIABLE	EXPECTED	UNCOVERED	COVERED	PERCENT	GOAL	WEIGHT
adder_io.b	4	0	4	100.00	100	1

Go to top

Summary for Variable adder_io.b

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
Automatically Generated Bins	4	0	4	100.00

Automatically Generated Bins for adder_io.b

Bins

NAME	COUNT	AT LEAST
auto[0:16383]	11	1
auto[16384:32767]	9	1
auto[32768:49151]	16	1
auto[49152:65535]	14	1

LOG FILE REPORT

```
Note: Bumping stack limit from 10240 to 65536 Kbytes.
Command: vcs -V -R -sverilog ./adder_test_top.sv ./adder_if.sv ./adder_tb.sv ./adder.v \
-o simv -l report.log
*** Using c compiler gcc instead of cc ...
/packages/synopsys/vcs_mx/G-2012.09-SP1/linux/bin/vcs1 -Mcc=gcc -Mcplusplus=g++ -Masflags=--32 \
-Mcfl= -pipe -m32 -O -I/packages/synopsys/vcs_mx/G-2012.09-SP1/include  -Mldflags= \
-melf_i386 -m32  -Mout=simv -Mamsrun="" -Mvcsaceobjs="" -Mobjects="
/packages/synopsys/vcs_mx/G-2012.09-SP1/linux/lib/libvirsim.so \
/packages/synopsys/vcs_mx/G-2012.09-SP1/linux/lib/librterrorinf.so
/packages/synopsys/vcs_mx/G-2012.09-SP1/linux/lib/libsnpsmalloc.so \
" -Msaverestoreobj=/packages/synopsys/vcs_mx/G-2012.09-SP1/linux/lib/vcs_save_restore_new.o \
-Mcrt0= -Mcrtn="" -Mcsrc=""
-Msyslibs=/packages/synopsys/vcs_mx/G-2012.09-SP1/linux/lib/ctype-stubs_32.a \
-ldl  -Xvcs_run_simv=1 -V -o simv -l report.log -sverilog -gen_obj ./adder_test_top.sv \
./adder_if.sv ./adder_tb.sv ./adder.v
                    Chronologic VCS (TM)
          Version G-2012.09-SP1 -- Tue Dec 15 11:38:57 2015
              Copyright (c) 1991-2012 by Synopsys Inc.
                    ALL RIGHTS RESERVED

This program is proprietary and confidential information of Synopsys Inc.
and may be used and disclosed only as authorized in a license agreement
controlling such use and disclosure.

Parsing design file './adder_test_top.sv'
Parsing design file './adder_if.sv'
Parsing design file './adder_tb.sv'
Parsing design file './adder.v'
Top Level Modules:
       adder_test_top
No TimeScale specified
Starting vcs inline pass...
3 modules and 0 UDP read.
            However, due to incremental compilation, no re-compilation is necessary.
( cd csrc ; make -f Makefile SNPS_VCS_TMPDIR=/tmp/vcs_20151215193854_20393 product \
)
ld -r -m elf_i386 -o pre_vcsobj_1_1.o --whole-archive pre_vcsobj_1_1.a --no-whole-archive \
```

```
ld -r -m elf_i386 -o pre_vcsobj_1_2.o --whole-archive pre_vcsobj_1_2.a --no-whole-archive \

if [ -x ../simv ]; then chmod -x ../simv; fi
g++  -o ../simv -melf_i386 -m32    -Wl,-whole-archive    -Wl,-no-whole-archive   \
SIM_l.o 5NrI_d.o 5NrIB_d.o    pre_vcsobj_1_1.o pre_vcsobj_1_2.o  rmapats_mop.o rmapats.o \
/packages/synopsys/vcs_mx/G-2012.09-SP1/linux/lib/libnplex_stub.so
/packages/synopsys/vcs_mx/G-2012.09-SP1/linux/lib/libvirsim.so \
/packages/synopsys/vcs_mx/G-2012.09-SP1/linux/lib/librterrorinf.so
/packages/synopsys/vcs_mx/G-2012.09-SP1/linux/lib/libsnpsmalloc.so \
/packages/synopsys/vcs_mx/G-2012.09-SP1/linux/lib/libvcsnew.so
/packages/synopsys/vcs_mx/G-2012.09-SP1/linux/lib/libuclinative.so \
/packages/synopsys/vcs_mx/G-2012.09-SP1/linux/lib/vcs_save_restore_new.o
/packages/synopsys/vcs_mx/G-2012.09-SP1/linux/lib/ctype-stubs_32.a \
-ldl  -lc -lm -lpthread -ldl
../simv up to date
Running simv -V -a report.log
Command: ./simv -V -a report.log
Chronologic VCS simulator copyright 1991-2012
Contains Synopsys proprietary information.
Compiler version G-2012.09-SP1; Runtime version G-2012.09-SP1;  Dec 15 11:39 2015
VCS Build Date = Feb 25 2013 20:36:30
Start run at Dec 15 11:39 2015

design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
```

design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
design correct
$finish at simulation time 2495

V C S S i m u l a t i o n R e p o r t

Time: 2495
CPU Time: 0.600 seconds; Data structure size: 0.0Mb
Tue Dec 15 11:39:03 2015
CPU time: .179 seconds to compile + .046 seconds to elab + .251 seconds to link + .709 seconds in simulation

FINAL RESULTS

Tasks Successfully Completed

- Verification Planning
- Interface Module
- Systemverilog Testbench
- Task for Randomization
- Constraint Random Stimulus Generation
- Building Top Module
- Task for Driving Inputs to DUT
- Task for Self Checking successfully done
- Functional Coverage 100% achieved

VERIFICATION CHALLENGES

Some of the challenges in Verification were -:

- Checking for Overflow and generating a carry
- Self Checking the design for every cycle
- Randomizing the Inputs
- Constraining the Randomized Inputs upto range 65535

REFERENCES

- _“Systemverilog for Verification”, 2nd Edition by Chris Spear
- “The complete Verilog Book”, by Vivek Sagdeo ,Springer Publication
- www.testbench.in
- www.asicworld.com
- www.edaplayground.com
- www.verificationacademy.com