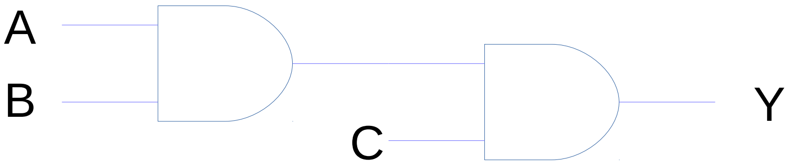
Introduction to Verilog and ModelSim

(Part 4 – Combinational Logic)

Combinational Logic

- Sometimes known as combinatorial logic
- One or more logic operations
  - No clock/gate/latch signals
  - Output is continuously updated
  - Inherent propagation delay (pull-up/pull-down of signal lines)
- It implements functionality of a system



Verilog Operators

- Important ones
  - Bitwise operators
  - Concatenation operator
- Useful ones
  - Conditional operator
  - Shift, reduction, arithmetic operators
- Not-so-direct ones
  - Logical, equality operators

Important Operators

- Bitwise operators ('~', '&', '|', '^')
  - Have been discussed earlier
  - Represent basic logic gates
- Concatenation ('{' and '}' pair)
  - Used to combine bit(s) of signal to form a bus or multi-bits signal (used in the testbench earlier)
  - Example:

```
assign result = { sigA, sigB, sigC }
```

Try this!

```
module merge
(
   paramA,
   paramB,
   paramC
);

input paramA, paramB;
output[1:0] paramC;
wire[1:0] paramC;

assign paramC =
{ paramB, paramA };

endmodule
```

```
module merge_tb ();
reg inputA, inputB;
wire[1:0] outputC;
reg[1:0] checkD;
integer loop;
initial
begin
   for (loop=0;loop<4;loop=loop+1)
   begin
      {inputA,inputB} = loop;
      checkD = {inputA,inputB};
      #10;
   end
end
merge dut (inputA, inputB, outputC);
endmodule
```

Things to Note

- Verilog array is a BUS!
  - Grouping of signal

```
wire [1:0] sigA; // i.e. wire sigA1, sigA0;
reg [7:0] acc;
```

- How to extend-replicate a signal
  - e.g. create an array of the same bit

```
input sigA;
wire [3:0] sigB;

assign sigB = {4{sigA}};
```

Useful operators

- Not really needed but can make things easier
- Conditional operator ('?' and ':' set)
  - Functions as multiplexer
  - Example:

    ```
    assign result = condition ? sigA : sigB;
    ```
- Shift operators ('>>' and '<<')
  - Function as shift logic (not necessarily register!)
  - Example:

    ```
    assign result = sigA >> 1; // 1-bit shift
    ```

Practical Session 4.1

- Implement 2 x 2-1 multiplexer modules
  - First using conditional operator, then using gates
- Create a suitable testbench for the modules
- Simulate and analyze the results to verify the functionality of the modules

Useful operators (cont.)

- Reduction operators ('&', '|', '^', '~&', '~|', '~^')
  - Multi-bits signal reduced to single bit through logic
  - Example:

```
assign result = ~|sigA ; // zero flag
```

- Arithmetic operators ('+', '-', '*', '/', '%')
  - Arithmetic operations (exactly like C!)
  - Example:

```
assign result = sigA + sigB; // addition
```

Practical Session 4.2

- Implement 4-bit adder modules
  - First using arithmetic operator, then using gates
- Create a suitable testbench for the modules
- Simulate and analyze the results to verify the functionality of the modules

You can change the display format in the waveform viewer by right-clicking on a signal and select 'Radix', followed by the desired format.

Not-so-direct Operators

- Logical operators ('&&', '||', '!')
  - Zero-non-zero kind of logic (single bit output)
- Equality: logical equal operators ('==', '!=')
  - Output is either 0 (false), 1 (true), or X (unknown)
- Equality: case equal operators ('===', '!==')
  - Input can be X (unknown) or Z (high impedance)
  - Output is either 0 (false), 1 (true)

Practical Session 4.3 (optional)

- Implement 2 x 4-bit input logic gate modules
  - First using bitwise operator, then logical operator
- Create a suitable testbench for the modules
- Simulate and analyze the results to find out the difference between the two modules

Try this!

```
module addsub
(
    sel,
    inputA,
    inputB,
    resultC
);

input sel;
input[3:0] inputA,
inputB;
output[3:0] resultC;
wire[3:0] resultC;

assign resultC = sel ?
inputA – inputB : inputA +
inputC;

endmodule
```

```
module addsub_tb ();
reg do_sub;
reg[3:0] inputA, inputB;
wire[3:0] outputC;
integer loopA, loopB;
initial
begin
    for(loopA=0;loopA<16;loopA=loopA+1)
    begin
        inputA = loopA;
    for(loopB=0;loopB<16;loopB=loopB+1)
    begin
        inputB = loopB;
        do_sub = 0;
        #10;
        do_sub = 1;
        #10;
    end
    end
end
addsub dut (do_sub, inputA, inputB, outputC);
endmodule
```

Practical Session 4.4

- Implement 4-bit ALU
  - Arithmetic (ADD,SUB) and logic (AND, OR)
- Create a suitable testbench for the modules
- Simulate and analyze the results to verify the functionality of the ALU