Functional And Formal Verification OF Digital Design

TOPIC:-Bitonic Sorter

5^{TH} SEM

ANIRUDH SIMHA(PES1UG21EC001)

ABHISHEK A SHETTY(PES1UG21EC008)

ADHOKSHAJA.R.B(PES1UG21EC011)

AKASH KATATE(PES1UG21EC022)

AKASH RAVI BHAT(PES1UG21EC025)

INTRODUCTION

A bitonic sorter is a type of parallel sorting algorithm that can efficiently sort a sequence of elements in a specific order. The bitonic sorter algorithm was first proposed by Ken Batcher in 1968. It is particularly well-suited for parallel processing architectures.

The basic idea behind bitonic sorting is to recursively build a bitonic sequence and then repeatedly split and merge the sequence until the entire sequence is sorted. A bitonic sequence is one that starts in ascending order, reaches a maximum element (the "peak"), and then continues in descending order.

DESIGN CODE

```
module bitonic_sorter(output wire [31:0] o1, o2, input wire [31:0] A, B,
input wire clk, rst, dir, enable);
  reg sel;
  reg [31:0] out1, out2;

  always @(posedge clk or posedge rst) begin
    if (rst) begin
      out1 <= 32'b0;
      out2 <= 32'b0;
    end
    else if (enable) begin
      if (dir == 1'b0) begin
        sel <= (A > B);
        out1 <= (sel) ? B : A;
        out2 <= (sel) ? A : B;
      end
      if (dir == 1'b1) begin
        sel <= (A < B);
        out1 <= (sel) ? B : A;
        out2 <= (sel) ? A : B;
      end
    end
  end

  assign o1 = out1;
  assign o2 = out2;
endmodule
```

LAYERED TESTBENCH CODE

TOP MODULE:

```
module tb_bitonic_top;
bit clk;
bit rst;
bitonic_intf intf(clk,rst);
bitonic_test test(intf);
bitonic_sorter dut(.clk(intf.clk),
.rst(intf.rst),
.o1(intf.o1),
.o2(intf.o2),
.dir(intf.dir),
.enable(intf.enable),
.A(intf.A),
.B(intf.B));
always #5 clk = ~clk;
initial begin
  rst = 1;
#10; rst = 0;
end

endmodule
```

CLASS:

```
class bitonic_bfm;

virtual bitonic_intf intf;

mailbox gen2bfm;

int no_transactions;

function new(virtual bitonic_intf intf,mailbox gen2bfm);

this.intf = intf;

this.gen2bfm = gen2bfm;

endfunction

task reset;

wait(intf.rst);

$display("Reset Initiated");

intf.bfm_cb.dir <= 0;

intf.bfm_cb.enable <= 0;

intf.bfm_cb.A <= 0;

intf.bfm_cb.B <= 0;

wait(!intf.rst);

$display("Reset finished");

endtask

task main;

forever begin

bitonic_trans trans;

gen2bfm.get(trans);

$display("Transaction No. = %0d", no_transactions);

intf.bfm_cb.dir <= trans.dir;

intf.bfm_cb.enable <= trans.enable;
intf.bfm_cb.A <= trans.A;
intf.bfm_cb.B <= trans.B;
repeat(2)@(posedge intf.clk);
trans.o1 = intf.bfm_cb.o1;
trans.o2= intf.bfm_cb.o2;
trans.display();
no_transactions++;
end
endtask
endclass
```

ENVIRONMENT:

```
`include "bitonic_cov.sv"
class bitonic_env;
bitonic_gen gen;
bitonic_bfm bfm;
bitonic_cov cov;
mailbox gen2bfm;
virtual bitonic_intf intf;
event ended;
function new(virtual bitonic_intf intf);
this.intf = intf;
gen2bfm = new();
gen = new(gen2bfm, ended);
bfm = new(intf, gen2bfm);
cov = new();
```

```
endfunction
task pre_test;
bfm.reset();
endtask
task test;
fork
gen.main();
bfm.main();
cov.main();
join_any
endtask
task post_test;
wait(ended.triggered);
wait(gen.repeat_count == bfm.no_transactions);
endtask
task run;
pre_test();
test();
post_test();
$finish;
endtask
endclass
```

DEFINES:

```
`include "design.sv"
```

```
`include "bitonic_trans.sv"

`include "bitonic_gen.sv"

`include "bitonic_intf.sv"

`include "bitonic_bfm.sv"

`include "bitonic_env.sv"

`include "bitonic_test.sv"

`include "tb_bitonic_tb.sv"
```

TRANS:

```
class bitonic_trans;

rand bit [31:0] A;

rand bit [31:0] B;

rand bit dir;

rand bit enable;

bit [31:0] o1;

bit [31:0] o2;

function void display();

$display(" ");

$display("\t dir = %0b, \t enable = %0b, \t A = %0b, \t B = %0b, \t o1 = %0b, \t o2 = %0b", dir,enable,A,B,o1,o2);

$display(" ");

endfunction

endclass
```

INTERFACE:

```
interface bitonic_intf(input logic clk,rst);

logic dir;

logic enable;
logic  [31:0] A;
logic  [31:0] B;
logic  [31:0] o1;
logic  [31:0] o2;

clocking bfm_cb @(posedge clk);
default input #1 output #1;
output dir;
output enable;
output A;
output B;
input o1;
input o2;
endclocking

clocking monitor_cb @(posedge clk);
default input #1 output #1;
input dir;
input enable;
input A;
input B;
input o1;
input o2;
endclocking
modport BFM (clocking bfm_cb, input clk,rst);
modport MONITOR (clocking monitor_cb, input clk, rst);
endinterface
```

COVERPOINT:

```
class bitonic_cov;
bitonic_trans trans = new();
covergroup cov_inst;
option.per_instance = 1;
DIR:coverpoint trans.dir {bins arm = {0,1};}
ENABLE: coverpoint trans.enable {bins trigger = {0,1};}
A: coverpoint trans.A {bins A = { [0: 4294967295]}; }
B: coverpoint trans.B {bins B = { [0: 4294967295]}; }
O1: coverpoint trans.o1 {bins o1 = { [0: 4294967295]}; }
O2: coverpoint trans.o2 {bins o2 = { [0: 4294967295]}; }
endgroup
function new();
cov_inst = new;
endfunction
task main;
cov_inst. sample();
endtask
endclass
```

TEST:

```
program bitonic_test(bitonic_intf intf);
```

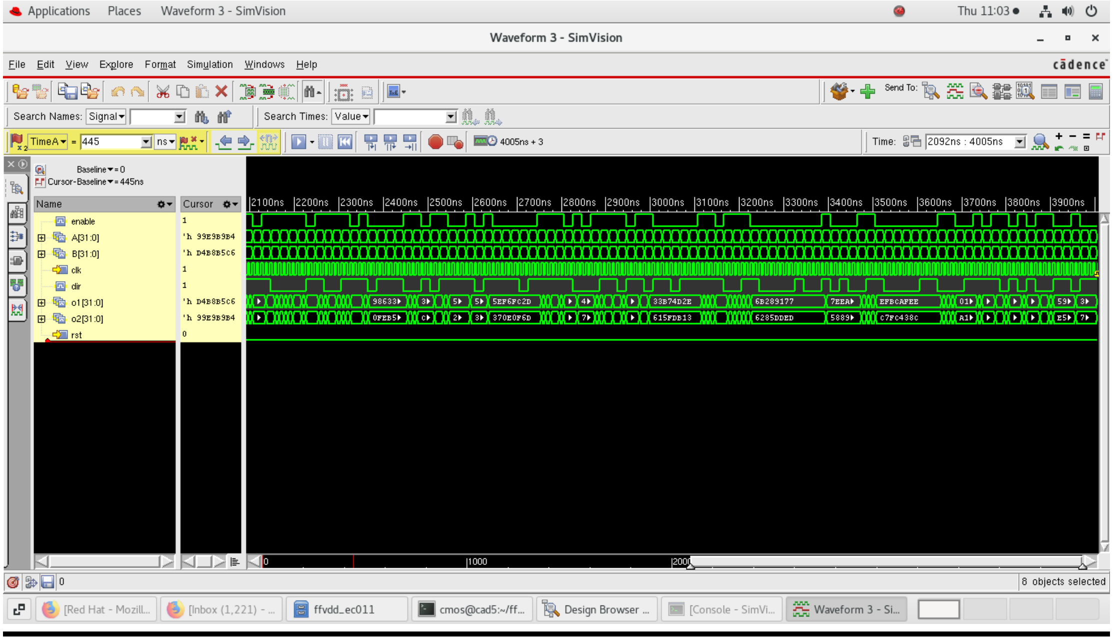
```
bitonic_env env;
initial begin
env = new(intf);
env.gen.repeat_count = 200;
env.run();
end
endprogram
```

GENERATOR:

```
class bitonic_gen;
rand bitonic_trans trans;
mailbox gen2bfm;
event ended;
int repeat_count;
function new(mailbox gen2bfm, event ended);
this.gen2bfm = gen2bfm;
this.ended = ended;
endfunction
task main;
repeat(repeat_count) begin
trans = new();
if(!trans.randomize()) $display("Randomization Failed");
gen2bfm.put(trans);
end
->ended;
endtask
```

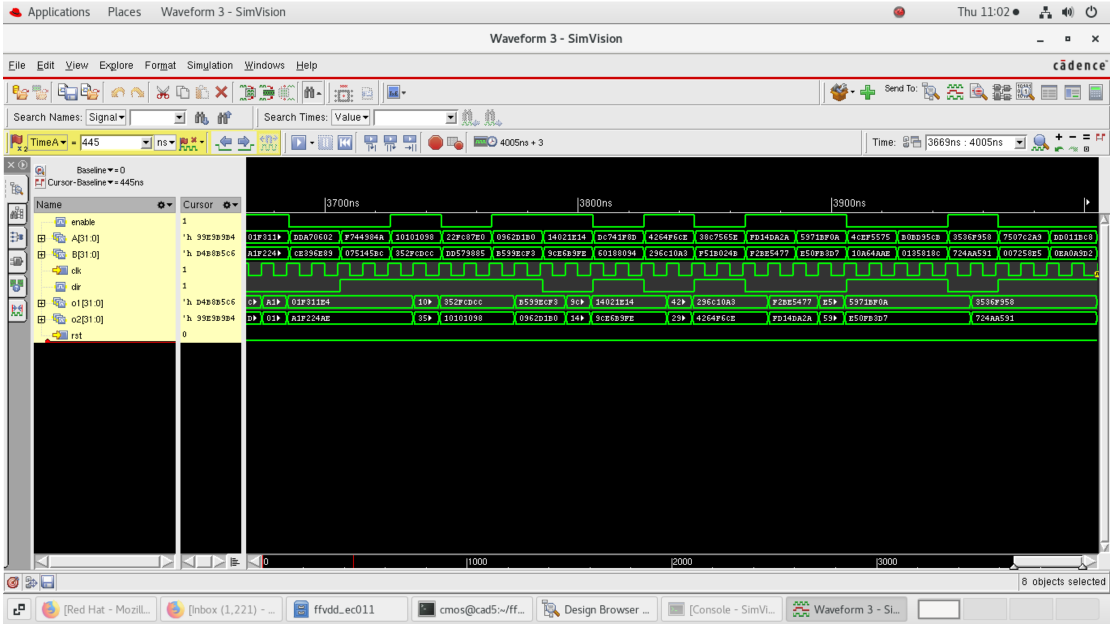
endclass

OUTPUT WAVEFORM

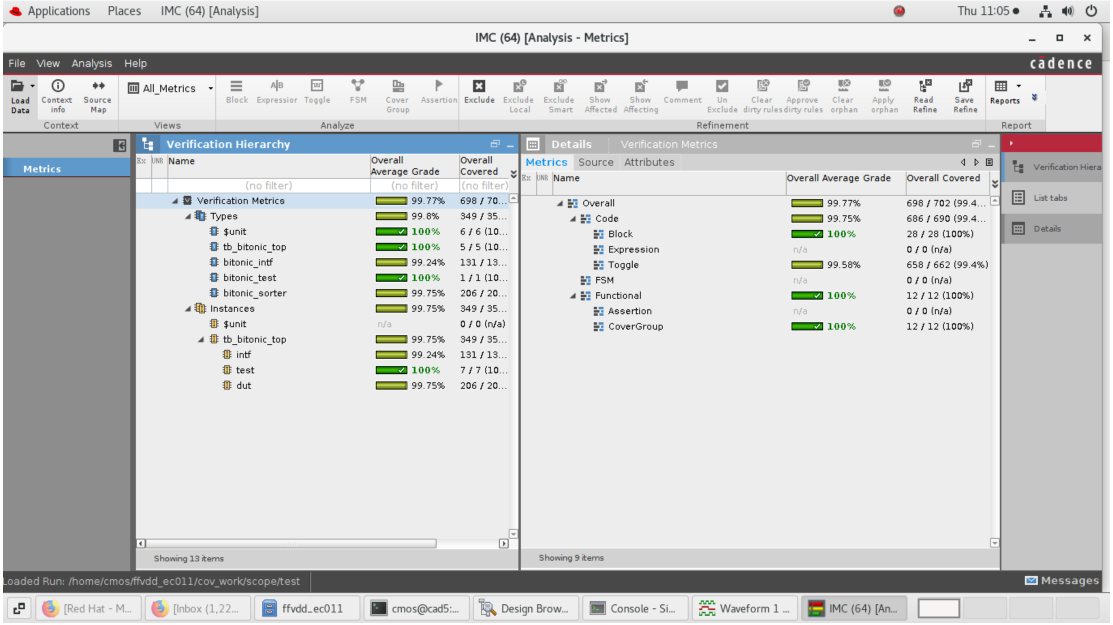


Baseline = 0
Cursor-Baseline = 445ns

Name	Cursor
enable	1
A[31:0]	'h 99E9B9B4
B[31:0]	'h D4B8B5C6
clk	1
dir	1
o1[31:0]	'h D4B8B5C6
o2[31:0]	'h 99E9B9B4
rst	0



COVERAGE REPORT



Verification Hierarchy

Name	Overall Average Grade	Overall Covered
(no filter)	(no filter)	(no filter)
Verification Metrics	99.77%	698 / 70...
Types	99.8%	349 / 35...
$unit	100%	6 / 6 (10...
tb_bitonic_top	100%	5 / 5 (10...
bitonic_intf	99.24%	131 / 13...
bitonic_test	100%	1 / 1 (10...
bitonic_sorter	99.75%	206 / 20...
Instances	99.75%	349 / 35...
$unit	n/a	0 / 0 (n/a)
tb_bitonic_top	99.75%	349 / 35...
intf	99.24%	131 / 13...
test	100%	7 / 7 (10...
dut	99.75%	206 / 20...

Showing 13 items

Details — Verification Metrics

Metrics Source Attributes

Name	Overall Average Grade	Overall Covered
Overall	99.77%	698 / 702 (99.4...
Code	99.75%	686 / 690 (99.4...
Block	100%	28 / 28 (100%)
Expression	n/a	0 / 0 (n/a)
Toggle	99.58%	658 / 662 (99.4%)
FSM	n/a	0 / 0 (n/a)
Functional	100%	12 / 12 (100%)
Assertion	n/a	0 / 0 (n/a)
CoverGroup	100%	12 / 12 (100%)

Showing 9 items

Loaded Run: /home/cmos/ffvdd_ec011/cov_work/scope/test