Registers and Counters

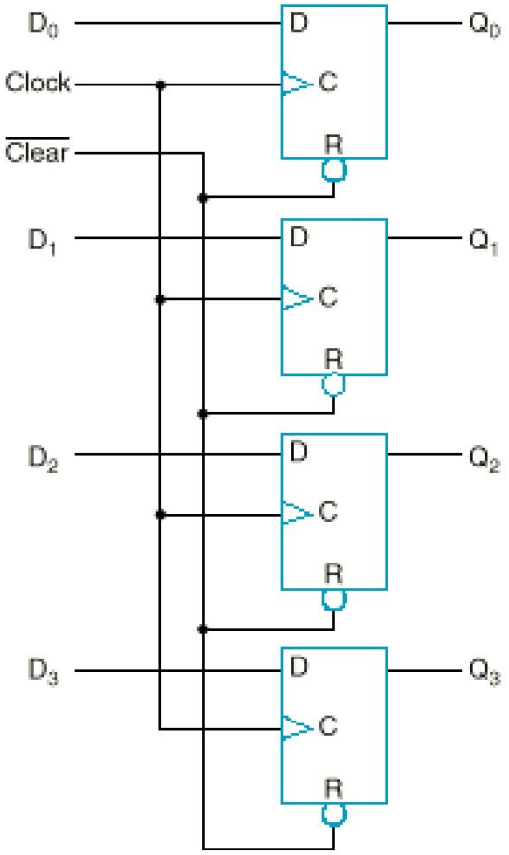
Overview

- **Parallel Load Register**
- **Shift Registers**
  - Serial Load
  - Serial Addition
- **Shift Register with Parallel Load**
- **Bidirectional Shift Register**
- **Counters**
  - Synchronous
  - Ripple

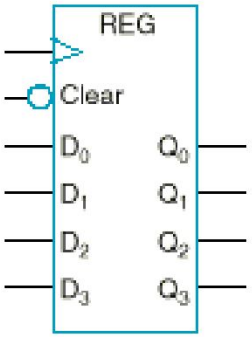
Registers and Counters

- An *n*-bit *register* = *n* flip-flops arranged to work together
- Capable of storing *n* bits of binary information

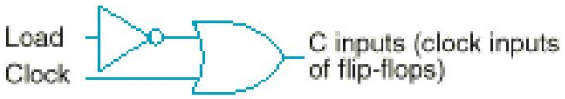
4-Bit Register (w/ clock gating)



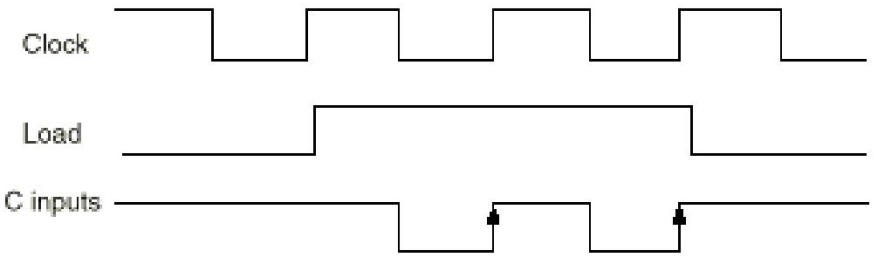
(a) Logic diagram



(b) Symbol

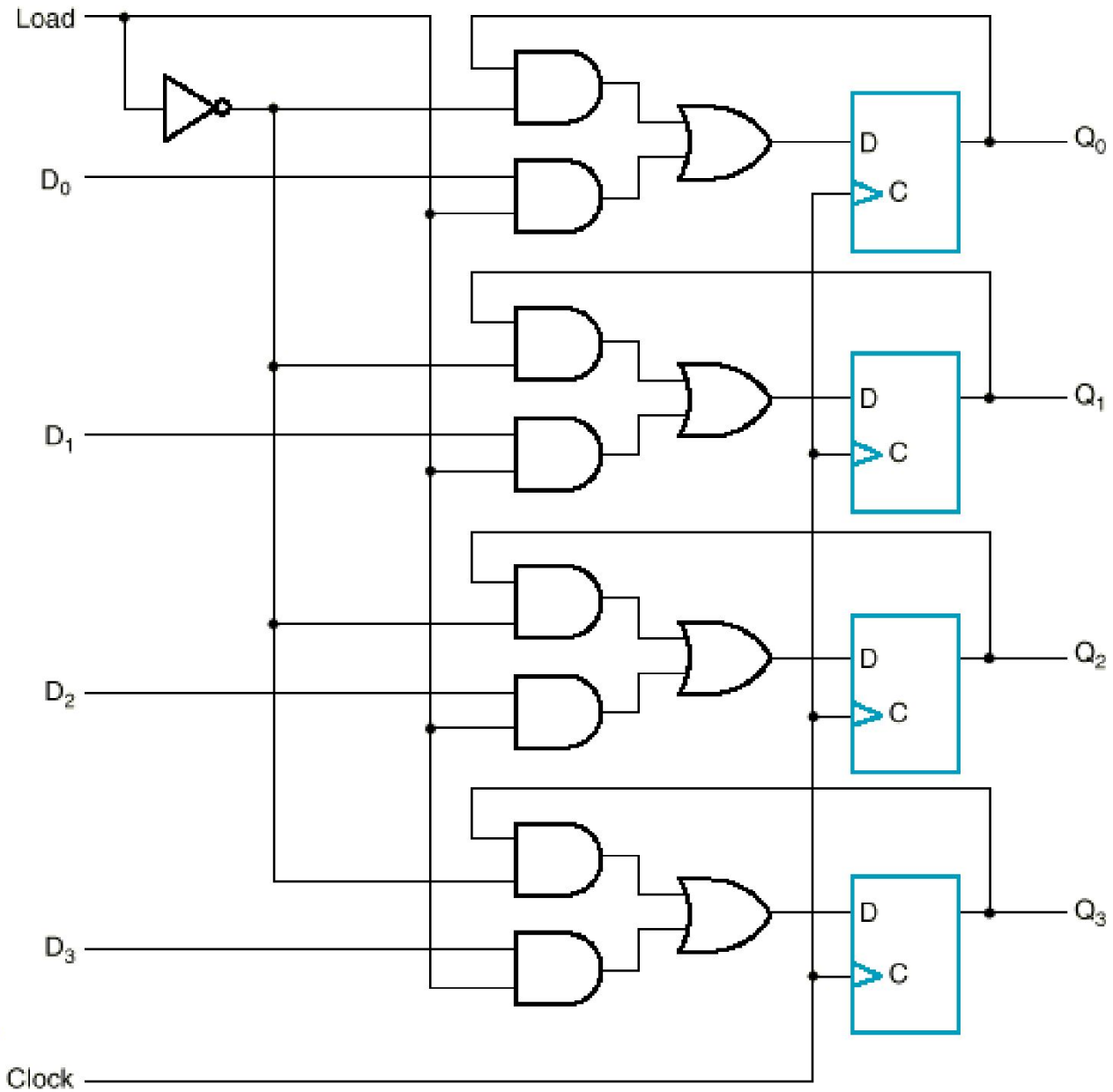


(c) Load control input

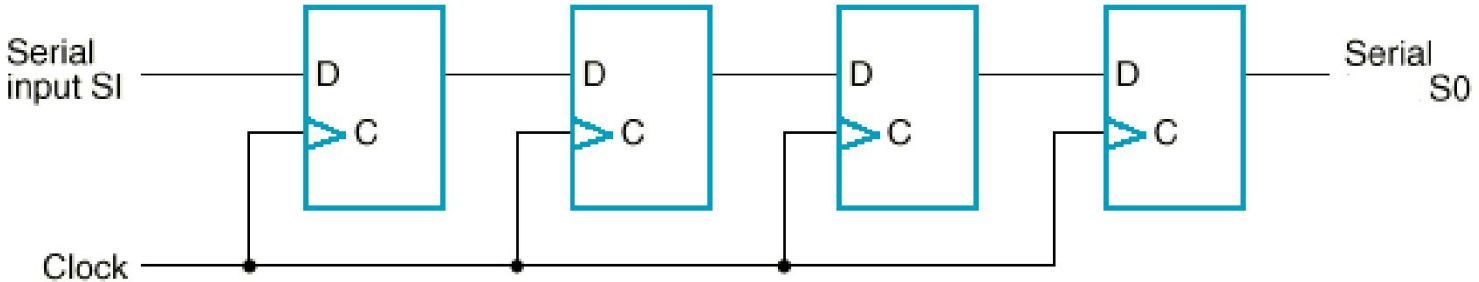


(d) Timing diagram

4-Bit Register (w/o clock gating)



Shift Register

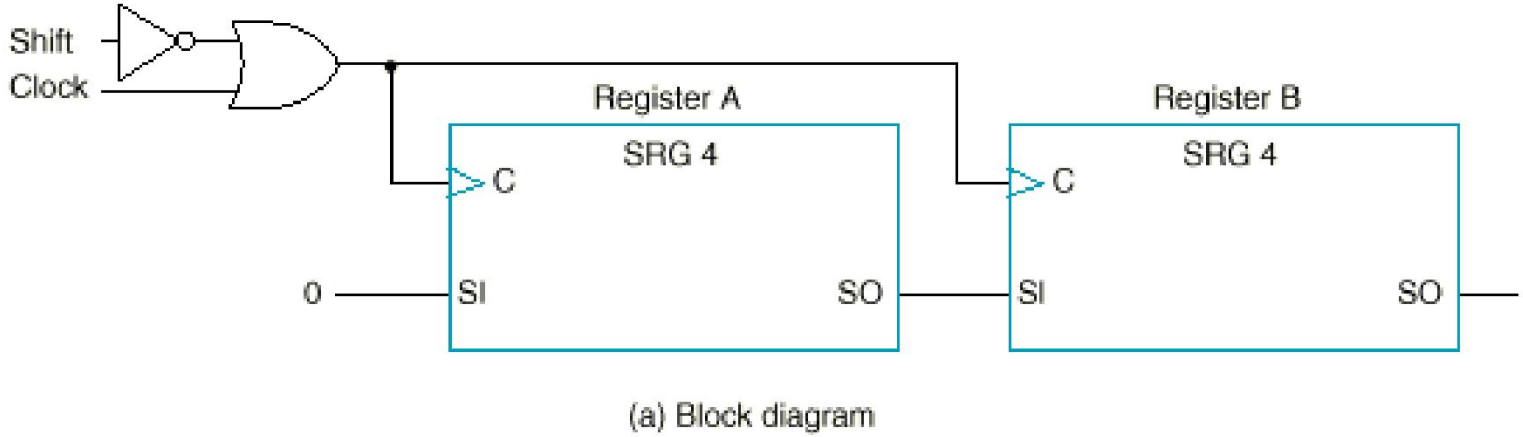


(a) Logic diagram

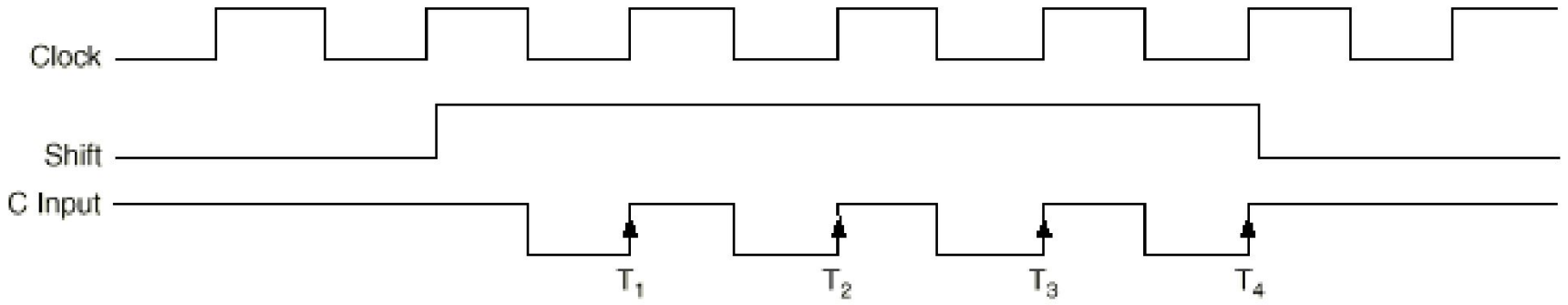
(b) Symbol

Serial Data Transfer

Serial transfer of information from register A to register B

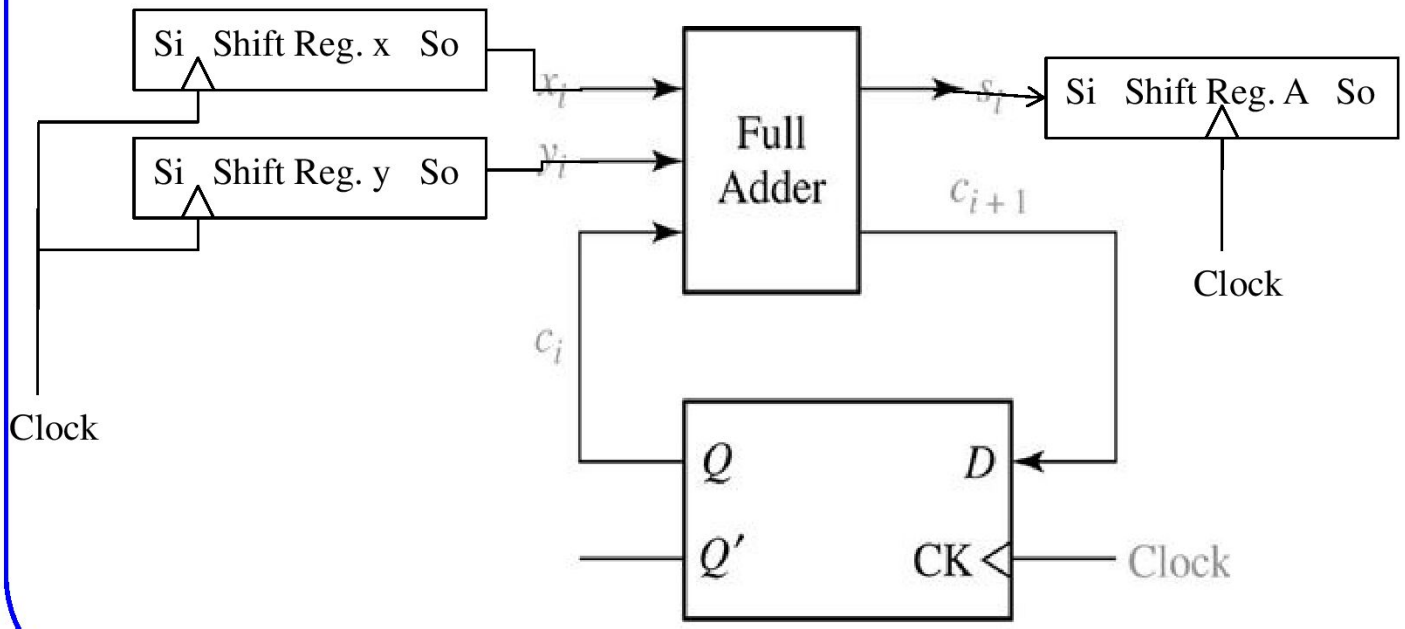


(a) Block diagram



Serial Addition Using Shift Registers

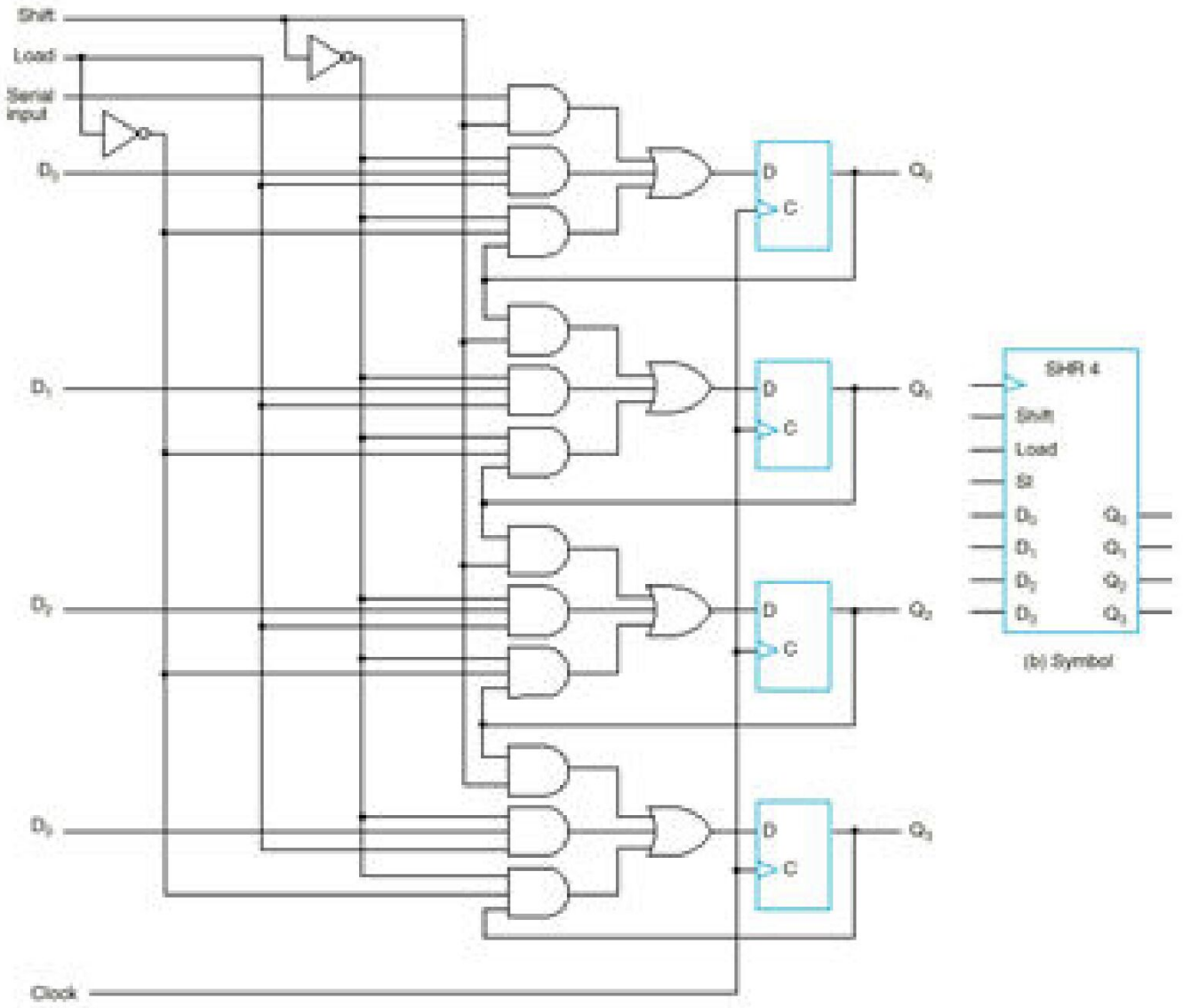
- The two binary numbers to be added serially are stored in two shift registers.
- The sum bit on the S output of the full adder is transferred into the result register A.



Serial *vs.* Parallel Addition

- The parallel adder is a combinational circuit, whereas the serial adder is a sequential circuit.
- The parallel adder has *n* full adders whereas the serial adder requires only one full adder.
- The serial circuit takes *n* clock cycles to complete an addition.
- The serial adder, although it is *n* times slower, is *n* times smaller in space.

Shift Register with Parallel Load



Shift Register with Parallel Load (unidirectional)

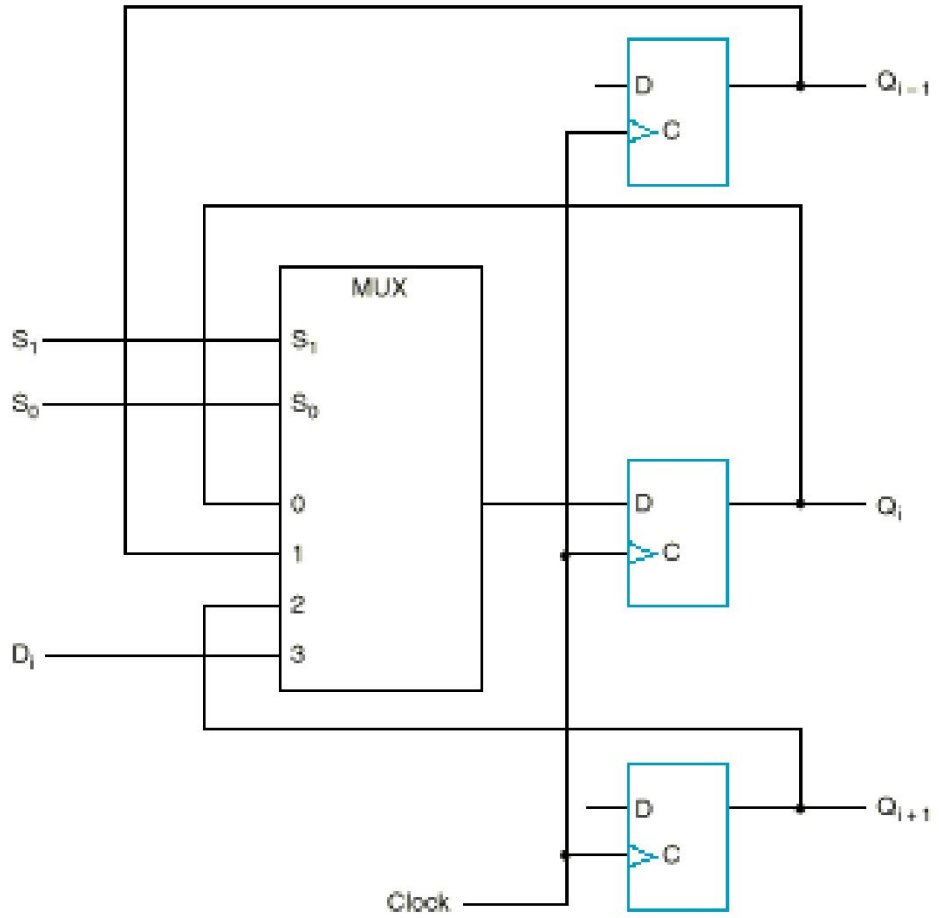
Shift	Load	Operation
0	0	Nothing
0	1	Load parallel
1	X	Shift Q0 → Q1, Q1→Q2…

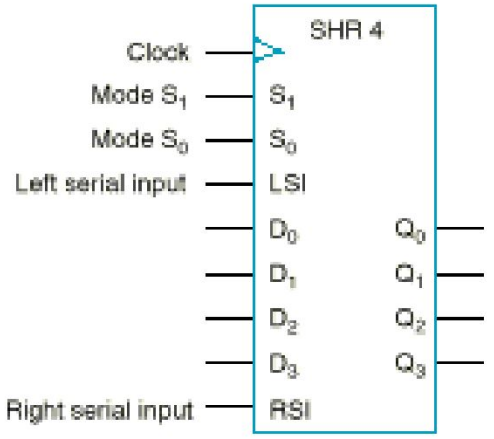
Shift Register with Parallel Load (bidirectional)

S_1S_0	Action
00	Nothing
01	Shift down
10	Shift up
11	Parallel load

Bidirectional Shift Register

Note: For simplicity, only one MUX is shown. Each FF needs its own MUX!





(b) Symbol

S_1S_0	Action
00	Nothing
01	Shift down
10	Shift up
11	Parallel load

Counters

- **A *counter* is a sequential circuit that goes through a predetermined sequence of states upon the application of clock pulses.**
- **Counters are categorized as:**
  - Synchronous Counters:
    All FFs receive a common clock pulse, and the change of state is determined from the present state.
  - Ripple Counters:
    One FF output transition serves as a source for triggering other FFs. No common clock.

Synchronous Binary Counters

- The design procedure for a binary counter is the same as any other synchronous sequential circuit.
- Most efficient implementations usually use T-FFs or JK-FFs. We will examine JK and D flip-flop designs.

Synchronous Binary Counters: JK FF Implementation

TABLE 4-10
Flip-Flop Excitation Tables

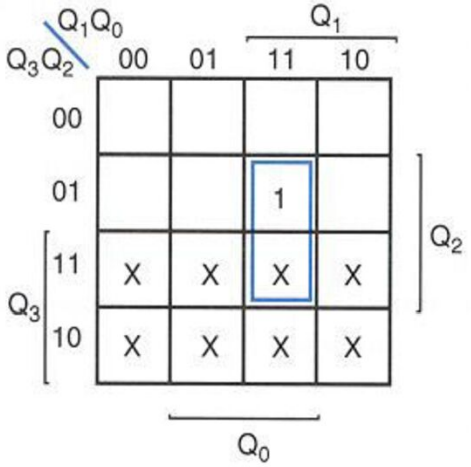
(a) *JK* Flip-Flop

$Q(t)$	$Q(t+1)$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Present state				Next state				Flip-flop inputs							
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	J_{Q3}	K_{Q3}	J_{Q2}	K_{Q2}	J_{Q1}	K_{Q1}	J_{Q0}	K_{Q0}
0	0	0	0	0	0	0	1	0	×	0	×	0	×	1	×
0	0	0	1	0	0	1	0	0	×	0	×	1	×	×	1
0	0	1	0	0	0	1	1	0	×	0	×	×	0	1	×
0	0	1	1	0	1	0	0	0	×	1	×	×	1	×	1
0	1	0	0	0	1	0	1	0	×	×	0	0	×	1	×
0	1	0	1	0	1	1	0	0	×	×	0	1	×	×	1
0	1	1	0	0	1	1	1	0	×	×	0	×	0	1	×
0	1	1	1	1	0	0	0	1	×	×	1	×	1	×	1
1	0	0	0	1	0	0	1	×	0	0	×	0	×	1	×
1	0	0	1	1	0	1	0	×	0	0	×	1	×	×	1
1	0	1	0	1	0	1	1	×	0	0	×	×	0	1	×
1	0	1	1	1	1	0	0	×	0	1	×	×	1	×	1
1	1	0	0	1	1	0	1	×	0	×	0	0	×	1	×
1	1	0	1	1	1	1	0	×	0	×	0	1	×	×	1
1	1	1	0	1	1	1	1	×	0	×	0	×	0	1	×
1	1	1	1	0	0	0	0	×	1	×	1	×	1	×	1

Synchronous Binary Counters: JK FF Implementation (Cont'd)

Present state				Next state					
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	J_{Q3}	K_{Q3}
0	0	0	0	0	0	0	1	0	X
0	0	0	1	0	0	1	0	0	X
0	0	1	0	0	0	1	1	0	X
0	0	1	1	0	1	0	0	0	X
0	1	0	0	0	1	0	1	0	X
0	1	0	1	0	1	1	0	0	X
0	1	1	0	0	1	1	1	0	X
0	1	1	1	1	0	0	0	1	X
1	0	0	0	1	0	0	1	X	0
1	0	0	1	1	0	1	0	X	0
1	0	1	0	1	0	1	1	X	0
1	0	1	1	1	1	0	0	X	0
1	1	0	0	1	1	0	1	X	0
1	1	0	1	1	1	1	0	X	0
1	1	1	0	1	1	1	1	X	0
1	1	1	1	0	0	0	0	X	1

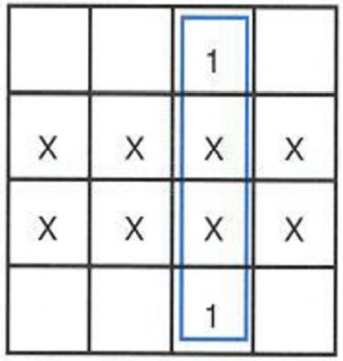


$J_{Q3} = Q_0Q_1Q_2$

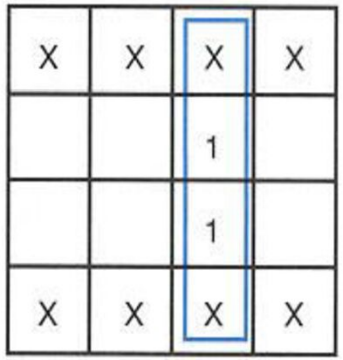
$K_{Q3} = Q_0Q_1Q_2$

Synchronous Binary Counters: JK FF Implementation (Cont'd)

Present state				Next state				Flip-flo	
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	J_{Q2}	K_{Q2}
0	0	0	0	0	0	0	1	0	X
0	0	0	1	0	0	1	0	0	X
0	0	1	0	0	0	1	1	0	X
0	0	1	1	0	1	0	0	1	X
0	1	0	0	0	1	0	1	X	0
0	1	0	1	0	1	1	0	X	0
0	1	1	0	0	1	1	1	X	0
0	1	1	1	1	0	0	0	X	1
1	0	0	0	1	0	0	1	0	X
1	0	0	1	1	0	1	0	0	X
1	0	1	0	1	0	1	1	0	X
1	0	1	1	1	1	0	0	1	X
1	1	0	0	1	1	0	1	X	0
1	1	0	1	1	1	1	0	X	0
1	1	1	0	1	1	1	1	X	0
1	1	1	1	0	0	0	0	X	1



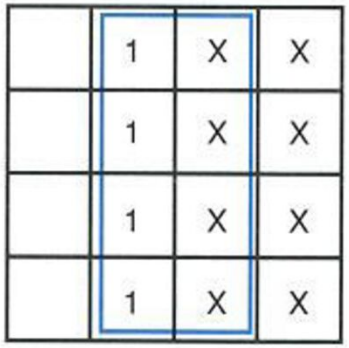
$J_{Q2} = Q_0Q_1$



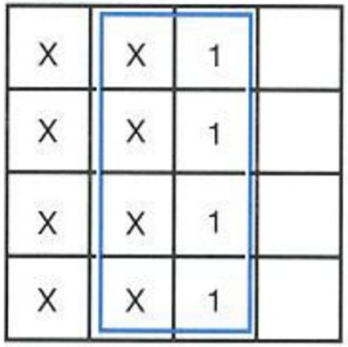
$K_{Q2} = Q_0Q_1$

Synchronous Binary Counters: JK FF Implementation (Cont'd)

Present state				Next state				p inputs	
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	J_{Q1}	K_{Q1}
0	0	0	0	0	0	0	1	0	X
0	0	0	1	0	0	1	0	1	X
0	0	1	0	0	0	1	1	X	0
0	0	1	1	0	1	0	0	X	1
0	1	0	0	0	1	0	1	0	X
0	1	0	1	0	1	1	0	1	X
0	1	1	0	0	1	1	1	X	0
0	1	1	1	1	0	0	0	X	1
1	0	0	0	1	0	0	1	0	X
1	0	0	1	1	0	1	0	1	X
1	0	1	0	1	0	1	1	X	0
1	0	1	1	1	1	0	0	X	1
1	1	0	0	1	1	0	1	0	X
1	1	0	1	1	1	1	0	1	X
1	1	1	0	1	1	1	1	X	0
1	1	1	1	0	0	0	0	X	1

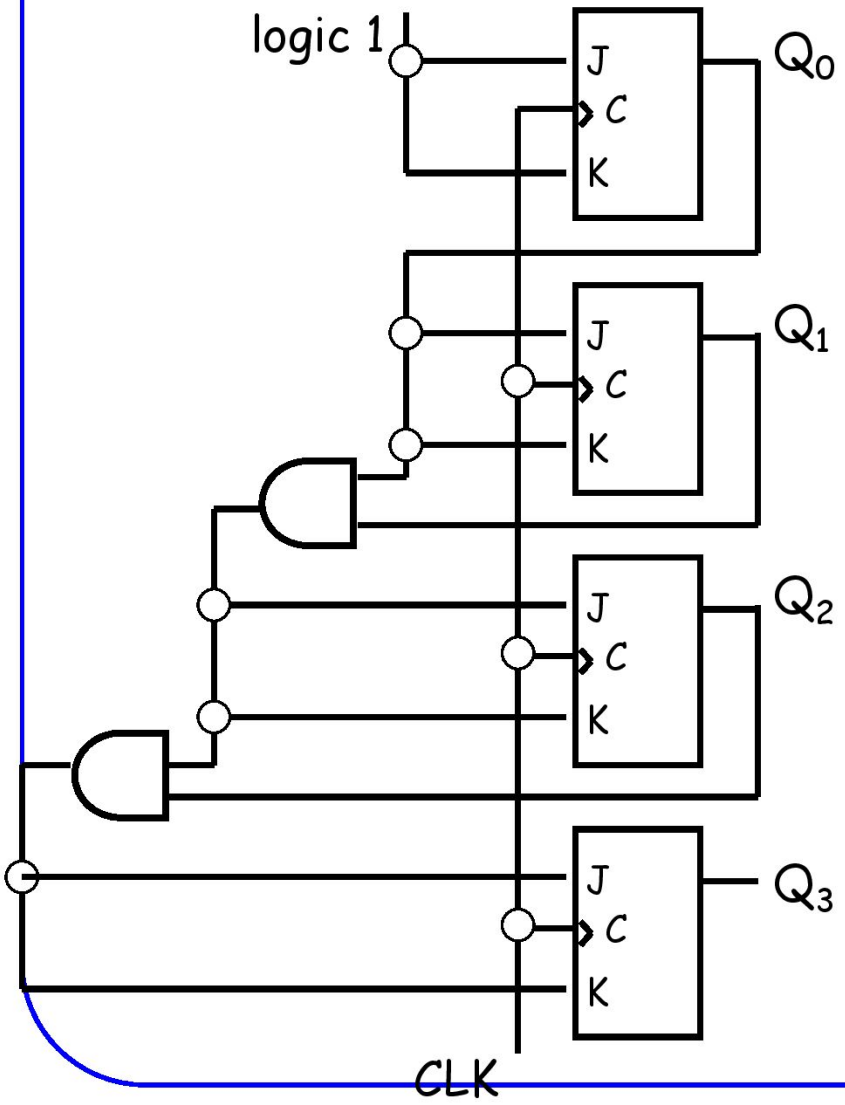


$J_{Q1} = Q_0$



$K_{Q1} = Q_0$

Synchronous Binary Counters: JK FF Implementation (Cont'd)



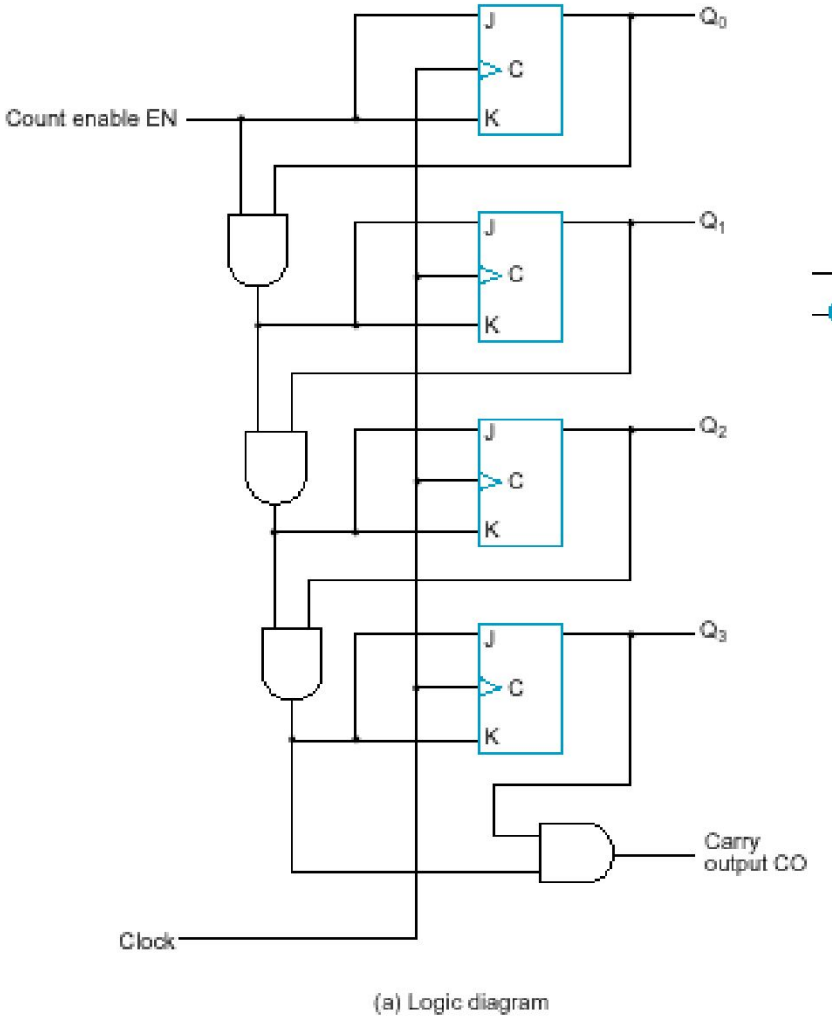
$J_{Q0} = 1$
$K_{Q0} = 1$

$J_{Q1} = Q_0$
$K_{Q1} = Q_0$

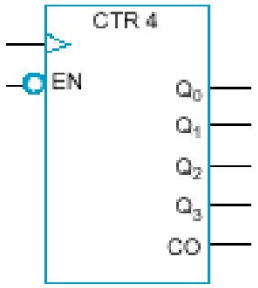
$J_{Q2} = Q_0 Q_1$
$K_{Q2} = Q_0 Q_1$

$J_{Q3} = Q_0 Q_1 Q_2$
$K_{Q3} = Q_0 Q_1 Q_2$

Synchronous Binary Counters: JK FF Implementation with EN and CO



(a) Logic diagram



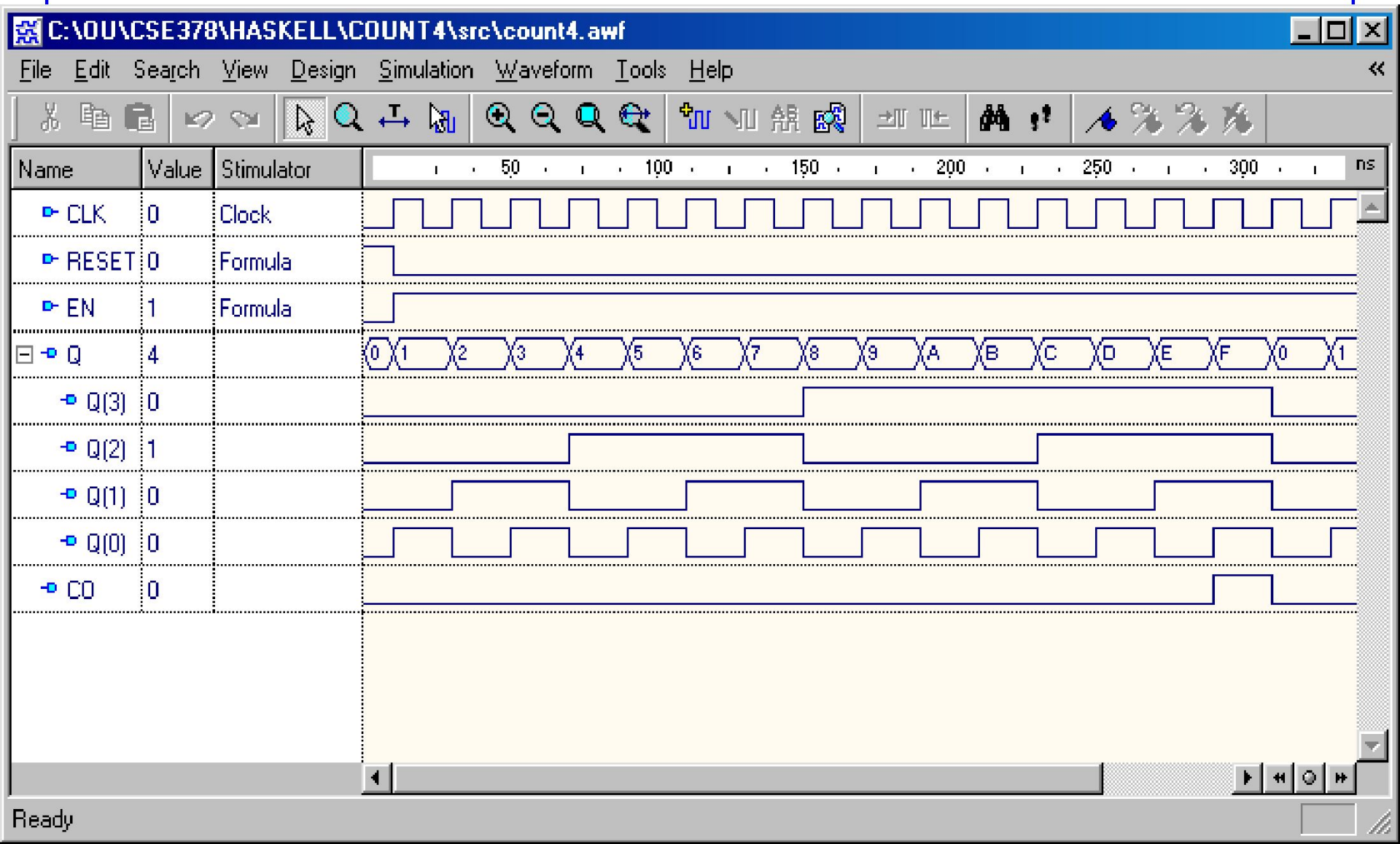
(b) Symbol

EN = enable control signal, when 0, counter remains in the same state, when 1, it counts

CO = carry output signal, used to extend the counter to more stages

$J_{Q0} = 1 \cdot EN$
$K_{Q0} = 1 \cdot EN$
$J_{Q1} = Q_0 \cdot EN$
$K_{Q1} = Q_0 \cdot EN$
$J_{Q2} = Q_0 Q_1 \cdot EN$
$K_{Q2} = Q_0 Q_1 \cdot EN$
$J_{Q3} = Q_0 Q_1 Q_2 \cdot EN$
$K_{Q3} = Q_0 Q_1 Q_2 \cdot EN$
$CO = Q_0 Q_1 Q_2 Q_3 \cdot EN$

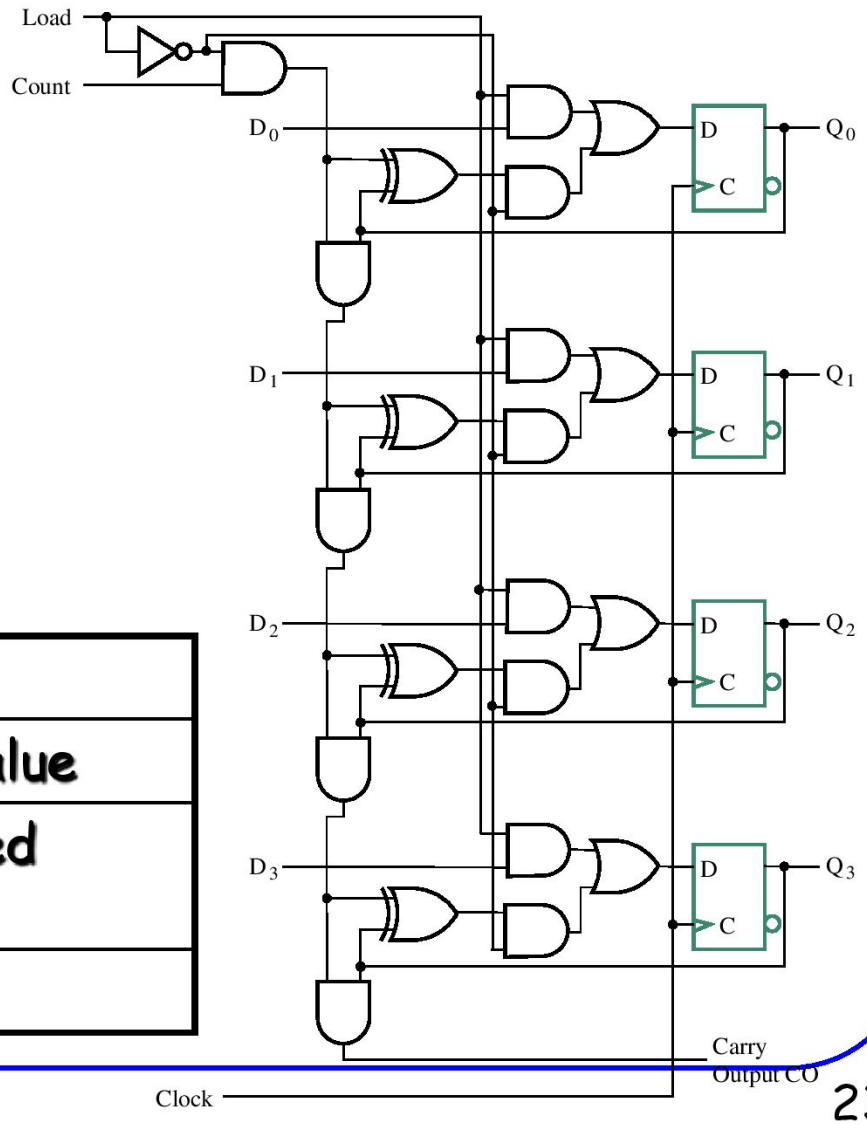
4-bit Upward Synchronous Binary Counter in HDL - Simulation



Counter with Parallel Load

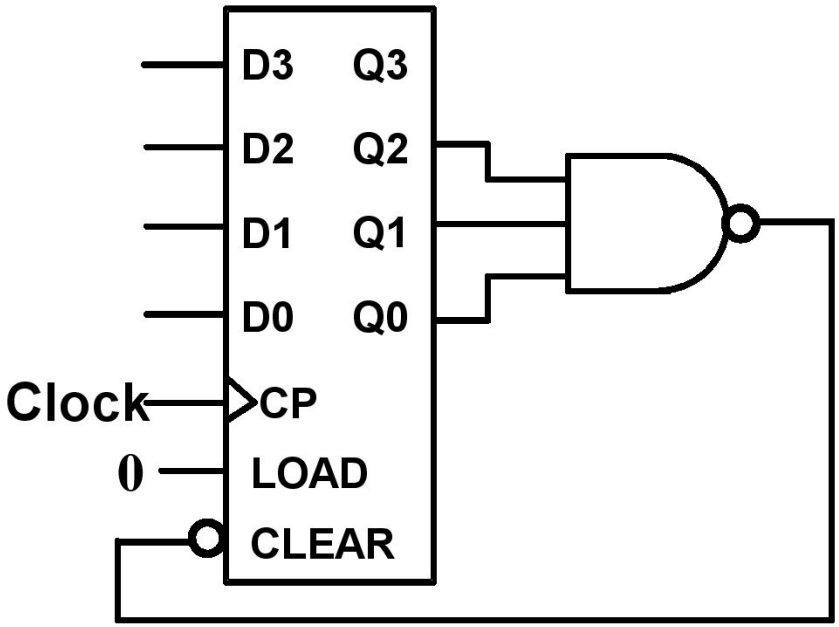
- **Design a path for input data**
  - enabled if Load = 1
- **Design some logic to:**
  - disable count logic when Load = 1
  - disable feedback from outputs when Load = 1
  - hold the stored value when Load = 0 and Count = 0
  - enable count logic when Load = 0 and Count = 1
- **The resulting function table:**

Load	Count	Action
0	0	Hold Stored Value
0	1	Count Up Stored Value
1	X	Load D



Counting Modulo 7: Detect 7 and Asynchronously Clear

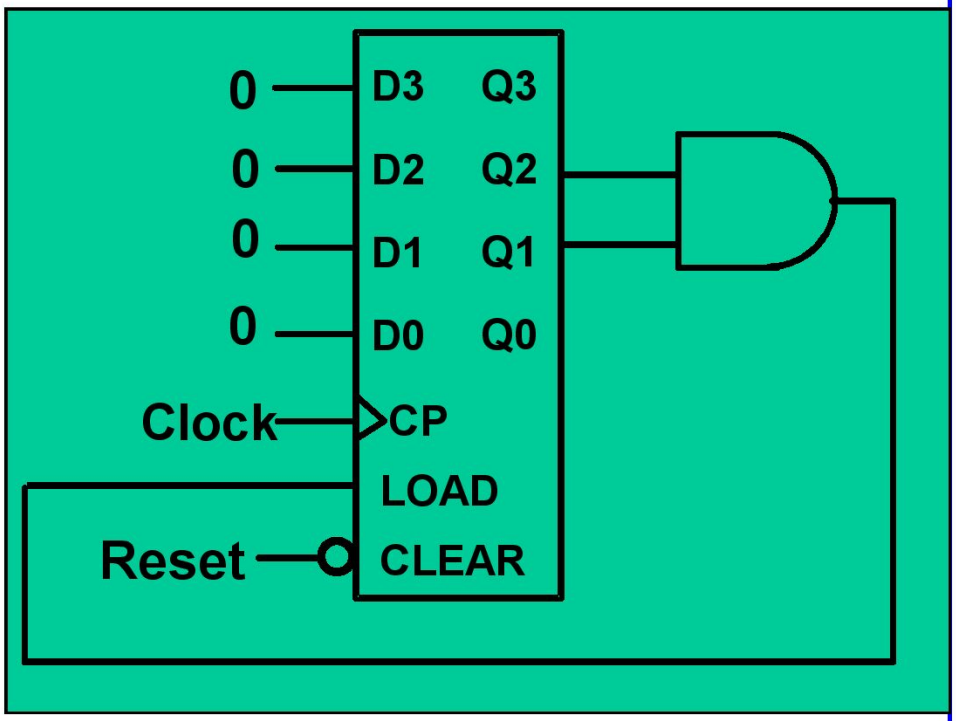
- In this figure, a synchronous 4-bit binary counter with an asynchronous Clear is used to make a Modulo 7 counter.
- It uses the Clear feature to detect the count 7 and clear the count to 0. This gives a count of 0, 1, 2, 3, 4, 5, 6, 7(short)0, 1, 2, 3, 4, 5, 6, 7(short)0, etc.



- Do NOT do this because count "7" shows up momentarily!

Counting Modulo 7: Synchronously Load on Terminal Count of 6

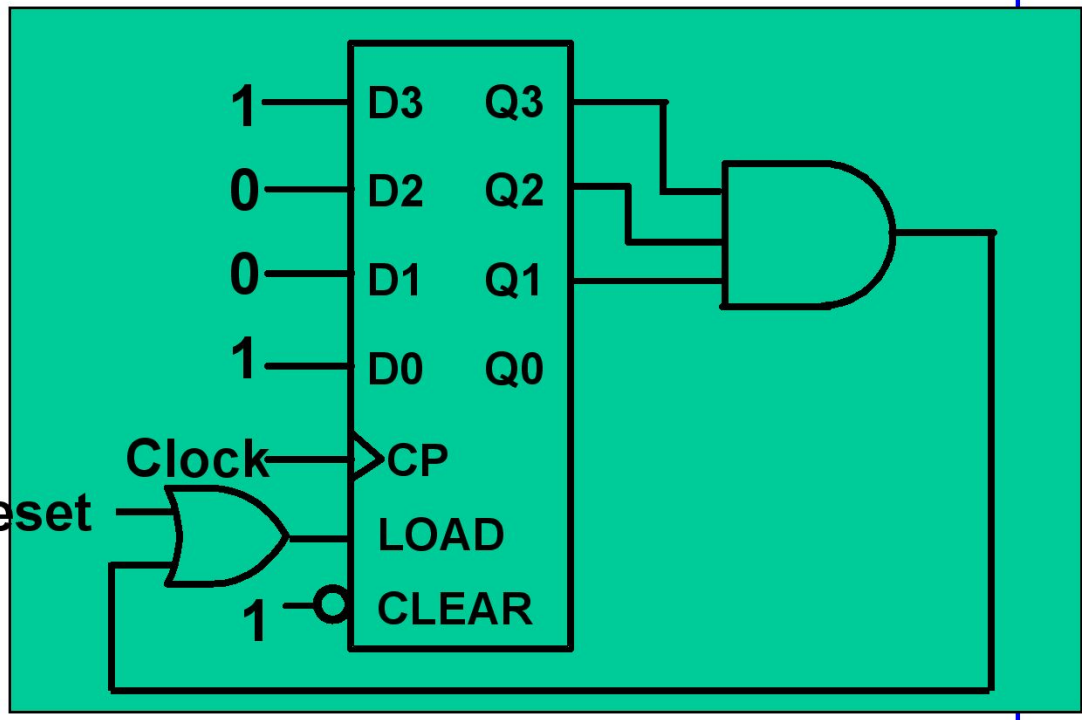
- A synchronous 4-bit binary counter with a synchronous load and an asynchronous clear is used to make a Modulo 7 counter
- Use the Load feature to detect the count "6" and load in "zero". This gives a count of 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, ...



- Using don't cares for states above 0110, detection of 6 can be done with Load = Q2.Q1

Counting Modulo 6: Synchronously Preset 9 on Reset and Load 9 on Terminal Count 14

- A synchronous, 4-bit binary counter with a synchronous Load is to be used to make a Modulo 6 counter.
- Use the Load feature to preset the count to 9 on Reset and detection of count 14.



- This gives a count of 9, 10, 11, 12, 13, 14, 9, 10, 11, 12, 13, 14, 9, …
- If the terminal count is 15, detection is usually built in as Carry Out (CO)

Ripple Counter Concept

Present state				Next state			
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	1	0	1	0
1	0	1	0	1	0	1	1
1	0	1	1	1	1	0	0
1	1	0	0	1	1	0	1
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	0	0	0	0

Toggles at every clock

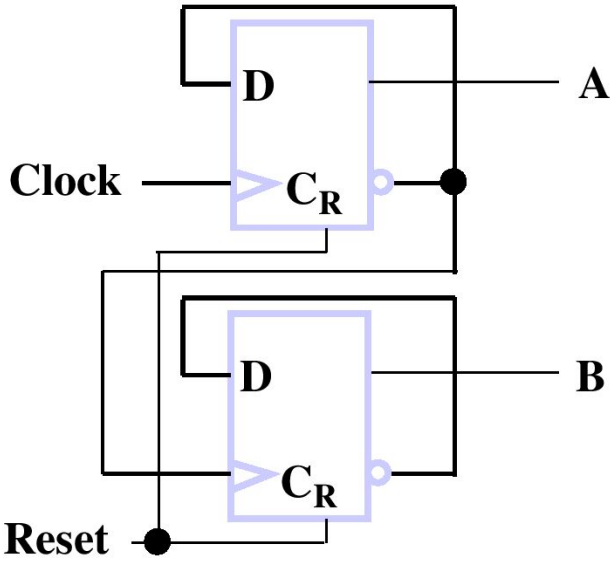
Toggles every 2 clocks

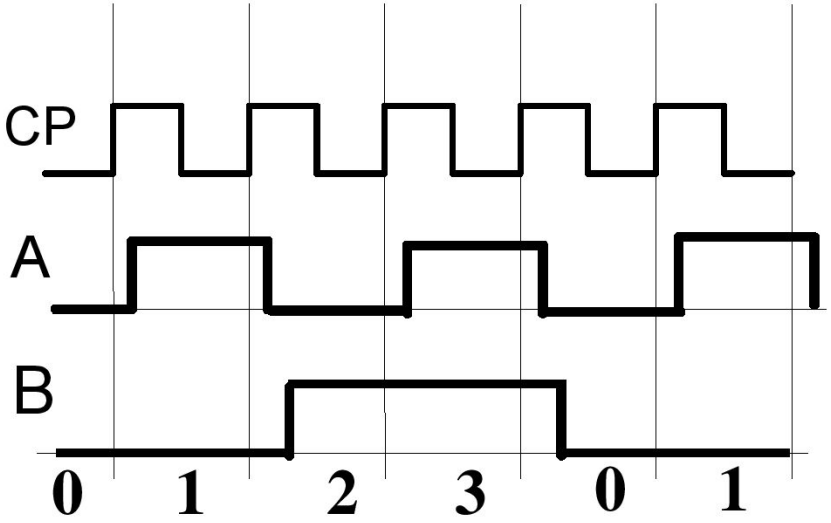
Toggles every 8 clocks

Toggles every 4 clocks

Ripple Counter

- **A 2-bit binary counter**
- **How does it work?**
  - When there is a positive edge on the clock input of A, A toggles
  - The clock input for flip-flop B is the complemented output of flip-flop A
  - When flip-flop A changes from 1 to 0, there is a positive edge on the clock input of B causing B to toggle





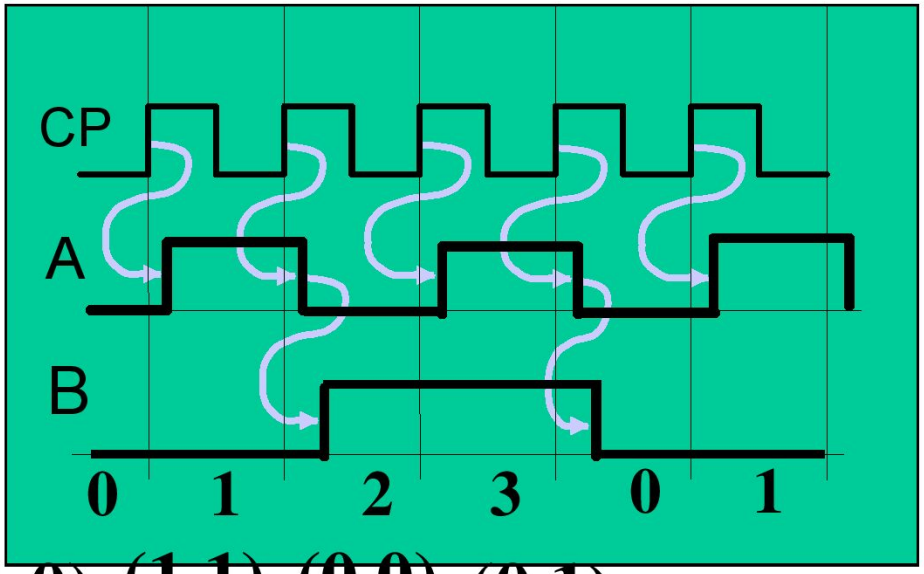
Example (Cont'd)

- **The main idea is that:**
  - Q_0 toggles at half the clock rate
  - Q_1 toggles at half Q_0 rate
  - Q_2 toggles at half Q_1 rate
  - And so on…

Upward Counting Sequence			
Q_3	Q_2	Q_1	Q_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Ripple Counter (Cont'd)

- The arrows show the cause-effect relation-ship from the prior slide =>
- The corresponding sequence of states => (B,A) = (0,0),

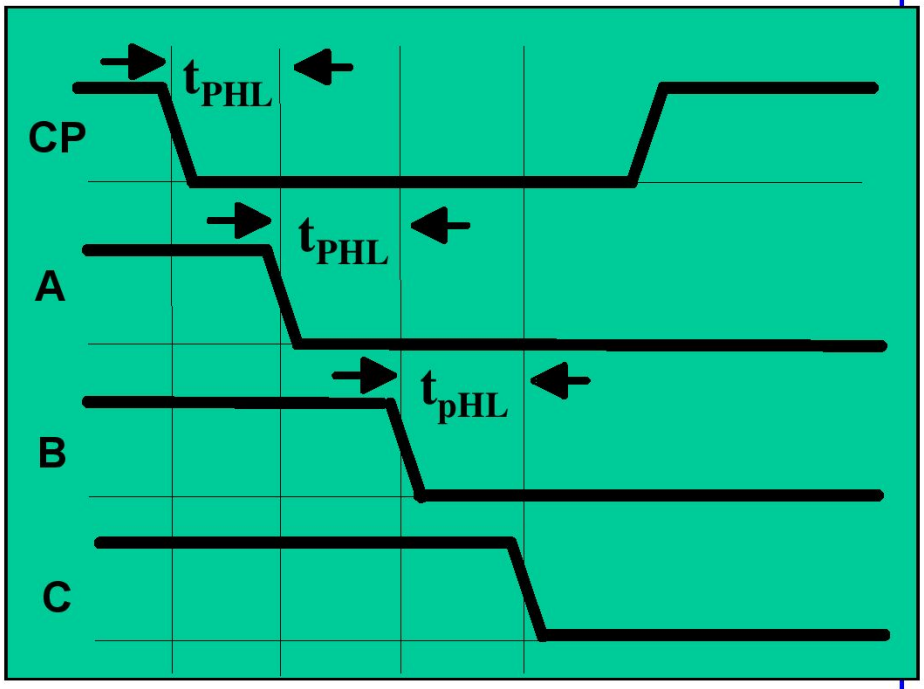


(0,1), (1,0), (1,1), (0,0), (0,1), …

- Each additional bit, C, D, …behaves like bit B, changing half as frequently as the bit before it.
- For 3 bits: (C,B,A) = (0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1), (0,0,0), …

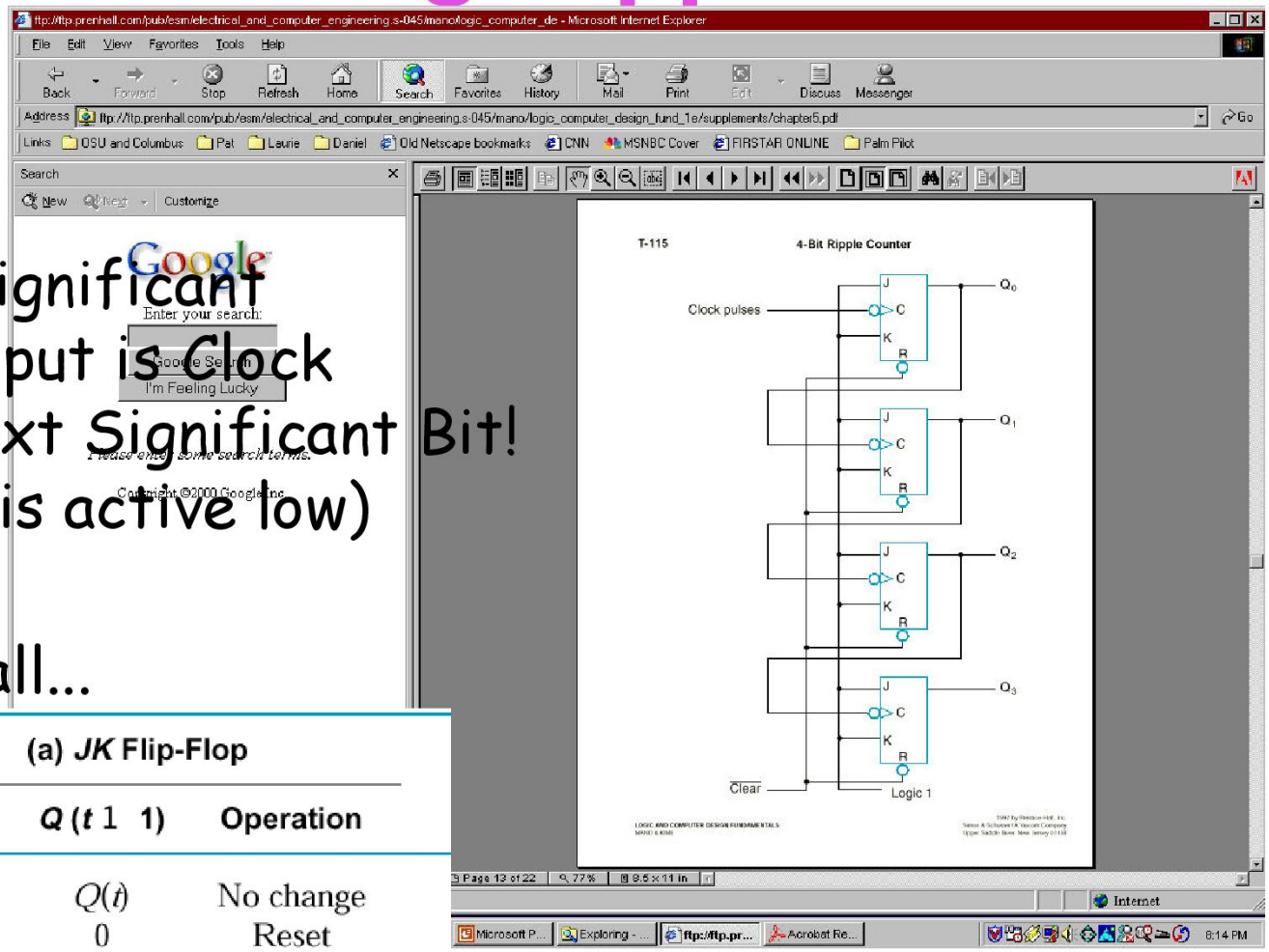
Ripple Counter (Cont'd)

- **Starting with C = B = A = 1, equivalent to (C,B,A) = 7 base 10, the next clock increments the count to (C,B,A) = 0 base 10. In fine timing detail:**
  - The clock to output delay t_{PHL} causes an increasing delay from clock edge for each stage transition.
  - Thus, the count "ripples" from least to most significant bit.
  - For *n* bits, total worst case delay is *n* t_{PHL}.



Example: A 4-bit Upward Counting Ripple Counter

Less Significant Bit output is Clock for Next Significant Bit! (Clock is active low)



Recall...

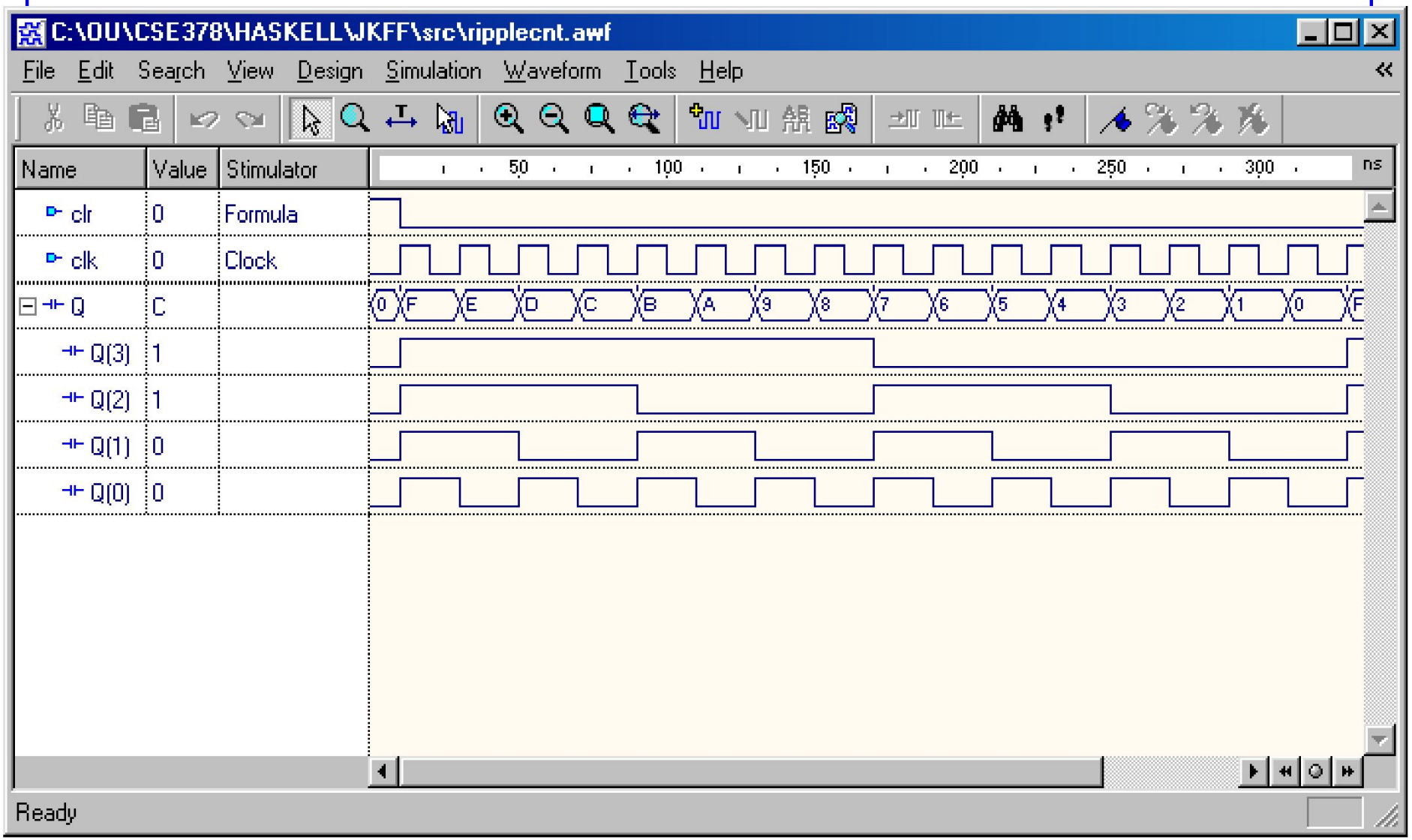
(a) *JK* Flip-Flop

J	K	$Q(t+1)$	Operation
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	$\overline{Q}(t)$	Complement

A 4-bit Downward Counting Ripple Counter

- **Use direct Set (S) signals instead of direct Reset (R), in order to start at 1111.**
- **Alternative designs:**
  - Change edge-triggering to positive
  - Connect the complement output of each FF to the C input of the next FF in the sequence.

Simulation ...


C:\OU\CSE378\HASKELL\JKFF\src\ripplecnt.awf
File Edit Search View Design Simulation Waveform Tools Help
Name
Value
Stimulator
50
100
150
200
250
300
ns
clr
0
Formula
clk
0
Clock
Q
C
0 F E D C B A 9 8 7 6 5 4 3 2 1 0 F
Q(3)
1
Q(2)
1
Q(1)
0
Q(0)
0
Ready