System Verification

Exercise 6 - Verification Plan

ANIK BARUA
UTU Student ID: 511580

Contents

1. Introduction

The target of this paper is to present an UVM verification environment for a PicoBlaze's ALU. Here, we will discuss about the environment setup for the verification, the use of components and the verification result after simulation.

In general, PicoBlaze is a kind of soft processor developed by the Xilinx. It was used for the CPLD and the FPGA. They have a speed of 100 MIPS and it used the RISC architecture and that is 8 bit. It has the core that has the implementation which is behavioral and is not dependent on device. Here, ALU is responsible for the operation that is waiting for execution and which operation will be executed can be selected by the address port. If we simply consider the structure of a microcontroller, it will be as follows:

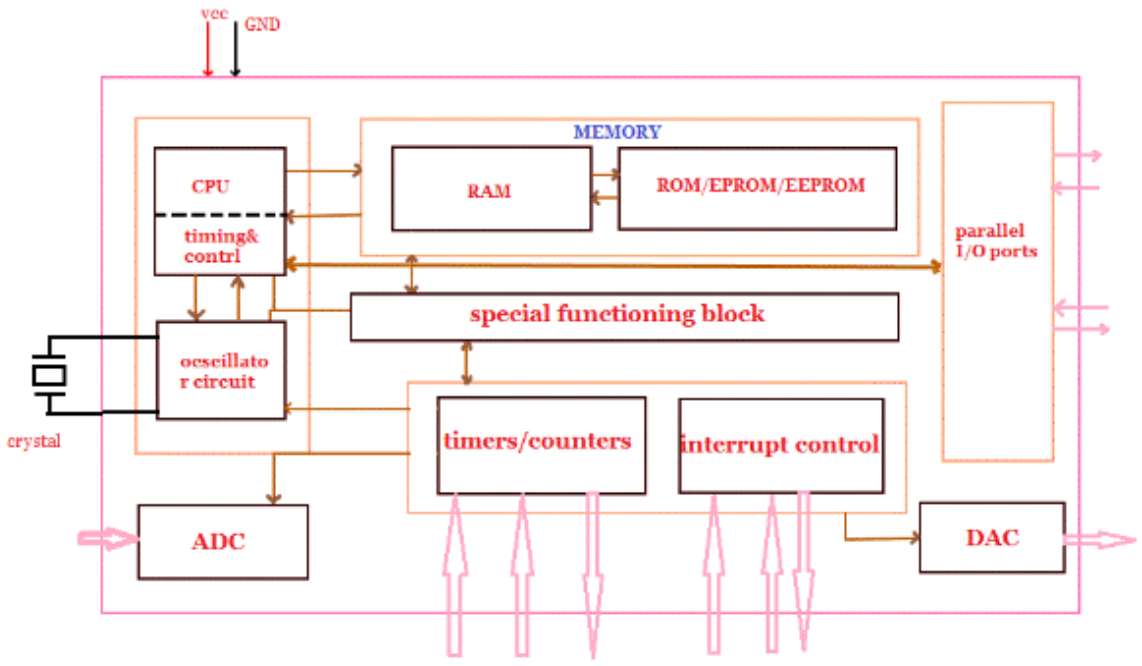


Fig 1: Structure of Microcontroller [2]

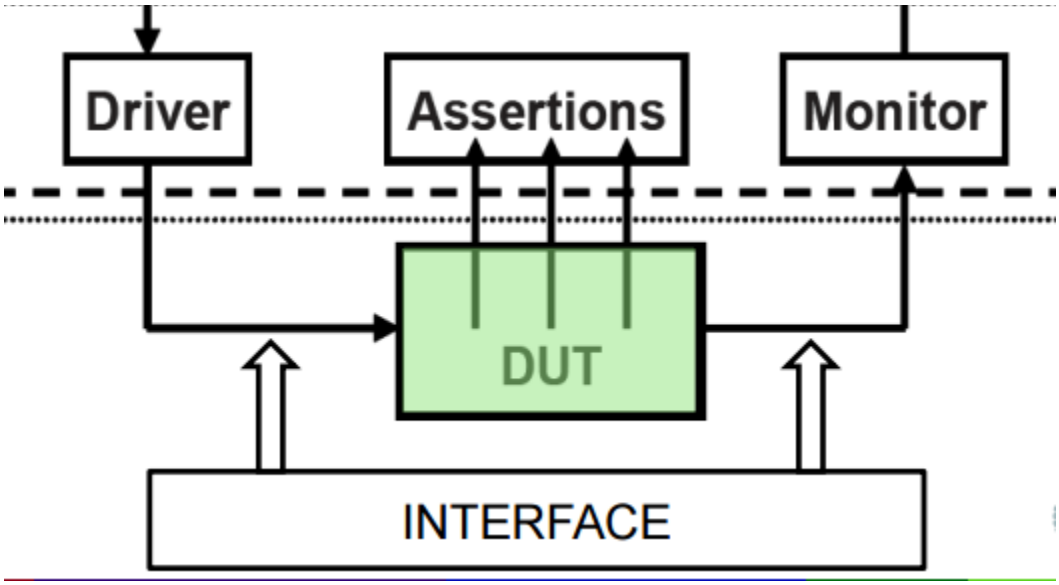
A microcontroller is consists of different small blocks and each block is responsible for executing a specific operation. Here, I need to mention about Arithmetic Logic Unit (ALU) that executes calculation such as addition, subtraction, multiplication, and division. Also, the ALU can execute some other logical operation such as AND, OR, XOR, calculation carry flag, shift and rotate operation.

2. Schedule:

Here, we will discuss about all the phases and schedule of the whole verification process with implemented code. A new module will be added in all steps and in this way we will get the final verification environment.

To create a verification environment, we need to create Design Under Test or DUT, environment, driver, interface and the mandatory top module as well. The functionalities of ALU will be checked by the Design Under Test. Also, the connection among the different components and Design Under Test will be maintained by interface which will define how many signals it will allow. We have used system Verilog to express the interface that represents the connections. Some other information such as the activity of pin level that is supported by the DUT.

We are going to start by designing a DUT. The core concept of DUT is that it will take an input signal and will execute an output and there will be some internal calculation or operation as well as we wants. If we want to consider the figure of Driver, DUT and Interface, then it will be like this -



There is a simple code of ALU is given below. It has two input x and y, both of them are 8 bits. As a result, the output is also 8 bit. There are 5 select bits have been used and it has some different operation.

```
module Sample_ALU (
  input wires [3:0] select,
  input wires [7:0] a,b,//Input data
  output result [7:0] output);// Output Result

  always @*
  begin
    case(select)//select that which one to execute
      2'z0:
        output <= x;

      2'z1:
        output <= (x && y);

      2'z2:
        output <= (x || y);

      2'z3:
        output <= ((!x) || y) && (x || (!y));

      2'z4:
        output <= (x + y);

      2'z5:
        output <= (x - y);

     default:
      output <= 4'z0;

        endcase
end
endmodule
```

After that, we need to connect the interface by the code mentioned below.

```
interface dut_if ();

logic [3:0] select;
logic [7:0] x, y;
logic [7:0] output;
logic zero, carry;

endinterface: dut_if
```

After that, we are going to declare the driver. The code is as follows.

```
class Ex6_driver extends uvm_driver #(Ex6_transaction);
  `uvm_component_utils(Ex6_driver)
  virtual Ex6_interface _interface;
  Ex6_config m_config;

  int _iterations;
  uvm_analysis_port #(Ex6_transaction) ap;

  function new(string name, uvm_component parent);
    super.new(name, parent);
    ap = new("ap", this);
  endfunction: new

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    if (!uvm_config_db #(Ex6_config)::get(this, "", "Ex6_config", m_config))
      `uvm_fatal("NOCONFIG", "Failed to get configuration object from driver!");
  endfunction: build_phase
  function void connect_phase(uvm_phase phase);
    _interface = m_config.vif;
    _iterations = m_config.iterations;
  endfunction: connect_phase

  task run_phase(uvm_phase phase);
    Ex6_transaction req_item, req_item_clone;
    repeat(_iterations) begin @(posedge _interface.clock)
      if (!_interface.reset) begin
        seq_item_port.get_next_item(req_item);

        _interface.value1 = req_item.value1;
        _interface.value2 = req_item.value2;
        _interface.mode = req_item.mode;
        ap.write(req_item);
        seq_item_port.item_done();
      end
    end
  endtask: run_phase
```

Then we have to declare the environment that will contain the driver. In general, the declaration and the all the connection is in the environment.

```
class Ex6_env extends uvm_env;

  `uvm_component_utils(Ex6_env)

  Ex6_driver _driver;
  Ex6_sequencer _sequencer;
  Ex6_agent _agent;
  Ex6_analysis _analysis;
  Ex6_scoreboard _sb;

  function new(string name, uvm_component parent = null);
    super.new(name, parent);
  endfunction: new

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    _agent = Ex6_agent::type_id::create("Ex6_agent", this);
    _analysis = Ex6_analysis::type_id::create("Ex6_analysis", this);
    _sb = Ex6_scoreboard::type_id::create("sb", this);
  endfunction: build_phase

  function void connect_phase(uvm_phase phase);
    _sequencer = _agent.m_sequencer;
    _driver = _agent.m_driver;

    _agent.output_analysis_port.connect(_analysis._export);
    _agent.input_analysis_port.connect(_analysis._import);

    _agent.output_analysis_port.connect(_sb.axp_out);
    _agent.input_analysis_port.connect(_sb.axp_in);
  endfunction: connect_phase

  task run_phase(uvm_phase phase);
    uvm_top.print_topology();
  endtask: run_phase

endclass: Ex6_env
```

Then we will declare the top module and it is responsible to hold the instances, DUT, other declaration and the components as well.

```
module top;

  import uvm_pkg::*;
  import Ex6_pkg::*;

  Ex6_interface _interface();
  Ex6_dut dut(_interface.dut_mp);
  Ex6_clock_driver clk(_interface.clock_driver_mp);
  Ex6_env env;

  initial begin
    uvm_config_db #(virtual Ex6_interface)::set(null, "*", "top_interface", _interface);
    run_test();
  end

endmodule: top
```

For the simulation, we will use the following commands –

- .main clear
- transcript file Ex6.log
- vdel -all -lib work
- vlib work
- vlog -f compile_questa_sv.f
- vsim +UVM_TESTNAME=Ex6_test -do "run -all"

After that, we need to declare the DUT sequence, the overall code is as follows –

```
class Ex6_sequence extends uvm_sequence #(Ex6_transaction);
  `uvm_object_utils(Ex6_sequence)
  Ex6_config _config;

  function new(string name="");
    super.new(name);
  endfunction: new

  task body();
    Ex6_transaction req;
    if(!uvm_config_db #(Ex6_config)::get(uvm_root::get(), "*", "Ex6_config", _config))
      `uvm_fatal("SEQUENCE", "Can't read config");
    uvm_test_done.raise_objection(this);
    repeat(_config.iterations)
    begin
      req = Ex6_transaction::type_id::create("req");
      start_item(req);
      if (_config.test_name == "test_zero") begin
    req.value1 = 0;
    req.value2 = 0;
    req.mode = DIV;
      end
      else if (_config.test_name == "test_lt_zero") begin
        req.value1 = 1;
    req.value2 = 2;
    req.mode = SUB;
      end
      else assert(req.randomize());
      finish_item(req);
    end
    uvm_test_done.drop_objection(this);
  endtask: body
endclass: Ex6_sequence
```

After, we also have to declare the driver which is be like –

```
class Ex6_driver extends uvm_driver #(Ex6_transaction);
  `uvm_component_utils(Ex6_driver)
  virtual Ex6_interface _interface;
  Ex6_config m_config;

  int _iterations;
  uvm_analysis_port #(Ex6_transaction) ap;

  function new(string name, uvm_component parent);
    super.new(name, parent);
    ap = new("ap", this);
  endfunction: new

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    if (!uvm_config_db #(Ex6_config)::get(this, "", "Ex6_config", m_config))
      `uvm_fatal("NOCONFIG", "Failed to get configuration object from driver!");
  endfunction: build_phase
  function void connect_phase(uvm_phase phase);
    _interface = m_config.vif;
    _iterations = m_config.iterations;
  endfunction: connect_phase

  task run_phase(uvm_phase phase);
    Ex6_transaction req_item, req_item_clone;
    repeat(_iterations) begin @(posedge _interface.clock)
      if (!_interface.reset) begin
        seq_item_port.get_next_item(req_item);

        _interface.value1 = req_item.value1;
        _interface.value2 = req_item.value2;
        _interface.mode = req_item.mode;
        ap.write(req_item);
        seq_item_port.item_done();
      end
    end
  endtask: run_phase
```

The code of the environment will be –

```
class Ex6_env extends uvm_env;

  `uvm_component_utils(Ex6_env)

  Ex6_driver _driver;
  Ex6_sequencer _sequencer;
  Ex6_agent _agent;
  Ex6_analysis _analysis;
  Ex6_scoreboard _sb;

  function new(string name, uvm_component parent = null);
    super.new(name, parent);
  endfunction: new

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    _agent = Ex6_agent::type_id::create("Ex6_agent", this);
    _analysis = Ex6_analysis::type_id::create("Ex6_analysis", this);
    _sb = Ex6_scoreboard::type_id::create("sb", this);
  endfunction: build_phase

  function void connect_phase(uvm_phase phase);
    _sequencer = _agent.m_sequencer;
    _driver = _agent.m_driver;

    _agent.output_analysis_port.connect(_analysis._export);
    _agent.input_analysis_port.connect(_analysis._import);

    _agent.output_analysis_port.connect(_sb.axp_out);
    _agent.input_analysis_port.connect(_sb.axp_in);
  endfunction: connect_phase

  task run_phase(uvm_phase phase);
    uvm_top.print_topology();
  endtask: run_phase

endclass: Ex6_env
```

In the configuration phase we will declare the config class and the code will be –

```
class Ex6_config extends uvm_object;
   `uvm_object_utils(Ex6_config);


   virtual Ex6_interface vif;

   int iterations = 3;
   string test_name = "test";

   function new(string name = "");
     super.new(name);
   endfunction

endclass
```

The code of the test class will be –

```
class Ex6_test extends uvm_test;
   `uvm_component_utils(Ex6_test)
   Ex6_env m_env;
   Ex6_config m_config;
   Ex6_sequence _sequence;
   Ex6_agent_config _agent_config;

   function new(string name = "Ex6_test", uvm_component parent = null);
     super.new(name, parent);
   endfunction

   function void set_config_params();
     m_config = Ex6_config::type_id::create("m_config");
     _agent_config = Ex6_agent_config::type_id::create("agent_config");
     if(!uvm_config_db #(virtual Ex6_interface)::get(this, "uvm_test_top", "top_interface", m_config.vif))
       `uvm_fatal("Ex6 TEST", "Can't read VIF");
     if(!uvm_config_db #(virtual Ex6_interface)::get(this, "uvm_test_top", "top_interface", _agent_config.vif))
       `uvm_fatal("Ex6 TEST", "Can't read VIF");
     m_config.iterations = 16;
     _agent_config.active = UVM_ACTIVE;

     uvm_config_db #(Ex6_config)::set(uvm_root::get(), "*", "Ex6_config", m_config);
     uvm_config_db #(Ex6_agent_config)::set(this, "*", "Ex6_agent_config", _agent_config);
   endfunction;
   function void build_phase(uvm_phase phase);
     super.build_phase(phase);
     set_config_params();
     m_env = Ex6_env::type_id::create("m_env", this);
   endfunction: build_phase
   task run_phase(uvm_phase phase);
     _sequence = Ex6_sequence::type_id::create("_sequence");
     phase.raise_objection(this);
       _sequence.start(m_env._sequencer);
     phase.drop_objection(this);
   endtask: run_phase
endclass
```

Then we have to create the agent and the monitor class. The code of the agent class is –

```
class Ex6_agent extends uvm_component;
  `uvm_component_utils(Ex6_agent)
  Ex6_agent_config m_cfg;
  uvm_analysis_port #(Ex6_transaction) input_analysis_port;
  uvm_analysis_port #(Ex6_transaction) output_analysis_port;

  Ex6_monitor m_monitor;
  Ex6_sequencer m_sequencer;
  Ex6_driver m_driver;

  function new(string name = "Ex6_agent", uvm_component parent = null);
    super.new(name, parent);
  endfunction

  function void build_phase(uvm_phase phase);
    if (!uvm_config_db #(Ex6_agent_config)::get(this, "", "Ex6_agent_config", m_cfg))
      `uvm_fatal("CONFIC_LOAD", "Cannot get() configuration Ex6_agent_config from uvm_config_db!")

    m_monitor = Ex6_monitor::type_id::create("m_monitor", this);

    if (m_cfg.active == UVM_ACTIVE) begin
      m_driver = Ex6_driver::type_id::create("m_driver", this);
      m_sequencer = Ex6_sequencer::type_id::create("m_sequencer", this);
    end

  endfunction: build_phase

  function void connect_phase(uvm_phase phase);
    output_analysis_port = m_monitor.ap;
    input_analysis_port = m_driver.ap;

    if (m_cfg.active == UVM_ACTIVE) begin
      m_driver.seq_item_port.connect(m_sequencer.seq_item_export);
    end
  endfunction: connect_phase

endclass: Ex6_agent
```

The code of the monitor is –

```
class Ex6_monitor extends uvm_monitor;
  `uvm_component_utils(Ex6_monitor)
  uvm_analysis_port #(Ex6_transaction) ap;
  Ex6_config m_config;

  virtual Ex6_interface m_vif;
  function new (string name, uvm_component parent);
    super.new(name, parent);
  endfunction: new

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    ap = new("ap", this);

    if (!uvm_config_db #(Ex6_config)::get(this, "", "Ex6_config", m_config))
      `uvm_fatal("NOCONF",{"Config must be set for: ", get_full_name(), ".m_config"});
  endfunction: build_phase

  function void connect_phase(uvm_phase phase);
    m_vif = m_config.vif;
  endfunction: connect_phase

  task run_phase(uvm_phase phase);
    Ex6_transaction item;
    item = Ex6_transaction::type_id::create("item");

    forever @(posedge m_vif.clock) begin
      item.value1 = m_vif.value1;
      item.value2 = m_vif.value2;
      item.result = m_vif.result;
      item.correct = m_vif.correct;
      item.mode = m_vif.mode;
      ap.write(item);
    end
  endtask: run_phase
endclass: Ex6_monitor
```

After that we need to complete the environment by declaring the predictor –

```
`uvm_component_utils(Ex6_predictor)

Ex6_transaction out_txn;
uvm_analysis_port #(Ex6_transaction) results_ap;

covergroup transaction_coverage;
  cov_value1: coverpoint out_txn.value1 {
    bins zero = {0};
    bins f1to10 = {[1:9]};
    bins f10to100 = {[10:100]};
    bins rest = default;
  }
  cov_value2: coverpoint out_txn.value2 {
    bins zero = {0};
    bins f1to10 = {[1:9]};
    bins f10to100 = {[10:100]};
    bins rest = default;
  }
  mode_value: coverpoint out_txn.mode {
    bins add = {ADD};
    bins sub = {SUB};
    bins mul = {MUL};
    bins div = {DIV};
  }
endgroup

function new(string name, uvm_component parent);
  super.new(name, parent);
  transaction_coverage = new;
endfunction

function void build_phase(uvm_phase phase);
  super.build_phase(phase);

  results_ap = new("results_ap", this);
endfunction
```

```
  function void write (Ex6_transaction t);
    $cast(out_txn, t.clone());
    out_txn.correct = 1;
    case (t.mode)
      ADD: out_txn.result = t.value1 + t.value2;
      SUB: begin
    if (t.value2 > t.value1) begin
      out_txn.correct = 0;
    end
    out_txn.result = t.value1 - t.value2;
      end

      MUL: out_txn.result = t.value1 * t.value2;
      DIV: if (t.value2 == 0) begin
          out_txn.correct = 0;
          out_txn.result = 0;
        end
        else
          out_txn.result = t.value1 / t.value2;
    endcase

    transaction_coverage.sample();
    results_ap.write(out_txn);
  endfunction
  function void report_phase(uvm_phase phase);
    real pct;
    int unsigned covered;
    int unsigned total;
    pct = transaction_coverage.get_coverage(covered, total);
    $display("Coverage of Req: covered = %0d, total = %0d (%5.2f%%)", covered, total , pct);
    $display("Coverage of Instance %e", transaction_coverage.get_coverage());
  endfunction
endclass
```

The result of the comparison will be shown in the comparison class.

```
class Ex6_comparator extends uvm_component;
  `uvm_component_utils(Ex6_comparator)
  uvm_analysis_export #(Ex6_transaction) axp_in;
  uvm_analysis_export #(Ex6_transaction) axp_out;
  uvm_tlm_analysis_fifo #(Ex6_transaction) expfifo;
  uvm_tlm_analysis_fifo #(Ex6_transaction) outfifo;

  function new(string name, uvm_component parent);
    super.new(name, parent);
  endfunction
  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    axp_in = new("axp_in", this);
    axp_out = new("axp_out", this);
    expfifo = new("expfifo", this);
    outfifo = new("outfifo", this);
  endfunction
  function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    axp_in.connect(expfifo.analysis_export);
    axp_out.connect(outfifo.analysis_export);
  endfunction
  task run_phase(uvm_phase phase);
    Ex6_transaction exp_tr, out_tr;
    outfifo.get(out_tr);
    forever begin
      `uvm_info("Ex6_comparator run task",
                "WAITING for expected output", UVM_DEBUG)
      expfifo.get(exp_tr);

      `uvm_info("Ex6_comparator run task",
                "WAITING for actual output", UVM_DEBUG)
      outfifo.get(out_tr);
      if (out_tr.compare(exp_tr)) begin
        PASS();
        `uvm_info ("PASS ", $sformatf("Actual value is =%s   Expected value is =%s \n", out_tr.output2string(),
        exp_tr.output2string()), UVM_LOW)
      end
```

```
      else begin
        ERROR();
        `uvm_info ("ERROR ", $sformatf("Actual value is=%s    Expected value is =%s \n",
        out_tr.output2string(), exp_tr.output2string()), UVM_LOW)
      end
    end
  endtask

  int VECT_CNT, PASS_CNT, ERROR_CNT;

  function void report_phase(uvm_phase phase);
    super.report_phase(phase);

    if (VECT_CNT && !ERROR_CNT)
      `uvm_info(get_type_name(),
                $sformatf(
                "\n\n\n*** TEST PASSED - %0d vectors ran, %0d vectors passed ***\n",
                VECT_CNT, PASS_CNT), UVM_LOW)
    else
          `uvm_info(get_type_name(),
      $sformatf(
                "\n\n\n*** TEST FAILED - %0d vectors ran, %0d vectors passed, %0d vectors failed ***\n",
                VECT_CNT, PASS_CNT, ERROR_CNT), UVM_LOW)
  endfunction
  function void PASS();
    VECT_CNT++;
    PASS_CNT++;
  endfunction
  function void ERROR();
    VECT_CNT++;
    ERROR_CNT++;
  endfunction
endclass
```

Analysis is very important to ensure the connection between different components.

```
class Ex6_analysis extends uvm_component;

  `uvm_component_utils(Ex6_analysis)
  uvm_analysis_export #(Ex6_transaction) _export;
  uvm_analysis_export #(Ex6_transaction) _import;);

  super.build_phase(phase);
    _export = new("_export", this);
    _fifo_in = new("_fifo_in", this);
    _import = new("_import", this);
    _fifo_out = new("_fifo_out", this);
  endfunction: build_phase

  function void connect_phase(uvm_phase phase);
    _export.connect(_fifo_in.analysis_export);
    _import.connect(_fifo_out.analysis_export);
  endfunction: connect_phase

  task run_phase(uvm_phase phase);
    Ex6_transaction item;
    forever begin
      _fifo_in.get(item);
      _fifo_out.get(item);
    end
  endtask
endclass: Ex6_analysis

  uvm_tlm_analysis_fifo #(Ex6_transaction) _fifo_in;
  uvm_tlm_analysis_fifo #(Ex6_transaction) _fifo_out;

  function new(string name, uvm_component parent);
    super.new(name, parent);
  endfunction

  function void build_phase(uvm_phase phase
```

Also we add all the necessary packages into the pkg class as follows-

```
package Ex6_pkg;

  `include "uvm_macros.svh"
  import uvm_pkg::*;
  import types_pkg::*;
  `include "Ex6_config.svh"
  `include "Ex6_agent_config.svh"

  `include "Ex6_transaction.svh"
  `include "Ex6_sequence.svh"

  typedef uvm_sequencer #( Ex6_transaction ) Ex6_sequencer;

  `include "Ex6_driver.svh"
  `include "Ex6_monitor.svh"
  `include "Ex6_agent.svh"
  `include "Ex6_analysis.svh"
  `include "Ex6_predictor.svh"
  `include "Ex6_comparator.svh"
  `include "Ex6_scoreboard.svh"
  `include "Ex6_env.svh"
  `include "Ex6_test.svh"
  `include "Ex6_test_zero.svh"
  `include "Ex6_test_lt_zero.svh"

endpackage: Ex6_pkg
```

3. Simulation Result:

Here, I am going to discuss about the result. The output log file is given already. From the result we can see that the ALU is showing the output as expected. The actual value and the expected value is same. Each of this is loaded in every100 ns. For the further improvement, we could have added more operations like AND, XOR, OR gate and so on and also may an extra instruction unit. Although, this ALU is simple but it works well as expected and good enough for the verification.

And the finally the simulation result is –

```
vlog -f compile_questa_sv.f
# QuestaSim vlog 10.4 Compiler 2014.12 Dec  2 2014
# Start time: 13:54:13 on May 13,2016
# vlog -reportprogress 300 -f compile_questa_sv.f
# -- Compiling package types_pkg
# -- Compiling module Ex6_dut
# -- Compiling module Ex6_clock_driver
# -- Compiling interface Ex6_interface
# -- Importing package types_pkg
# -- Compiling package Ex6_pkg
# ** Note: (vlog-2286) Using implicit +incdir+/soft/Mentor/Questa_10.4/questasim/uvm-
1.1d/../verilog_src/uvm-1.1d/src from import uvm_pkg
# -- Importing package mtiUvm.uvm_pkg (uvm-1.1d Built-in)
# -- Compiling module top
# -- Importing package Ex6_pkg
#
# Top level modules:
#         top
# End time: 13:54:14 on May 13,2016, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
vsim +UVM_TESTNAME=Ex6_test -do "run -all" top
# vsim
# Start time: 13:54:47 on May 13,2016
# ** Note: (vsim-3812) Design is being optimized...
# Loading sv_std.std
# Loading work.types_pkg(fast)
# Loading mtiUvm.uvm_pkg
# Loading work.Ex6_pkg(fast)
# Loading work.top(fast)
# Loading work.Ex6_dut(fast)
# Loading work.Ex6_clock_driver(fast)
# Loading mtiUvm.questa_uvm_pkg(fast)
```

```
# Loading work.Ex6_interface(fast)
# Loading /soft/Mentor/Questa_10.4/questasim/uvm-1.1d/linux/uvm_dpi.so
# run -all
# ----------------------------------------------------------------
# UVM-1.1d
# (C) 2007-2013 Mentor Graphics Corporation
# (C) 2007-2013 Cadence Design Systems, Inc.
# (C) 2006-2013 Synopsys, Inc.
# (C) 2011-2013 Cypress Semiconductor Corp.
# ----------------------------------------------------------------
#
#   ***********       IMPORTANT RELEASE NOTES         ************
#
#   You are using a version of the UVM library that has been compiled
#   with `UVM_NO_DEPRECATED undefined.
#   See http://www.eda.org/svdb/view.php?id=3313 for more details.
#
#   You are using a version of the UVM library that has been compiled
#   with `UVM_OBJECT_MUST_HAVE_CONSTRUCTOR undefined.
#   See http://www.eda.org/svdb/view.php?id=3770 for more details.
#
#      (Specify +UVM_NO_RELNOTES to turn off this notice)
#
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(215) @ 0: reporter
[Questa UVM] QUESTA_UVM-1.2.2
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(217) @ 0: reporter
[Questa UVM]  questa_uvm::init(+struct)
# UVM_INFO @ 0: reporter [RNTST] Running test Ex6_test...
# UVM_INFO @ 0: reporter [UVMTOP] UVM testbench topology:
# ----------------------------------------------------------------
# Name                    Type                    Size  Value
# ----------------------------------------------------------------
# uvm_test_top              Ex6_test                -     @464
#   m_env                   Ex6_env                 -     @484
#     Ex6_agent             Ex6_agent               -     @492
#       m_driver            Ex6_driver              -     @525
#         ap                uvm_analysis_port       -     @551
#         rsp_port          uvm_analysis_port       -     @542
#         seq_item_port     uvm_seq_item_pull_port  -     @533
#       m_monitor           Ex6_monitor             -     @517
#         ap                uvm_analysis_port       -     @684
#       m_sequencer         uvm_sequencer           -     @560
#         rsp_export        uvm_analysis_export     -     @568
#         seq_item_export   uvm_seq_item_pull_imp   -     @674
#         arbitration_queue array                   0     -
#         lock_queue        array                   0     -
```

```
#       num_last_reqs      integral                32    'd1
#       num_last_rsps      integral                32    'd1
#    Ex6_analysis          Ex6_analysis              -     @500
#      _export             uvm_analysis_export       -     @699
#      _fifo_in            uvm_tlm_analysis_fifo #(T)  -     @708
#        analysis_export   uvm_analysis_imp          -     @752
#        get_ap            uvm_analysis_port         -     @743
#        get_peek_export   uvm_get_peek_imp          -     @725
#        put_ap            uvm_analysis_port         -     @734
#        put_export        uvm_put_imp               -     @716
#      _fifo_out           uvm_tlm_analysis_fifo #(T)  -     @770
#        analysis_export   uvm_analysis_imp          -     @814
#        get_ap            uvm_analysis_port         -     @805
#        get_peek_export   uvm_get_peek_imp          -     @787
#        put_ap            uvm_analysis_port         -     @796
#        put_export        uvm_put_imp               -     @778
#      _import             uvm_analysis_export       -     @761
#    sb                    Ex6_scoreboard            -     @508
#      axp_in              uvm_analysis_export       -     @823
#      axp_out             uvm_analysis_export       -     @832
#      cmp                 Ex6_comparator            -     @858
#        axp_in            uvm_analysis_export       -     @866
#        axp_out           uvm_analysis_export       -     @875
#        expfifo           uvm_tlm_analysis_fifo #(T)  -     @884
#          analysis_export uvm_analysis_imp          -     @928
#          get_ap          uvm_analysis_port         -     @919
#          get_peek_export uvm_get_peek_imp          -     @901
#          put_ap          uvm_analysis_port         -     @910
#          put_export      uvm_put_imp               -     @892
#        outfifo           uvm_tlm_analysis_fifo #(T)  -     @937
#          analysis_export uvm_analysis_imp          -     @981
#          get_ap          uvm_analysis_port         -     @972
#          get_peek_export uvm_get_peek_imp          -     @954
#          put_ap          uvm_analysis_port         -     @963
#          put_export      uvm_put_imp               -     @945
#      prd                 Ex6_predictor             -     @841
#        analysis_imp      uvm_analysis_imp          -     @849
#        results_ap        uvm_analysis_port         -     @990
# ----------------------------------------------------------------
#
# UVM_INFO Ex6_comparator.svh(45) @ 30: uvm_test_top.m_env.sb.cmp [PASS ] Actual
value is =Result=3060   Expected value is =Result=3060
#
# UVM_INFO Ex6_comparator.svh(45) @ 50: uvm_test_top.m_env.sb.cmp [PASS ] Actual
value is =Result=0   Expected value is =Result=0
#
```

```
# UVM_INFO Ex6_comparator.svh(45) @ 70: uvm_test_top.m_env.sb.cmp [PASS ] Actual
value is =Result=65535   Expected value is =Result=65535
#
# UVM_INFO Ex6_comparator.svh(45) @ 90: uvm_test_top.m_env.sb.cmp [PASS ] Actual
value is =Result=0   Expected value is =Result=0
#
# UVM_INFO Ex6_comparator.svh(45) @ 110: uvm_test_top.m_env.sb.cmp [PASS ] Actual
value is =Result=158   Expected value is =Result=158
#
# UVM_INFO Ex6_comparator.svh(45) @ 130: uvm_test_top.m_env.sb.cmp [PASS ] Actual
value is =Result=156   Expected value is =Result=156
#
# UVM_INFO Ex6_comparator.svh(45) @ 150: uvm_test_top.m_env.sb.cmp [PASS ] Actual
value is =Result=0   Expected value is =Result=0
#
# UVM_INFO Ex6_comparator.svh(45) @ 170: uvm_test_top.m_env.sb.cmp [PASS ] Actual
value is =Result=2870   Expected value is =Result=2870
#
# UVM_INFO Ex6_comparator.svh(45) @ 190: uvm_test_top.m_env.sb.cmp [PASS ] Actual
value is =Result=42   Expected value is =Result=42
#
# UVM_INFO Ex6_comparator.svh(45) @ 210: uvm_test_top.m_env.sb.cmp [PASS ] Actual
value is =Result=60   Expected value is =Result=60
#
# UVM_INFO Ex6_comparator.svh(45) @ 230: uvm_test_top.m_env.sb.cmp [PASS ] Actual
value is =Result=525   Expected value is =Result=525
#
# UVM_INFO Ex6_comparator.svh(45) @ 250: uvm_test_top.m_env.sb.cmp [PASS ] Actual
value is =Result=0   Expected value is =Result=0
#
# UVM_INFO Ex6_comparator.svh(45) @ 270: uvm_test_top.m_env.sb.cmp [PASS ] Actual
value is =Result=153   Expected value is =Result=153
#
# UVM_INFO Ex6_comparator.svh(45) @ 290: uvm_test_top.m_env.sb.cmp [PASS ] Actual
value is =Result=8   Expected value is =Result=8
#
# UVM_INFO Ex6_comparator.svh(45) @ 310: uvm_test_top.m_env.sb.cmp [PASS ] Actual
value is =Result=65485   Expected value is =Result=65485
#
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1268) @ 310: reporter
[TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO Ex6_comparator.svh(63) @ 310: uvm_test_top.m_env.sb.cmp [Ex6_comparator]
#
#
# *** TEST PASSED - 15 vectors ran, 15 vectors passed ***
#
```

```
# Coverage of Req: covered = 9, total = 10 (88.89%)
# Coverage of Instance 8.888889e+01
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   21
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Ex6_comparator]     1
# [PASS ]    15
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [UVMTOP]     1
# ** Note: $finish    : /soft/Mentor/Questa_10.4/questasim/linux/../verilog_src/uvm-
1.1d/src/base/uvm_root.svh(430)
#    Time: 310 ns  Iteration: 68  Instance: /top
# 1
# Break in Task uvm_pkg/uvm_root::run_test at
/soft/Mentor/Questa_10.4/questasim/linux/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line
430
quit –sim
```

4. References:

1. https://en.wikipedia.org/wiki/PicoBlaze
2. http://www.circuitstoday.com/basics-of-microcontrollers