Chapter 4

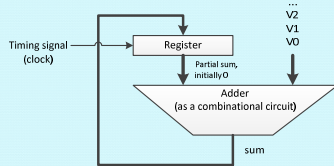## SEQUENTIAL CIRCUITS:
CORE MODULES

In this Chapter

- **Basic modules for saving sequential circuit's state**
  - **Latches and flip flops**
- **Timing constraints**
- **Other flip-flops**
- **HDL models**

# Sequential Circuit Example

- **Register retains state of circuit as *sum***
- **Adder computes next *sum* (circuit's next state)**
- **Timing signal must guarantee *sum* is computed before register loading**

...
V2
V1
V0

Timing signal (clock) → Register

Partial sum, initially 0

Adder (as a combinational circuit)

sum

SR latch core circuit

- **Three basic functions:**
  - **retain previous value (keeps *q* as is)**
  - **store 0 (reset *q*)**
  - **store 1 (set *q*)**
- **However, four possible cases, only three used**
  1. *s* = 0 and *r* = 0, retain q
  2. *s* = 0 and *r* = 1, reset
  3. *s* = 1 and *r* = 1, set
  4. *s* = 1 and *r* = 1, not used
     - makes $q = \bar{q}$, invalid
     - causes *q* and $\bar{q}$ oscillate when both *s* and *r* instantly become 0 next

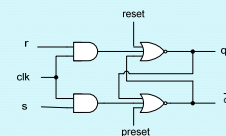| s	r	q^{t+1}
0	0	q^t
0	1	0
1	0	1
1	1	–
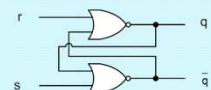
(a)   (b)

SR Latch
(added clock signal)

- **Goal: Prevent Case 4 (*s* = 1 and *r* = 1 affecting core circuit)**
- **Includes a signal gating logic to time sampling of inputs (*s* and *r*)**
  - **Only when both *s* and *r* are both stable (not changing) should affect core circuit**
- **Naming convention**
  - **positive-level, inputs are sampled when *clk* = 1**
  - **negative-level, inputs are sampled when *clk* = 0**
- **Assume positive-level (figure below)**
  - **clk = 0, core retains *q* value**
  - **clk = 1, *s* and *r* gated (sampled), enter the core**
  - **Timing constraint: *s* and *r* must stabilize at or before *clk* changes to 1 (sampling level)**
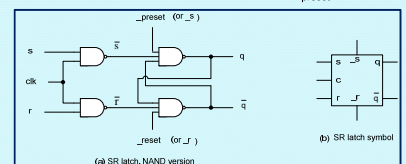   - ***s* and *r* called synchronous inputs**

Core circuit

Signal gating

r → G1

clk

s → G2

q

$\bar{q}$
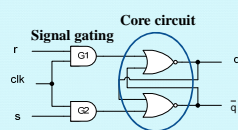
Asynchronous Inputs

- **Function: Changes *q* and $\bar{q}$ independent of *clk***
- **Used for circuit initialization during startup**

reset

r

clk

s

q

$\bar{q}$

preset

SR latch, NAND Version And Symbol

_preset (or _s)

s

clk

r

$\bar{s}$

$\bar{r}$

q

$\bar{q}$

_reset (or _r)

(a) SR latch, NAND version

s   _S   q
c
r   _r $\bar{q}$

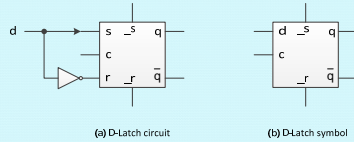(b) SR latch symbol

D Latch

- Requires one (synchronous) input *d* instead of two in SR latch
- Reduces number of signal routing wires, saves space
- Timing constraint: *d* must stabilize <u>before</u> *clk* changes to 1

(a) D-Latch circuit   (b) D-Latch symbol

Disadvantage of Latches

- Signal chasing problem
  - Input *d1* can change values of both *q1* and *q0* during the same *clk* cycle
  - *d1 s*hould only affect *q1*
- Limited application: Cannot be used to design circuits when *d* dependents on a *q* (or $\overline{q}$)
 - E.g., shift register
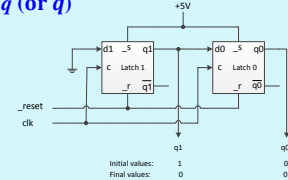
D Flip-Flop

- No signal chasing problem
- Can use two latches, one positive-level and one negative-level
  - The two latches act like "double-door entry"
- Called edge triggered, *d* sampled at *clk* edge
  - Positive edge when *clk* makes 0-1 transition
  - Negative edge when *clk* makes 1-0 transition
- Timing constraint: *d* must remain stable <u>before</u> and <u>after</u> the sampling *clk* edge
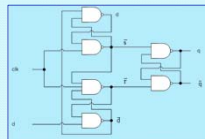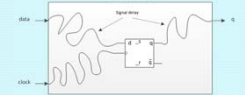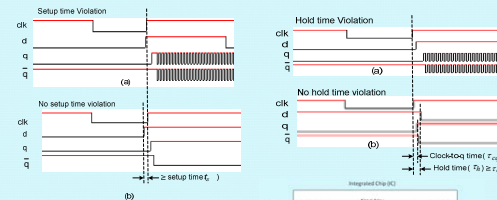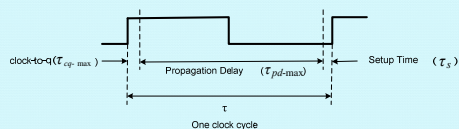
Three latches version, requires fewer total gates
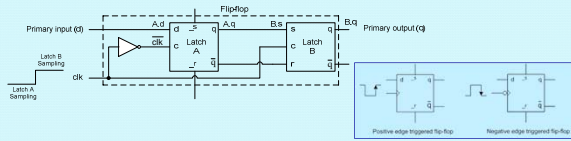
Setup and Hold times

- Set-up time: Minimum time *d* must be stable before *clk* sampling edge
- Hold time: Minimum time *d* must remain stable after *clk* sampling edge
  - May be negative due to difference in *d* and *clk* wire delays

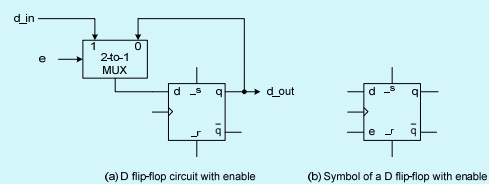# Clock Frequency Estimation – Without Clock Skew

Minimum clock frequency, not including wire delays

$$\tau \geq \tau_{cq-max} + \tau_{pd-max} + \tau_{st} \qquad f \leq \frac{1}{\tau} \text{ cycles/sec or Hertz}$$

D Flip-Flop with Enable

- Retains *q* when *e = 0*
- Used when one or more flip-flops selected among many

(a) D flip-flop circuit with enable   (b) Symbol of a D flip-flop with enable

2

Other flip-flops

- **JK flip-flop**
  - **Uses two inputs called *j* and *k***
  - ***j = k = 1* will not cause oscillation**
  - **Better when wires require less space than gates**
    - **Not necessarily true today**
- **T (toggle) flip-flop**
  - ***q* alternates every clock cycle when *t = 1***
  - **E.g., consider when *t = 1*, *q* controls a 2-to-1 MUX**
- **Both can be designed to operate edge triggered**

j	k	q(t+1)
0	0	q(t)
0	1	0
1	0	1
1	1	!q(t)

(a) JK Flip-flop

t	q(t+1)
0	q(t)
1	!q(t)

(b) T flip-flop
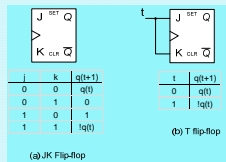
D Latch HDL Model (Behavioral)

```
module d_latch
(
        input clock, _reset, _preset, d,
        output reg q,
        output  nq
);

assign nq = ~q;

always@(clock or !_reset or !_preset or d)
begin
        if(!_reset)
                q <= 0;
        else if(!_preset)
                q <= 1;
        else if(clock)
                q <= d;
end
endmodule
```

Designed like a combinational circuit.

Except, note the missing else here (creates a D-latch)

"<=" called non-blocking or concurrent assignment

D flip-flop HDL Model (Behavioral)

```
module dff
(
        input clock, _reset, _preset, d, e,
        output reg q,
        output nq
);

assign nq = ~q;

always@(posedge clock, _reset, _preset)
begin
        if(!_reset)
                q <= 0;
        else if(!_preset)
                q <= 1;
        else if(e)
                q <= d;
end
endmodule
```

Note the missing synchronous inputs *d* and *e* here

Note also the missing else here (creates a D flip-flop)