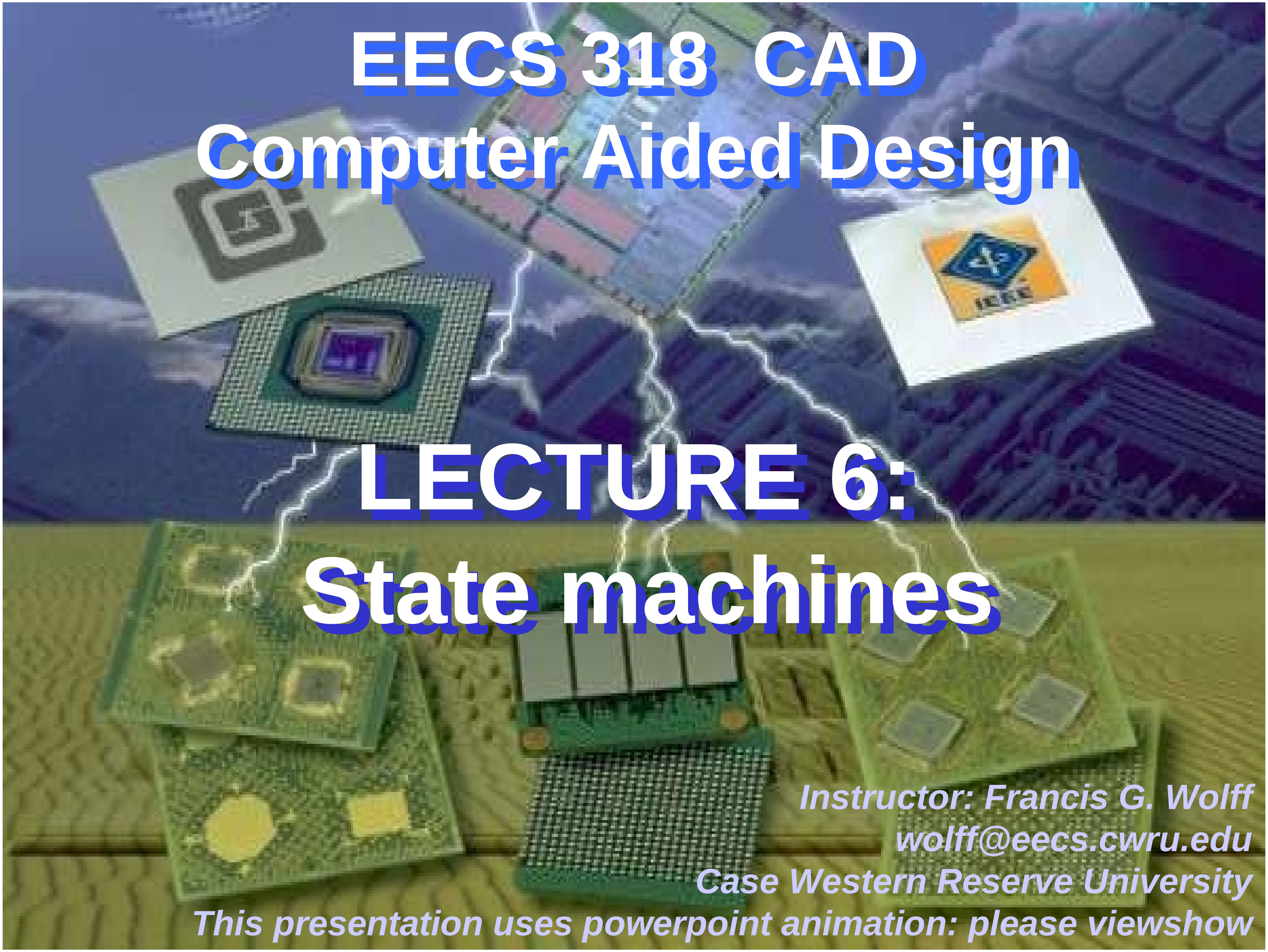
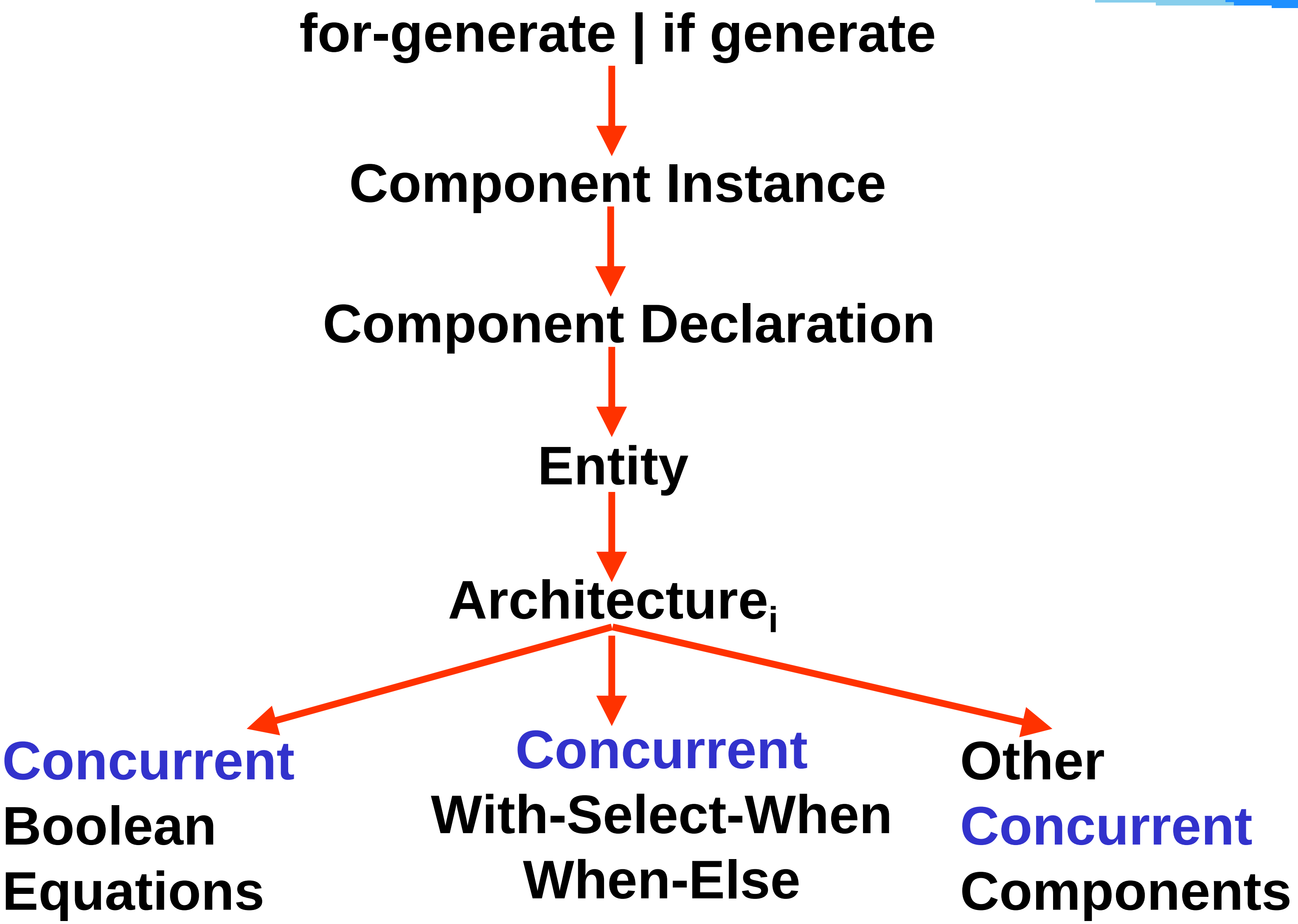


EECS 318 CAD
Computer Aided Design

LECTURE 6:
State machines

Instructor: Francis G. Wolff
wolff@eecs.cwru.edu
Case Western Reserve University
This presentation uses powerpoint animation: please viewshow

VHDL Component, Entity, and Architecture



for-generate | if generate

Component Instance

Component Declaration

Entity

Architecture$_i$

Concurrent
Boolean
Equations

Concurrent
With-Select-When
When-Else

Other
Concurrent
Components

VHDL Components

Component Declaration

[Optional] { repeat }

```
COMPONENT component_entity_name
   [ GENERIC ( { identifier: type [:= initial_value ]; } ) ]
   [ PORT      ( { identifier: mode type; } ) ]
END;
```

Add ; only if another identifier

Component Instance

```
identifier : component_entity_name
   [ GENERIC MAP ( identifier { ,identifier } ) ]
   [ PORT  MAP ( identifier { ,identifier } ) ]
;
```

```
mode  := IN | OUT | INOUT
type   := std_logic | std_logic_vector(n downto 0) | bit
```

VHDL Concurrent Statements

Boolean Equations

relation ::= relation LOGIC relation | NOT relation | (relation)

LOGIC ::= AND | OR | XOR | NAND | NOR | XNOR

Example: y <= NOT (NOT (a) AND NOT (b))

Multiplexor case statement

WITH select_signal SELECT

signal <= signal_value$_1$ WHEN select_compare$_1$,

• • •

WHEN select_compare$_n$;

Example: 2 to 1 multiplexor

WITH s SELECT y <= a WHEN ‘0’, b WHEN OTHERS;

VHDL Concurrent Statements

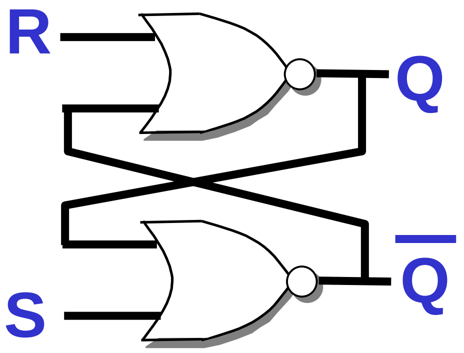
Conditionial signal assignment

```
signal <= signal_value$_1$    WHEN condition$_1$ ELSE
                              • • •
          signal_value$_n$    WHEN condition$_n$; ELSE
          signal_value$_{n+1}$
```

Example: Priority Encoder

```
    y <= a WHEN s='0' ELSE b;
```

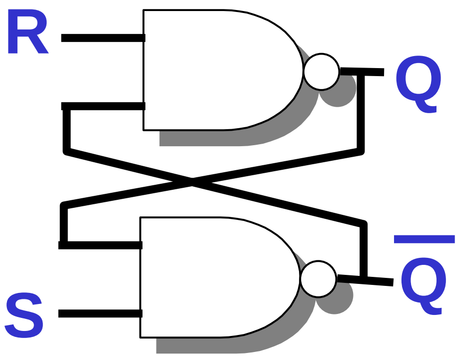
SR Flip-Flop (Latch)



NOR

R	S	Q_{n+1}
0	0	Q_n
0	1	1
1	0	0
1	1	U

```
Q   <= R NOR NQ;
NQ <= S NOR Q;
```



NAND

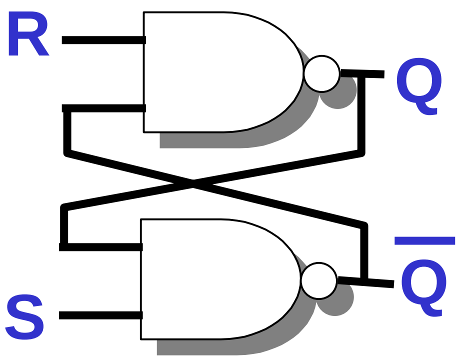
R	S	Q_{n+1}
0	0	U
0	1	1
1	0	0
1	1	Q_n

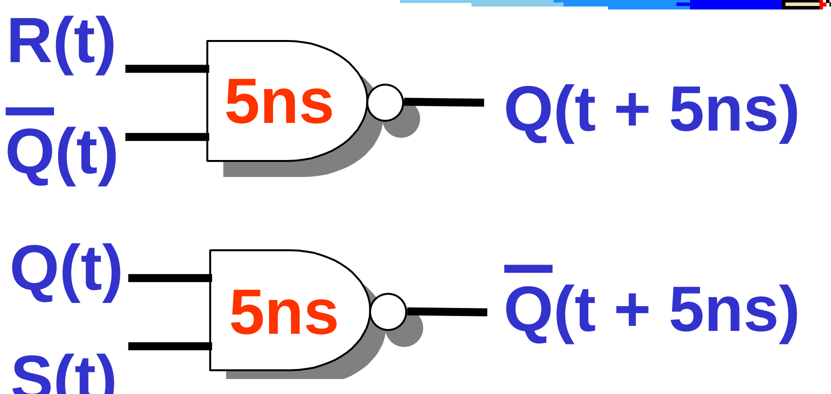
```
Q   <= R NAND NQ;
NQ <= S NAND Q;
```

SR Flip-Flop (Latch)

NAND

R	S	Q_{n+1}
0	0	U
0	1	1
1	0	0
1	1	Q_n



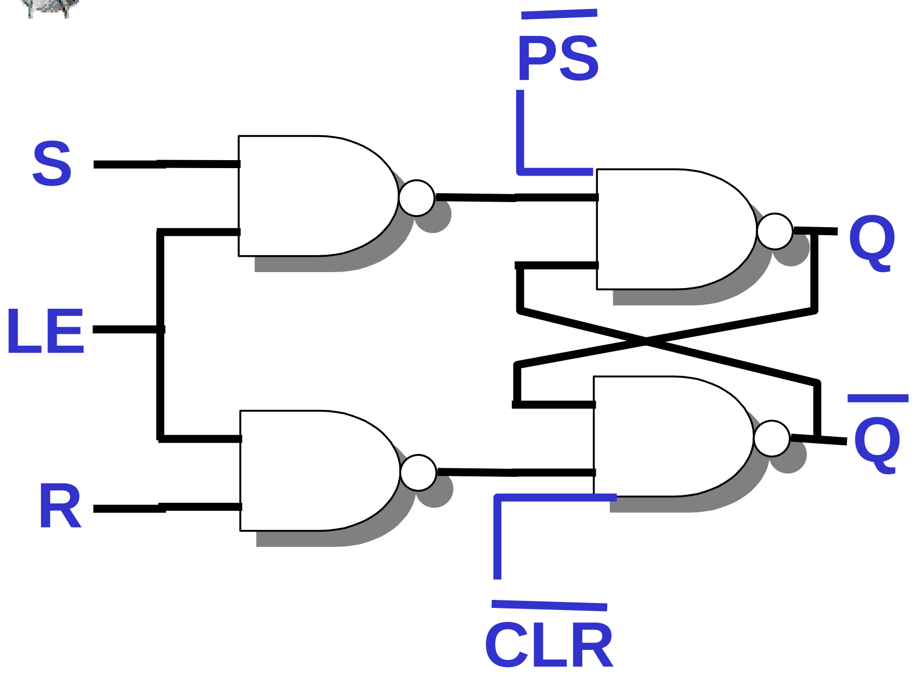


With Delay

Example: R <= ‘1’, ‘0’ after 10ns, ‘1’ after 30ns; S <= ‘1’;

t	0	5ns	10ns	15ns	20ns	25ns	30ns	35ns	40ns
R	1	1	0	0	0	0	1	1	1
$\overline{Q}$	U	U	U	U	0	0	0	0	0
Q	U	U	U	1	1	1	1	1	1
S	1	1	1	1	1	1	1	1	1

Gated-Clock SR Flip-Flop (Latch Enable)



Q <= (S NAND LE) NAND NQ;

NQ <= (R NAND LE) NAND Q;

Synchronous:

Set and Reset

Asynchronous:

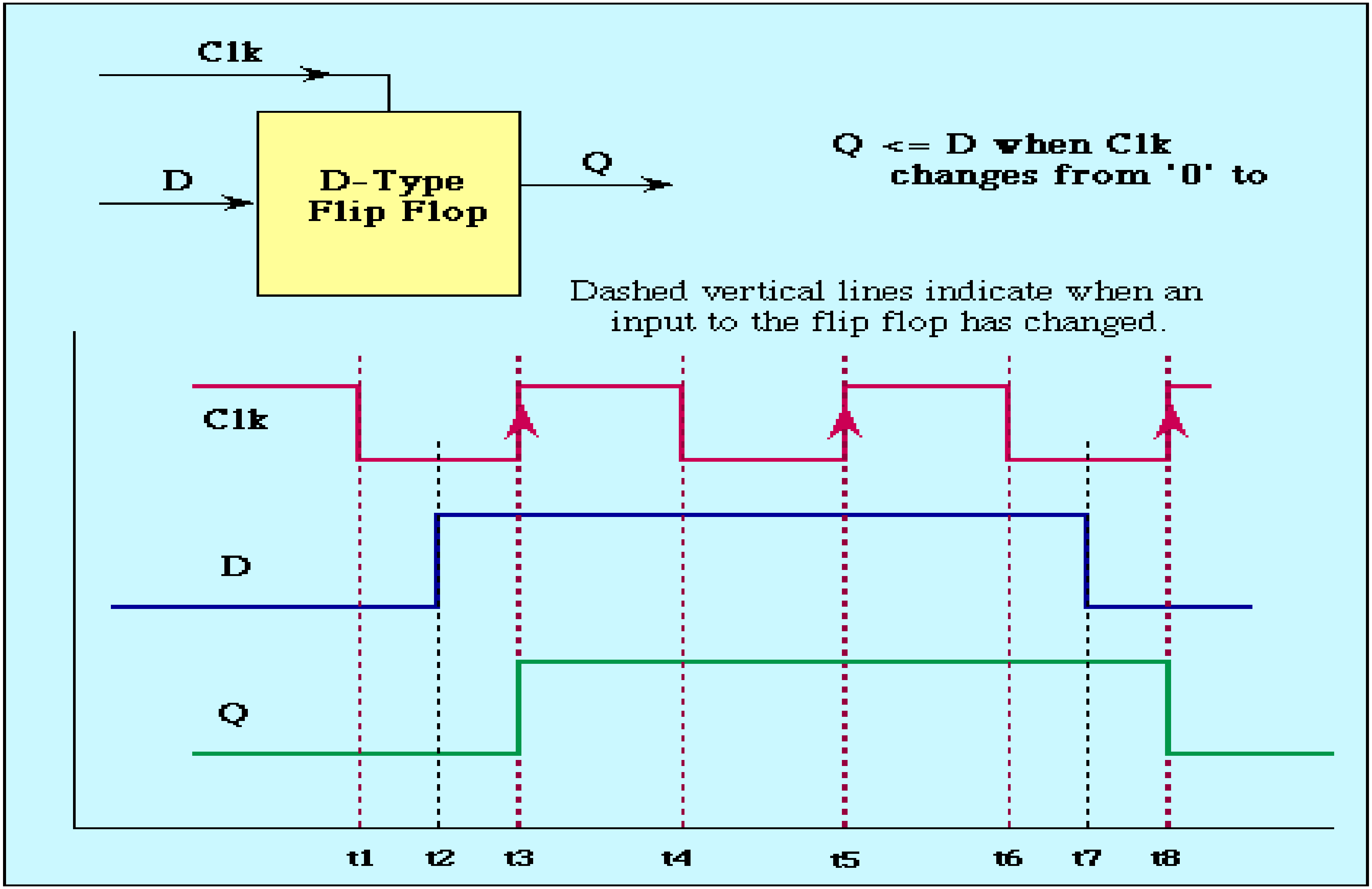
Preset and Clear

Latches require that during the gated-clock the data must also be stable (i.e. S and R) at the same time

Suppose each gate was 5ns: how long does the clock have to be enabled to latch the data?

Answer: 15ns

Rising-Edge Flip-flop



Q <= D when Clk changes from '0' to

Dashed vertical lines indicate when an input to the flip flop has changed.

Rising-Edge Flip-flop logic diagram

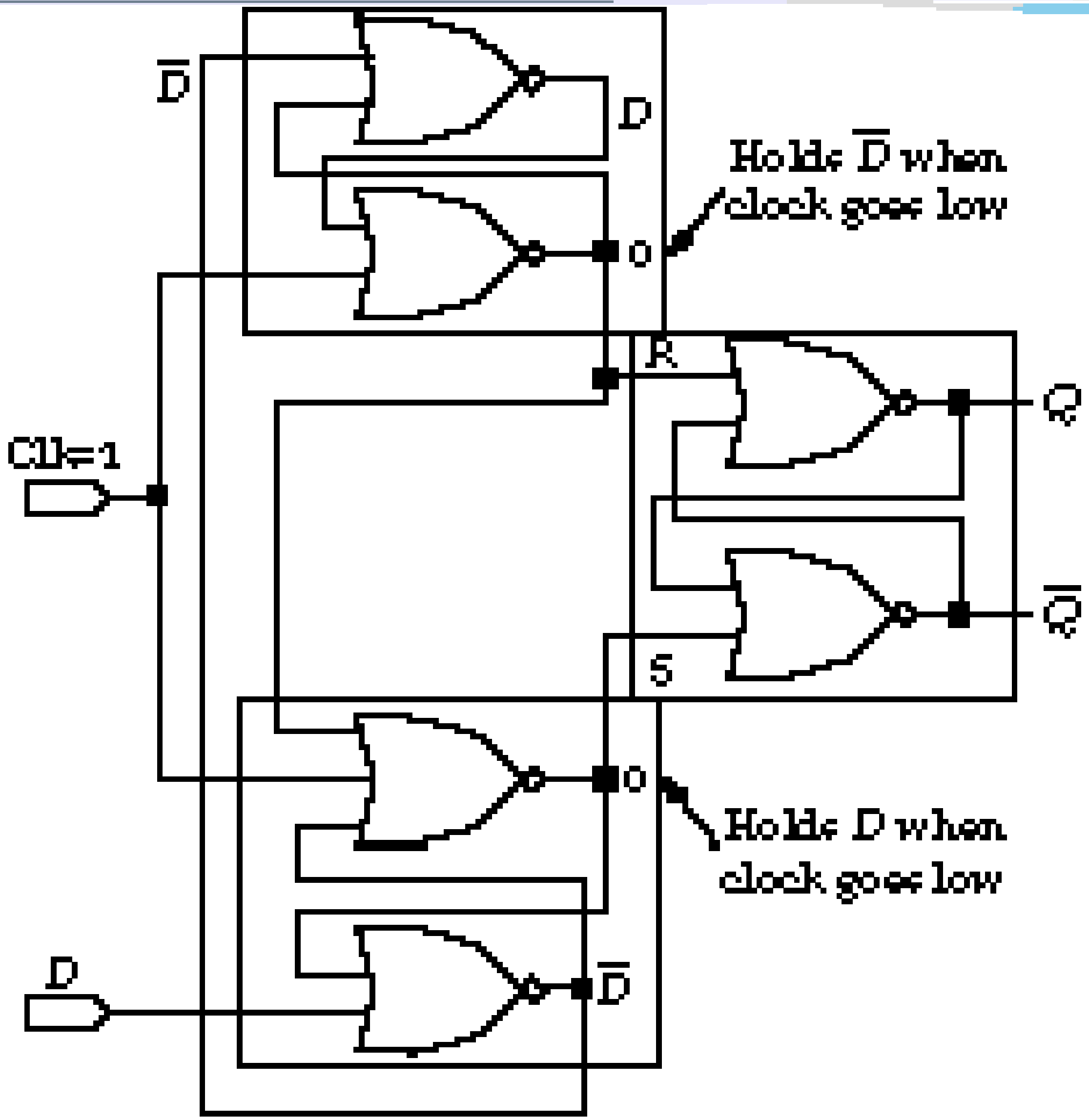
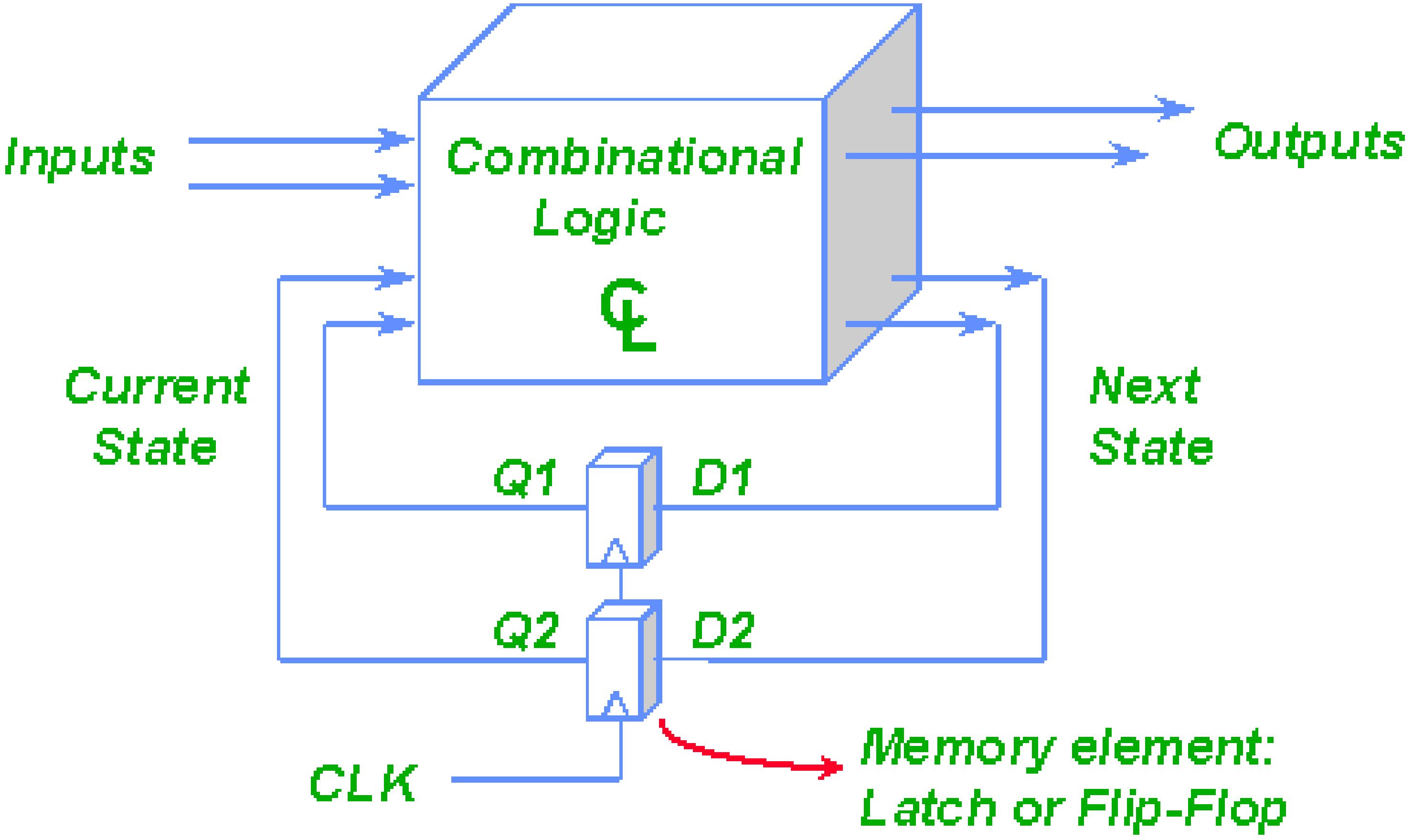

$\overline{D}$
D
Holds $\overline{D}$ when clock goes low
0
R
Q
Clk=1
$\overline{Q}$
S
0
Holds D when clock goes low
D
$\overline{D}$

Figure 6.24 Negative edge-triggered *D* flip-flop when clock is

Synchronous Sequential Circuit



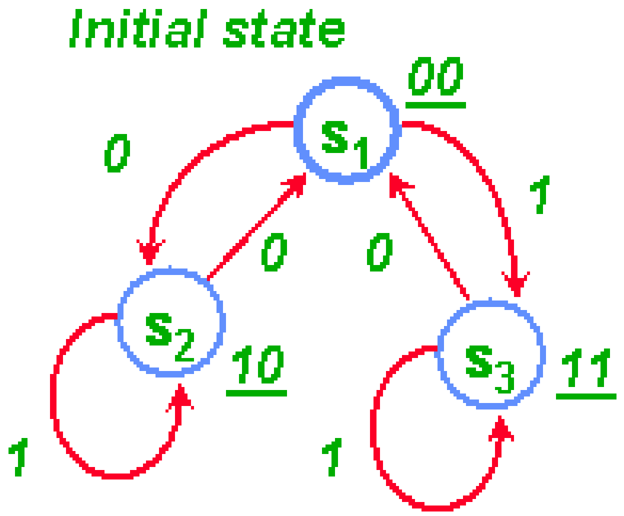
Issues: <u>Specification, design</u>, clocking and timing

Abstraction: Finite State Machine

- A Finite State Machine (FSM) has:
  - K states, $S = \{s_1, s_2, \ldots, s_K\}$, initial state s_1
  - N inputs, $I = \{i_1, i_2, \ldots, i_N\}$
  - M outputs, $O = \{o_1, o_2, \ldots, o_M\}$
  - Transition function $T(S, I)$ mapping each current state and input to a next state
  - Output function $O(S)$ mapping each current state to an output
- Given a sequence of inputs the FSM produces a sequence of outputs which is dependent on s_1 , $T(S, I)$ and $O(S)$

FSM Representations

State Transition Graph



	t	t+1	t+2
Inputs:	0	1	0
Outputs:	00	___	___

State Transition Table

T(S, I)

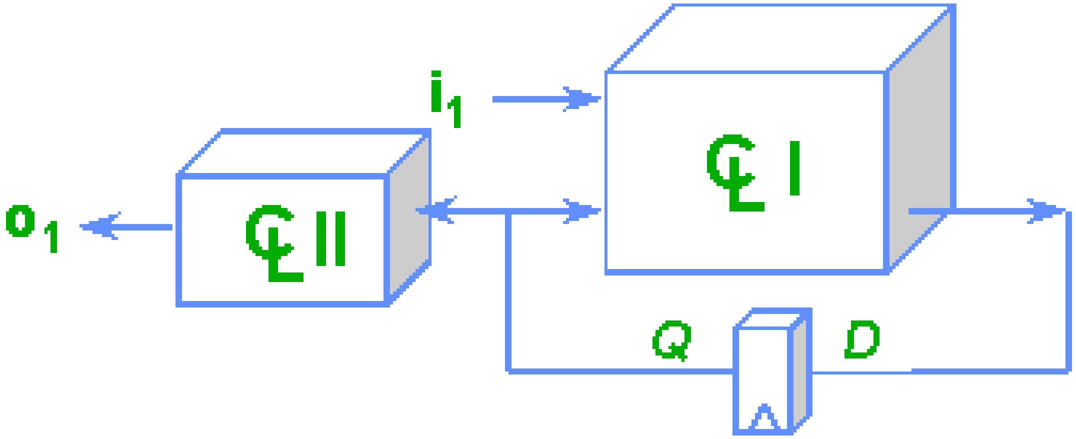
0	s_1	s_2
1	s_1	s_3
0	s_2	s_1
1	s_2	s_2
0	s_3	s_1
1	s_3	s_3

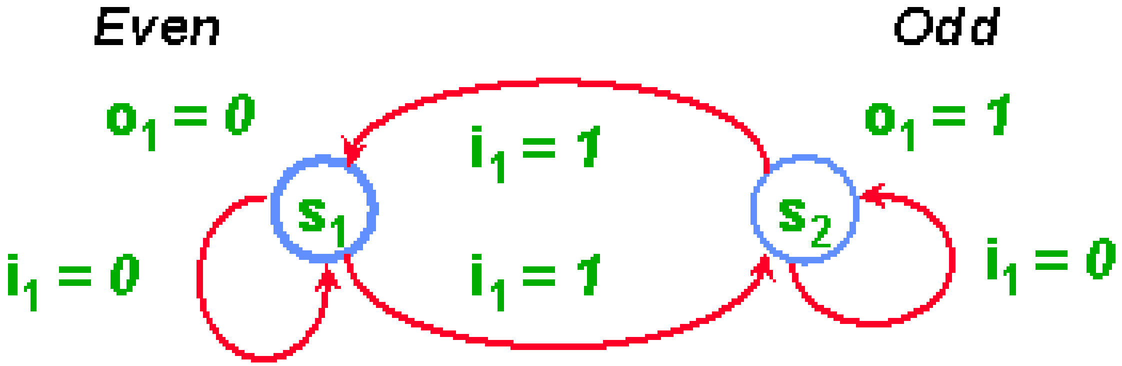
O(S)

s_1	00
s_2	10
s_3	11

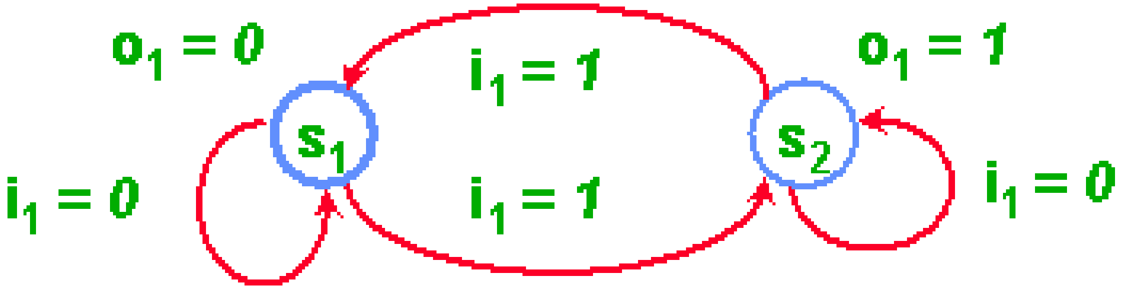
Simple Design Example

- Design a FSM that outputs a 1 if and only if the number of 1's in the input sequence is odd





State Encoding



- State Encoding: Choose a unique binary code for each s_i so the combinational logic can be specified
  - Choose $s_1 = 0$ and $s_2 = 1$
  - Choose $s_1 = 1$ and $s_2 = 0$

Logic Implementations

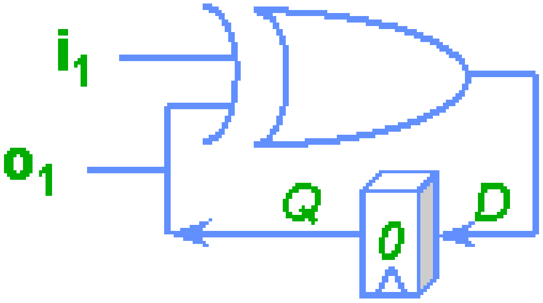
Choose $s_1 = 0$ and $s_2 = 1$

CL I

i_1	Q	D
0	0	0
1	0	1
0	1	1
1	1	0

CL II

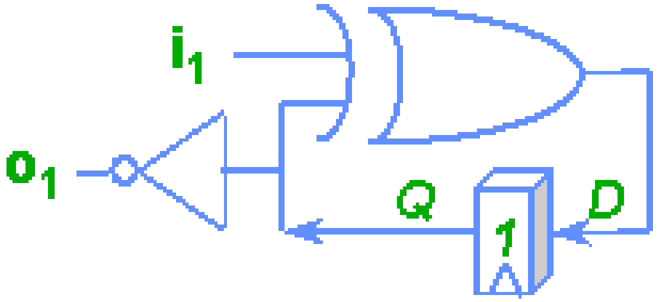
Q	o_1
0	0
1	1



Choose $s_1 = 1$ and $s_2 = 0$

i_1	Q	D
0	1	1
1	1	0
0	0	0
1	0	1

Q	o_1
1	0
0	1

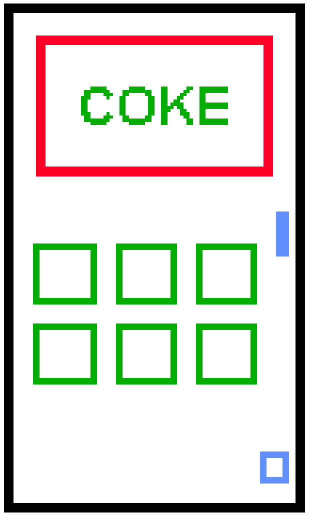


Observations

- Number of bits required to encode K states is $\lceil \log_2 K \rceil$
- Encoding states results in combinational logic specifications for T(S, I) and O(S)
- Choice of encoding affects complexity of logic implementation
  - How does one find the optimum state encoding?

Coke Machine Example

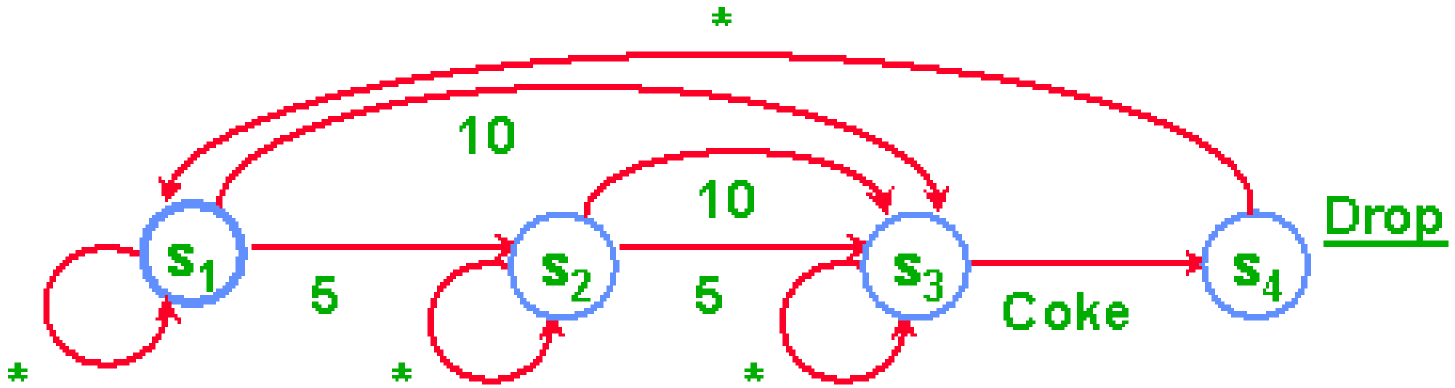
- Coke costs $.10
- Only nickels and dimes accepted
- FSM inputs:
  - 5: Nickel
  - 10: Dime
  - Coke: Give me a coke
  - Return: Give me my money back
- FSM outputs:
  - Drop: Drop a coke
  - Ret5: Return $.05
  - Ret10: Return $.10



Coke Machine State Diagram

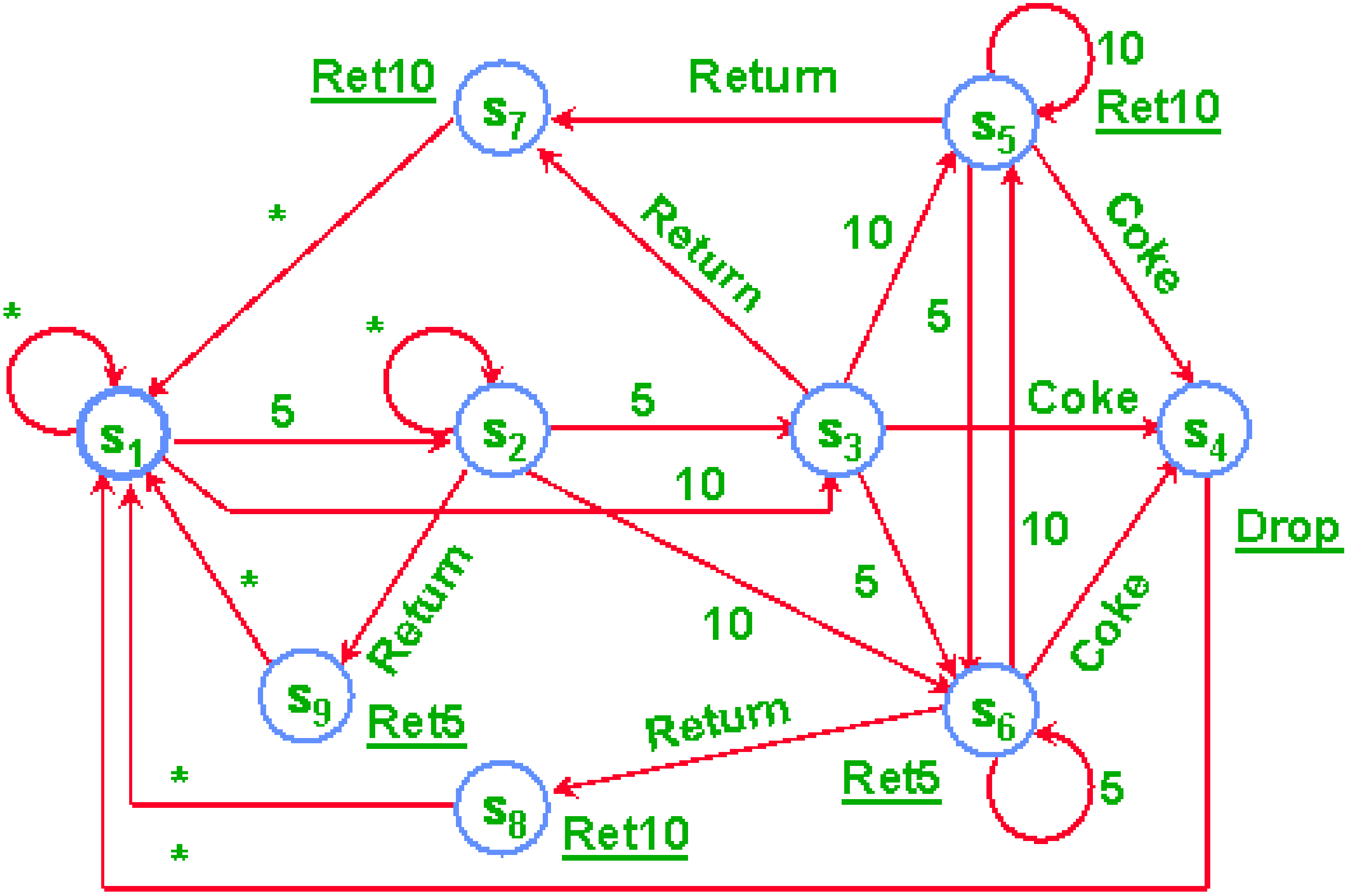
Assumption: At most one input among Coke, 5, 10, and Return is asserted

* represents all unspecified transitions from state



Does this work? ______________________

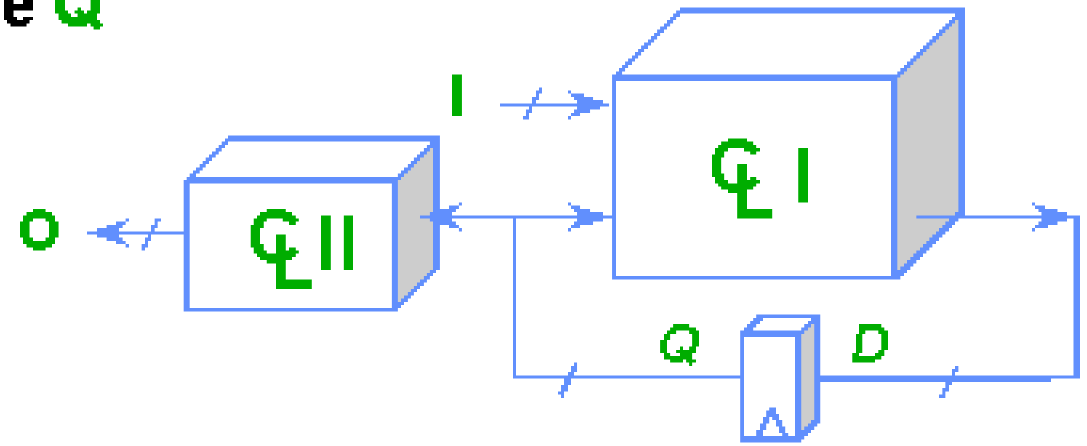
Coke Machine Diagram - II



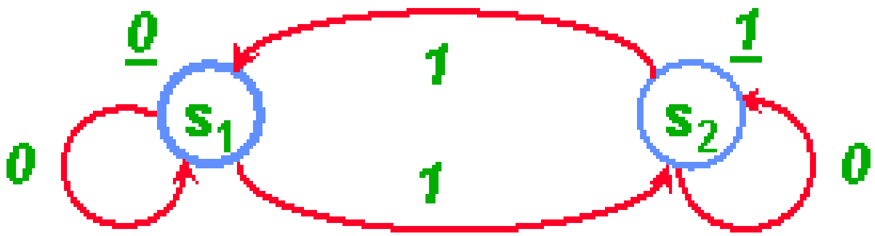
After Return input, any input in the next cycle is ignored!

Moore Machines

- So far we considered Moore machines where the output O is a function of only the current state Q

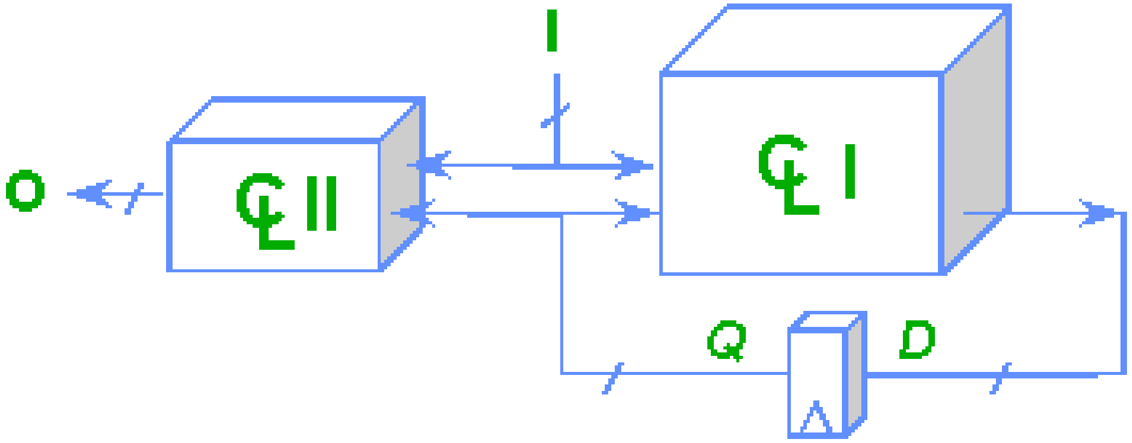


- Moore FSM State Transition Graph

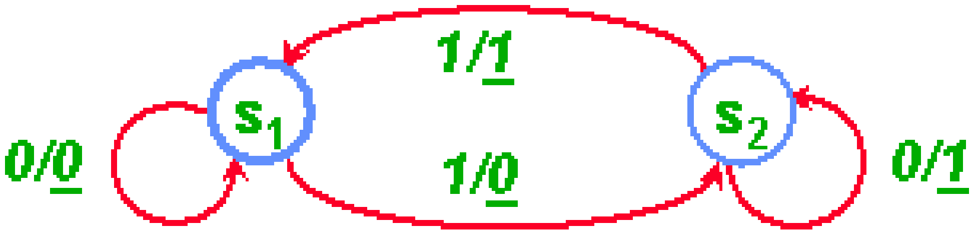


Mealy Machines

- In Mealy machines the output O is a function of the current state Q and input I



- Mealy FSM State Transition Diagram


1/1
0/0
s_1
s_2
0/1
1/0