**Midterm Solutions – 600pts total**

1. Compare and contrast verifying at the block level and system level in terms of:
2. Control – 10pts

*Verifying at the block level provides maximum control and verifying at the system level provides minimal control. It is much harder to cause specific events in a system level testbench than a block level testbench due to the lack of access to the I/O of the block.*

1. Ease of debug – 10pts

Debugging a block level testbench is easier because there are less signals in the testbench to contend with and less levels of hierarchy to traverse. Also, if a bug is flagged in a block level testbench the bug is in the block. If a bug is flagged in a system level testbench it is hard to determine which block the bug is in.

1. Simulation performance – 10pts

A block level testbench performs faster because there are less signals to simulate.

1. Testbench complexity – 10pts

The block level testbench can be as complex as a system level testbench due to emulation of all the sources of the inputs and sinks of the outputs. Generally, though, a system level testbench is more complex than a block level testbench.

1. Create a 4-bit 2-state variable called addr. Generate a random number between 0 and 15 inclusive and assign it to addr. 10pts

*bit [3:0] addr;*

*assign addr = $random;*

1. Create a 4-bit 4-state variable called addr. Generate a random number between 0 and 5 inclusive and assign it to addr. 10pts

*logic [3:0] addr;*

*assign addr = $unsigned($random % 6);*

1. What aspects of a design should be randomized? Why? 20pts
2. Device configuration. Think about all the ways you can configure a processor, cell phone chip, etc
3. Environment configuration. Your testbench mimics the environment your device operates in. the processor is connected to memory, video chips, etc. Need to test all the ways this environment can be configured.
4. Protocol exceptions. For example, what happens if an I2C command fails to complete or an AHB/PCI/SCSI bus aborts? Does the device lock up?
5. Errors and violations. It’s easy to design a device that works when there are no errors. Need to test error conditions like invalid operations. There is always the debate of how much hardware to implement to protect the user from himself. In a low power chip or where the firmware is supplied by the device supplier not much. For a device that the general public can program, need lots.
6. Delays. Send in stimulus at different rates. For example, back to back reads, back to back writes. read immediately followed by write, etc.
7. Test order. For regression testing where you run many tests randomize the order they are run to eliminate any test order dependencies.
8. Seed for the random test. Use a different seed so that the test changes every time it’s run.
9. What is the initial value of a 4-state logic type? 10pts

X

1. What is the initial value of a 2-state logic type? 10pts

0

1. Given the following declarations:

byte my\_byte;

integer my\_integer = 32’b0000\_1111\_xxxx\_zzzz;

int my\_int = my\_integer;

1. What is the range of my\_byte? Why? 10pts

*Variable my\_byte is a 8-bit signed value so it can range from to = -128 to 127*

1. What is the final value of my\_int? Why? 10pts

*my\_int is 32’h0000\_0F00 because the X’s and Z’s in 4-state my\_integer resolve to 0 when assigned to 2-state my\_int.*

1. Declare an array of size 7x128 (7-bit word, 128 words). 10pts

*bit [6:0] my\_array [128];*

1. For the array just declared what happens if a write to location 129 is attempted? 10pts

*The write is ignored.*

1. For the array just declared what happens if a read to location 129 is attempted? What is returned? 10pts

*For a 2-state array a 0 is returned. For a 4-state array a X is returned?*

1. Answer the following array questions: 10pts each
2. Declare a 2-dimensional array of size 8x3 (8-bit words by 3 words) of type bit called my\_mem. Initialize all locations to 8’hA5

*bit [7:0] my\_mem [3] = '{default:8'hA5};*

1. Declare a 2-dimensional array of size 4x4 (4-bit words by 4 words) of type logic called my\_logicmem. Initialize my\_logicmem[0]=0, my\_logicmem[1]=1, my\_logicmem[2]=2, my\_logicmem[3]=3.

*logic [3:0] my\_logicmem [4] = '{0,1,2,3}; or logic [3:0] my\_logicmem [3:0] = '{3,2,1,0};*

1. Declare a 4-bit variable of type logic called my\_logic. Initialize my\_logic to 4’hF.

*logic [3:0] my\_logic = 4’hF;*

1. What is the value of my\_mem after the following assignment: my\_mem[2] = my\_logicmem[4]; Why?

*my\_mem = {8’hA5, 8’hA5, 0} because reading from an out of bounds 4-valued type returns an X but it is resolved to a 0 when assigned to 2-value logic.*

1. What is the value of my\_logic after the following assignment: my\_logic = my\_logicmem[4]; Why?

*my\_logic = x reading from an out of bounds 4-valued type returns an X*

1. What is the value of my\_logicmem after the following assignment: my\_logicmem[3] = my\_mem[3]; Why?

*my\_logicmem= {0,1,2,0} because reading from an out of bounds 2-value type returns a 0*

1. What is the value of my\_mem after the following assignment: my\_mem[3] = my\_logic; Why?

*my\_mem does not change since the write to my\_mem[3] is out of range.*

1. Declare an associative array to hold the reset value of configuration registers. The index of the associative array is the name of the configuration register. initialize the associative array according to the following table: 10pts

|  |  |  |
| --- | --- | --- |
| **Register** | **Width** | **Reset Value** |
| adc0\_reg | [15:0] | 16'hFFFF |
| analog\_test | [15:0] | 16'hABCD |
| digital\_test | [15:0] | 16'h0 |
| digital\_config | [15:0] | 16'h1 |

*bit [15:0] reset\_array[string];*

*initial begin*

*reset\_array[“adc0\_reg“ ] = 16'hFFFF;*

*reset\_array[“analog\_test“] = 16'hABCD;  
reset\_array[“digital\_test“] = 16'h0;*

*reset\_array[“digital\_config“] = 16'h1;*

*end*

* 1. Print out the reset value of the analog test register. 10pts

*$display(“Reset value of analog test register = %h”, reset\_array[“analog\_test “]);*

1. Given the following array declaration

byte byte\_array[6]=`{0, 127, -128, 1, -1, 0};

what will be displayed by:

1. $display("%p", byte\_array.min); 10pts

`{-128}

1. $display("%p", byte\_array.max); 10pts

`{127}

1. $display("%p", byte\_array.unique); 10pts

`{0, 127, -128, 1, -1}

1. $display("%p”, byte\_array.find with (item > 0)); 10pts

`{127, 1}

1. $display("%p", byte\_array.find\_index with (item <0)); 10pts

`{2, 4}

1. byte\_array.sort; $display("%p”, byte\_array); 10pts

=`{-128, -1, 0, 0, 1, 127};

1. Suppose you are modeling a 4GB memory array that you expect to write only occasionally. What would be an appropriate storage type? Why? 10pts

*An associative array because only those locations actually written consume memory.*

1. What signals in your testbench should be of type 2-state to minimize memory usage? Why? 10pts

*Inputs to the DUT and signals that are never expected to be X or Z because it’s not necessary to detect if these signals are X or Z.*

1. What signals in your testbench should be of type 4-state? Why? 10pts

*Outputs to the DUT and signals that can be X or Z need to be 4-state because it is necessary to detect if these signals are X or Z.*

1. Should an array of 16-bit values be modeled as an unpacked or packed array to minimize memory usage? Why? 10pts

*A packed array because it is more memory efficient. A packed array will store two 16-bit words per 32-bit storage location. An unpacked array will store one 16-bit word per 32-bit storage location which wastes ½ of the memory.*

1. Answer the following questions regarding strings.
2. Create a variable of type string and assign to it “UCCS Engineering” 10pts

*string my\_string = “UCCS Engineering”;*

1. Append to the end of the string “ 2009” 10pts

*my\_string = {my\_string, “ 2009”};*

1. Display the value of the new variable 10pts

*$display(“%s”, my\_string);*

1. Given the following packet structure:



1. Define a user defined 7-bit type 10pts
2. Encapsulate the fields of the following packet in a structure using your new type 10pts
3. Instantiate the structure and assign the header to 7’h5A; 10pts

*`default\_nettype none*

*module test;*

*typedef bit [6:0] bit7;*

*typedef struct {*

*bit7 header;*

*bit7 cmd;*

*bit7 data;*

*bit7 crc;*

*} packet;*

*packet my\_packet;*

*initial begin*

*my\_packet.header = 7'h5A;*

*end*

*endmodule // test*

1. Why would a function be defined in an interface? 10pts

*A function would be defined in an interface for protocol checkers and for commonly used functions.*

1. If the function performed non-synthesizable operations like $display how can a DUT that uses this interface be synthesizable? 10pts

*Only those functions called by the DUT are synthesized. The DUT cannot call any function that contains non-synthesizable constructs.*

1. Given the following array:

shortint shortint\_array[1023:0];

1. Create a function that prints out the last element of the array. Pass the array by reference. 10pts

*function automatic void print\_func(ref shortint localint\_array[1023:0]);*

*$display(“%h”, localint\_array[1023]);*

*end*

1. Call the function and pass in the values by name not position 10pts

*print\_func(.localint\_array(shortint\_array));*

1. For the following interface:

interface reg\_if(input bit clk);

bit write, bit [15:0] data\_in, bit [7:0] address, logic [15:0] data\_out;

modport slave (input clk, write, data\_in, address, output data\_out);

endinterface

1. Develop a clocking block called cb that: 10pts
2. is sensitive to the positive edge of clock
3. all I/O are synchronous to the clock and direction is relative to the testbench.

*clocking cb @(posedge clk);*

*output write, data\_in, address;*

*input data\_out;*

*endclocking;*

1. Expand the interface to create a modport for the testbench that uses the clocking block. 10pts

*modport master(clocking cb, input clk);*

1. Complete the following timing diagram assuming the interface is instantiated as reg\_bus. 20pts

****

1. Define a class in a package that has the following characteristics: 50pts
   1. Member variables of 8-bit logic type called *header,* of 16-bit logic type called *payload*, and of 8-bit logic type called *parity,* and a static integer called *error.*
   2. A function that calculates the parity by the following diagram. Each parity bit is determined by the calculating the XOR of the rows above it. For example, parity[0]=payload[0]^payload[8]^header[0]



* 1. A custom constructor that by default initializes the *header* to 8’hA5 and the *payload* to all F’s. Values for header and payload can also be initialized through arguments passed into the constructor. The custom constructor calls the function in b. to compute the correct parity.
  2. A function that prints all the member variables of an object.

package my\_package;

class packet;

logic [7:0] header;

logic [15:0] payload;

logic [7:0] parity;

static int error;

function void calc\_parity();

parity = header ^ payload[15:8] ^ payload[7:0];

endfunction // calc\_parity

function void print();

$display("header = 0x%0h, payload = 0x%0h, parity = 0x%0h, error = 0d%0d", header, payload, parity, error);

endfunction // print

function new(logic [7:0] header\_init = 8'hA5, logic [15:0] payload\_init = 16'hFFFF);

header = header\_init;

payload = payload\_init;

calc\_parity();

endfunction // new

endclass

endpackage // my\_package

1. Design a program using the package defined in question 22) that completes the following tasks: 40pts
2. Initialize member variable *error* through the class prior to construction.
3. Create 2 objects. The first object will be created using the default values and the second object will be created with header = 8’h3C and payload=16’h1234 passed to the constructor by name not position.
4. Call the print function from 22d. for each object.
5. Explicitly deallocate the 2nd object.

program test();

import my\_package::\*;

packet p1, p2;

initial begin

packet::error = 0;

p1 = new();

p2 = new(.header\_init(8'h3C), .payload\_init(16'h1234));

p1.print();

p2.print();

p2 = null;

$finish;

end

endprogram // test

1. When is a SystemVerilog construct expected to be synthesizable? Why? 10pts

*When the construct is not dynamic because everything is known about the construct at compile time, not run time as is the case for dynamic constructs.*

1. When is verification of a design complete? Why? 20pts

*When statement coverage is 100% since all statements have been tested.*

*When branch coverage is 100% since all branches have been tested.*

*When conditional coverage is around 90% since most of the ways that a branch can be true and false have been tested.*

*When expression coverage is around 90% since most of the ways that an expression can be true and false have been tested.*

*When state machine (FSM) state coverage is 100% because all states have been entered.*

*When state machine (FSM) transition coverage is 100% except for all paths back to reset because all transitions have been tested.*

*When the schedule does not permit any more time for verification.*

*When no more bugs are found.*

*When all tests in the test plan have been verified.*