Section 7

Common Issues When Connecting Modules
Administrivia

Lab 7: Report due next Wednesday (5/15) @ 2:30 pm, demo by last OH on Friday (5/17), but expected during your assigned slot.
  ○ Tunable cyber player opponent (counter, LFSR, adder).

Lab 8: Final project is coming up!
  ○ Choose from 8 possible projects or suggest your own.
  ○ Range of difficulties (and point values).
  ○ Extra credit opportunities for early finish and bonus features!
General Debugging Tips
SystemVerilog Debugging

- Many things are similar to software debugging!
  - Have compiler messages and program output to work with.
  - Identify the behavior(s) that differ from what you expect and work backwards from there.
  - Need to understand data representation and manipulation.
SystemVerilog Debugging

- Many things are similar to software debugging!
  - Have compiler messages and program output to work with.
  - Identify the behavior(s) that differ from what you expect and work backwards from there.
  - Need to understand data representation and manipulation.

- However, many things are different in hardware!
  - Parallel (instead of sequential) execution makes interpreting programs more difficult.
    - We often rely on simulation waveforms instead of terminal output.
    - Timing is always a factor/consideration (e.g., timing delays, sensitivity lists, blocking vs. non-blocking assignments).
  - Can make mistakes between simulation and bit file (e.g., clock_divider).
**General Debugging Tips**

- The best debugging advice is to not have to debug at all!
  - **Focus on the design** (i.e., block diagrams, state diagrams) to avoid impractical designs and major logical flaws.
General Debugging Tips

- The best debugging advice is to not have to debug at all!
  - **Focus on the design** *(i.e., block diagrams, state diagrams)* to avoid impractical designs and major logical flaws.

- Staring at code until you think you spot a bug is generally not an effective way to debug.
  - Of course it looks logically correct to you – you wrote it!
  - SystemVerilog is a really tricky language – we've only scratched the surface and the code often obfuscates the synthesized hardware.

- Instead, lean on the available tools, which are intended to help you.
  - We’ll cover some tips in the following slides.
Quartus Debugging Tips

- The built-in syntax highlighting can help find typos in keywords (black vs. blue text) and what is currently commented out (green vs. not green).
- Double-clicking a word will highlight all instances of that word in your code, making for easier visual scanning and spotting of typos.
- Pay attention to compiler output messages, which usually point out the problematic line of code!
  - Can filter by (1) Errors , (2) Critical Warnings , and (3) Warnings .
  - Some common messages and their suggested fixes can be found in our SystemVerilog Warnings & Errors Doc.
  - Double-click a warning or error message to have it automatically take you to the appropriate point in the code.
ModelSim Debugging Tips

- ModelSim has its own compiler so pay attention to output messages here as well.
- Add internal signals from any instantiated module to your simulation!
  - For a buggy signal, add *all* signals involved in the computation of that signal.
- Make sure you’re using the appropriate radix (*e.g.*, binary vs. decimal vs. unsigned) for that signal’s particular use case.
- **Red lines** have multiple causes; it’s important to identify which is the case so you can narrow down your fix.
  - Undefined signal (*e.g.*, no initialization), net contention (*e.g.*, multiple drivers), explicit don’t care in code (*e.g.*, `default: leds = 7'bX;`).
Debugging Time!
Exercise 1 – Interpreting Messages

Given the following modules and error messages, identify & fix the bug.

2 module ex1 (output logic dout, input logic [2:0] upc);
3   assign dout = upc[1] & upc[0] ^ upc[2];
4 endmodule  // ex1

2 module DE1_SoC (input logic [9:0] SW, output logic [9:0] LEDR);
3   ex1 e1 (.dout(LEDR[0]), .upc(SW[1:0]));
4 endmodule  // DE1_SoC

122411 hierarchies have connectivity warnings - see the Connectivity Checks report folder
144001 Generated suppressed messages file C:/369/sec7/output_files/DE1_SoC.map.smsg
16010 Generating hard_block partition "hard_block:auto_generated_inst"
21057 Implemented 20 device resources after synthesis - the final resource count might be different
Quartus Prime Analysis & Synthesis was successful. 0 errors, 1 warning

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Severity</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>upc</td>
<td>Input</td>
<td>Warning</td>
<td>Input port expression (2 bits) is smaller than the input port (3 bits) it drives. Extra input bit(s) &quot;upc[2..2]&quot; will be connected to GND.</td>
</tr>
</tbody>
</table>
Exercise 1 (Solution)

- Given the following modules and error messages, identify & fix the bug.

```vhdl
2 module ex1 (output logic dout, input logic [2:0] upc);
3 assign dout = upc[1] & upc[0] ^ upc[2]);
4 endmodule // ex1

2 module DE1_SoC (input logic [9:0] SW, output logic [9:0] LEDR);
3 ex1 e1 (.dout(LEDR[0]), .upc(SW[2:0])); ← changed to [2:0]
4 endmodule // DE1_SoC
```

⚠️ 122411 hierarchies have connectivity warnings - see the Connectivity Checks report folder
○ 144001 Generated suppressed messages file C:/369/sec7/output_files/DE1_SoC.map.smsg
○ 16010 Generating hard_block partition "hard_block:auto_generated_inst"
○ 21057 Implemented 20 device resources after synthesis - the final resource count might be different
  ○ Quartus Prime Analysis & Synthesis was successful. 0 errors, 1 warning

Port Connectivity Checks: "ex1:e1"

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Severity</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>upc</td>
<td>Input</td>
<td>Warning</td>
<td>Input port expression (2 bits) is smaller than the input port (3 bits) it drives. Extra input bit(s) &quot;upc[2:2]&quot; will be connected to GND.</td>
</tr>
</tbody>
</table>
Exercise 2 – Port Connection Analysis

- Given the modules to the right, analyze the ports instantiations in `ex2` independently.
  - Is there an issue?
  - If so, what is it? Do you think it will produce a warning or an error?

```plaintext
2 module ports (input logic a, 
3     input logic [1:0] b, 
4     output logic c);
5 endmodule  // ports

2 module ex2 (input logic a, b, d, 
3     input logic [1:0] e, 
4     output logic c);
5
6   ports option1 (.a, .c);
7   ports option2 (.a, .b(e), .c);
8   ports option3 (.*);
9   ports option4 (.a, .b(e), .c, .d);
10  ports option5 (.a, .b(e), .c(d));
11 12 endmodule  // ex2
```
Exercise 2-1 (Solution)

- option1:

```verilog
module ports (input logic a,
              input logic [1:0] b,
              output logic c);
endmodule // ports
```

```verilog
module ex2 (input  logic a, b, d,
            input  logic [1:0] e,
            output logic c);
ports option1 (.a, .c);
ports option2 (.a, .b(e), .c);
ports option3 (.*);
ports option4 (.a, .b(e), .c, .d);
ports option5 (.a, .b(e), .c(d));
endmodule // ex2
```
Exercise 2-1 (Solution)

- **option1**:  
  - Only 2 ports connected to 3-port module.  
  - Compiler warning:

```
2 module ports (input logic a,
3      input logic [1:0] b,
4      output logic c);  
5 endmodule // ports
```

```
2 module ex2 (input logic a, b, d,
3      input logic [1:0] e,
4      output logic c);
5
6   ports option1 (.a, .c);
7   ports option2 (.a, .b(e), .c);
8   ports option3 (.*);
9   ports option4 (.a, .b(e), .c, .d);
10  ports option5 (.a, .b(e), .c(d));
```

⚠ Warning (12241): 1 hierarchies have connectivity warnings - see the Connectivity Checks report folder

ⓘ Info: Quartus Prime Analysis & Synthesis was successful. 0 errors, 2 warnings
Exercise 2-2 (Solution)

- option2:

```module ports (input logic a,
    input logic [1:0] b,
    output logic c);
endmodule // ports
```
Exercício 2-2 (Solution)

- **option2:**
  - No issues!
  - Though this is confusing port naming and is not recommended.

```verilog
2 module ports (input logic a,
3     input logic [1:0] b,
4     output logic c);
5 endmodule // ports

2 module ex2 (input logic a, b, d,
3     input logic [1:0] e,
4     output logic c);
5
6 ports option1 (.a, .c);
7 ports option2 (.a, .b(e), .c);
8 ports option3 (.*);
9 ports option4 (.a, .b(e), .c, .d);
10 ports option5 (.a, .b(e), .c(d));
11
12 endmodule // ex2
```
Exercise 2-3 (Solution)

- option3:

```markdown
module ports (input logic a,
             input logic [1:0] b,
             output logic c);
endmodule // ports

module ex2 (input logic a, b, d,
             input logic [1:0] e,
             output logic c);

ports option1 (.a, .c);
ports option2 (.a, .b(e), .c);
ports option3 (.*);
ports option4 (.a, .b(e), .c, .d);
ports option5 (.a, .b(e), .c(d));
endmodule // ex2
```
Exercise 2-3 (Solution)

- **option3:**
  - Implicit port connections fail for `b` because the types don't match.
  - Compiler errors:
    - Error (10897): SystemVerilog error at ex2.sv(8): can't implicitly connect port "b" on instance "option3" of module "ports" - matching object in present scope does not have an equivalent data type
    - Error (10784): HDL error at ports.sv(3): see declaration for object "b"
    - Error (10784): HDL error at ex2.sv(2): see declaration for object "b"
    - Error (12153): Can't elaborate top-level user hierarchy
    - Error: Quartus Prime Analysis & Synthesis was unsuccessful. 4 errors, 0 warnings
Exercise 2-4 (Solution)

- **option4:**

```verilog
2 module ports (input logic a,
3     input logic [1:0] b,
4     output logic c);
5 endmodule // ports

2 module ex2 (input logic a, b, d,
3     input logic [1:0] e,
4     output logic c);
5
6   ports option1 (.a, .c);
7   ports option2 (.a, .b(e), .c);
8   ports option3 (.*);
9   ports option4 (.a, .b(e), .c, .d);
10  ports option5 (.a, .b(e), .c(d));
11
12 endmodule // ex2
```
Exercise 2-4 (Solution)

- **option4:**
  - There is no port for ports (only 3 ports, too).
  - Compiler errors:

```verilog
2 module ports (input logic a,
3     input logic [1:0] b,
4     output logic c);
5 endmodule // ports

2 module ex2 (input logic a, b, d,
3     input logic [1:0] e,
4     output logic c);
5   ports option1 (.a, .c);
6   ports option2 (.a, .b(e), .c);
7   ports option3 (.*);
8   ports option4 (.a, .b(e), .c, .d);
9   ports option5 (.a, .b(e), .c(d));
10 endmodule // ex2
```

- Error (10284): Verilog HDL Module Instantiation error at ex2.sv(9): port "d" is not declared by module "ports"
- Error: Quartus Prime Analysis & Synthesis was unsuccessful. 1 error, 0 warnings
Exercise 2-5 (Solution)

- option5:

```verilog
2 module ports (input logic a,  
3          input logic [1:0] b,  
4          output logic c);  
5 endmodule  // ports

2 module ex2 (input logic a, b, d,  
3          input logic [1:0] e,  
4          output logic c);  
5 ports option1 (.a, .c);  
6 ports option2 (.a, .b(e), .c);  
7 ports option3 (.*);  
8 ports option4 (.a, .b(e), .c, .d);  
9 ports option5 (.a, .b(e), .c(d));  
10 endmodule  // ex2
```
Exercise 2-5 (Solution)

- option5:
  - Multiple drivers: c is an output port for ports but connected to an input port.
  - Compiler errors:
    - Error (10031): Net "d" at ex2.sv(10) is already driven by input port "d", and cannot be driven by another signal
    - Error (10032): "d" was declared at ex2.sv(2)
    - Error (12153): Can't elaborate top-level user hierarchy
    - Error: Quartus Prime Analysis & Synthesis was unsuccessful. 3 errors, 0 warnings
Contention on a Net

- A situation where two or more drivers (sources of signals) are trying to drive a net to different values at the same time is known as *contention*.
  - This will show up as an X value in simulation, and could result in damage to the drivers in a physical device!
Contention on a Net

- A situation where two or more drivers (sources of signals) are trying to drive a net to different values at the same time is known as **contention**.
  - This will show up as an X value in simulation, and could result in damage to the drivers in a physical device!

- Example:

```verilog
module DE1_SoC (input logic [9:0] SW, output logic [6:0] HEX0, HEX1);

  seg7 display(.leds(HEX0), .bcd({SW[3], SW[2], SW[1], SW[0]}));

  // Default values, turns off the HEX displays
  assign HEX0 = 7'b1111111;
  assign HEX1 = 7'b1111111;

endmodule  // DE1_SoC
```
module DE1_SoC (input logic [9:0] SW, output logic [6:0] HEX0, HEX1);

seg7 display(.leds(HEX0), .bcd({SW[3], SW[2], SW[1], SW[0]}));

// Default values, turns off the HEX displays
assign HEX0 = 7'b1111111;
assign HEX1 = 7'b1111111;

endmodule // DE1_SoC

multiply driven!

Error (12014): Net "HEX0[6]", which fans out to "HEX0[6]", cannot be assigned more than one value
Error (12014): Net "HEX0[5]", which fans out to "HEX0[5]", cannot be assigned more than one value
Error (12014): Net "HEX0[4]", which fans out to "HEX0[4]", cannot be assigned more than one value
Error (12014): Net "HEX0[3]", which fans out to "HEX0[3]", cannot be assigned more than one value
Error (12014): Net "HEX0[2]", which fans out to "HEX0[2]", cannot be assigned more than one value
Error (12014): Net "HEX0[1]", which fans out to "HEX0[1]", cannot be assigned more than one value
Error (12014): Net "HEX0[0]", which fans out to "HEX0[0]", cannot be assigned more than one value
Error (12015): Net is fed by "VCC"
Error (12015): Net is fed by "seg7:display|leds[0]"

Error: Quartus Prime Analysis & Synthesis was unsuccessful. 21 errors, 0 warnings
Example 1

Identify the bug in the following code:

```verilog
module parent (
    input logic Clk, in,
    output logic out
);
    child c (.Clk, .in, .out);
endmodule

module child (
    input logic clk, in,
    output logic out
);
    logic ps, ns;
    always_comb
        case (ps)
            1'b0: ns = (in ? 1'b1 : 1'b0);
            1'b1: ns = (in ? 1'b0 : 1'b1);
            default: ns = ps;
        endcase
    assign out = ns;
    always_ff @(posedge clk)
        ps <= ns;
endmodule
```

*Error: Port “Clk” does not exist in macrofunction “f”*
Example 1

Clk ≠ clk!

Make sure port connections are sound (esp. c vs. C is hard to tell on Quartus)

```verilog
module parent (  
    input logic clk, in,  
    output logic out
);

    // alternatively, .clk(Clk)
    child c (.clk, .in, .out);

endmodule // parent

module child (  
    input logic clk, in,  
    output logic out
);

    logic ps, ns;

always_comb
    case (ps)
        1'b0: ns = (in ? 1'b1 : 1'b0);
        1'b1: ns = (in ? 1'b0 : 1'b1);
        default: ns = ps;
    endcase

assign out = ns;

always_ff @(posedge clk)
    ps <= ns;

endmodule // child
```

*Note that reset is omitted for simplicity*
Example 2

Identify the bug in the following code:

```verbatim
module DE1_SoC (  
    input logic CLOCK_50,  
    output logic [6:0] HEX0, ...  
    ...  
    output logic [9:0] SW  
);

    // instantiate an fsm that lights LEDR[0] on  
    // when KEY[0] is pressed for two frames in a row  
    fsm f (.clk(CLOCK_50), .in(KEY[0]) .out(LEDR[0]));

endmodule  // DE1_SoC
```

Buggy Simulation:
Example 2

KEYs are **Active-Low**!

Make sure to account for hardware realities.

Correct Simulation:
Timing Issues
Example 3 - Different Clocks Issue

- A situation where not all module instances are using the same clock.
- For example, say we have this register A that captures and outputs a 4-bit input signal:

```verilog
module A(
    input logic clk,
    input logic reset,
    input logic [3:0] in,
    output logic [3:0] out);

    always_ff @(posedge clk)
    if (reset)
        out <= 0;
    else
        out <= in;
endmodule
```

// moduleA
In the top-level module, we instantiate module A twice.
Example 3 - Different Clocks Issue

- Instance a1 with the clock_50
- Instance a2 with the divided clock

```verbatim
module top_module(
   input logic CLOCK_50,
   input logic reset,
   input logic [3:0] in,
   output logic [3:0] out1, out2
);

   // clock divider
   logic [31:0] clock;
   clock_divider cdiv (.clock(clk), .divided_clocks(clock));

   // a1 is using a 50Mhz clock
   // a2 is using a slower clock with 12.5 MHz
   A a1(.clk(CLOCK_50), .reset, .in, .out(out1));
   A a2(.clk(clock[1]), .reset, .in, .out(out2));
endmodule // top_module
```

```verbatim
module A(
   input logic clk,
   input logic reset,
   input logic [3:0] in,
   output logic [3:0] out);

   always_ff @(posedge clk)
      if (reset)
         out <= 0;
      else
         out <= in;
endmodule // moduleA
```
Example 3 - Different Clocks Issue

- This will cause a **desynchronization** in our system!
- Easy to miss when going between hardware ↔ sim.

```verilog
module top_module(
    input logic CLOCK_50,
    input logic reset,
    input logic [3:0] in,
    output logic [3:0] out1, out2
);

    // clock divider
    logic [31:0] clock;
    clock_divider cdiv (.clock(clk), .divided_clocks(clock));

    // a1 is using a 50Mhz clock
    // a2 is using a slower clock with 12.5 MHz
    A a1(.clk(CLOCK_50), .reset, .in, .out(out1));
    A a2(.clk(clock[1]), .reset, .in, .out(out2));
endmodule  // top_module
```

```verilog
module A(
    input logic clk,
    input logic reset,
    input logic [3:0] in,
    output logic [3:0] out);

    always_ff @(posedge clk)
        if (reset)
           out <= 0;
        else
           out <= in;
endmodule  // moduleA
```
Different Clocks Issue (Simulation)

Output of the module A that does not use divided clock (50 MHz clock)

Output of the module A that does use divided clock (12.5 MHz clock)
Different Clocks Issue (Simulation)

Let’s take a closer look at the clock in each module

The clocks in module a1 and a2 have different clock frequency. Module a2 has a slower clock than a1 does, causing desynchronization issue.
Different Clocks Issue (Simulation)

Using the same clock!

module top_module(...);
...
A a1(.clk(CLOCK_50), .reset, .in, .out(out1));
A a2(.clk(CLOCK_50), .reset, .in, .out(out2));
endmodule  // top_module
Exercise 3

- In Section 6, we worked on a design of the **psychic tester**, where the user needs to correctly guess 8 consecutive 4-bit patterns to be declared a psychic.
- Say we want to modify our design so the next signal comes directly from user_input.
Exercise 3

- In Section 6, we worked on a design of the **psychic tester**, where the user needs to correctly guess 8 consecutive 4-bit patterns to be declared a psychic.
- Say we want to modify our design so the next signal comes directly from `user_input`. 
Exercise 3

Let’s see how one might modify `psychic_tester`

```verilog
module psychic_tester (clk, rst, guess_ext, submit_ext, psychic);
  input  logic  clk, rst, submit_ext;
  input  logic [3:0] guess_ext;
  output logic  psychic;

  logic [3:0] pattern, guess;
  logic correct, next, submit;

  genPatt pat (.clk, .rst, .pattern, .next);
  userIn inp (.clk, .rst,
               .guess_ext, .submit_ext, .guess, .submit);
  checkGuess chk (.pattern, .guess, .correct);
  countRight cnt (.clk, .rst, .correct, .submit, .next, .psychic);
endmodule  // psychic_tester
```
Exercise 3

- Remove the next signal from `countRight`:

```verilog
module psychic_tester (clk, rst, guess_ext, submit_ext, psychic);
  input  logic    clk, rst, submit_ext;
  input  logic [3:0] guess_ext;
  output logic   psychic;
  logic [3:0] pattern, guess;
  logic correct, next, submit;
  genPatt   pat (.clk, .rst, .pattern, .next);
  userIn    inp (.clk, .rst,
                  .guess_ext, .submit_ext, .guess, .submit);
  checkGuess chk (.pattern, .guess, .correct);
  countRight cnt (.clk, .rst, .correct, .submit, .psychic);
endmodule // psychic_tester
```
Exercise 3

- Assign `next` to `submit_ext`

```verilog
module psychic_tester (clk, rst, guess_ext, submit_ext, psychic);
  input logic clk, rst, submit_ext;
  input logic [3:0] guess_ext;
  output logic psychic;

  logic [3:0] pattern, guess;
  logic correct, next, submit;
  assign next = submit_ext;
  genPatt pat (.clk, .rst, .pattern, .next);
  userIn inp (.clk, .rst, .guess_ext, .submit_ext, .guess, .submit);
  checkGuess chk (.pattern, .guess, .correct);
  countRight cnt (.clk, .rst, .correct, .submit, .psychic);
endmodule // psychic_tester
```
Exercise 3

- Because we didn’t modify how our input and outputs work and how the design should behave, we should be able to use the same testbench!
- Let’s take a look at ModelSim and see if things are behaving the way we expect...
Exercise 3

- Because we didn’t modify how our input and outputs work and how the design should behave, we should be able to use the same testbench!
- Let’s take a look at ModelSim and see if things are behaving the way we expect...
Exercise 3a

- We don’t get a high signal for psychic anymore...
- **Group brainstorm:** What could be causing this issue? What would you investigate?
Exercise 3a

- We don’t get a high signal for psychic anymore...
- **Group brainstorm:** What could be causing this issue? What would you investigate?
  - Check intermediate signals!
    - Counter signal
    - Pattern signal
  - Investigate internal signal of submodules that could have been affected by our modifications!
Exercise 3b

- Let’s analyze our internal signals

- Group Brainstorming: What incorrect behavior do you notice?
Exercise 3b

- Let’s analyze our internal signals

- It’s clear that our pattern is not changing like we expect anymore!
- **Group Brainstorming:** What other weird behaviors do you notice?
**Exercise 3b**

- Let’s analyze our internal signals
- It's clear that our pattern is not changing like we expect anymore!
- **Group Brainstorming:** What other weird behaviors do you notice?
- Notice how at 310 ps our pattern changes, even though the next signal should be synchronized to have a 1 clock delay!
Exercise 3b

- Let’s analyze our internal signals

- It's clear that our pattern is not changing like we expect anymore!

- **Group Brainstorming:** What other weird behaviors do you notice?

- Notice the difference between signals **next** and **submit**! One is not edge detected!
Exercise 3c

- **Group Brainstorm:** Let’s go back to the code and see if we can spot the bug! What lines could be causing our timing issue?

```verilog
module psychic_tester (clk, rst, guess_ext, submit_ext, psychic);
    input logic clk, rst, submit_ext;
    input logic [3:0] guess_ext;
    output logic psychic;

    logic [3:0] pattern, guess;
    logic correct, next, submit;
    assign next = submit_ext;
    genPatt pat (.clk, .rst, .pattern, .next);
    userIn inp (.clk, .rst,
                .guess_ext, .submit_ext, .guess, .submit);
    checkGuess chk (.pattern, .guess, .correct);
    countRight cnt (.clk, .rst, .correct, .submit, .psychic);
endmodule // psychic_tester
```
**Exercise 3c**

- **Group Brainstorm:** Remember the two weird behaviors we noticed: pattern is not synchronized and next is not being edge detected.

```verilog
module psychic_tester (clk, rst, guess_ext, submit_ext, psychic);
    input  logic   clk, rst, submit_ext;
    input  logic [3:0] guess_ext;
    output logic psychic;

    logic [3:0] pattern, guess;
    logic correct, next, submit;
    assign next = submit_ext;
    genPatt pat (.clk, .rst, .pattern, .next);
    userIn inp (.clk, .rst,
                .guess_ext, .submit_ext, .guess, .submit);
    checkGuess chk (.pattern, .guess, .correct);
    countRight cnt (.clk, .rst, .correct, .submit, .psychic);
endmodule // psychic_tester
```
Exercise 3c

- We incorrectly assigned `next` to `submit_ext` instead of `submit`!

```verilog
module psychic_tester (clk, rst, guess_ext, submit_ext, psychic);
    input  logic      clk, rst, submit_ext;
    input  logic [3:0] guess_ext;
    output logic     psychic;

    logic [3:0] pattern, guess;
    logic correct, next, submit;

    assign next = submit_ext;

    genPatt    pat (.clk, .rst, .pattern, .next);
    userIn     inp (.clk, .rst,
                    .guess_ext, .submit_ext, .guess, .submit);
    checkGuess chk (.pattern, .guess, .correct);
    countRight cnt (.clk, .rst, .correct, .submit, .psychic);
endmodule  // psychic_tester
```
We incorrectly assigned `next` to `submit_ext` instead of `submit`!

Fixing the bug:

```
module psychic_tester (clk, rst, guess_ext, submit_ext, psychic);
  input  logic          clk, rst, submit_ext;
  input  logic [3:0]    guess_ext;
  output logic         psychic;

  logic [3:0] pattern, guess;
  logic correct, next, submit;
  assign next = submit;
  genPatt    pat (.clk, .rst, .pattern, .next);
  userIn     inp (.clk, .rst,
                  .guess_ext, .submit_ext, .guess, .submit);
  checkGuess chk (.pattern, .guess, .correct);
  countRight cnt (.clk, .rst, .correct, .submit, .psychic);
endmodule // psychic_tester
```
That's all!
Thanks for coming!