NATIONAL UNIVERSITY OF HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY
FACULTY OF COMPUTER ENGINEERING

LECTURE

Subject:

VERILOG
Hardware Description Language

Chapter5: Structural Model

Lecturer: Lam Duc Khai

Agenda

1. Chapter 1: Introduction (Week1)
2. Chapter 2: Fundamental concepts (Week1)
3. Chapter 3: Modules and hierarchical structure (Week2)
4. Chapter 4: Primitive Gates – Switches – User defined primitives (Week2)
5. Chapter 5: Structural model (Week3)
6. Chapter 6: Behavioral model – Combination circuit (Week4)
7. Chapter 7: Behavioral model – Sequential circuit (Week5)
8. Chapter 8: Tasks and Functions (Week6)
9. Chapter 9: State machines (Week6)
10. Chaper 10: Testbench and verification (Week7)

Agenda

1. What is structural model
2. Structural model in combinational circuit
3. Structural model in sequential circuit

Structural Model

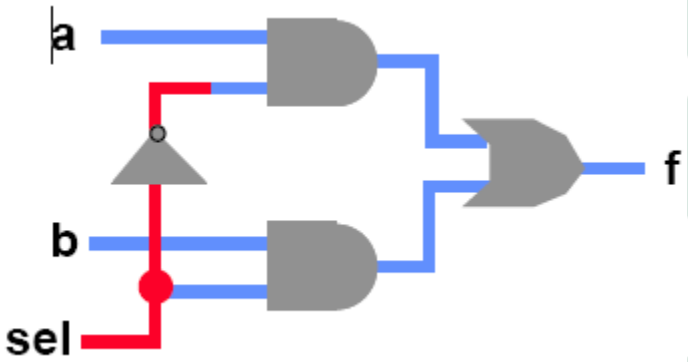
- When Verilog was first developed (1984) most logic simulators operated on netlists
- Netlist: list of gates and how they're connected
- A natural representation of a digital logic circuit
- Not the most convenient way to express test benches

Structural Model (Cont'd)

- Structural
  - Explicit structure of the circuit
  - How a module is composed as an interconnection of more primitive modules/components
  - E.g., each logic gate instantiated and connected to others
  - Structural Verilog
    - List of components and how they are connected
    - Just like schematics, but using text
      - A net list
    - tedious to write, hard to decode
    - Essential without integrated design tools

Structural Model (Cont'd)

- Are built from gate primitives, switches and other modules
- They describe the circuit using logic gates – much as you would see in an implementation of a circuit.

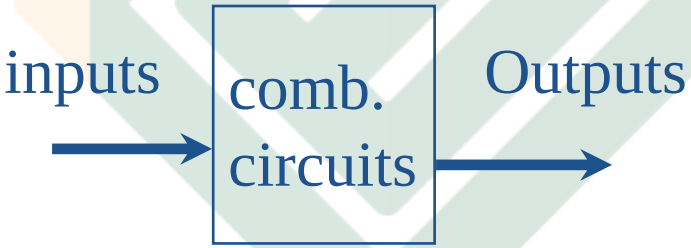



```
module mux (f, a, b, sel);
      output   f;
      input    a, b, sel;

      and #5   g1 (f1, a, nsel),
               g2 (f2, b, sel);
      or   #5  g3 (f, f1, f2);
      not      g4 (nsel, sel);
endmodule
```

Structural Model – Combinational circuit

➢Combinational circuit

- Outputs are functions of inputs

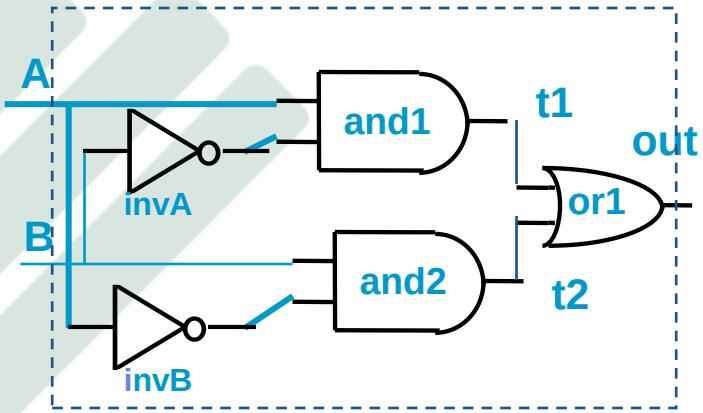


- Examples
  - MUX
  - decoder
  - priority encoder
  - adder

Structural Model – Combinational circuit (Cont'd)

➢Example1

```
module xor_gate ( out, a, b );
  input     a, b;
  output    out;
  wire      abar, bbar, t1, t2;

  not invA (abar, a);
  not invB (bbar, b);
  and and1 (t1, a, bbar);
  and and2 (t2, b, abar);
  or  or1 (out, t1, t2);
endmodule
```

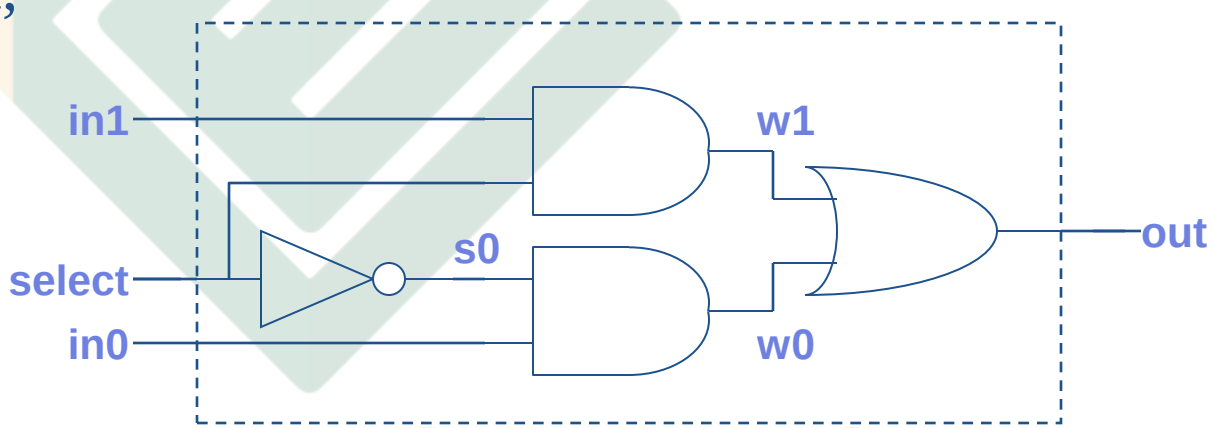


- Composition of primitive gates to form more complex module
- Note use of wire declaration!

Structural Model – Combinational circuit (Cont'd)

➢Example2

```
//2-input multiplexor in gates
module mux2 (in0, in1, select, out);
   input in0,in1,select;
   output out;
   wire s0,w0,w1;

   not (s0, select);
   and (w0, s0, in0),
       (w1, select, in1);
   or  (out, w0, w1);
endmodule // mux2
```

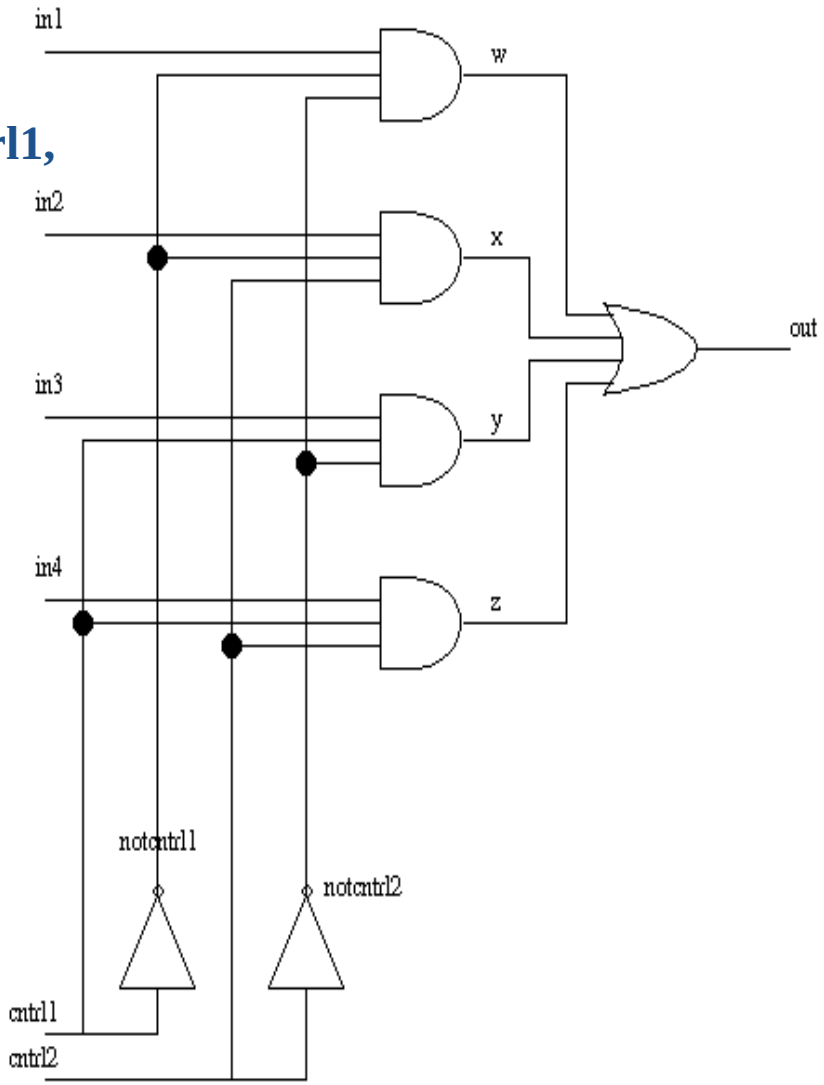


Structural Model – Combinational circuit (Cont'd)

➢Example3

```
module multiplexor4_1(out, in1, in2, in3, in4, cntrl1,
cntrl2);
output out;
input in1, in2, in3, in4, cntrl1, cntrl2;
wire notcntlr1, notcntrl2, w, x, y, z;

not (notcntrl1, cntrl1);
not (notcntrl2, cntrl2);
and (w, in1, notcntrl1, notcntrl2);
and (x, in2, notcntrl1, cntrl2);
and (y, in3, cntrl1, notcntrl2);
and (z, in4, cntrl1, cntrl2);
or (out, w, x, y, z);
endmodule
```

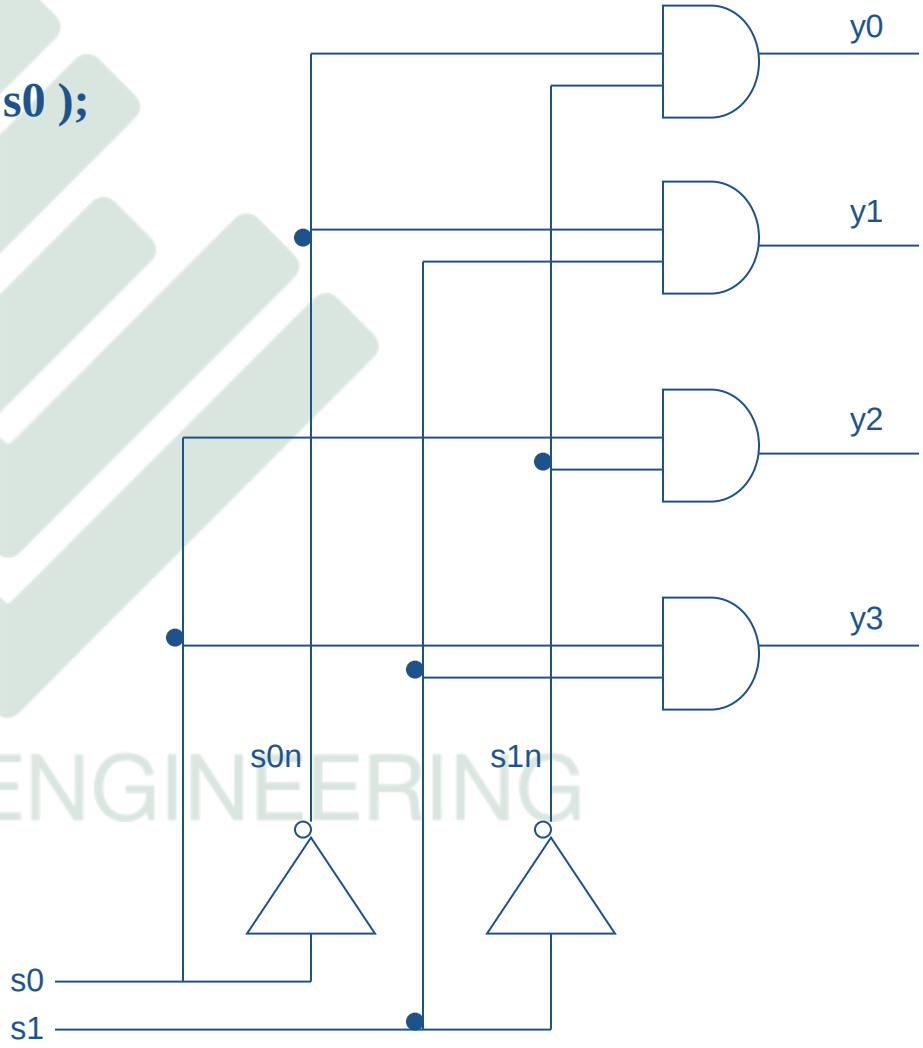
Recall default type is wire.



Structural Model – Combinational circuit (Cont'd)

➢Example4

```
module decoder2_to_4 ( y0, y1, y2, y3, s1, s0 );
// Inputs and outputs
output y0, y1, y2, y3;
input s1, s0;
// Internal wires
wire s1n, s0n;
// Create complements of s1 and s0
not ( s1n, s1 );
not ( s0n, s0 );
and ( y0, s1n, s0n );
and ( y1, s1n, s0 );
and ( y2, s1, s0n );
and ( y3, s1, s0 );


endmodule
```



Structural Model – Combinational circuit (Cont'd)

➢Example5

- 8-to-3 encoder truth table (Folder Encoder)

Input								Output		
D7	D6	D5	D4	D3	D2	D1	D0	A2	A1	A0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Structural Model – Combinational circuit (Cont'd)

➢Example5 (Cont'd)

- 8-to-3 encoder logic equations (Folder Encoder)
  - **A0 = D1 + D3 + D5 + D7**
  - **A1 = D2 + D3 + D6 + D7**
  - **A2 = D4 + D5 + D6 + D7**

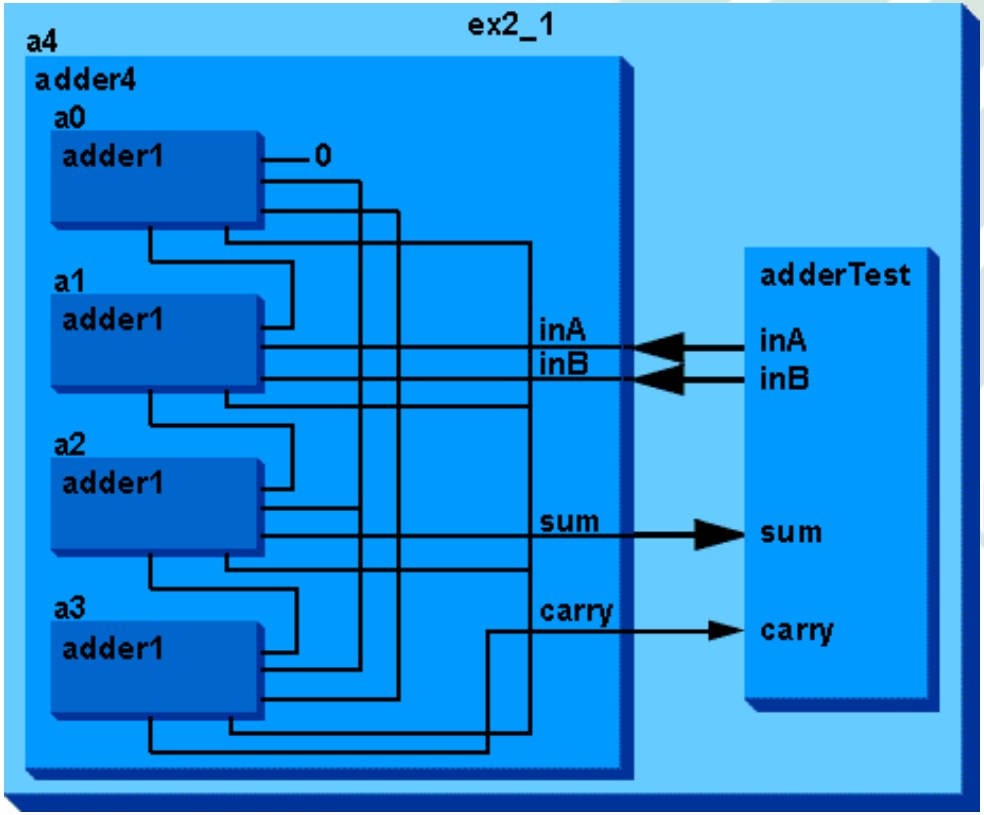
- 8-to-3 encoder structural model (Folder Encoder)

```
module encoder8_3( A , D );
    output[2:0] A;
    input[7:0] D;
    or( A[0], D[1], D[3], D[5], D[7] );
    or( A[1], D[2], D[3], D[6], D[7] );
    or( A[2], D[4], D[5], D[6], D[7] );
endmodule
```

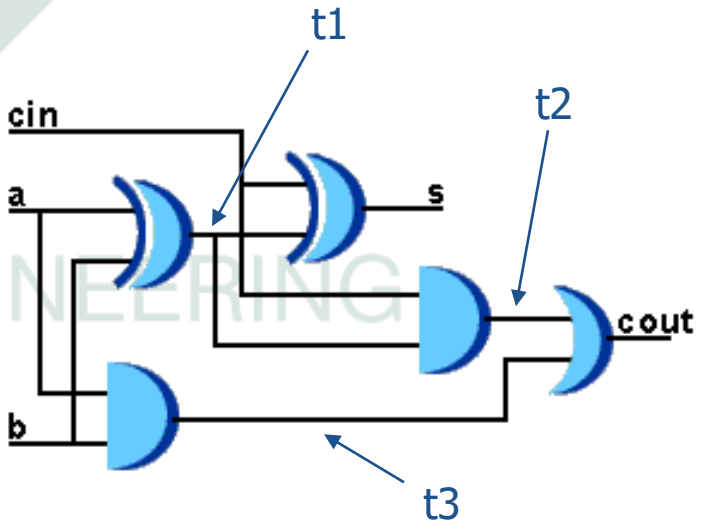
Structural Model – Combinational circuit (Cont'd)

➢ **Example6**

4-bit Adder (Folder 4BitAdder)



- 4-bit adder. It takes two 4-bit operands, inA and inB, and produces a 4-bit result, sum, and a 1-bit carry. It is composed of four
- 1-bit adders, each of which has a carry in as well as the two operand inputs



Full 1-bit

Structural Model – Combinational circuit (Cont'd)

➢Example6 (Cont'd)

- 1bit-adder structural model

```
module adder1 (s, cout, a, b, cin);
    output s, cout;
    input a, b, cin;

    xor (t1, a, b);
    xor (s, t1, cin);
    and (t2, t1, cin),
        (t3, a, b);
    or  (cout, t2, t3);
endmodule
```

1-bit full adder module. Refer the circuit diagram before.

Structural Model – Combinational circuit (Cont'd)

➢Example6 (Cont'd)

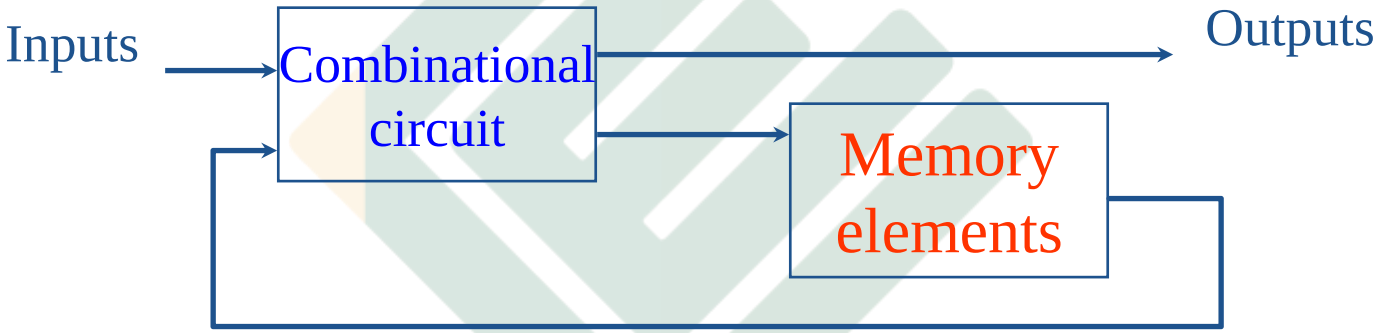
- 4bit-adder structural model

```
module adder4 (sum, carry, inA, inB);
    output [3:0] sum;
    output carry;
    input [3:0] inA, inB;

    adder1 a0 (sum[0], c0,    inA[0], inB[0], 1'b0);
    adder1 a1 (sum[1], c1,    inA[1], inB[1], c0);
    adder1 a2 (sum[2], c2,    inA[2], inB[2], c1);
    adder1 a3 (sum[3], carry, inA[3], inB[3], c2);
endmodule
```

4-bit adder module composed of 4 1-bit adders modules. Structural code.

Structural Model – Sequential circuit

➢Sequential circuit

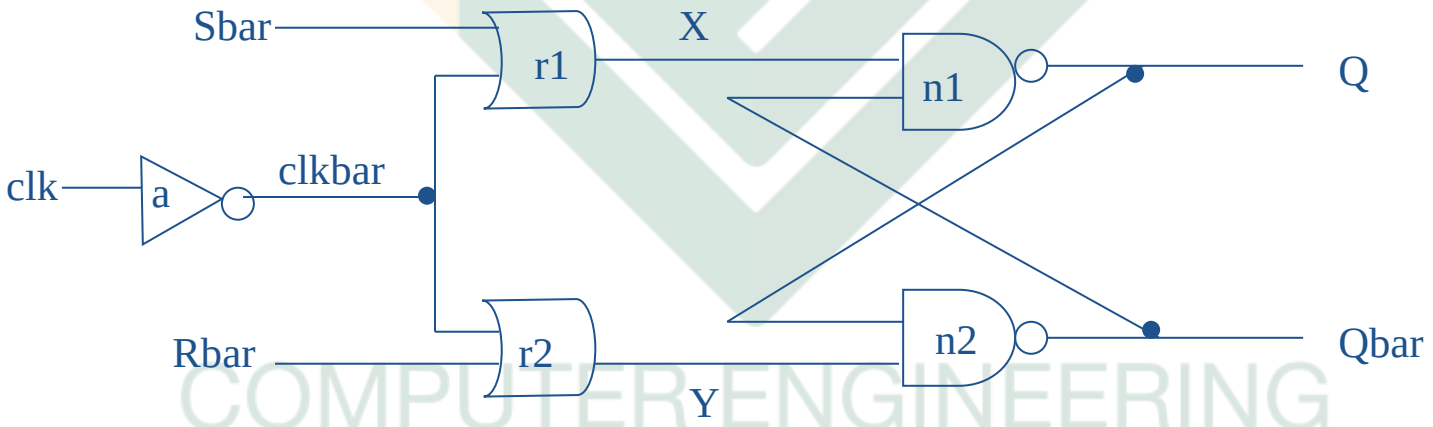


- a feedback path
- the state of the sequential circuits
- the state transition
  - synchronous circuits
  - asynchronous circuits

Structural Model – Sequential circuit (Cont'd)

➢ **Example1**

- Set-Reset (SR-) latch (clocked)



Clocked SR-latch:

(1) State can change only when clock is high

(2) Potential non-deterministic behavior if both input Sbar and Rbar are 0

Structural Model – Sequential circuit (Cont'd)

➢Example1 (Cont'd)

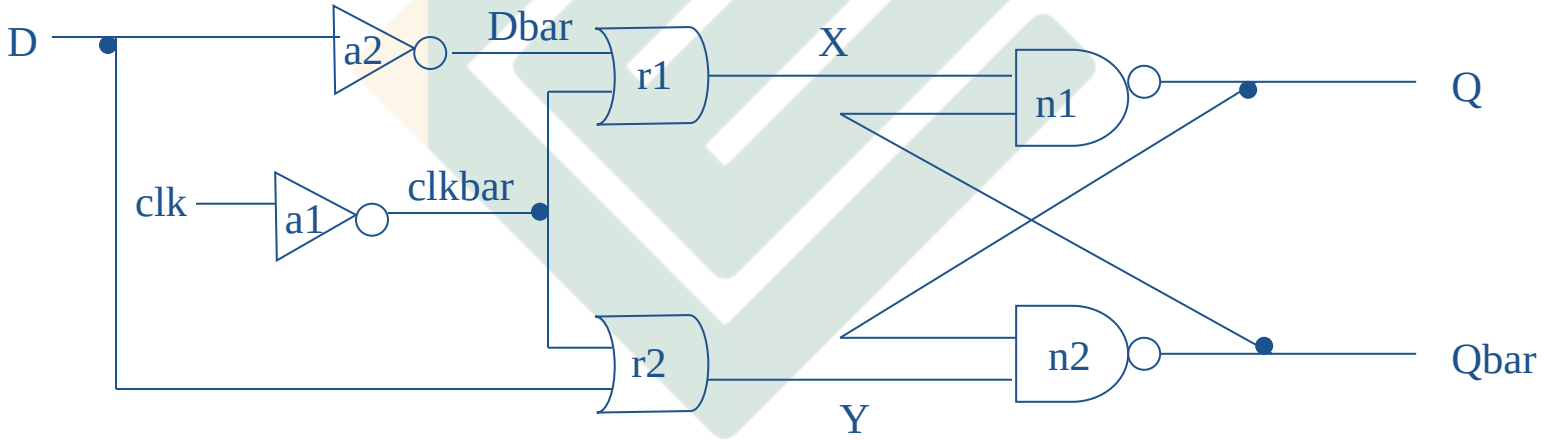
- Set-Reset (SR-) latch (clocked) structural model

```
module clockedSR_latch(Q, Qbar, Sbar, Rbar, clk);

//Port declarations
output Q, Qbar;
input Sbar, Rbar, clkbar;
wire X, Y;
// Gate declarations
not a(clkbar, clk);
or r1(X, Sbar, clkbar);
or r2(Y, Rbar, clkbar);
nand n1(Q, X, Qbar);
nand n2(Qbar, Y , Q);
endmodule
```

Structural Model – Sequential circuit (Cont'd)

➢Example2

- D latch (clocked)



Clocked D-latch:

(1) State can change only when clock is high

(2) Single data input

(3) No problem with non-deterministic behavior

Structural Model – Sequential circuit (Cont'd)

➢Example2 (Cont'd)

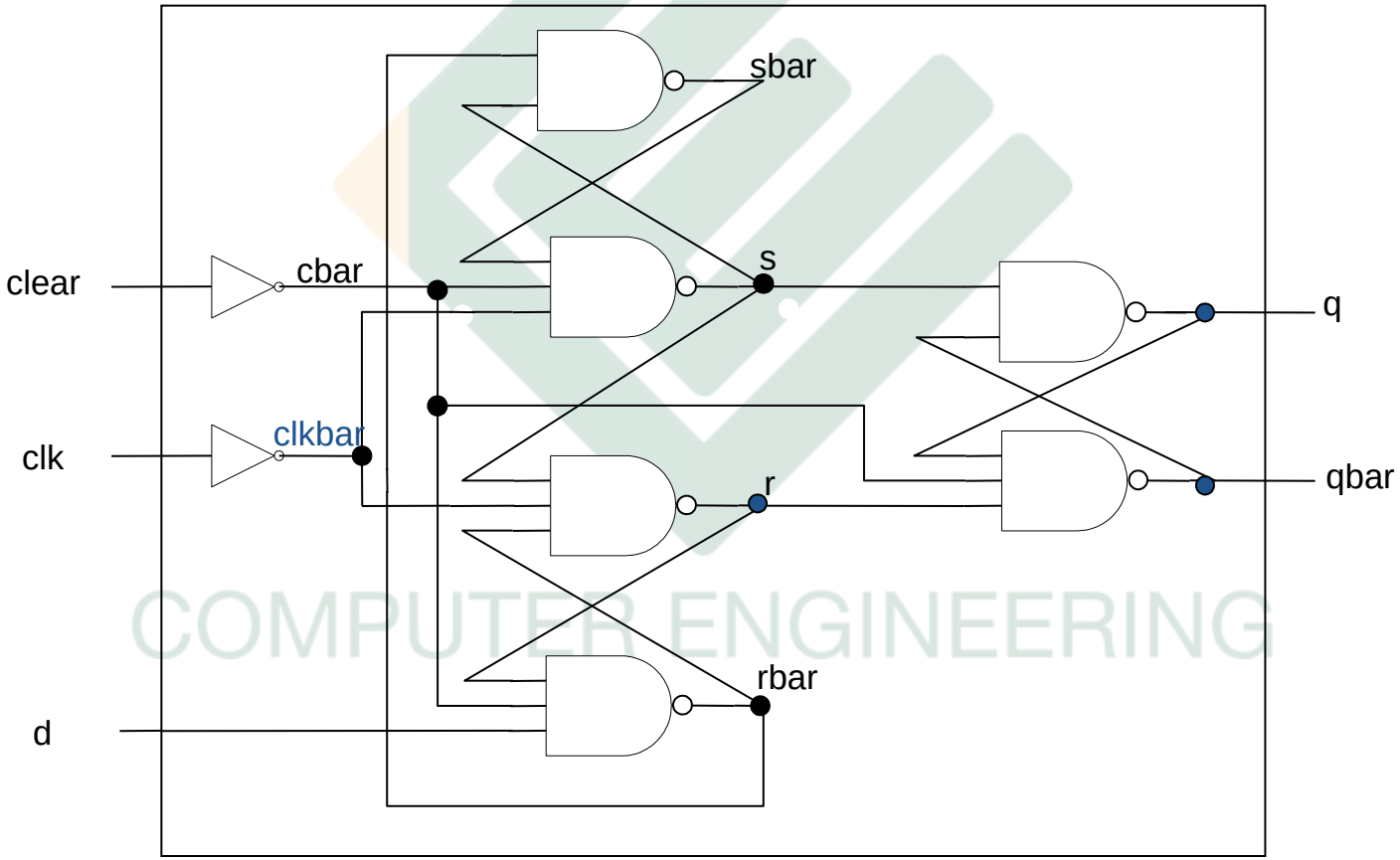
- D latch (clocked) structural model

```
module clockedD_latch(Q, Qbar, D, clk);

//Port declarations
output Q, Qbar;
input D, clk;
wire X, Y, clkbar, Dbar;
// Gate declarations
not a1(clkbar, clk);
not a2(Dbar, D);
or r1(X, Dbar, clkbar);
or r2(Y, D, clkbar);
nand n1(Q, X, Qbar);
nand n2(Qbar, Y , Q);
endmodule
```

Structural Model – Sequential circuit (Cont'd)

➢ Example3

- Negative edge-triggered D-flipflop



Negative edge-triggered D-flipflop implemented using 3 SR latches

Structural Model – Sequential circuit (Cont'd)

➢Example3 (Cont'd)

- Negative edge-triggered D-flipflop structural model

```
module edge_dff(q, qbar, d, clk, clear);
output q,qbar;
input d, clk, clear;
wire s, sbar, r, rbar,cbar;
not (cbar, clear);
not (clkbar, clk);
// Input latches
nand (sbar, rbar, s);
nand (s, sbar, cbar, clkbar);
nand (r, rbar, clkbar, s);
nand (rbar, r, cbar, d);
// Output latch
nand (q, s, qbar);
nand (qbar, q, r, cbar);
endmodule
```

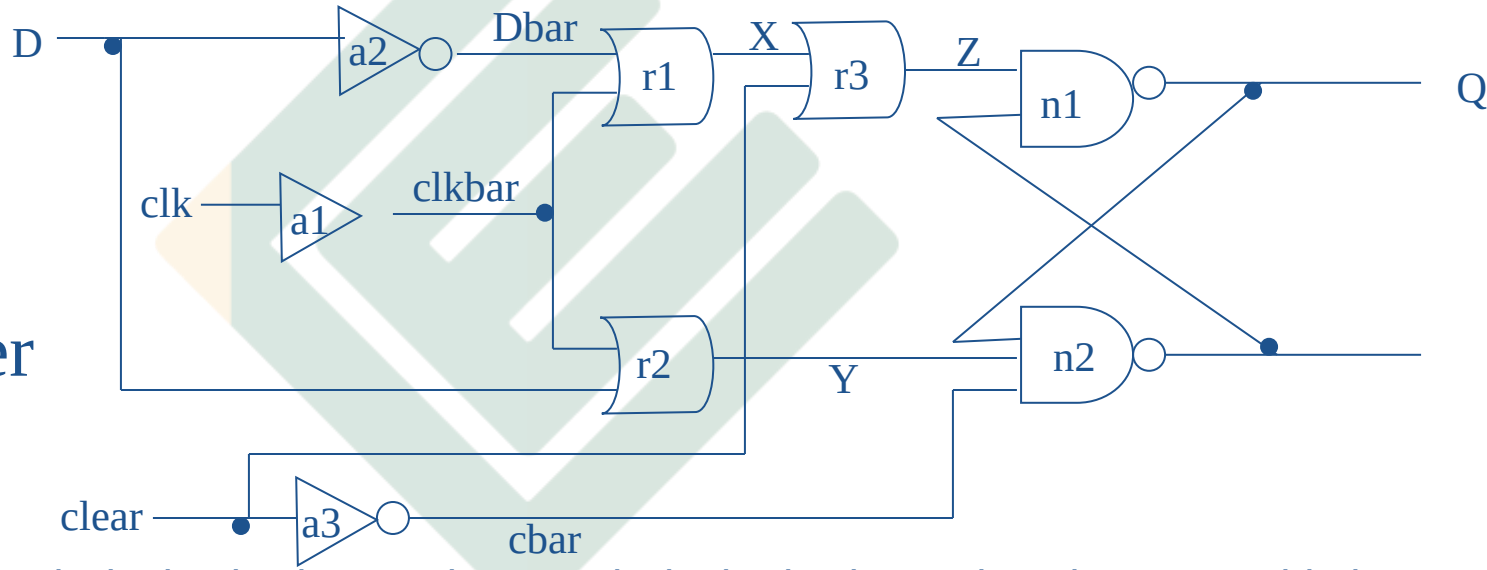
Gate-level D-flipflop
(1) Negative edge-triggered
(2) Made from 3 SR-latches (see circuit)

Extremely important module – it is the fundamental unit of computer memory!

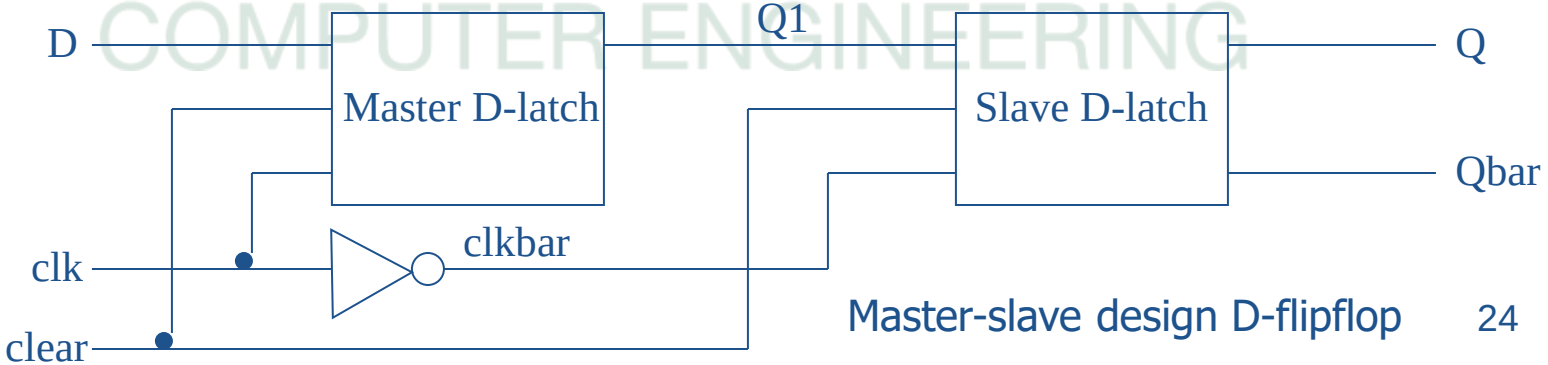
Structural Model – Sequential circuit (Cont'd)

➢Example4

- Negative edge-triggered master-slaver D-flipflop



Clocked D-latch, exactly as in clockedD_latch.v with a clear wire added



Master-slave design D-flipflop

Structural Model – Sequential circuit (Cont'd)

➢Example4 (Cont'd)

- Negative edge-triggered master-slaver D-flipflop structural model

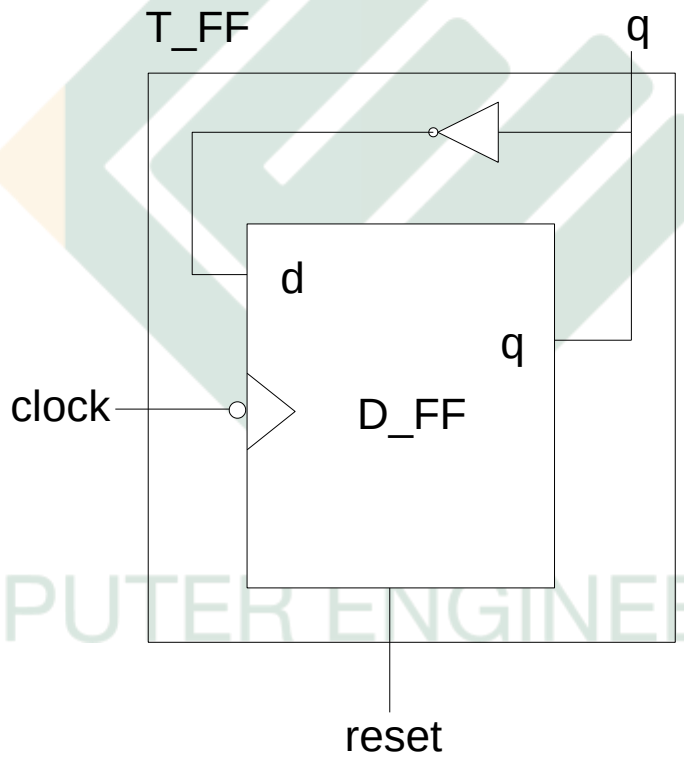
```
// Clocked D-latch as in clockedD_latch.v with a clear signal added
module clockedD_latch(Q, Qbar, D, clk, clear);
output Q, Qbar;
input D, clk, clear;
wire X, Y, Z, clkbar, Dbar, cbar;
not a1(clkbar, clk);
not a2(Dbar, D);
not a3(cbar, clear);
or r1(X, Dbar, clkbar);
or r2(Y, D, clkbar);
or r3(Z, X, clear);
nand n1(Q, Z, Qbar);
nand n2(Qbar, Y, Q, cbar);
endmodule
```

```
// Negative edge-triggered D-flipflop with 2 D-latches in master-slave relation
module edge_dff(q, qbar, d, clk, clear);

output q, qbar;
input d, clk, clear;
wire q1;

clockedD_latch master(q1, , d, clk, clear); // master D-latch
not(clkbar, clk);
clockedD_latch slave(q, qbar, q1, clkbar, clear, writeCtr); // slave D-latch
endmodule
```

Structural Model – Sequential circuit (Cont'd)

➢Example5

- T(Toggle)-flipflop



Negative edge-triggered T-flipflop implemented using a D-flipflop and an inverter gate – *toggles* every clock cycle

Structural Model – Sequential circuit (Cont'd)

➢Example5 (Cont'd)

- T(Toggle)-flipflop structural model

```
module t_ff(q, clk, clear);
output q;
input clk, clear;

// Instantiate the edge triggered DFF
// Complement of output q is fed back.
// Notice qbar not needed. Empty port.
edge_dff ff1(q, ,~q, clk, clear);

endmodule
```

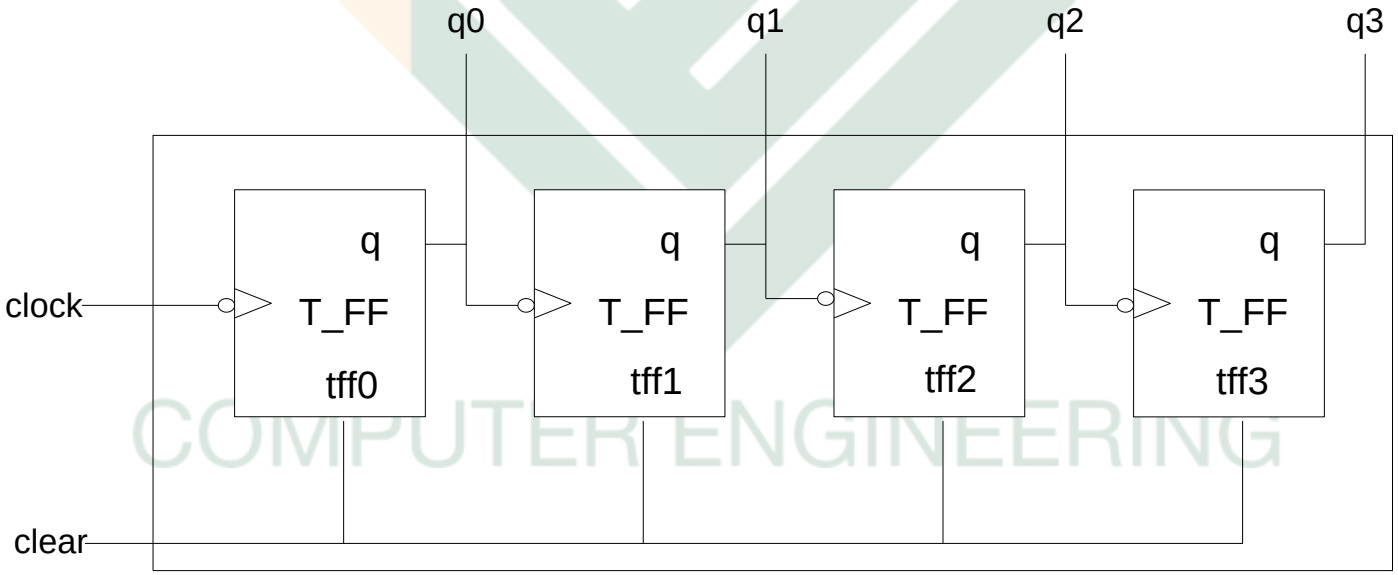
Negative edge-triggered T-flipflop made from a D-flipflop and an inverter gate. It toggles every clock cycle.

Empty port.

Structural Model – Sequential circuit (Cont'd)

➢Example6

- Four-bit ripple counter



4-bit ripple counter made from a series of T-flipflops

Structural Model – Sequential circuit (Cont'd)

➢Example6 (Cont'd)

- Four-bit ripple counter structural model

```
module counter(Q , clock, clear);
output [3:0] Q;
input clock, clear;

// Instantiate the T flipflops
t_ff tff0(Q[0], clock, clear);
t_ff tff1(Q[1], Q[0], clear);
t_ff tff2(Q[2], Q[1], clear);
t_ff tff3(Q[3], Q[2], clear);

endmodule
```

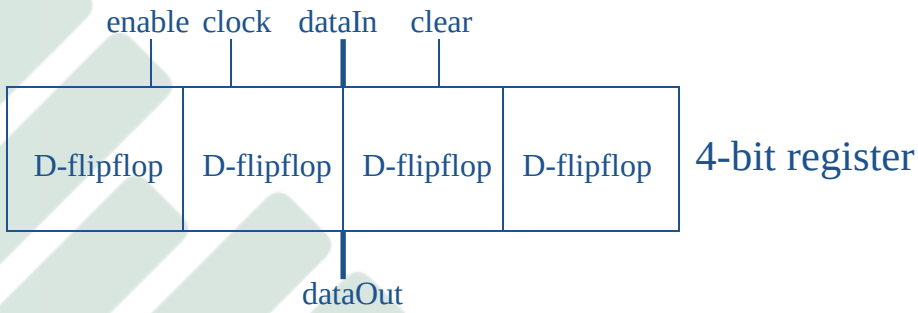
Counter module from 4 T-flipflops.

How does it work?!
The output from tff*i* is the clock for tff*i+1*.

Structural Model – Sequential circuit (Cont'd)

➢Example7

- Register- 4bits




```
// Register module - 4-bit register

module register4bits( dataOut, dataIn, enable, clock, clear );
// Inputs and outputs
output [3:0] dataOut;
input [3:0] dataIn;
input enable, clock, clear;
// 4 D-flipflops
edge_dff ff0( dataOut[0], dataIn[0], enable, clock, clear );
edge_dff ff1( dataOut[1], dataIn[1], enable, clock, clear );
edge_dff ff2( dataOut[2], dataIn[2], enable, clock, clear );
edge_dff ff3( dataOut[3], dataIn[3], enable, clock, clear );

endmodule
```

END