

Agenda

Introduction:

SystemVerilog Motivation

Vassilios Gerousis, Infineon Technologies
Accellera Technical Committee Chair

Session 1:

SystemVerilog for Design

Language Tutorial

Johny Srouji, Intel

User Experience

Matt Maidment, Intel

Session 2:

SystemVerilog for Verification

Language Tutorial

Tom Fitzpatrick, Synopsys

User Experience

Faisal Haque, Verification Central

Lunch: 12:15 – 1:00pm

Session 3: SystemVerilog Assertions

Language Tutorial

Bassam Tabbara, Novas Software

Technology and User Experience

Alon Flaisher, Intel

Using SystemVerilog Assertions and Testbench Together

Jon Michelson, Verification Central

Session 4: SystemVerilog APIs

Doug Warmke, Model Technology

Session 5: SystemVerilog Momentum

Verilog2001 to SystemVerilog

Stuart Sutherland, Sutherland HDL

SystemVerilog Industry Support

Vassilios Gerousis, Infineon

End: 5:00pm



Goals of this presentation

- Understand the testbench model for SystemVerilog
- Show how SystemVerilog constructs are used to build testbenches
- Walk through the construction of a testbench for an Ethernet MAC



My Background

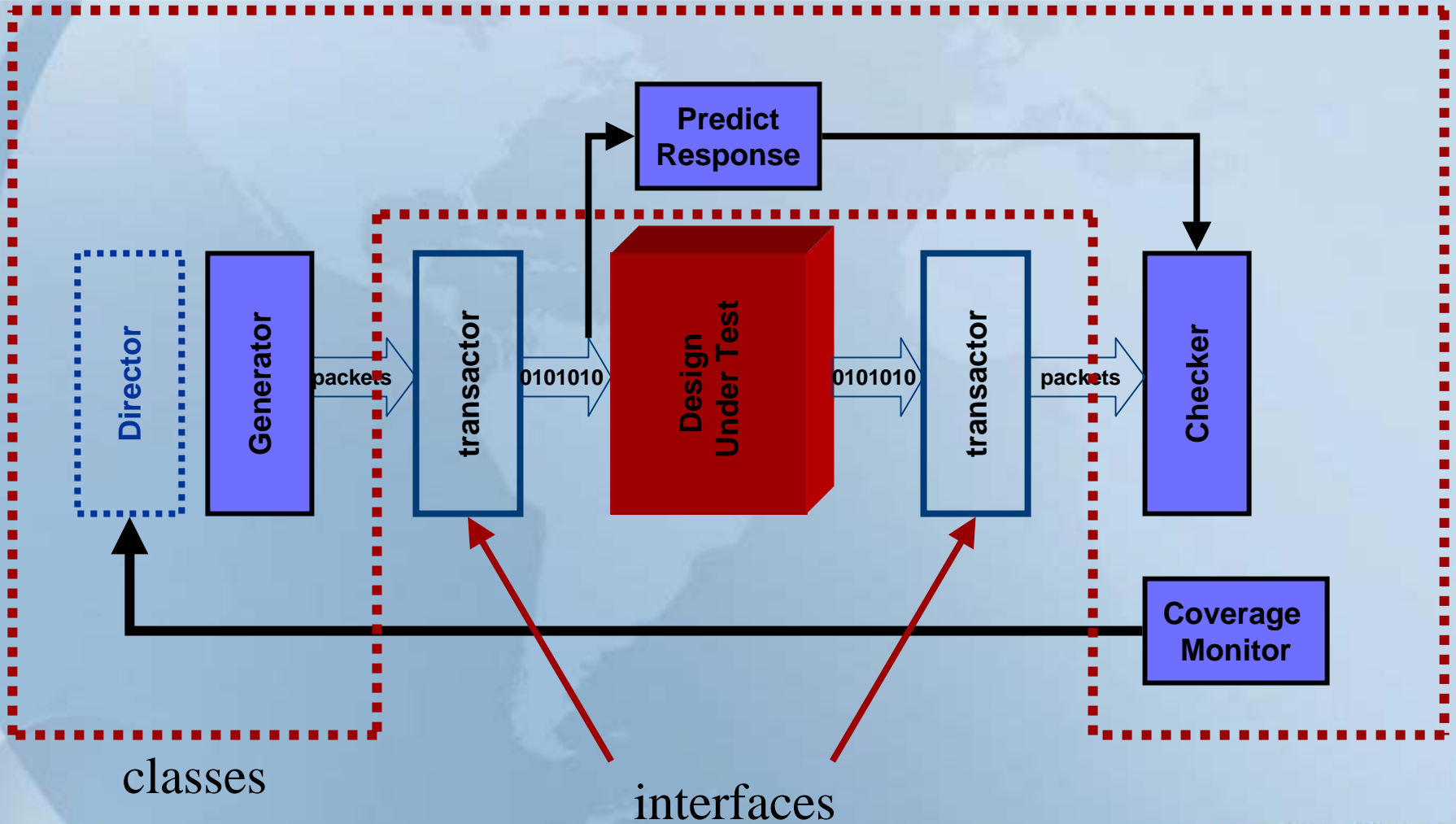
- Chair of SystemVerilog Assertions committee
- Co-author of “The Art of Verification with Vera”
- Co-author of upcoming SystemVerilog book

www.verificationalcentral.com



A Typical Testbench

Adapted from "The Art of Verification with Vera"



Recommended SV constructs for use in testbench

- Program block
- Classes
- Interfaces
- Clocking
- Mailboxes, semaphores
- Constraints and random variables
- Assertions



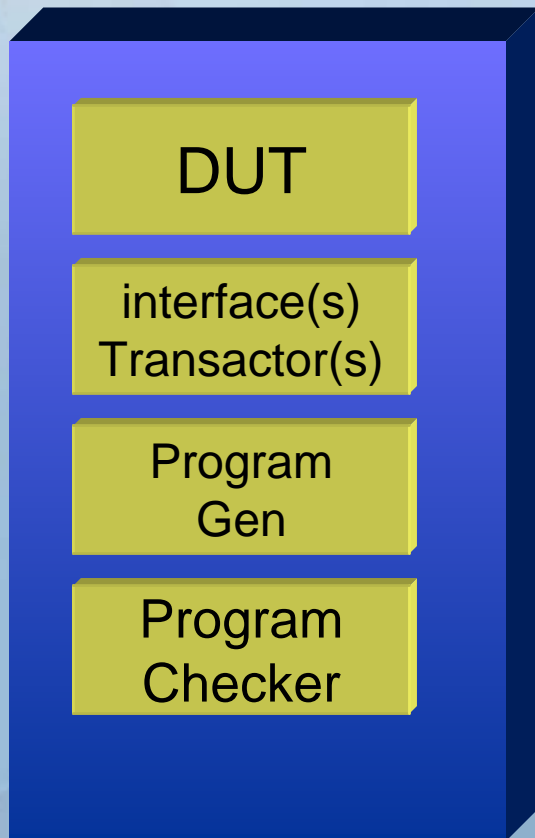
Program Block

- Testbench implemented as multiple **program** blocks
 - For example: generator, checker in separate **program** blocks
- Testbench implemented as a single **program** block
 - Classes instantiated within **program** block
- Program block with classes instantiated within the **interface**

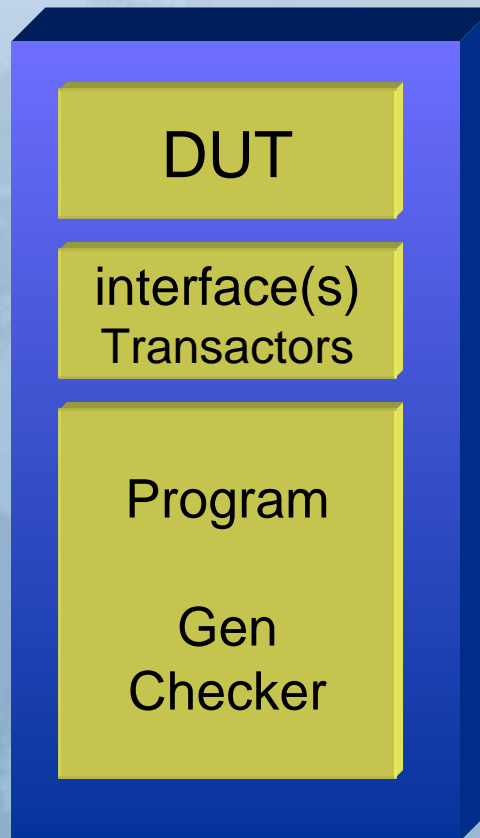


Testbench implementation options

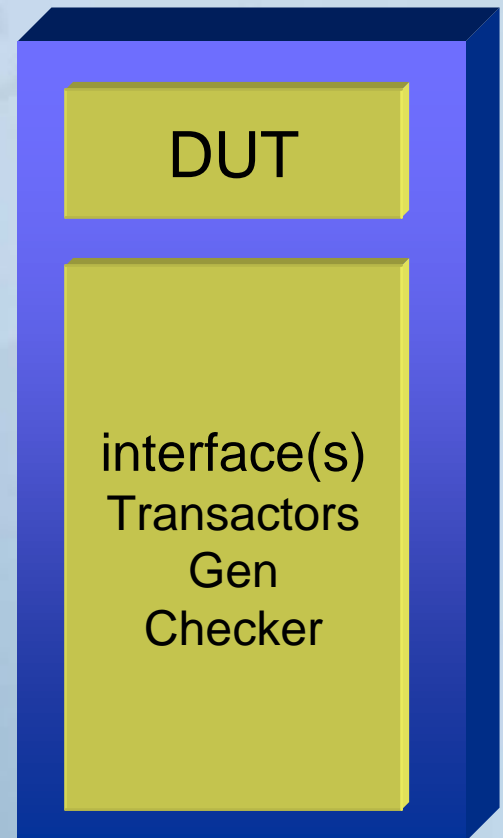
test_top.v



test_top.v

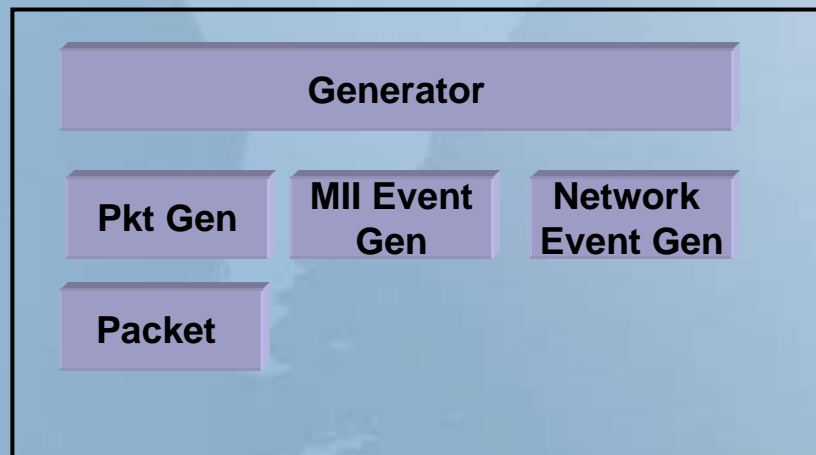


test_top.v



Classes

- Provide a mechanism to encapsulate data
- Recommended for almost all testbench components
- Layer classes to build powerful structures



Interfacing to the Design Under Test (DUT)

- Use **interface** to logically group ports
- **interfaces** can also instantiate tasks, program blocks or classes to act as transactors
 - Classes instantiated inside program blocks will execute in the reactive region of SystemVerilog (Cycle based)
 - Classes instantiated outside program blocks will display event based semantics
- Use **clocking** to define synchronization of the DUT ports with the Testbench

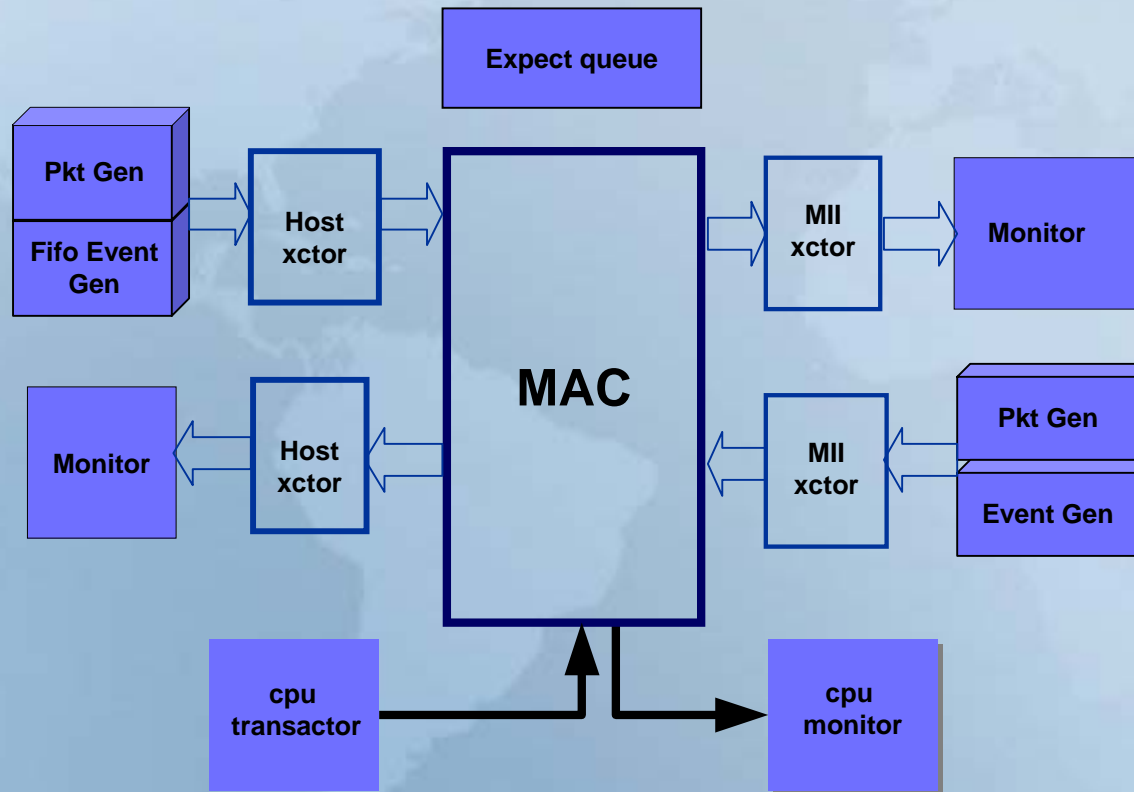


Mailboxes and Semaphores

- Use Mailboxes to pass messages between two threads
 - Can be used to represent FIFO implementations
- Use semaphores to arbitrate between two or more threads



Ethernet MAC Testbench



classes

Interface-tasks



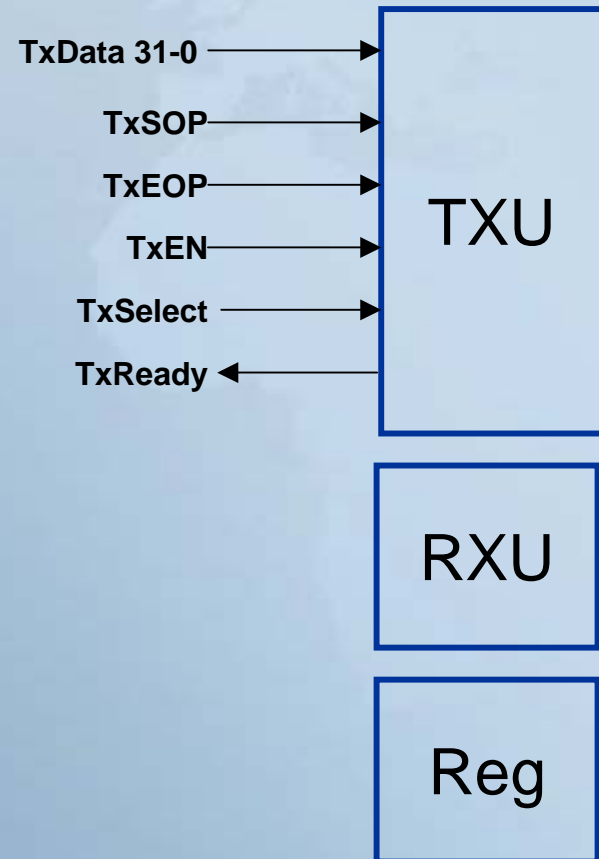
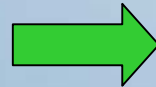
Ethernet MAC

```
interface host_tx;
  input [31:0] txdata;
  input txselect;
  input txsop;
  input txeop;
  input txen;
  output txrdy;

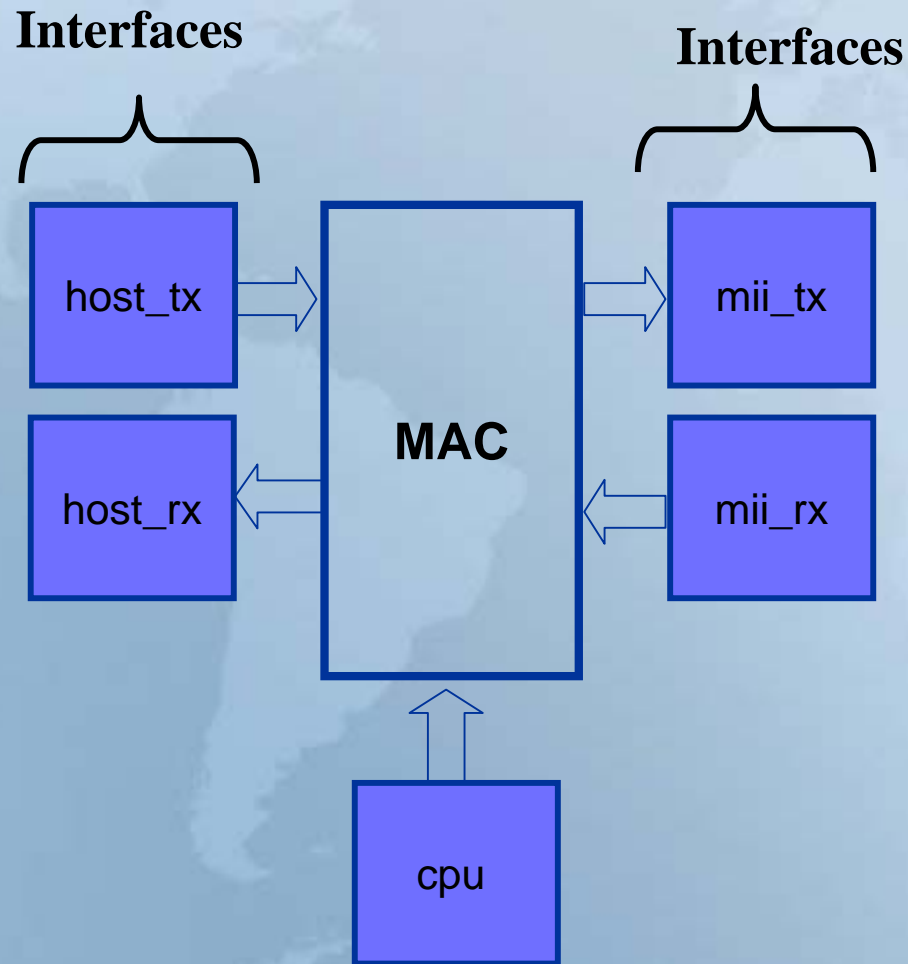
  task drive_packet (Packet packet);
  ...
  endtask
endinterface

interface host_rx;
...
interface mii_tx;
...
interface mii_rx;
...

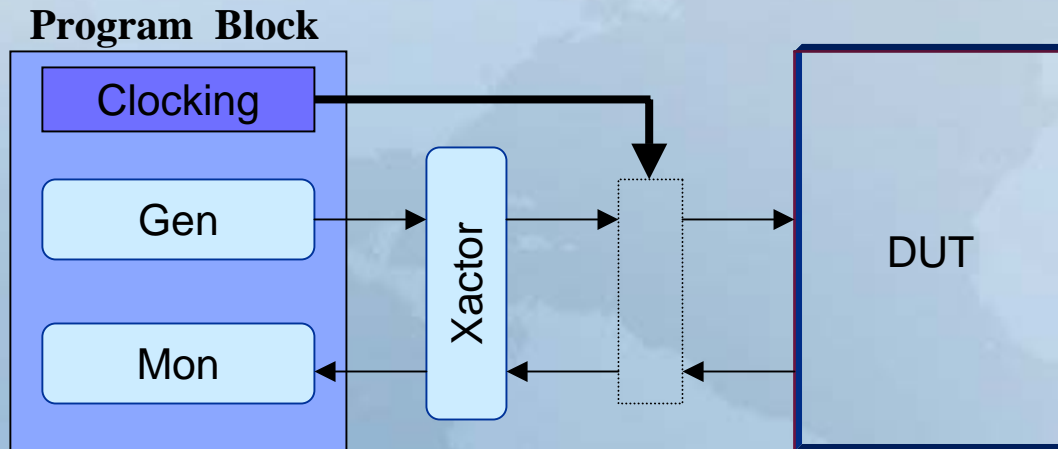
```



interfaces as Transactors



Using clocking to interface to DUT



```
clocking mac_host_tx_host @(posedge CLK);  
  input  [31:0] txdata;  
  input  txselect;  
  input  txsop;  
  input  txeop;  
  input  txen;  
  output txrdy;  
endclocking
```



Program Block

```
program mac_tb (host_tx, host_rx, mii_rx, mii_tx);

    clocking mac_host_tx_host @(posedge CLK);
    . . .
endclocking

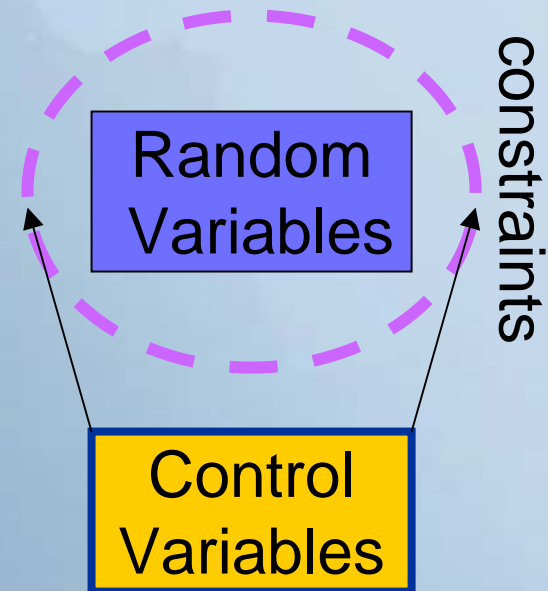
Host_gen          host_gen;
Host_mon          host_mon;
MII_gen           mii_gen;
Mii_mon           mii_mon;

initial begin
    host_gen = new (...);
    host_mon = new ();
    mii_gen = new ();
    mii_mon = new (...);
end
endprogram
```



Generators

- Implemented as classes instantiated inside **program** block
- Use random variables and constraints to generate stimulus
- Three components
 - Random variables
 - Control variables
 - Constraints



Packet Generator: Overview

- Must generate Ethernet packets
- Host Side, MII Side
- Vary length from 64- to 1522– byte packages
- Generate destination address
 - Dest. address match MAC address or not match MAC address
 - Unicast, multicast or broadcast addresses
- Calculate CRC



A Simple Packet Class

```
class Packet;
  rand bit [47:0] sa;
  rand bit [47:0] da;
  rand bit [15:0] typeLen;
  rand byte      payld [];
  rand bit [31:0] crc;

  //control variables
  int min_length;
  int max_length;
  bit [47:0] mac_addr;

  constraint len_lim {
    payld.size >= min_length;
    payld.size <= max_length;
  }

  extern task new ();
  extern task calc_crc ();
  extern task set_lim (int maxlen,
                      int minlen=64);
  extern task pr ();
  extern function Packet cp ();
  extern task set_mac_addr ();

endclass
```

Random
variables

Dynamic array
Both length and contents
will be randomized

Control
variables

Constraints

Methods



Destination Address Generation

```
typedef enum {unicast, multicast, bcast} pkt_type;

class GenDa ;
    bit [47:0] mac_addr;
    rand bit [47:0] dest_addr;
    rand pkt_type pktType;
    rand bit match_mac_addr;
    rand bit [1:0] cast;

    constraint da_lim {
        match_mac_addr=>(dest_addr == mac_addr);
    }

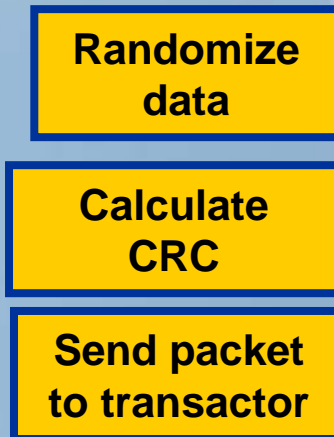
    constraint cast_lim {
        (pktType == unicast)=> cast = 2'b10;
        (pktType == multicast) => cast = 2'b11;
        (pktType == bcast)=> dest_addr= 48{1'b1};
    }
    extern task new ();
    extern task set_mac_addr ();
    task post_randomize ();
        if ((pktType == unicast) || (pktType == multicast))
            dest_addr [47:46] = cast [1:0];
    endtask
endclass
```

User-Defined
post_randomize
method called
automatically after
randomize



Simple Packet Generator

- Uses
 - Packet randomization class
 - Destination address generation class



```
class Gen;
    rand Packet    packet;
    rand Destaddr  destaddr;
    rand int  pktcnt;
    extern task genPkt ();
    extern task send_packet ();
endclass

task Gen::genPkt ();
    set_ctl_var ();
    if (!this.randomize ())
        $display ("randomize failed");
    for (i=0; i < pktcnt; i++) begin
        void=packet.randomize();
        packet.calc_crc ();
        send_packet ();
        repeat (10)@(posedge CLOCK);
    end
endtask
```

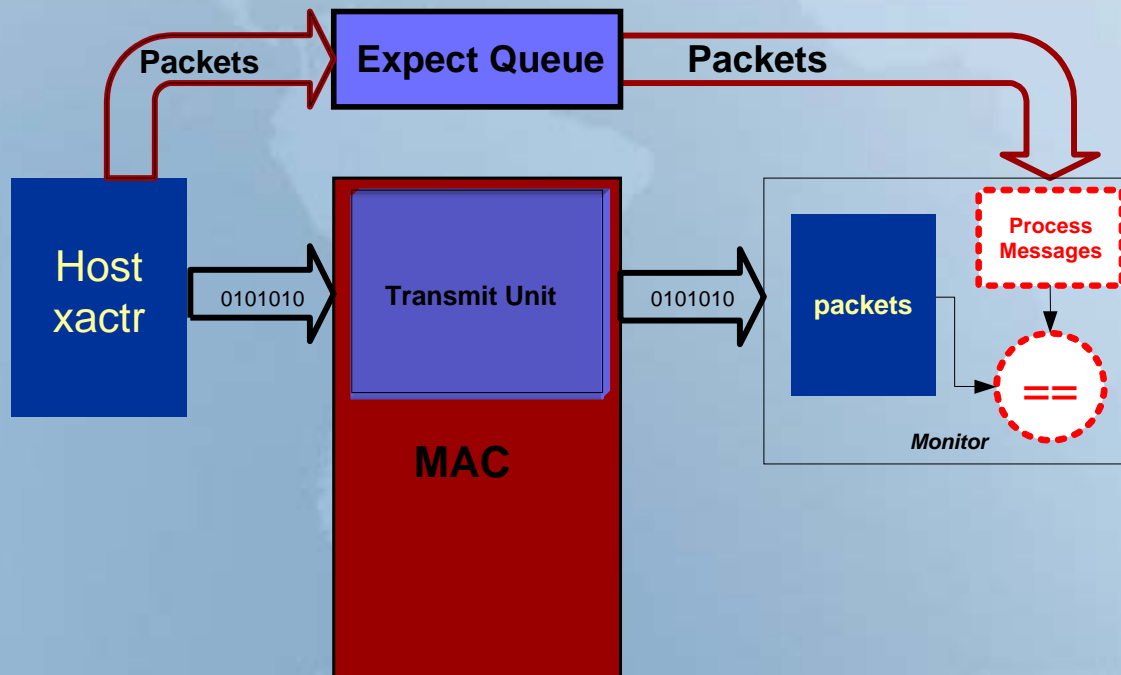
Checkers and monitors: Overview

- One checker on each side
 - Host side
 - MII side
- Relies on expect queues to determine correctness of packet received
 - Implemented using mailboxes
- Issue with packet drops



Testbench Implementation : Response Checking

- Expect Queue predicts packet arrival order
 - Issue with discontinuities in packet arrival
 - Packet drops



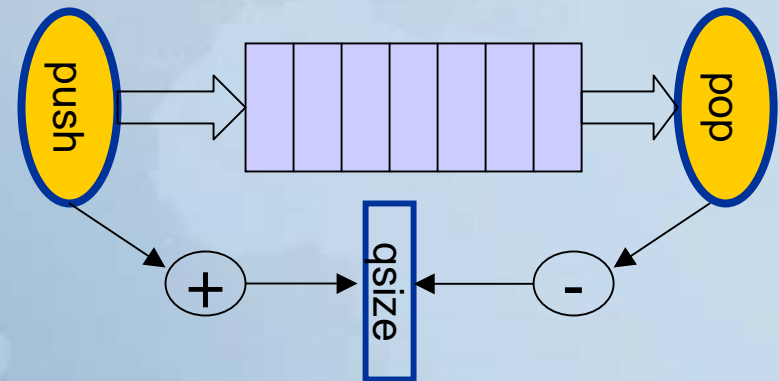
Implementation of Expect Queue

- Simple FIFO implemented using mailboxes
 - Handling dropped packets
 - Occupancy counter to predict overflows in the FIFO
 - Drop incoming packets on FIFO overflow
- Linked list implementation using classes
 - Packets removed from queue if dropped

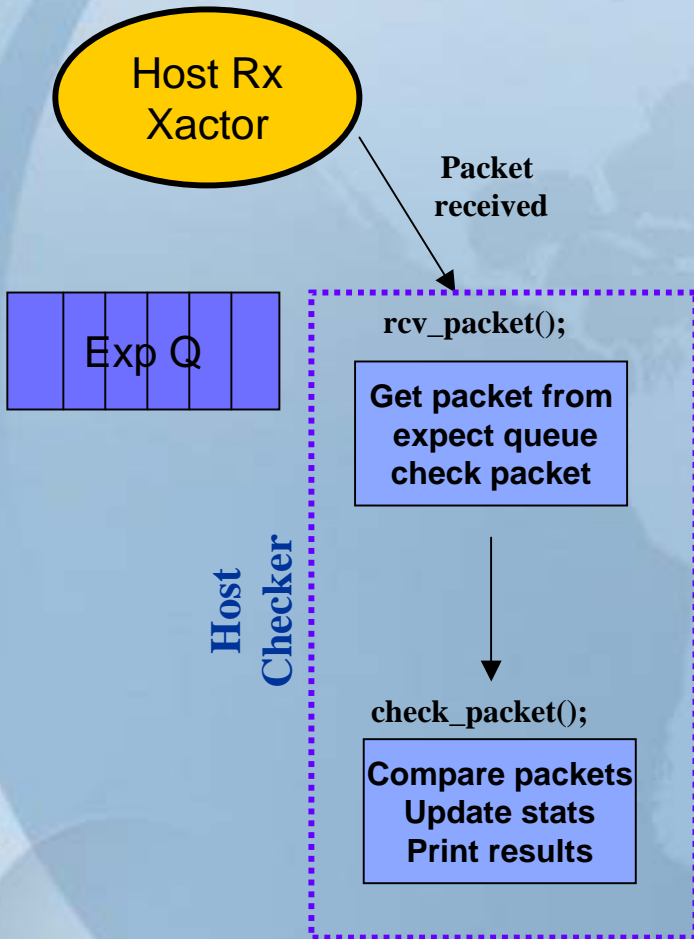


Implementation of Expect Queue

```
class Expqueue;  
    mailbox#(Packet) exp_mbx;  
    int qsize;  
    int max_qsize;  
  
    function new (int max_size, int mbxid);  
    extern task push(Packet p);  
    extern task pop (Packet p);  
endclass  
  
task Expqueue::push (Packet p);  
    if ((qsize+1) <= max_qsize) begin  
        exp_mbx.put (p);  
        qsize++;  
    end  
endtask  
task Expqueue::pop (Packet p);  
    exp_mbx.get (p);  
    qsize --;  
endtask
```



Example Checker



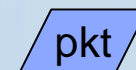
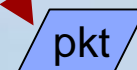
```
class Hostchk ;  
    mailbox#(Packet) exp_mbx;  
    Packet  rx_packet;  
    Packet  exp_packet;  
  
    Stats   test_stats;  
  
    extern function new (Stats stats, int mblen=0);  
    extern task  rcv_packet (Packet rcv_p);  
    extern task  check_packet ();  
endclass  
  
task Hostchk::new (Stats stats, int mblen=0);  
    test_stats =stats;  
    exp_mbx = new(mblen);  
endtask
```



Example Checker (cont'd)

```
task Hostchk::rcv_packet (Packet rcv_p) ;  
  rx_packet=rcv_p;  
  exp_pkt_cnt=expmbx.try_get(exp_packet);  
  if (exp_pkt_cnt == 0)  
    $display ("ERROR:Unexpcted packet Rcvd");  
  else  
    check_packet ();  
endtask
```

```
task Hostchk::check_packet ();  
  if (rx_packet.cmp(exp_packet)) begin  
    $display ("Packet Received");  
    rx_packet.pr();  
    test_stats.rx_good_packet(rx_packet);  
  end  
  else begin  
    $display ("ERROR: Incorrect packet Rcvd");  
    rx_packet.pr();  
    exp_packet.pr();  
    test_stats.rx_err_packet(rx_packet,err_cause);  
  end  
endtask
```



Coverage tracking

- Need to grade the effectiveness of the test
 - Did the desired events happen
 - What is the quality of the stimulus
- Create a statistics class
 - Used to track stimulus and response statistics
 - For example: Total packets generated and total packets received
 - Can also be used to track events of interest
 - For example: Total packets dropped, Queue occupancy



Example of coverage class

Track statistics to grade test effectiveness

```
class Counter ;
  int  count;
  int  limit;

  extern task new (int lim=1000);
  extern task inc ();
  extern task dec ();
endclass

class Stats;
  Counter counter [string];
  Counter event_counter [string];
  task new ();
    counter["GenPkts"] = new;
    counter ["RxPkts"] = new;
    counter ["UcPkts"] = new;
    counter ["McPkts"] = new;
    event_counter ["DropPkts"] = new;
  endtask
  extern task inc_stats (string PktType);
  extern task inc_events (string EventType);
  extern task pr ();
endclass
```

Assoc array
of counters

Indexed by
counter name



Style Guidelines

- One class hierarchy per file
- File name same as base class name
- Class name capitalized
 - Hostxactor
- Define methods external to class
- Constraints should be defined external to class
- Use interfaces for BFM, drivers or transactors



Summary

- Use classes where possible
 - Layered classes make it easier to build sophisticated structures
- Use interfaces to abstract out physical ports
 - More concise code
- Keep to higher levels of abstraction
- New SystemVerilog features let you add automation to your testbench
- Let's you focus on corner cases



Agenda

Introduction:

SystemVerilog Motivation

Vassilios Gerousis, Infineon Technologies
Accellera Technical Committee Chair

Session 1:

SystemVerilog for Design

Language Tutorial

Johny Srouji, Intel

User Experience

Matt Maidment, Intel

Session 2:

SystemVerilog for Verification

Language Tutorial

Tom Fitzpatrick, Synopsys

User Experience

Faisal Haque, Verification Central

Lunch: 12:15 – 1:00pm

Session 3: SystemVerilog Assertions

Language Tutorial

Bassam Tabbara, Novas Software

Tecnology and User Experience

Alon Flaisher, Intel

Using SystemVerilog Assertions and Testbench Together

Jon Michelson, Verification Central

Session 4: SystemVerilog APIs

Doug Warmke, Model Technology

Session 5: SystemVerilog Momentum

Verilog2001 to SystemVerilog

Stuart Sutherland, Sutherland HDL

SystemVerilog Industry Support

Vassilios Gerousis, Infineon

End: 5:00pm

