Verilog HW#1: 4-bit Two's Complement

Chun-Jen Tsai
National Chiao Tung University

4/1/2020

Verilog HW#1

- Goal: Design a Verilog module that computes two’s complement of an input 4-bit binary number using gate-level operators: ^ and |.

- Deadline: 4/13, 23:55pm. You must upload your Verilog module to the E3 website by the deadline.

- In the following slides, we will show you how to test a Verilog module through Verilog simulation.

Simulation of a Digital Design

- We can use a PC to simulate the operation of a digital design. For that, we need two programs:
  - A compiler that can compile an Verilog program to a simulation model (executable). Executing the simulation model can produce a dump file that contains all signals transitions during the execution process.
  - A waveform viewer that can show graphically the signal diagrams in the dump file.

- For this course, we use the Icarus Verilog compiler, and the Gtkwave waveform viewer for these purposes.

Icarus Verilog

- Icarus Verilog is a Verilog compiler that can simulation a Verilog program or synthesize it into a real FPGA circuit. You can get its source code and binaries for Linux from the following website:

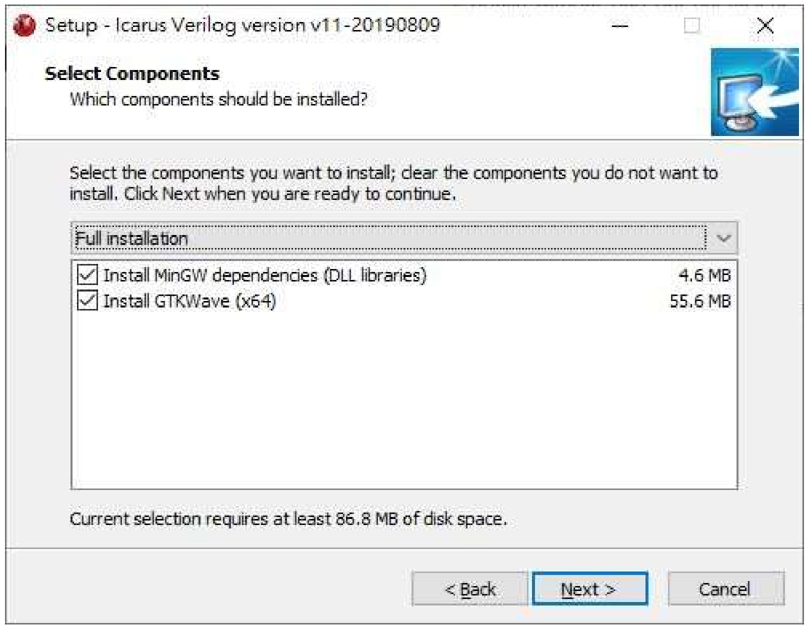
  http://iverilog.icarus.com/

- For a Windows version installer, you can download it from:

  https://bleyer.org/icarus/

Installation of Icarus & Gtkwave

- ❑ Under Windows, just download and run the installation program, `iverilog-v11-20190809-x64_setup.exe`, and it will install both Icarus and Gtkwave:



Setup - Icarus Verilog version v11-20190809

Select Components

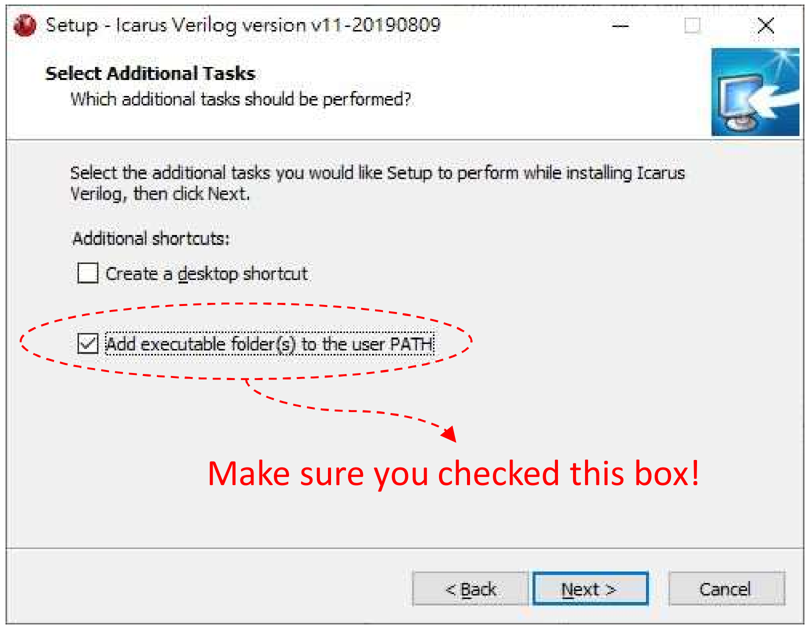
Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

Full installation

☑	Install MinGW dependencies (DLL libraries)	4.6 MB
☑	Install GTKWave (x64)	55.6 MB

Current selection requires at least 86.8 MB of disk space.



Setup - Icarus Verilog version v11-20190809

Select Additional Tasks

Which additional tasks should be performed?

Select the additional tasks you would like Setup to perform while installing Icarus Verilog, then click Next.

Additional shortcuts:

☐ Create a desktop shortcut

☑ Add executable folder(s) to the user PATH

Make sure you checked this box!

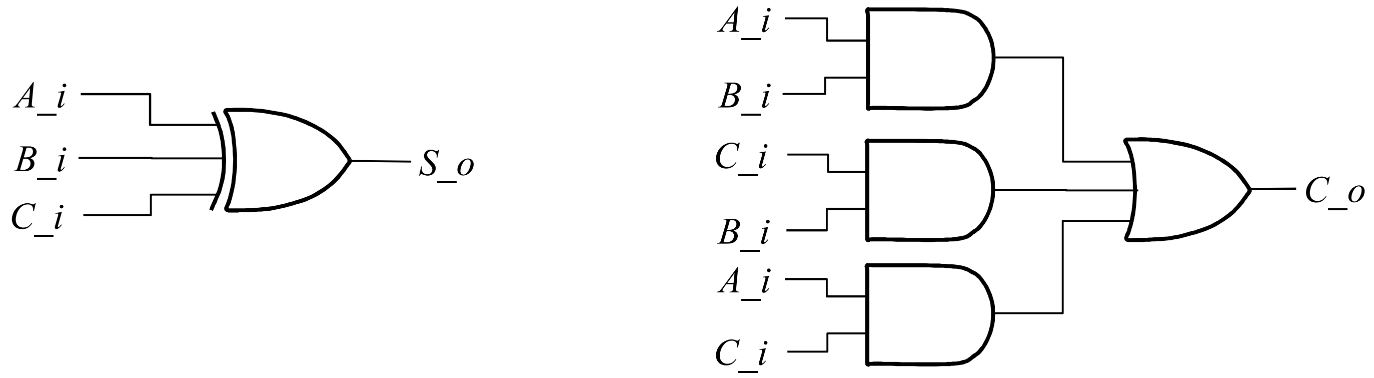
Simulation of a Verilog Program

- Icarus Verilog is a command-line program. You must run it under the Command Prompt (Windows or Linux).
- Let us try to simulate a 1-bit full adder. A 1-bit full adder implements the following truth table:

A_i	*B_i*	*C_i*	*C_o*	*S_o*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Verilog Program of the Full Adder

- A gate-level design of the 1-bit full adder is as follows:
  - The logic diagram of the 1-bit full adder:



  - The Verilog module:

```
module fa_1bit(
  input A_i, input B_i, input C_i,
  output S_o, output C_o);

  assign S_o = C_i ^ A_i ^ B_i;
  assign C_o = (A_i & B_i) | (C_i & B_i) | (C_i & A_i);
endmodule
```

Testbench Program (1/2)

- In order to test the execution of the full adder module, we must design a testbench module to provide input signals and output verification for the full adder
- The testbench is only for simulation, not for synthesis

```
module full_adder_tb;

// Input signals
reg A, B;
reg Cin;          // Carry in

// Output signals
wire Sum;
wire Cout;        // Carry out

// Instantiate the Unit Under Test (UUT)
fa_1bit uut (.A_i(A), .B_i(B), .C_i(Cin), .S_o(Sum), .C_o(Cout));
```

Testbench Program (2/2)

- The main part of the testbench module is composed of an initial-block that sets input to the full adder module:

```
initial begin
    // Setup the signal dump file.
    $dumpfile("full_adder.vcd"); // Set the name of the dump file.
    $dumpvars;                   // Save all signals to the dump file.

    // Display the signal values whenever there is a change.
    $monitor("A=%b, B=%b Cin=%b | Cout=%b Sum=%b",
                                        A, B, Cin, Cout, Sum);

    // Initialize the input signals
    A = 0; B = 0; Cin = 0;

    // Change input signals; wait 20 time units for each change
    #20; A = 1'b0; B = 1'b1; Cin = 1’b0;
    #20; A = 1'b1; B = 1'b0; Cin = 1’b1;
    #20; A = 1'b1; B = 1'b1; Cin = 1'b1;
    #20; $finish; // End the simulation.
end
endmodule
```

Compile and Execute the Simulation

- Download `full_adder.v` from E3, and use the following commands to compile the program into an executable, `full_adder.vvp`, then run the simulation:

```
C:\workspace>iverilog -o full_adder.vvp full_adder.v

C:\workspace>vvp full_adder.vvp
VCD info: dumpfile full_adder.vcd opened for output.
A = 0, B = 0 Cin = 0 | Cout = 0 Sum = 0
A = 0, B = 1 Cin = 0 | Cout = 0 Sum = 1
A = 1, B = 0 Cin = 1 | Cout = 1 Sum = 0
A = 1, B = 1 Cin = 1 | Cout = 1 Sum = 1

C:\workspace>gtkwave full_adder.vcd

GTKWave Analyzer v3.3.100 (w)1999-2019 BSI

[0] start time.
[60000] end time.
```

compile

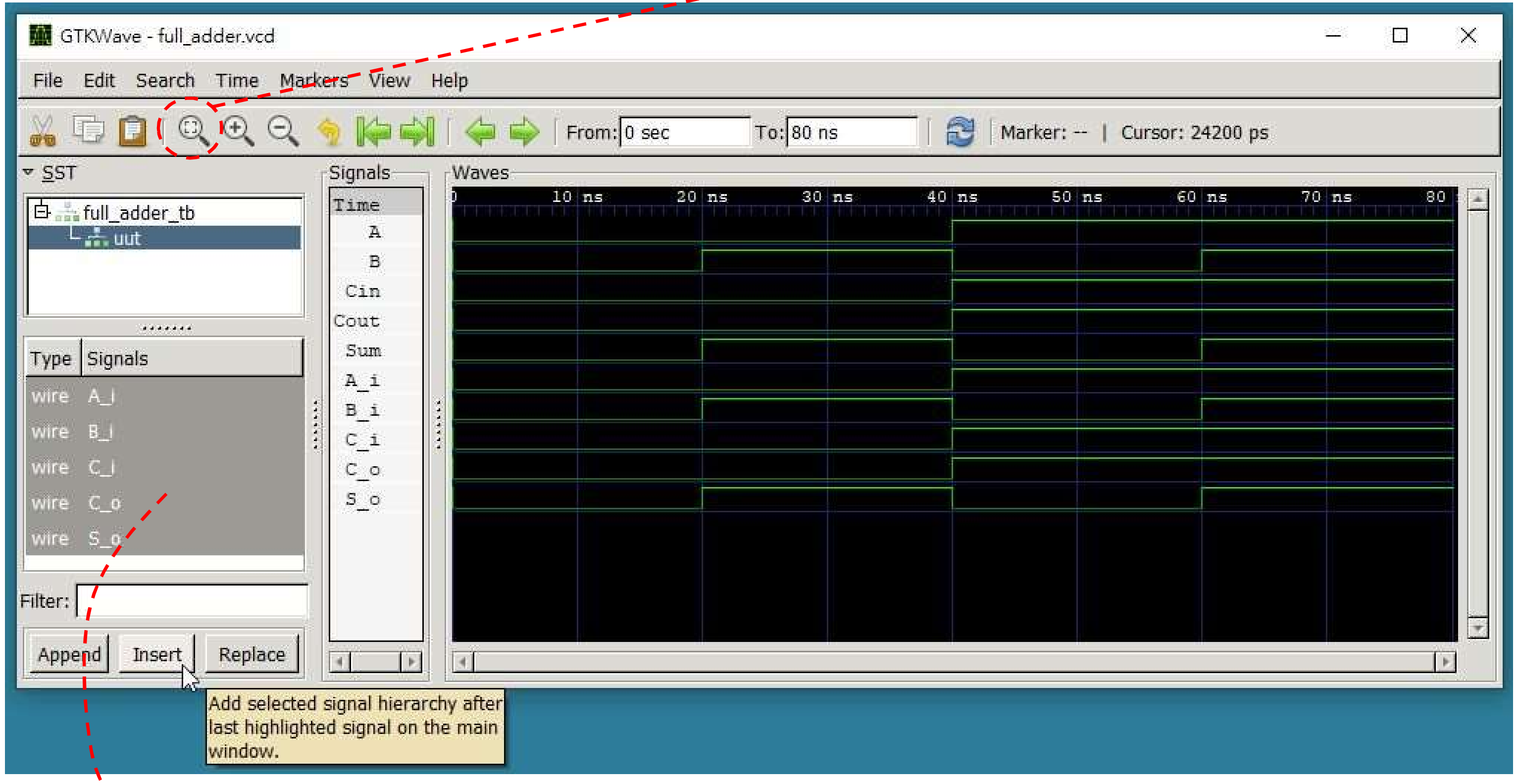
run simulation

show wavefroms in the dump file.

Using GTKWave to Check Wavefroms

- GTKWave can be used to check the correctness of all binary signals in your design:

Click the zoom-fit button to see the entire time-line.



Add selected signal hierarchy after last highlighted signal on the main window.

Select and insert all the signals to the watch window.

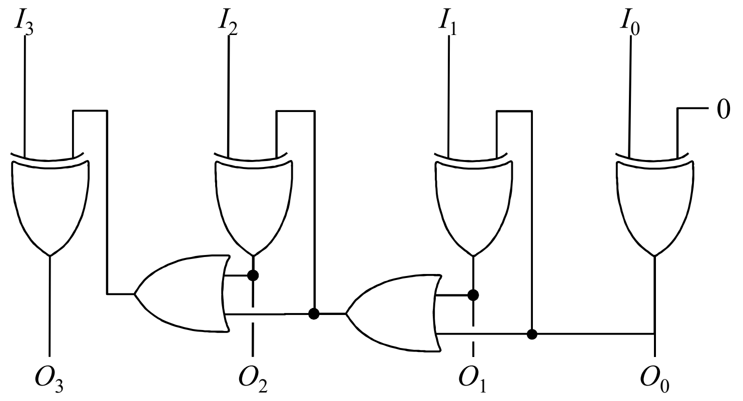
Code Coverage of a Testbench

- Typically, you should try enough combinations of the input signals in the initial-block of the testbench module to make sure that the module under test is correct.

- Try modifying `full_adder.v` to play with Icarus Verilog and GTKWave.

Requirements for Verilog HW#1 (1/2)

- Please design a module that implements the following 4-bit Two's Complement logic diagram:



The logic diagram of a 4-bit 2's complement circuit. I_0*~*I_3 *are inputs,* O_0 *~* O_3 *are outputs.*

Note: you must write an exact Verilog gate-level design of the above logic diagram.

Requirements for Verilog HW#1 (2/2)

- The module must be declared as follows:

```
module comp2_4bit(input [3:0] I, output [3:0] O);

  /* Implement your design here. */

endmodule
```

- Do not upload your testbench module, just upload the `comp2_4bit()` module to E3 (and name the file `comp2_4bit.v`). The TA's will use their own testbench to test your module.