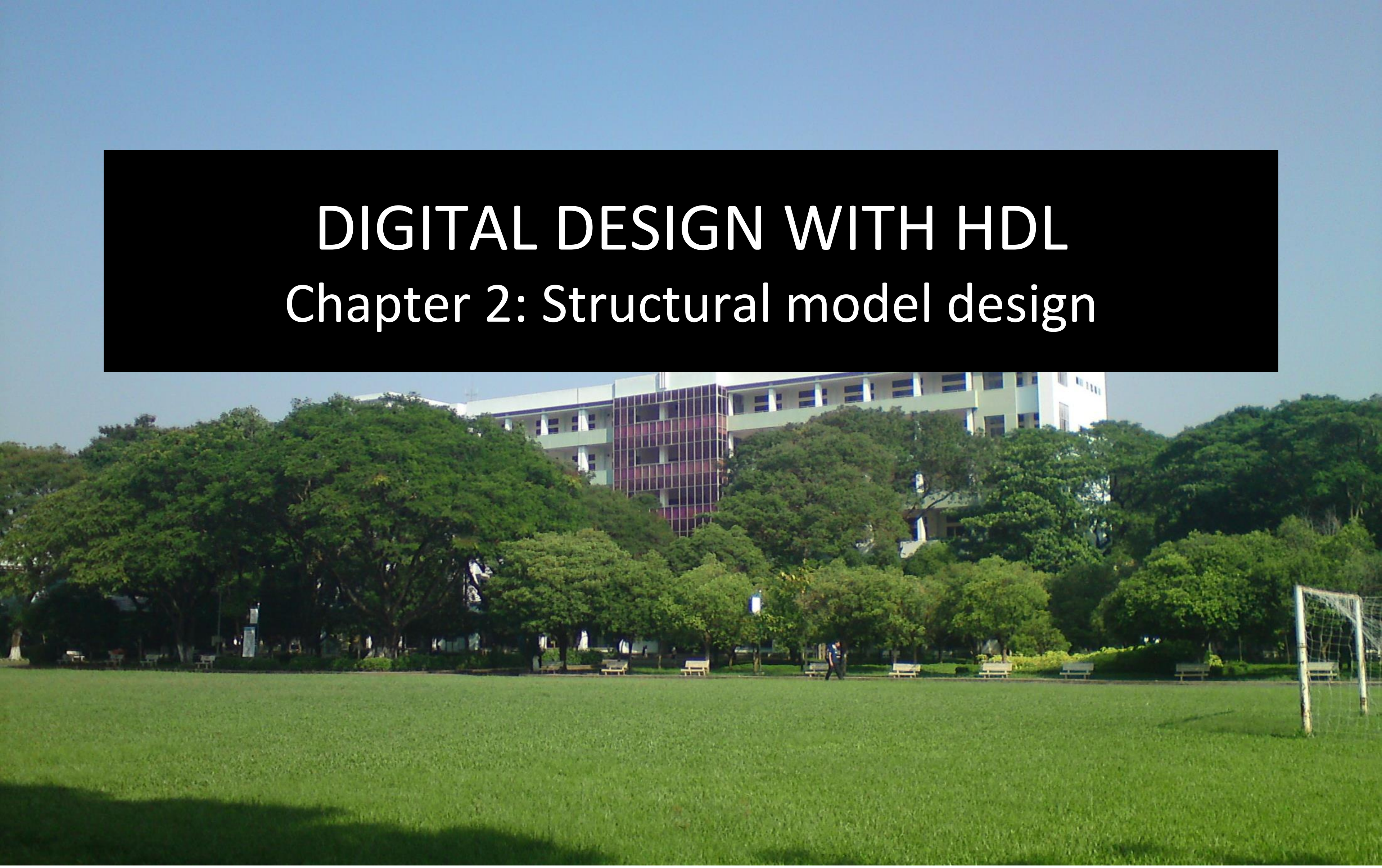


DIGITAL DESIGN WITH HDL

Chapter 2: Structural model design

Structure model

- A schematic in text form
- Structural module can be designed using (**instances**):
  - **Primitive** gates (predefined simple gates)
  - User defined modules (previously designed and tested)
- Interconnections between instances are specified using nets data-type

Structural model - design steps

1. Create module interface
   - Define the module name
   - Declare the port list
2. Declare the internal wires needed to connect gates
3. Instantiate the primitive gates in the circuit
4. Instantiate user defined modules in the circuit (if required)
5. Put the names of the wires in the correct port locations of the gates
   - For primitives, outputs always come first
   - For user defined instantiation, use connection methods (discussed later)

Example

```
module half_adder_struc(a, b, sum, cout);
    input a, b;
    output sum, cout;

    xor(sum, a, b);
    and(cout, a, b);

endmodule
```

```
module fulladder_struc(a, b, cin, sum, cout);
    input a, b, cin;
    output sum, cout;
    wire c1, c2, s1;

    half_add_struc PARTSUM(a, b, s1, c1);
    half_add_struc SUM(s1, cin, sum, c2);

    or(cout, c2, c1);

endmodule
```

Connection wires →

User defined modules →

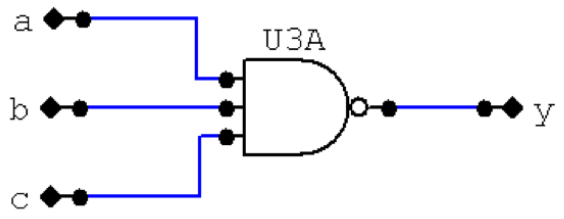
Primitive gates →

Primitive gates

- Verilog provides a set of gate primitives (part of language)
  - One or more scalar inputs and only one scalar output
    - eg. and, nand, or, nor, xor, xnor, not, buf, etc.
  - Names of primitive are keywords
  - Known "behavior"
  - Cannot access "inside" description
- Can also model at the transistor level
  - Most people don't, we won't
- Note: Flip-flops are not primitives

Primitive gates

- Output port is always the first port
  - Inputs follows (unlimited) - except not
- Optional: instantiation name & delay



```
nand (y, a, b, c);
```

```
nand N1 (y, a, b, c);
```

Optional instantiation name

Inputs

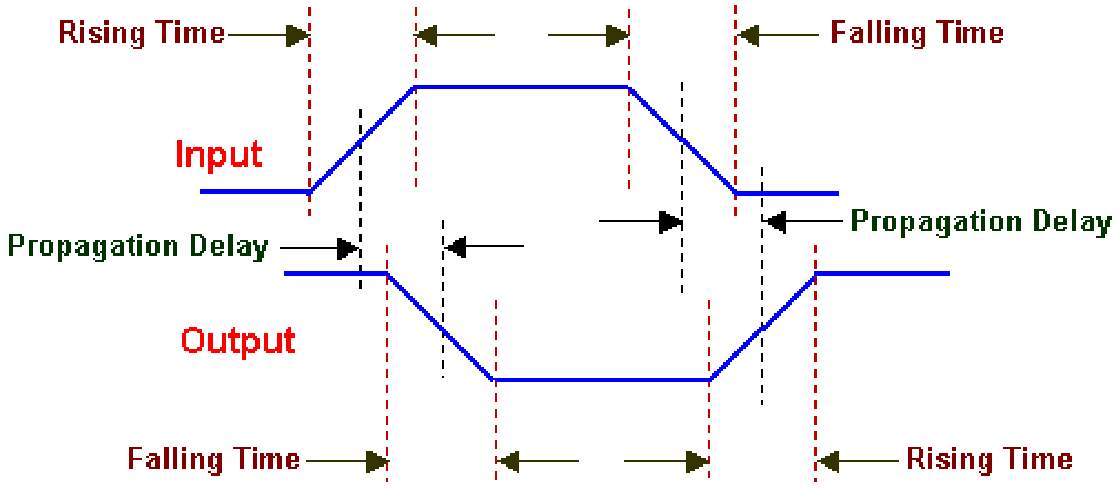
syntax:

```
gate_type delay inst1 (output_1, input_11, input_12, . . . , input_1n),
          delay inst2 (output_2, input_21, input_22, . . . , input_2n),
          ...
          delay instm (output_m, input_m1, input_m2, . . . ,input_mn);
```

Optional

Gate delay

- All gates have propagation delay
  - Time for processing signals
  - Technology-dependent
  - Only simulation cares delay
  - Synthesis process will not take into account
- Types of delay
  - Rise delay: $0 \rightarrow 1$
  - Fall delay: $1 \rightarrow 0$
  - Turn-off delay: $0{,}1 \rightarrow x, z$



Gate delay - Verilog

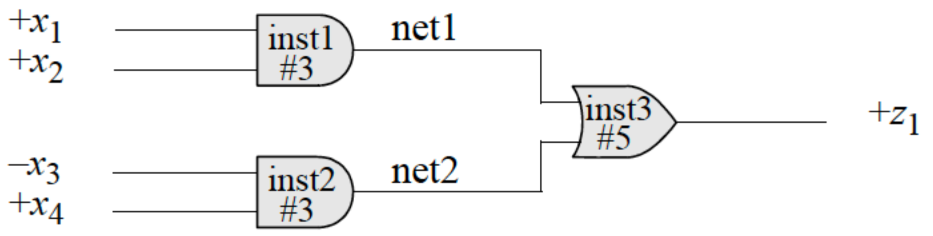
- Can specify all three types of delay
  - But, mainly use one value for all

Syntax: #(rise, fall, turn-off)

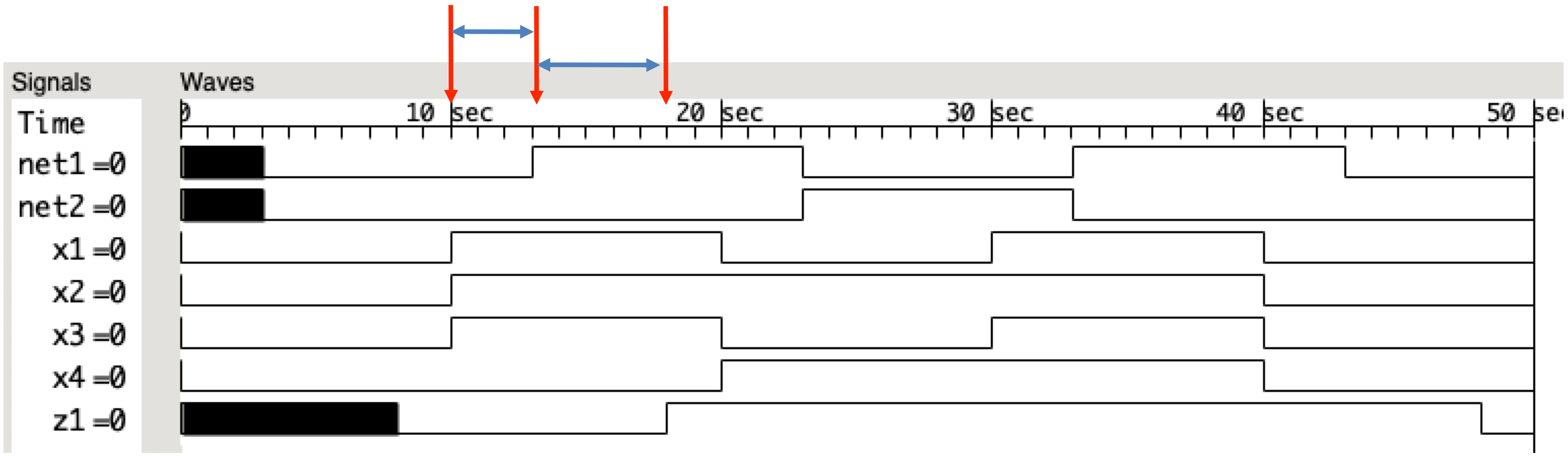
Mainly used: #delay

- Notes:
  - All values must be constant expression
  - Can be specify more precisely: <minimum : typical : maximum>

Example



```
module log_eqn_sop8 (x1, x2, x3, x4, z1);
    input x1, x2, x3, x4;
    output z1;
    wire net1, net2;
    and #3 inst1 (net1, x1, x2);
    and #3 inst2 (net2, ~x3, x4);
    or #5 inst3 (z1, net1, net2);
endmodule
```



Waveform generated by icarus-verilog and displayed by gtkwave

Data types: wire

- Verilog defines two data types
  - Net: physical wire or group wires (bus) for connecting hardware elements
    - wire, tri, wand,…
  - Register: storage elements whose values are retained until updated
    - reg
- Structural model uses only net data type, especially wire
  - Values are determined by logic driving the net

Scalar & vector signals

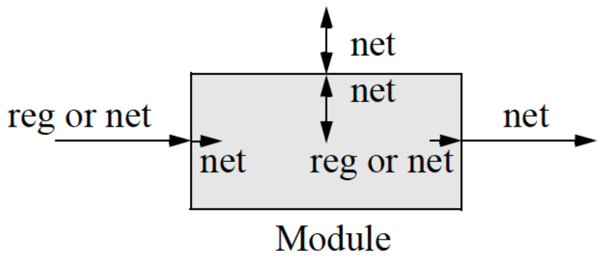
- Scalar: signal bit - do not need to specify the size
  - wire a; //the size of signal a is one bit
- Vector: multiple bit (bus) - specify size using range
  - Syntax: $[MSB: LSB]$
    - $MSB, LSB \in \mathbb{Z}$, constant expression
  - Size of the vector: $|MSB - LSB| + 1$
    - $MSB > LSB$ ✓
    - $MSB < LSB$ ✓
  - Example:
    - wire [3:0] a; //4 bit a bus
    - wire [1:4] b; //4 bit b bus
  - Can access the whole vector, a part of vector, or a single bit

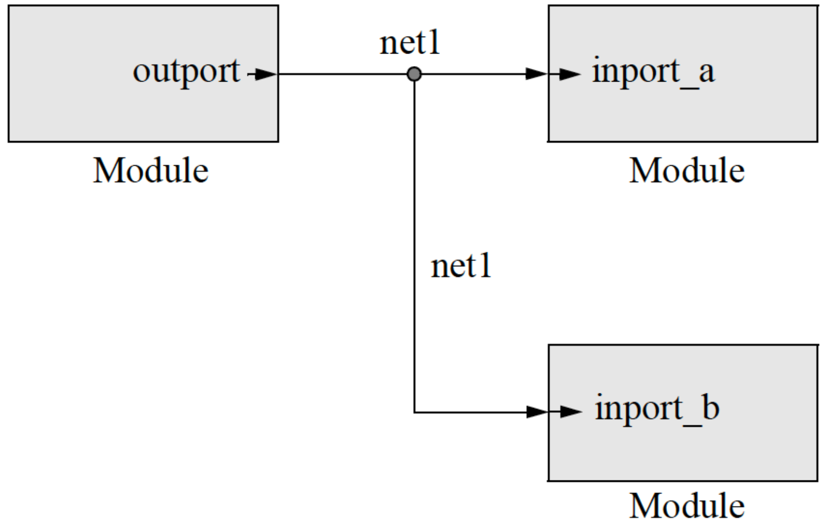
Vectors

- Part select
  - Syntax: *<name>*[*MSB*: *LSB*]
  - The relationship between MSB and LSB must be in accordance with declaration
  - Example:
    - wire [3:0] a;
    - a[3:1]; //correct part-select
    - a[1:2]; //wrong part-select because of reverse order
    - a[4:2]; //wrong part-select because of out-of-range
- Bit select
  - Syntax: *<name>*[*position*]
  - Example: a[1]; a[3];

Connections

- Port connection rules
  - input:
    - Internally: net
    - Externally: reg or net
  - output:
    - Internally: reg or net
    - Externally: net
  - inout:
    - Internally/externally: net
- Error messages will be indicated if violated





Intermodules connections

- Method 1: positional or implicit connections
  - Used for the primitive gates (no choice)
  - Look like function/procedure call in software
  - Acceptable with few ports or interchangeable ports
  - Need to remember the position of each port
- Method 2: connecting by port names
  - Easy to read and manage
  - No need to follow the order of ports declared
  - Syntax: .<port name>(<connected signal>)

Example

```
module decoder(In, En, Out);
        input [1:0] In;
        input En;
        output [3:0] Out;
        ….
endmodule
```

```
module positional(…);
        wire [1:0] pIn;
        wire pEn;
        wire [3:0] pOut;
        ….
        decoder D1(pIn, pEn, pOut);
endmodule
```

```
module name(…);
        wire [1:0] nIn;
        wire nEn;
        wire [3:0] nOut;
        ….
        decoder D1(.En(nEn),
                        .In(nIn),
                        .Out(nOut));
endmodule
```

Empty port connections

- General rules of unconnected ports
  - Input: high-impedance state (high-Z)
  - Output: unused port
- Note: do not leave an input port empty due to unmanageable high-Z states
  - Connect the input to the inactive value
    - Active high $\Rightarrow 0$; active low $\Rightarrow 1$
- Make a port unconnected
  - Position method: a blank between two commas
    - `decoder D1(pIn, , pOut);`
  - Name method: empty signal name
    - `decoder D1(.In(nIn), .En(), .Out(nOut));`

Hierarchy and scope

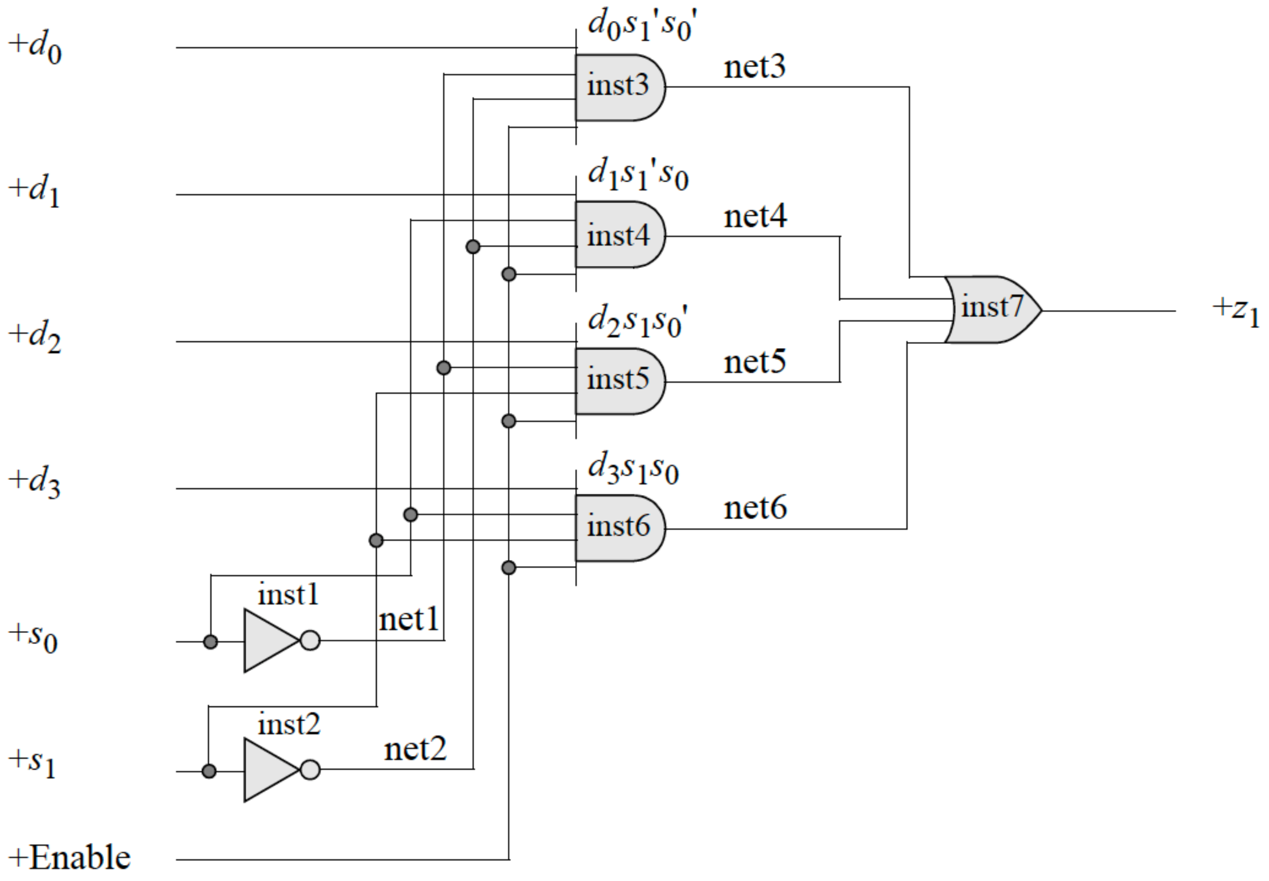
- A module cannot access "internal" signals of instantiations
  - Must make a port when needed

```
module add8bit(cout, sum, a, b);
   output [7:0] sum;
   output cout;
   input [7:0] a, b;
   wire cout0, cout1,… cout6;
   FA A0(cout0, sum[0], a[0], b[0], 1'b0);
   FA A1(cout1, sum[1], a[1], b[1], cout0);
   …
   FA A7(cout, sum[7], a[7], b[7], cout6);
endmodule
```

To access cout6 when instantiating the add8bit must output cout6

Design example

- A 4:1 multiplexer diagram



Design example - Verilog code

```
module mux_4to1 (d, s, enbl, z1);
    input [3:0] d;
    input [1:0] s;
    input enbl;
    output z1;

    not inst1 (net1, s[0]),
        inst2 (net2, s[1]);

    and inst3 (net3, d[0], net1, net2, enbl),
        inst4 (net4, d[1], s[0], net2, enbl),
        inst5 (net5, d[2], net1, s[1], enbl),
        inst6 (net6, d[3], s[0], s[1], enbl);

    or inst7 (z1, net3, net4, net5, net6);
endmodule
```

Exercise 1

- Using Verilog HDL to implement a three bit comparator that consists of two 3-bit inputs a, b and three 1-bit output a_lt_b, a_eq_b, and a_gt_b to indicate if a is less than b, a is equal to b, and a is greater than b, respectively. Given that the following boolean expressions can be used to compare (a[2], b[2] : MSB):

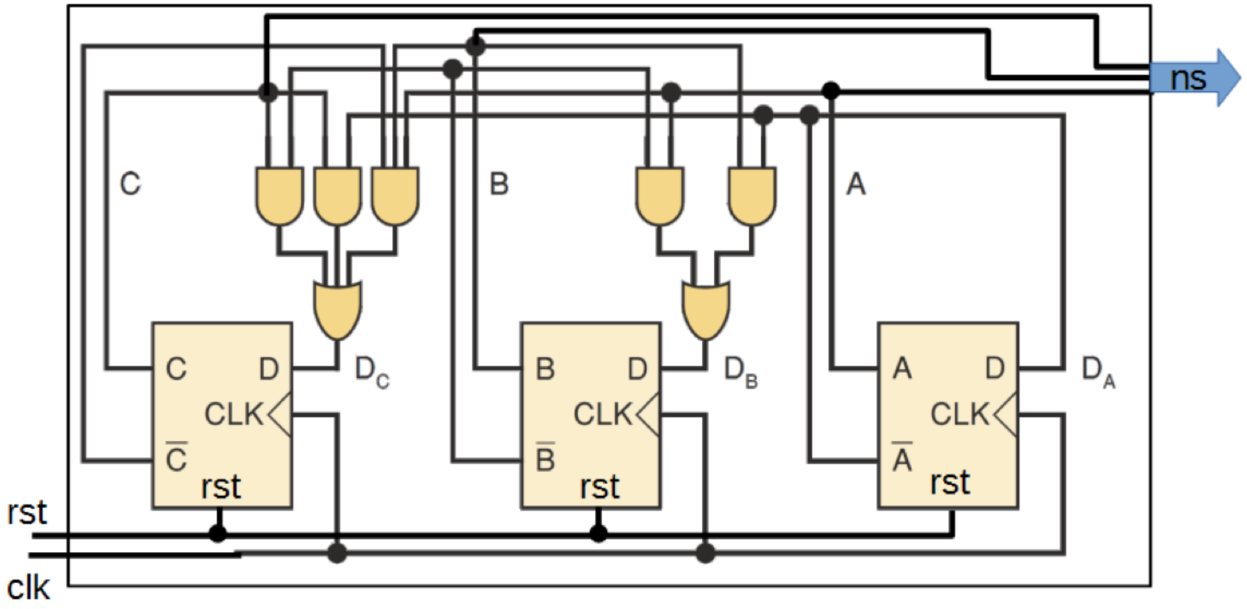
$$\begin{aligned}(a < b) = {} & \overline{a[2]} \cdot b[2] + \overline{a[2] \oplus b[2]} \cdot \overline{a[1]} \cdot b[1] \\ & + \overline{a[2] \oplus b[2]} \cdot \overline{a[1] \oplus b[1]} \cdot \overline{a[0]} \cdot b[0]\end{aligned}$$

$$(a = b) = \overline{a[2] \oplus b[2]} \cdot \overline{a[1] \oplus b[1]} \cdot \overline{a[0] \oplus b[0]}$$

$$\begin{aligned}(a > b) = {} & a[2] \cdot \overline{b[2]} + \overline{a[2] \oplus b[2]} \cdot a[1] \cdot \overline{b[1]} \\ & + \overline{a[2] \oplus b[2]} \cdot \overline{a[1] \oplus b[1]} \cdot a[0] \cdot \overline{b[0]}\end{aligned}$$

Exercise 2

2. Using Verilog HDL to implement a mod 8 counter whose diagram is shown as follow. Assume that there exist a D-flip-flop module whose interface is shown below



```
module dff(input clk, d, output q, input rst);
```

The end