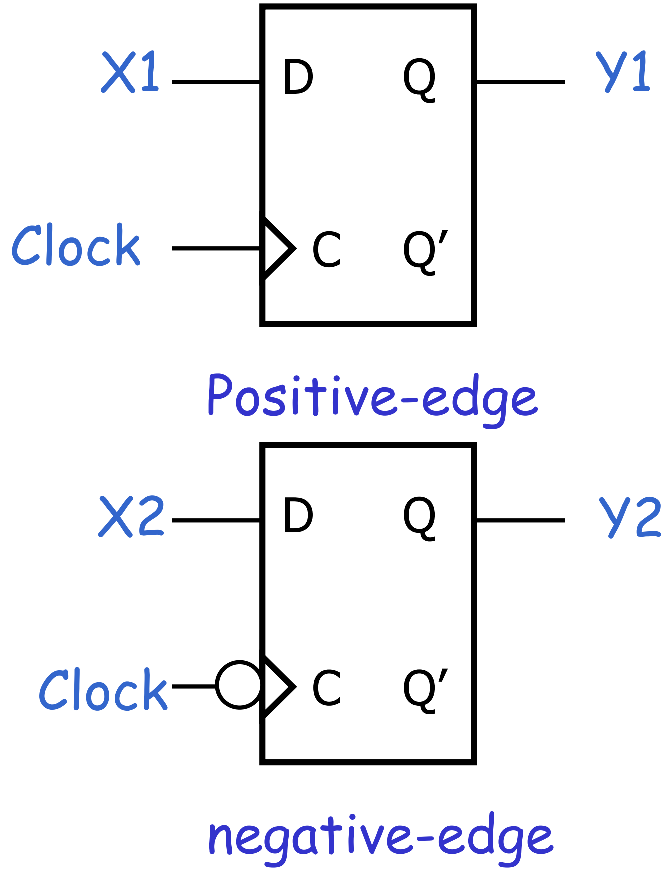
Notes for Sequential Circuits

Instructor : Pei-Yun Tsai

Flip-Flop Inference

- Wire (port) assigned in the synchronous section

```
module FF_PN (Clock, X1, X2, Y1, Y2);
    input  Clock;
    input  X1, X2;
    output  Y1, Y2;
    reg  Y1, Y2;

   always @(posedge Clock)
    Y1 <= X1;

   always @(negedge Clock)
    Y2 <= X2;

endmodule
```



Reset (1/2)

- Synchronous reset
  - The reset signal will only affect or reset the state of the flip-flop on the active edge of a clock.
  - Easy to synthesize
  - Problem
    - The reset could be a “late arriving signal” relative to the clock period, due to the high fanout of the reset tree.
    - The glitches occurring near the active clock edge make the flip-flop could go metastable.

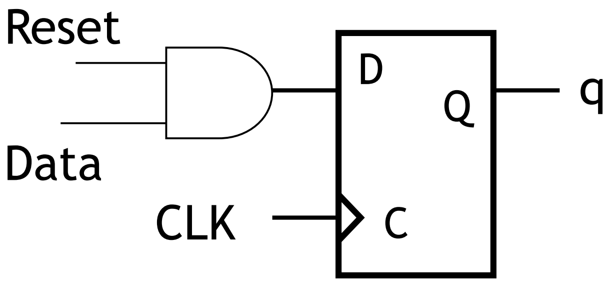
Reset (2/2)

- Asynchronous reset
  - Do not require a free-running clock
  - It is guaranteed not to have the reset added to the data path.
  - Problem
    - Asynchronous nature at the assertion and at the de-assertion If the asynchronous reset is released at or near the active clock edge of a flip-flop, the output of the flip-flop could go meta-stable and thus the reset state of the ASIC could be lost.
    - The spurious reset may be generated due to noise or glitches on the board or system reset
  - Asynchronous reset is more popular so far.

Modeling Reset

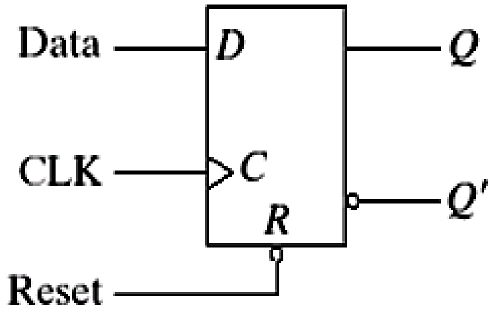
- Synchronous reset

```
always@(posedge CLK)
if (!Reset) q<=1'b0;
else q<=Data;
```



- Asynchronous reset

```
always@(posedge CLK or
negedge Reset )
  if (!Reset) q<=1'b0;
  else q<=Data;
```



Flip-Flop with Set/Reset

- Asynchronous set and reset

```
module dff3(q,d,clk,rst_n,set_n);
output reg q,
input d, clk, rst_n, set_n
always @(posedge clk or negedge rst_n or negedge set_n)
    if (!rst_n) q <= 1'b0; // asynchronous reset
    else if (!set_n) q <= 1'b1; // asynchronous set
    else q <= d;

endmodule
```

Flip-Flop with Load

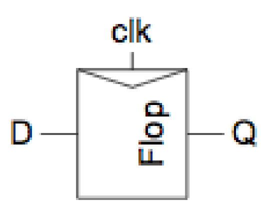
- Synchronous load

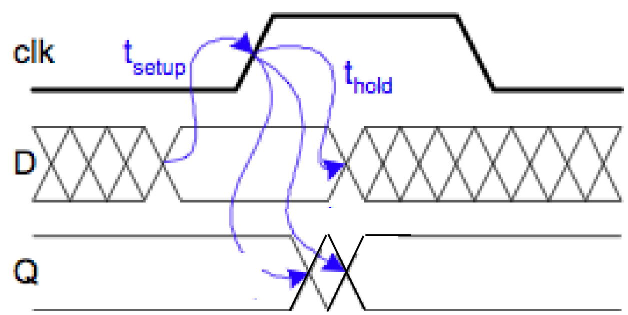
```
module dff4 (q,co,d,ld,rst_n,clk);
output reg [7:0] q,
output reg co;
input [7:0] d;
input ld, rst_n, clk

always @(posedge clk or negedge rst_n)
    if (!rst_n) {co,q} <= 9'b0; // async reset
    else if (ld) {co,q} <= d; // sync load
    else {co,q} <= q + 1'b1; // sync increment

endmodule
```

Timing Requirement

- Gate delay, setup time, and hold time

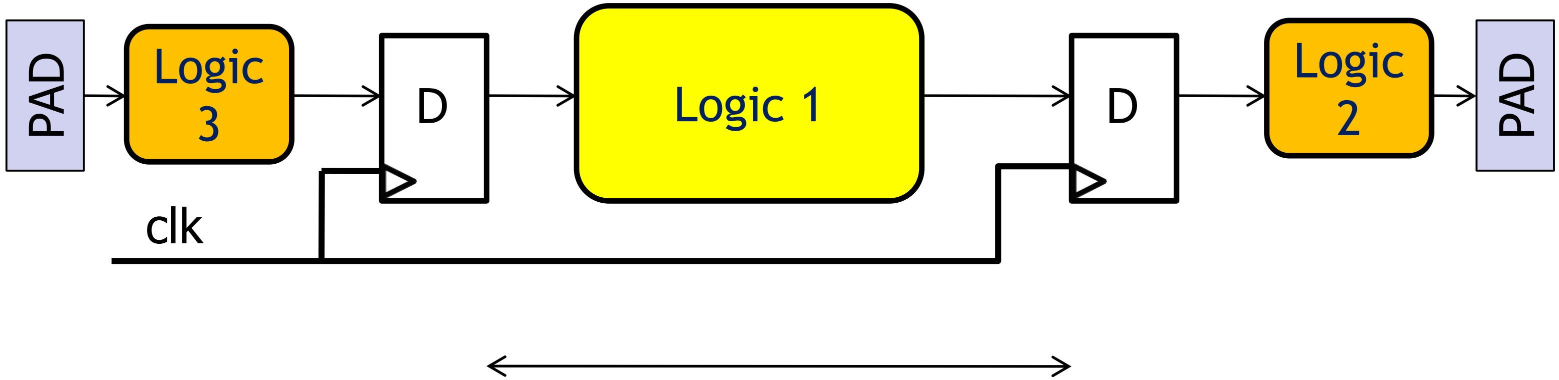
t_{PLH}	Propagation Delay from Low to High
t_{PHL}	Propagation Delay from High to Low
t_{Setup}	Setup Time
t_{Hold}	Hold Time





Propagation Delay

Maximum Operating Frequency



C2Q delay+ gate delay (wire delay) + setup time
=critical path delay

Maximum operating frequency=1/critical path delay

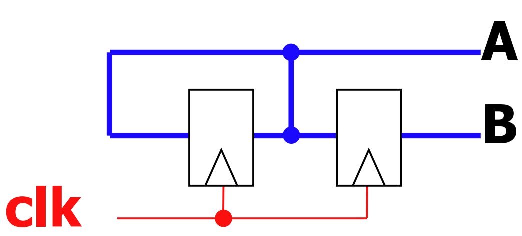
Blocking and Non-blocking Assignments

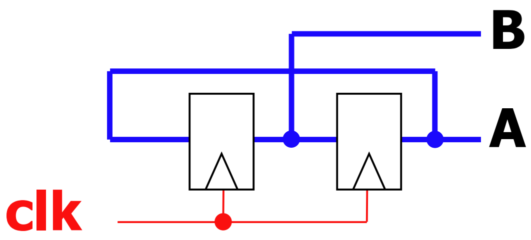
Bad: Circuit from blocking assignment.

Good: Circuit from nonblocking assignment.

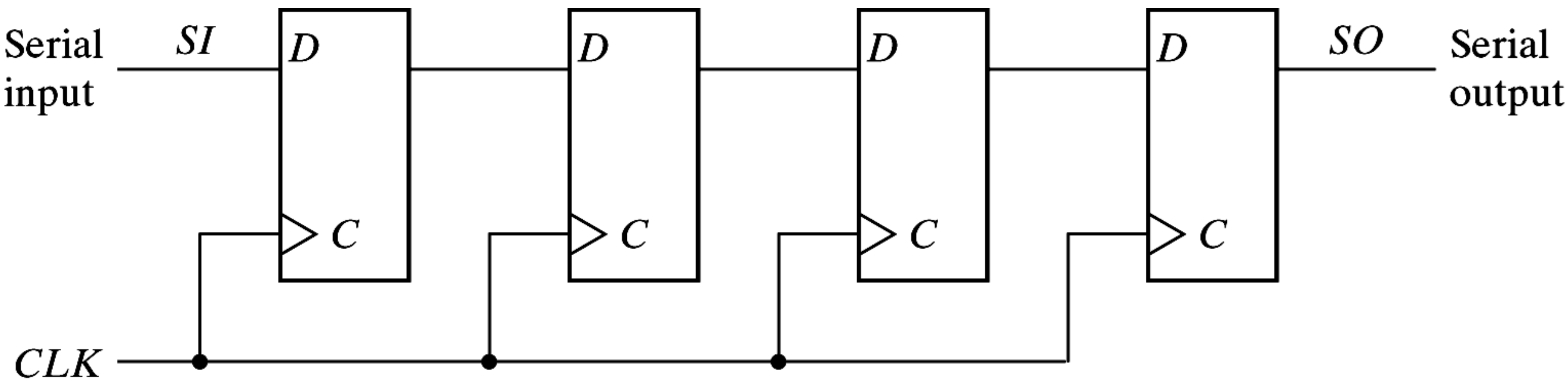
```
always @(posedge clk)
begin
    b=a;
    a=b;
end
```

```
always @(posedge clk)
begin
    b<=a;
    a<=b;
end
```





The Simplest Shift Register



HDL Modeling for Shift Reg.

- Blocking assignment

```
assign SO = D;
always @(posedge CLK) begin
  if (reset) begin
    A = 0;  B = 0; C = 0; D = 0;
  end
  else begin
    D = C;
    C = B;
    B = A;
    A = SI;
  end
end
```

order dependent

- Non-blocking assignment

```
assign SO = D;
always @(posedge CLK) begin
  if (reset) begin
    A <= 0;  B <= 0; C <= 0; D <= 0;
  end
  else begin
    A <= SI;
    C <= B;
    D <= C;
    B <= A;
  end
end
```

can be any order

More Examples

```
// Blocking assignment:
initial
begin
    CLR = #5 1;
    CLR = #4 0;
    CLR = #10 1;
end

/* CLR is assigned 1 at time 5,
0 at time 9, and 1 at time 19 */
```

```
// Non-blocking assignment:
initial
begin
    CLR <= #5 1;
    CLR <= #4 0;
    CLR <= #10 1;
end

/* CLR is assigned 0 at time 4,
1 at time 5, and 1 at time 10 */
```

* Value is undetermined if multiple values are assigned at the same time.

Inconsistent Results

- The following situations may cause simulation to disagree with synthesis
  - Incomplete sensitivity list
    - Sensitivity list is ignored in synthesis routines
  - Code with delays
    - The specified delay values are also ignored
  - Non-local reference within a function
  - Order dependency of concurrent statements
  - Comparisons to X or Z
- Those cases should be avoided at all

Parameters

- Parameters are used to define a constant which can be changed for each compilation.
- Common usage
  - Specify delays and widths
    - module Adder(cout, s, cin, a, b)

      parameter Input_Size=5;

      // Input port declaration

      input [Input_Size-1:0] a,b;

Adder#(8) Myadder(.cout(C1),.s(S1),.cin(C0),.a(A1),.b(B1));