CSCE 312: Computer Organization

David Kebo Houngninou

Digital Logic Design

Sequential Systems

Combinational logic circuits:

Output values only depend on present input values.

Sequential logic circuits:

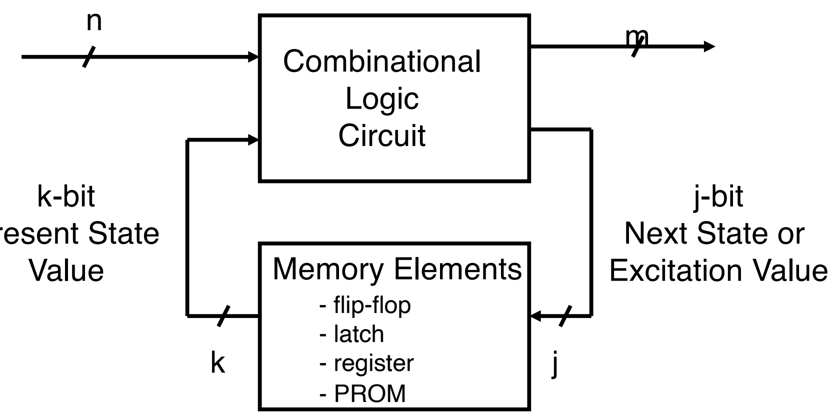
Output values depend on input values and the state.

Sequential circuits must have a way to retain states with memory devices.

Sequential logic circuits can be synchronous or asynchronous:

- A synchronous sequential circuit uses a clock signal to control the changes between states.
- An asynchronous sequential circuit does not use a clock.

Sequential System Diagram


n
m
Combinational Logic Circuit
k-bit Present State Value
j-bit Next State or Excitation Value
Memory Elements
- flip-flop
- latch
- register
- PROM
k
j

Mealy finite state machine

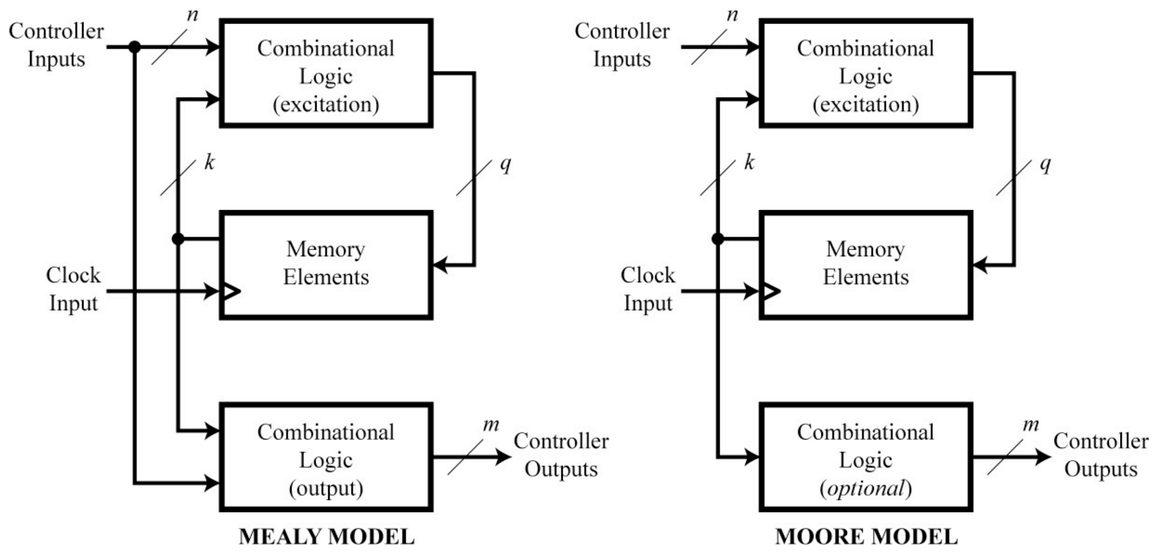
The m outputs depend on k present state bits and n inputs.

Moore finite state machine

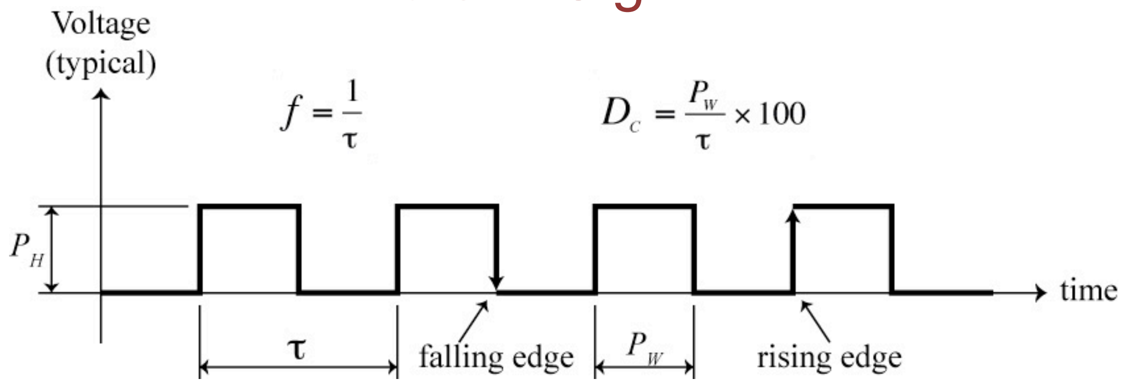
The m outputs only depend on k present state bits

Remember: *Moore is Less!*

Controller Block Diagrams



Clock Signal



τ: period (in seconds)

f: frequency (in Hertz) = $1/\tau$

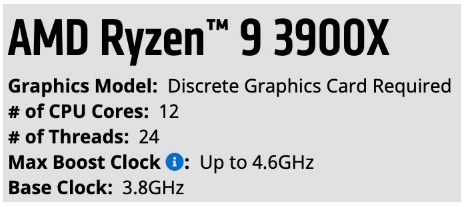
duty cycle: ratio of pulse width to period (%)

$D_C = P_w / \tau$

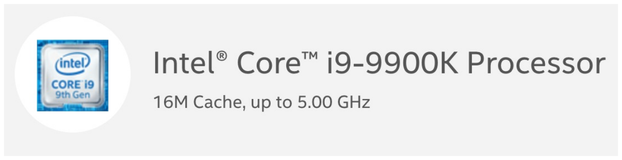
P_w : pulse width (in seconds)

millisecond (ms) 10^{-3}	Kilohertz (kHz) 10^{3}
microsecond (µs) 10^{-6}	Megahertz (MHz) 10^{6}
nanosecond (ns) 10^{-9}	Gigahertz (GHz) 10^{9}

Microprocessor clock frequency


AMD Ryzen™ 9 3900X

Graphics Model: Discrete Graphics Card Required
# of CPU Cores: 12
# of Threads: 24
Max Boost Clock: Up to 4.6GHz
Base Clock: 3.8GHz


Intel® Core™ i9-9900K Processor

16M Cache, up to 5.00 GHz

A microprocessor frequency is the internal frequency of CPU core.

The higher the frequency, the faster the processor is!

Another factor of performance is the number of Instructions Per Clock that the CPU can process (IPC).

The clock frequency and the IPC give us the total number of instructions per second that the CPU can process:

# of instructions/second = Frequency * IPC

Clock Signal Example

What is the pulse-width of a 4.77 MHz clock with a 30% duty cycle?

Clock Signal Example

What is the pulse-width of a 4.77 MHz clock with a 30% duty cycle?

$\tau = 1/f = (4.77\times10^{6})^{-1} = 2.096 \times10^{-7} = 210$ ns

Clock Signal Example

What is the pulse-width of a 4.77 MHz clock with a 30% duty cycle?

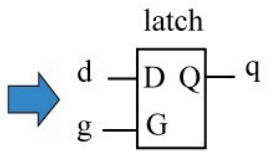
$\tau = 1/f = (4.77\times10^{6})^{-1} = 2.096 \times10^{-7} = 210$ ns

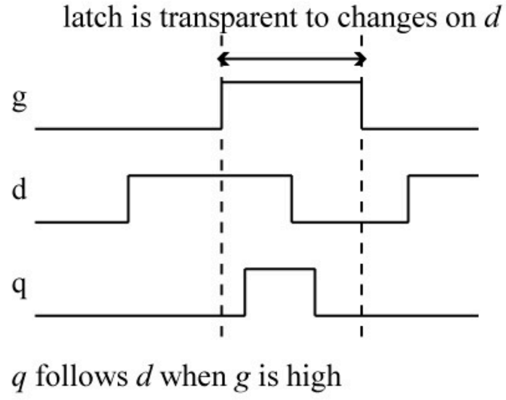
Pw = (duty cycle) $\times\ \tau$ = (0.3) $\times$ (210 ns) = 63 ns

Storage Elements

(a) level-sensitive storage element (D-latch)

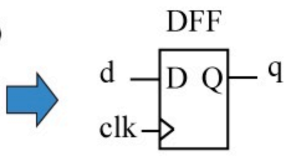
```
always @(g or d)
begin
  if (g) q <= d;
end
```

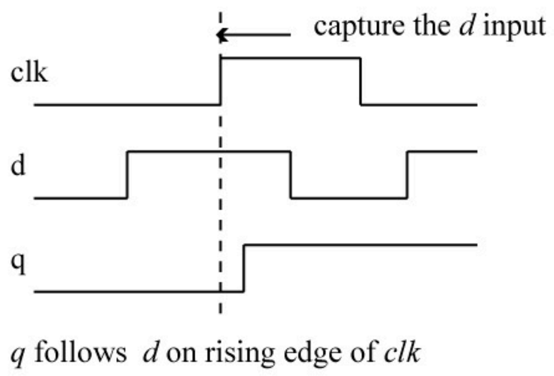



latch is transparent to changes on *d*
g
d
q
q follows *d* when *g* is high

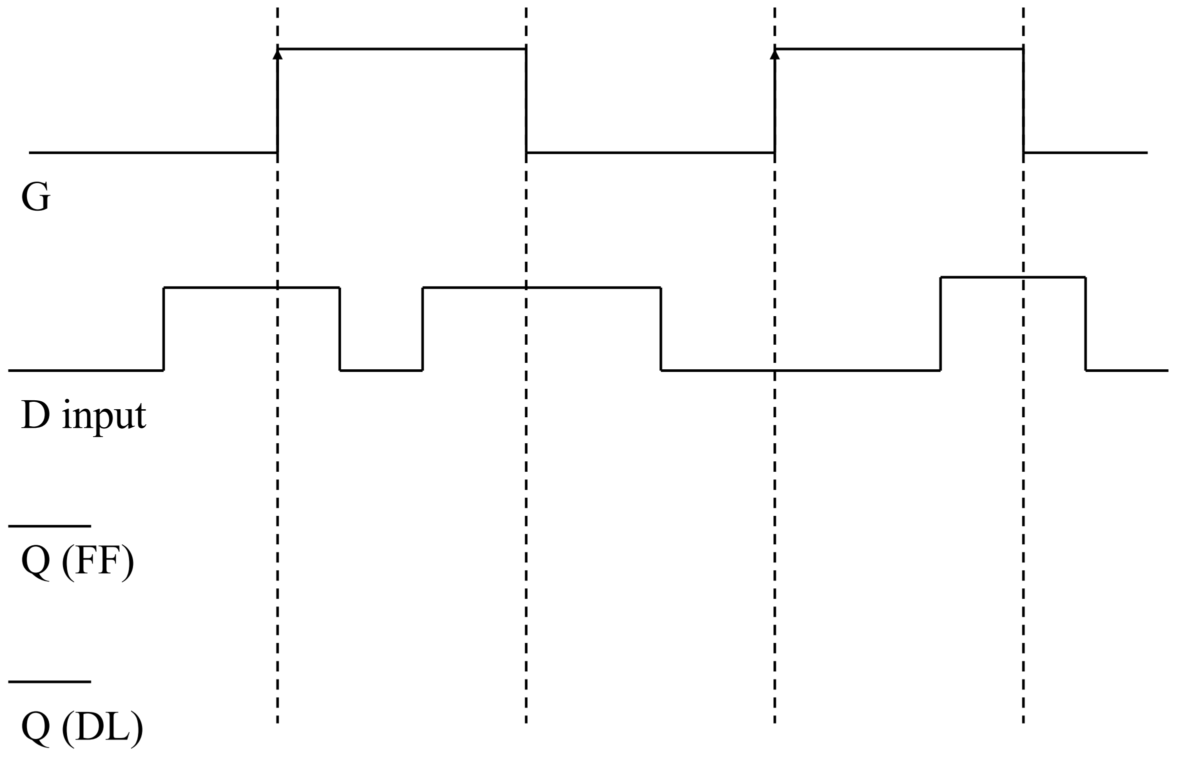
(b) edge-triggered storage element (data flip-flop, or DFF)

```
always @(posedge clk)
begin
  q <= d;
end
```

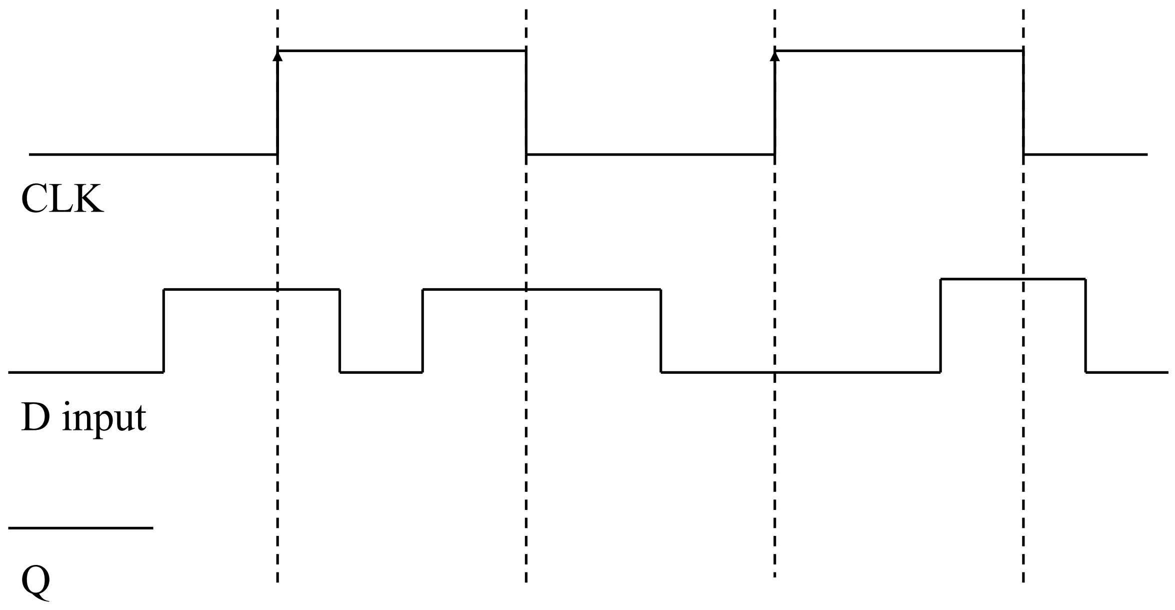




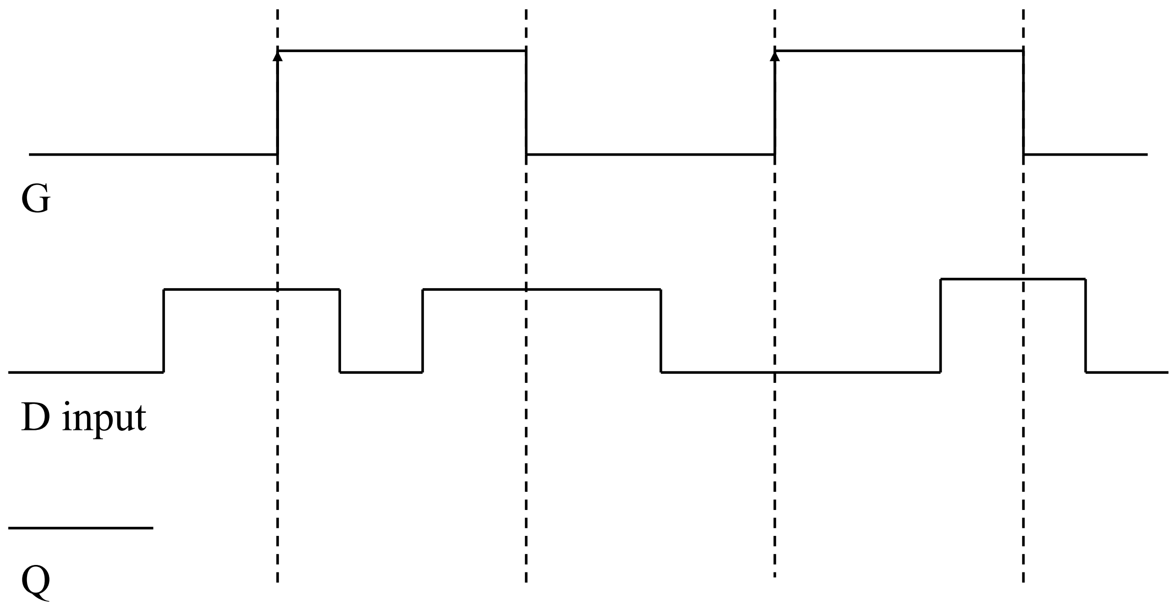
D-Latch operation



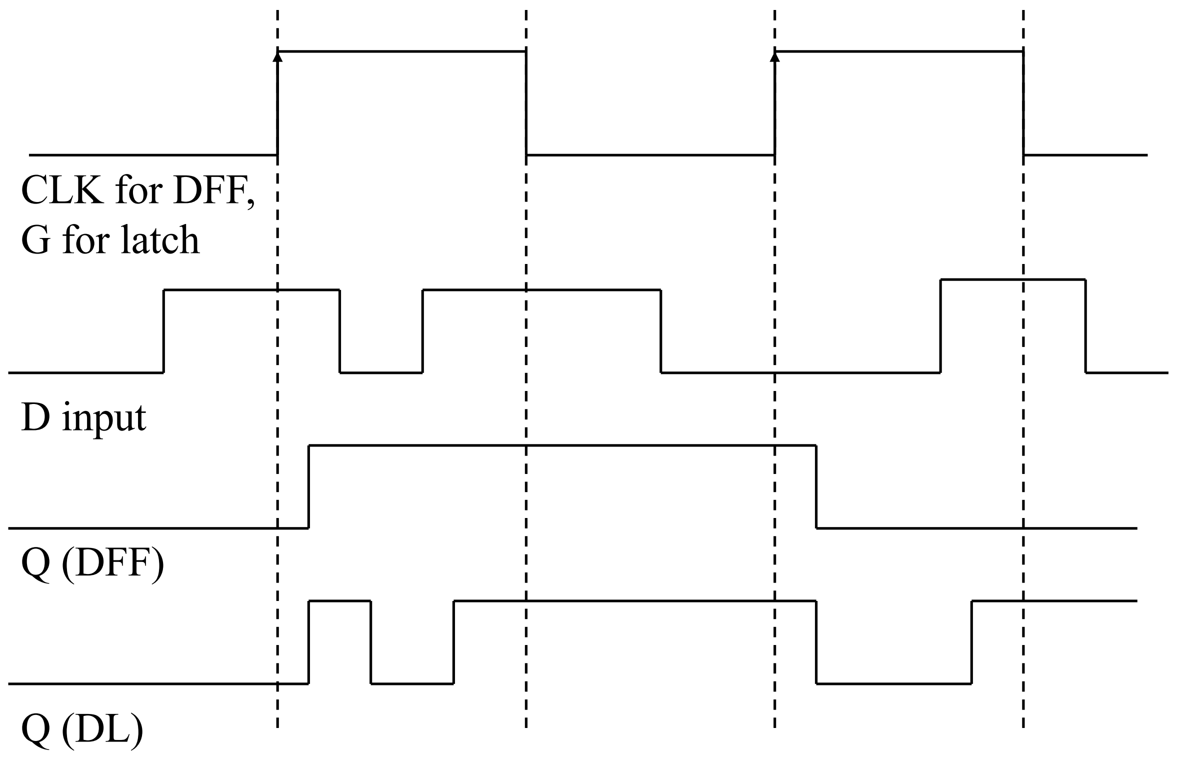
DFF operation



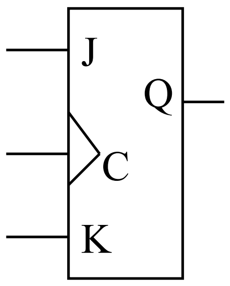
DFF operation



D FF, D-Latch operation



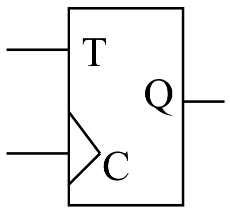
Other State Elements



J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q'(t)

JK can be useful for single bit flags with separate set(J), reset(K) control.

RARELY USED



T	Q(t+1)
0	Q(t)
1	Q'(t)

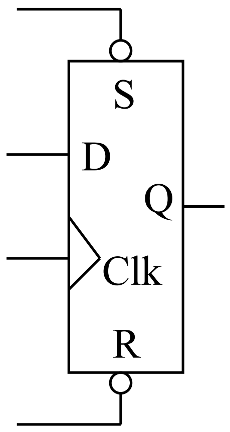
Can be useful for asynchronous counter design.

RARELY USED

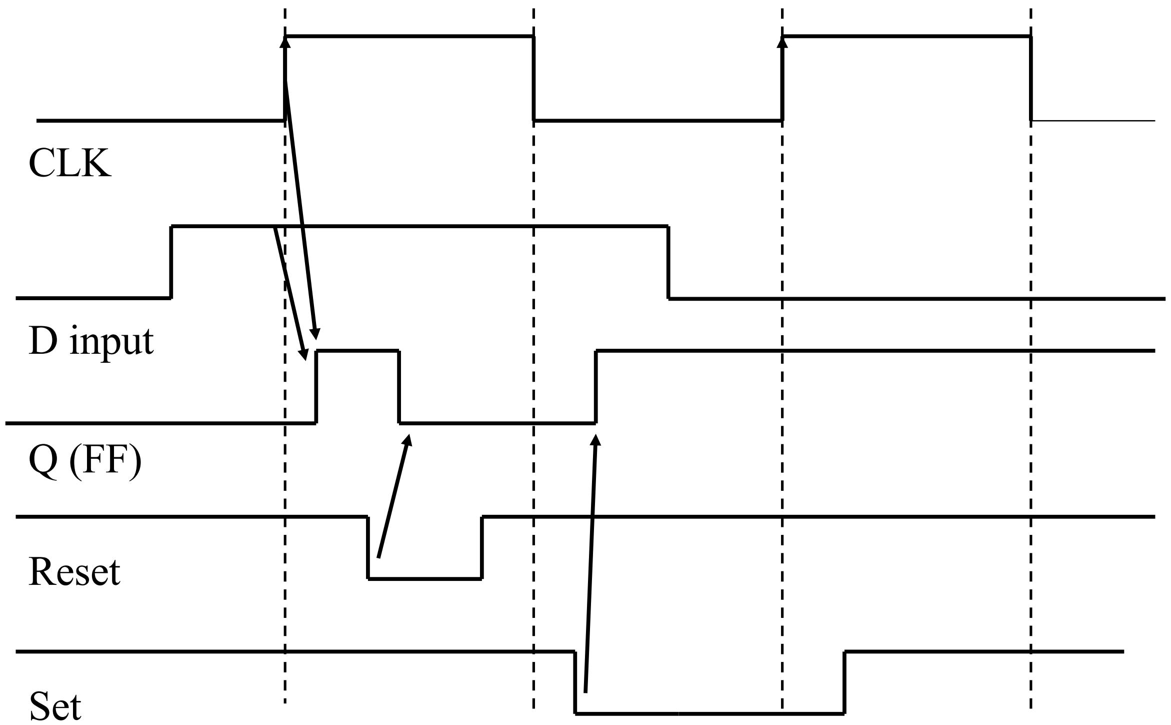
Synchronous vs Asynchronous inputs

Synchronous input: Output changes after active clock edge
Asynchronous input: Output changes independently of clock

- State elements often have an asynchronous set and reset control.
- The D input is synchronous with respect to Clk.
- S and R are asynchronous inputs. S and R can change the output Q independently of Clk.
- Asynchronous inputs are dominant over Clk.

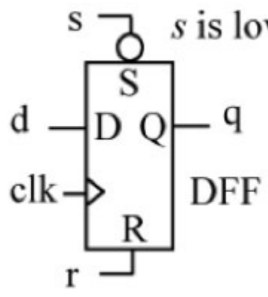


D FF with async control



D-Flip-Flop with Async. Set/Res

(a) DFF with asynchronous Set/Reset

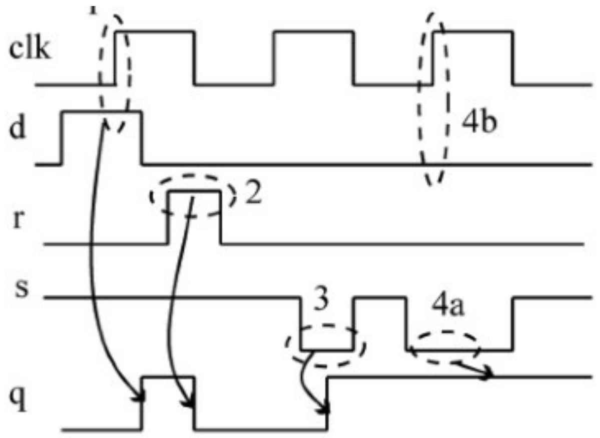


s is low-true, *r* is high-true

r	*s*	*clk*	*q*
1	x	x	0
0	0	x	1
0	1	↑	d
0	1	1 or 0	q_{old}

```
always @(posedge clk or
         posedge r or
         negedge s)
begin
  if (r) q <= 1'b0;
  else if (!s) q <= 1'b0;
  else q <= d;
end
```

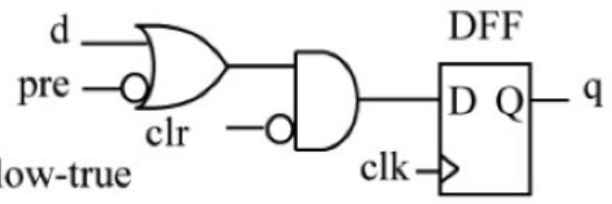
D-Flip-Flop with Async. Set/Res



1. *d* input is captured on rising clock edge as *r*, *s* are negated; *q* becomes '1'.
2. *r* input is asserted; *q* becomes '0'.
3. *s* input is asserted; *q* becomes '1'.
4. assertion of *s* input (a) overrides clock input (b), so *q* output remains as '1'.

D-Flip-Flop with Sync. Set/Res

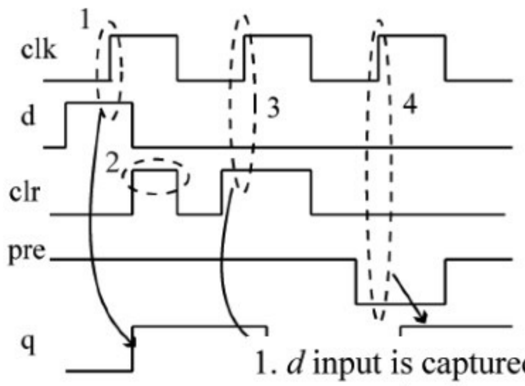
(b) DFF with synchronous preset/clear



pre is low-true
clr is high-true
clr has priority over *pre*

```
always @(posedge clk)
begin
  q <= d;  //lowest priority
  if (!pre) q <= 1'b1;
  if (clr) q <= 0'b0; //highest priority
end
```

D-Flip-Flop with Sync. Set/Res



1. *d* input is captured on rising clock edge as *clr*, *pre* are negated; *q* becomes '1'.
2. assertion of synchronous input has no effect if does not occur on the active clock edge
3. *clr* input is asserted on rising clock edge; *q* becomes '0'.
4. *pre* input is asserted on rising clock edge; *q* becomes '1'.

FF Timing: Propagation Delay

- C2Q: Q will change some propagation delay after change in C. Value of Q is based on D input for DFF.
- S2Q, R2Q: Q will change some propagation delay after change on S input, R input
- Note that there is NO propagation delay D2Q for DFF!
- D is a Synchronous INPUT, no prop delay value for synchronous inputs

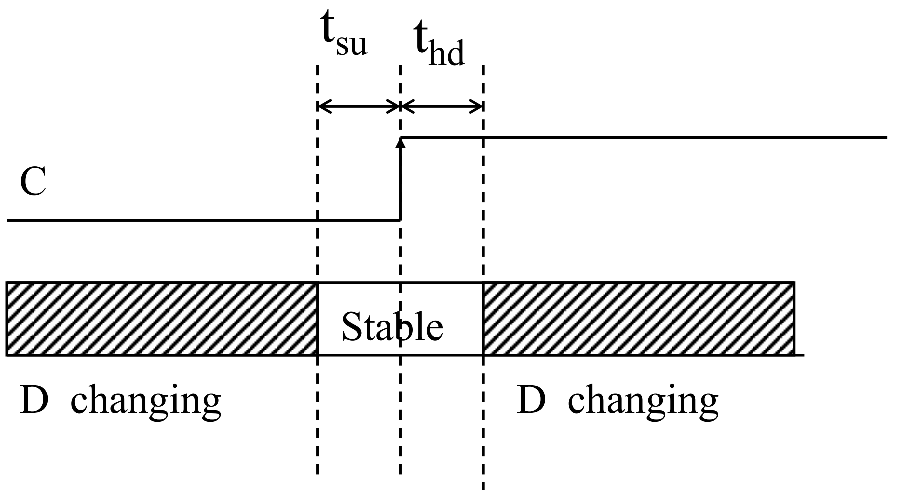
Setup, Hold Times

Synchronous inputs (e.g. D) have Setup, Hold time specification with respect to the CLOCK input

Setup Time: the amount of time the synchronous input (D) must be stable before the active edge of clock.

Hold Time: the amount of time the synchronous input (D) must be stable after the active edge of clock.

Setup, Hold Time



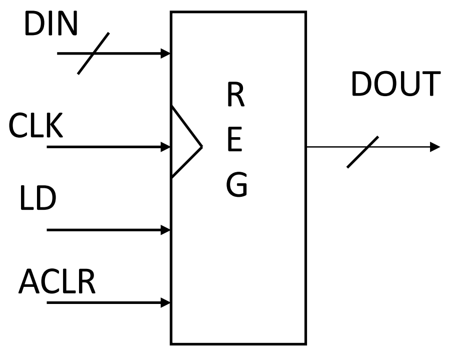
If changes on D input violate either setup or hold time, then correct FF operation is not guaranteed.

Setup/Hold measured around active clock edge.

Registers

The most common sequential building block is the register.

A register is n bits wide, and has a load line for loading in a new value into the register.



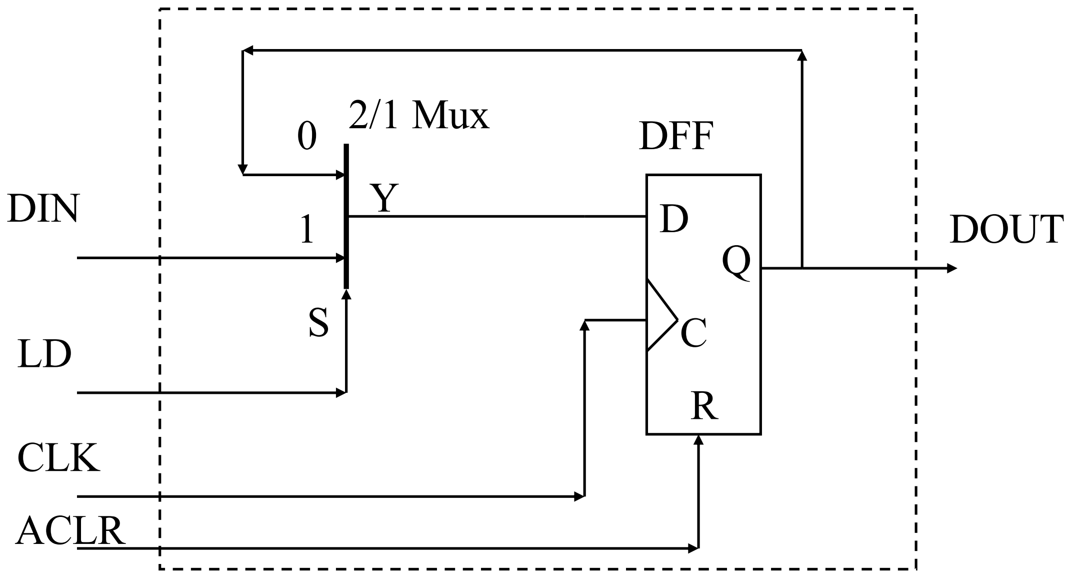
Register contents do not change unless LD = 1 on active edge of clock.

A DFF is NOT a register!

DFF contents change every clock edge.

ACLR used to asynchronously clear the register

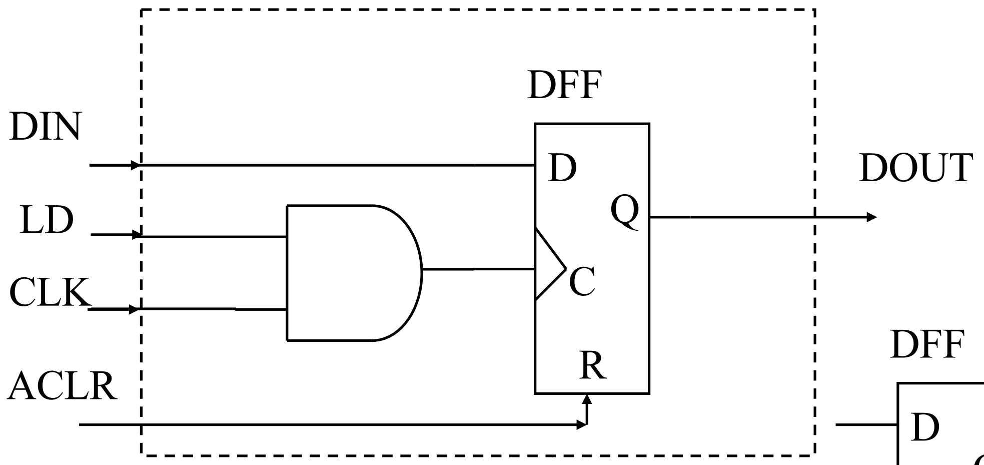
1 Bit Register using DFF, Mux

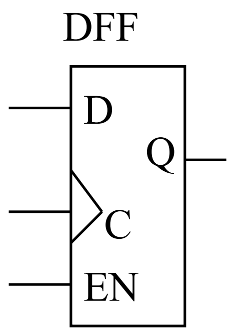


Note that the DFF simply loads an old value when LD = 0.

DFF is loaded every clock cycle.

1 Bit Register using Gated Clock

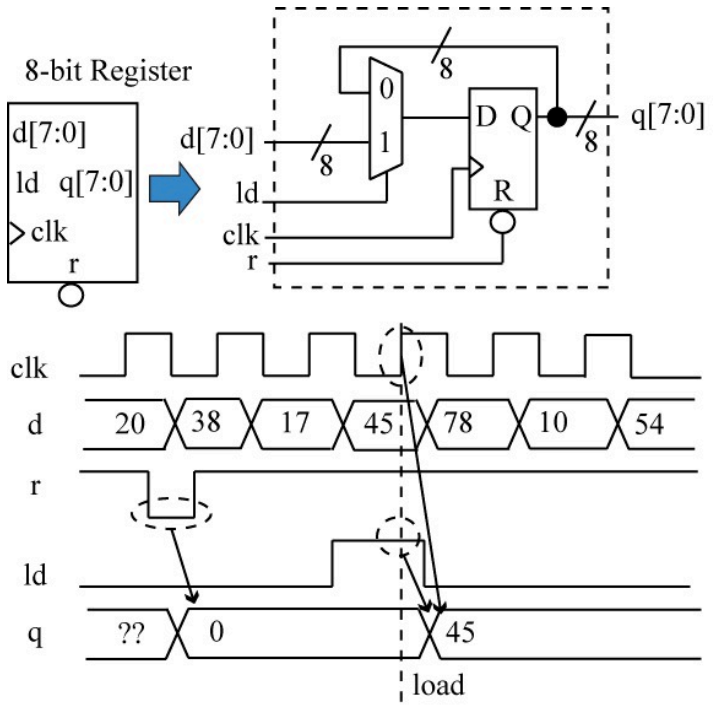




Saves power over previous design since DFF is not clocked every clock cycle.

Many FPGAs offer an 'enabled' DFF as an integrated unit.

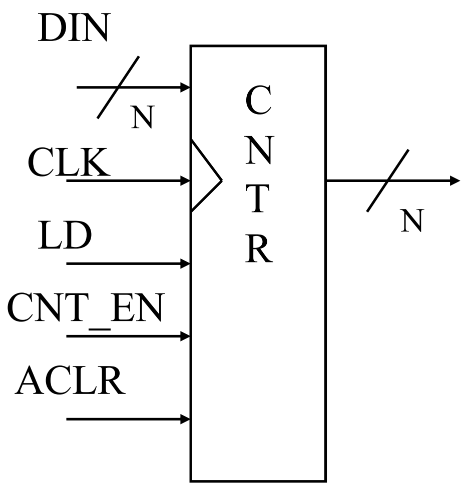
8-bit Register



```
module reg8bit(clk,r,d,ld,q);

input clk,r,ld;
input [7:0] d;
output [7:0] q;

reg [7:0] q;

//register
always @(posedge clk
         or negedge r)
begin
 //async reset
 if (!r) q <= 8'b0;
 else begin
 //synchronous inputs
  if (ld) q <= d;
 end
end

endmodule
```

Counter

Very useful sequential building block. Used to generate memory addresses, or keep track of the number of times a datapath operation is performed.

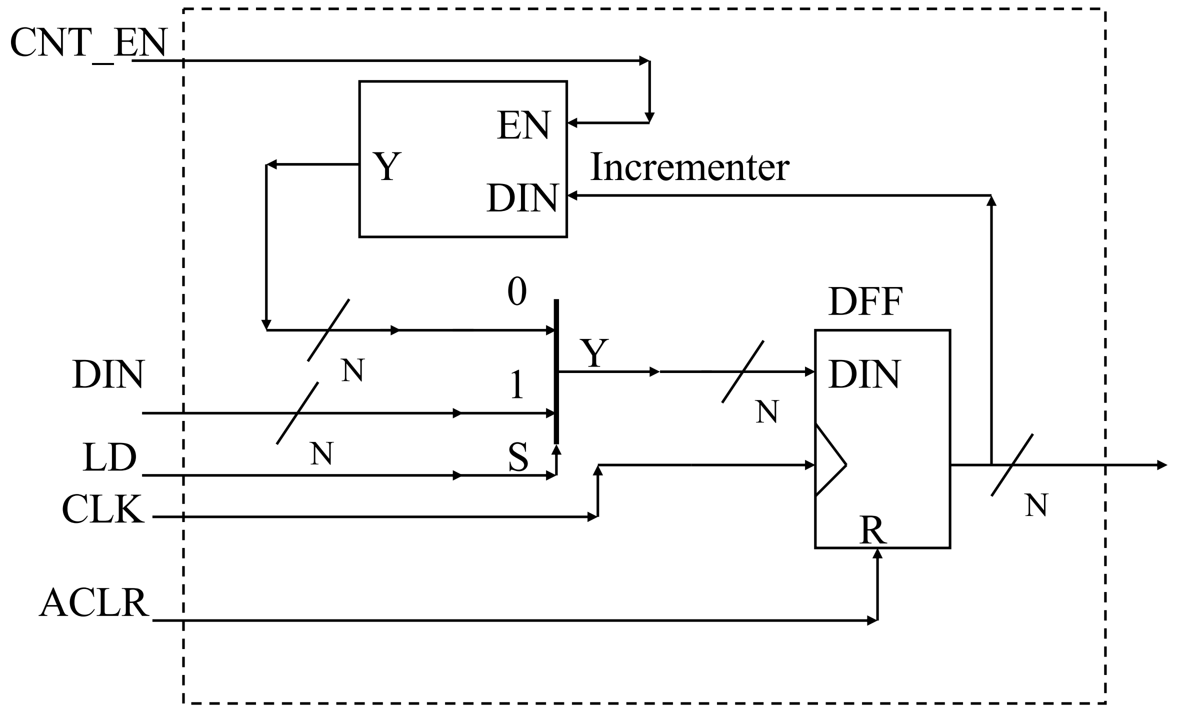


LD asserted: loads counter with DIN value.

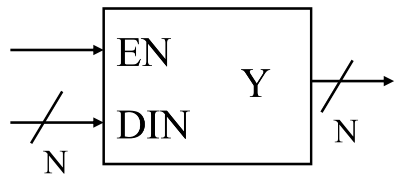
CNT_EN asserted will increment counter on next active clock edge.

ACLR will asynchronously clear the counter.

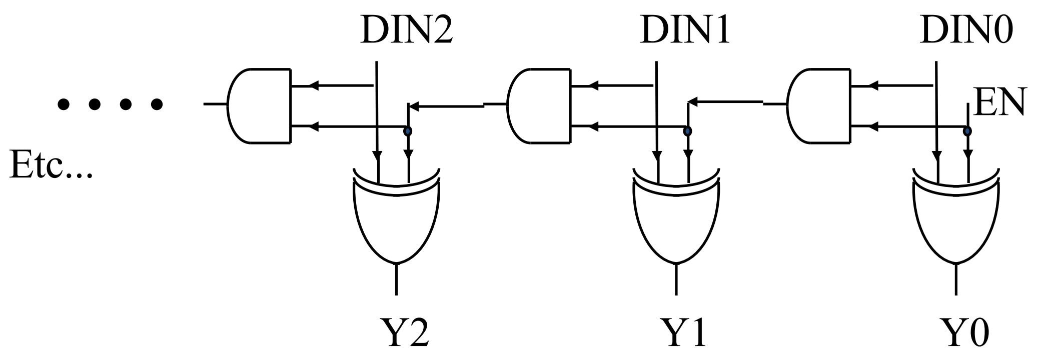
One way to build a Counter



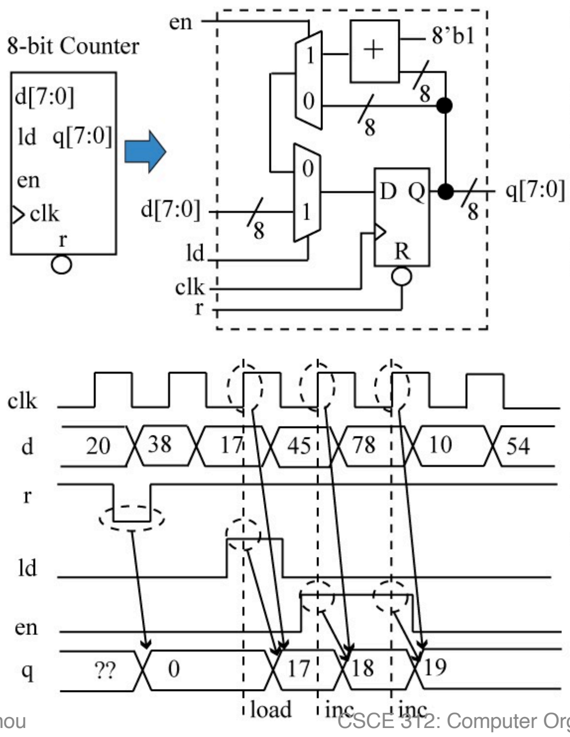
Incrementer: Combinational Building Block



When EN=1, Y = DIN + 1
When EN=0, Y = DIN



8-bit Counter

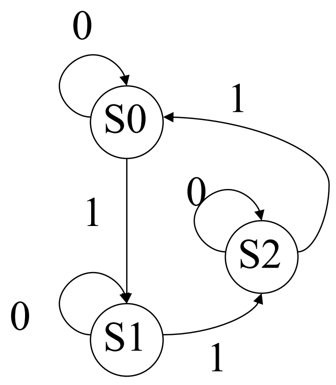


```
module cnt8bit (clk,r,d,
                ld,en,q);

input clk,r,ld,en;
input [7:0] d;
output [7:0] q;
reg [7:0] q;

//register
always @(posedge clk
         or negedge r) begin
 //async reset
 if (!r) q <= 8'b0;
 else begin
 //synchronous inputs
  if (en) q <= q + 8'b1;
  if (ld) q <= d;
 end
end

endmodule
```

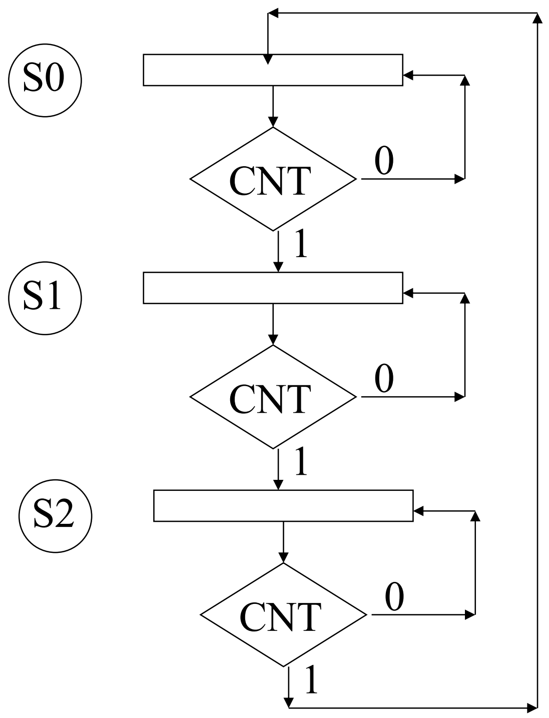
Describing FSMs

1. State Tables
2. State Equations
3. State Diagrams
4. Algorithmic State Machine (ASM) Charts
    1. Preferred method in this class
5. HDL descriptions

Example State Machine

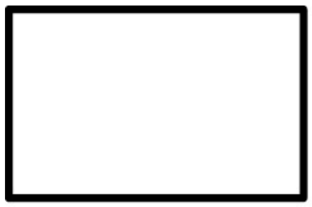


State Diagram
(Bubble Diagram)

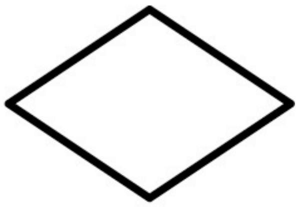


ASM Chart

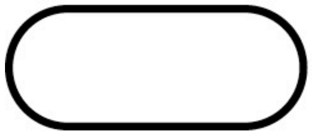
ASM Chart Symbols



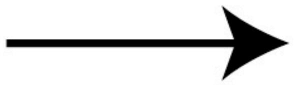
State-Representing
Symbol



Basic Decision
Symbol



Conditional Output
Symbol



Flow Direction
Symbol

FSM Implementation

Use DFFs, State assignment: S0 = 00, S1 = 01, S2 = 10

State Table

PS			NS			
CNT	Q1	Q0	Q1	Q0	D1	D0
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	1	0	1	0
0	1	1	x	x	x	x
1	0	0	0	1	0	1
1	0	1	1	0	1	0
1	1	0	0	0	0	0
1	1	1	x	x	x	x

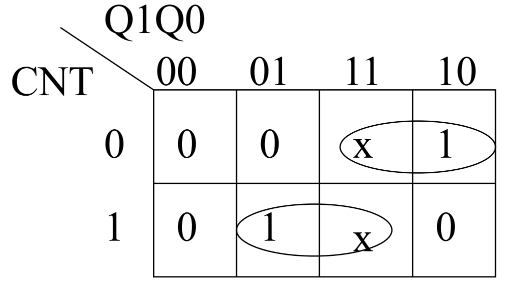
Equations

D1 = CNT'Q1Q0' + CNT Q1'Q0

D0 = CNT'Q1'Q0 + CNT Q1'Q0'

Minimize Equations

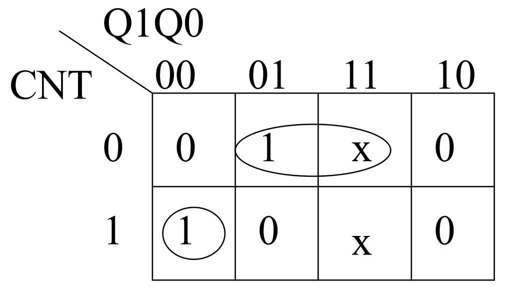
D1



CNT \ Q1Q0	00	01	11	10
0	0	0	x	1
1	0	1	x	0

D1 = CNT'Q1 + CNT Q0

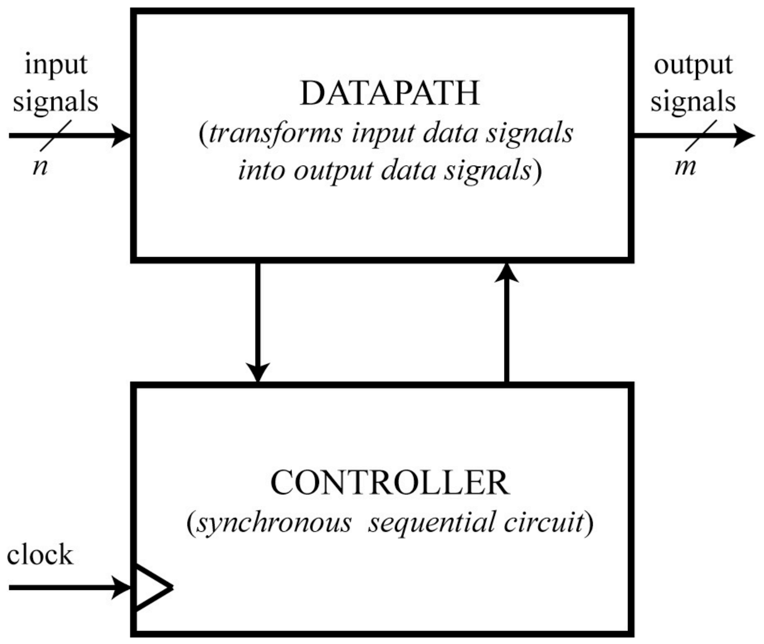
D0



CNT \ Q1Q0	00	01	11	10
0	0	1	x	0
1	1	0	x	0

D0 = CNT'Q0 + CNT Q1'Q0'

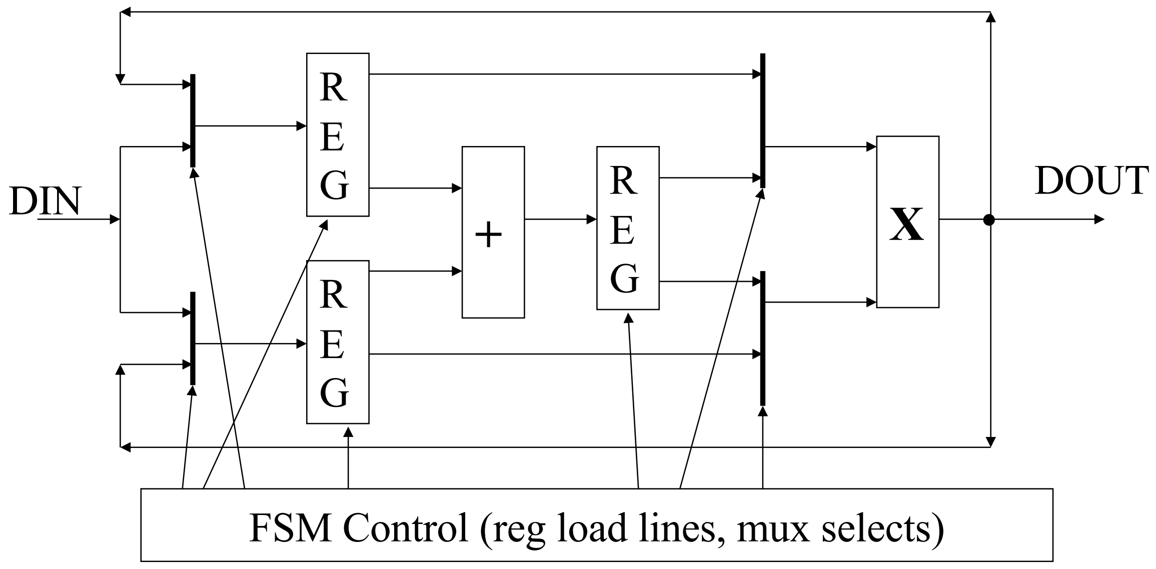
FSMD Block Diagram


input signals
n
DATAPATH
(transforms input data signals into output data signals)
output signals
m
CONTROLLER
(synchronous sequential circuit)
clock

FSM Usage as Controller

Custom counters

Datapath control



Summary

- We will be describing sequential systems via HDL.
- The HDL allows the synthesis of the design
- Will use common sequential building blocks extensively:
- Registers, Counters, Shift registers, Memories
- Basic storage element will be DFF