AEDVICES Consulting
166B Rue du Rocher de Lorzier
38430 Moirans– France
trainings@aedvices.com
http://www.aedvices.com

Training on
Functional Verification Methodology
Using UVM

LAB
Random Variable
Constraint Programming

Objectives

This lab goes through the concept of random variables and constraints. It shows how random variables are created and how they interact together using constraints. The lab will show how random variables can be used to generate input stimulus for test purposes.

Introduction

In constraint programming, the relations between variables are stated in the form of constraints. Constraints do not specify a step or a sequence of steps to execute, but they rather define the properties of the solution which needs to be found.
Constraints are used in verification to declare valid states of test inputs and the randomization is used to generate tests amongst the valid states.

SystemVerilog provides the variable modifier “rand” to declare a random variable and the block construct “constraint *name* { *boolean_expressions;* } ” to declare constraints such as in the following example:

```
class myclass;
      integer a;    // this is a classical non random variable
      rand integer b;   // this is a random variable
      rand integer c;   // this is a random variable
      constraint b_lt_c { b < c; };
endclass
```

Global Explanation

The lab contains the following files

lab.sv	Includes all others.
tb.sv	Simple testbench driving clocks and reset
lab_verif_pkg.sv	Defines the main transactions to be used. You will have to make the variables random.
lab_prog.sv	The main program. You will have to use transactions created in lab_verif_pkg

The lab_verif_pkg defines two transaction classes:

```
//Transaction Class
class transactionBase ;
  rand data_t      addr;
       data_t      data;
       direction_t dir;
       action_t    action;
endclass
```

```
class adderTransaction extends transactionBase;
  // Address is in the range from 0 to 5
  constraint addr_range {
    addr inside {['h0:'h5]};
  }
endclass
```

These two classes are used in the program in lab_prog.

- Transactions are generated using randomize()
- Driven using the drive function

As follow

```
  // A random base transaction
  ok = trans_base.randomize();
  if ( ! ok )
    $display("Error randomizing trans_base");
  drive(trans_base);
```

The drive task actually drive signals of the testbench and also prints the generated transaction.

For the purpose of this lab, there is further designs.

Instructions

Follow instructions given in “aedv_training_labs_intructions_for_questa.pdf”.
Open the file

lab.sv
Select

Step 1 – Understand existing code

- Open the file: <SANDBOX>/labs-Xdays/labNN- random_generation/lab_verif_pkg.sv

Search for LAB-TODO-STEP-1

- Question:
  - What is the difference between *addr* and *data* generation?

This transaction is used in lab_prog.
Run the simulation:
  - Why *data* and *dir* are always X?

Step 2 – Create random transactions

Search for “LAB-TODO-STEP-2-a”

a) Identify the declaration of the variable: *data*.

- Add “rand” keyword in front of the variables “*data*” , “dir” and “action”
  Recompile the test and reload the simulation.

  Question:
  - Are values properly generated ?

- Takes notes of the first few generated values
- Rerun the simulation and look at the random values that are generated.

- Question:
  - Are the value random again ?

b) Change the seed value & Reload the simulation again.
   Check the values that are generated.

Search for “LAB-TODO-STEP-2-b”

c) Add a constraint to specify that addresses 4 and 5 are read only area.

```
constraint read_only_area {
      addr inside {'h4, 'h5} -> dir == READ;
}
```

Step 3 – Create sequences of random transactions

- Open the file: lab_prog.sv

Search for “LAB-TODO-STEP-3-a”
- The first generated transaction is “trans_base”
- The second and successive generated transaction is “trans_adder”

Question:
Explain the value see in the log on the first two transactions. Compare with constraints.

Search for “LAB-TODO-STEP-3-b”
- Add a inline constraint to the randomize() so that:
  - Transaction is a READ
  - Address is 4

Rerun & Check

Search for “LAB-TODO-STEP-3-c”
- Constrain the address to perform a write to address 4

Rerun & Check

Question:
What happened ? Why ?

Step 4 - Create more complex sequences

Search for “LAB-TODO-STEP-4-a”

- Replace
  `ok = trans_adder.randomize()`
  with
  `ok = my_seq.randomize() ;`

Rerun & Check

- Question: Can you explain what happen ?

Search for “LAB-TODO-STEP-4-b”

- Add the “rand” keyword before “adderTransaction”

```
    rand adderTransaction trans[10];
```

Rerun & Check

- Does it fix your problem ?

Search for “LAB-TODO-STEP-4-c”

- Add constraints so that
  - the first transaction is a WRITE to address 1
  - All transactions are action == ACC
  - The second transaction data == the first transaction data.