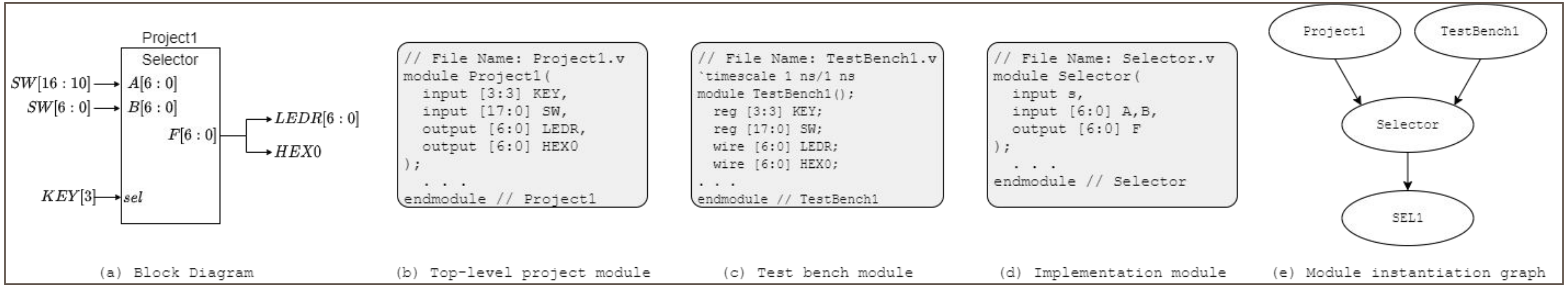
Bit Equality Checker Project

Design Specification

- Going to contain an overview/goal of what we want our project to do
- Will also contain many important diagrams
    - Block Diagram, Top-Level Project Module, Test Bench Module, Module Instantiation Graph

- Ex. Project 1 Design Specification Diagrams



```
// File Name: Project1.v
module Project1(
  input  [3:3]  KEY,
  input  [17:0] SW,
  output [6:0]  LEDR,
  output [6:0]  HEX0
);
  . . .
endmodule // Project1
```

```
// File Name: TestBench1.v
`timescale 1 ns/1 ns
module TestBench1();
  reg  [3:3]  KEY;
  reg  [17:0] SW;
  wire [6:0]  LEDR;
  wire [6:0]  HEX0;
. . .
endmodule // TestBench1
```

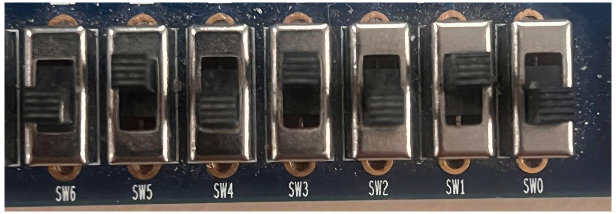
```
// File Name: Selector.v
module Selector(
  input s,
  input  [6:0] A,B,
  output [6:0] F
);
  . . .
endmodule // Selector
```

(a) Block Diagram (b) Top-level project module (c) Test bench module (d) Implementation module (e) Module instantiation graph

Project 0.5 Design Specification: Goal

- We have 2 sets of switches
- We want to compare each respective switch of both sets
  - If the 2 switches match, we want its respective LED and HEX-bit to light up
  - Otherwise, we do not want the LED and HEX to light up

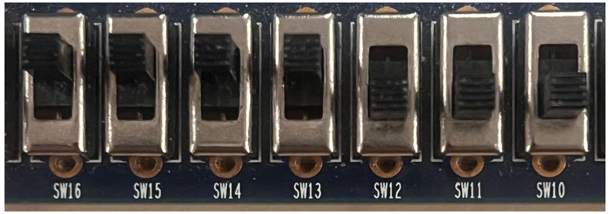
SW SET 1



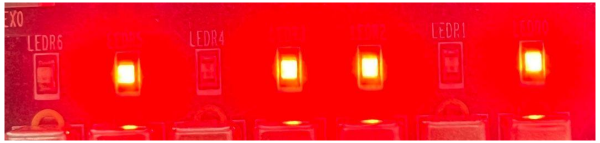
0 1 0 1 0 1 0

1 1 1 1 0 0 0

SW SET 2

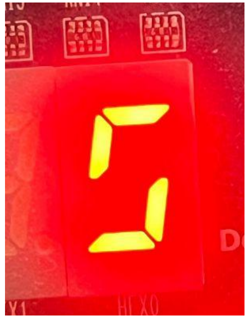


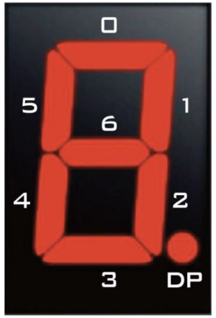
LEDs



0 1 0 1 1 0 1

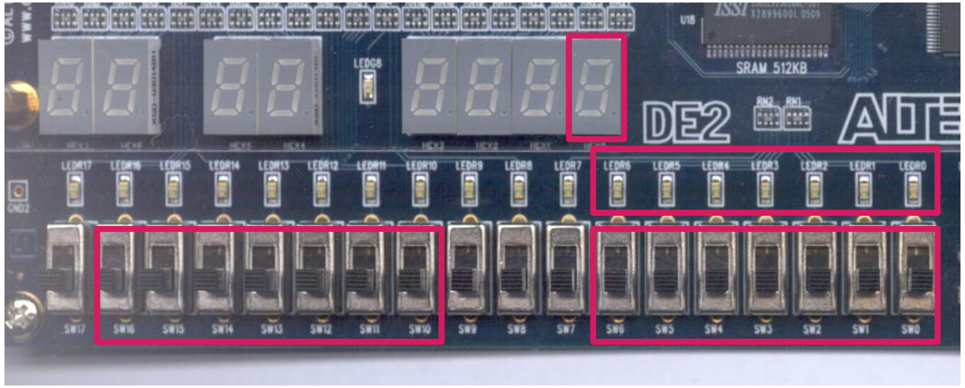
HEX Display

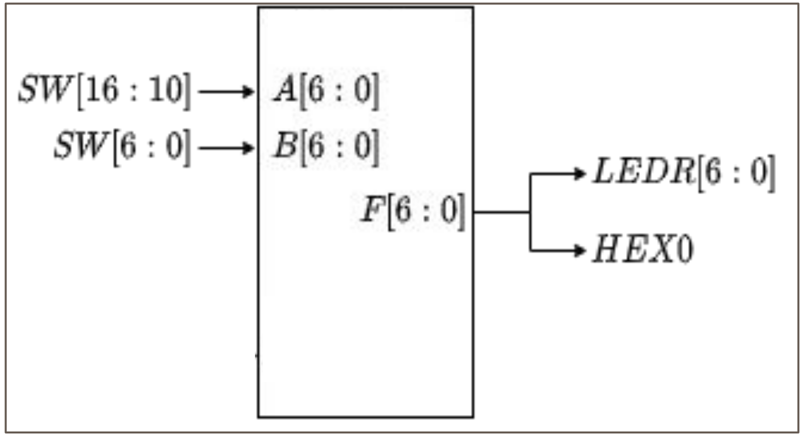




Project 0.5 Design Specification: Block Diagram

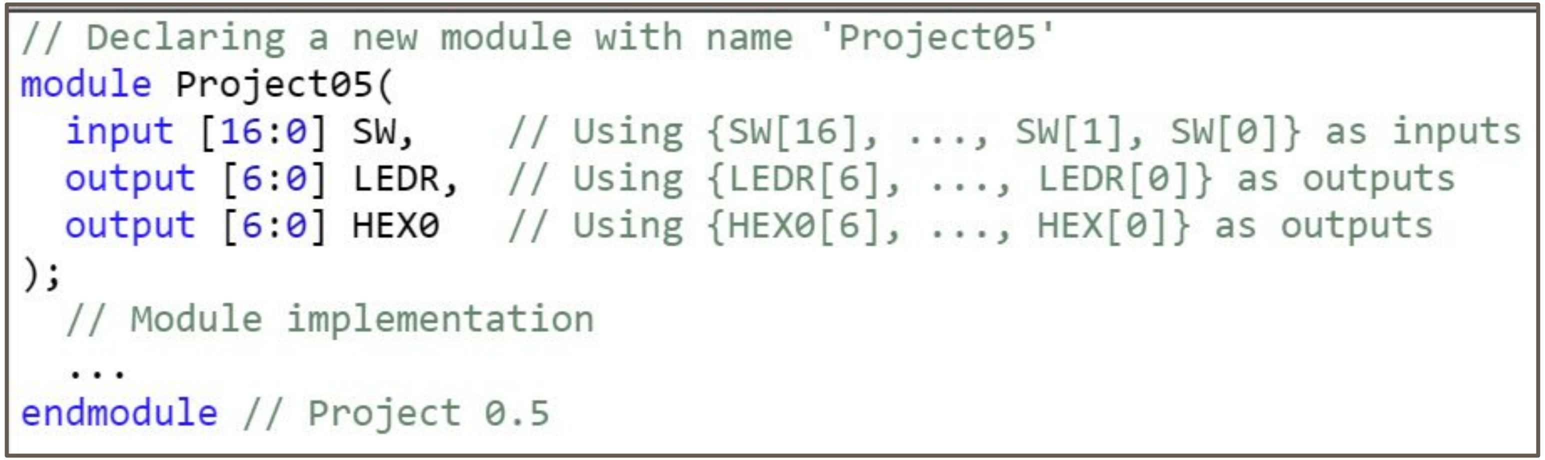
- We have 2 sets of switches as INPUTS
  - SW [16:10] - Switches explicitly labelled on board {SW16, SW15, …, SW11, SW10}
  - SW [6:0] - Switches explicitly labelled on board {SW6, SW5, …, SW1, SW0}
- Outputs
  - LEDR[6:0] - Red LEDs {LEDR[6], …, LEDR[0]}
  - HEX0 - One of the 7-Segment HEX Displays




$$\begin{tikzcd} SW[16:10] \arrow[r] & A[6:0] & \\ SW[6:0] \arrow[r] & B[6:0] & \\ & F[6:0] \arrow[r] \arrow[dr] & LEDR[6:0] \\ & & HEX0 \end{tikzcd}$$

Project 0.5 Design Specification: Top-Level Module

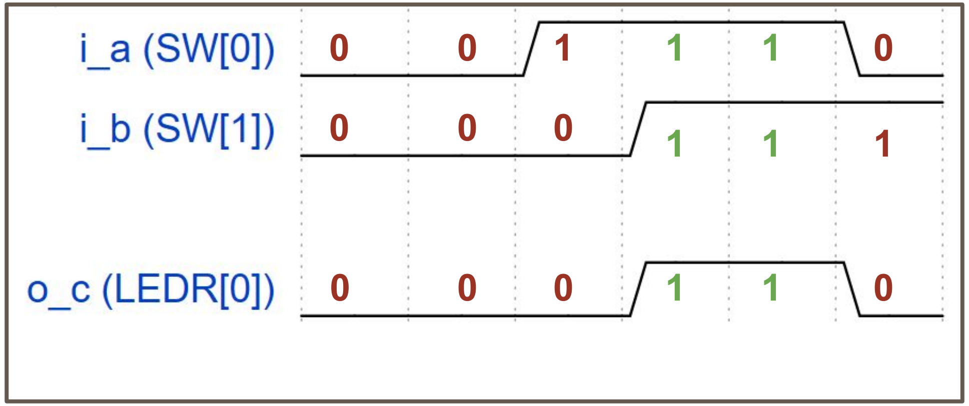
- Top-Level Module - Main entry point that interfaces w/ external components
    - Allows us to interact with the board's switches, buttons, HEX displays, LEDs, …
- **MUST BE COPIED EXACTLY AS DEFINED IN THE SPEC**
    - Required for out Autograder to work properly

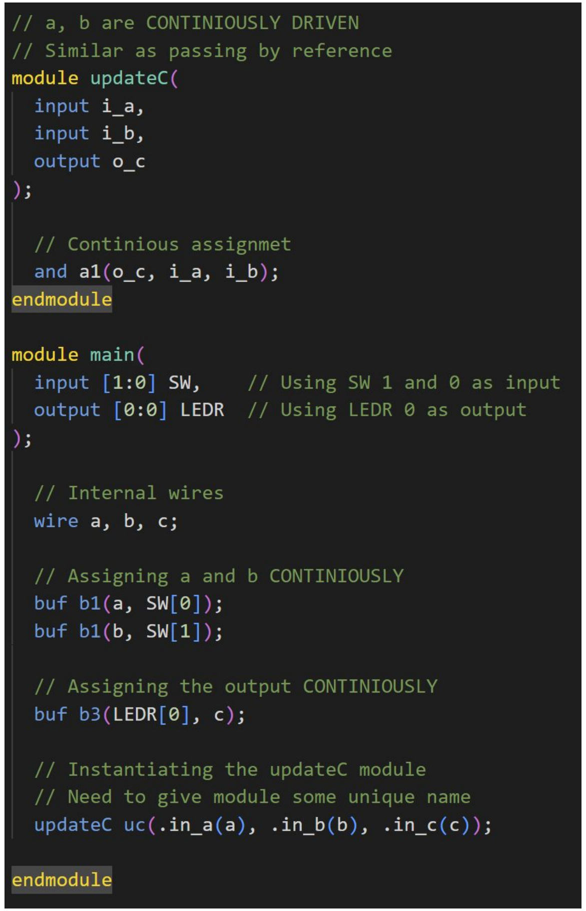


```
// Declaring a new module with name 'Project05'
module Project05(
  input [16:0] SW,    // Using {SW[16], ..., SW[1], SW[0]} as inputs
  output [6:0] LEDR,  // Using {LEDR[6], ..., LEDR[0]} as outputs
  output [6:0] HEX0   // Using {HEX0[6], ..., HEX[0]} as outputs
);
  // Module implementation
  ...
endmodule // Project 0.5
```

Verilog Modules (Structural)

- Module - Task that is **INSTANTIATED**
    - **Continuously running** once instantiated
    - Resources (Logic Elements) allocated only once instantiated
    - IT IS NOT BEING CALLED UPON



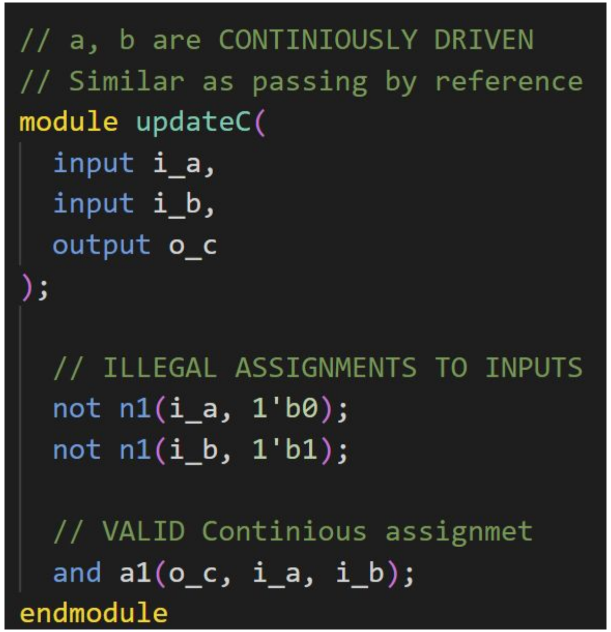


```
// a, b are CONTINIOUSLY DRIVEN
// Similar as passing by reference
module updateC(
  input i_a,
  input i_b,
  output o_c
);

  // Continious assignmet
  and a1(o_c, i_a, i_b);
endmodule

module main(
  input [1:0] SW,     // Using SW 1 and 0 as input
  output [0:0] LEDR   // Using LEDR 0 as output
);

  // Internal wires
  wire a, b, c;

  // Assigning a and b CONTINIOUSLY
  buf b1(a, SW[0]);
  buf b1(b, SW[1]);

  // Assigning the output CONTINIOUSLY
  buf b3(LEDR[0], c);

  // Instantiating the updateC module
  // Need to give module some unique name
  updateC uc(.in_a(a), .in_b(b), .in_c(c));

endmodule
```

- **REMEMBER: Verilog is describing the behavior of our parallel system**

Verilog Modules Ports (Ignoring Inouts)

- Inputs - Continuously driven by an outside element
  - You CANNOT reassign within the module
- Outputs - Needs to be driven by module
- Inouts - Can be used as input/output
  - Will not be using in 270

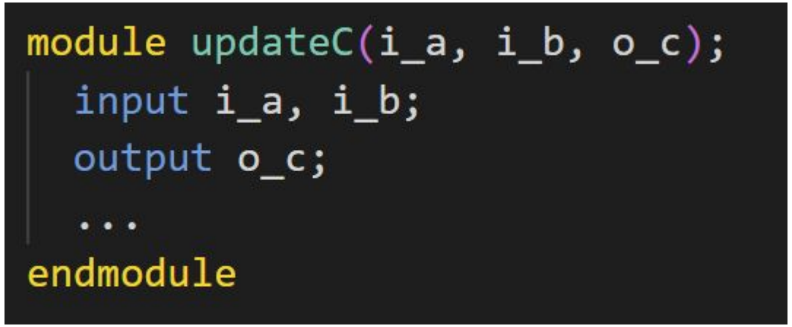


```
// a, b are CONTINIOUSLY DRIVEN
// Similar as passing by reference
module updateC(
  input i_a,
  input i_b,
  output o_c
);

  // ILLEGAL ASSIGNMENTS TO INPUTS
  not n1(i_a, 1'b0);
  not n1(i_b, 1'b1);

  // VALID Continious assignmet
  and a1(o_c, i_a, i_b);
endmodule
```

NOTE: 2 Ways to Write Verilog Module Declarations

Legacy (Verilog-1995 Standard)

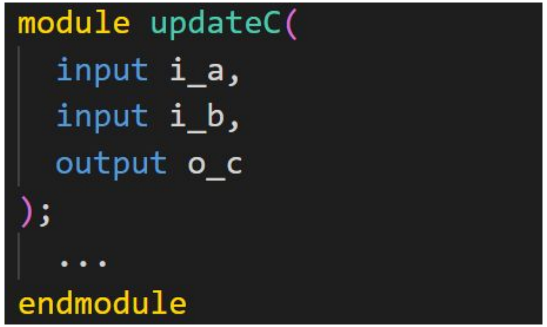
- Non-ANSI C Style
- Port declaration type outside list
- You might see this format in lab
    - Good to know that it exists



```
module updateC(i_a, i_b, o_c);
  input i_a, i_b;
  output o_c;
  ...
endmodule
```

Modern (Verilog-2001 Standard)

- ANSI C Style
- Cleaner to writer usually
    - Leads to less unusual bugs
- Less duplication

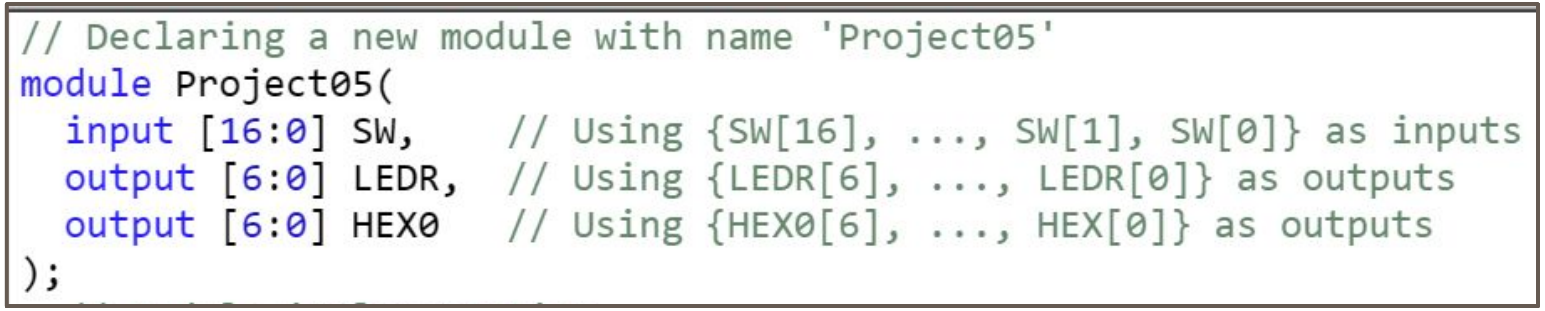


```
module updateC(
  input i_a,
  input i_b,
  output o_c
);
  ...
endmodule
```

SRC

Bit Vectors in Verilog (Structural)

- Bit Vectors - Multiple bits associated with the value
  - Allows for representation of multi-bit signal
  - By default, variables are 1-bit
- Representation
  - ‘wire [most significant bit : least significant bit] name’
  - wire [3:0] a === {a[3], a[2], a[1], a[0]}
  - wire [0:4] b === {b[0], b[1], b[2], b[3], b[4]}
  - Bit range = Absolute value(msb - lsb) + 1

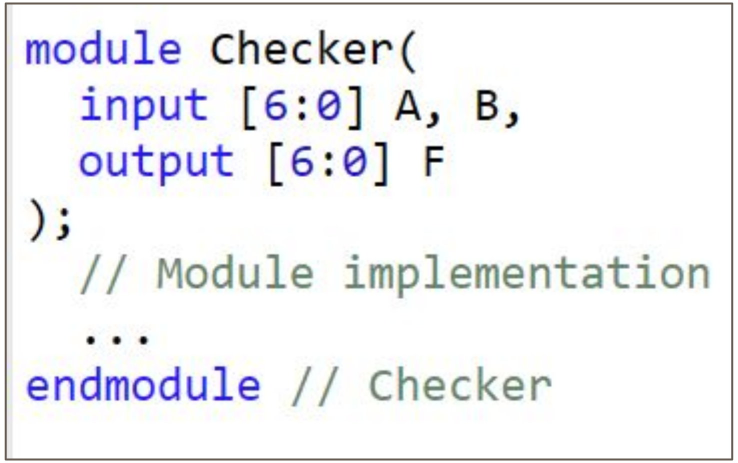


```
// Declaring a new module with name 'Project05'
module Project05(
  input [16:0] SW,    // Using {SW[16], ..., SW[1], SW[0]} as inputs
  output [6:0] LEDR,  // Using {LEDR[6], ..., LEDR[0]} as outputs
  output [6:0] HEX0   // Using {HEX0[6], ..., HEX[0]} as outputs
);
```

Project 0.5 Design Specification: Implementation Module(s)

- We may give you modules we want you to implement in a certain way
  - Sometimes we will test specific modules instead of the whole design

- Project 0.5 Checker Module
  - Inputs: 7-bit A, 7-bit B
  - Outputs: 7-bit F
  - Go through each bit of A and B
  - **If A[i] == B[i], then F[i] = 1**
  - **Otherwise, if A[i] != B[i], then F[i] = 0**



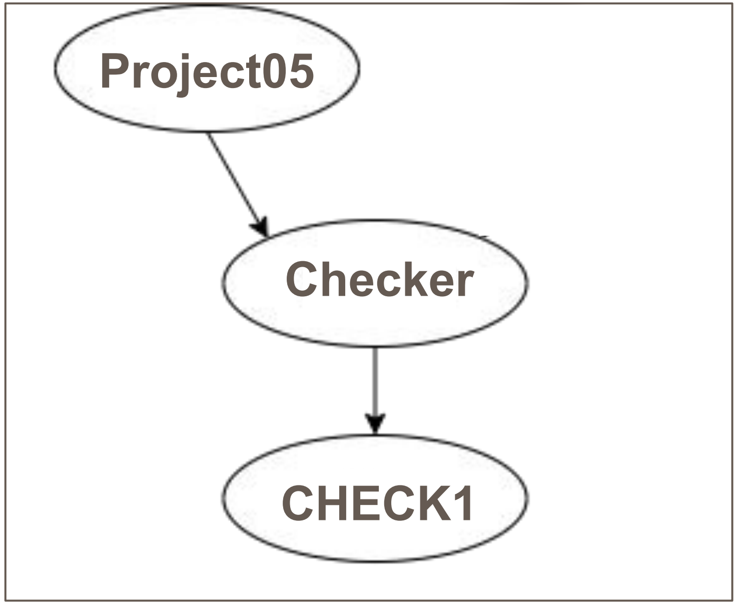
```
module Checker(
  input [6:0] A, B,
  output [6:0] F
);
  // Module implementation
  ...
endmodule // Checker
```

Project 0.5 Design Specification: Module Instantiation Graph

- Graph to help split up the work we need to accomplish for our design

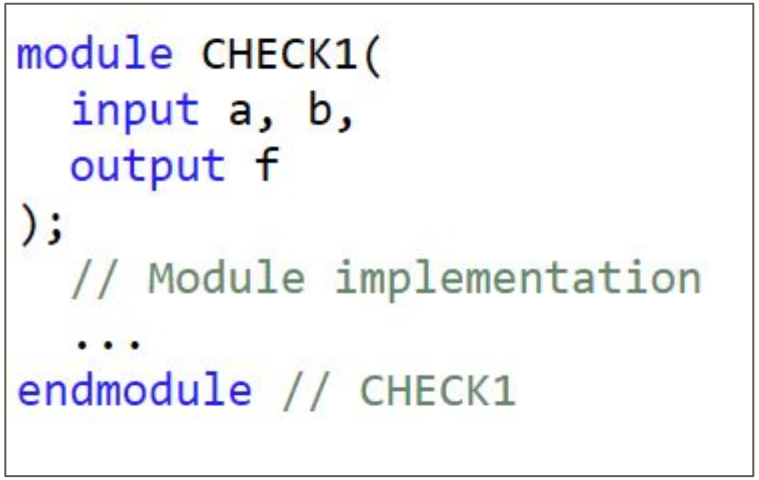
- ‘Project05’ instantiates ‘Checker’
- ‘Checker’ instantiates ‘CHECK1’

- CHECK1 is a low-level helper function
  - Want to separate work as much as possible



Project 0.5: Giving Away CHECK1 Helper Function

- Project 0.5 CHECK1 Module
  - Inputs: 1-bit a, 1-bit b
  - Outputs: 1-bit f
  - Check ‘a’ and ‘b’ bit value
  - **If a == b, then f = 1**
  - **Otherwise, if a != b, then f = 0**



```
module CHECK1(
  input a, b,
  output f
);
  // Module implementation
  ...
endmodule // CHECK1
```

Start Small
(Building the CHECK1 Circuit)

Figuring Out CHECK1 Module (Step 1. Truth Table)

- Project 0.5 CHECK1 Module
  - If a == b, then f = 1
  - Otherwise, if a != b, then f = 0

a	b	f

How many rows do we need for this truth table?

Figuring Out CHECK1 Module (Step 1. Truth Table)

- Project 0.5 CHECK1 Module
  - If a == b, then f = 1
  - Otherwise, if a != b, then f = 0

- # Rows = 2^(# Inputs)
  - 2 Inputs → 4 Combinations / Rows

What are the inputs to this truth table?

a	b	f

Figuring Out CHECK1 Module (Step 1. Truth Table)

- Project 0.5 CHECK1 Module
  - If a == b, then f = 1
  - Otherwise, if a != b, then f = 0
- # Rows = 2^(# inputs)
  - 2 Inputs → 4 Combinations / Rows

What are the outputs of this truth table?

a	b	f
0	0	
0	1	
1	0	
1	1	

Figuring Out CHECK1 Module (Step 1. Truth Table)

- Project 0.5 CHECK1 Module
  - If a == b, then f = 1
  - Otherwise, if a != b, then f = 0
- # Rows = 2^(# inputs)
  - 2 Inputs → 4 Combinations / Rows

a	b	f
0	**0**	**1**
0	1	0
1	0	0
1	**1**	**1**

Figuring Out CHECK1 Module (Step 2. Equation)

Now Build an Equation for ‘f’

a	b	f
0	0	1
0	1	0
1	0	0
1	1	1

Figuring Out CHECK1 Module (Step 2. Equation)

1. Determine which rows resulted in ‘f’ being True (1)

a	b	f
0	0	1
0	1	0
1	0	0
1	1	1

Figuring Out CHECK1 Module (Step 2. Equation)

1. Determine which row resulted in ‘f’ being True (1)

2. Generate an Expression for Each Combination
   a. Only that combination should result in output being true

For combination (0, 0) → a’ && b’

The only time this expression is True is when (0, 0) is passed

Fot combination (1, 1) → a && b

The only time this expression is True is when (1, 1) is passed

a	b	f
0	0	1
0	1	0
1	0	0
1	1	1

Figuring Out CHECK1 Module (Step 2. Equation)

1. Determine which row resulted in ‘f’ being True (1)
2. Generate an Expression for Each Combination
   a. Only that combination should result in output being true

For combination (0, 0) → a’ && b’

The only time this expression is True is when (0, 0) is passed

Fot combination (1, 1) → a && b

The only time this expression is True is when (1, 1) is passed

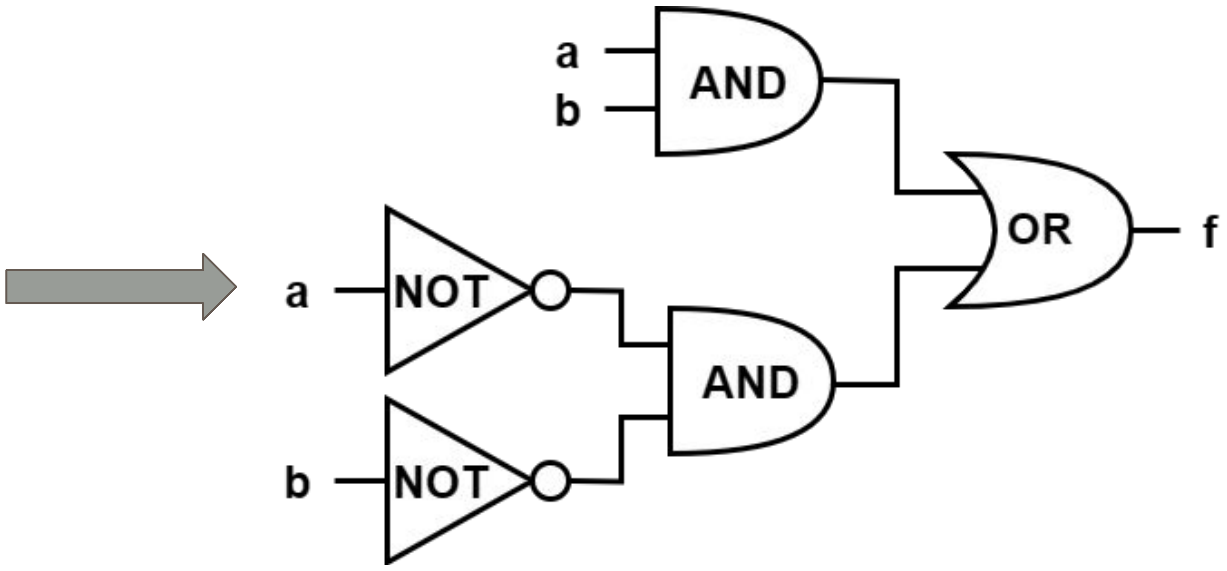
3. OR each of the combinations

f = (a’ && b’) || (a && b)

a	b	f
0	0	1
0	1	0
1	0	0
1	1	1

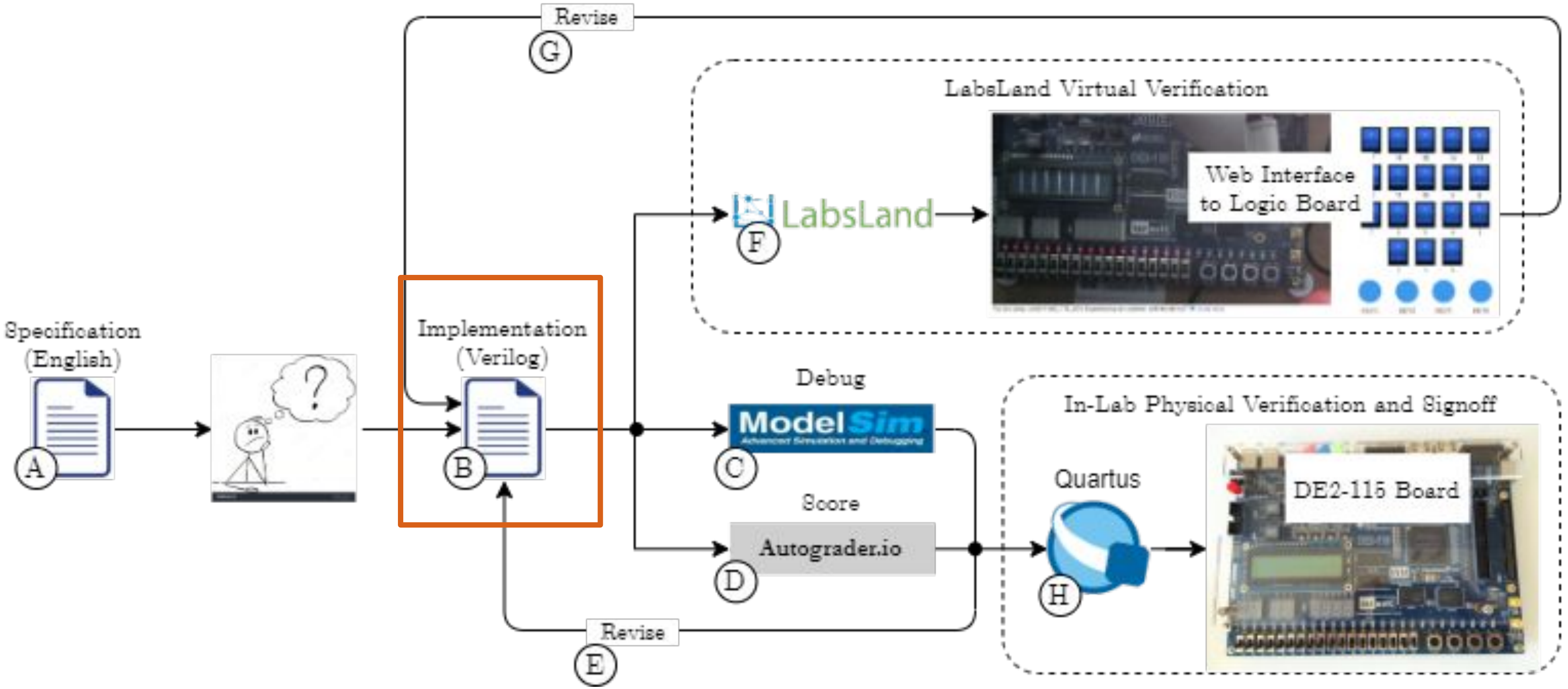
Figuring Out CHECK1 Module (Step 3. Logic Gate Diagram)

f = (a’ && b’) || (a && b)



Implementing Verilog

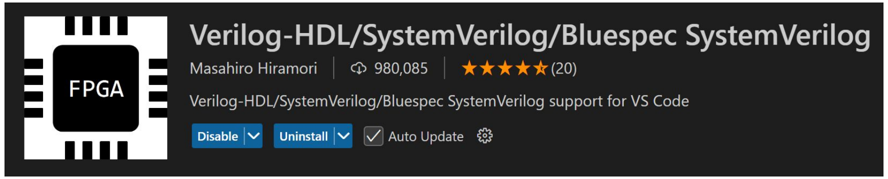
Overview of the Tools


Revise
G
LabsLand Virtual Verification
Web Interface to Logic Board
LabsLand
F
Specification (English)
A
Implementation (Verilog)
B
Debug
ModelSim
Advanced Simulation and Debugging
C
Score
Autograder.io
D
In-Lab Physical Verification and Signoff
Quartus
H
DE2-115 Board
Revise
E

Text Editor Recommendation for Verilog Coding

- Can use whatever you want to write Verilog
  - I've seen notepad, notepad++, Quartus Prime, … used as text editors

- My Recommendation - Use **VS Code**
  - Use Intellisense Extension below



Verilog-HDL/SystemVerilog/Bluespec SystemVerilog

Masahiro Hiramori | 980,085 | ★★★★★ (20)

Verilog-HDL/SystemVerilog/Bluespec SystemVerilog support for VS Code

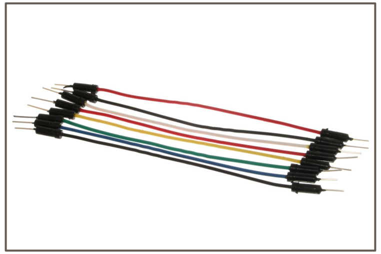
- Please please please don't use ModelSim as a text editor
  - It has led to many issues

Implementing Verilog
(Implementing the CHECK1 Module)

Verilog Wire Data Type

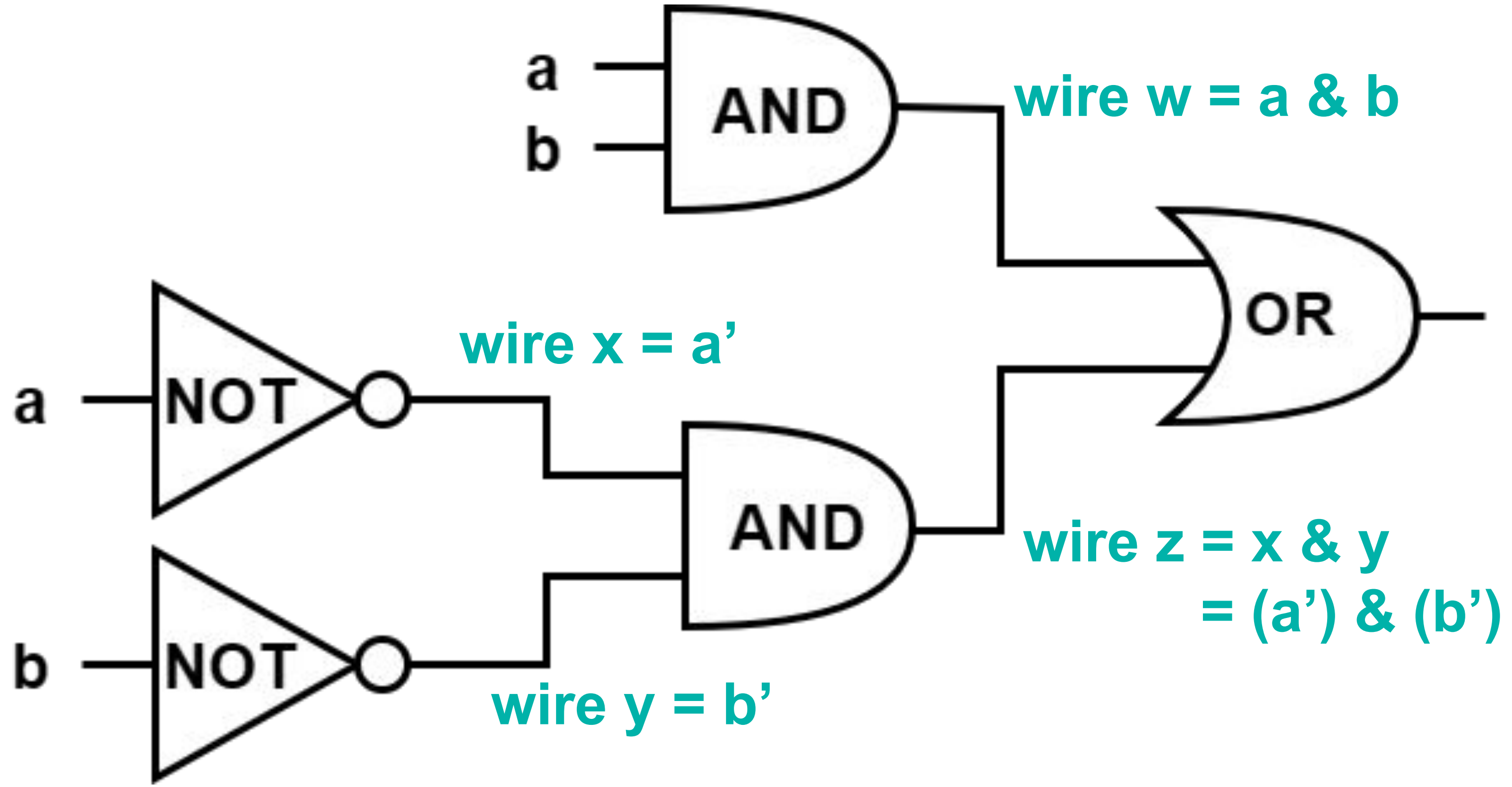
- Wire - Used to model physical connections between components in digital circuits
    - **They do not store values**
    - **They are continuously driven (assigned)**
        - The value is continuously determined by the entity driving the net
    - **If nothing is driving it, it holds no value (Undefined ‘Z’)**

- **Wire** - The only Net data type we will be using in class
    - Can be thought of as actual physical wires



Determining Circuit Wires

NOTE: Inputs and Outputs are implicitly declared as wires already



Verilog’s Many Levels of Abstraction

“BEHAVIORAL VERILOG”	Behavioral Level	Project 3B - 7
	Dataflow Level	
“STRUCTURAL VERILOG”	Gate Level	Projects 0 - 3A
	Switch Level	Will Discuss Later

USE STRUCTURAL VERILOG FOR Projects 0-3A!!!

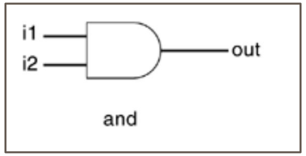
- **You SHOULD NOT be using**
  - **Continuous ‘assign’ statements**
  - **Procedural statements ‘always’ or ‘initial’**

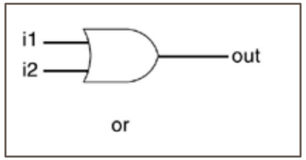
- These abstractural statements are used in “Behavioral” Verilog
  - Will be using in Projects 3B - 7

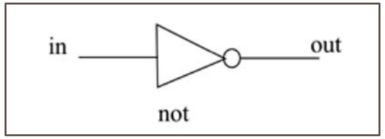
- **You SHOULD ONLY be using**
  - **Primitive gate instantiations as described in this slide**
  - **Module instantiations as described in this slide**

Structural Verilog (Gate Level Modeling)

- One of the lowest levels of Verilog abstraction!
- Describing you circuit with primitive gates
  - **Creating a one-to-one mapping between logic circuit diagram ←→ Verilog Description**





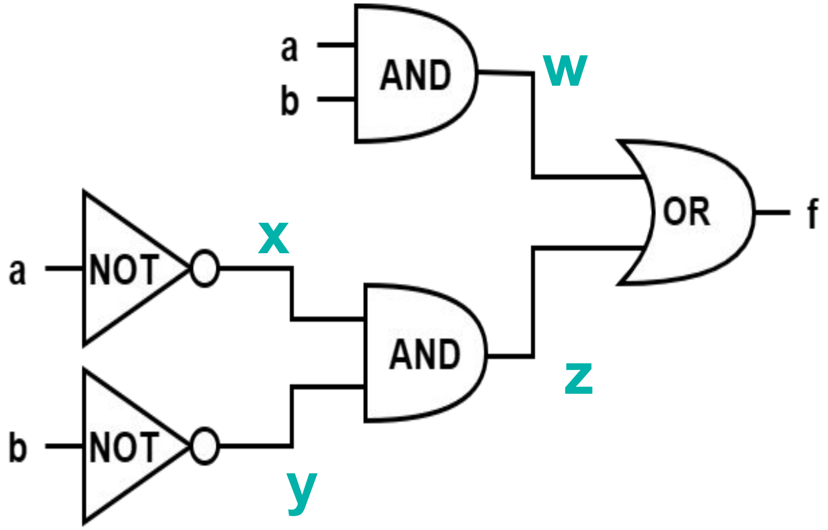


```
wire OUT, IN1, IN2;

// basic gate instantiations.
and a1(OUT, IN1, IN2);
nand na1(OUT, IN1, IN2);
or or1(OUT, IN1, IN2);
```

<gate type> <instance name>(<output name> <input list>)

Implementation in Verilog (Declaring Wires)

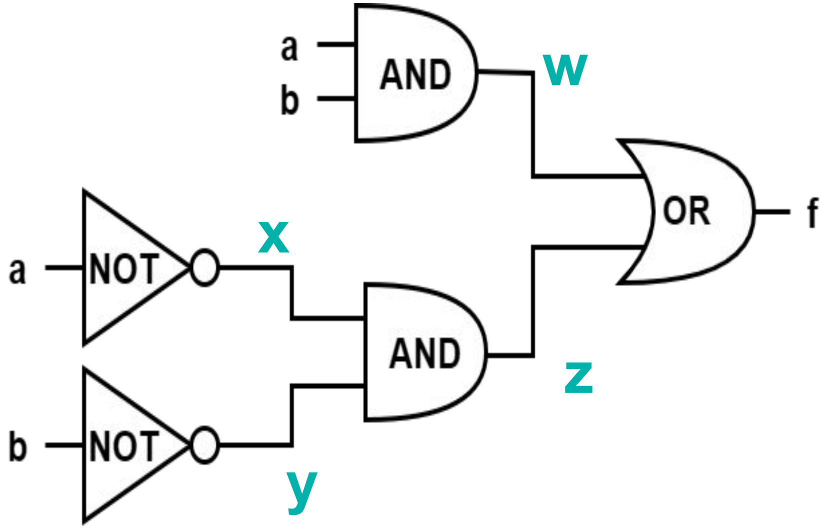
```
module CHECK1(
  input a, b,
  output f
);

  // Step 1. Declare Your Wires
  ...

endmodule // CHECK1
```



```
wire OUT, IN1, IN2;
```

Implementation in Verilog (Instantiating Gates)

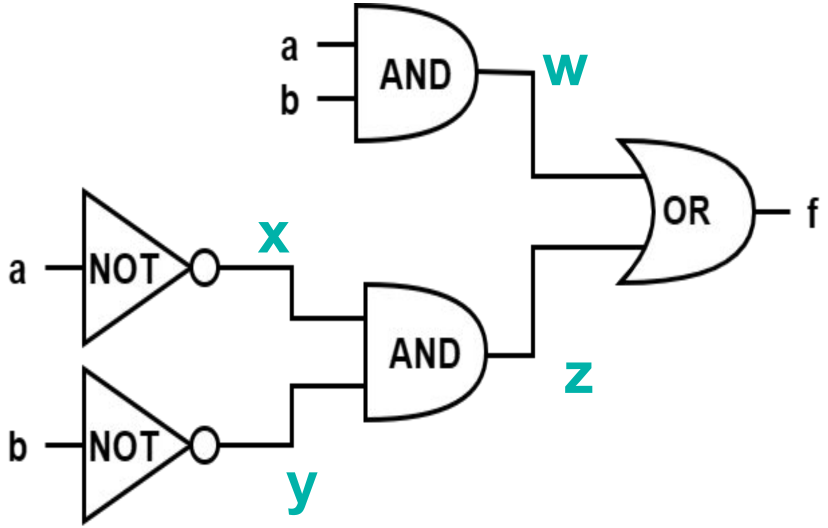
```
module CHECK1(
  input a, b,
  output f
);

  // Step 1. Declare Your Wires
  // Declaring 1-bit wires w, x, y, z
  wire w, x, y, z;

  // Step 2. Instantiate the gates
  ...

endmodule // CHECK1
```



```
// basic gate instantiations.
and a1(OUT, IN1, IN2);
nand na1(OUT, IN1, IN2);
or or1(OUT, IN1, IN2);
```

Implementation in Verilog (DONE!!!)

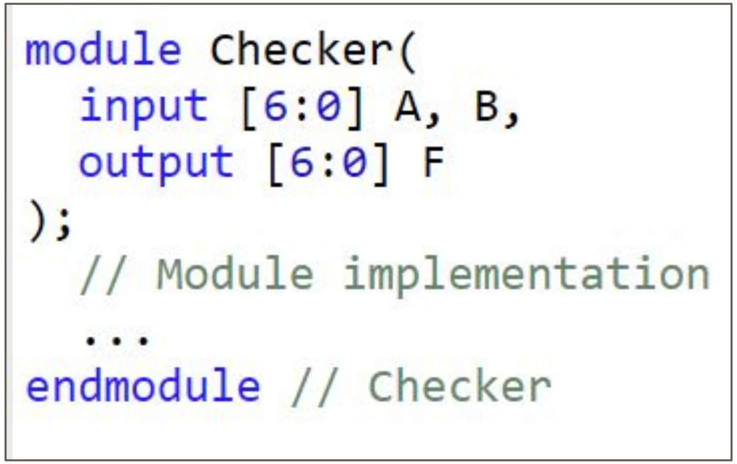
```
module CHECK1(
  input a, b,
  output f
);

  // Step 1. Declare Your Wires
  // Declaring 1-bit wires w, x, y, z
  wire w, x, y, z;

  // Step 2. Instantiate the gates
  and a1(w, a, b);
  not n1(x, a);
  not n2(y, b);
  and a2(z, x, y);
  or o1(f, w, z);

endmodule // CHECK1
```



Implementing Verilog
(Implementing the Checker Module)

Checker Module (Overview)

- Inputs: 7-bit A, 7-bit B
- Outputs: 7-bit F
- Go through each bit of A and B
- **If A[i] == B[i], then F[i] = 1**
- **Otherwise, if A[i] != B[i], then F[i] = 0**



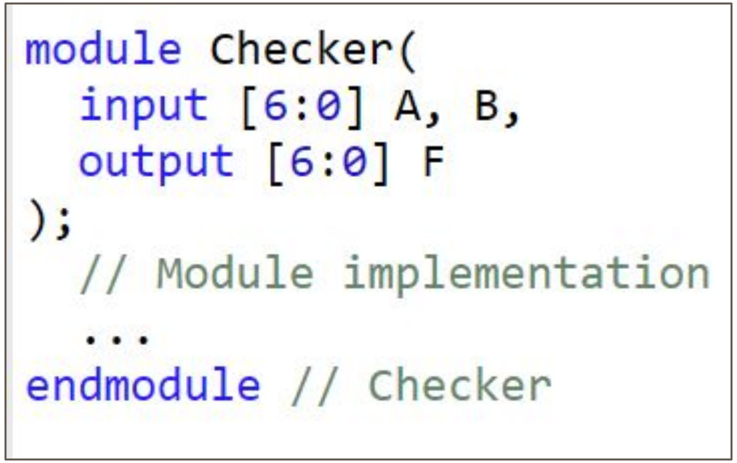
```
module Checker(
  input [6:0] A, B,
  output [6:0] F
);
  // Module implementation
  ...
endmodule // Checker
```

How can we use CHECK1 to implement the Checker module?

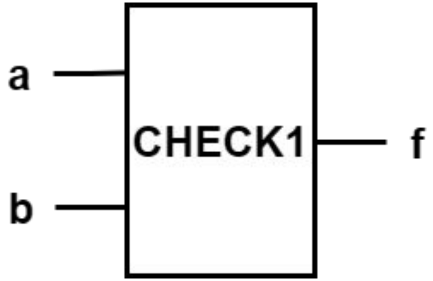
Checker Module (Building the Logical Circuit)

- **If A[i] == B[i], then F[i] = 1**
- **Otherwise, if A[i] != B[i], then F[i] = 0**

Build the Checker logical circuit using CHECK1



```
module Checker(
  input [6:0] A, B,
  output [6:0] F
);
  // Module implementation
  ...
endmodule // Checker
```



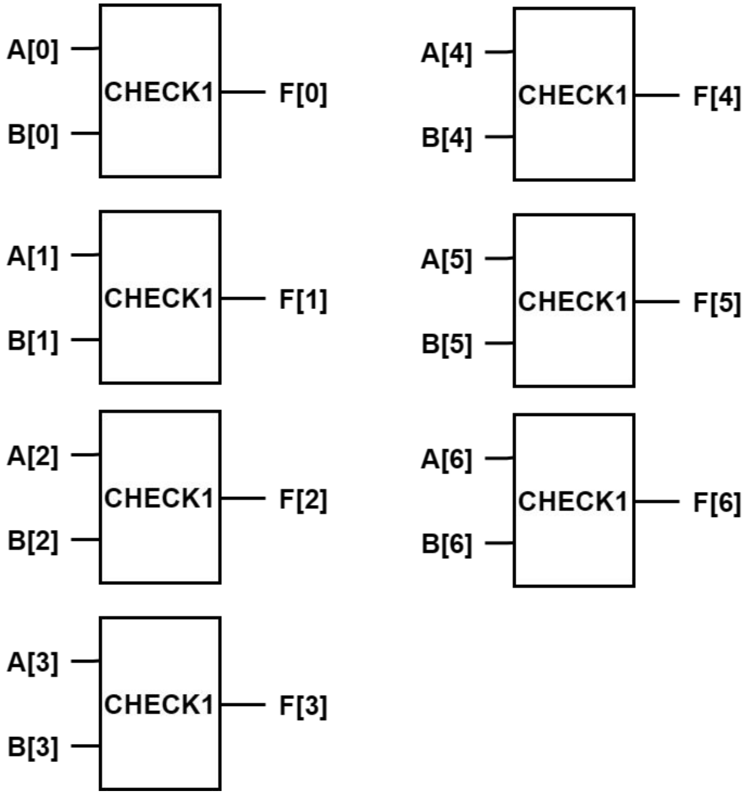
Checker Module (Building the Logical Circuit)

- **If A[i] == B[i], then F[i] = 1**
- **Otherwise, if A[i] != B[i], then F[i] = 0**

- 7 Instances of CHECK1
- Each checking on their own bit pair

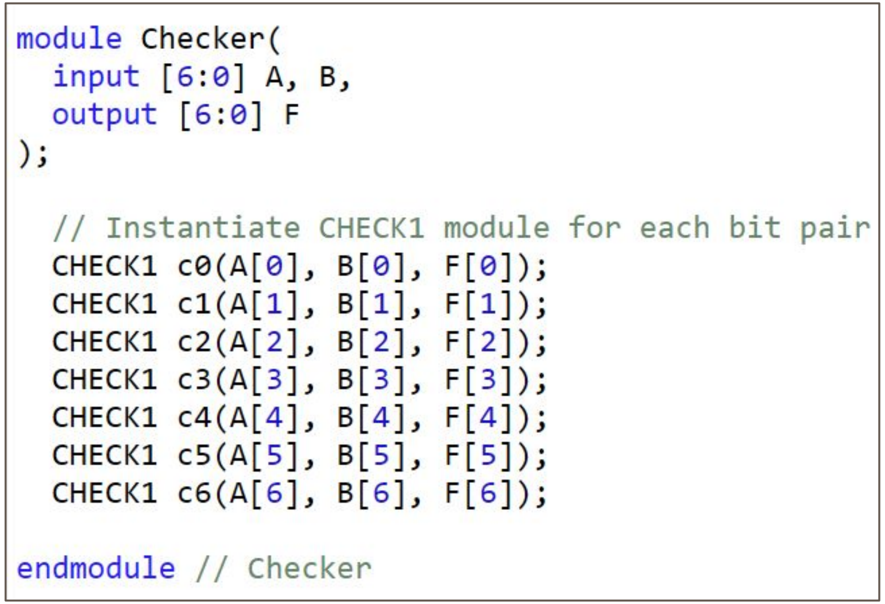
Now Implement Checker in Verilog!

```
CHECK1 c0(A[0], B[0], F[0]);
```

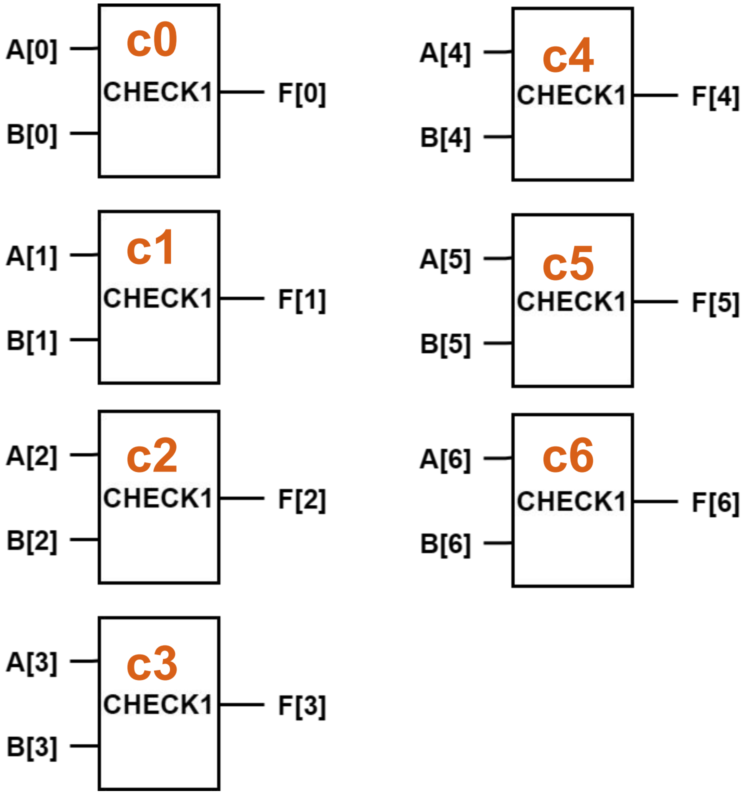


Checker Module (almost DONE!!)

- **If A[i] == B[i], then F[i] = 1**
- **Otherwise, if A[i] != B[i], then F[i] = 0**

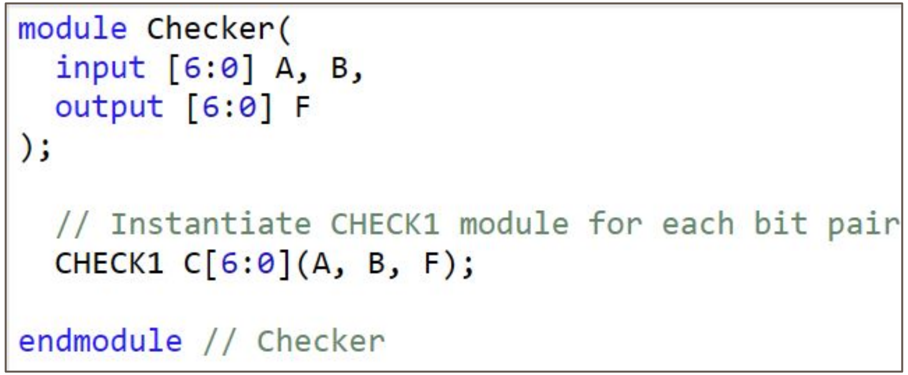


```
module Checker(
  input [6:0] A, B,
  output [6:0] F
);

  // Instantiate CHECK1 module for each bit pair
  CHECK1 c0(A[0], B[0], F[0]);
  CHECK1 c1(A[1], B[1], F[1]);
  CHECK1 c2(A[2], B[2], F[2]);
  CHECK1 c3(A[3], B[3], F[3]);
  CHECK1 c4(A[4], B[4], F[4]);
  CHECK1 c5(A[5], B[5], F[5]);
  CHECK1 c6(A[6], B[6], F[6]);

endmodule // Checker
```

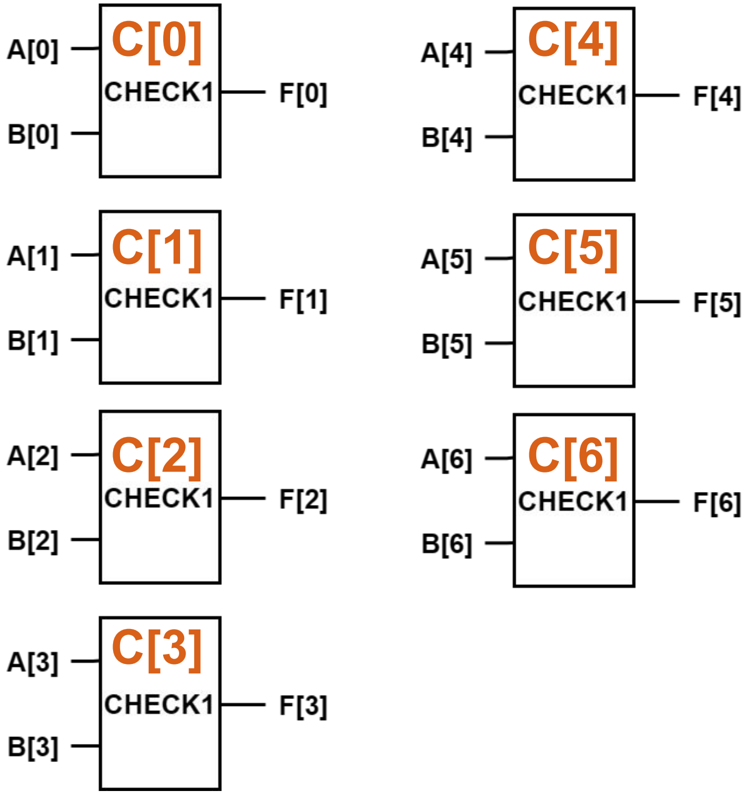


Checker Module (Shortcut Implementation with Array)

- Can create an array of instantiations!
- Shortcut implementation of previous



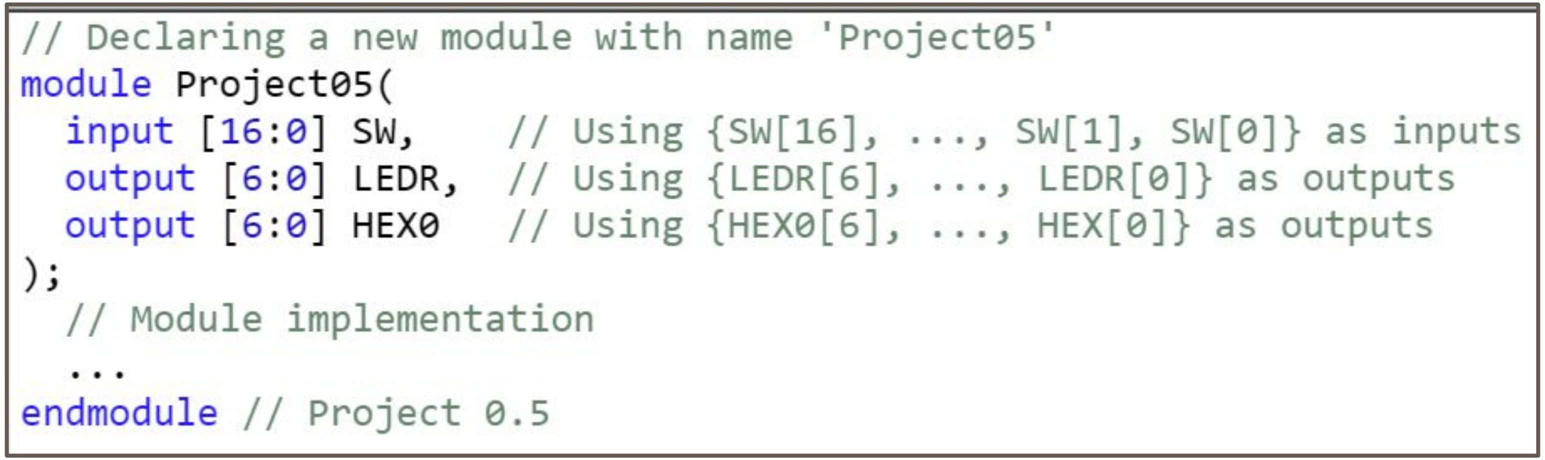
```
module Checker(
  input [6:0] A, B,
  output [6:0] F
);

  // Instantiate CHECK1 module for each bit pair
  CHECK1 C[6:0](A, B, F);

endmodule // Checker
```



Implementing Verilog (Implementing the Top-Level Module)

Project 0.5 Design Specification: Top-Level Module

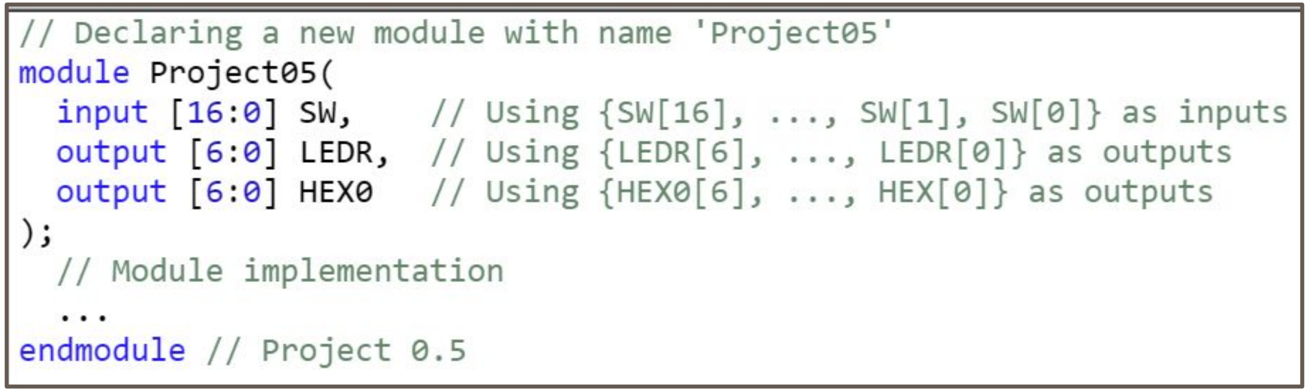
- Top-Level Module - Main entry point that interfaces w/ external components
    - Allows us to interact with the board's switches, buttons, HEX displays, LEDs, …
- **MUST BE COPIED EXACTLY AS DEFINED IN THE SPEC**
    - Required for out Autograder to work properly



```
// Declaring a new module with name 'Project05'
module Project05(
  input [16:0] SW,    // Using {SW[16], ..., SW[1], SW[0]} as inputs
  output [6:0] LEDR,  // Using {LEDR[6], ..., LEDR[0]} as outputs
  output [6:0] HEX0   // Using {HEX0[6], ..., HEX[0]} as outputs
);
  // Module implementation
  ...
endmodule // Project 0.5
```

Goal of Project05 Top-Level Module

- Handle the incoming Switch inputs values
- Instantiate Checker module to handle computation
- Drive HEX0 and LEDR outputs

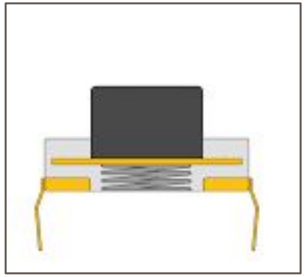


```
// Declaring a new module with name 'Project05'
module Project05(
  input [16:0] SW,    // Using {SW[16], ..., SW[1], SW[0]} as inputs
  output [6:0] LEDR,  // Using {LEDR[6], ..., LEDR[0]} as outputs
  output [6:0] HEX0   // Using {HEX0[6], ..., HEX[0]} as outputs
);
  // Module implementation
  ...
endmodule // Project 0.5
```

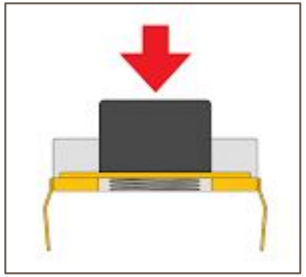
Active Low vs. Active High

- Active High - ON = 1 and OFF = 0
- Active Low - ON = 0 and OFF = 1

FOR PUSH-BUTTONS (ACTIVE LOW)



Unpressed
Logical High (1)

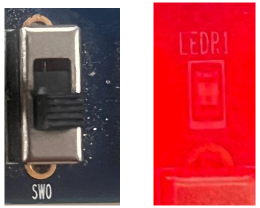


Pressed
Logical Low (0)

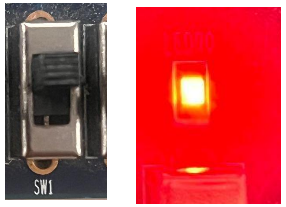
Component	Standard Name	Active High/Low
Slider Switches	SW[17:0]	High
Push-Button Keys	KEY[3:0]	Low
Red LEDs	LEDR[17:0]	High
Green LEDs	LEDG[8:0]	High
HEX Displays	HEX7[6:0], . . .,HEX0[6:0]	Low
50 MHz Clock	CLOCK_50	N/A

Active Low vs. Active High

For Switches / LED (ACTIVE HIGH)



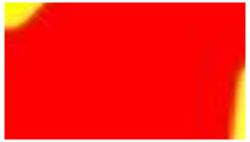
Switch / LED Off
Logical Low (0)



Switch / LED On
Logical High (1)

For HEX Displays (ACTIVE LOW)

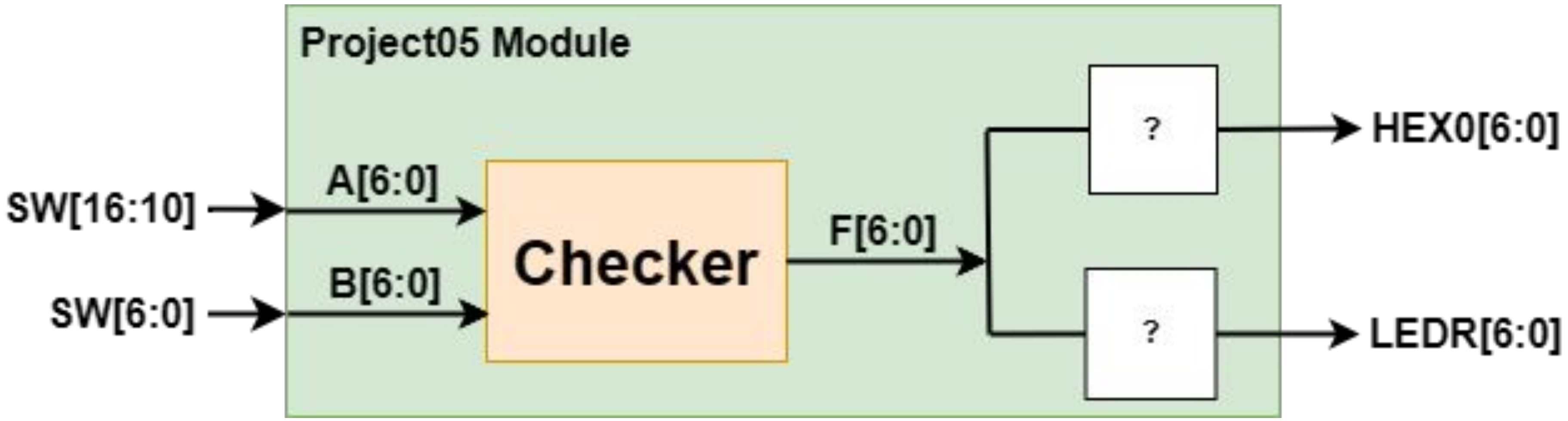
Segment On
Logical Low (0)



Segment Off
Logical High (1)

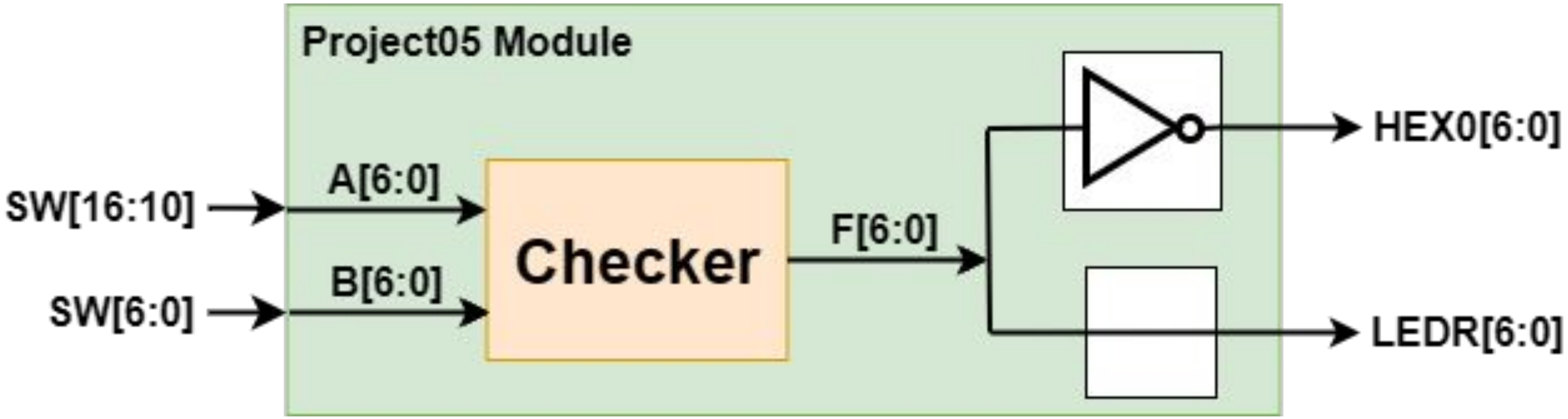
Component	Standard Name	Active High/Low
Slider Switches	SW[17:0]	High
Push-Button Keys	KEY[3:0]	Low
Red LEDs	LEDR[17:0]	High
Green LEDs	LEDG[8:0]	High
HEX Displays	HEX7[6:0], . . .,HEX0[6:0]	Low
50 MHz Clock	CLOCK_50	N/A

Project05 Logical Gate Diagram



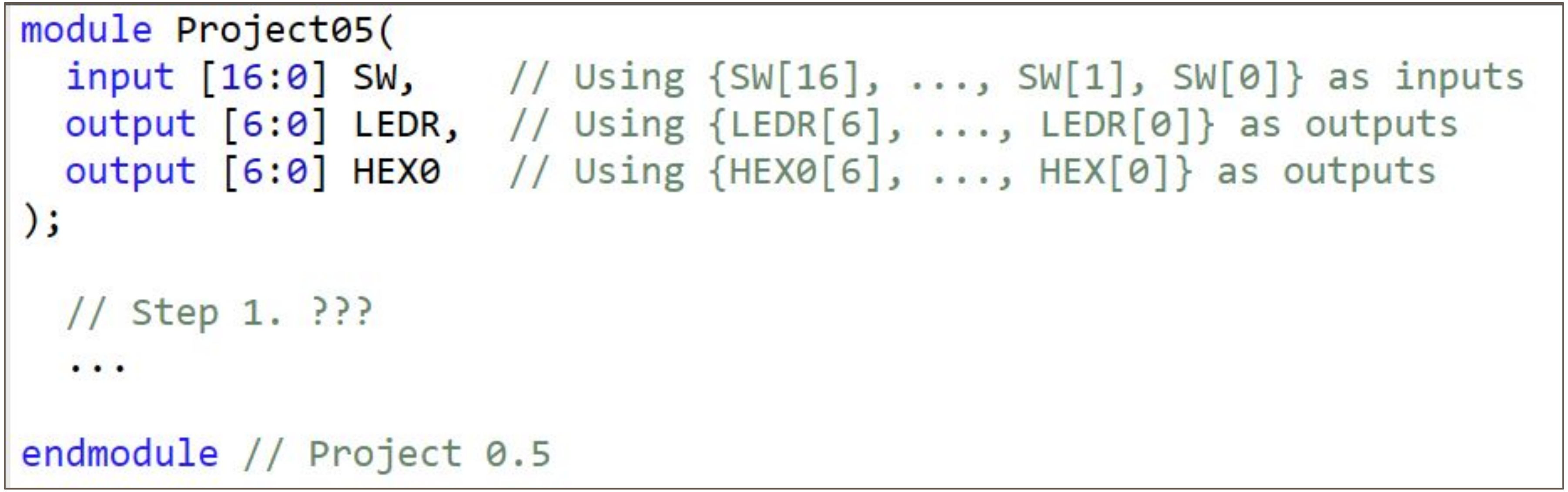
What logical gate should we connect each of the outputs from F?
HINT: They won’t be the same

Project05 Logical Gate Diagram



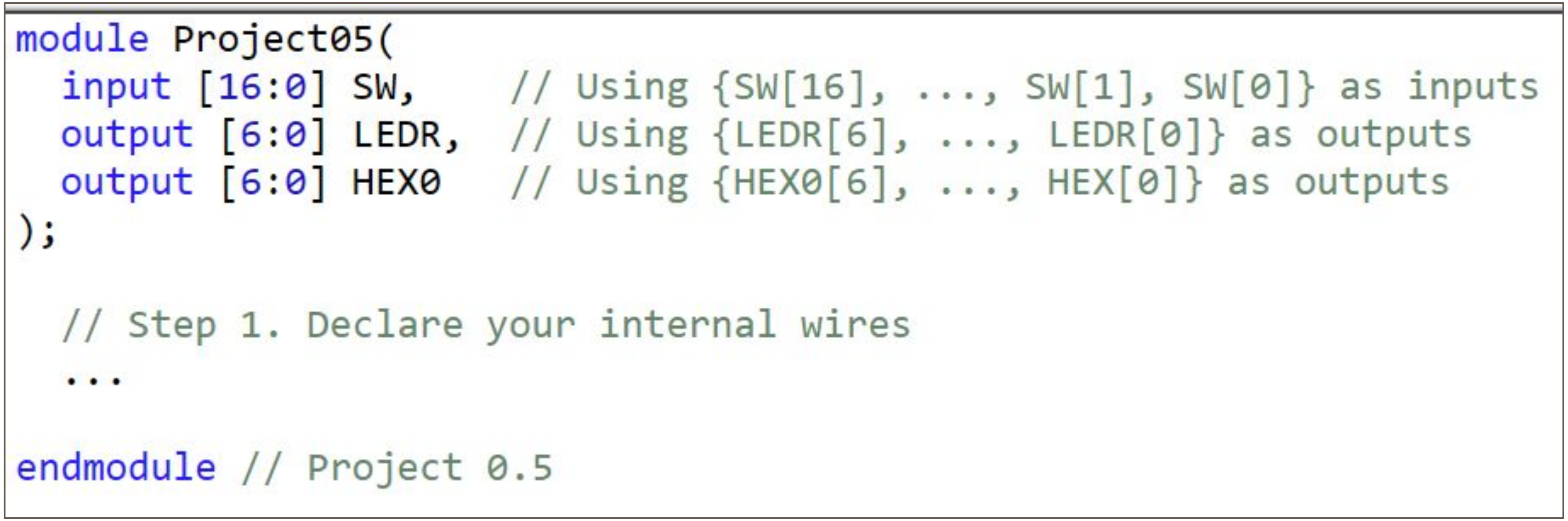
What are the internal wires of the Project05 Module?

Implementing Project05 Module



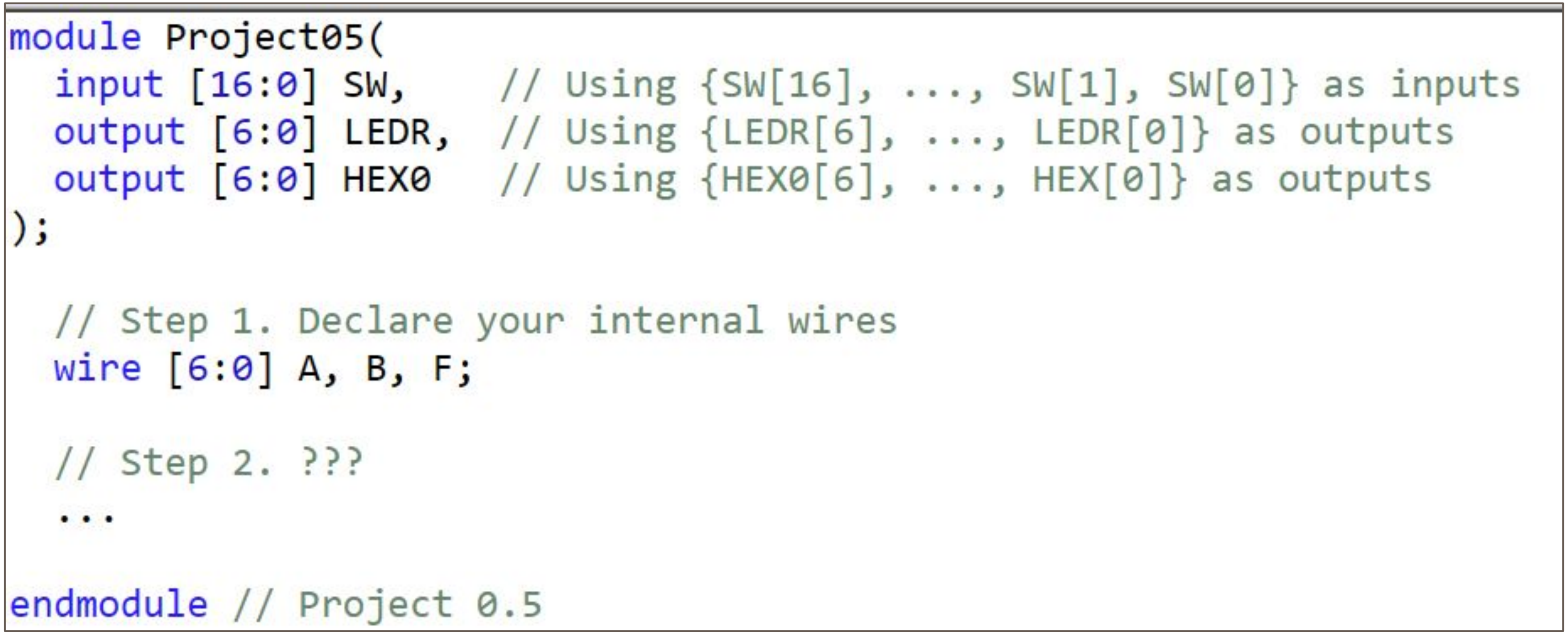
```
module Project05(
  input [16:0] SW,    // Using {SW[16], ..., SW[1], SW[0]} as inputs
  output [6:0] LEDR,  // Using {LEDR[6], ..., LEDR[0]} as outputs
  output [6:0] HEX0   // Using {HEX0[6], ..., HEX[0]} as outputs
);

  // Step 1. ???
  ...

endmodule // Project 0.5
```

Implementing Project05 Module



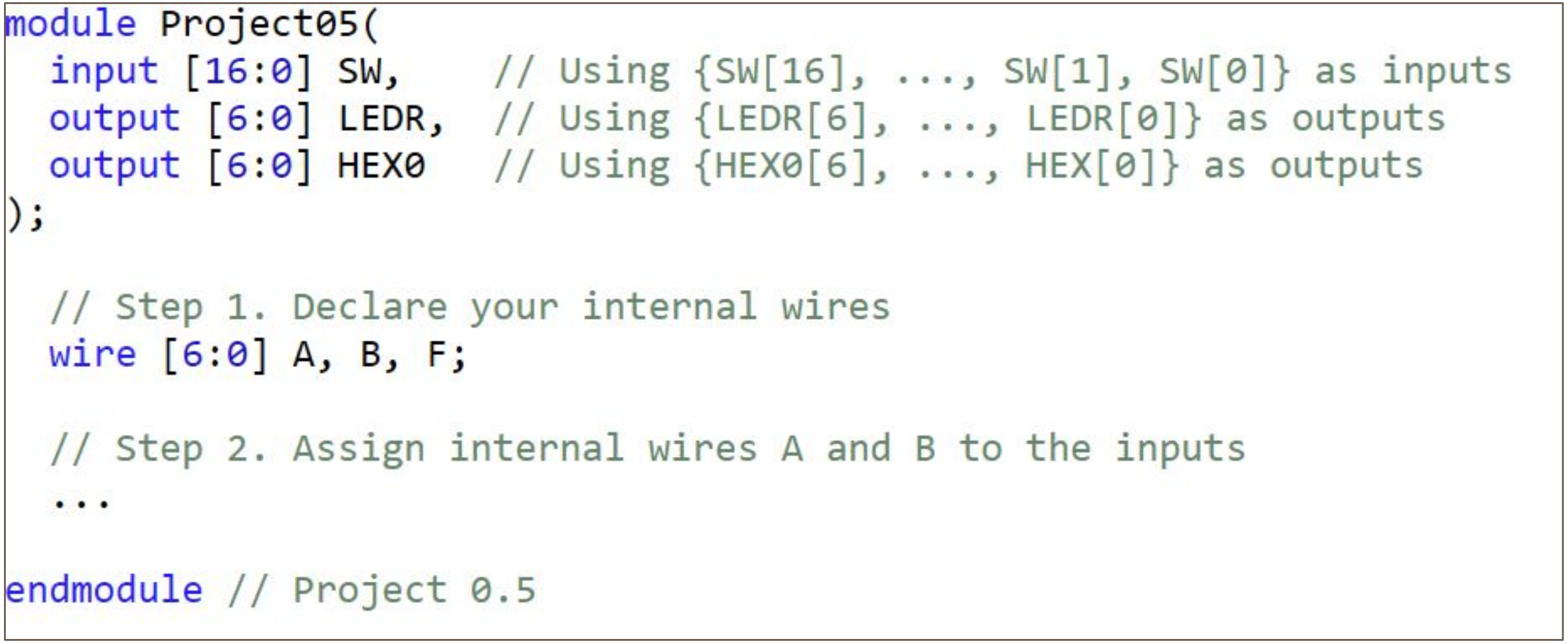
```
module Project05(
  input [16:0] SW,    // Using {SW[16], ..., SW[1], SW[0]} as inputs
  output [6:0] LEDR,  // Using {LEDR[6], ..., LEDR[0]} as outputs
  output [6:0] HEX0   // Using {HEX0[6], ..., HEX[0]} as outputs
);

  // Step 1. Declare your internal wires
  ...

endmodule // Project 0.5
```

Implementing Project05 Module



```
module Project05(
  input [16:0] SW,    // Using {SW[16], ..., SW[1], SW[0]} as inputs
  output [6:0] LEDR,  // Using {LEDR[6], ..., LEDR[0]} as outputs
  output [6:0] HEX0   // Using {HEX0[6], ..., HEX[0]} as outputs
);

  // Step 1. Declare your internal wires
  wire [6:0] A, B, F;

  // Step 2. ???
  ...

endmodule // Project 0.5
```

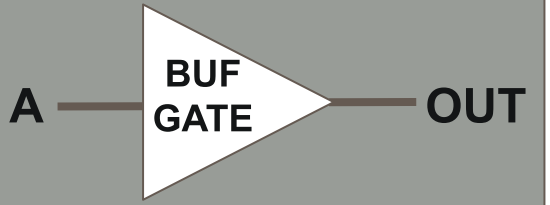
Implementing Project05 Module



```
module Project05(
  input [16:0] SW,    // Using {SW[16], ..., SW[1], SW[0]} as inputs
  output [6:0] LEDR,  // Using {LEDR[6], ..., LEDR[0]} as outputs
  output [6:0] HEX0   // Using {HEX0[6], ..., HEX[0]} as outputs
);

  // Step 1. Declare your internal wires
  wire [6:0] A, B, F;

  // Step 2. Assign internal wires A and B to the inputs
  ...

endmodule // Project 0.5
```

Buffer (Pass-Through) Logical Gate

BUF

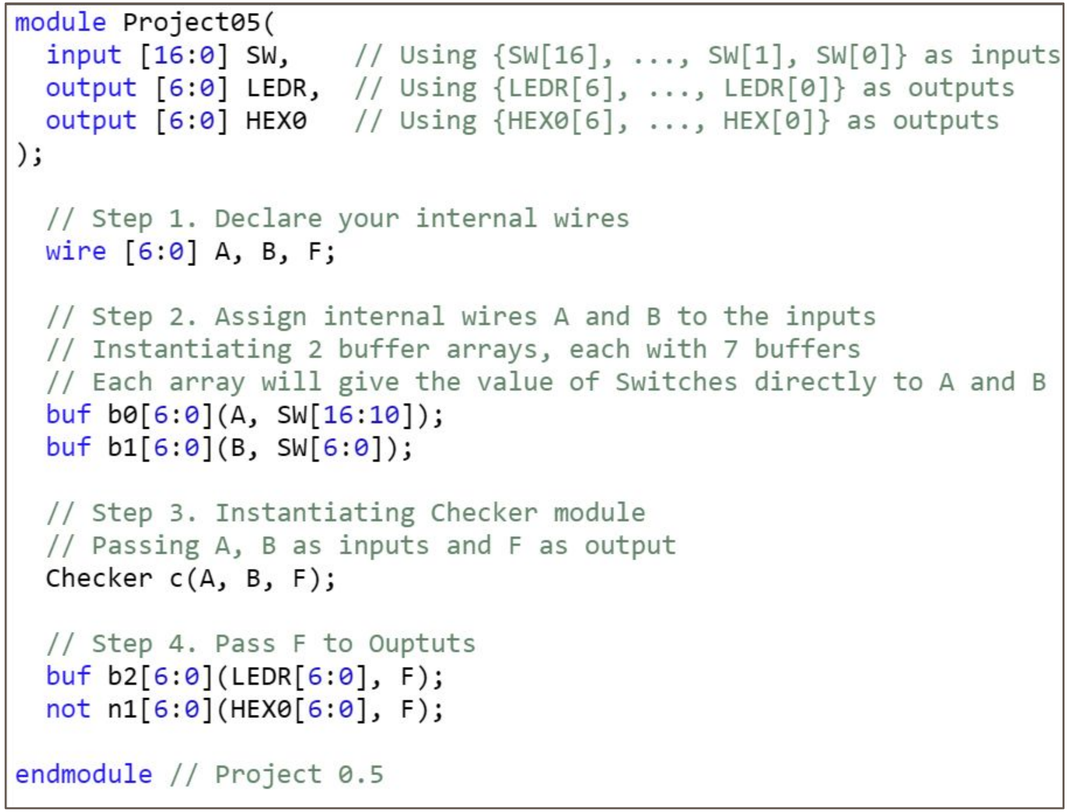


A	OUT
0	0
1	1

Pass-Through Operator

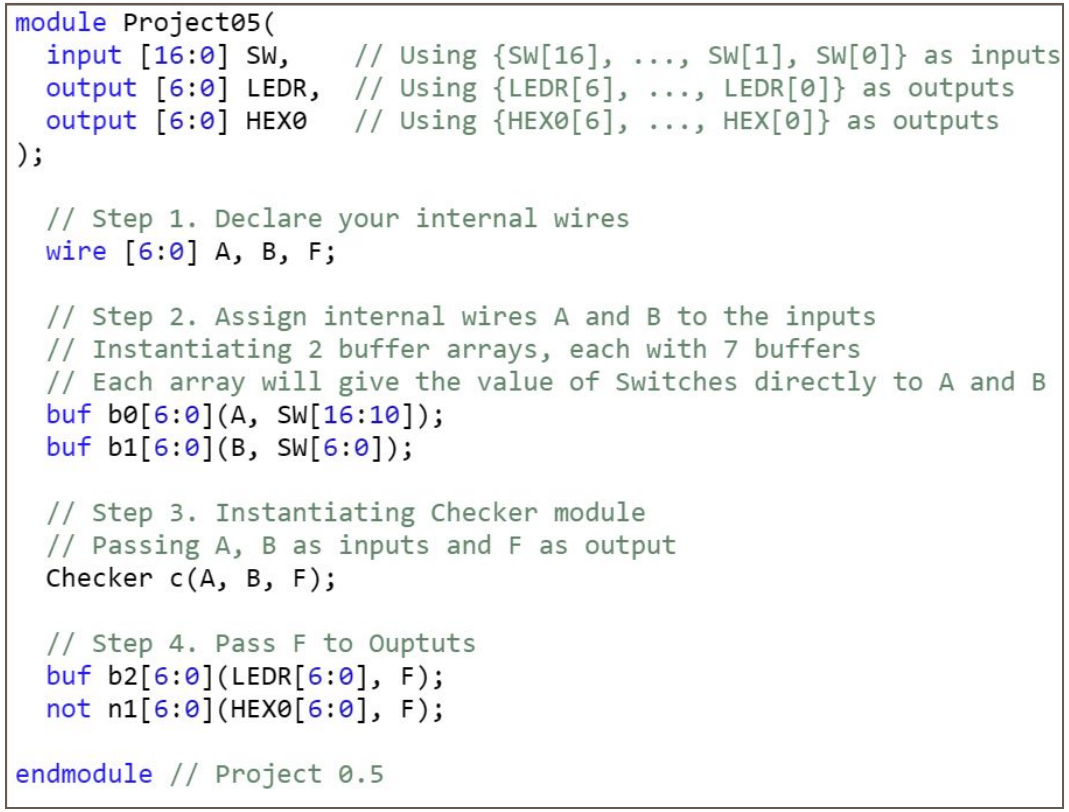
```
module Project05(
  input [16:0] SW,     // Using {SW[16], ..., SW[1], SW[0]} as inputs
  output [6:0] LEDR,   // Using {LEDR[6], ..., LEDR[0]} as outputs
  output [6:0] HEX0    // Using {HEX0[6], ..., HEX[0]} as outputs
);

  // Step 1. Declare your internal wires
  wire [6:0] A, B, F;

  // Step 2. Assign internal wires A and B to the inputs
  // Instantiating 2 buffer arrays, each with 7 buffers
  // Each array will give the value of Switches directly to A and B
  buf b0[6:0](A, SW[16:10]);
  buf b1[6:0](B, SW[6:0]);

endmodule // Project 0.5
```

Implementing Project05 Module (Finished)

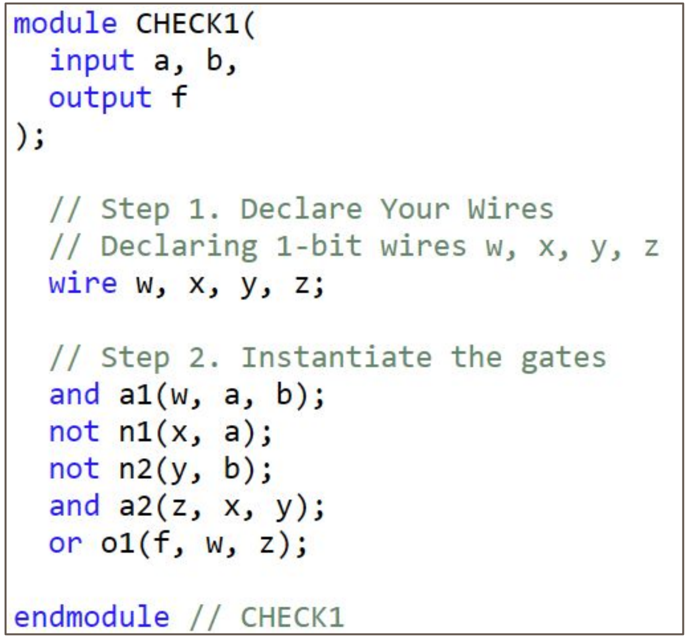


```verilog
module Project05(
  input [16:0] SW,    // Using {SW[16], ..., SW[1], SW[0]} as inputs
  output [6:0] LEDR,  // Using {LEDR[6], ..., LEDR[0]} as outputs
  output [6:0] HEX0   // Using {HEX0[6], ..., HEX[0]} as outputs
);

  // Step 1. Declare your internal wires
  wire [6:0] A, B, F;

  // Step 2. Assign internal wires A and B to the inputs
  // Instantiating 2 buffer arrays, each with 7 buffers
  // Each array will give the value of Switches directly to A and B
  buf b0[6:0](A, SW[16:10]);
  buf b1[6:0](B, SW[6:0]);

  // Step 3. Instantiating Checker module
  // Passing A, B as inputs and F as output
  Checker c(A, B, F);

  // Step 4. Pass F to Ouptuts
  buf b2[6:0](LEDR[6:0], F);
  not n1[6:0](HEX0[6:0], F);

endmodule // Project 0.5
```

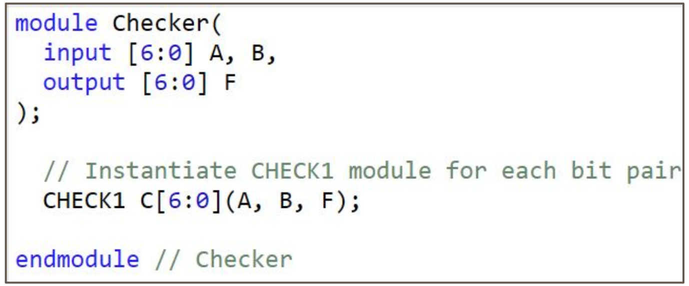
Final Verilog



```verilog
module Project05(
  input [16:0] SW,    // Using {SW[16], ..., SW[1], SW[0]} as inputs
  output [6:0] LEDR,  // Using {LEDR[6], ..., LEDR[0]} as outputs
  output [6:0] HEX0   // Using {HEX0[6], ..., HEX[0]} as outputs
);

  // Step 1. Declare your internal wires
  wire [6:0] A, B, F;

  // Step 2. Assign internal wires A and B to the inputs
  // Instantiating 2 buffer arrays, each with 7 buffers
  // Each array will give the value of Switches directly to A and B
  buf b0[6:0](A, SW[16:10]);
  buf b1[6:0](B, SW[6:0]);

  // Step 3. Instantiating Checker module
  // Passing A, B as inputs and F as output
  Checker c(A, B, F);

  // Step 4. Pass F to Ouptuts
  buf b2[6:0](LEDR[6:0], F);
  not n1[6:0](HEX0[6:0], F);

endmodule // Project 0.5
```



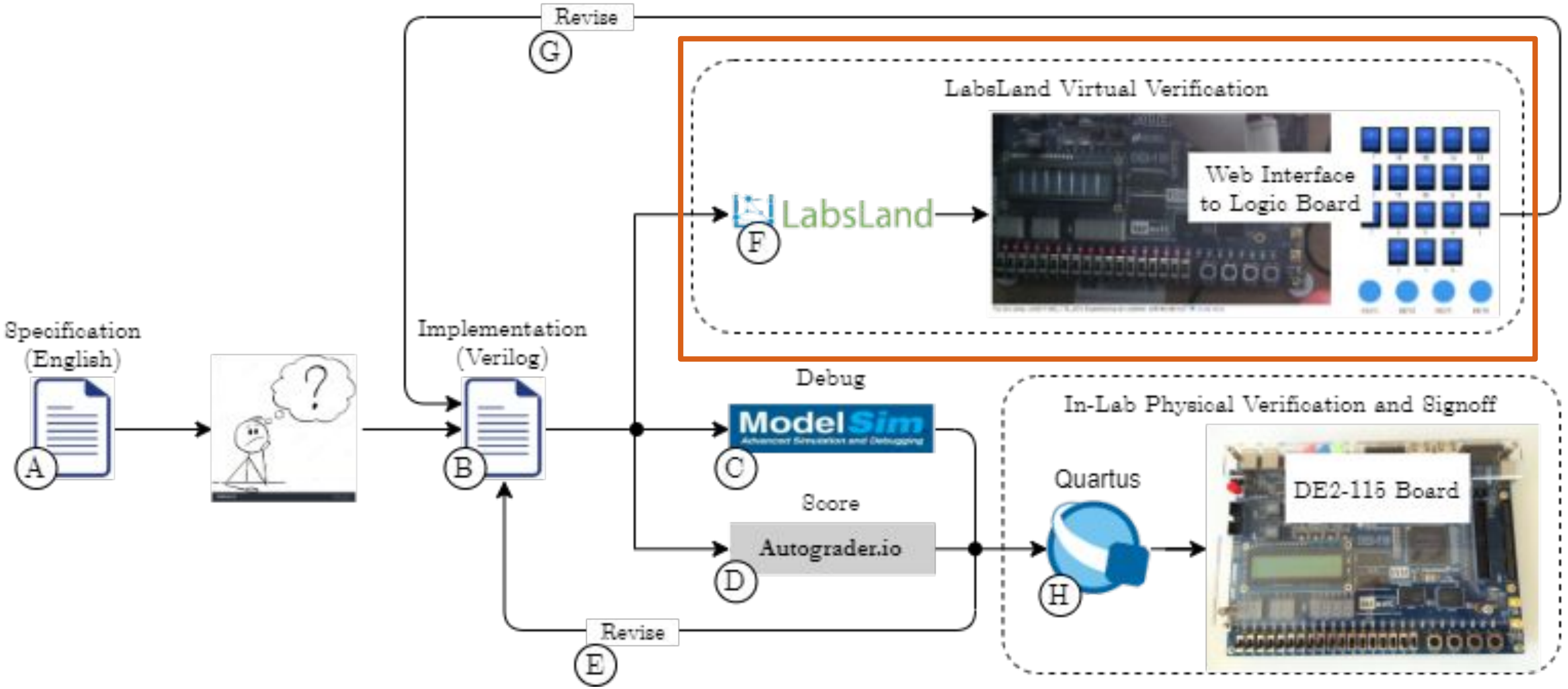
```verilog
module CHECK1(
  input a, b,
  output f
);

  // Step 1. Declare Your Wires
  // Declaring 1-bit wires w, x, y, z
  wire w, x, y, z;

  // Step 2. Instantiate the gates
  and a1(w, a, b);
  not n1(x, a);
  not n2(y, b);
  and a2(z, x, y);
  or o1(f, w, z);

endmodule // CHECK1
```



```verilog
module Checker(
  input [6:0] A, B,
  output [6:0] F
);

  // Instantiate CHECK1 module for each bit pair
  CHECK1 C[6:0](A, B, F);

endmodule // Checker
```

Let’s Try It Out (LabsLand)

Overview of the Tools



LabsLand - Virtually Connection to the Boards

Link to LabsLand Site

- Ability to connect to an FPGA board virtually!

1. Create a new account (Possibly)
2. Access the DE2-115 IDE Verilog Option (Not VHDL)

- WARNING: There is a per day capacity on how many times you can flash onto the board (Not sure how many times)

Textual Solution to Project 0.5 (For Copy-Paste)

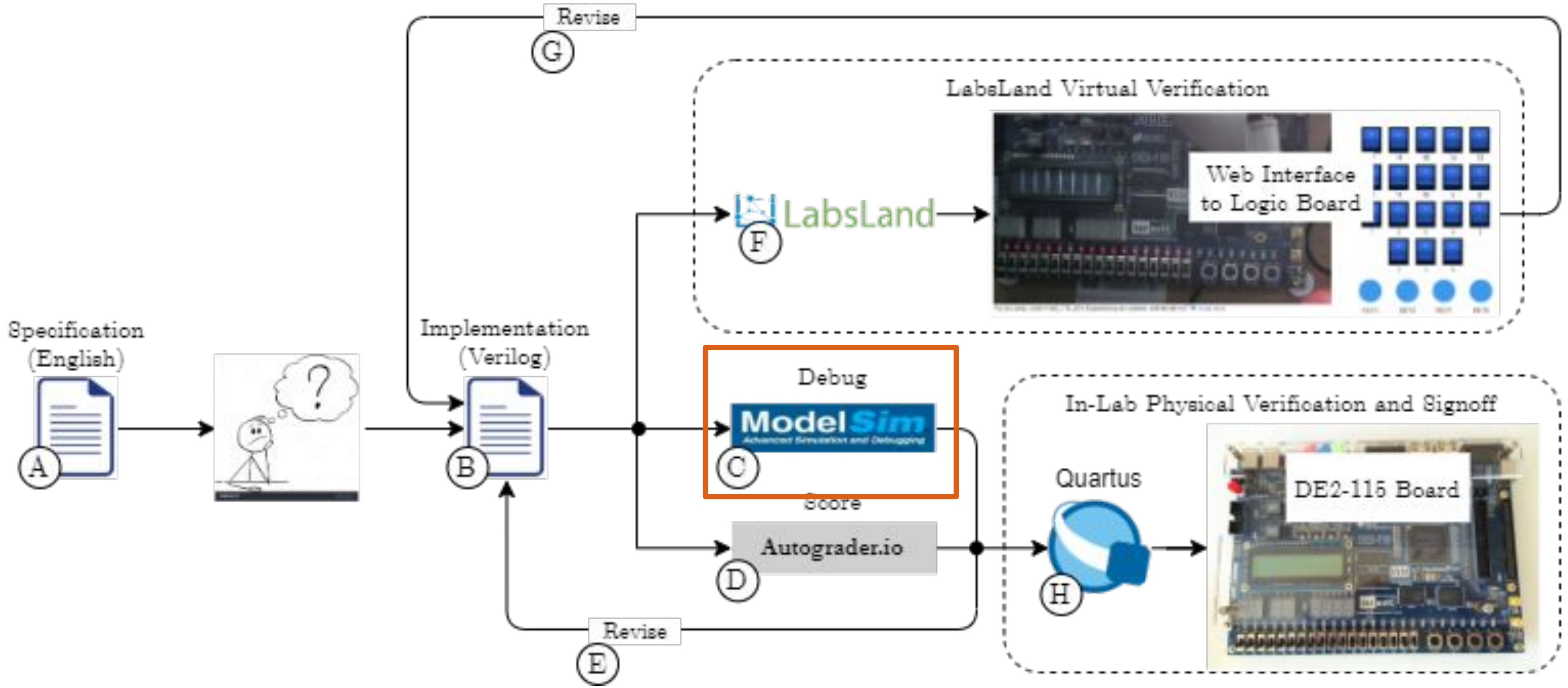
```
module Project05(
  input [16:0] SW,    // Using {SW[16], ..., SW[1], SW[0]} as inputs
  output [6:0] LEDR,  // Using {LEDR[6], ..., LEDR[0]} as outputs
  output [6:0] HEX0   // Using {HEX0[6], ..., HEX[0]} as outputs
);

  // Step 1. Declare your internal wires
  wire [6:0] A, B, F;

  // Step 2. Assign internal wires A and B to the inputs
  // Instantiating 2 buffer arrays, each with 7 buffers
  // Each array will give the value of Switches directly to A and B
  buf b0[6:0](A, SW[16:10]);
  buf b1[6:0](B, SW[6:0]);

  // Step 3. Instantiating Checker module
  // Passing A, B as inputs and F as output
  Checker c(A, B, F);

  // Step 4. Pass F to Outputs
  buf b2[6:0](LEDR[6:0], F);
  not n1[6:0](HEX0[6:0], F);

endmodule // Project 0.5
```

```
module Checker(
  input [6:0] A, B,
  output [6:0] F
);

  // Instantiate CHECK1 module for each bit pair
  CHECK1 C[6:0](A, B, F);

endmodule // Checker
```

```
module CHECK1(
  input a, b,
  output f
);

  // Step 1. Declare Your Wires
  // Declaring 1-bit wires w, x, y, z
  wire w, x, y, z;

  // Step 2. Instantiate the gates
  and a1(w, a, b);
  not n1(x, a);
  not n2(y, b);
  and a2(z, x, y);
  or o1(f, w, z);

endmodule // CHECK1
```

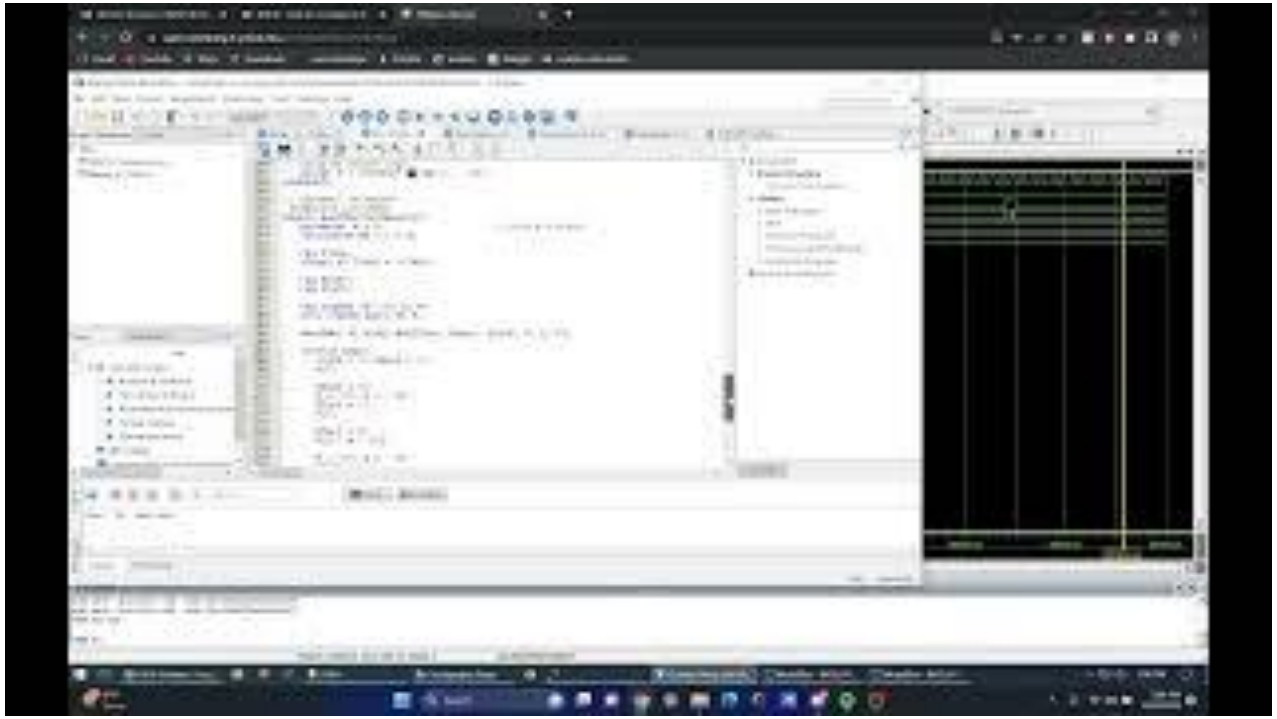
Let’s Simulate (ModelSim)

Overview of the Tools

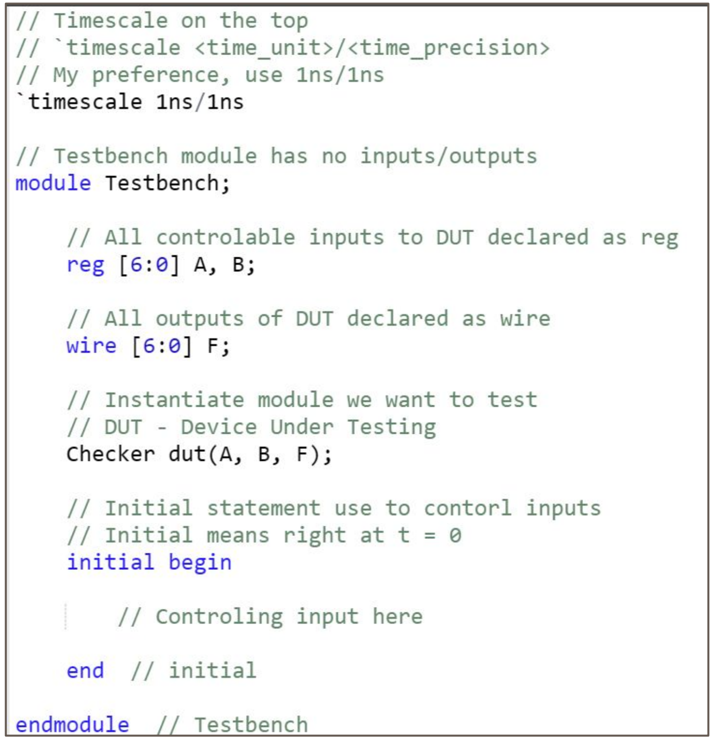

Revise
G
LabsLand Virtual Verification
Web Interface to Logic Board
LabsLand
F
Specification (English)
A
Implementation (Verilog)
B
Debug
ModelSim
Advanced Simulation and Debugging
C
Score
Autograder.io
D
In-Lab Physical Verification and Signoff
Quartus
H
DE2-115 Board
Revise
E

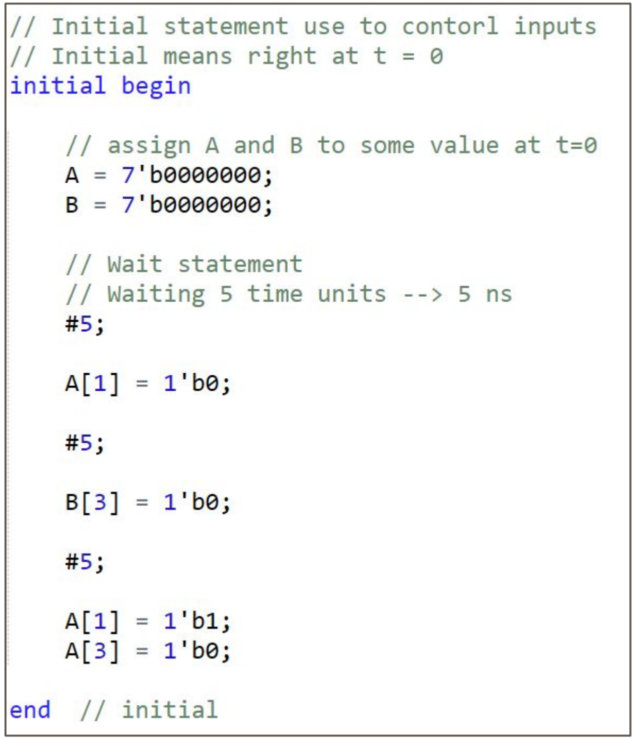
ModelSim - Debugging and Simulating Testbenches

- Great ModelSim tutorial created by previous GSI Ryan!



General Structure of a Testbench (Overview)


```
// Timescale on the top
// `timescale <time_unit>/<time_precision>
// My preference, use 1ns/1ns
`timescale 1ns/1ns

// Testbench module has no inputs/outputs
module Testbench;

    // All controlable inputs to DUT declared as reg
    reg [6:0] A, B;

    // All outputs of DUT declared as wire
    wire [6:0] F;

    // Instantiate module we want to test
    // DUT - Device Under Testing
    Checker dut(A, B, F);

    // Initial statement use to contorl inputs
    // Initial means right at t = 0
    initial begin

        // Controling input here

    end  // initial

endmodule  // Testbench
```


```
// Initial statement use to contorl inputs
// Initial means right at t = 0
initial begin

    // assign A and B to some value at t=0
    A = 7'b0000000;
    B = 7'b0000000;

    // Wait statement
    // Waiting 5 time units --> 5 ns
    #5;

    A[1] = 1'b0;

    #5;

    B[3] = 1'b0;

    #5;

    A[1] = 1'b1;
    A[3] = 1'b0;

end  // initial
```

Number Specification in Verilog

- You can specify sized number in Verilog through {size}’{base format}{number}
  - Size - Number of bits associated with the number
  - Base Format - Can be decimal (D or d), Binary (B or b), Hex (H or h), Octal (O or o)
  - Number - Number specified in the provided base format

Ex. 4’b1001 → **4-bit** number with **binary** value **1001** (9 in decimal)

Ex. 5’d13 → **5-bit** number with **decimal** value **13** (**01101** in binary)

Ex. 8’hAA → **8-bit** number with **hex** value **AA** (**1010_1010** in binary)

- Valid to use underscores anywhere for readability → 12’b0110_1001_0011
- Valid to leave number unsized (**BUT NOT ADVISED**) → ‘b1001, ‘d32
- If no size or base format specified, default is decimal → 123

Textual Testbench for Project 0.5

```
// Timescale on the top
// `timescale <time_unit>/<time_precision>
// My preference, use 1ns/1ns
`timescale 1ns/1ns

// Testbench module has no inputs/outputs
module Testbench;

  // All controlable inputs to DUT declared as reg
  reg [6:0] A, B;

  // All outputs of DUT declared as wire
  wire [6:0] F;

  // Instantiate module we want to test
  // DUT - Device Under Testing
  Checker dut(A, B, F);

  // Initial statement use to contorl inputs
  // Initial means right at t = 0
  initial begin

    // assign A and B to some value at t=0
    A = 7'b1111111;
    B = 7'b1111111;

    // Wait statement
    // Waiting 5 time units --> 5 ns
    #5;

    A[1] = 1'b0;

    #5;

    B[3] = 1'b0;

    #5;

    A[1] = 1'b1;
    A[3] = 1'b0;

  end  // initial

endmodule  // Testbench
```