class seq;

rand logic [7:0] a;

rand logic [7:0] b;

logic [7:0] out;

endclass

class generator;

seq s = new();

mailbox mbx;

function new(mailbox mbx);

this.mbx = mbx;

endfunction

task run();

for (int i = 0; i < 10; i++) begin

s.randomize();

mbx.put(s);

#10;

end

endtask

endclass

interface sum\_intf;

logic [7:0] a;

logic [7:0] b;

logic [8:0] out;

logic clk;

logic valid;

endinterface

class driver;

seq s;

mailbox mbx;

virtual sum\_intf intf;

function new(mailbox mbx);

this.mbx = mbx;

endfunction

task run();

for (int i = 0; i < 10; i++) begin

mbx.get(s);

@(posedge intf.clk);

if (intf.valid) begin

intf.a <= s.a;

intf.b <= s.b;

intf.out <= s.out;

$display("[DRV] The val in s is %0d, %0d", s.a, s.b);

end

else begin

intf.a <= intf.a;

intf.b <= intf.b;

intf.out <= intf.out;

end

#10;

end

endtask

endclass

module tb;

mailbox mbx = new();

sum\_intf intf();

driver drv = new(mbx);

generator gen = new(mbx);

add dut(.a(intf.a), .b(intf.b), .out(intf.out));

initial begin

drv.intf = intf;

fork

drv.run();

gen.run();

for (int i = 0; i < 10; i++) begin

#10;

$display("The addition value is: %0d", intf.out);

end

join

end

initial begin

intf.clk = 0;

forever #5 intf.clk = ~intf.clk;

end

initial begin

intf.valid = 0;

forever begin

#20 intf.valid = ~intf.valid;

end

end

initial begin

$dumpvars;

$dumpfile("dump.vcd");

end

endmodule

module add(input [7:0] a, b, output [8:0] out);

assign out = a + b;

endmodule

class seq;

rand logic [7:0] a;

rand logic [7:0] b;

logic [7:0] out;

endclass

class generator;

seq s=new();

mailbox mbx;

function new(mailbox mbx);

this.mbx=mbx;

endfunction

task run();

for (int i=0; i<10;i++) begin

s.randomize();

mbx.put(s);

// $display("[Gen] The val in s is %0d ",s.a);

#10;

end

endtask

endclass

interface sum\_intf;

logic [7:0] a;

logic [7:0] b;

logic [8:0] out;

endinterface

class driver;

seq s;

mailbox mbx;

virtual sum\_intf intf;

function new(mailbox mbx);

this.mbx=mbx;

endfunction

task run();

for (int i=0; i<10;i++) begin

mbx.get(s);

intf.a = s.a;

intf.b = s.b;

intf.out = s.out;

$display("[DRV] The val in s is %0d ",s.a,s.b);

#10;

end

endtask

endclass

module tb;

mailbox mbx=new();

sum\_intf intf();

driver drv=new(mbx);

generator gen=new(mbx);

add dut(.a(intf.a),.b(intf.b),.out(intf.out));

initial begin

drv.intf = intf;

fork

drv.run();

gen.run();

for(int i=0; i<10;i++) begin

#10;

$display("the addition value is: %0d", intf.out);

end

join

end

initial begin

$dumpvars;

$dumpfile("dump.vcd");

end

endmodule

// Code your design here

module add(input [7:0]a,b, output [8:0] out);

assign out = a+b;

endmodule

Generate Eth packets and drive to DUT according to interface protocol. Packet Generator: Generate Ethernet packets with following layers: 1. L2: Eth header - DA: 48b, SA: 48b VLAN may or may not be present: 32b Eth type: 16b 2. L3: IP header should be one of the following types IPv4: 20B IPv6: 40B 3. L4: Should be one of the following types TCP: 20B UDP: 8B Driver: Driver gets the packet from generator and drives to DUT according to interface protocol: Interface Protocol: The protocol is a simple data/valid type interface where valid qualifies data and other control signals. Data transfer stops if valid is low. Valid can be low for any number of times during packet transfer and may remain low for any number of cycles Packet is bounded by SOP and EOP. Interface signals: 1. Signal: sop Width (bit): 1 Description: If set, indicates start of the packet; 2. Signal: valid Width (bit): 1 Description: If high, indicates all other signals in the interface is valid. All signals should hold their previous value, each cycle valid is low; 3. Signal: data Width (bit): 64 Description: Packet data; 4. Signal: eop Width (bit): 1 Description: If set, indicates end of the packet. Implementation note: Use SV constraint block to generate different packet types Implement packet functions to assemble header and payload bytes into an array Packets represented as byte arrays is useful for the driver Use SV interface to dive the packets

**Step 1: Define the Interface**

First, we define the interface with the required signals.

interface eth\_if(input logic clk);

logic sop;

logic valid;

logic [63:0] data;

logic eop;

modport master (output sop, valid, data, eop);

modport slave (input sop, valid, data, eop);

endinterface

**Step 2: Create the Packet Generator**

The packet generator will create Ethernet packets with the specified layers using constraint blocks to generate different packet types.

class eth\_packet;

rand bit [47:0] da;

rand bit [47:0] sa;

rand bit [31:0] vlan;

rand bit [15:0] eth\_type;

rand bit [159:0] ip\_header; // Max size for IPv6

rand bit [159:0] l4\_header; // Max size for TCP

rand bit [7:0] payload[];

constraint c\_vlan { vlan == 32'h0 || vlan inside {32'h81000000, 32'h91000000}; }

constraint c\_eth\_type { eth\_type inside {16'h0800, 16'h86DD}; } // IPv4 or IPv6

constraint c\_ip\_header { ip\_header.size() == (eth\_type == 16'h0800) ? 160 : 320; } // IPv4 or IPv6

constraint c\_l4\_header { l4\_header.size() == (eth\_type == 16'h0800) ? 160 : 64; } // TCP or UDP

function new();

this.payload = new; // Default payload size

endfunction

function void assemble\_packet();

// Assemble the packet into a byte array

// This function can be expanded to include more detailed assembly logic

endfunction

endclass

class packet\_generator;

eth\_if intf;

function new(eth\_if intf);

this.intf = intf;

endfunction

task generate\_packet();

eth\_packet pkt = new();

pkt.randomize();

pkt.assemble\_packet();

// Drive the packet to the interface

intf.sop <= 1;

intf.valid <= 1;

intf.data <= {pkt.da, pkt.sa, pkt.vlan, pkt.eth\_type};

#1 intf.sop <= 0;

// Drive IP header

intf.data <= pkt.ip\_header;

#1;

// Drive L4 header

intf.data <= pkt.l4\_header;

#1;

// Drive payload

foreach (pkt.payload[i]) begin

intf.data <= pkt.payload[i];

#1;

end

// End of packet

intf.eop <= 1;

#1 intf.eop <= 0;

intf.valid <= 0;

endtask

endclass

**Step 3: Create the Driver**

The driver will take the generated packets and drive them to the DUT according to the interface protocol.

class driver;

eth\_if intf;

function new(eth\_if intf);

this.intf = intf;

endfunction

task drive\_packet();

packet\_generator pkt\_gen = new(intf);

pkt\_gen.generate\_packet();

endtask

endclass

**Step 4: Create the Testbench**

Finally, we create the testbench to instantiate the interface, driver, and packet generator, and run the simulation.

module tb;

logic clk;

eth\_if intf (clk);

// Clock generation

initial clk = 0;

always #5 clk = ~clk;

// Instantiate driver

driver drv = new(intf);

initial begin

// Run the driver to generate and drive packets

drv.drive\_packet();

#100 $finish;

end

endmodule

**Summary**

1. **Interface**: Defines the signals and their directions.
2. **Packet Generator**: Creates Ethernet packets with the specified layers using constraints.
3. **Driver**: Drives the generated packets to the DUT according to the interface protocol.
4. **Testbench**: Instantiates and connects everything, and runs the simulation.

## Intel Verification Engineer Lead

#### Responsibilities:

1. **Develop and execute verification strategies for complex ASICs and SoCs**
2. **Architect test bench infrastructure and verification methodologies**
3. **Collaborate with cross-functional teams to understand chip architecture and plan verification tasks**
4. **Work with design and architecture teams to ensure alignment on feature scoping, staging, and verification coverage**
5. **Develop and integrate verification environment components using UVM/System Verilog**
6. **Create verification/test plans from functional specifications**
7. **Develop, simulate, and debug UVM and C-based test cases**
8. **Drive initiatives for continuous improvement of IP/SOC quality**
9. **Mentor engineers and provide technical guidance to the team**

#### **Qualifications:**

1. **Bachelor's degree in electrical engineering, computer engineering, computer science, or related field**
2. **10+ years of total experience, with experience in verifying ASICs in full chip or SoC area, microcontroller-based IP verification, and Ethernet protocol experience**
3. **Strong expertise in System Verilog, OVM/UVM, and scripting languages**
4. **Experience with debugging, verification of microarchitecture using industry-standard formal verification tools, and verification of FPGA-based architecture designs**

## Develop UVM sequences, tests, scoreboards, monitors and checkers.

## argument pass by name

In argument pass by name, arguments can be passed in any order by specifying the name of the subroutine argument.

module argument\_passing;

  int x,y,z;

  function void display(int x,string y);

    $display("\tValue of x = %0d, y = %0s",x,y);

  endfunction

  initial begin

    display(.y("Hello World"),.x(2016));

  end

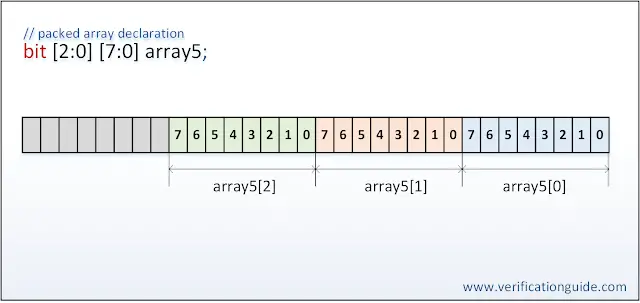
endmodule

The expression da[47 - 8\*i -: 8] is used to extract 8-bit chunks from the 48-bit

**Packed array example**

bit [2:0] [7:0] array5;

The below diagram shows storing packed array as a **contiguous**set of bits.



## Function to Modify bashrc and source ./bashrc to run the shell script so that always full path can be seen from the prompt.

**function cpwd() {  
  DIR="$\*";  
  # if no DIR given, go home  
  if [ $# -lt 1 ]; then  
    DIR=$HOME;  
  fi;  
  builtin cd "${DIR}" && PS1="`pwd`> "  
}**

**alias cd='cpwd'**

**How to use constraint?**

**typedef enum** {low, mid, high} AddrType;

**class** MyBus **extends** Bus;

**rand** AddrType atype;

**constraint** addr\_range

{

(atype == low ) -> addr **inside** { [0 : 15] };

(atype == mid ) -> addr **inside** { [16 : 127]};

(atype == high) -> addr **inside** {[128 : 255]};

}

**Endclass**

1. **Necessary git commands**
2. Create a directory pkt\_driver (something different from last time)
3. copy all your code into pkt\_driver/src directory
4. Copy all run scripts into pkt\_driver/run directory
5. In verif 'git add pkt\_driver'
6. git status
7. git commit
8. git push
9. **Add the remote repository URL**:
10. git remote add origin http://gitlab.dbs.corp/dream-big-semi/mars/deimos-devs/deimos-eth
11. **Push your changes to the remote repository**:
    1. git push -u origin other/shovan/no-jira/cpb\_dev

**git config --global user.name shovan**

**git config --global user.email** [**shovan@dreambigsemi.com**](mailto:shovan@dreambigsemi.com)

**cd cpb\_dev/**

**git init // Initialized empty Git repository in /Users/user/cpb\_dev/.git/**

**git add –all**

**git commit -m "First release of Hello World!"**

**98 git push --set-upstream origin other/shovan/no-jira/cpb\_dev**

**99 git branch**

**105 git clone http://gitlab.dbs.corp/dream-big-semi/mars/deimos-devs/deimos-eth/ cpb\_dev**

**106 cd cpb\_dev/**

**107 ls**

**108 git branch**

**109 git checkout**

**110 git checkout other/shovan/no-jira/cpb\_dev**

**112 git status**

**113 ls**

**114 cd verif/**

**115 ls**

**116 mkdir -p cpb/src**

**117 cd cpb/src/**

**118 touch test.sv**

**119 git status**

**120 cd ..**

**git status**

**git add cpb/**

**git status**

**git commit**

**git push**

**126 cd ..**

**127 ls -lh**

**128 cd verif/**

**129 ls**

**130 cd cpb/**

**131 ls**

**132 mkdir run**

**133 ls**

**134 cd src/**

**135 cp /home/shovan/deimos-eth/verif/cpb/src/\* .**

**136 ls**

**137 cd ..**

**138 cd run/**

**139 cp -rf /home/shovan/deimos-eth/verif/cpb/run/\* .**

**class** Packet ;

**static integer** fileID = $fopen( "data", "r" );

Packet p;

c = $fgetc( p.fileID );

**typedef** C T; // T is a default specialization, not an alias to

// the name "C"

*Example 1:* Unsized literal constant numbers

659 // is a decimal number

'h 837FF // is a hexadecimal number

'o7460 // is an octal number

4af // is illegal (hexadecimal format requires 'h)

*Example 2:* Sized literal constant numbers

4'b1001 // is a 4-bit binary number

5 'D 3 // is a 5-bit decimal number

3'b01x // is a 3-bit number with the least

// significant bit unknown

12'hx // is a 12-bit unknown number

16'hz // is a 16-bit high-impedance number

*Example 3:* Using sign with literal constant numbers

8 'd -6 // this is illegal syntax

-8 'd 6 // this defines the two's-complement of 6,

// held in 8 bits—equivalent to -(8'd 6)

Example 4: Automatic left padding of literal constant numbers

logic [11:0] a, b, c, d;

logic [84:0] e, f, g;

initial begin

a = 'h x; // yields xxx

e = 'h5; // yields {82{1'b0},3'b101}

f = 'hx; // yields {85{1'hx}}

g = 'hz; // yields {85{1'hz}}

**end**

*Example 5:* Automatic left padding of constant literal numbers using a single-bit value

**logic** [15:0] a, b, c, d;

a = '0; // sets all 16 bits to 0

b = '1; // sets all 16 bits to 1

c = 'x; // sets all 16 bits to x

d = 'z; // sets all 16 bits to z

*Example 6:* Underscores in literal constant numbers

27\_195\_000 // unsized decimal 27195000

16'b0011\_0101\_0001\_1111 // 16-bit binary number

32 'h 12ab\_f001 // 32-bit hexadecimal number

236.123\_763\_e-12 (underscores are ignored)

Time is written in integer or fixed-point format, followed without a space by a time unit (**fs ps ns us ms s**).

For example:

2.1ns

40ps

For example, to store the 12-character string "Hello world\n" requires a variable 8  12, or 96 bits wide.

**bit** [8\*12:1] stringvar = "Hello world\n";

Alternatively, a multidimensional packed array can be used, with 8-bit subfields, as in:

**bit** [0:11] [7:0] stringvar = "Hello world\n" ;

A string literal can be assigned to an unpacked array of bytes. If the size differs, it is left justified.

**byte** c3 [0:12] = "hello world\n" ;

Struct

**typedef struct** {**int** a; **shortreal** b;} ab;

ab c;

c = '{0, 0.0}; // structure literal type determined from

// the left-hand context (c)

Nested braces shall reflect the structure. For example:

ab abarr[1:0] = '{'{1, 1.0}, '{2, 2.0}};

Replication Operator:

The syntax for the replication operator is {N{value}}, where N is the number of times the value is replicated.

For example:

* {4{1'b0}} creates a 4-bit vector with all bits set to 0.
* {3{2'b01}} creates a 6-bit vector with the pattern 010101.

Constraint & Randomization:

**rand bit** [7:0] len;

**rand integer** data[];

**constraint** db { data.size == len; }

**class** C;

**rand int** x;

**constraint** proto1; // implicit form

**extern constraint** proto2; // explicit form

**endclass**

For both forms the constraint can be completed by providing an *external constraint block* using the class

scope resolution operator, as in the following example:

**constraint** C::proto1 { x **inside** {-4, 5, 7}; }

**constraint** C::proto2 { x >= 0; }

**rand integer** x, y, z;

**constraint** c1 {x **inside** {3, 5, [9:15], [24:32], [y:2\*y], z};}

**integer** fives[4] = '{ 5, 10, 15, 20 };

**rand integer** v;

**constraint** c3 { v **inside** {fives}; }

class RandC;

int unsigned range;

int unsigned values[\*]; // Associative array to store values

// Function to mimic randc behavior

function int my\_randc();

int index;

// Initialize the range and values array if not already done

if (values.num() == 0) begin

if (!$value$plusargs("RANGE=%d", range)) begin

$display("ERROR: Please provide a range using +RANGE=<value>");

$finish;

end

for (int i = 0; i < range; i++) begin

values[i] = i;

end

end

// Randomly select an index from the values array

index = $urandom\_range(values.num() - 1);

// Get the value at the selected index

int result = values[index];

// Remove the selected value from the values array

values.delete(index);

// If all values have been used, reset the array

if (values.num() == 0) begin

for (int i = 0; i < range; i++) begin

values[i] = i;

end

end

return result;

endfunction

endclass

**Bit Slicing Syntax**

+:**Operator**

* **Syntax**: base\_index +: width
* **Meaning**: Extract width bits starting from base\_index.
* **Example**: DA[8\*i +: 8]
  + 8\*i is the starting bit position.
  + +: 8 means extract 8 bits starting from 8\*i.

-:**Operator**

* **Syntax**: base\_index -: width
* **Meaning**: Extract width bits starting from base\_index and moving backwards.
* **Example**: DA[47 -: 8]
  + 47 is the starting bit position.
  + -: 8 means extract 8 bits starting from 47 and moving backwards.