CSE 141L: Introduction to Computer Architecture Lab
SystemVerilog Verification

Pat Pannuto, UC San Diego
ppannuto@ucsd.edu

Milestone 2 is due in 9 days

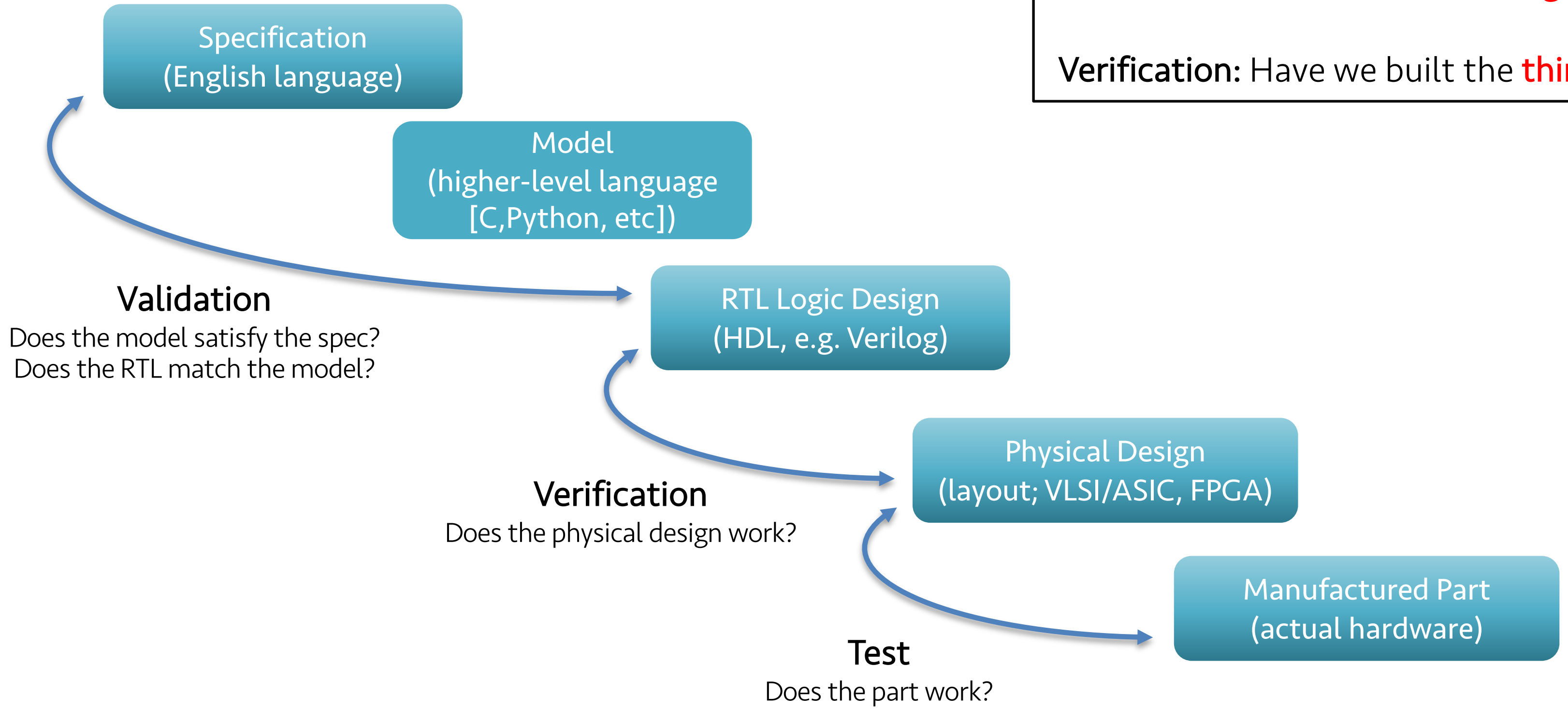
- What to submit?
  - SOMETHING
- M1 feedback
  - Should be released by Wednesday
  - Pay attention to things that should be revised for M2
- M2 is about proving individual components work
  - How would you prove to your manager your component works?

Today's Objectives: Validation & Verification in Hardware Design

- Real-world hardware design process
  - And where we are cutting corners to simplify for class

- Mapping verification in theory to verification in practice

The hardware design process



Validation: Have we built the right thing?

Verification: Have we built the thing right?

V&V is standard practice across engineering disciplines

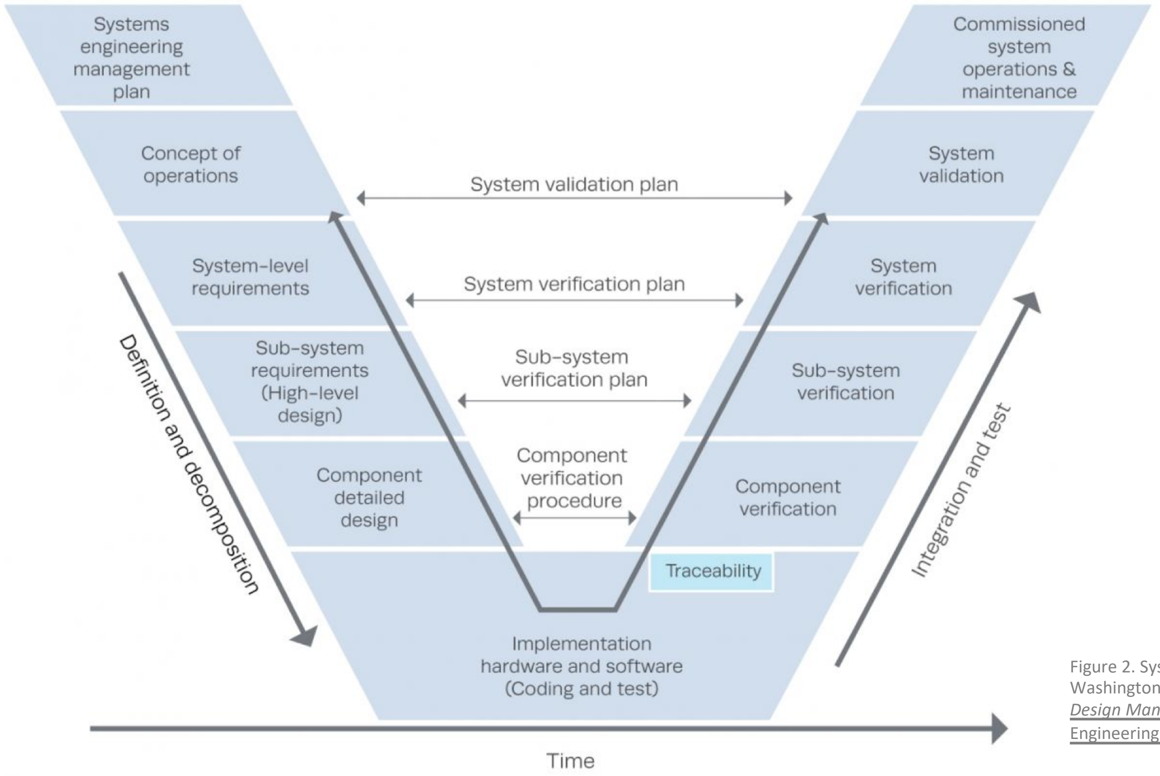

Systems engineering management plan
Concept of operations
System-level requirements
Sub-system requirements (High-level design)
Component detailed design
Implementation hardware and software (Coding and test)
Component verification
Sub-system verification
System verification
System validation
Commissioned system operations & maintenance
System validation plan
System verification plan
Sub-system verification plan
Component verification procedure
Traceability
Definition and decomposition
Integration and test
Time

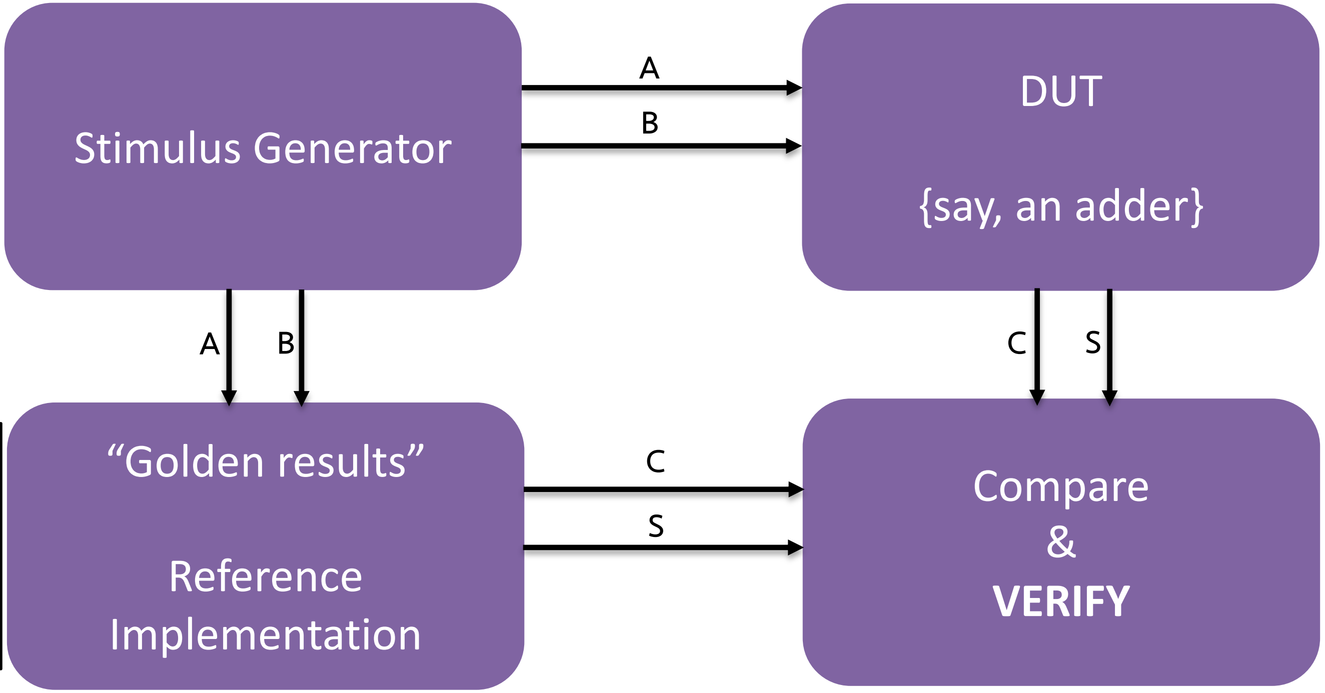
Figure 2. Systems Engineering "V," Washington State DOT, July 2010, *WSDOT Design Manual*, Chapter 1050.03, Systems Engineering: Systems Engineering "V."

So what is verification to us?

- Complete validation of all functionality of Device Under Test (DUT)
  - Q: Did the ALU testbench example last week do this?
    - How do you know that it did / didn’t?

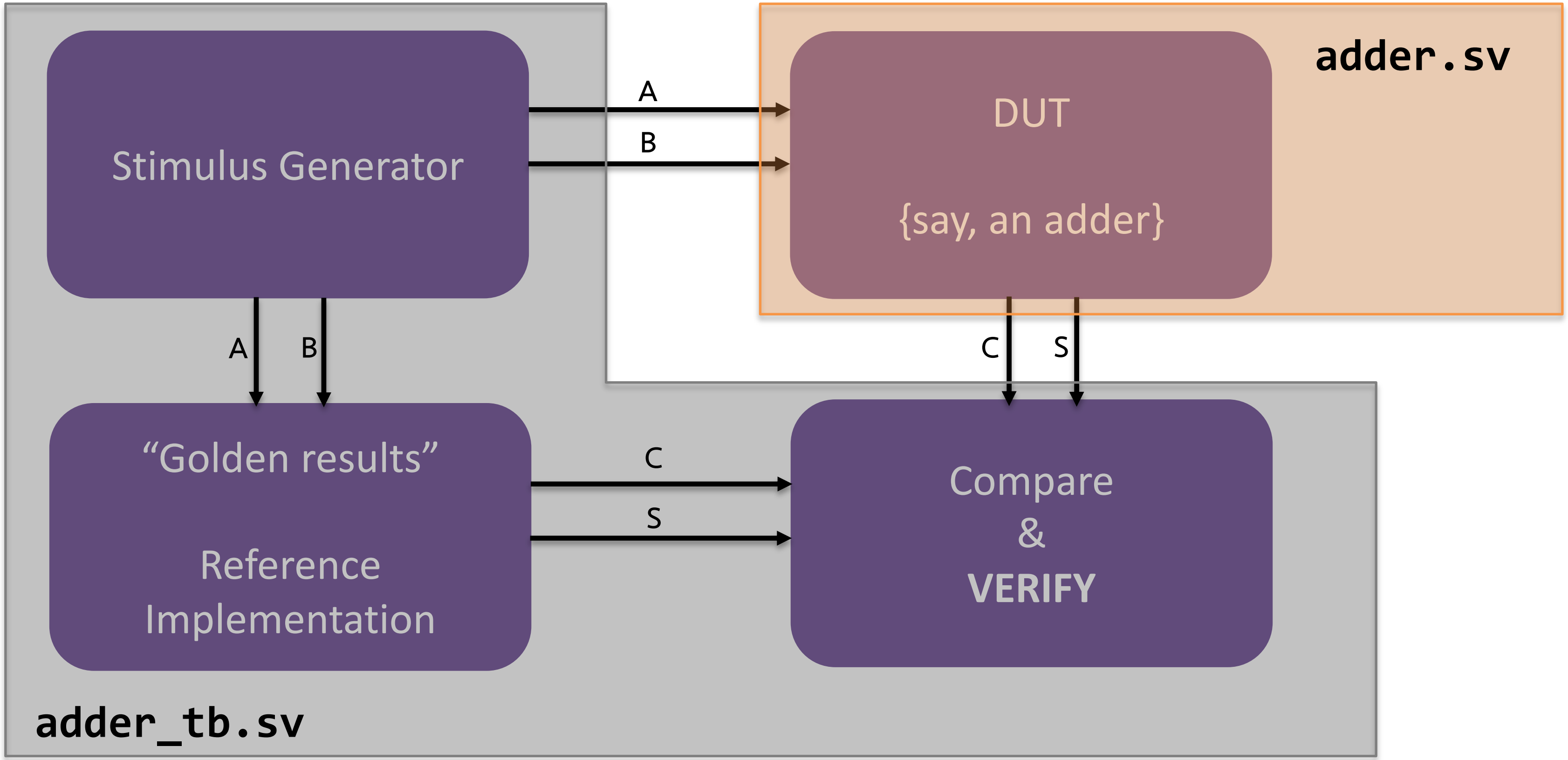
- Mechanistically: Stimulating DUT with all possible inputs

Conceptual view of Verification



A	B	C	S
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

Pragramatic view of Verification [for this class!]



Can we find these pieces in last week's `alu_tb.sv`?

```
`timescale 1ns/ 1ps

module ALU_tb;


// Signals to interface with the ALU module
logic [ 7:0] INPUTA;   // data inputs
logic [ 7:0] INPUTB;
logic [ 2:0] op;       // ALU opcode
bit SC_IN = 'b0;
wire[ 7:0] OUT;
wire Zero;

// Define a helper wire for comparison
logic [ 7:0] expected;

// Loop variables
integer i, j;

// Instatiate and connect Unit Under Test
ALU uut(
  .InputA(INPUTA),
  .InputB(INPUTB),
  .SC_in(SC_IN),
  .OP(op),
  .Out(OUT),
  .Zero(Zero)
);
```

```
// The actual testbench logic
initial begin
 $display("STarting!");

 INPUTA = 1;
 INPUTB = 1;
 op= 'b000; // ADD
 test_alu_func; // void function call
 #5;

 INPUTA = 4;
 INPUTB = 1;
 op= 'b100; // AND
 test_alu_func; // void function call
 #5;

op= 'b011; // XOR
 for (i=0; i<256; i++) begin
   for (j=0; j<256;j++) begin
     INPUTA = i;
     INPUTB = j;
     test_alu_func;
     #5;
 end // j end
 end // i end


 $display("End: all test cases passed.");

 end // initial begin's end
```

```
task test_alu_func; begin
  case (op)
    0: expected = INPUTA + INPUTB;  // ADD
    1: expected = {INPUTA[6:0], SC_IN};  // LSH
    2: expected = {1'b0, INPUTA[7:1]};  // RSH
    3: expected = INPUTA ^ INPUTB;  // XOR
    4: expected = INPUTA & INPUTB;     //AND
    5: expected = INPUTA - INPUTB;   // SUB
  endcase
  #1;
  if(expected == OUT) begin
    //$display("%t YAY!! inputs = %h %h, opcode
= %b, Zero %b",$time, INPUTA,INPUTB,op, Zero);
  end else begin
    $display("%t FAIL! inputs = %h %h, opcode =
%b, zero %b",$time, INPUTA,INPUTB,op, Zero);
    $stop;
  end
end
endtask


endmodule
```

Straight talk: Some of the ‘little tests’ feel silly

- But I promise it feels worse when a bug was a typo in an ‘easy’ module

- Take advantage of groups
  - One of you implement your ALU, according to your specification
  - Someone else implement the ALU testbench, according to your specification
  - … does it actually match?

Pragmatic considerations for verification

- A few slides back…
  - "Mechanistically: Stimulating DUT with all possible inputs"

- What defines "all possible inputs" for
  - A half-adder?
  - Our example ALU?
  - Your processor?

So what can we do to actually test well?

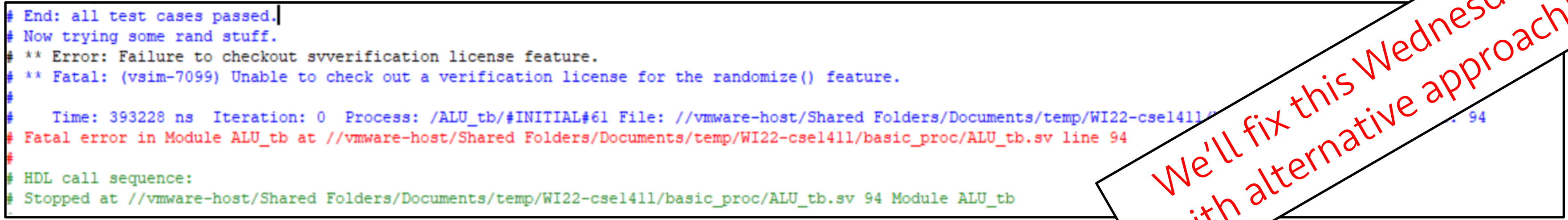
- Exhaustive coverage?
- Principled, corner-case test design?
- Randomized coverage?
- All of the above?

Questions on anything so far? Then hopping over to look at some examples of `rand`

- SystemVerilog random testing?

```
rand  bit [7:0] inputA // rand picks random values independently
rand  bit [7:0] inputB // and can repeat choices throughout the run
randc bit [2:0] opc    // “cycle” random won’t repeat until all seen


constraint legal_ops { opc < 6; /* can add more here */ }
```



```
# End: all test cases passed.
# Now trying some rand stuff.
# ** Error: Failure to checkout svverification license feature.
# ** Fatal: (vsim-7099) Unable to check out a verification license for the randomize() feature.
#
#    Time: 393228 ns  Iteration: 0  Process: /ALU_tb/#INITIAL#61 File: //vmware-host/Shared Folders/Documents/temp/WI22-cse1411/                  . 94
# Fatal error in Module ALU_tb at //vmware-host/Shared Folders/Documents/temp/WI22-cse1411/basic_proc/ALU_tb.sv line 94
#
# HDL call sequence:
# Stopped at //vmware-host/Shared Folders/Documents/temp/WI22-cse1411/basic_proc/ALU_tb.sv 94 Module ALU_tb
```

We’ll fix this Wednesday with alternative approach!

Open Q&A on testbench design, more live examples