System Verilog

Zach Stechly

Goals of Presentation

- Increase familiarity with the basics of SystemVerilog
- Improve testbenches with constrained random tests
- Utilize Direct Programming Interface (DPI) to interact with functions written in C
- Getting it all to work in Windows with Modelsim or Questasim

Presentation is based on SystemVerilog for verification purposes, not for design

SystemVerilog

- SystemVerilog is an IEEE approved Hardware Verification Language (HVL)
- SystemVerilog = Verilog + Object Oriented Programming
  - Classes, inheritance, structures, functions
  - Interfaces, new data types, dynamic arrays
  - Queues, type casting
  - Parallel threads
  - Constrained Randomization
  - A way to access individual bits of a multidimensional array

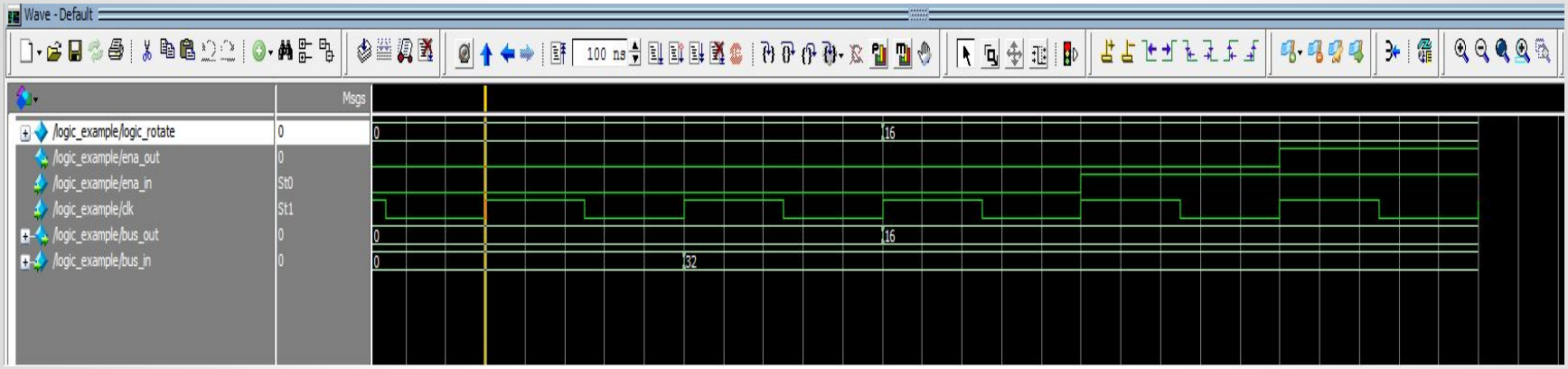
Data Type Overview -- 4-State

- Reg / Wire -- still in SystemVerilog
  - Can be 0,1,x,z
  - Known as 4-State variables
  - Reg can be driven in continuous statement now
    - Still cannot be driven by a module output
- Logic
  - Generic 4-state, can be used any way you want
    - (Module output, continuous assignment)
  - Cannot have more than one driver
    - No bidirectional bus modeling, use wire

Logic Example

```
module logic_example(
  input    wire              clk,
  input    logic [15:00]   bus_in,
  input    logic           ena_in,
  output  logic [15:00]  bus_out,
  output   logic          ena_out
);

logic [15:00] logic_rotate;
always @(posedge clk) begin
  logic_rotate   <= {bus_in[0],bus_in[15:01]};
  ena_out        <= ena_in;
end
  assign bus_out  = logic_rotate;

endmodule
```

Modelsim Results



Data Type Overview -- 2 State

Can only take on values based on 0,1

- bit -- single 0 or 1
  - Can be used like any single bit object
  - bit [31:00] my_bits; // equal to int unsigned
- int -- 32-bit signed integer
- int unsigned -- 32-bit unsigned integer
- byte -- 8-bit signed integer
- shortint -- 16-bit signed integer
- longint -- 64-bit signed integer
- real -- double precision variable

2-State Variable Examples

```
module two_state();
real       pi;
bit [3:0] bit_test;
byte       byte_test; // same as above
initial begin
   pi             = 3.1415;
   bit_test       = 4'd7;
   byte_test      = 4'd7;
end
endmodule
```

Modelsim Results

Static Arrays

- All previous data types can become arrays
  - int my_array[16] --array of [15 downto 0]
  - real my_double[16][16] -- square array
  - logic [4:0] my_bits[34];
- (Actually in Verilog-2001)
  - Individual bits of my_bits can be accessed
    - my_bits[0] = first element in array
    - my_bits[1][0] = bit [0] of element [1]

Static Array Example with foreach

```
int static_array[10][20];

initial begin
foreach(static_array[i,k])
   static_array[i][k] = i*k;
foreach(static_array[i,k])
   $display(static_array[i][k]);
end
```

Modelsim Result

```
run 1 us
#           0  #           0  #           0  #           0  #           0
#           0  #           2  #           4  #           6  #           8
#           0  #           4  #           8  #          12  #          16
#           0  #           6  #          12  #          18  #          24
#           0  #           8  #          16  #          24  #          32
#           0  #          10  #          20  #          30  #          40
#           0  #          12  #          24  #          36  #          48
#           0  #          14  #          28  #          42  #          56
#           0  #          16  #          32  #          48  #          64
#           0  #          18  #          36  #          54  #          72
#           0  #          20  #          40  #          60  #          80
#           0  #          22  #          44  #          66  #          88
#           0  #          24  #          48  #          72  #          96
#           0  #          26  #          52  #          78  #         104
#           0  #          28  #          56  #          84  #         112
#           0  #          30  #          60  #          90  #         120
#           0  #          32  #          64  #          96  #         128
#           0  #          34  #          68  #         102  #         136
#           0  #          36  #          72  #         108  #         144
#           0  #          38  #          76  #         114  #         152
#           0  #           0  #           0  #           0  #           0
#           1  #           3  #           5  #           7  #           9
#           2  #           6  #          10  #          14  #          18
#           3  #           9  #          15  #          21  #          27
#           4  #          12  #          20  #          28  #          36
#           5  #          15  #          25  #          35  #          45
#           6  #          18  #          30  #          42  #          54
#           7  #          21  #          35  #          49  #          63
#           8  #          24  #          40  #          56  #          72
#           9  #          27  #          45  #          63  #          81
#          10  #          30  #          50  #          70  #          90
#          11  #          33  #          55  #          77  #          99
#          12  #          36  #          60  #          84  #         108
#          13  #          39  #          65  #          91  #         117
#          14  #          42  #          70  #          98  #         126
#          15  #          45  #          75  #         105  #         135
#          16  #          48  #          80  #         112  #         144
#          17  #          51  #          85  #         119  #         153
#          18  #          54  #          90  #         126  #         162
#          19  #          57  #          95  #         133  #         171
```

Dynamic Arrays

- Arrays can be declared dynamically
  - int x[]; reg [13:00] x[];
  - Memory is allocated with new
  - Can be cleaned up when finished with delete
- Queues
  - A dynamic array with fifo-like qualities
  - Queues are declared, filled, and emptied as needed
  - Great for dealing with time alignment between ideal output and PL output
  - Really an extension of arrays, so all array methods should work with queues

Dynamic Array / Queue Methods

find() returns all the elements satisfying the given expression
find_index() returns the indexes of all the elements satisfying the given expression
find_first() returns the first element satisfying the given expression
find_first_index() returns the index of the first element satisfying the given expression
find_last() returns the last element satisfying the given expression
find_last_index() returns the index of the last element satisfying the given expression.

Queue Specific Methods

Method/Operation	Description
q[a]	Indexing Operation -- returns the element at the given index. If a is out of range, then the default value of the elements' type is returned.
q[a:b]	Slice operation -- returns a queue with the specified range of elements. If a is negative start with index 0, if b is greater than the index of the last element of the queue, the result is limited to the last element of the given queue.
q.size()	Returns the count of elements in the queue.
q.insert(index,item)	Inserts the *item* at the given *index*. The first argument *index* has to be an `int`, and the second argument *item* has to be of the type as specified for queue's elements. In case the list is *bounded*, and the queue is full up to the specified bounding size, the last element of the queue would be silently dropped.
q.delete(index)	The element at the specified *index* is dropped from the queue.
q.delete()	The method *delete* is overloaded. When no *index* is specified, the queue is purged of all its elements. Equivalent to saying `q = {}`.
q.push_back(item)	The given item is added to the back of the queue. If the queue is bounded and full, this operation fails.
q.pop_back()	Returns and drops the last element from the queue.
q.push_front(item)	The given item is added to the front of the queue. If the queue is bounded and full, the last element of the queue would be silently dropped.
q.pop_front()	Returns and drops the first element from the queue.

Queues and Dynamic Array Examples

Queue Example

```
logic [35:00] even_hdr_q[$];
logic [35:00] even_data0_q[$];
logic [35:00] even_data1_q[$];
logic [35:00] odd_hdr_q[$];
logic [35:00] odd_data0_q[$];
logic [35:00] odd_data1_q[$];

// push everything into queues
always @(posedge clk) begin
if (even_sop) begin
  even_hdr_q.push_back(even_hdr);
end
if (even_ena) begin
  even_data0_q.push_back(even_data0);
  even_data1_q.push_back(even_data1);
end
```

Queue Example Continued...

```
always @* begin
if ((even_data0_q.size() > 1440) && (odd_data0_q.size() > 1440)) begin
// output something
for (int y = 0; y < 48; y++) begin
  for (int x = 0; x < 15; x++) begin
  @(posedge clk);
  if (x == 0) begin
    bsi_sos    <= 1'b1;
  end else begin
  bsi_sos  <= 1'b0;
  end
  bsi_ena  <= 1'b1;
  bsi_data0 <= even_data0_q.pop_front;
  bsi_data2  <= even_data1_q.pop_front;
  bsi_data2  <= even_data0_q.pop_front;
  bsi_data3  <= even_data1_q.pop_front;
  end
  end
  end else begin
    bsi_ena    <= 1'b0;
    bsi_sos    <= 1'b0;
    bsi_prb    <= 1'b0;
  end
  end
```

Dynamic Array Example

```
int dynamic_array[];
initial begin
  dynamic_array = new[100];
  dynamic_array[5] = 96;
  $display(dynamic_array[5]);
  $display(dynamic_array[4]);
  $display(dynamic_array.size());
  dynamic_array.delete();
end
```

Modelsim Result

Note that accessing a non-initialized array element returns the default case for that type of variable (in our case, a 2-state variable returns 0)

```
# Loading sv_std.std
# Loading work.dyn_array
VSIM 37> run 1 us
#           96
#            0
#          100
```

Data Type Summary

- Different data types can be used for different purposes
  - In larger simulations where memory use is going to become a problem, 2-state variables come in handy
- New dynamic array abilities allow real-time memory allocation
- Queues are great, have all the features of arrays (sorting, find_next, find), plus more

Functions and Tasks

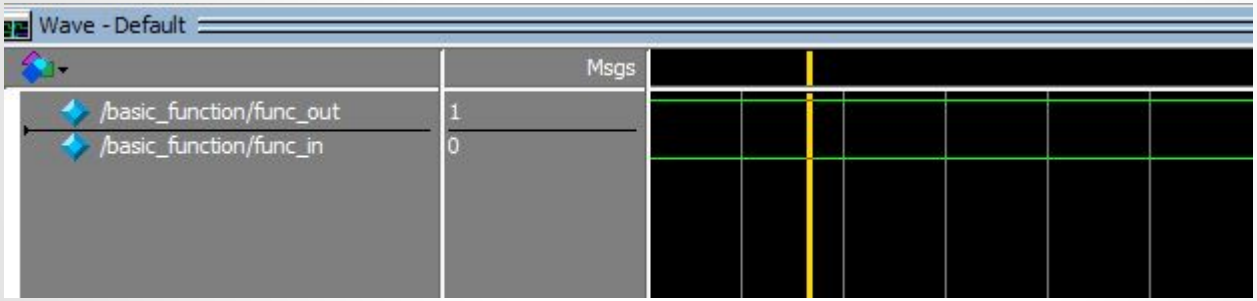
- Work more like functions in C then functions in Verilog
- Functions cannot consume time in System Verilog, so no blocking statements
  - #1000, @(posedge clk)...
  - tasks, however, can
- Functions in SystemVerilog can pass copies, or references
- Functions in SystemVerilog can use 'return' statements
  - This includes returning arrays or classes, etc
  - Arrays must be passed via reference

Function / Task Example

```
// task example with C-style formatting...
task task_name(input logic [13:00] x, output   logic [14:00] y);

endtask

// return example
function bit negate(bit a);
   return ~a;
endfunction
```

Modelsim Results

```
VSIM 42> run 1 us
# 0
# 1
add wave \
```



Function with Reference

```
module automatic function_ref();

//// function example passing an array...via reference
function void change_array(ref int a[]);
  foreach(a[i])
   a[i] = i*2;
endfunction
```

Functions with References Cont.

```
int my_array[];
initial begin
  my_array = new[10];
// initialize array
 foreach(my_array[i])
   my_array[i] = i;
$display("Display initial array results");
 foreach(my_array[i])
   $display(my_array[i]);
// call function on array
change_array(my_array);
// now print this out again
$display("Display post function array results");
 foreach(my_array[i])
   $display(my_array[i]);
end
endmodule
```

Modelsim Result

```
VSIM 60> run 1 us
# Display initial array results
#           0
#           1
#           2
#           3
#           4
#           5
#           6
#           7
#           8
#           9
# Display post function array results
#           0
#           2
#           4
#           6
#           8
#          10
#          12
#          14
#          16
#          18

VSIM 61>
```

Some notes...

We needed to declare that module 'automatic', which means that variables are written to the stack

Non-automatic programs use a common, static memory, and passing things by reference (or using threads) can cause sharing issues

To pass arrays via reference, they need to be dynamic

The Idea of Scoreboards

- Combining functions, new array sorting / searching abilities, and queues, we can create a 'scoreboard'
- A scoreboard compares ideal output with module under test output, and verify that said output occurs at some time, and only one time, or whatever ruleset you define
- examples make better sense...
  - a memory can be written to randomly but each address is only written to once

Scoreboard Example

```
module  basic_scoreboard();
// create queue of values
initial begin
// create random array..with intentional error
int data[] = '{1,3,2,5,7,6,8,11,7,9,4,10,12,14,13};
// check our data

int results[$];
// imagine I put this in a function
for (int x = 0; x < 14; x++) begin
  results = data.find_index() with (item == x);
  case(results.size())
  0: $display("Value %h not found",x);
  1: $display("Value %h found exactly once",x);
  default: $display("COLLISION AT VALUE OF %h",x);
  endcase
end // end loop
end
endmodule
```

Modelsim Results

```
# Loading sv_std.std
# Loading work.basic_scoreboard
VSIM 92> run 1us
# Value 00000000 not found
# Value 00000001 found exactly once
# Value 00000002 found exactly once
# Value 00000003 found exactly once
# Value 00000004 found exactly once
# Value 00000005 found exactly once
# Value 00000006 found exactly once
# COLLISION AT VALUE OF 00000007
# Value 00000008 found exactly once
# Value 00000009 found exactly once
# Value 0000000a found exactly once
# Value 0000000b found exactly once
# Value 0000000c found exactly once
# Value 0000000d found exactly once
```

Interfaces

- Abstract representations of functions, similar to interfaces in Java / C++
- Lets say Module 1 connects to Module 2 and Module 3
  - Can just declare a single interface variable with logic ports, and assign that interface variable to all three modules
    - recall logic variables can be inputs or outputs
    - Interface just declares the signal name as logic, not if it is an input or output
  - If something in the interface changes, instead of updating 3 modules, you only update the interface

Interface Examples

- First, the interface is defined

```
interface pkt_interface( input bit clk);
                logic [31:0]   hdr_in;
                logic          sop_in;
                logic          ena_in;
                logic [31:00]  data0_in;
                logic          ena_out;
                logic          sop_out;
                logic [31:00]  data0_out;
endinterface
```

Interface Examples

- Then define modules to use the interface

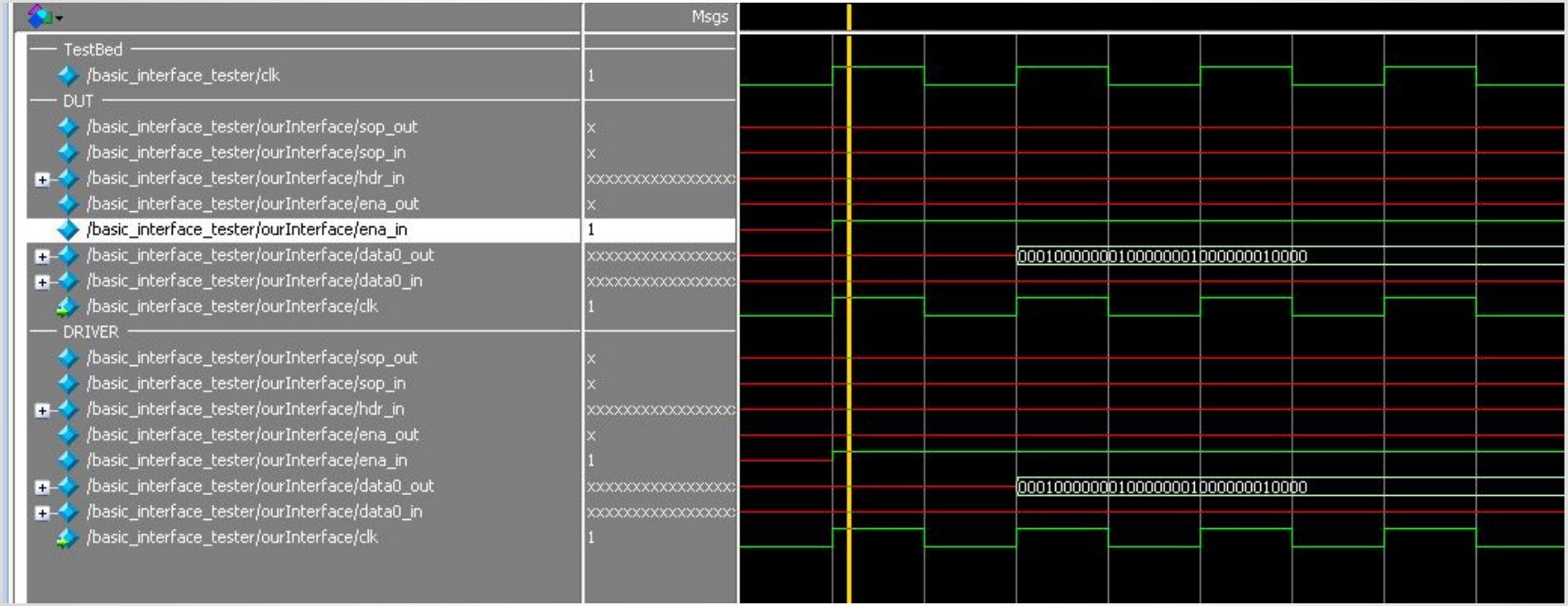
```
module base_interface_device(pkt_interface filter_sig);
  always @(posedge filter_sig.clk) begin
     if (filter_sig.ena_in) begin
       filter_sig.data0_out       <= 32'h10101010;
     end
  end
endmodule

module basic_interface_driver(pkt_interface test_sig);
   always @(posedge test_sig.clk) begin
     test_sig.ena_in    <= 1;
   end
endmodule
```

Interface Examples

- Finally, in a top level file...tie everything together
  - Treat as any other variable

```
    // simulate me
module basic_interface_tester();
  logic clk;
  pkt_interface ourInterface(clk);
  initial begin
    clk = 0;
    // send our clock off
    forever #50 clk = ~clk;
    $display("ending initial block");
  end
  base_interface_device our_device(
    .filter_sig    ( ourInterface)
  );basic_interface_driver out_driver(
    .test_sig      (  ourInterface)
  );
```

Modelsim Results



More Detailed Interfaces

- Modports
  - Allows multiple specific defining of input, output ports
- Clocking Blocks
  - Can specify timing with respect to signals in interface
  - ie: A signal will only change with respect to which clock it is blocked with, regardless of 'system time'

Interface with Modport

```
// define interface
interface eth_interface;
  bit clk;
  bit reset;
  Eth_Packet pkt_in;
  Eth_Packet pkt_out;

  modport TEST (input clk, input reset, input pkt_out,  output pkt_in);
  modport DUT  (input clk, input reset, output pkt_out, input  pkt_in);


endinterface
```

Interface with Clocking Block

```
interface arb_if(input bit clk);
  logic [1:0] grant, request;
  logic rst;

  clocking cb @(posedge clk);
    output request;
    input grant;
  endclocking

  modport TEST (clocking cb, output rst);

  modput DUT (input request, rst, output grant);
endinterface
```

Interface with Clocking Block cont.

```
module test(arb_if.TEST arbif);
  initial begin
     arbif.cb.request = 0;
     @arbif.cb;  // change occurs when next clock edge since request is tied to clock
  end
endmodule
```

Parallel Computing with Threads

Moving on from Fork...

- Join
  - The parent process blocks until all the processes spawned by this fork complete.
- Join Any
  - The parent process blocks until any one of the processes spawned by this fork complete.
- Join None
  - The parent process continues to execute concurrently with all the processes spawned by the fork.

Fork / Join Example

```
initial
begin
    clk = 0;
    a = 1;
    b = 1;
    #5;
    fork
       #5  a = 0;
       #10 b = 0;
    join
    clk = 1;
end
//clk becomes 1 when //time=20
```

Fork / Join_Any example

```
initial
begin
    clk = 0;
    a   = 1;
    b   = 1;
    #5;
    fork
       #5  a = 0;
       #10 b = 0;
    join_any
    clk = 1;
end
//clk becomes 1 when //time=10
```

Fork / Join_None

```
initial
begin
    clk = 0;
    a   = 1;
    b   = 1;
    #5;
    fork
       #5  a = 0;
       #10 b = 0;
    join_none
    clk = 1;
end
//clk becomes 1 when //time=5
```

Classes and Object Oriented Notes

- Like in software, classes:
  - Have Constructors
  - Have Methods
  - Have Private variables
- Are created with a call via new, can be deallocated
- Can be declared in other classes
- Can be copied
- Can be extended
- Going to be used to create randomized transaction objects in our testbenches

Class Examples

- Class definition with constructor

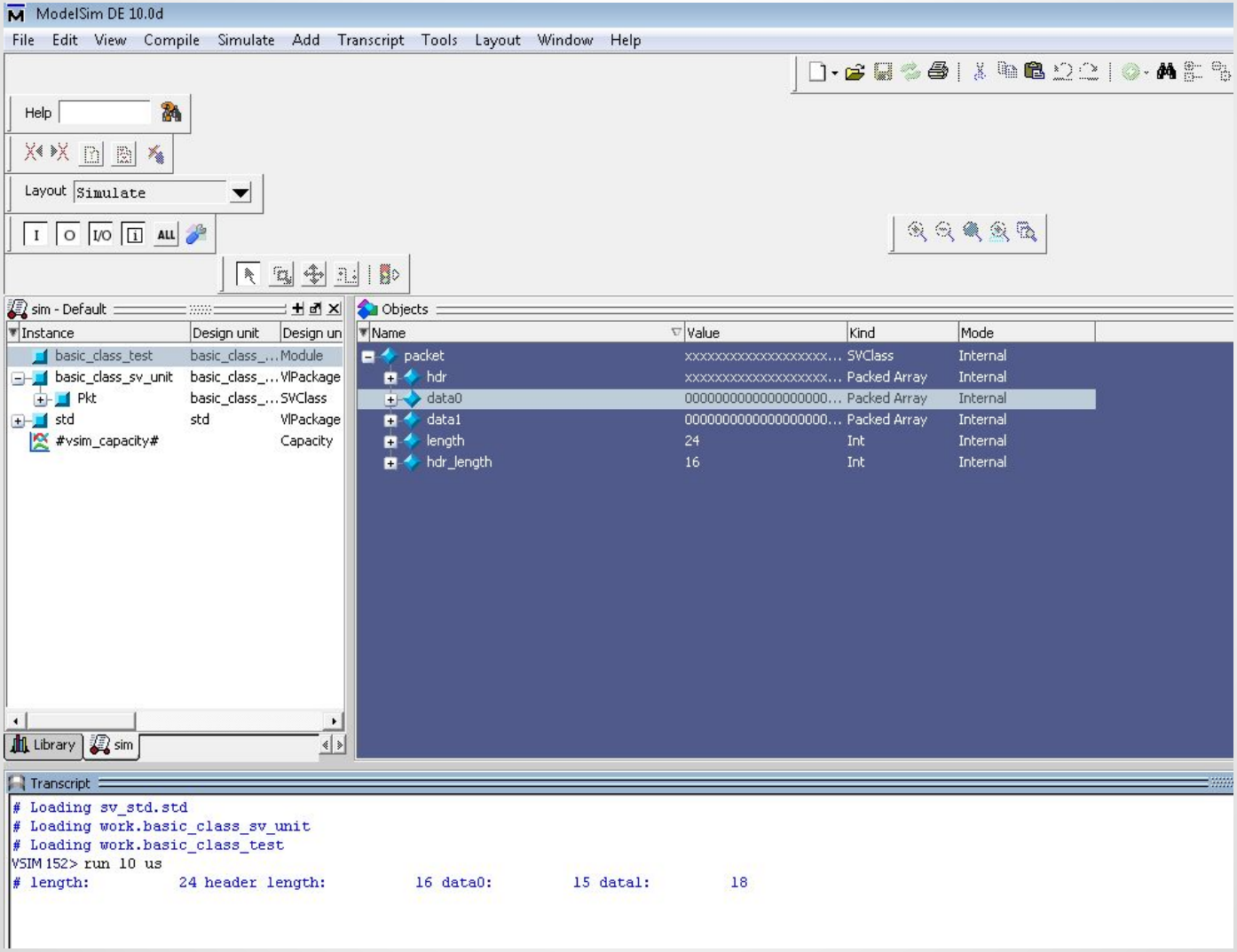
```
class Pkt;
   logic [31:00] hdr;
   logic [31:00] data0;
   logic [31:00] data1;
   int          length;
   int      hdr_length;
function new(int len=31,hdr_len=16);
   length    = len;
   hdr_length = hdr_len;
   data0     = 0;
   data1     = 0;
endfunction
function set_data(logic [31:00] data_0,data_1);
   data0      = data_0;
   data1      = data_1;
endfunction
function print_data();
   $display("length: %d header length: %d data0: %d data1: %d",length,hdr_length,data0,data1);
endfunction
```

Class Examples

- Class instantiation

```
module basic_class_test();

  Pkt packet;

  initial begin
     packet = new(24,16);  // create new packet with length 24, hdr length 16
     packet.set_data(32'd15,32'd18);
     packet.print_data();
  end

endmodule
```

Modelsim Result



Building a Better Testbench

- Direct Testing and Constrained Random Testing
- Direct Testing is what it sounds like -- send a packet which meets some criteria
  - Ideal how-data-is-supposed-to-look
  - Broken how-data-looked-on-logic-analyzer
- This can allow quick movement from simulation to implementation, but leaves you open for unanticipated errors

Building a Better Testbench

- Our ideal testbench would:
  - Generate Stimulus
  - Capture Response
  - Determine Correctness
  - Measure Completion Progress

Building a Better Testbench

- Constrained Random Testing
  - More elaborate testbench is setup
  - Data is randomized within certain constraints
    - (there is absolutely no way software will program this value to be less than 15 or greater than 35)
  - Randomized input data is sent into the device under test, as well as an ideal data processor
  - The outputs are generated compared in real time, errors are highlighted
- Direct v. Constrained Random is a tradeoff between initial setup time and debugging in lab

Randomization In SystemVerilog

- The limits can be set within a class itself with use of constraint parameter
- rand bit [5:0] pkt_len;
  - constraint x {pkt_len > 15; pkt_len < 55}
- Custom weights can be used via the dist (ribution) command
- Exponential, normal, poisson, uniform distributions all available
- Randomize everything -- delay, enable length, etc
- Specify seed at beginning to randomize
  - srandom, get_randstate, set_randstate

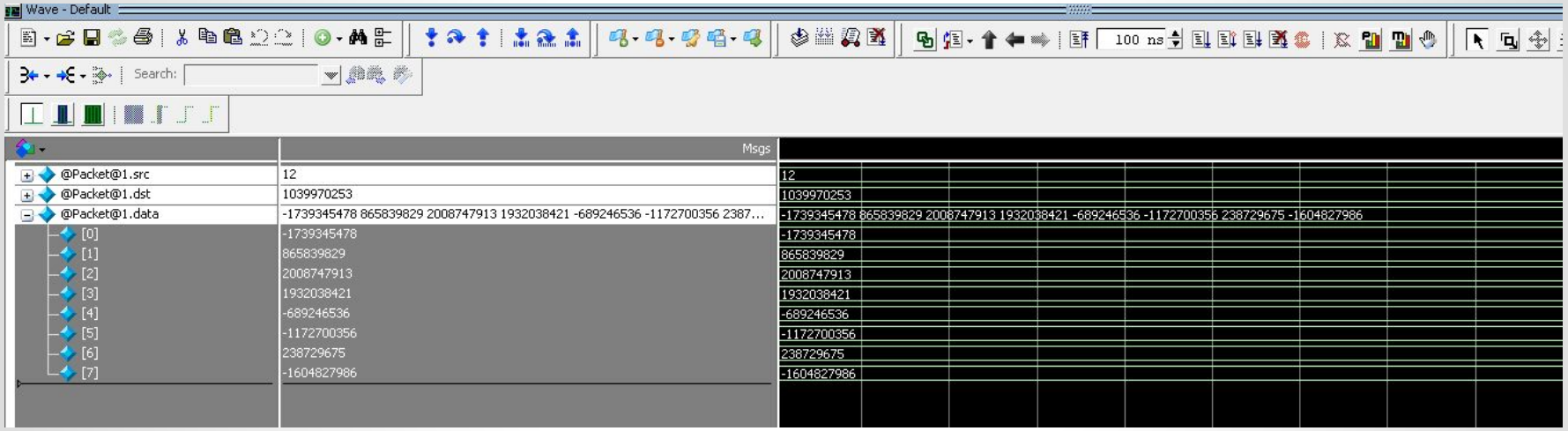
Randomization Examples

```
class Packet;
rand bit [31:0] src, dst, data[8];
constraint c {src > 10; src < 15;}
endclass

Packet p;
initial begin
p = new();
p.randomize();
end
```

License fun.....

- It appears our journey in Modelsim has to temporarily end, at least where the .randomize(); functionality is concerned
- C:/projects/systemverilog/basic_randomize.sv(10): Unable to check out verification license for randomize() feature.
- However, ViaSat does have licenses to Questa Sim, ModelSim's..brother?
- Look on the wiki for instructions

QuestaSim Results



Randomization Example Cont.

- Note: Can define variables as rand or randc
- randc cyclic, meaning it does not repeat a value until all possible values have been used
- Distributions
  - Make your own! src is 0 20% of the time, 1,2,3 80% of the time

```
rand int src, dst
constraint c_dist{
  src dist {0:=20,[1:3]:=80};
  dst dist {0:=20,[1:3]:=80};
}
```

Randomization Example Cont.

- Constraint Solver

```
rand logic [15:0] r,s,t;

constraint c_bidir{
r < t;
s == r;
t < 30;
s > 25;

}
```

Randomization Example Cont.

- Implications

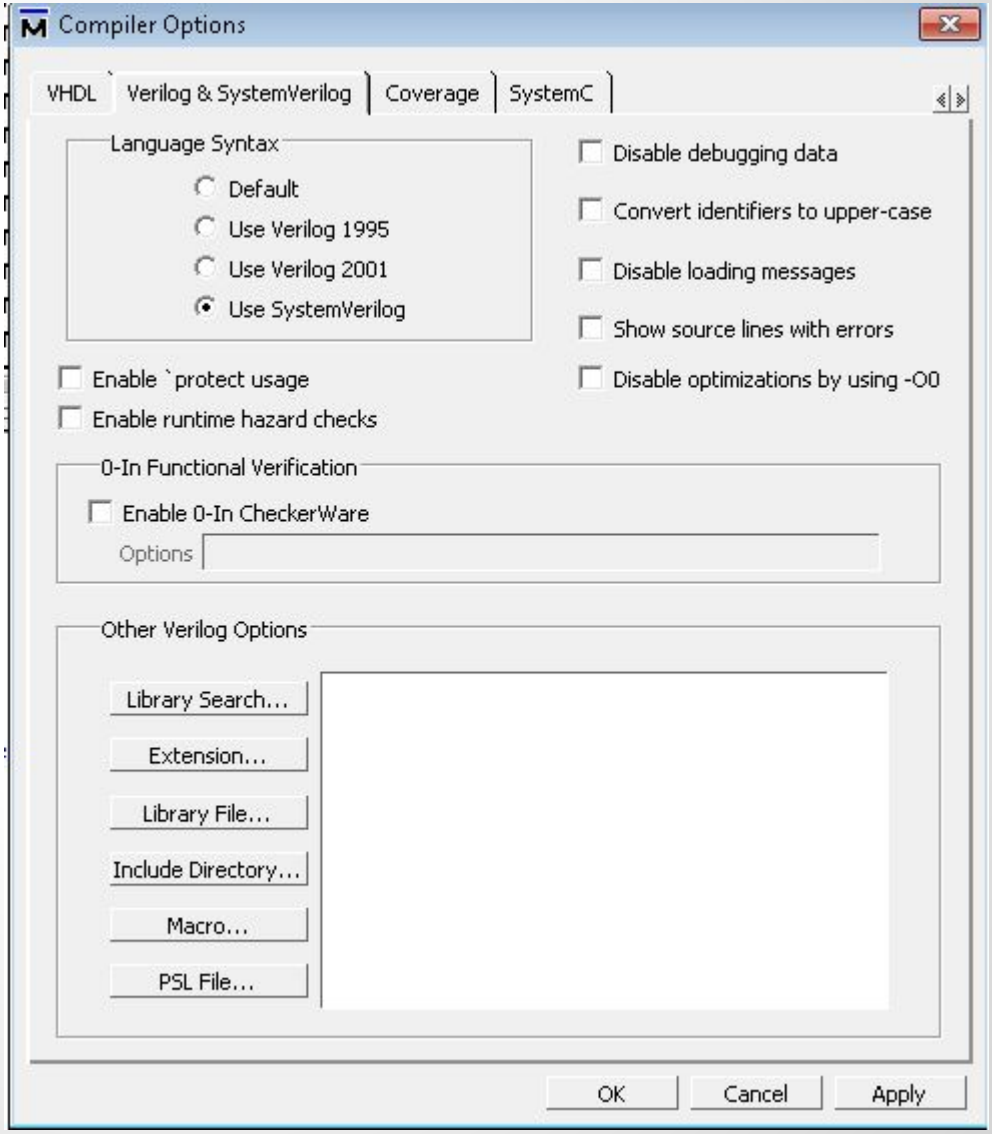
```
rand logic [15:0] r,s,t;

constraint c_bidir{
(r==0) -> t = 1;
}
```

- Fancy random number generators

```
a = $dist_exponential();
  = $dist_normal();
  = $dist_poisson();
etc
```

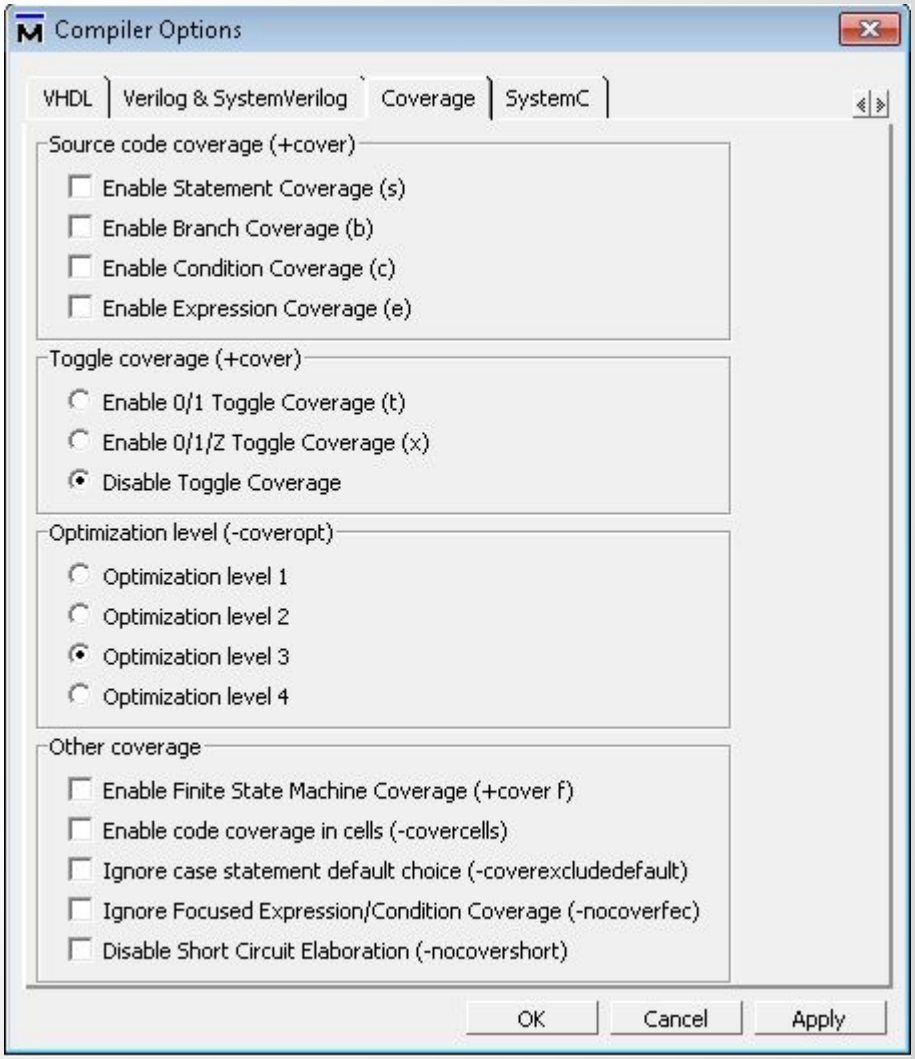
Using SystemVerilog in ModelSim

Set Compile Option to SystemVerilog

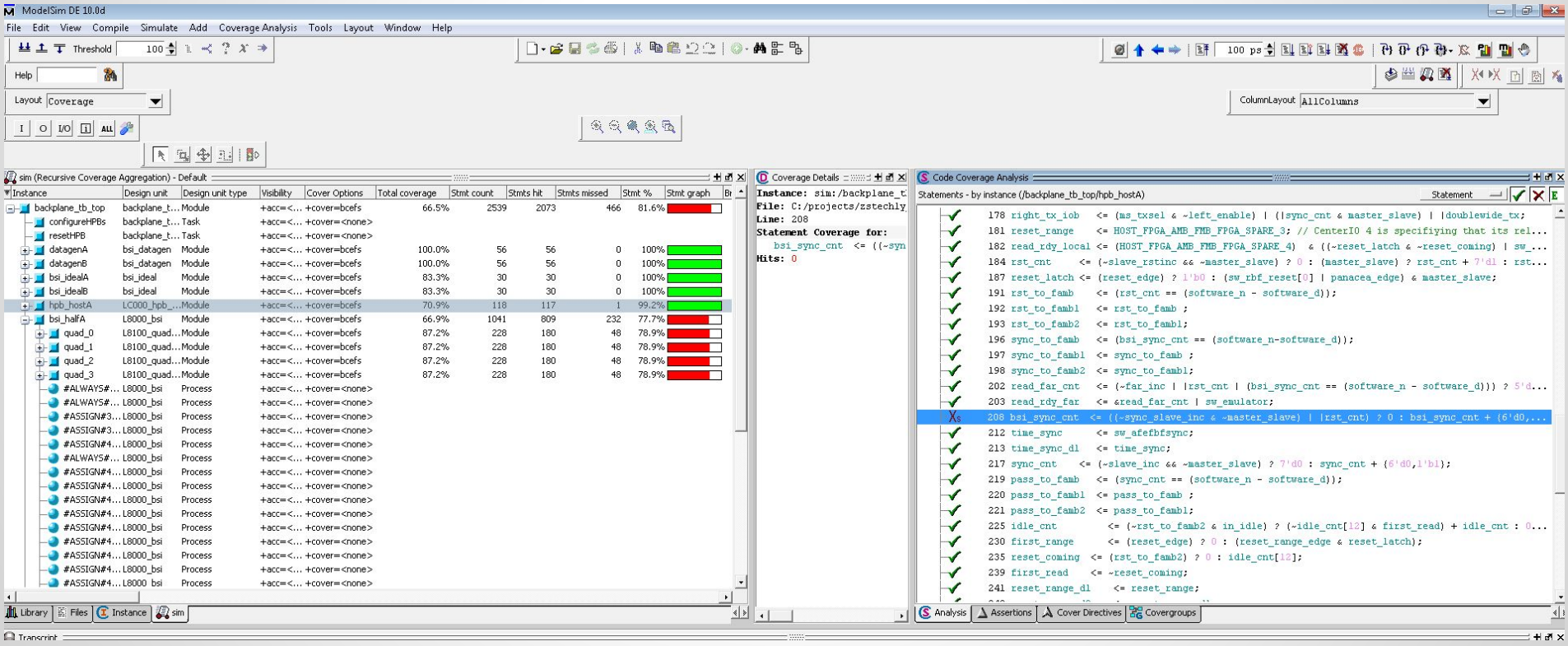


Code Coverage In Modelsim

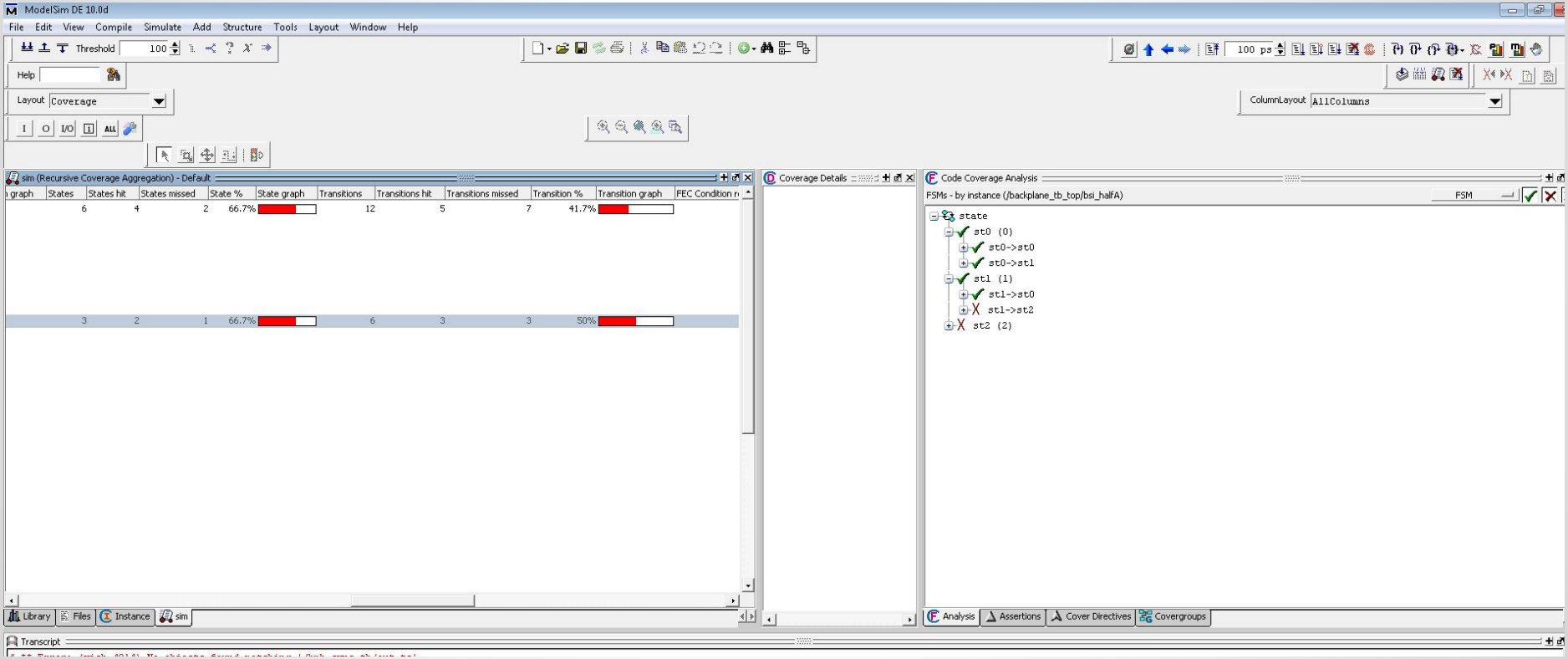
Allows many options to see how many times certain pieces of code have been touched



Code Coverage Example Report



Code Coverage Example Report



Code Coverage

- Reports for Statements, branches, expressions, FSMs, etc
  - Shows number of times each line has been touched, state machine state visited, etc
- Goal is for as close to 100% coverage as possible

Direct Programming Interface

Allows for functions written in C to be compiled in Modelsim and called from SystemVerilog testbenches

- Some things are computationally easy but prone to error in PL
- Examples
  - Real time generation of sine / cosine as simulation stimulus
  - Encrypting / Decrypting algorithm Ideal output verification
  - Ideal bit-true filter written in C, used to compare PL output in real time across all possible inputs /outputs

Example C / Verilog Code Interaction

- C Code (in linux sin,cos can be used directly)

```
#include "dpiheader.h"
#include <math.h>

double sin_func(double rTheta)
{
   return sin(rTheta);
}
```

- Verilog Code Part 1

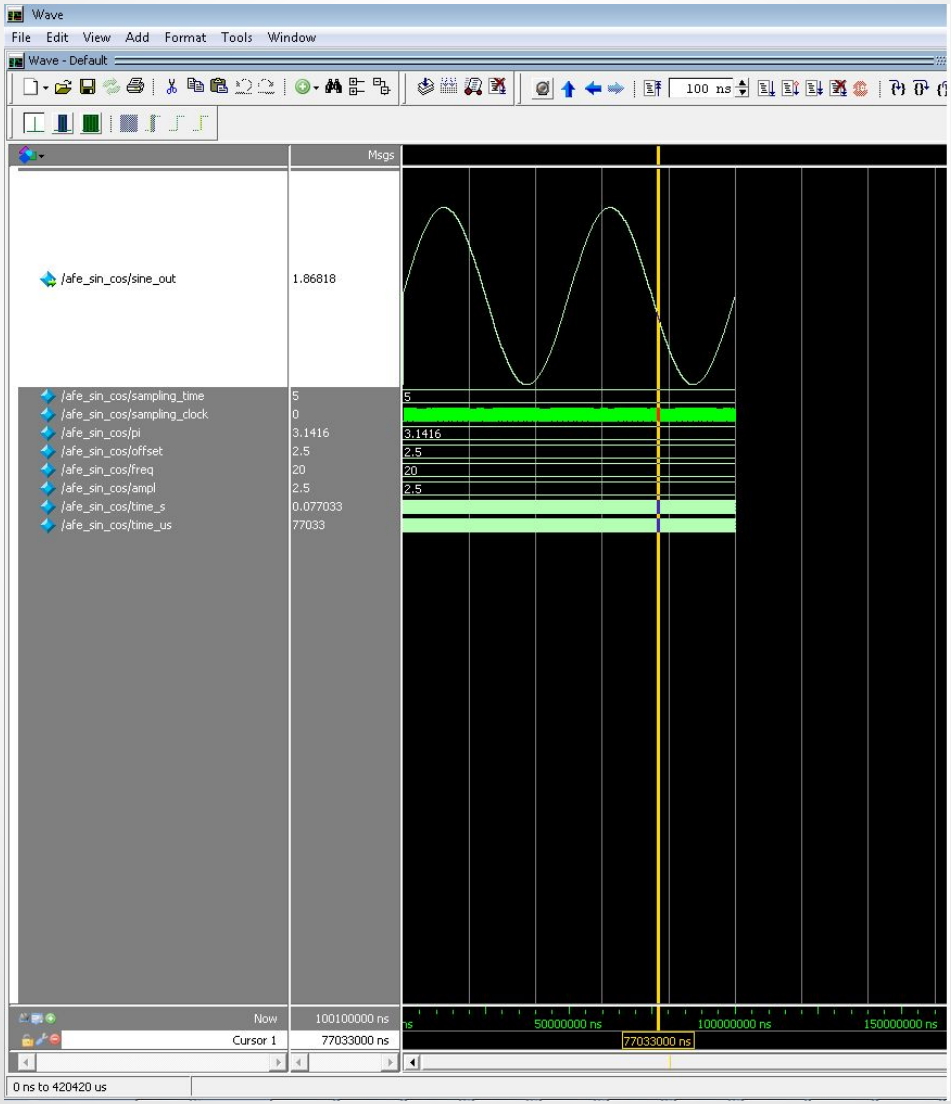
```
package math_pkg;
  //import dpi task      C Name = SV function name
  import "DPI-C" pure function real sin_func (input real rTheta);
endpackage : math_pkg
```

Example C / Verilog Code Interaction

- Verilog Code Part 2

```
module afe_sin_cos(output real sine_out);
  import math_pkg::*;
  parameter  sampling_time = 5;
  const real pi = 3.1416;
  real       time_us, time_s ;
  bit        sampling_clock;
  real       freq = 20;
  real       offset = 2.5;
  real       ampl = 2.5;

  always sampling_clock = #(sampling_time) ~sampling_clock;

  always @(sampling_clock) begin
    time_us = $time/1000;
    time_s = time_us/1000000;
  end
  assign sine_out = offset + (ampl * sin_func(2*pi*freq*time_s));
endmodule
```

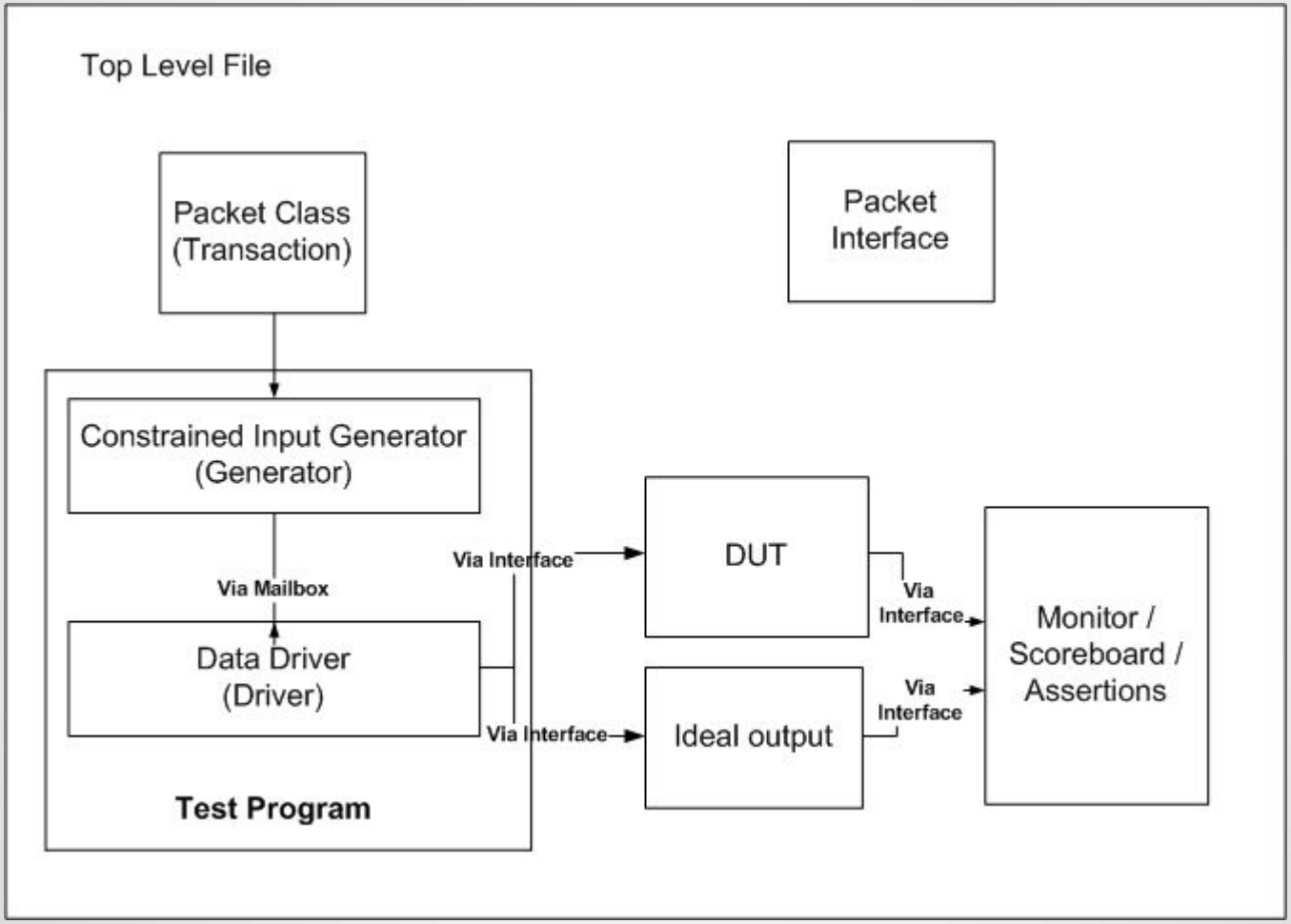
Example C / Verilog Code Interaction



Setting it up in Windows

- Unzip gcc-4.2.1-mingw32vc9.zip into top level of Modelsim Directory
  - If you use a different version of Modelsim than Modelsim DE version ~10.xx, you may need a different GCC compiler (good luck)
  - File can be found on the P Drive in <>
- Generate dpiheader.h by typing 'vlog -dpiheader dpiheader.h verilog_files_with_c.sv'
  - *.v, *.sv also works
- Make sure C code includes #include "dpiheader.h"
- Compile C code with 'vlog *.c'
- Simulate the verilog as normal

A (nearly) Complete Testbench



Packet Class

```
class Packet;
   randc int timestamp;
   rand int data0[8], data1[8];
   constraint c {timestamp > 0; timestamp < 1435;}
endclass
```

Interface between DUT and Test

```
interface pkt_interface( input bit clk);
                int         timestamp;
                logic           sop_in;
                logic           ena_in;
                int          data0_in;
                int          data1_in;
                logic          ena_out;
                logic          sop_out;
                int         data0_out;
                int         data1_out;
endinterface
```

Pkt Generator Class

```
class Generator;
   mailbox gen2drv;
   Packet pkt;

   function new (input mailbox gen2drv);
     this.gen2drv = gen2drv;
   endfunction

   task run();
     $display("inside generator run");
     //forever begin
     for (int x = 0; x < 10; x++) begin
        pkt = new();
        assert(pkt.randomize());
        gen2drv.put(pkt);
     end // end loop
     //end
    endtask
endclass
```

Packet Driver class

```
class Driver;
 mailbox gen2drv;
 virtual pkt_interface pkt_intf;
 function new (input mailbox gen2drv, virtual pkt_interface intf);
   this.gen2drv = gen2drv;
   pkt_intf = intf;
 endfunction
```

Packet Driver Class Cont.

```
task main;
   Packet pkt;
   $display("inside driver main");
   for (int x = 0; x < 10; x++) begin
     gen2drv.get(pkt);
     @(posedge(pkt_intf.clk));
       pkt_intf.sop_in = 1;
       pkt_intf.ena_in = 1;
       pkt_intf.data0_in = pkt.data0[0];
       pkt_intf.data1_in = pkt.data1[0];
       pkt_intf.timestamp = pkt.timestamp;
     @(posedge(pkt_intf.clk));
       pkt_intf.sop_in = 0;
       pkt_intf.ena_in = 1;
       pkt_intf.data0_in = pkt.data0[1];
       pkt_intf.data1_in = pkt.data1[1];
       pkt_intf.timestamp = pkt.timestamp;
.
.
.
       endtask
endclass
```

TestBench 'program'

```
program basic_randomization_test(pkt_interface intf);
   Generator gen;
   Driver drv;
   mailbox gen2drv;

   initial begin
      gen2drv = new();
      gen = new(gen2drv);
      drv = new(gen2drv, intf);
      //fork
      gen.run();
      drv.main();
      //join_none
   end
endprogram
```

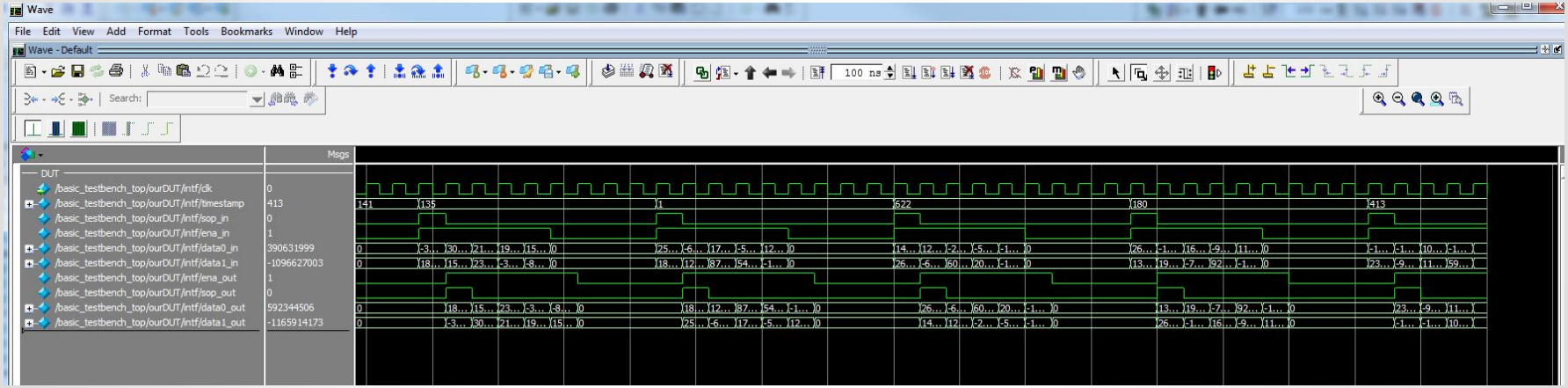
Our DUT

```
//advanced chip flips data
module DUT(pkt_interface intf);
  always@(posedge intf.clk) begin
      intf.sop_out   <= intf.sop_in;
      intf.ena_out   <= intf.ena_in;
      intf.data0_out <= intf.data1_in;
      intf.data1_out <= intf.data0_in;
  end
endmodule
```

Top Level Module

```
module basic_testbench_top();
  logic clk;
  initial begin
     clk = 0;
  end

  always #1000 clk =~clk;

  pkt_interface ourInterface(clk);
  DUT ourDUT(.intf    (ourInterface));
  basic_randomization_test ourTest(.intf (ourInterface));
endmodule
```

QuestaSim Results



A smarter person might...

- Create a generic generator / data driver / transaction and extend it for each unique test
- Run multiple drivers and DUTs in parallel using forks / join_none
- Delve into better coding style and verification methodologies, such as following the OVM / UVM Styles (Open / Universal Verification Methodology)
  - See UVM cookbook on next slide

References

- SystemVerilog for Verification – Chris Spear ISBN 978-0-387-76529-7
- FPGA Simulation A Complete Step-by-Step Guide – Ray Salemi
- http://electrosofts.com/systemverilog/fork.html
- https://verificationacademy.com/uvm-ovm
- http://www.asic-world.com/systemverilog/interface8.html#Virtual_Interface