Sequential Circuit

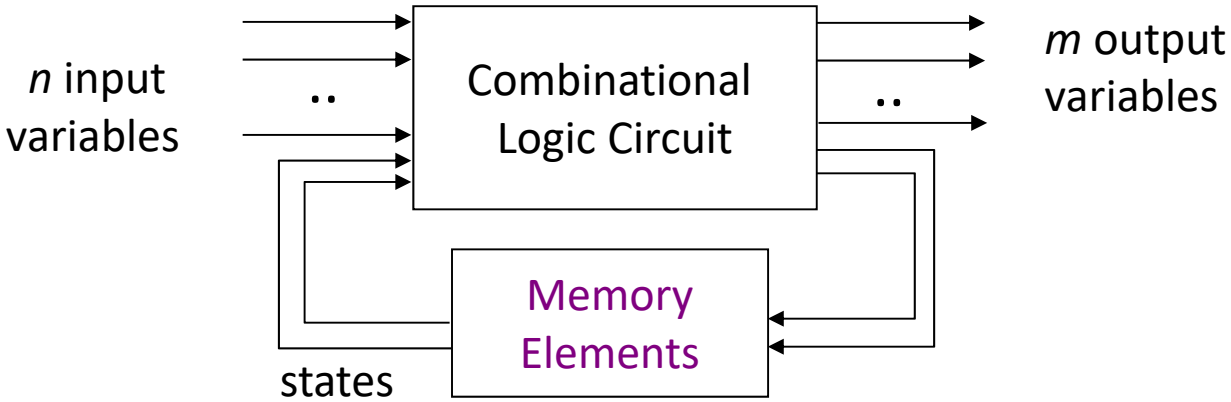
Ing-Chao Lin
Adapted from slides by Prof. Pei-Yin Chen

Outline

- Sequential Circuit
- Clock Period
- Latch
- Flip-Flop
- A Register file
- Counters
- Watch Out for Unintentional Latches
- Blocking vs. Non-Blocking

Sequential Circuit (1/2)

A sequential circuit is a system whose outputs at any time are determined from the present combination of inputs and the previous states.



- Sequential components contain memory elements

  Ex: Ring counter that starts the answering machine after 4 rings

Sequential Circuit (2/2)

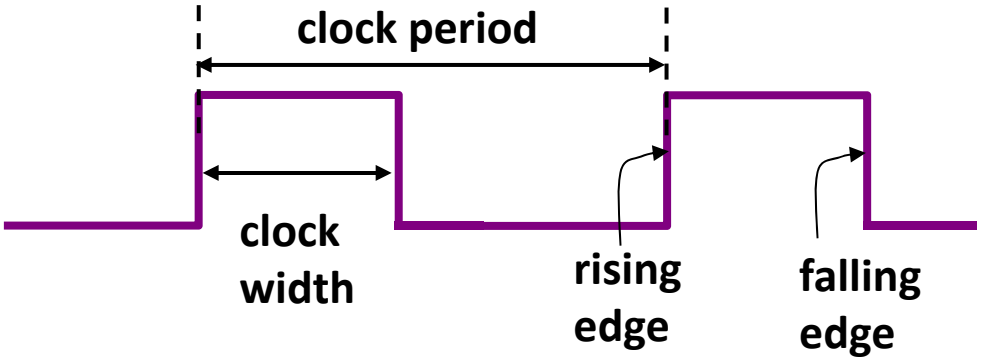
- Sequential components can be: **asynchronous** or **synchronous**
- Asynchronous sequential circuit:
  - Change their states and outputs whenever a change in inputs occurs
- Synchronous sequential circuit:
  - Change their states and outputs at fixed points of time (specified by clock signal)

Most circuits are synchronous circuits (easy and tool-supportable).

Synchronous storage components store data and perform some simple operations.

Synchronous storage components include:
(1) registers (2) counters (3) register files
(4) memories (5) queues (6) stacks

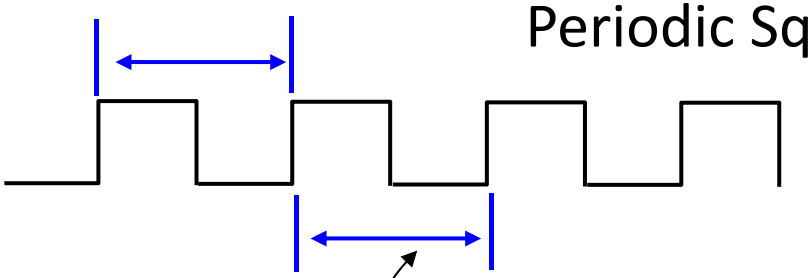
Clock Period



- **Clock period** (measured in micro or nanoseconds) is the time between successive transitions in the same direction
- **Clock frequency** (measured in MHz or GHz) is the reciprocal of clock period
- **Clock width** is the time interval during which clock is equal to 1
- **Duty cycle** is the ratio of the clock width and clock period
- Clock signal is active high if the changes occur at the rising edge or during the clock width. Otherwise, it is active low

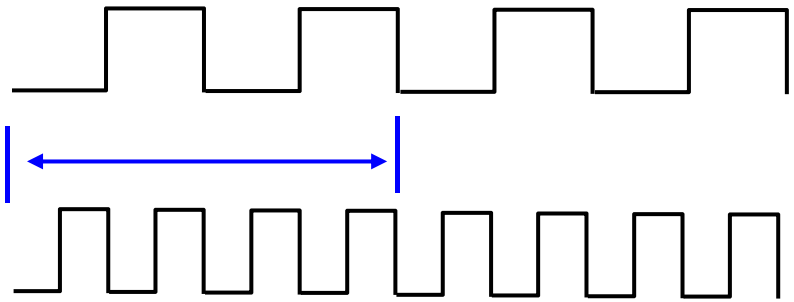
Clock

Periodic Square Wave



20 *ns* ➔ Freq = $\frac{1}{20\ ns} = 50\ \text{MHz}$

50×10^6 /s

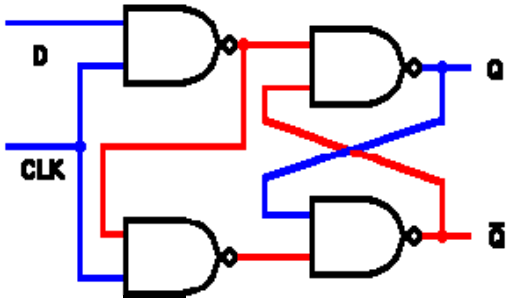


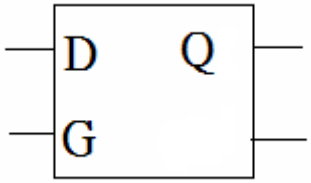
50 MHz

100 MHz

Latch

D	G	Q(t+1)
X	0	Q(t)
0	1	0
1	1	1



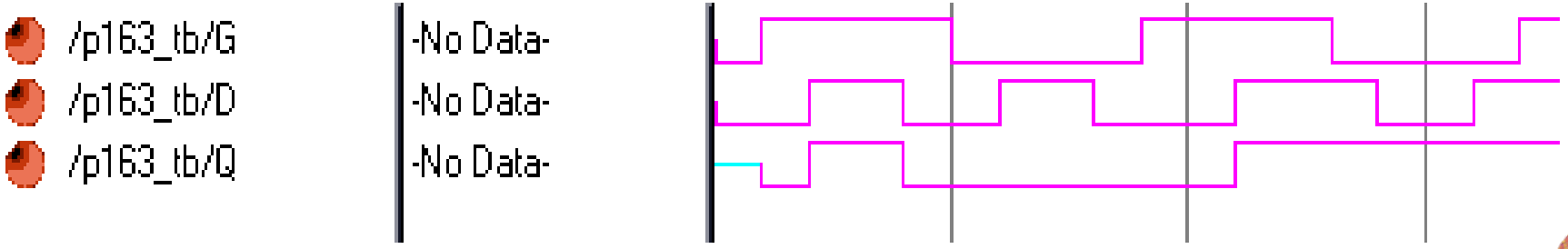



```
module p163(G, D, Q);
  input  G, D;
  output Q;
  reg    Q;

always @(D or G) begin
  if(G)
    Q = D;
end

endmodule
```

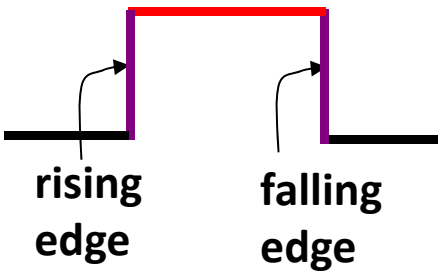
D-type latch
(ignoring delay)

Latches are level-sensitive since they respond to input changes during clock width.

Latches are difficult to work with for this reason.



Flip-Flop



- Flip-Flops respond to input changes only during the change in clock signal (the rising edge or the falling edge).

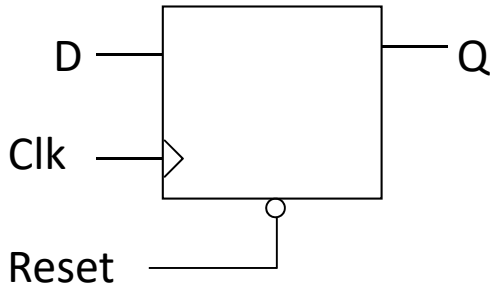
- They are easy to work with, although they are more expensive than latches.

D Flip-flop with Reset

D Flip-flop with asynchronous reset

If Reset changes from 1 to 0,
then reset D flip-flop anyway.
Otherwise, Q=D.

```
module DFF_AR(Clk, Reset, D, Q);
input  Clk, Reset, D;
output Q;
reg    Q;

always @( posedge Clk or
negedge Reset)
begin
  if(!Reset)
    Q<=0;
  else
    Q<=D;
end
endmodule
```

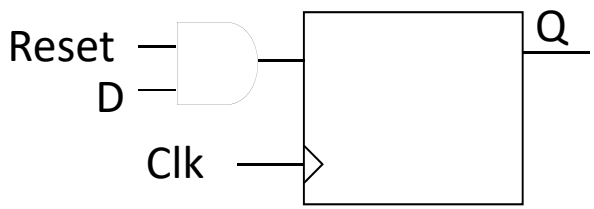


Asynchronous -- Respond immediately !

D Flip-flop with synchronous reset

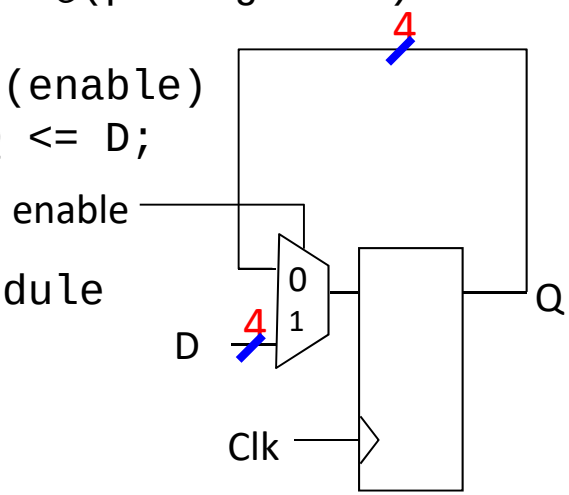
At every positive edge of Clk,
if Reset==0, then reset D flip-flop
(if Reset==1, then Q=D).

```
module DFF_SR(Clk, Reset, D, Q);
input  Clk, Reset, D;
output Q;
reg    Q;
always @(posedge Clk)
begin
  if(!Reset)
    Q=0;
  else
    Q=D;
end
endmodule
```



A Register file

```
module RegFile
(Clk, enable, D, Q);

input   Clk, enable;
input  [3:0] D;
output [3:0] Q;
reg    [3:0] Q;

always @(posedge Clk)
begin
  if (enable)
    Q <= D;
end

endmodule
```



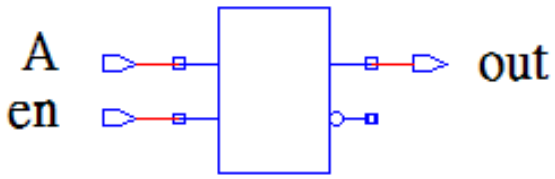
4-bit register = 4 flip-flops

```
// an 16-word register file with one-write and
// two-read ports
module register_file
(clk, rd_addra, rd_addrb, wr_addr,
wr_enable, din, douta, doutb);

input   clk, wr_enable;
input  [7:0] din;
output [7:0] douta, doutb;
input  [7:0] rd_addra, rd_addrb, wr_addr;
reg    [7:0] reg_file [15:0];

// the body of the N-word register file
assign douta = reg_file[rd_addra];
assign doutb = reg_file[rd_addrb];

always @(posedge clk)
  if (wr_enable)
    reg_file[wr_addr] <= din;

endmodule
```

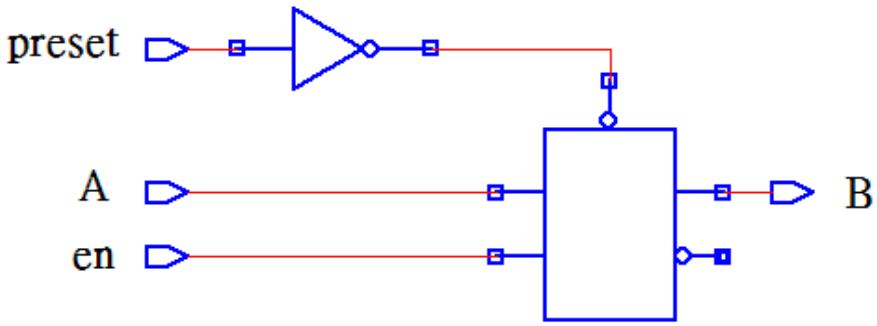
Watch Out for Unintentional Latches (1/4)

```
module latch_if1(en, A, out);
input en, A;
output out;
reg out;

always @(en) begin
  if (en)
    out = A;
end

endmodule
```

```
module latch_4(en, preset, A, B);
input en, preset, A;
output B;
reg B;

always @(en or preset or A) begin
  if (preset)
    B = 1;
  else if (en)
    B = A;
end

endmodule
```



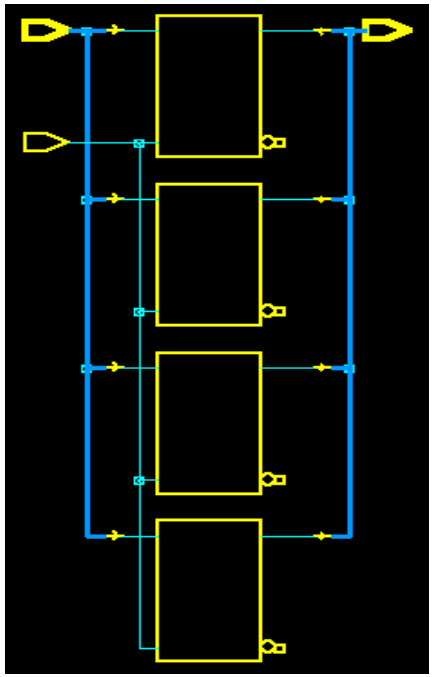
If en ==1 out = A
else out (new) = out (old)



Watch Out for Unintentional Latches (2/4)

```
Module Latch (In, Enable, Out);
input  Enable;
input  [3:0] In;
output [3:0] Out;

always @(In or Enable) begin
  if(Enable)
     Out=In;
end

endmodule
```

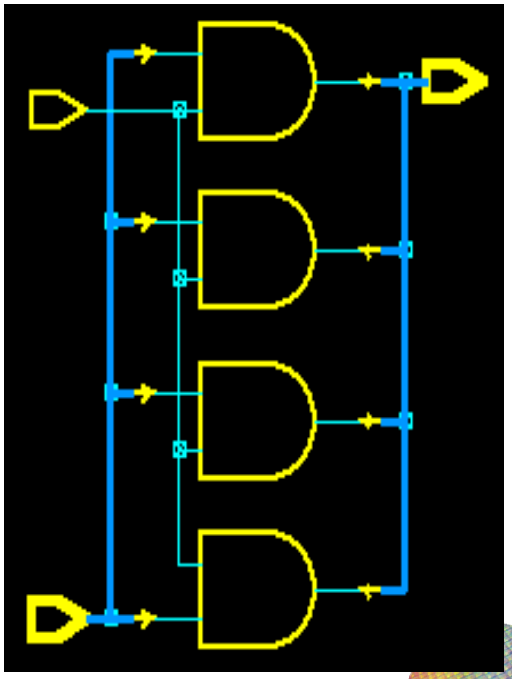
If Enable ==1
Out (new) = In
If Enable==0
Out (new) = Out (old)



```
Module Latch (In, Enable, Out);
input  Enable;
Input  [3:0] In;
output [3:0] Out;

always @(In or Enable) begin
  if(Enable)
    Out=In;
  else
    Out=0;
end

endmodule
```

```
Out=0;
if(Enable)
      Out=In;
```

No latch inference



Watch Out for Unintentional Latches (3/4)

Watch Out for Unintentional Latches

- Completely specify all clauses for every ***case*** and ***if*** statement
- Completely specify all output for every clause of each ***case*** or ***if*** statement
- Fail to do so will cause latches or flip-flops to be synthesized

Missing Case

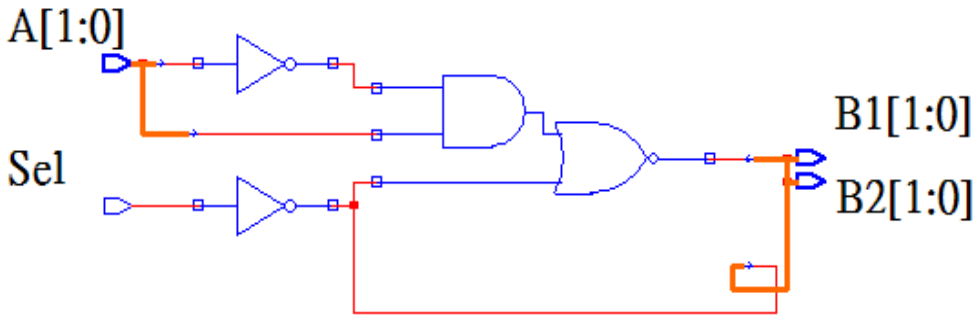
```
always @(d) begin
  case(d)
  2'b00: z = 1'b0;
  2'b01: z = 1'b0;
  2'b10:          ;
  endcase
end
```

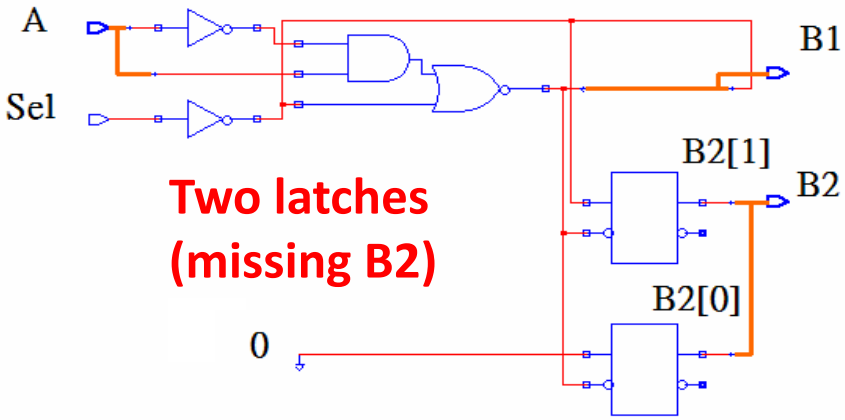
Missing Outputs

Watch Out for Unintentional Latches (4/4)

```
module code3(Sel, A, B1, B2);
Input  Sel, [1:0] A;
output [1:0] B1,  B2;
reg    [1:0] B1,  B2;
always @ (Sel or A) begin
  if(Sel) begin
    if(A == 1) begin
      B1 = 0; B2 = 0;    end
    else begin
      B1 = 1; B2 = 1;    end
  end
  else begin
    B1 = 2;  B2 = 2;     end
end
endmodule
```

```
module code4(Sel, A, B1, B2);
Input  Sel, [1:0] A;
output [1:0] B1,  B2;
reg    [1:0] B1,  B2;
always @ (Sel or A) begin
  if(Sel) begin
    if(A == 1) begin
      B1 = 0; B2 = 0;    end
    else begin
      B1 = 1;            end
  end
  else begin
    B1 = 2;  B2 = 2;     end
end
endmodule
```





Blocking vs. Non-Blocking (1)

- Blocking assignment (=) are order sensitive
- Non-Blocking assignment (<=) are order independent
- Use blocking assignment for combinational circuits and non-blocking assignment for sequential circuits

Blocking assignment

```
initial begin
  a=#12 1;
  b=#3  0;
  c=#2  3;
end
```

Non-Blocking assignment

```
initial begin
  d<=#12 1;
  e<=#3  0;
  f<=#2  3;
end
```

Time-unit	a	b	c	d	e	f
0	x	x	x	x	x	x
2	x	x	x	x	x	3
3	x	x	x	x	0	3
12	1	x	x	1	0	3
15	1	0	x	1	0	3
17	1	0	3	1	0	3

Blocking vs. Non-Blocking (2)

Blocking assignment

```
Initial begin
  ..
  A=1;
  B=0;
  ..
  A=B; // B=0 is used
  B=A; // A=0 is used
```

```
Initial begin
  ..
  A=1;
  B=0;
  ..
  B=A; // A=1 is used
  A=B; // B=1 is used
```

Non-Blocking assignment

```
Initial begin
  ..
  A=1;
  B=0;
  ..
  A<=B; // B=0 is used
  B<=A; // A=1 is used
```

```
Initial begin
  ..
  A=1;
  B=0;
  ..
  B<=A; // A=1 is used
  A<=B; // B=0 is used
```

Blocking vs. Non-Blocking (3)

Blocking assignment

```
module test_n(clk, a,
 b, c, out);
input clk, a, b, c;
output out;
reg t1, t2;
reg out;
always @(posedge clk) begin
  t1  = a&b;                    ①
  t2  = t1&c;   assigned in order  ②
  out = t1 & t2;                ③
end
endmodule
```

Blocking assignment

Non-Blocking assignment

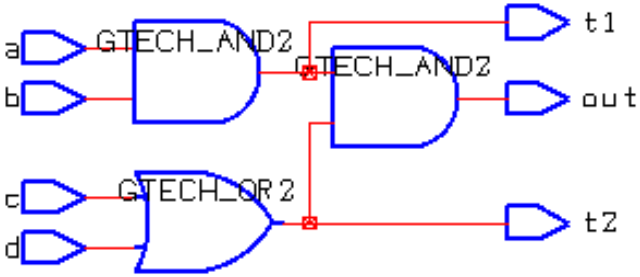
```
module test_n(clk, a, b,
 c, out);
input clk, a, b, c;
output out;
reg t1, t2;
reg out;
always @(posedge clk) begin
 t1  <= a&b;                    ①
 t2  <= t1&c;   assigned immediately  ①
 out <= t1 & t2;                ①
end
endmodule
```

Non-blocking assignment

Blocking vs. Non-Blocking (in combinational circuit) (4)

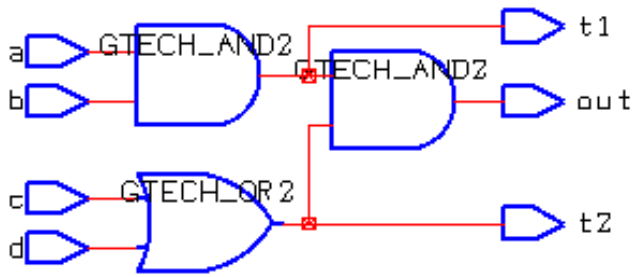
```
module test_n(a, b, c, d,
t1, t2, out);
input a, b, c, d;
output out, t1, t2;
reg t1, t2, out;
always @(a or b or c or d)
begin
  t1 = a&b;
  t2 = c | d;
  out = t1 & t2;
end
endmodule
```

Combinational circuit




```
module test_n(a, b, c, d,
t1, t2, out);
input a, b, c, d;
output out, t1, t2;
reg t1, t2, out;
always @(a or b or c or d)
begin
  t1 <= a&b;
  t2 <= c | d;
  out <= t1 & t2;
end
endmodule
```

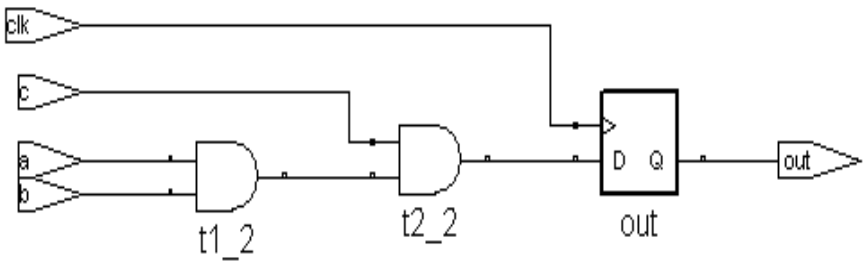
Combinational circuit



Blocking vs. Non-Blocking (Example) (5)

Blocking assignment

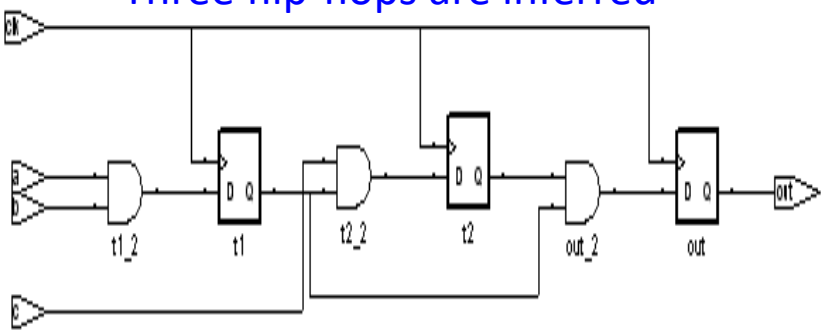
```
module test_n(clk, a, b,
c, out);
input clk, a, b, c;
output out;
reg t1, t2;
reg out;
always @(posedge clk) begin
  t1 = a&b;          ①
  t2 = t1&c;         ②
  out = t1 & t2;     ③
end  endmodule
```



Non-blocking assignment

```
module test_n(clk, a, b,
c, out);
input clk, a, b, c;
output out;
reg t1, t2; reg out;
always @(posedge clk) begin
   t1 <= a&b;          ①  // old t1 is used
   t2 <= t1&c;         ①  // old t1 and t2
   out <= t1 & t2;     ①     are used
end
endmodule
```

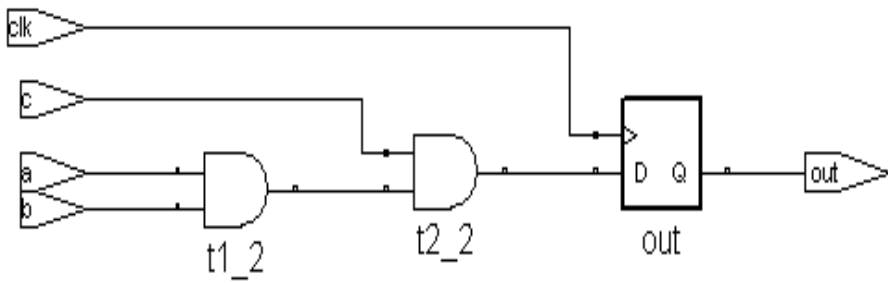
Three flip-flops are inferred



Blocking vs. Non-Blocking (Example) (6)

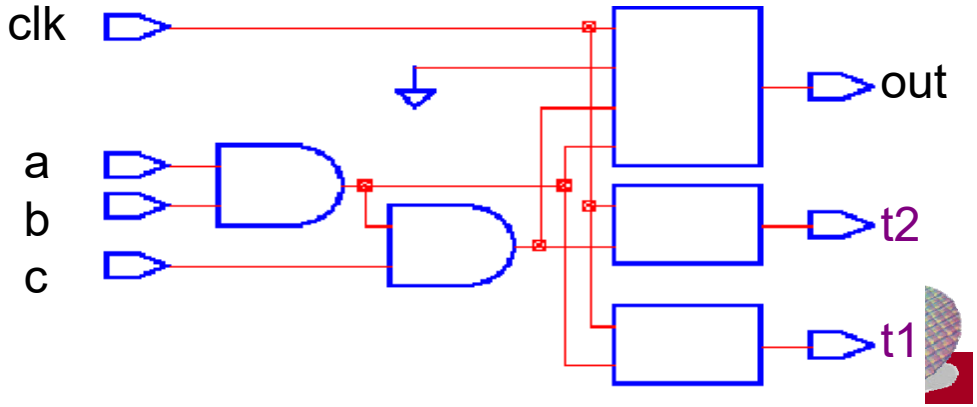
Blocking assignment

```
module test_n(clk, a, b,
c, out);
input clk, a, b, c;
output out;
reg t1, t2;
reg out;
always @(posedge clk) begin
 t1 = a&b;          ①
 t2 = t1&c;         ②
 out = t1 & t2;     ③
end
endmodule
```



Blocking assignment

```
module test_n(clk, a, b,
c, t1, t2, out);
input clk, a, b, c;
output out, t1, t2;
reg t1, t2;reg out;
always @(posedge clk) begin
  t1 = a&b;          ①
  t2 = t1&c;         ② // new t1 is used
  out = t1 & t2;     ③ // new t1 and t2
                        are used
end
endmodule
```

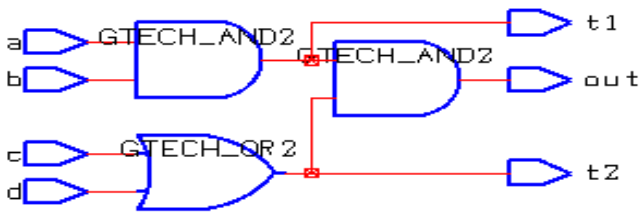


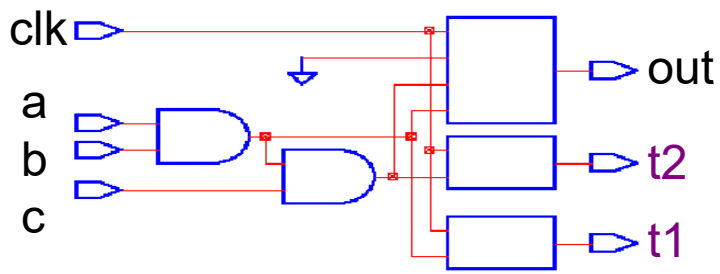
Blocking vs. Non-Blocking (7)

- Better to use nonblocking assignment in sequential circuits

```
module test_n(a, b, c, d,
t1, t2, out);
input a, b, c, d;
output out, t1, t2;
reg t1, t2, out;
always @(a or b or c or d)
begin
  t1 = a&b;
  t2 = c | d;
  out = t1 & t2;
end
endmodule
```

```
module test_n(clk, a, b,
c, t1, t2, out);
input clk, a, b, c;
output out, t1, t2;
reg t1, t2;reg out;
always @(posedge clk) begin
  t1 = a&b;
  t2 = t1&c;
  out = t1 & t2;
end
endmodule
```





Backup slides