**Министерство науки и высшего образования Российской Федерации**
**Федеральное государственное автономное образовательное учреждение**
**высшего образования**
**«Московский государственный технический университет имени Н.Э. Баумана**
**(национальный исследовательский университет)»**
**(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

# Отчет
# по лабораторной работе № 3

по дисциплине «Организация ЭВМ и систем»

Тема: «Написание последовательностей и тестов»

Студент группы ИУ6-73Б _____ В.К. Залыгин
(Подпись, дата) (И.О. Фамилия)

Преподаватель _____ М.П. Калитвенцев
(Подпись, дата) (И.О. Фамилия)

2025 г.

## Цель работы

Освоить практические приёмы построения верификационного окружения на SystemVerilog: научиться рандомизировать транзакции и ограничивать их констрейнами, организовывать параллельные процессы и их синхронизацию, а также разработать последовательности (sequences) и тесты (tests), которые генерируют входные воздействия на DUT по заданным сценариям и запускаются как отдельные тест-кейсы и в составе регрессии.

## Теоретические сведения

SystemVerilog позволяет строить верификационное окружение на основе объектных транзакций, которые генерируются автоматически. Для этого поля классов помечаются как rand и получают случайные значения при вызове randomize(). Чтобы случайные данные оставались корректными и «осмысленными» для DUT, используются констрейны (constraint), задающие диапазоны и зависимости между полями (в том числе условные ограничения через импликацию или if-else).

Так как в тестбенче присутствуют блокирующие операции (ожидание событий, чтение из mailbox и т.п.), применяется параллельное выполнение процессов с помощью fork...join, а для синхронизации используются стандартные примитивы взаимодействия: события (event), семафоры (semaphore) и почтовые ящики (mailbox). Это позволяет одновременно запускать генерацию воздействий, ожидание реакции и сбор статистики без взаимной блокировки.

Генерация входных воздействий выполняется через последовательности (sequences): в методе body() создаются и рандомизируются транзакции, которые затем передаются в агент через sequencer. Для сложных сценариев применяются виртуальные последовательности, которые не формируют транзакции напрямую, а управляют запуском других последовательностей на разных секвенсорах.

## Выполнение работы

В задании 1 был реализован класс, описывающий инструкцию для АЛУ, с рандомизируемыми полями (код операции, два операнда и бит подмены) и заданными констрейнами на диапазоны значений и зависимость операнда от бита подмены. Код приведен в листинге 1. Запуск задания показан на рисунке 1.

Листинг 1 – Результат выполнения задания 1.

```
/*
  Напишите класс, содержащий данные об инструкции для АЛУ
  Поля класса следующие: трехразрядный код операции,
целочисленные операнды 1 и 2, бит подмены операнда
  Все поля рандомизируемые. Условия рандомизации следующие:
  1. Код операции не больше 5, т.к. в этом АЛУ всего 6 операций.
  2. Операнды не могут превышать (2**16)-1 по модулю.
  3. Если выставлен бит подмены операнда, тогда младшие 8 бит
операнда 1 должны содержать число, меньшее 128.
*/

/* Your classes here */

class alu_data;

  rand bit [2:0] opcode;
  rand int op1, op2;
  rand bit oprep;

  constraint opcode_constr {
    opcode <= 5;
  }

  constraint opconstr {
    op1 < 65536;
    op1 > -65536;
    op2 < 65535;
    op2 > -65536;
  }

  constraint oprepconstr {
    oprep == 1'b1 -> (op1 % 256) < 128;
  }

endclass : alu_data

module constraints_task;

  initial begin
```

```
    alu_data alu_data_inst = new ();
    void'(alu_data_inst.randomize());

    $display("opcode: %d", alu_data_inst.opcode);
    $display("op1:    %d,    op2:    %d",    alu_data_inst.op1,
alu_data_inst.op2);
    $display("oprep: %d", alu_data_inst.oprep);

    $finish;
  end

endmodule : constraints_task
```

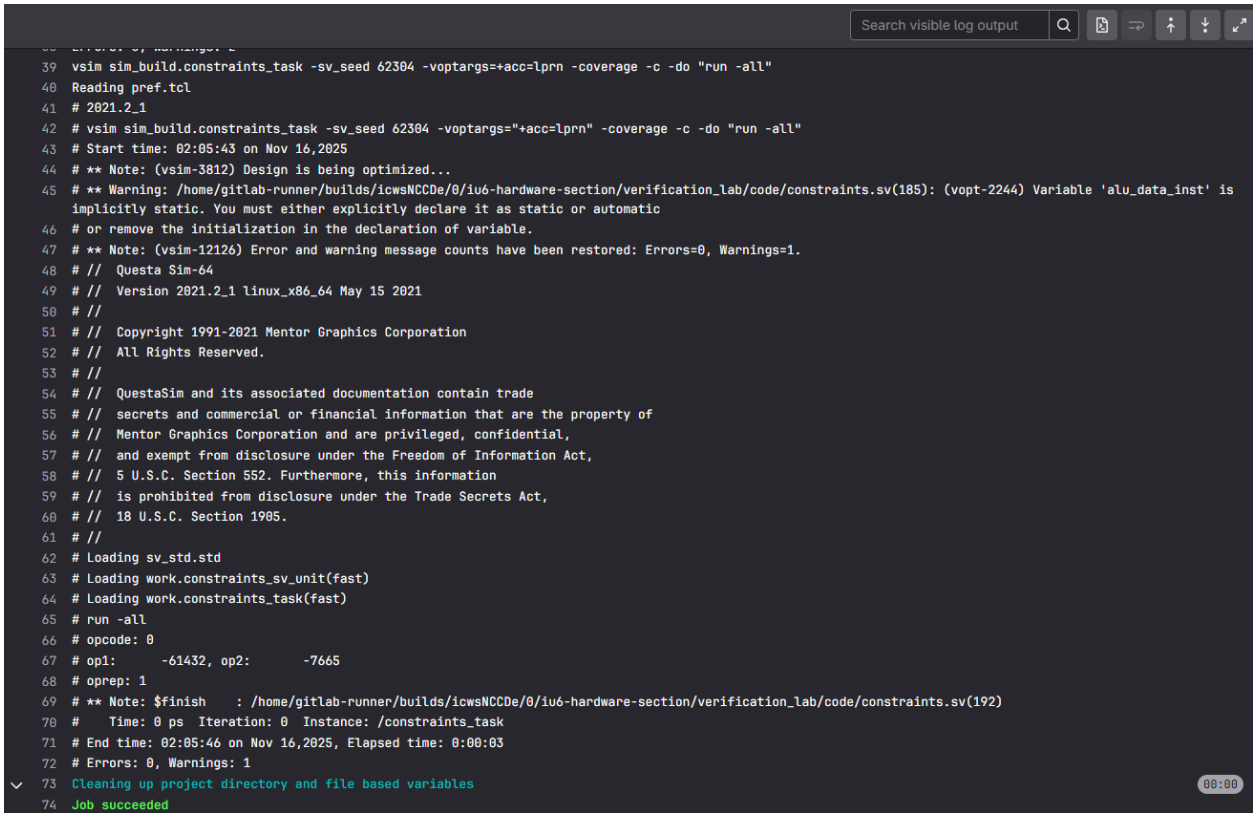


Рисунок 1 – Запуск задания 1

В задании 2 была реализована система из трёх параллельных процессов: два циклических процесса изменяют общий счётчик с разными задержками (инкремент/декремент), а третий процесс ожидает результаты от первых двух и выводит число инкрементов и декрементов. Код приведен в листинге 2. Результат запуска задания показан на рисунке 2.

Листинг 2 – Результат выполнения задания 2

```
/*
  Есть целочисленная переменная, представляющая собой счетчик. Начальное значение равно 5.
  Два параллельных процесса работают с этим счетчиком. Оба процесса циклические.
  Первый процесс ждет N ед. времени и инкрементирует счетчик. Второй процесс ждет 2N ед. времени и декрементирует счетчик.
  Первый процесс должен завершиться, когда счетчик достигнет значения 10. По завершении кол-во инкрементов должно быть передано процессу 3.
  Второй процесс должен завершиться, когда счетчик достигнет значения 0 По завершении кол-во декрементов должно быть передано процессу 3.
  Процесс 3 ждет, пока не получит данные от процессов 1 и 2, после чего выводит их на экран и завершается.
  Реализуйте вышеописанную конструкцию.
*/

typedef enum {INC=0, DEC=1} optype;

class event_c;

  optype optype_inst;
  int value;

  function new(optype optype_inst, int value);
    this.optype_inst = optype_inst;
    this.value = value;
  endfunction

endclass : event_c

module process_interaction_task;
  initial begin
    int counter = 5;
    int N = 100;
    mailbox #(event_c) mbx = new();
    semaphore sem = new(1);

    fork
      begin : proc1
        int inc = 0;
        event_c event_inst;

        while (counter != 10) begin
          #N sem.get();
          counter++;
          inc++;
          sem.put();
        end
```

```
        event_inst = new(INC, inc);
        mbx.put(event_inst);
      end

      begin : proc2
        int dec = 0;
        event_c event_inst;

        while (counter != 0) begin
          #(2*N) sem.get();
          counter--;
          dec++;
          sem.put();
        end

        event_inst = new(DEC, dec);
        mbx.put(event_inst);
      end

      begin : proc3
        int inc;
        int dec;
        event_c event_inst;

        mbx.get(event_inst);
        if (event_inst.optype_inst == INC)
          inc = event_inst.value;
        else
          dec = event_inst.value;
        mbx.get(event_inst);
        if (event_inst.optype_inst == INC)
          inc = event_inst.value;
        else
          dec = event_inst.value;

        $display("inc: %d", inc);
        $display("dec: %d", dec);
      end
    join

    $finish;
  end
endmodule
```

В задании 3 в уже созданные в предыдущей лабораторной транзакции для агентов была добавлена рандомизация (rand) и введены констрейны, ограничивающие значения полей диапазонами, корректными для DUT. Итоговый код приведен в листинге 3.

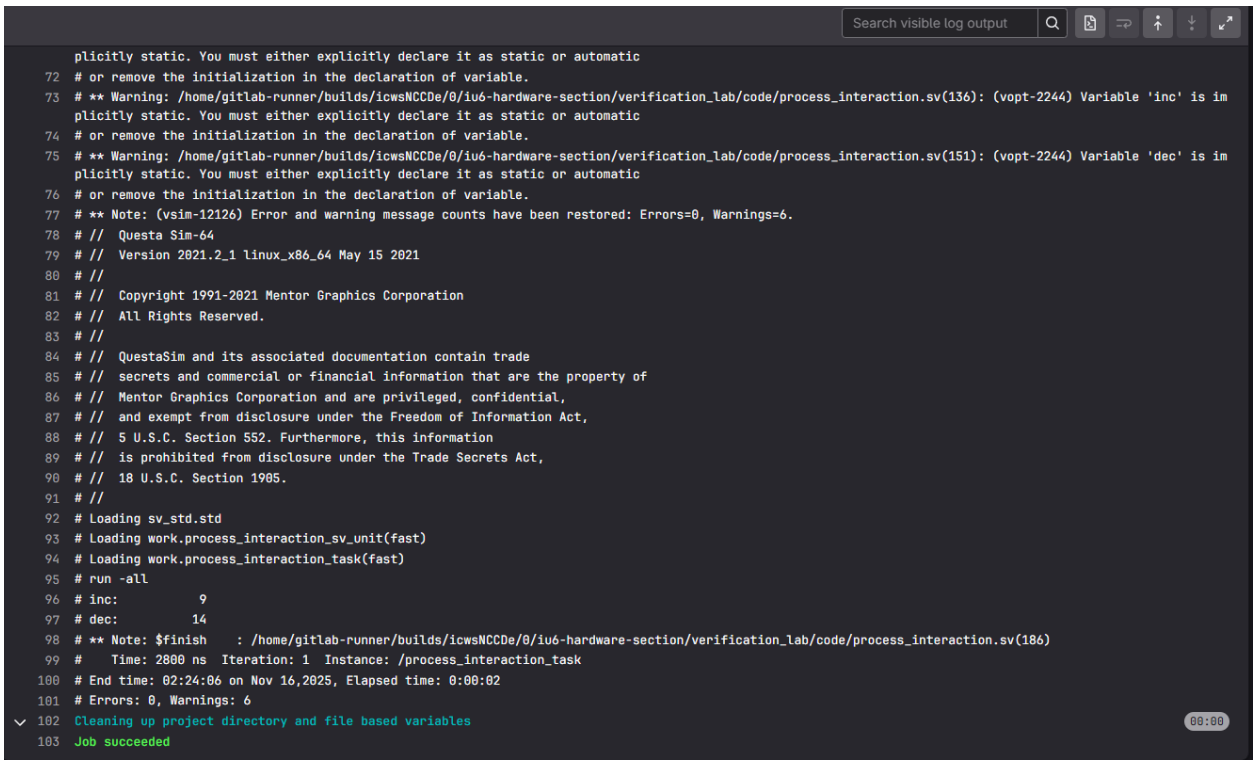


Рисунок 2 – Запуск задания 2

Листинг 3 – Результат выполнения задания 3

```
////////////////////////////////////////////////////////////////////
//////////////////////////
// Company: Bauman Moscow State Technical University
// Engineer:
//
// Create date: 09/02/2025
// Project Name:
// Module Name:
// Target Devices:
// Tool Versions:
// Description:
////////////////////////////////////////////////////////////////////
//////////////////////////

class mealy_in_transaction extends base_transaction;

  bit in_valid;
  rand bit [3:0] in_data;

  constraint in_data_cstr {
     in_data inside { 4'b0001, 4'b0010, 4'b0011, 4'b0100,
4'b0101, 4'b0110, 4'b0111, 4'b1000, 4'b1101, 4'b1110 };
  }

   extern virtual function bit compare(base_transaction
rhs);
  extern virtual function void copy(base_transaction rhs);
  extern virtual function void print();
```

```
endclass : mealy_in_transaction

function                                                    bit
mealy_in_transaction::compare(base_transaction rhs);
  mealy_in_transaction that;
  if (!$cast(that, rhs)) begin
       $display("FATAL:  compare(rhs)  is  not  of  type
mealy_in_transaction!");
    $finish;
  end

  compare = 1;
  compare &= this.in_valid == that.in_valid;
  compare &= this.in_data == that.in_data;
endfunction : compare

function  void  mealy_in_transaction::copy(base_transaction
rhs);
  mealy_in_transaction that;
  if (!$cast(that, rhs)) begin
       $display("FATAL:  compare(rhs)  is  not  of  type
mealy_in_transaction!");
    $finish;
  end

  this.in_valid = that.in_valid;
  this.in_data = that.in_data;
endfunction : copy

function void mealy_in_transaction::print();
  $display("--MEALY_IN_TRANSACTION--");
  $display("in_valid: %d", this.in_valid);
  $display("in_data: %d", this.in_data);
  $display("------------------------");
endfunction : print


class mealy_out_transaction extends base_transaction;

  bit [2:0] out_data;

   extern  virtual  function  bit  compare(base_transaction
rhs);
  extern virtual function void copy(base_transaction rhs);
  extern virtual function void print();

endclass : mealy_out_transaction

function                                                    bit
mealy_out_transaction::compare(base_transaction rhs);
  mealy_out_transaction that;
```

```
      if (!$cast(that, rhs)) begin
           $display("FATAL:  compare(rhs)  is  not  of  type
    mealy_out_transaction!");
        $finish;
      end

      compare = this.out_data == that.out_data;
    endfunction : compare

    function void mealy_out_transaction::copy(base_transaction
    rhs);
      mealy_out_transaction that;
      if (!$cast(that, rhs)) begin
           $display("FATAL:  compare(rhs)  is  not  of  type
    mealy_out_transaction!");
        $finish;
      end

      this.out_data = that.out_data;
    endfunction : copy

    function void mealy_out_transaction::print();
      $display("--MEALY_OUT_TRANSACTION--");
      $display("out_data: %d", this.out_data);
      $display("------------------------");
    endfunction : print
```

В задании 4 были разработаны последовательности для всех входных агентов, наследованные от base_seq, реализующие генерацию транзакций в body() и их отправку на соответствующие sequencer. Код созданной последовательности содержится в листинге 4.

Листинг 4 – Результат выполнения задания 4

```
//////////////////////////////////////////////////////////////////
//////////////////
// Company: Bauman Moscow State Technical University
// Engineer:
//
// Create date: 11/02/2025
// Project Name:
// Module Name:
// Target Devices:
// Tool Versions:
// Description: Mealy in sequence.
//////////////////////////////////////////////////////////////////
//////////////////
```

```
class mealy_in_seq extends base_seq;

  protected mealy_in_transaction mealy_in_item;

  bit in_valid          = 1; // бит валидности входных данных
  bit force_data            = 0; // разрешение на выставление
нерандомизированного значения
  bit [3:0] forced_data = 0; // нерандомизированние значение

  extern protected virtual task body();

  extern protected function void prepare_item();

endclass : mealy_in_seq

task mealy_in_seq::body();
  prepare_item();
  seqr.driver_req(mealy_in_item);
endtask : body

function void mealy_in_seq::prepare_item();
  mealy_in_item = new();

  if (!mealy_in_item.randomize()) begin
    $display("ERROR: Failed to randomize mealy_in_item!");
    $finish;
  end

  mealy_in_item.in_valid = this.in_valid;
  //        $display("[MEALY_IN_SEQ]        in_valid        %b",
mealy_in_item.in_valid);
  if (force_data == 1) begin
    mealy_in_item.in_data = this.forced_data;
  end
endfunction : prepare_item
```

В задании 5 по спецификации DUT был определён набор проверяемых функций и составлен полный список тестовых сценариев, покрывающий эти функции, включая комбинированные сценарии. Описание функций и сценариев приведено в таблице 1.

В задании 6 для сценариев, требующих сложного управления генерацией, были разработаны виртуальные последовательности и/или добавлены настроечные параметры в обычные последовательности для управления типами транзакций и их количеством.

Таблица 1 – Описание тестов

| Имя теста | Сценарий теста |
|---|---|
| base test | Базовый тест, инициализирующий окружение и запускающий основные последовательности.<br>1. Запуск последовательности тактового сигнала<br>2. Запуск последовательности сброса.<br>3. Конец. |
| walk test | Автомат корректно переходит по всем ветвям.<br>1. То же, что в базовом тесте.<br>2. Запуск последовательности из N транзакций данных с разрешающим сигналом для перехода по всем ветвям.<br>3. Конец. |
| ignore test | Автомат не реагирует на входной сигнал при опущенном разрешающем сигнале.<br>1. То же, что в базовом тесте.<br>2. Запуск последовательности из 1 транзакции данных без разрешающего сигнала.<br>3. Конец. |

Т.к. тестируемое устройство достаточно простое, в тестировании можно просто полностью положиться на рандомизацию входных последовательностей, предварительно увеличив вероятности того, что в входных данных будут все 1 и все 0 в констрейнах транзакций. Код транзакций приведен в листинге 3.

В задании 7 были реализованы тесты для каждого сценария как классы-наследники base_test с переопределением run_main(), где запускаются необходимые последовательности и выполняются ожидания по тактам/событиям. Код тестов содержится в листинге 5.

Листинг 5 – Результат выполнения задания 7, текст файла encoder_test.sv

```
//////////////////////////////////////////////////////////////////
//////////////////
// Company: Bauman Moscow State Technical University
// Engineer:
//
```

```
// Create date: 15/03/2025
// Project Name:
// Module Name:
// Target Devices:
// Tool Versions:
// Description:
//////////////////////////////////////////////////////////////////
//////////////////

class walk_test extends base_test;

  all_states_vseq seq;

  extern function new(string name = "walk_test");

  extern protected virtual task run_main();

endclass : walk_test

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

task walk_test::run_main();
  seq = new();
  seq.seqr = env0.mealy_in_sig_agent.sequencer;
  seq.start(null);
endtask : run_main

class ignore_test extends base_test;

  mealy_in_seq seq;

  extern function new(string name = "ignore_test");

  extern protected virtual task run_main();

endclass : ignore_test

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

task ignore_test::run_main();
  seq = new();
  seq.in_valid = 0;
  seq.start(env0.mealy_in_sig_agent.sequencer);
endtask : run_main
```

В задании 8 созданные тесты были подключены к тестбенчу верхнего уровня tb_top.sv: добавлены таски запуска тестов по имени и добавлены

вызовы этих тасков в общий селектор тестов. Итоговые изменения приведены в листинге 6.

Листинг 6 – Результат выполнения задания 8, текст файла tb_top.sv

```
//////////////////////////////////////////////////////////////////////
//////////////////
// Company: Bauman Moscow State Technical University
// Engineer:
//
// Create date: 06/02/2025
// Project Name:
// Module Name:
// Target Devices:
// Tool Versions:
// Description: Top-level Testbench
//////////////////////////////////////////////////////////////////////
//////////////////

module tb_top();

  // Interface: dut_intf
  // Main DUT interface.
  env_if         dut_intf ();
  // Variable: dut
  // Wrapper module for RTL Device-Under-Test.
  tb_dut_wrapper dut (dut_intf);
  // Component: tests
  // Associative all possible test names.
  bit            tests [string];
  // Component: test_names
  // Names of the tests to run.
  string         test_names [$];

  initial begin
    init_tests();

    // Get test names
    foreach (tests[t_name])
      check_plusarg_test_name(t_name);

    // Launch all the tests, one by one.
    foreach(test_names[i]) begin
      case (test_names[i])
        "test_base": run_base_test("test_base");
        // Add your test names here
        "walk_test": run_walk_test("walk_test");
        "ignore_test": run_ignore_test("ignore_test");
      endcase
    end

    $stop; // End the simulation
```

```
  end

  function void init_tests();
    tests["test_base"] = 1;

    // Rest of the tests here
    tests["walk_test"] = 1;
    tests["ignore_test"] = 1;
  endfunction : init_tests

  task run_base_test(string t_name);
    base_test test;

    test = new(t_name);    // Create class object
    test.vif = dut_intf;   // Set vif
    test.build();          // Build test
    test.run();            // Run test

    tests[t_name] = test.test_res;
  endtask : run_base_test

  // Other test run methods go here
  task run_walk_test(string t_name);
    walk_test test;

    test = new(t_name);    // Create class object
    test.vif = dut_intf;   // Set vif
    test.build();          // Build test
    test.run();            // Run test

    tests[t_name] = test.test_res;
  endtask : run_walk_test

  task run_ignore_test(string t_name);
    ignore_test test;

    test = new(t_name);    // Create class object
    test.vif = dut_intf;   // Set vif
    test.build();          // Build test
    test.run();            // Run test

    tests[t_name] = test.test_res;
  endtask : run_ignore_test

  // Don't touch methods below.

  function void check_plusarg_test_name(string name);
    if ($test$plusargs(name)) begin
      if (check_test_name(name)) begin
        test_names.push_back(name);
        $display("Test %s is found in plusargs, added to queue.",
name);
```

```
      end
    end
  endfunction : check_plusarg_test_name

  function bit check_test_name(string name);
    if (!tests.exists(name)) begin
      $display("ERROR: There is no test with name %s, skipping.",
name);
      return 0;
    end
    return 1;
  endfunction : check_test_name

endmodule : tb_top
```

В задании 9 корректность выполнения тестовых сценариев была проверена по временным диаграммам Modelsim, подтверждая, что тесты подают на DUT именно предусмотренные сценариями воздействия.

В задании 10 все тесты были добавлены в регрессию через Makefile: для каждого теста создана отдельная цель запуска и добавление в общий таргет run_all_iterations, после чего обеспечен последовательный запуск всех тестов командой make regression. Фрагмент Makefile приведён в листинге 7.

Листинг 7 – Результат выполнения задания 10, текст файла Makefile

```
##################################################################
##################
# Company: Bauman Moscow State Technical University
# Engineer: Maksim Kalitventsev (maksim.kalitventsev@yandex.ru)
#
# Create date: 20/03/2025
# Project Name:
# Module Name:
# Target Devices:
# Tool Versions: vsim.
# Description: This is base makefile for simulating with
QuestaSim.
##################################################################
##################

PWD       := $(shell pwd)
MAKEFILE  := $(lastword $(MAKEFILE_LIST))
RAND_SEED := $(strip $(shell head -1 /dev/urandom | od -N 2 -D
-A n | awk '{print $1}'))

# Randomization seed
SEED         ?= random
```

```
# Name of Questa worklib - DO NOT NAME THIS 'WORK'!!!
WORKLIB_NAME ?= sim_build
# Name of the file to run
FILENAME     ?= data_types
# Name of the top-level testbench module
TB_NAME      ?= tb_top
# Name of the top-level RTL module
DUT          ?= mealy_machine_5state_var6_wrap

UNIT_HOME = $(PWD)

CODE_HOME = $(UNIT_HOME)/code

RTL_HOME  = $(UNIT_HOME)/rtl
LAB1_HOME = $(UNIT_HOME)/labs/lab1
LAB2_HOME = $(UNIT_HOME)/labs/lab2
HW_HOME   = $(UNIT_HOME)/homework
EX_HOME   = $(UNIT_HOME)/labs/example

VLOG_OPTS += -incr
VLOG_OPTS += -sv
VLOG_OPTS += -timescale 1ns/1ps
VLOG_OPTS += -cover est
VLOG_OPTS += +fcover

CODE_VLOG_OPTS += $(CODE_HOME)/$(FILENAME).sv
CODE_VLOG_OPTS += +incdir+$(CODE_HOME)
CODE_VLOG_OPTS += -work $(WORKLIB_NAME)

TEST_VLOG_OPTS += -work $(WORKLIB_NAME)/$(DUT)

VSIM_OPTS  = -voptargs=+acc=lprn
VSIM_OPTS += -coverage

CONSOLE_OPTS = -c

TB_OPTS =
include $(UNIT_HOME)/files_tb.mk

RTL_OPTS =
include $(UNIT_HOME)/files_rtl.mk

BASE_TEST_OPTS = +test_base

ifeq ($(SEED), random)
     SET_SEED = -sv_seed $(RAND_SEED)
else
     SET_SEED = -sv_seed $(SEED)
endif

USER_DEF_OPTS =
```

```
run_log_parser = \
     sh script/log_parser.sh transcript
$(WORKLIB_NAME)/$(DUT)/stat.log 1

run_log_parser_regression = \
     sh script/log_parser.sh transcript
$(WORKLIB_NAME)/$(DUT)/stat.log 0

run_stat_parser = \
sh script/stat_parser.sh $(WORKLIB_NAME)/$(DUT)/stat.log

merge_cov = \
     vcover new $(WORKLIB_NAME)/$(DUT)/cov_base.ucdb; \
     vcover merge $(WORKLIB_NAME)/$(DUT)/cov_base.ucdb
run_cov.ucdb $(WORKLIB_NAME)/$(DUT)/cov_base.ucdb; \
     rm run_cov.ucdb

run_cov_report = \
     bash script/cov_check.sh
$(WORKLIB_NAME)/$(DUT)/cov_base.ucdb $(DUT)

code: $(WORKLIB_NAME) run_code

task: $(WORKLIB_NAME) run_task

$(WORKLIB_NAME):
     vlib $(WORKLIB_NAME)

worklib_dut: $(WORKLIB_NAME)
     rm -rf $(WORKLIB_NAME)/$(DUT)
     vlib $(WORKLIB_NAME)/$(DUT)

run_code:
     vlog $(VLOG_OPTS) $(CODE_VLOG_OPTS)
     vsim $(WORKLIB_NAME).$(FILENAME) $(SET_SEED) $(VSIM_OPTS)
$(CONSOLE_OPTS) -do "run -all"

run_task:
     vlog $(VLOG_OPTS) $(CODE_VLOG_OPTS)
     vsim $(WORKLIB_NAME).$(FILENAME)_task $(SET_SEED)
$(VSIM_OPTS) $(CONSOLE_OPTS) -do "run -all"

create_snapshot: worklib_dut
     vlog $(VLOG_OPTS) $(TEST_VLOG_OPTS) $(RTL_OPTS) $(TB_OPTS)

base_test: create_snapshot
     vsim $(WORKLIB_NAME)/$(DUT).$(TB_NAME) $(BASE_TEST_OPTS)
$(USER_DEF_OPTS) $(SET_SEED) $(VSIM_OPTS) \
     ||:
     $(run_log_parser);

base_test_c: create_snapshot
```

```
     vsim $(WORKLIB_NAME)/$(DUT).$(TB_NAME) $(BASE_TEST_OPTS)
$(USER_DEF_OPTS) $(SET_SEED) $(VSIM_OPTS) $(CONSOLE_OPTS) \
     -do script/run_all.tcl \
     ||:
     $(run_log_parser);

test: create_snapshot
     vsim $(WORKLIB_NAME)/$(DUT).$(TB_NAME) $(USER_DEF_OPTS)
$(SET_SEED) $(VSIM_OPTS) \
     -do script/add_signals.tcl
     ||:
     $(run_log_parser);

test_c: create_snapshot
     vsim $(WORKLIB_NAME)/$(DUT).$(TB_NAME) $(USER_DEF_OPTS)
$(SET_SEED) $(VSIM_OPTS) $(CONSOLE_OPTS) \
     -do "run -all" \
     ||:
     $(run_log_parser);



run_iteration__base:
     vsim $(WORKLIB_NAME)/$(DUT).$(TB_NAME) $(USER_DEF_OPTS)
$(SET_SEED) $(VSIM_OPTS) $(CONSOLE_OPTS) \
     +test_base -do script/run_all.tcl ||:
     $(run_log_parser_regression);
     $(merge_cov)

# Add your iterations below
run_iteration__walk:
     vsim $(WORKLIB_NAME)/$(DUT).$(TB_NAME) $(USER_DEF_OPTS)
$(SET_SEED) $(VSIM_OPTS) $(CONSOLE_OPTS) \
     +walk_test -do script/run_all.tcl ||:
     $(run_log_parser_regression);
     $(merge_cov)

run_iteration__ignore:
     vsim $(WORKLIB_NAME)/$(DUT).$(TB_NAME) $(USER_DEF_OPTS)
$(SET_SEED) $(VSIM_OPTS) $(CONSOLE_OPTS) \
     +ignore_test -do script/run_all.tcl ||:
     $(run_log_parser_regression);
     $(merge_cov)

# Add your iteration launches here
run_all_iterations:
     make -f $(MAKEFILE) run_iteration__base
     make -f $(MAKEFILE) run_iteration__walk
     make -f $(MAKEFILE) run_iteration__ignore

stat_closure:
     $(run_stat_parser)
```

```
	cat $(WORKLIB_NAME)/$(DUT)/stat.log

cov_closure:
	$(run_cov_report)

regression: create_snapshot run_all_iterations stat_closure

cov_report:
	echo $(DUT_PATH)

clean:
	rm -rf $(WORKLIB_NAME)/$(DUT)
	rm transcript
	rm *.wlf
	rm *.vstf

veryclean:
	rm -rf $(WORKLIB_NAME)
	rm transcript
	rm *.wlf
	rm *.vstf
```

**Вывод**

В ходе лабораторной работы №3 были освоены практические приёмы построения тестового окружения на SystemVerilog: реализованы рандомизируемые транзакции и ограничения на генерацию данных с помощью констрейнов, а также отработаны механизмы параллельного выполнения и синхронизации процессов с использованием fork...join, mailbox и semaphore. Для тестируемого устройства (приоритетного шифратора) были доработаны транзакции входного и выходного агентов, разработана последовательность генерации входных воздействий и сформирован набор тестовых сценариев, отражающий проверяемые функции устройства. Итоговые тесты были подключены на верхнем уровне тестбенча и добавлены в регрессию через Makefile, а корректность формирования воздействий подтверждена по временным диаграммам Modelsim.