ASIC Verification

Randomization

Fall 2011
Meeta Yadav
Topics

- Controlling Constrains
- Controlling Random Variables
- Array Constraints
- Scenario Generation
- Random Device Configuration
Controlling Constraints

• In-line Constraints
  - SV allows you to add an extra constraint to any existing ones in effect using `randomize() with`

```vhdl
class Transaction;
    rand bit [31:0] addr, data;
    constraint c1{addr inside {[0:100],[1000:2000]};}
endclass

Transaction t;

initial begin
    t=new();
    assert(t.randomize() with {addr>=50; addr<=1500; data<10;});
    driveBus(t);
    assert(t.randomize() with {addr==2000; data>10;});
    driveBus(t);
end
```

- `randomize() with statement`
Controlling Constraints

- Constraint with variables

```plaintext
class Bounds;
    rand int size;
    int max_size=100;
    constraint c_size{size inside {[1:max_size]};};
endclass
```

By changing the variable max_size you can vary the upper limit

Creates random sizes between 1 and 100

Using random variables
Controlling Constraints

- Controlling constraints with `constraint_mode()`
  - Can be used to control whether a constraint is active or inactive
    - When a constraint is inactive it is not considered by `randomize()`
    - All constraints are initially active
    - `constraint_mode` method is built in and cannot be overridden

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OFF</td>
<td>Sets the specified constraint block to inactive so that it is not enforced by subsequent calls to <code>randomize()</code> method</td>
</tr>
<tr>
<td>1</td>
<td>ON</td>
<td>Sets the specified constraint block to active so that it is considered by subsequent calls to <code>randomize()</code> method</td>
</tr>
</tbody>
</table>
Controlling Constraints

- Controlling multiple constraint blocks
  - A class may contain multiple constraints
  - At run time you can use `constraint_mode()` routine to turn constrains on and off
  - To control all constraints in an object use

  ```c
  handle.constraint_mode()
  ```

- A single constraint can be controlled with

  ```c
  handle.constraint.constraint_mode()
  ```
Controlling Constraints

- Controlling multiple constraint blocks

```verilog
class Packet;
    rand int length;
    constraint c_short{length inside {[1:32]};}  // Create short packets
    constraint c_long{length inside {[1000:1023]};}  // Create long packets
endclass

Packet p;

initial begin
    p=new();
    p.c_short.constraint_mode(0);  // Create long packets by disabling short constraint
    assert(p.randomize());
    transmit(p);
    p.constraint_mode(0);
    p.c_short.constraint_mode(1);  // Create short packets by disabling all constraints and enabling only the short constraint
    assert (p.randomize());
    transmit (p);
end
```

**Using constraint_mode**

Create long packets by disabling short constraint

Create short packets by disabling all constraints and enabling only the short constraint
Controlling Constraints

• Specifying a constraint in a test with external constraints
  ◆ Body of constraint can be defined outside the class
  ◆ This allows data class to be defined in one file with an empty constraint
  ◆ This enables each test to define its own version of this constraint to generate its own flavors of stimulus
  ◆ Can only add constraints and not alter existing ones

```
class Packet;
  rand bit [7:0] length;
  rand bit [7:0] payload[];
  constraint c_valid {length>0; payload.size()==length;}
  constraint c_external;
endclass

program test
  constraint Packet::c_external {length==1;}
  ...
end
```

*Class and Program with an external constraint*
Disabling Random Variables

- Disabling random variables with `rand_mode()`
  - Can be used to control whether a random variable is active or inactive
  - When a random variable is inactive it implies that the variable was never declared `rand` or `randc`
  - `rand_mode()` method is inbuilt and cannot be overridden

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OFF</td>
<td>Sets the specified variable to inactive so that they are not randomized on subsequent calls to <code>randomize()</code> method</td>
</tr>
<tr>
<td>1</td>
<td>ON</td>
<td>Sets the specified variable to active so that they are randomized on subsequent calls to <code>randomize()</code> method</td>
</tr>
</tbody>
</table>

```plaintext
class Packet;
    rand integer src, dst;
endclass

int r;
Packet packet_a=new();
packet_a.rand_mode(0);
packet_a.src.rand_mode(1);
r=packet_a.dst.rand_mode();
```
Controlling Random Variables

- Using non random values: `rand_mode()`

```verilog
class Packet;
    rand bit[7:0] length, payload[];
    constraint c_valid{length > 0 ; payload.size() == length;}
endclass

initial begin
    Packet p;
    p.length.rand_mode(0); Make length non random
    p.length=42; Set length to a constant value
    assert(p.randomize()); Randomize the payload
end
```

Using `rand_mode()` to disable variables
Controlling Random Variables

• Randomizing individual variables
  ◆ To randomize a few variables inside a class, `randomize()` can be called with the subset of variables
  ◆ Only the variables passed in the argument are randomized
  ◆ The rest of the variables are not randomized
  ◆ The constraints remain in effect

```plaintext
class Rising;
    byte low;
    rand byte med, hi;
    constraint up {low<med; med<hi}
endclass

initial begin
    Rising r;
    r=new();
    r.randomize();
    r.randomize(med);
    r.randomize(low);
end
```

*Randomization of a subset of variables in a class*
Iterative and Array Constraints

- Array Size

```vhdl
class dyn_size;
    rand logic [31:0] d[];
    constraint d_size {d.size() inside {[1:10]};}
endclass
```

Constraining dynamic size array
Iterative and Array Constraints

• Example
  ◆ You have an interface that has to transfer 4 data words
  ◆ The words can be transferred consecutively or over 10 cycles
  ◆ A strobe signal will tell you when data is ready to be transferred

Random Strobe Pattern
Iterative and Array Constraints

• Example

```verbatim
parameter MAX_TRANSFER_LEN=10;
class StrobePat;
    rand bit strobe[MAX_TRANSFER_LEN];
    constraint c_set_four {strobe.sum()==4'h4;}
endclass
initial begin
    StrobePat sp;
    int count=0;
    sp=new();
    assert(sp.randomize());
    foreach (sp.strobe[i]) begin
        @bus.cb
        bus.cb.strobe<=sp.strobe[i];
        if(sp.strobe[i])
            bus.cb.data<=data[count++];
    end
end
```

Random Strobe Pattern
Iterative and Array Constraints

• Constraining Individual Array and Queue Elements
  ◆ An array of unsigned integers
  ◆ Each entry should be between 1 and 255
  ◆ The array should have between 1 to 8 elements
  ◆ All those elements to add up to less than 1024

```plaintext
class good_sum;
    rand unint len[];
    constraint c_len {for each (len[i])
        len[i] inside {[1:255]};
        len.sum<1024;
        len.size() inside {[1:8]};
    }
endclass
```

*Simple foreach constraint*
Scenario Generation

- Scenario Generation using randsequence
  - So far we have seen atomic stimulus generation
    - Can make a single bus transaction, a single network packet or a single processor instruction
  - Verifying a design in the real-world stimuli needs long sequences of transactions
    - DMA transfers
    - Cache fills
    - Network traffic due to browsing internet, reading e-mails etc.
    - Deep pipelines for processors
  - Use randsequence SystemVerilog constructs to generate a sequence of transactions
Scenario Generation

• Scenario Generation using randsequence

Example of randsequence

```
randsequence( main )
    main: first second done ;
    first: add | dec ;
    second: pop | push ;
    done: { $display("done"); };
    add: { $display("add"); };
    dec: { $display("dec"); };
    pop: { $display("pop"); };
    push: { $display("push"); };
endsequence
```

Possible Outcomes

- add pop done
- add push done
- dec pop done
- dec push done

When main is chosen it generates the sequence first, second and done

First chooses between add or dec

Second chooses between pop or push
Scenario Generation

- Scenario Generation using randsequence

Stream can be either cfg_read, io_read or mem_read with the respective weights

Either a single call to cfg_read task or multiple calls

```
initial begin
  for(int i=0; i<15; i++) begin
    randsequence (stream)
      stream : cfg_read:=1 |
                io_read:=2 |
                mem_read:=5;
      cfg_read: {cfg_read_task;} |
                {cfg_read_task;} cfg_read;
      mem_read: {mem_read_task;} |
                {mem_read_task;} mem_read;
      io_read:  {io_read_task;} |
                {io_read_task;} io_read;
  endsequence
end
```

Command generator using randsequence
Random Device Configuration

• Example of Creating Random Testbench Configuration

```plaintext
class eth_cfg;
    rand bit [ 3:0] in_use; // Ports used in test
    rand bit [47:0] mac_addr[4]; // MAC addresses
    rand bit [ 3:0] is_100; // 100mb mode
    rand int run_for_n_frames; // # frames in test

    // Force some addr bits when running in unicast mode
    constraint local_unicast {
        foreach (mac_addr[i])
            mac_addr[i][41:40] == 2'b00;
    }

    constraint reasonable {// Limit test length
        run_for_n_frames inside {[1:100]};
    }
endclass : eth_cfg
```

*Ethernet Switch Configuration Class*
Random Device Configuration

- Building Environment with Random Configuration

```java
Class Environment
    eth_cfg cfg;
    eth_src gen[4];
    eth_mii drv[4];
    function new();
        cfg=new();
    endfunction
    function void gen_cfg;
        assert(cfg.randomize());
    endfunction
    function void build();
        foreach (gen[i])
            if(cfg.in_use[i]) begin
                gen[i]=new();
                drv[i]=new();
                if (cfg.is_100[i])
                    drv[i].set_speed(100);
            end
        end
    endfunction

task run();
    foreach (gen[i])
        if(cfg.in_use[i]) begin
            gen[i].run();
            ...
        end
endtask

task wrap_up();
    ...
endtask
endclass
```

Building environment with random configurations
Random Device Configuration

• Building Environment with Random Configuration

```verbatim
program test;
    Environment env;
    initial begin
        Construct environment
        env=new();
        Create random configurations
        env.gen_cfg;
        Build the testbench environment
        env.build();
        Run the test
        env.run();
        Clean up after test and report
        env.wrap_up();
    end
endprogram
```

Simple test using random configurations
Thank You