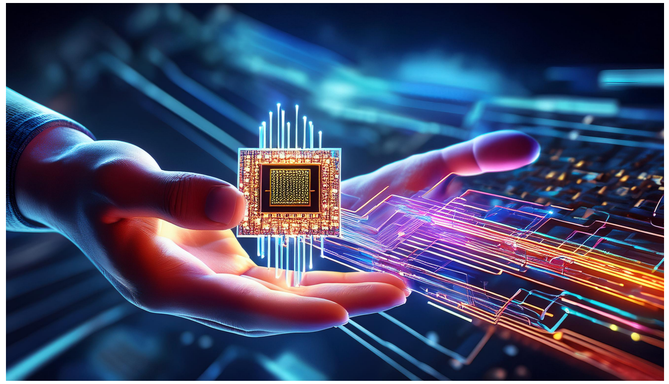
1/5/26 09:54

CSCI 250
Introduction to Computer Organisation
Lecture 1: Beyond Integer Arithmetics III



Jetic Gū

Overview

- Focus: Course Introduction
- Architecture: Logical Circuits
- Textbook: LW Chapter 7
- Core Ideas:
  1. VHDL, Binary Adder
  2. Lab 1 Part 2: Adder-Subtractor

Verilog Hardware Description Language

What is HDL

- Programming Languages: e.g. Python, C, C++
  - Compiles/Interprets to machine code
  - Executed sequentially by a CPU
- Hardware Description Language: VHDL, Verilog
  - Describes hardware logic, how gates are connected
  - Loaded onto FPGA board, fully parallel (because it’s a real circuit)

Concept

HDL IDE Platforms

- AMD Xilinx
  - [Chipset] Spartan 6-: ISE Suite
  - [Chipset] Spartan 7+: Vivado
- Intel Altera FPGA: Quartus Prime
- This is the industry standard, not as easy to get into

Previous: Register Transfer Operations (VHDL Syntax)

	Operator	Example
Assignment	`<=`	`ax <= 12h`
Reg. Transfer	`<=`	`ax <= bx`
Addition	`+`	`ax + bx`
Subtraction	`-`	`ax - bx`
Shift Left	`sll`	`ax sll 2`
Shift Right	`srl`	`ax srl 2`

	Operator	Example
Bitwise AND	`and`	`ax and bx`
Bitwise OR	`or`	`ax or bx`
Bitwise NOT	`not`	`not ax`
Bitwise XOR	`xor`	`ax xor bx`
Vectors	`ax(3 down to 0)`	`ax(3 down to 0)`
Concatenate	`&`	`ax(7 down to 4)` `&ax(3 down to 0)`

Review

Register Transfer Operations (Verilog Syntax)

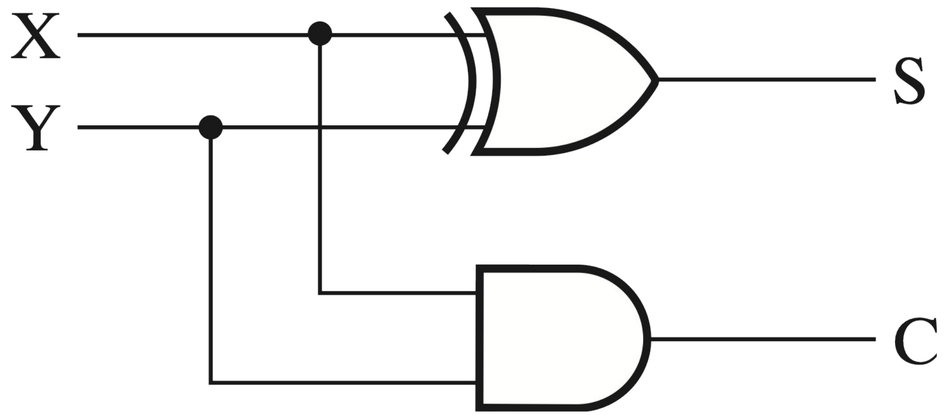
	Operator	Example
Assignment	=	`wire [7:0] x = 8'b10101010;`
Reg. Transfer (cont' assign')	assign =	`assign out = a & b;`
Addition	+	`a + b`
Subtraction	-	`a - b`
Shift Left	<<	`a << 3`
Shift Right	>>	`b >> 2`

	Operator	Example
Bitwise AND	&	`a & b`
Bitwise OR	|	`a | b`
Bitwise NOT	~	`~b`
Bitwise XOR	^	`a ^ b`
Vectors	[3:0]	`a[3:0]`
Concatenate	{a, b}	`c = {a, b}`

Concept

Previous: 16bit Adder

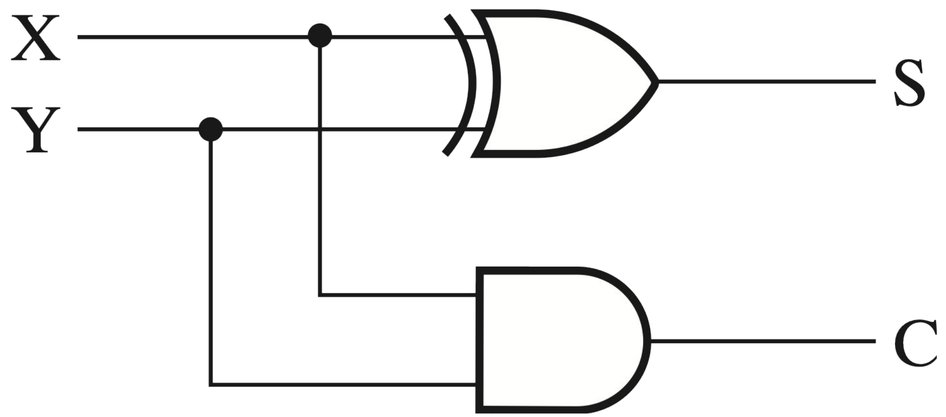
- Create a new component in VHDL called `HalfAdder1`
  - Input: X, Y
  - Output: S, C
  - Don’t use `AFTER`



Practice

Previous: 1-bit Half Adder

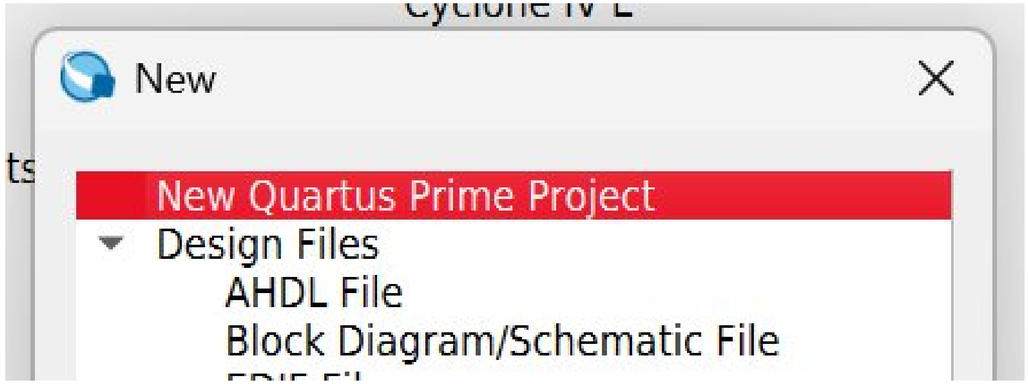
```
architecture arch1 of HalfAdder is

begin

     S <= X XOR Y;

     C <= X AND Y;

end arch1;
```

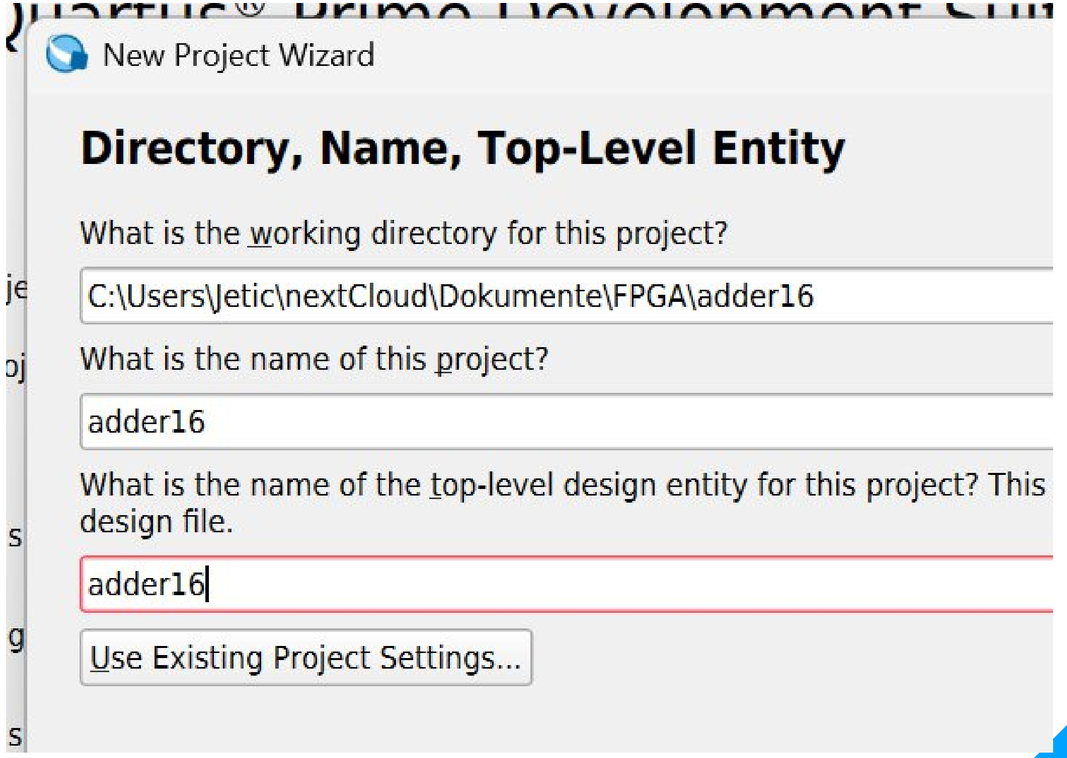


Practice

P1
Verilog

1bit Binary Adder



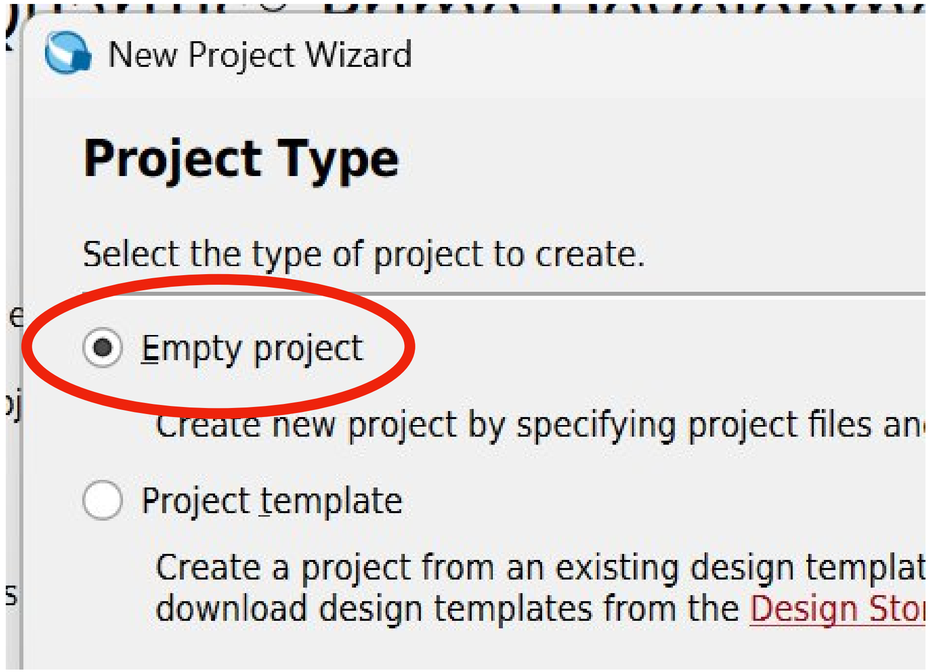


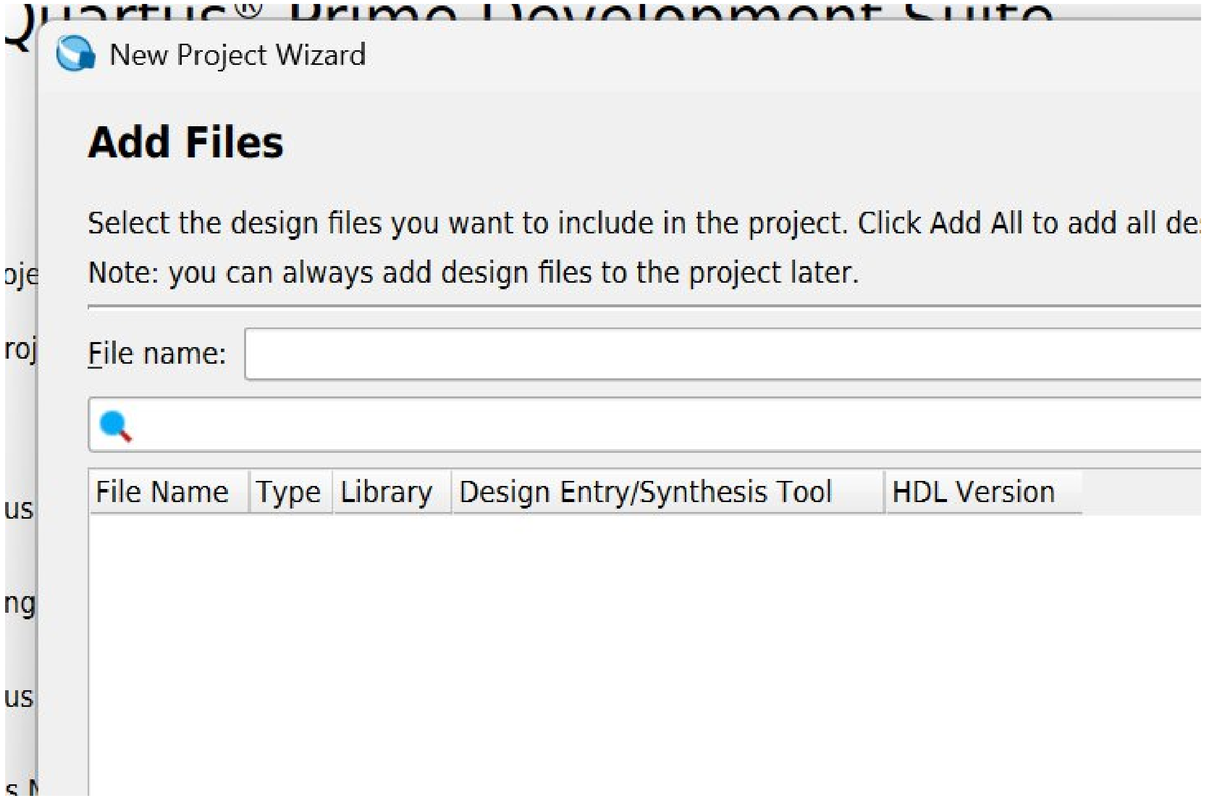
Working directory must be **local**, cannot be OneDrive or a network directory
Recommendation: **no space** in working directory
Project name should begin with **a letter, no space**

1. Select `File->New`

Tutorial 1

1bit Binary Adder



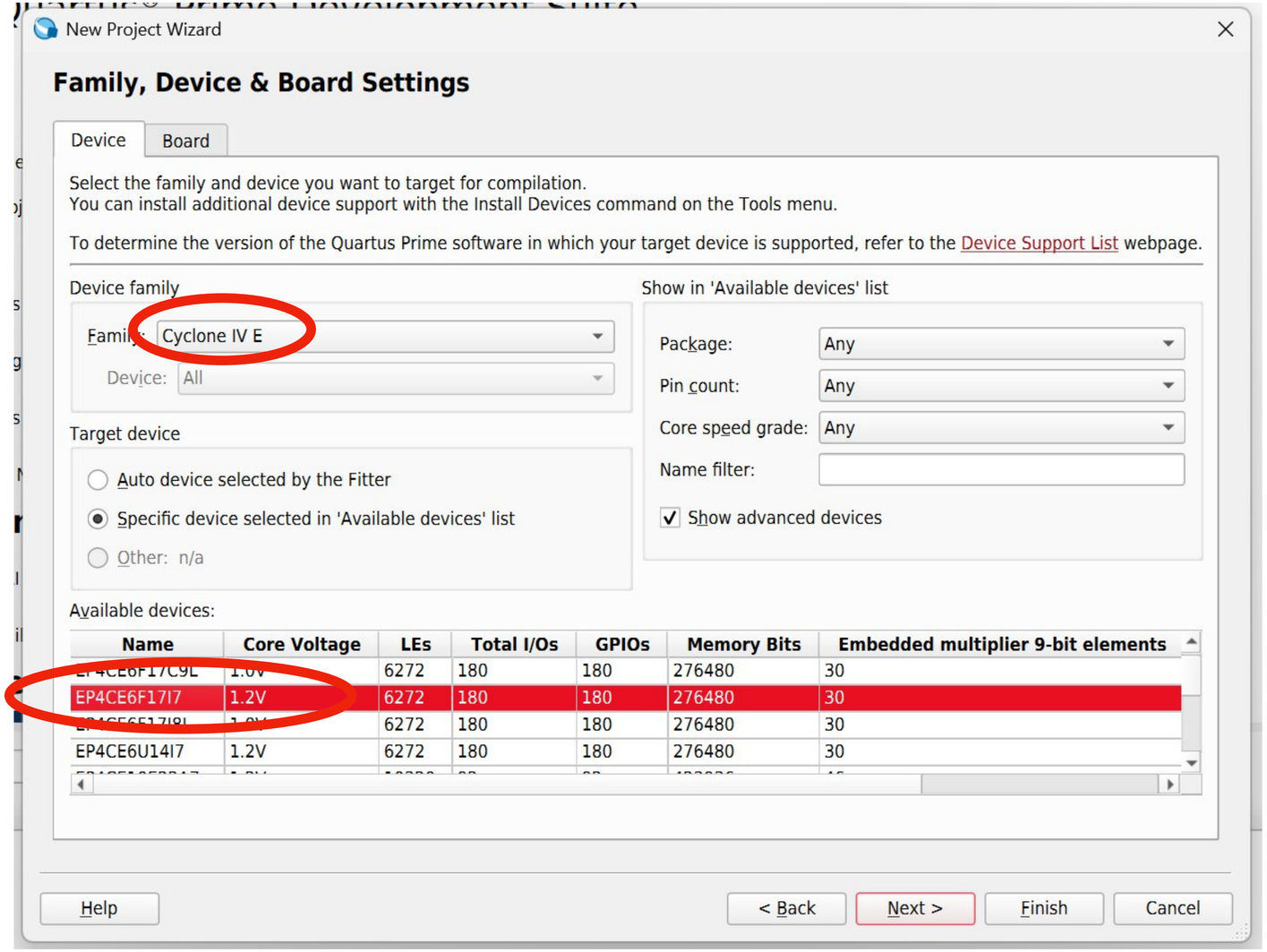


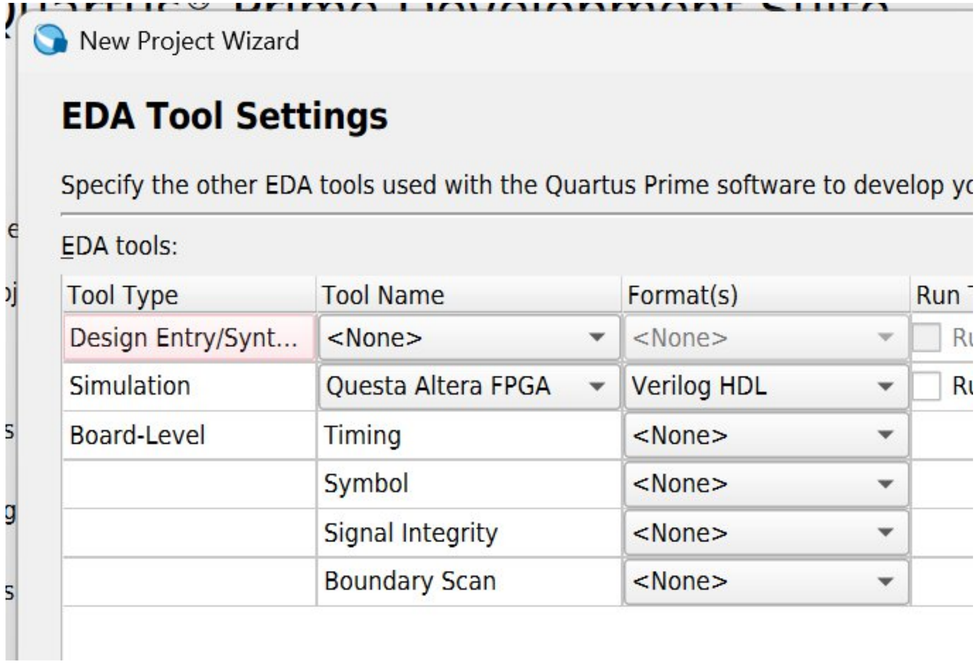
2. Select `Empty Project`, select `Empty Project`

Tutorial 1

P1
Verilog

1bit Binary Adder



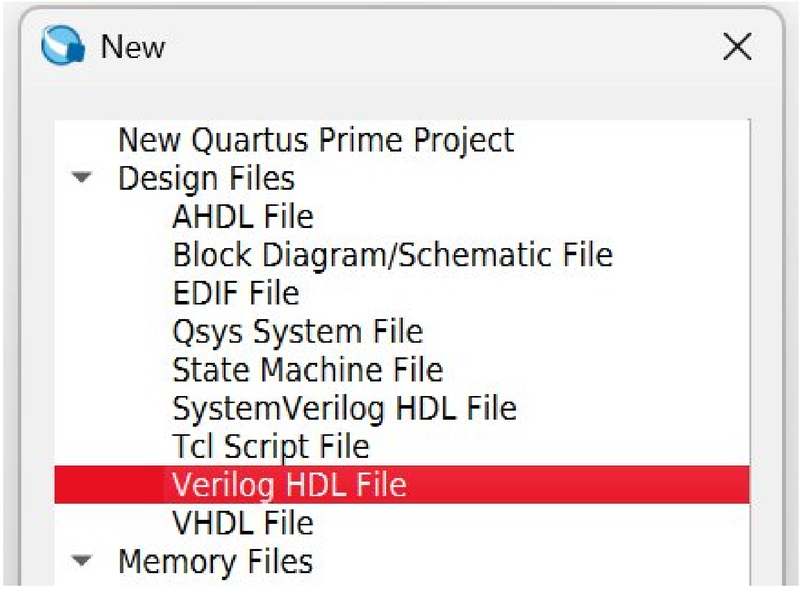


3. I will use `Cyclone IV E` family, and device `EP4CE6F17I7`, as this is the board I have. For the class, this isn't important, any one should be OK

Tutorial 1

P1
Verilog

1bit Binary Adder




```
module adder16(
    input  [15:0] x,
    input  [15:0] y,
    input         z,
    output [15:0] s,
    output        c
);
    assign {c, s} = x + y + z;
endmodule
```

4. Go to `Files -> New` for a `Verilog HDL File`

Tutorial 1

1bit Binary Adder

```
module adder16(
    input  [15:0] x,
    input  [15:0] y,
    input         z,
    output [15:0] s,
    output        c
);
    assign {c, s} = x + y + z;
endmodule
```

- In Verilog, the syntax is very similar to C/C++
- Every `module` (component) starts with the name and its interface definition

5. This is what your Verilog code looks like

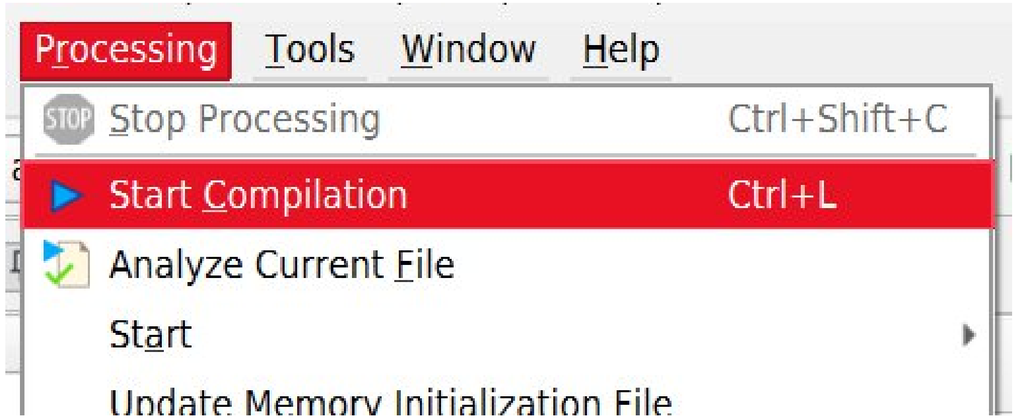
1bit Binary Adder

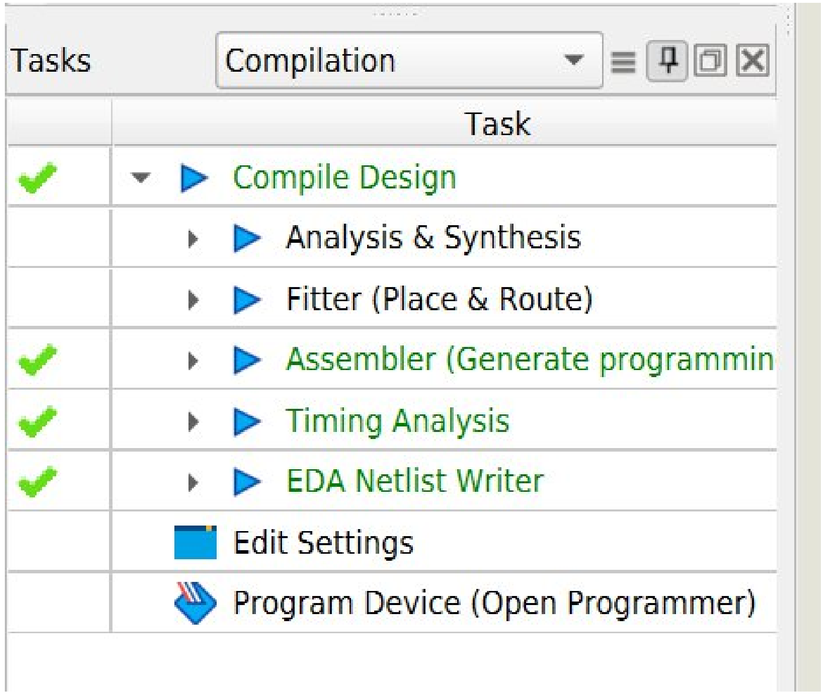
```
module adder16(
    input  [15:0] x,
    input  [15:0] y,
    input         z,
    output [15:0] s,
    output        c
);
    assign {c, s} = x + y + z;
endmodule
```

- This is continuous assignment
- The stuff you write this way is part of a combinational logic circuit, and will be executed in full parallel

5. This is what your Verilog code looks like

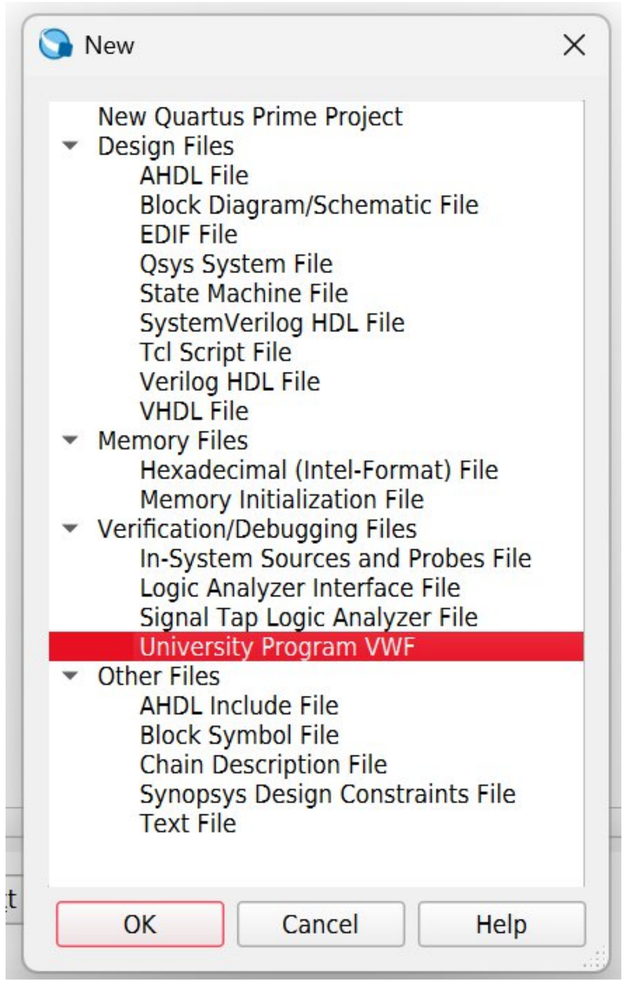
1bit Binary Adder

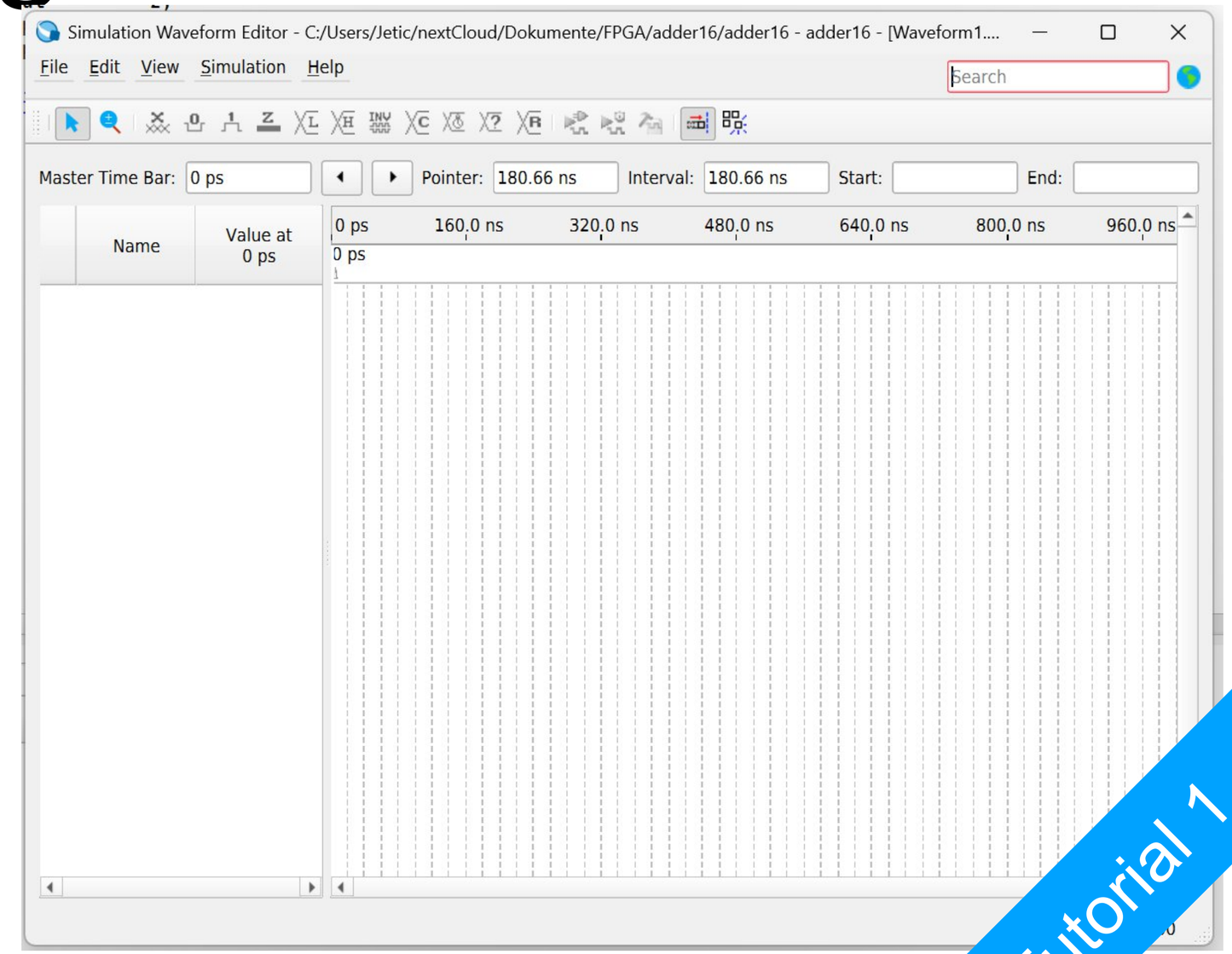




6. You will then compile, and wait for it to finish

Running Simulation

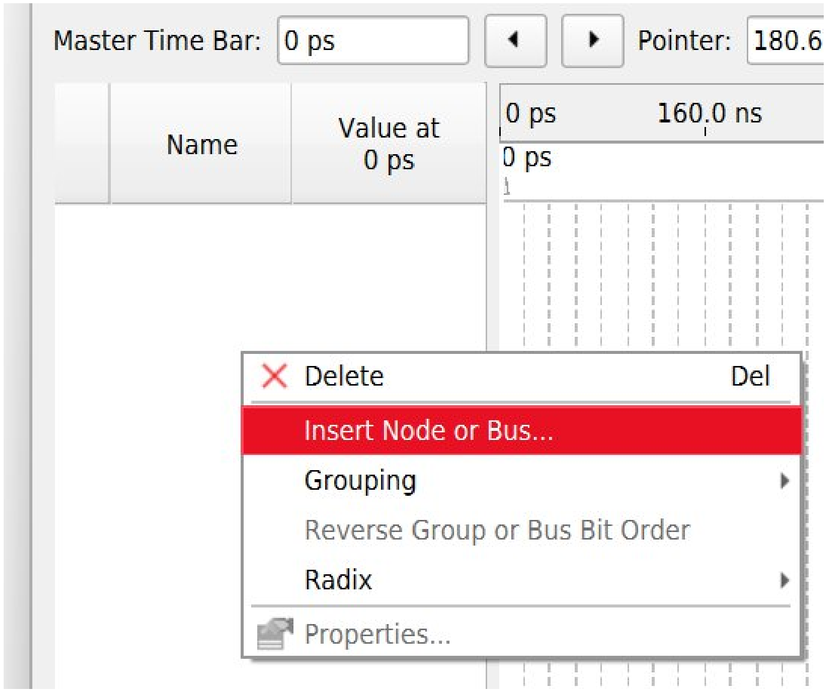


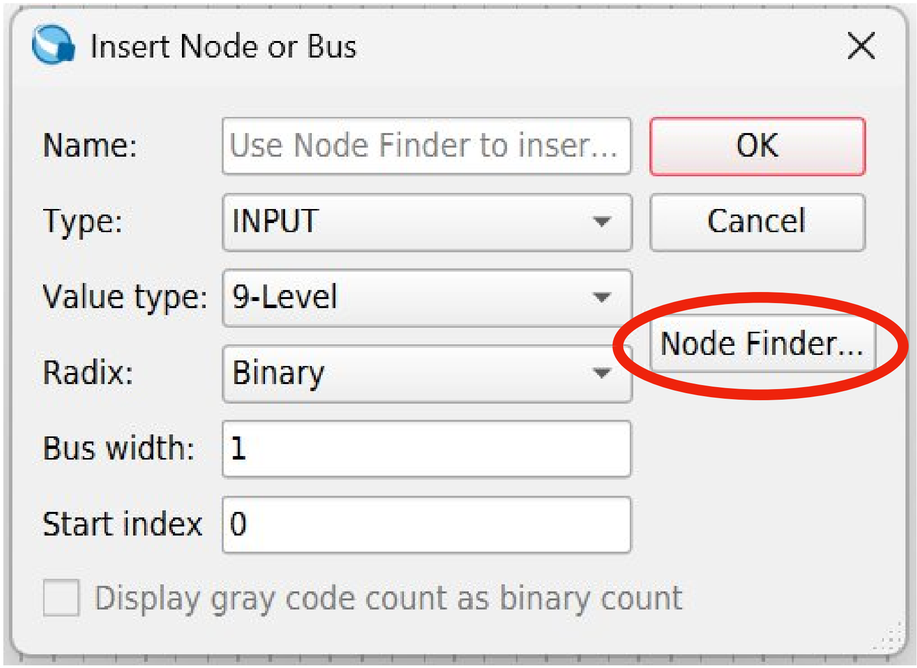


7. To run simulation, go to `Files -> New`, then select `University Program VWF`. You will then see the `Simulation Waveform Editor`

Tutorial 1

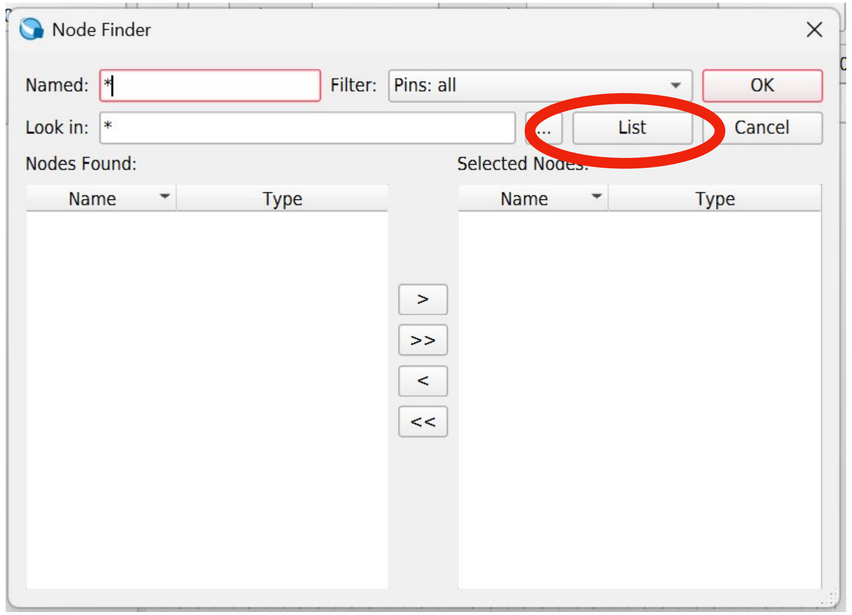
Running Simulation

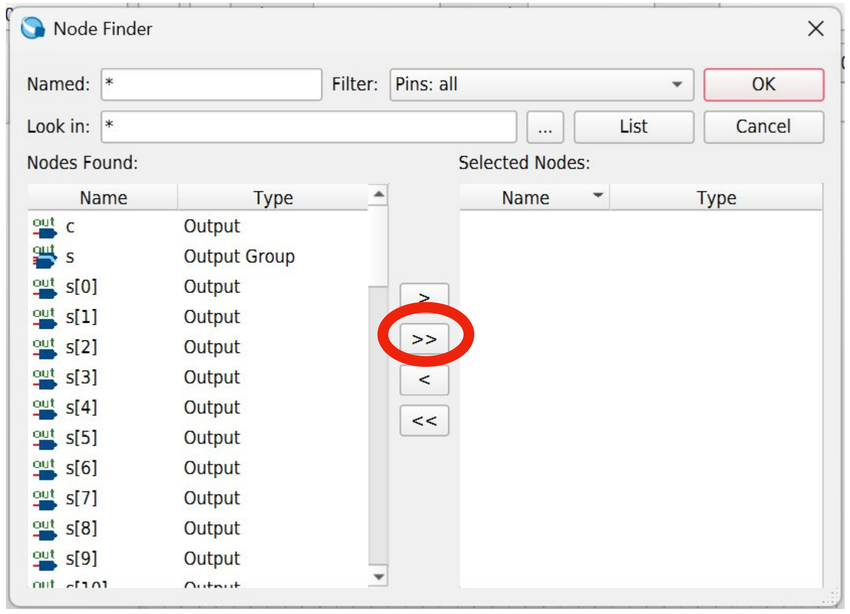


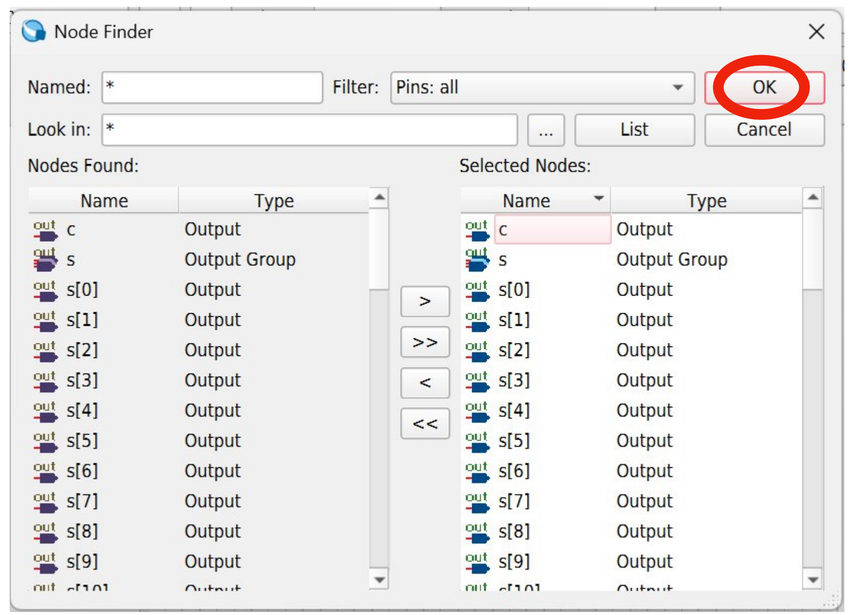


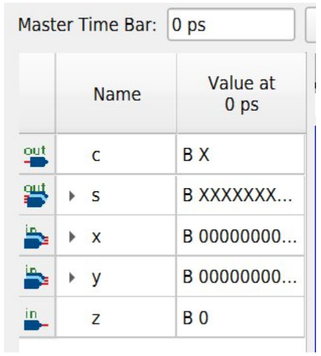
8. Right click here to add the IO, then select `Node Finder`

Running Simulation





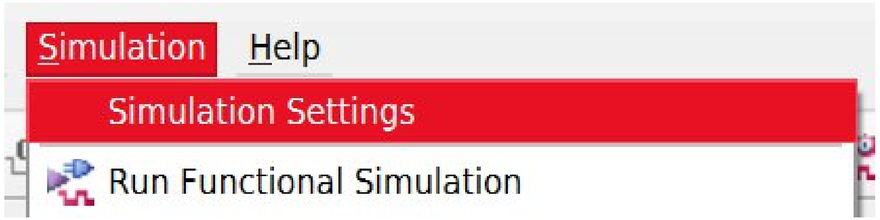




9. Click `List`, here to add the IO, then `>>`, then `OK`

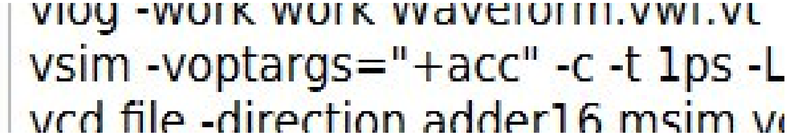
Tutorial 1

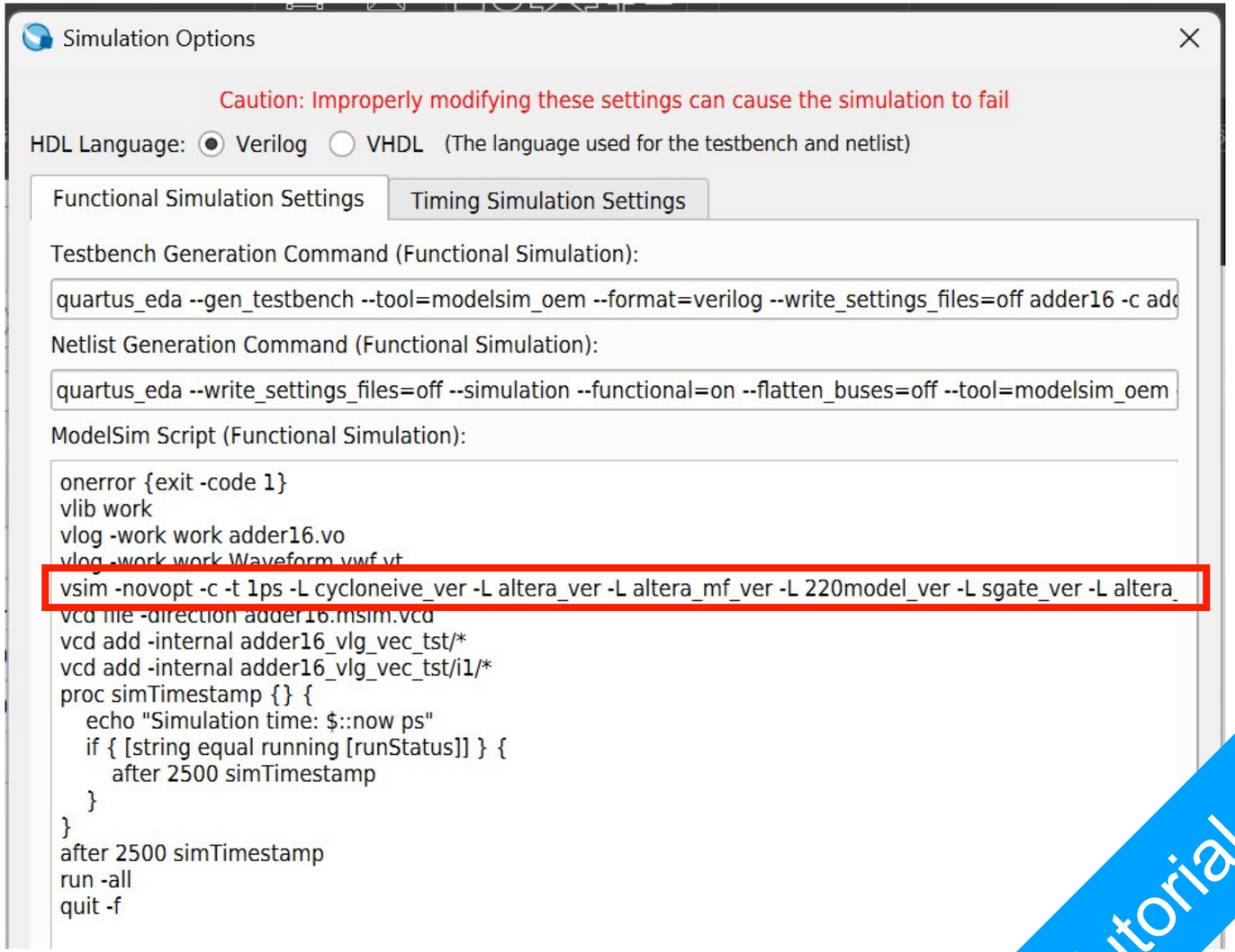
Running Simulation



On the line where it begins with `vsim`, do this:

1. after `vsim`, remove `-novopt` option
2. after `vsim`, add `-voptargs="+acc"`
3. save

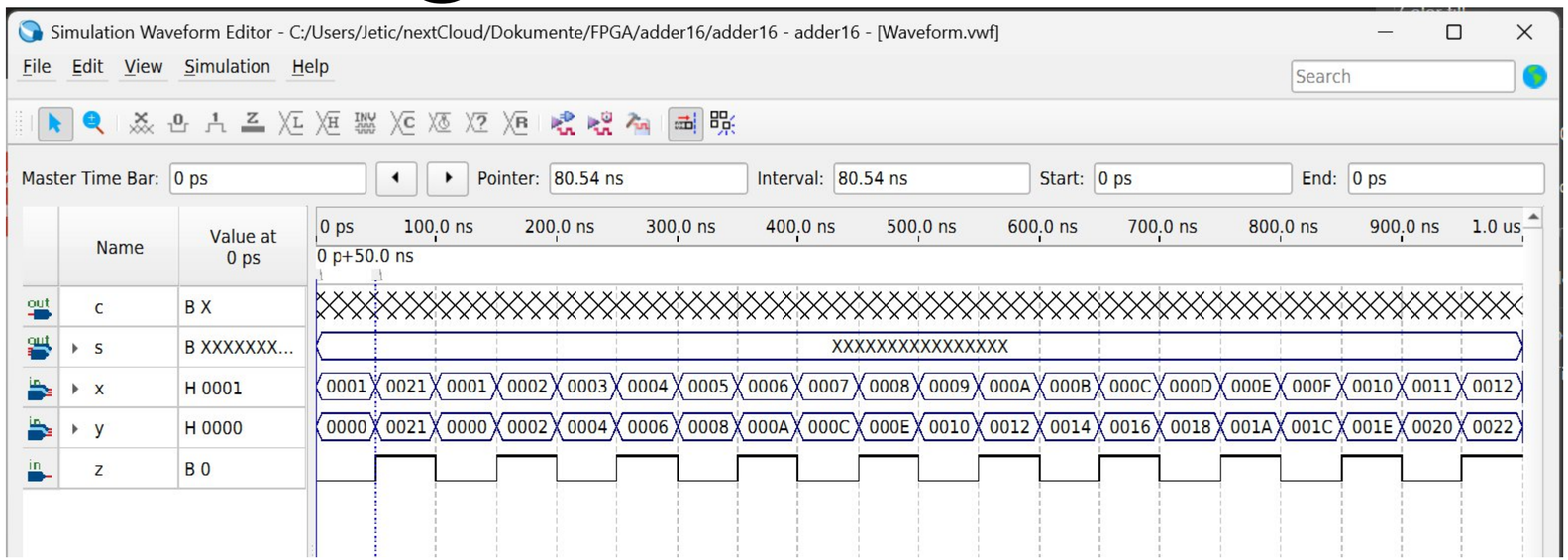




10. Go to `Simulation Settings`, in `Functional Simulation Settings`, on the line where it begins with `vsim`, do some changes

Tutorial 1

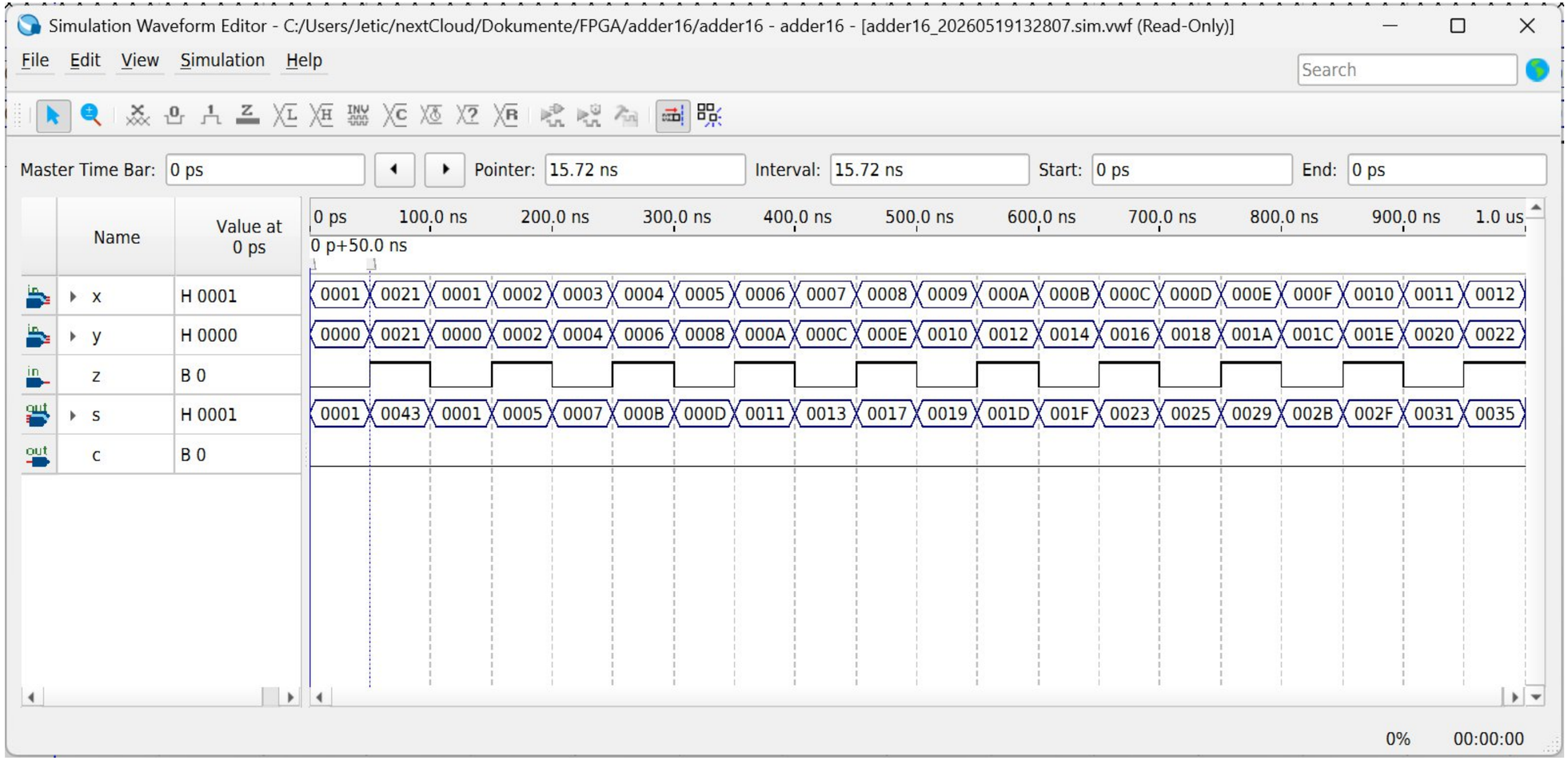
Running Simulation



My recommendations:

1. set `Grid Size` to 50ns
2. set `End Time` to 1us
3. use Count to increase the value of `X` and `Y` every 50ns, and then double click every grid to change the values.
4. change the value of `Z` as well
5. You are ready to `Run Functional Simulation`, which will generate a readonly `.vwf` file for you under `simulation/qsim`

Running Simulation



LAB 1 Part 2
A Verilog Exercise

LAB 1 Part 2
A Verilog Exercise

- Task 1: Implement `adder16.v`
  - You must show `adder16` working in `adder16.sim.vwf`

Concept

LAB 1 Part 2
A Verilog Exercise

- Task 1: Implement `adder16.v`
  - `16bit X, Y input; 1bit Z input`
  - `16bit S output; 1bit C output`
- You must show `adder16` working in `adder16.sim.vwf`

Concept

LAB 1 Part 2
A Verilog Exercise

- Task 2: Implement `addsub16.v`
  - `16bit X, Y input; 1bit AS input`
  - `16bit O output; 1bit C output`
- You must show `addsub16` working in `addsub16.sim.vwf`

Concept