

Chapter 7 – Counters and Registers

ELEVENTH EDITION

Digital Systems

Principles and Applications

Ronald J. Tocci

Monroe Community College

Neal S. Widmer

Purdue University

Gregory L. Moss

Purdue University

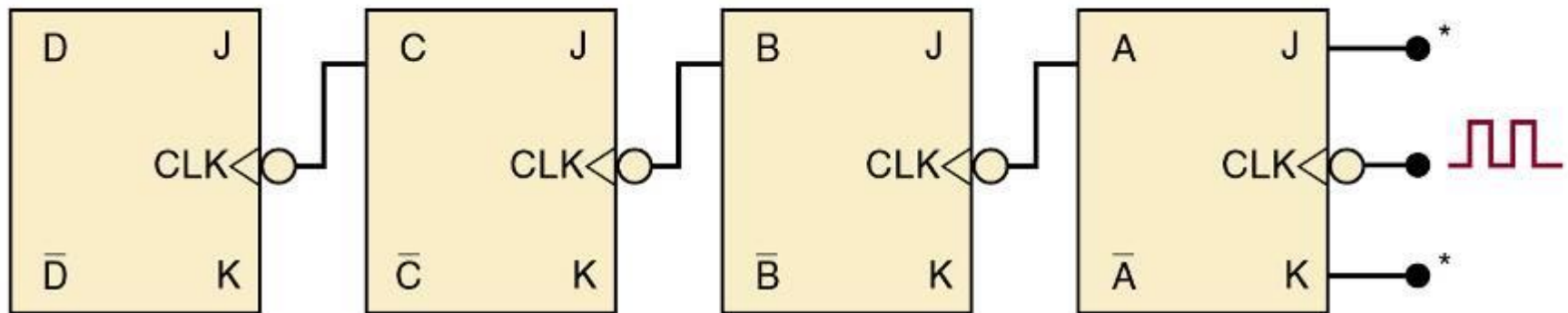
PEARSON

Chapter 7 Objectives

- *Selected areas covered in this chapter:*
 - Operation & characteristics of synchronous and asynchronous counters.
 - Analyzing and evaluating various types of counters.
 - Schemes used to decode different types of counters.
 - Counter circuits using different levels of abstraction in HDL.
 - Operation of various types of IC registers.
 - Shift registers and shift register counters using HDL.
 - Troubleshooting techniques used for combinational logic systems to troubleshoot sequential logic systems.

7-1 Asynchronous (Ripple) Counters

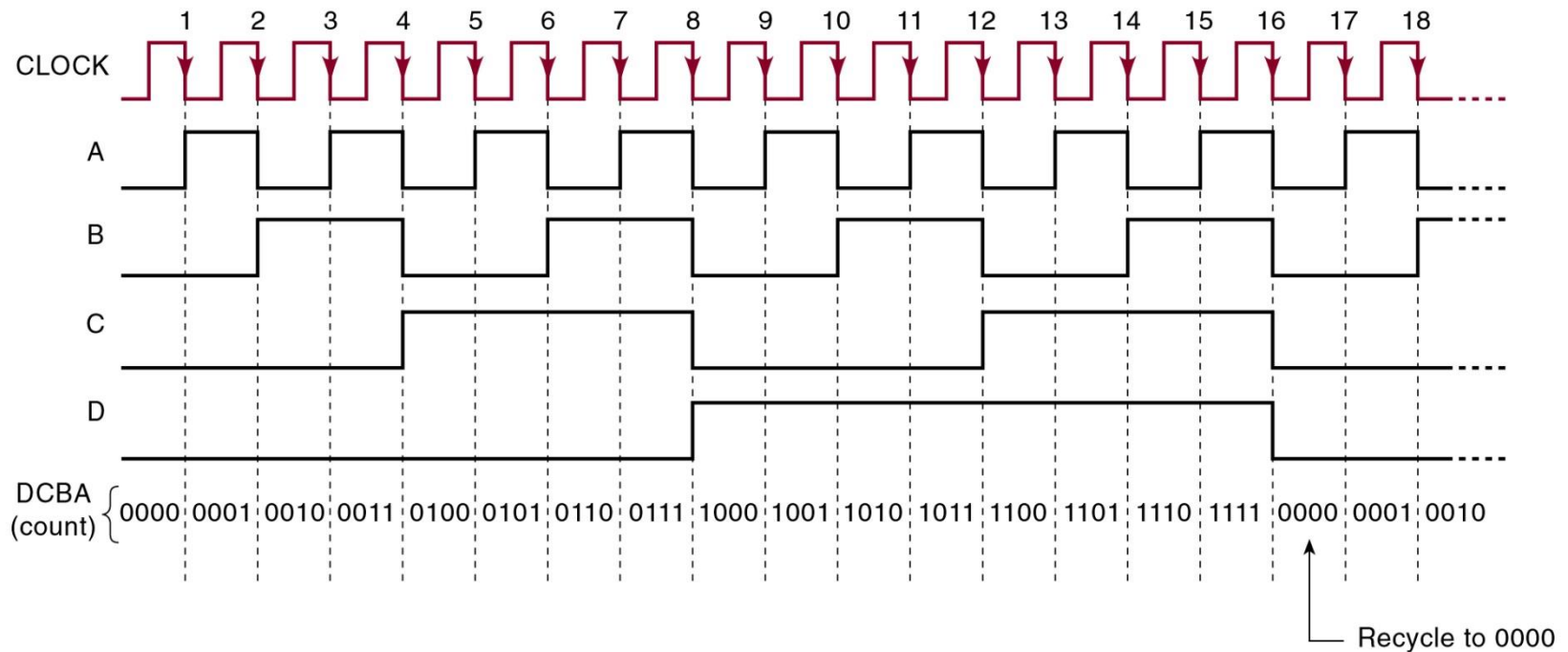
- Review of four bit counter operation:
 - Clock is applied only *CLK* input to FF *A*.
 - *J* & *K* are HIGH in all FFs.
 - Output of FF *A* is *CLK* of FF *B*, etc.
 - FF outputs *D*, *C*, *B*, and *A*, are a 4-bit binary number with *D* as the MSB.



*All *J* and *K* inputs
assumed to be 1.

7-1 Asynchronous (Ripple) Counters

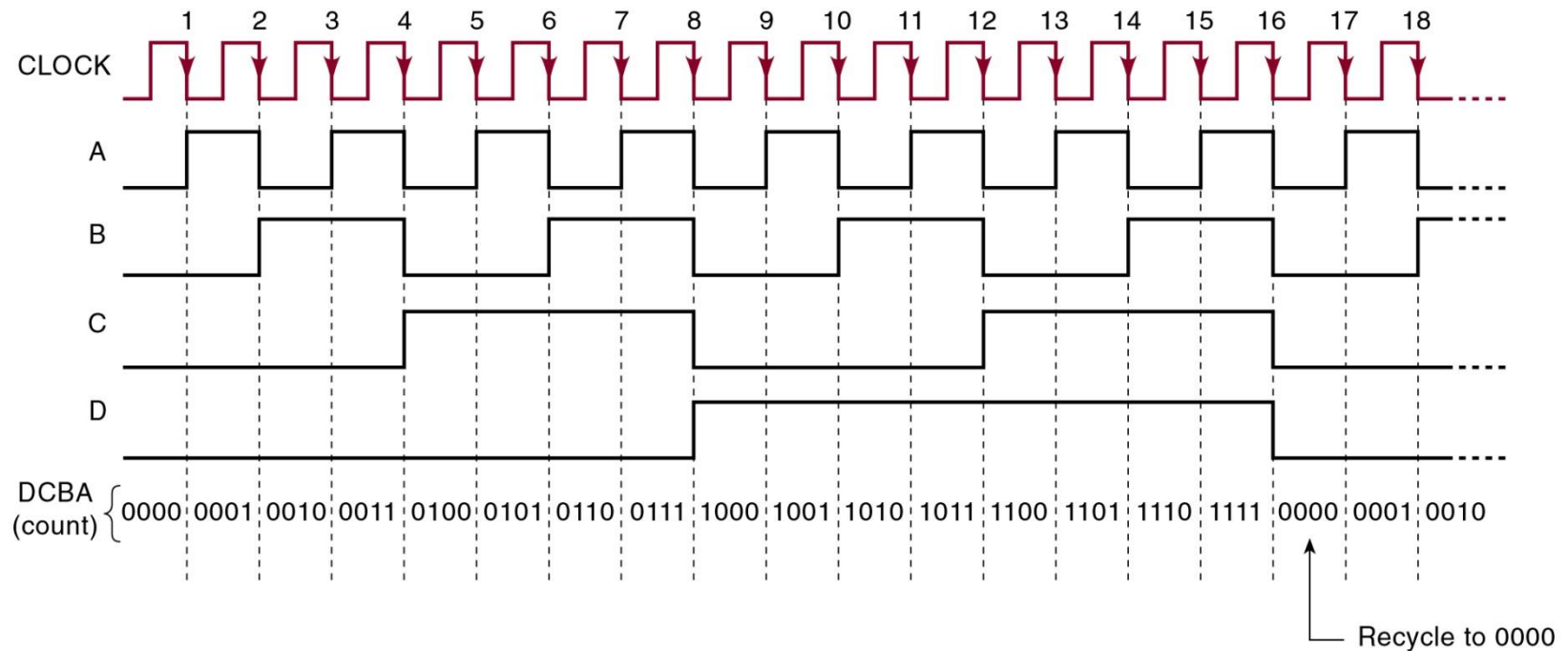
- Review of four bit counter operation:
 - After the NGT of the 15th clock pulse, the counter FFs are *recycled* back to 0000.



An asynchronous counter—state is *not* changed in exact synchronism with the clock.

7-1 Asynchronous (Ripple) Counters – MOD Number

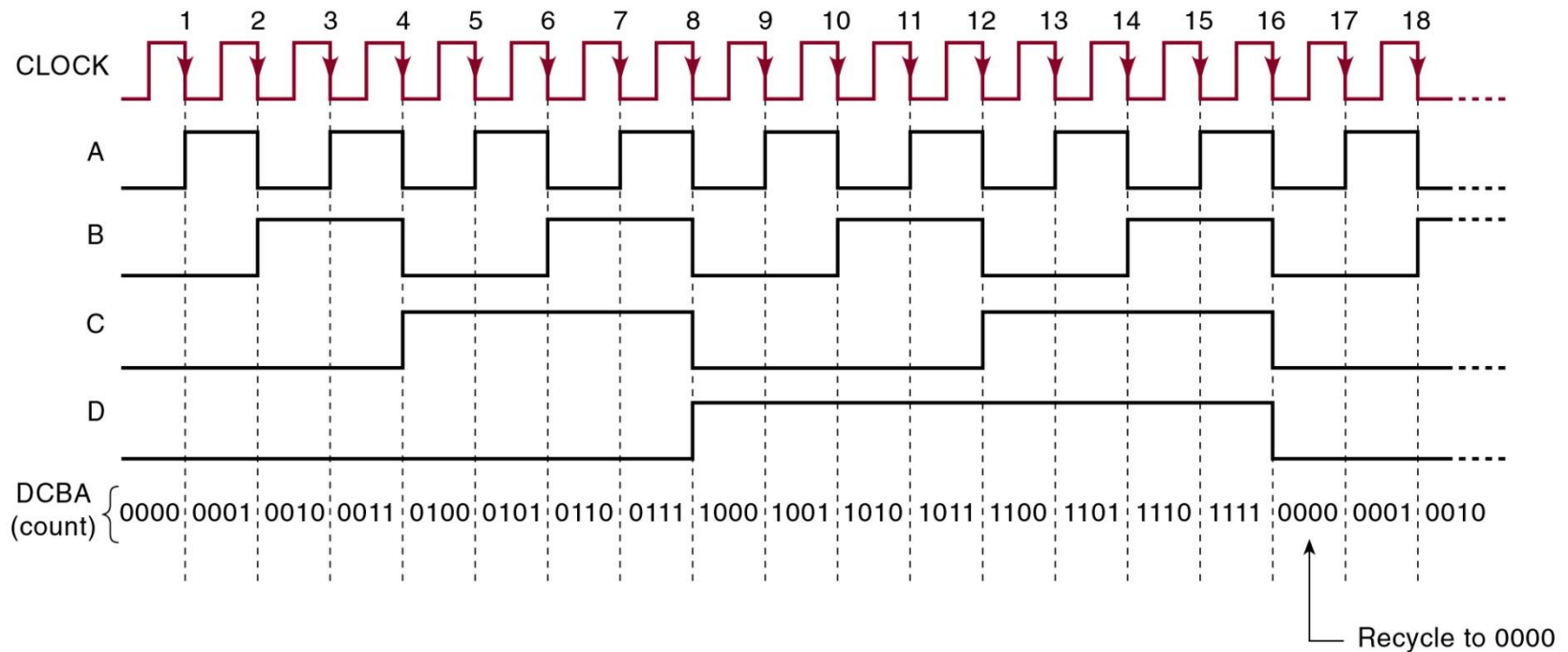
- MOD number is equal to the number of states the counter goes through before recycling.



Adding FFs will increase the MOD number.

7-1 Asynchronous (Ripple) Counters – MOD Number

- Frequency division – each FF will have an output frequency of 1/2 the input.



Output frequency of the last counter FF will be clock frequency divided by MOD of the counter.

7-1 Asynchronous (Ripple) Counters – Signal Flow

- Schematics are normally drawn from left to right.
 - Counters will be drawn from right to left so that the MSB and LSB appear in the appropriate positions.

7-2 Propagation Delay in Ripple Counters

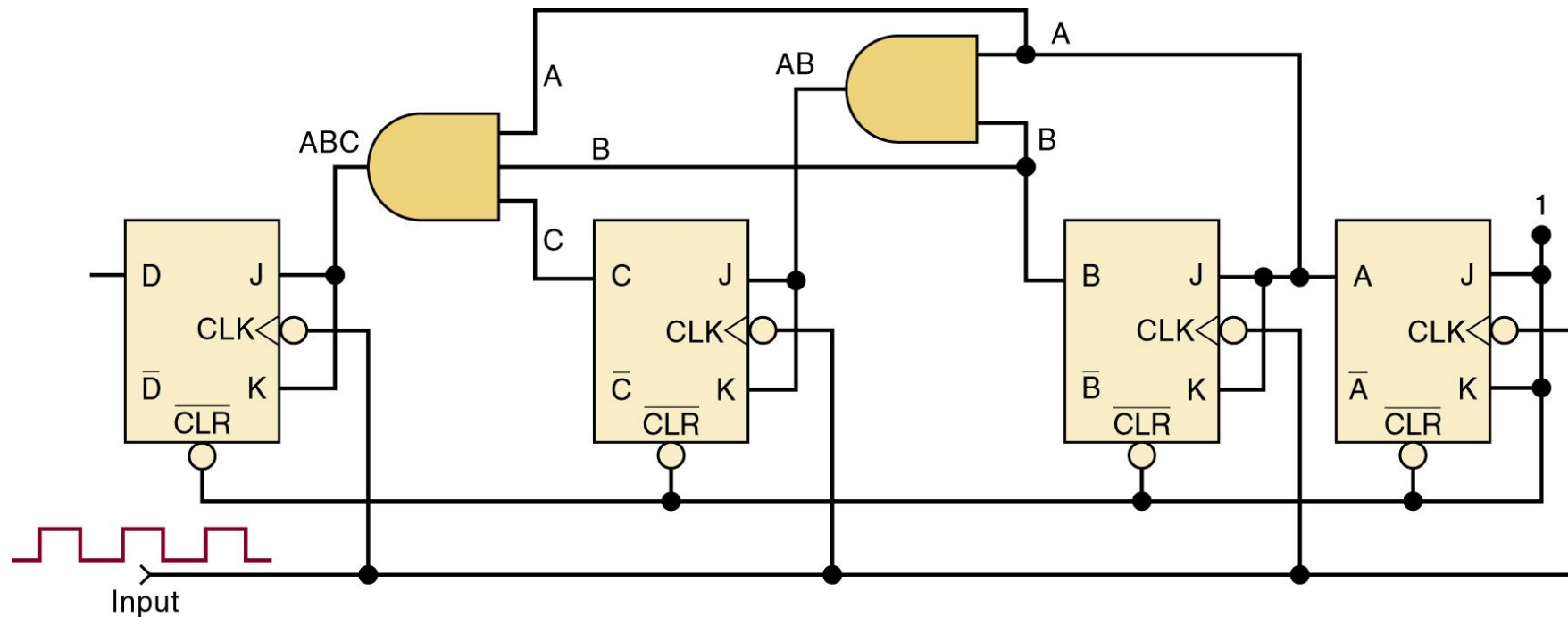
- Ripple counters are simple—requiring the fewest components to produce a given operation.
 - Cumulative propagation delay can cause problems at high frequencies.
- If the period between input pulses is made longer than the total propagation delay of the counter, problems can be avoided
 - For proper operation: $T_{\text{clock}} \geq N \times t_{\text{pd}}$
 - Maximum frequency: $F_{\text{max}} = 1/N \times t_{\text{pd}}$

7-2 Propagation Delay in Ripple Counters

- Asynchronous counters are not useful at very high frequencies—especially for counters with large numbers of bits.
 - Erroneous count patterns can generate **glitches**
 - Signals produced by systems using asynchronous counters.

7-3 Synchronous (Parallel) Counters

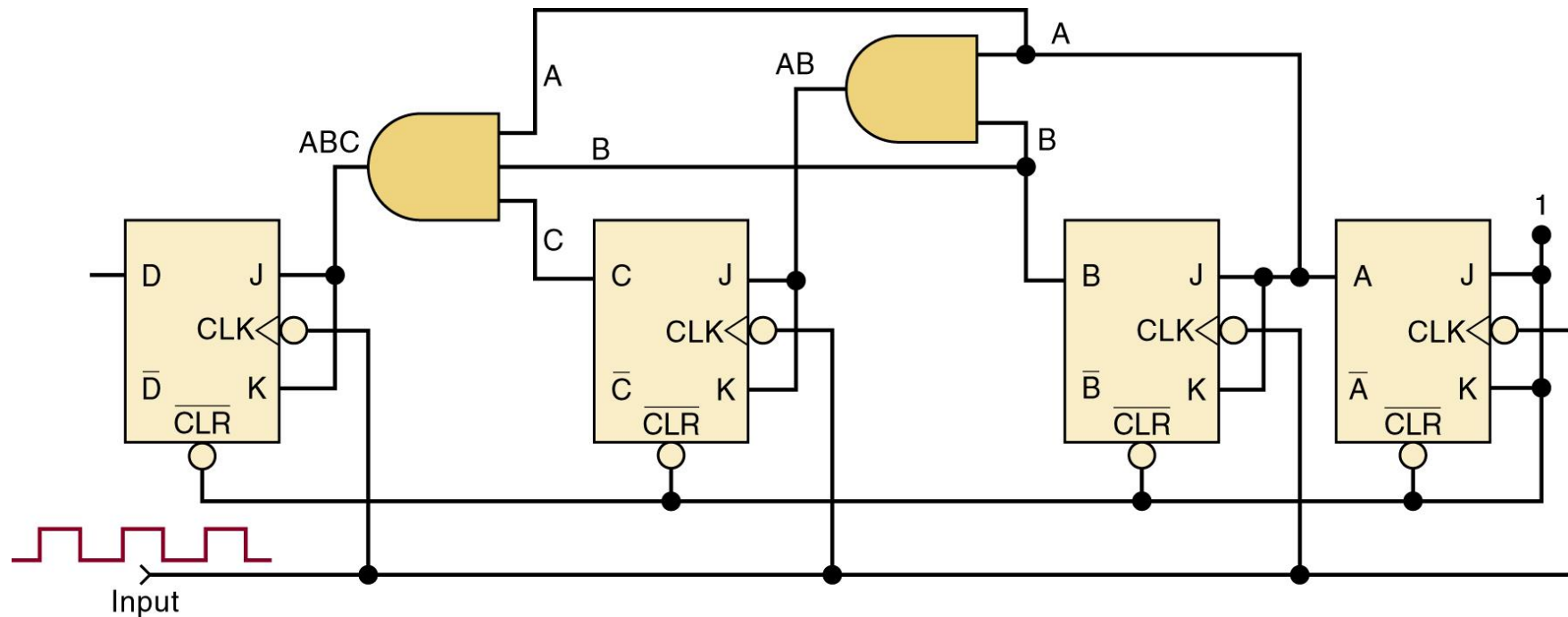
- In **synchronous** or **parallel counters**, all FFs are triggered simultaneously (in parallel) by the clock.



Synchronous counters can operate at much higher frequencies than asynchronous counters.

7-3 Synchronous (Parallel) Counters

- Each FF has J & K inputs which are HIGH only when outputs of all lower-order FFs are HIGH.



The total propagation delay will be the *same* for *any* number of FFs.

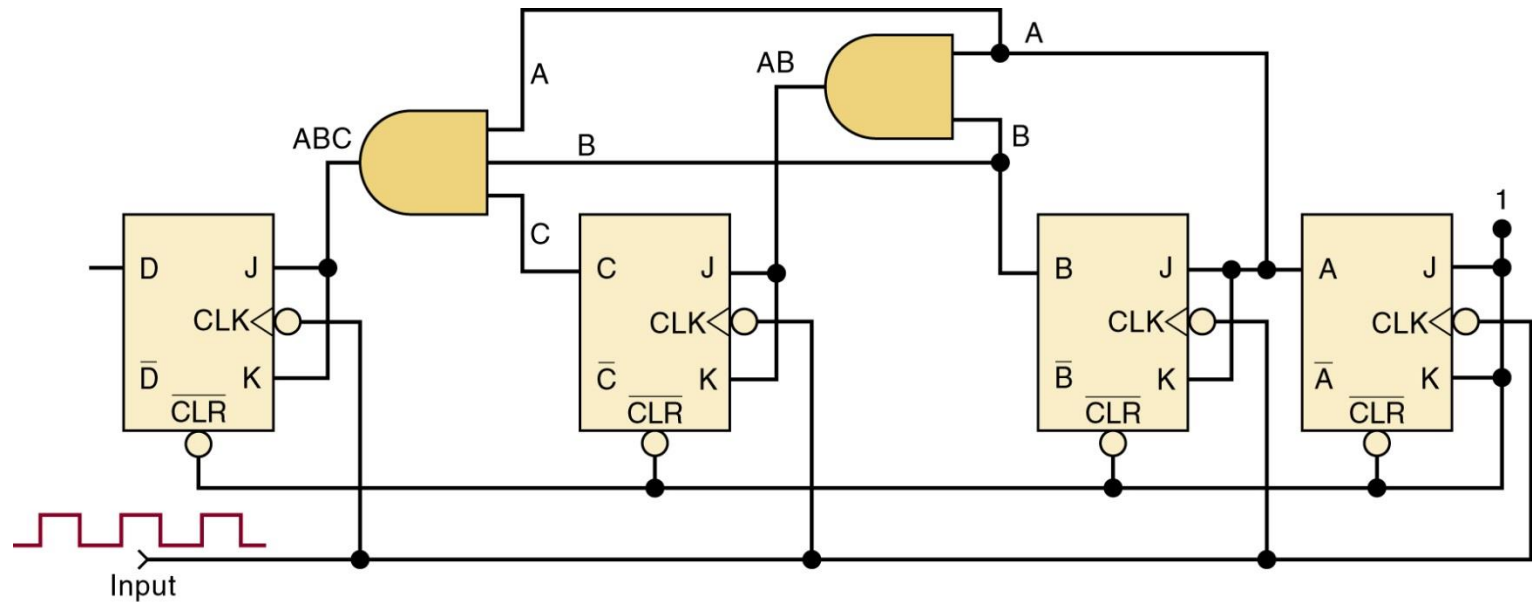
7-3 Synchronous (Parallel) Counters

For this circuit to count properly, on a given NGT of the clock, only those FF that are supposed to toggle on that NGT should have $J = K = 1$ when NGT occurs.

Count	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
0	0	0	0	0
.
.
.	.	etc.	.	.

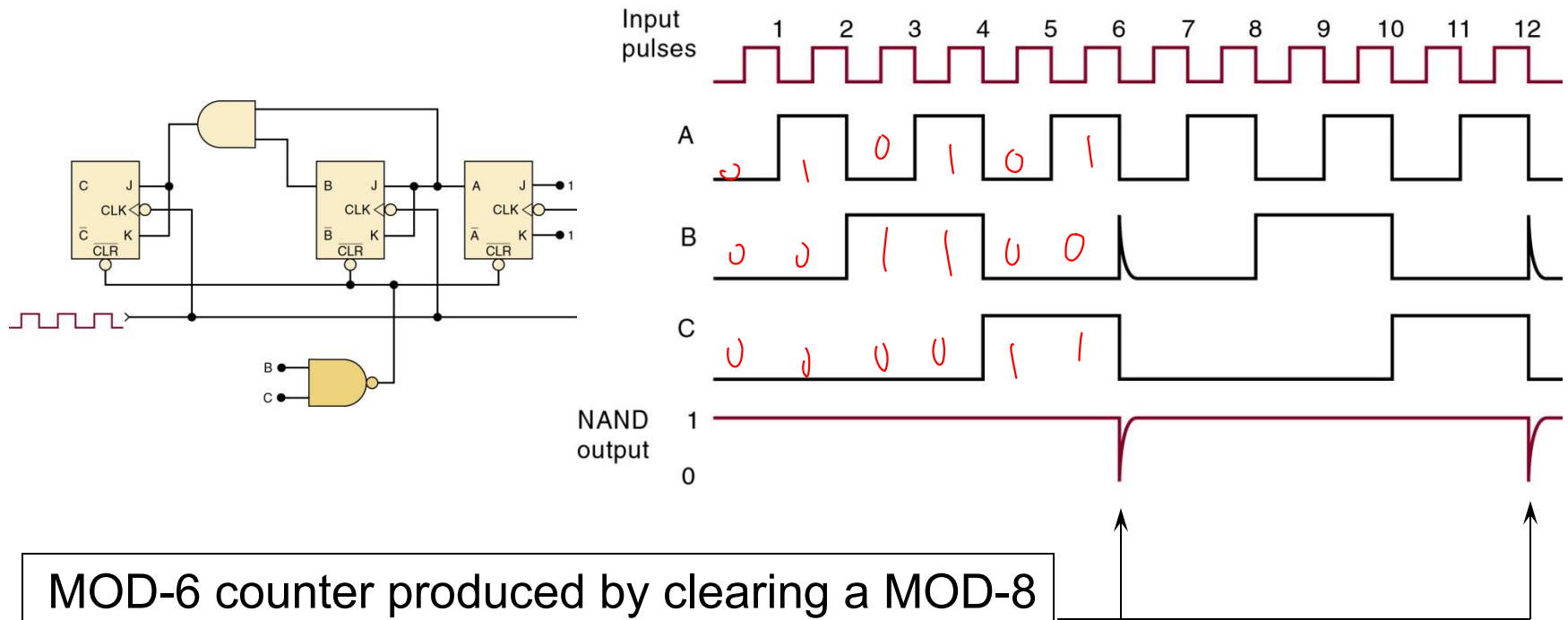
7-4 Counters with MOD Number $< 2^N$

- The basic synchronous counter shown is limited to MOD numbers that are equal to 2^N .
 - Where N is the number of FFs.



7-4 Counters with MOD Number $< 2^N$

- The basic counter can be modified to produce MOD numbers *less* than 2^N .
 - By allowing the counter to *skip states* that are normally part of the counting sequence.

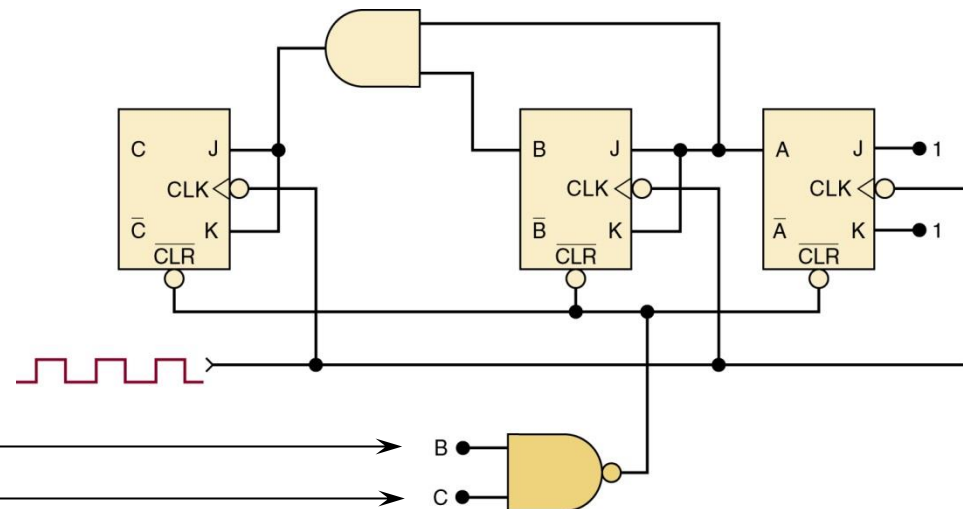


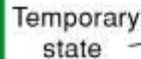
MOD-6 counter produced by clearing a MOD-8 counter when a count of six (110) occurs.

7-4 Counters with MOD Number $\leq 2^N$

- Changing the MOD number.
 - Find the smallest MOD required so that 2^N is less than or equal to the requirement.
 - Connect a **NAND** gate to the asynchronous CLEAR inputs of all FFs.

Determine which FFs are **HIGH** at the desired count and connect the outputs of these FFs to the **NAND** gate inputs.





advance to that state.

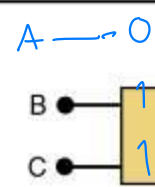
The 111 state *can* occur

Each circle represents one of

Arrows indicate how one state

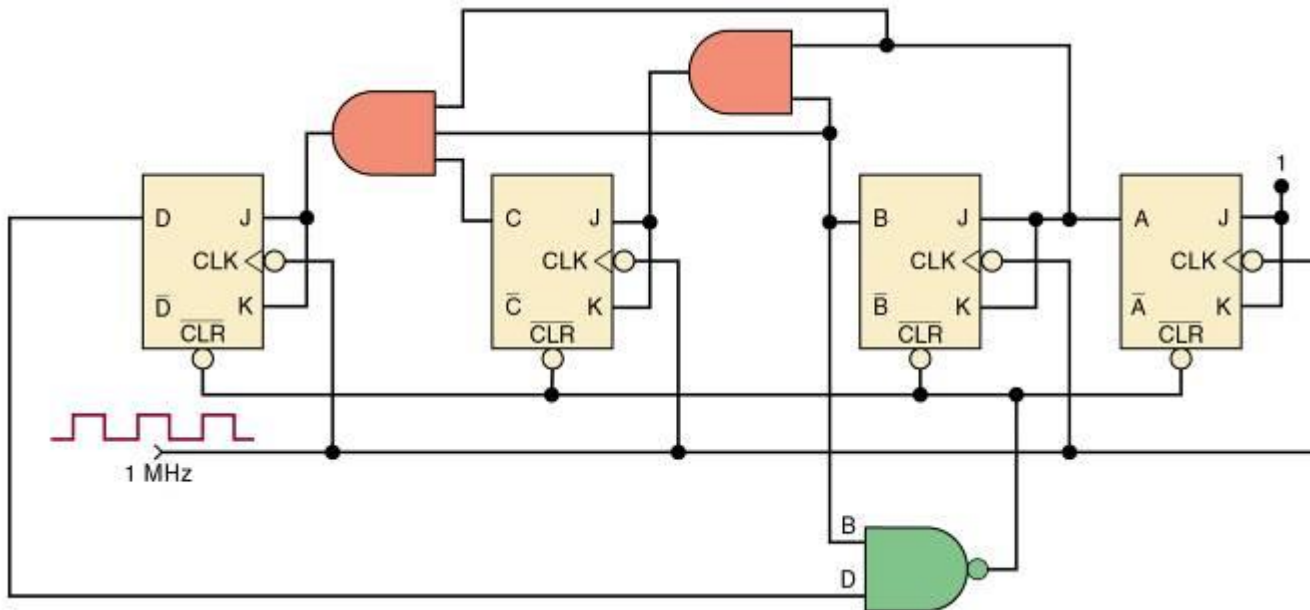


**The MOD-6 counter
lights the LED on
HIGH output.**



7-4 Counters with MOD Number $< 2^N$

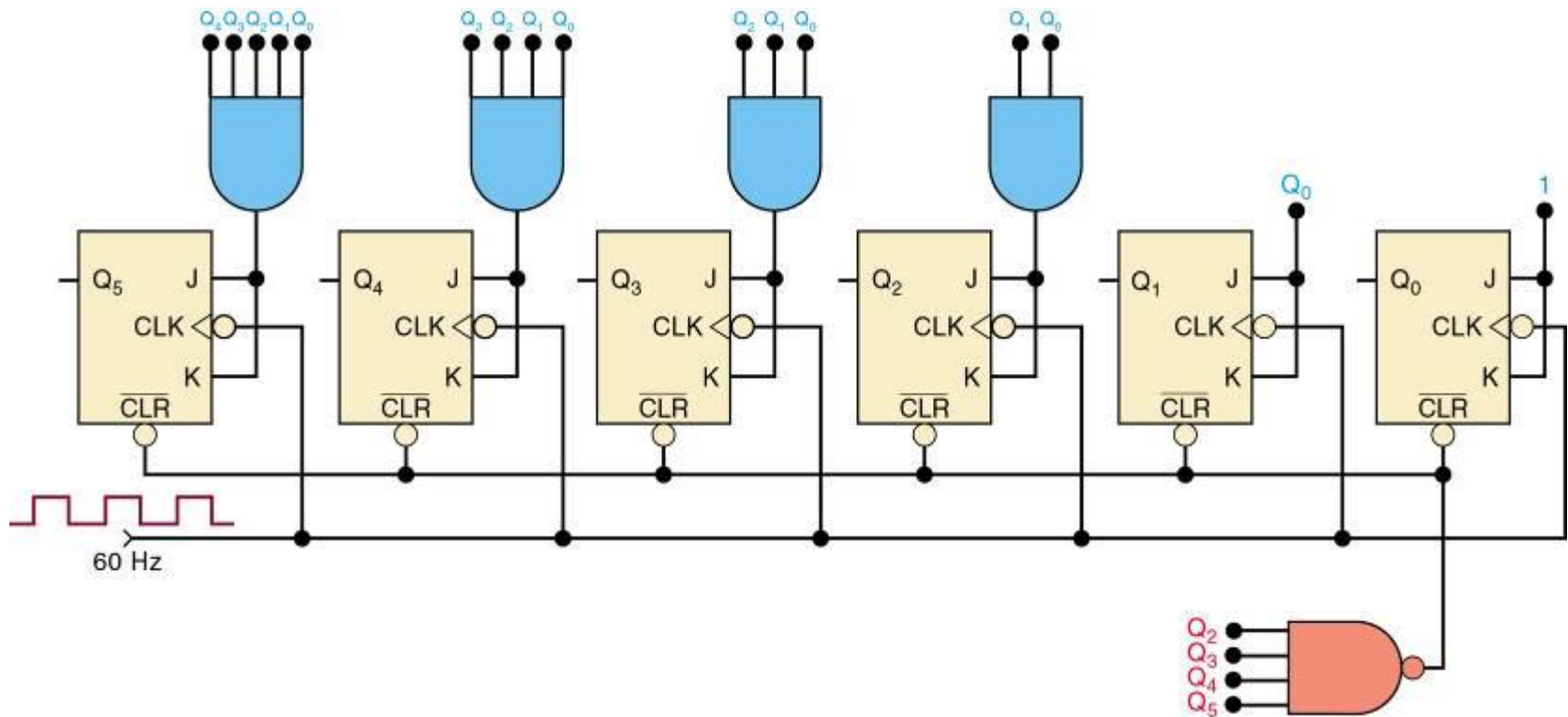
- Decade counters are widely used for counting events, and displaying results in decimal form.
 - A decade counter is any counter with 10 distinct states, regardless of the sequence.
 - A BCD counter is a decade counter that counts from binary 0000 to 1001.



Any MOD-10 counter is a decade counter.

7-4 Counters with MOD Number $< 2^N$

Mod-60 Counter

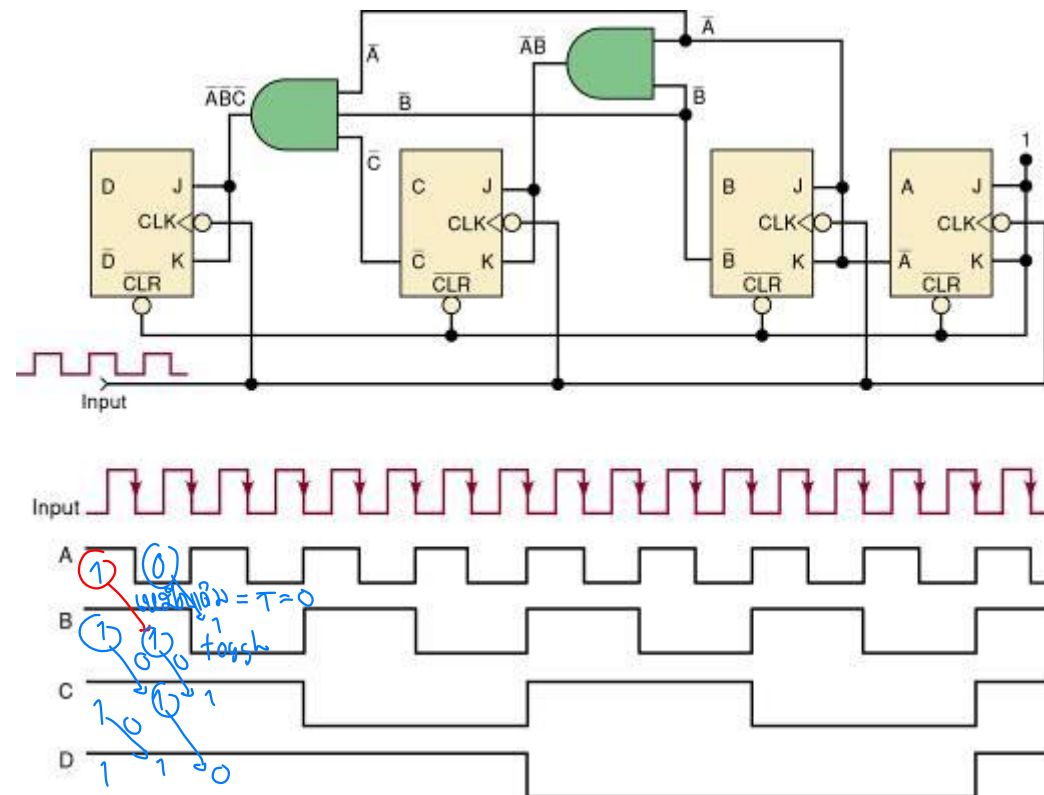


↳ $\lim_{x \rightarrow 0} \frac{1}{x} = \infty$

7-5 Synchronous Down and Up/Down Counters

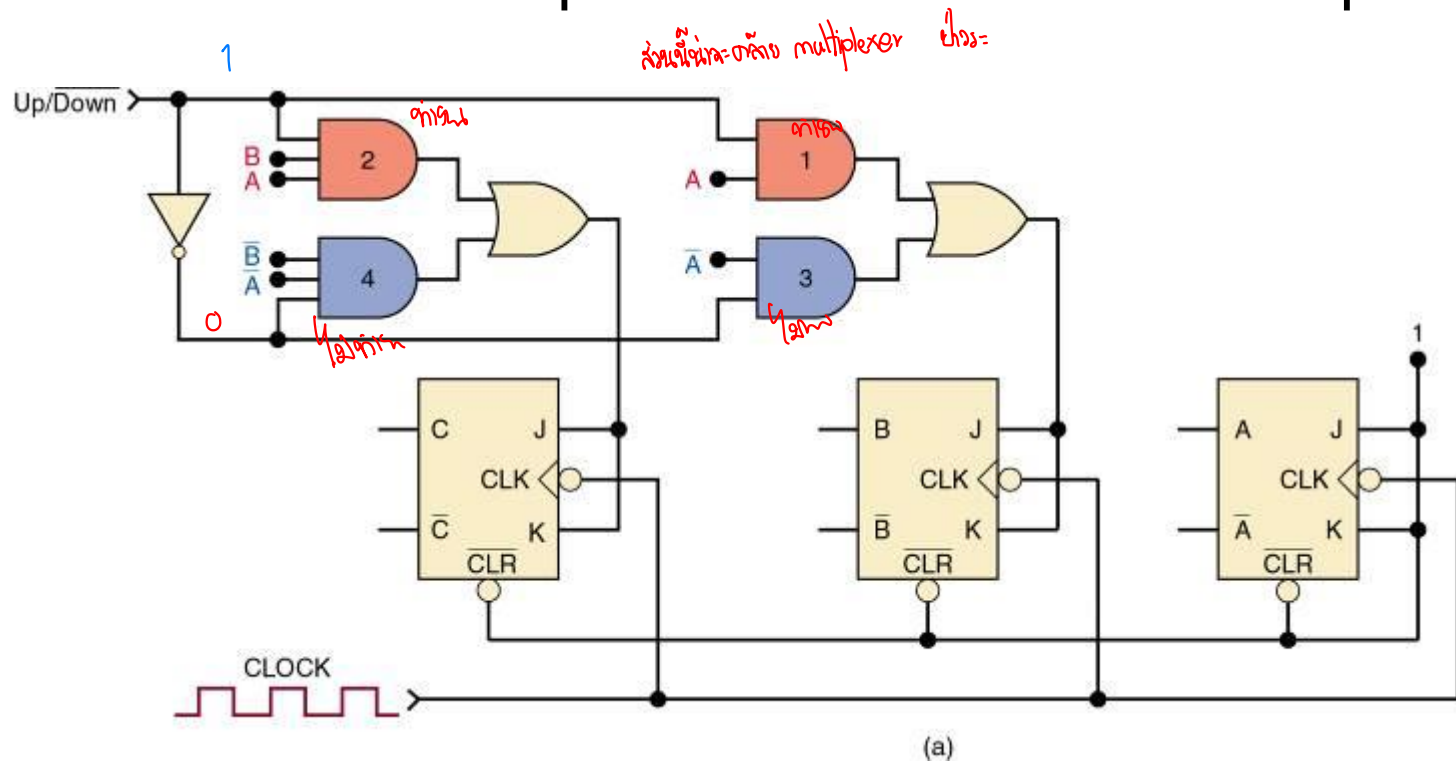
- A synchronous **down counter** is constructed in a similar manner to an up counter.
 - It uses the inverted FF outputs to control the higher-order J, K inputs.

**Synchronous, MOD-16,
down counter and
output waveforms.**



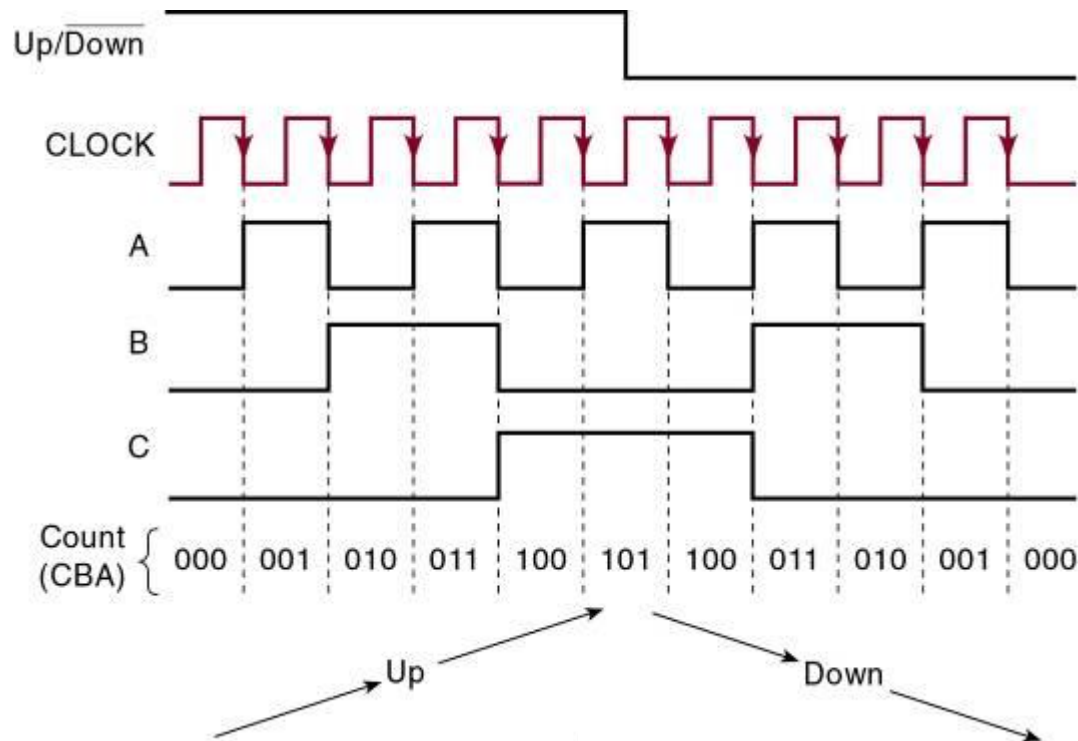
7-5 Synchronous Down and Up/Down Counters

- In a parallel **up/down counter**, the control input controls the values fed to the J and K inputs of the successive FFs.
 - The *normal* FF outputs or the *inverted* FF outputs.



7-5 Synchronous Down and Up/Down Counters

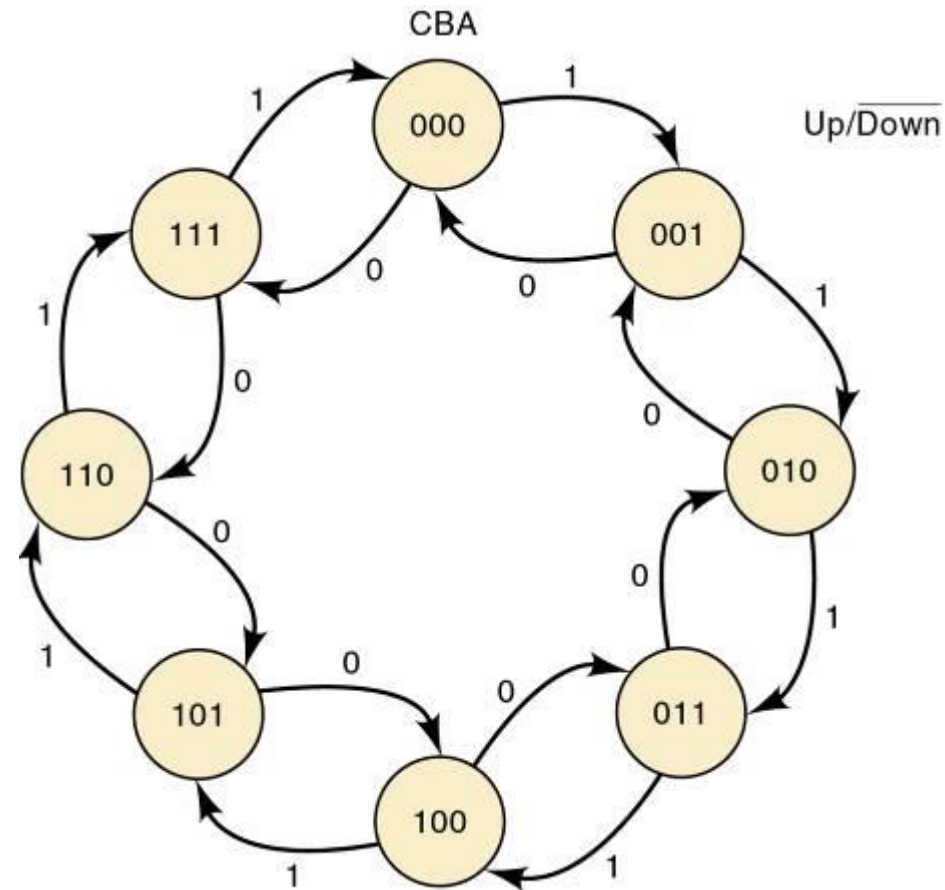
- For the first five clock pulses, $\text{Up/Down} = 1$.
 - The counter counts up.
- For the last five pulses, $\text{Up/Down} = 0$.
 - The counter counts down.



7-5 Synchronous Down and Up/Down Counters

- Note that there are two arrows leaving each state's bubble—a **conditional transition**.

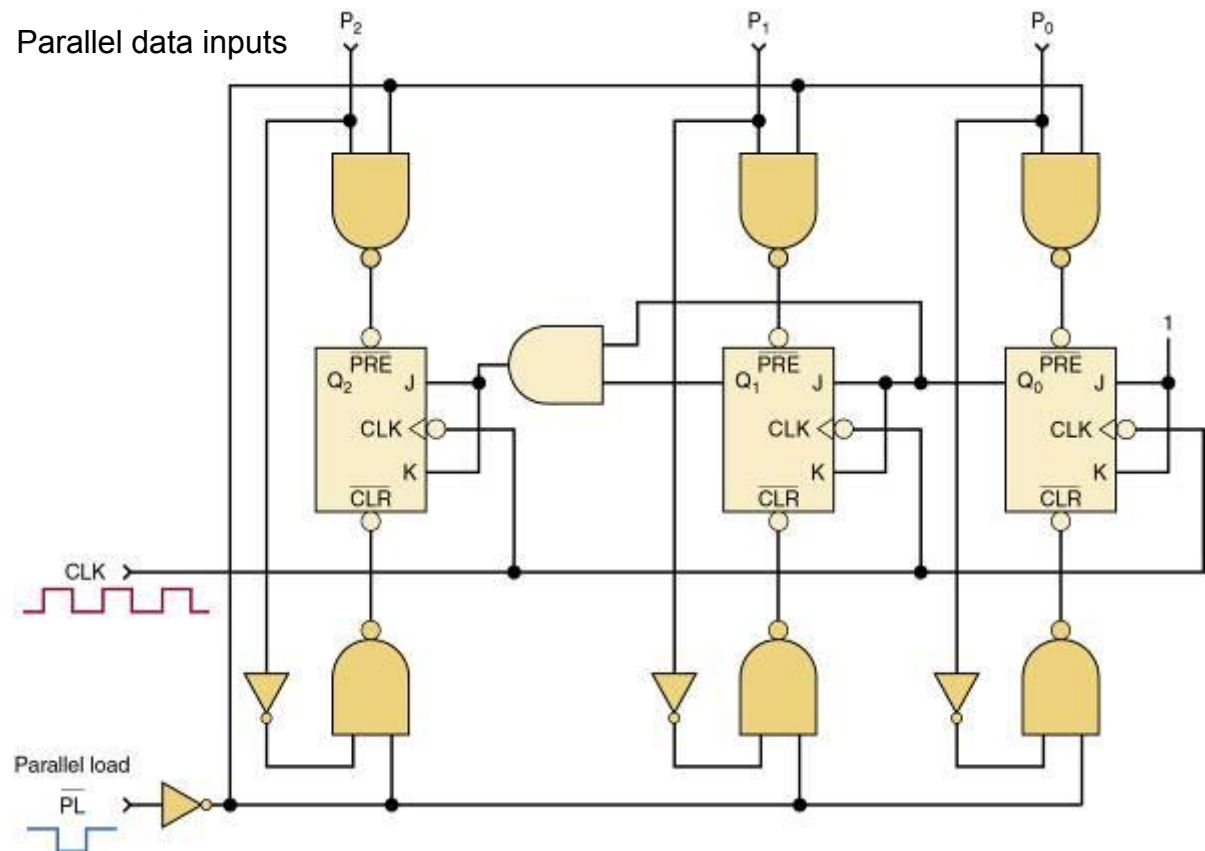
The next state for this counter is dependent upon the logic level applied to the control input.



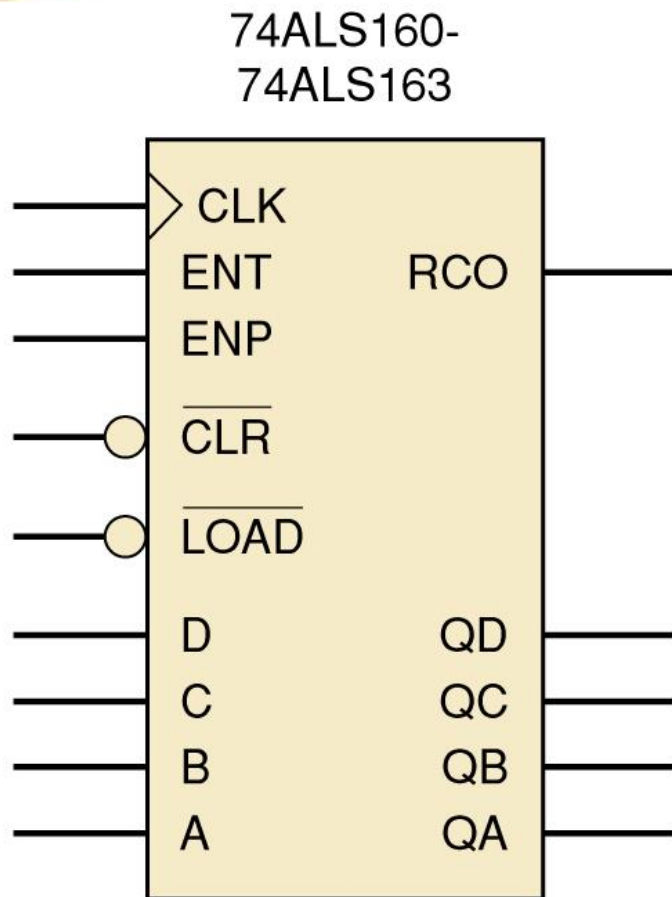
7-6 Presettable Counters

- A presettable counter can be set to any desired starting point—synchronously or asynchronously.
 - Preset is also called *parallel loading* the counter.

**Synchronous
counter with
asynchronous
parallel load.**



7-7 IC Asynchronous Counters



Counter contains four FFs.

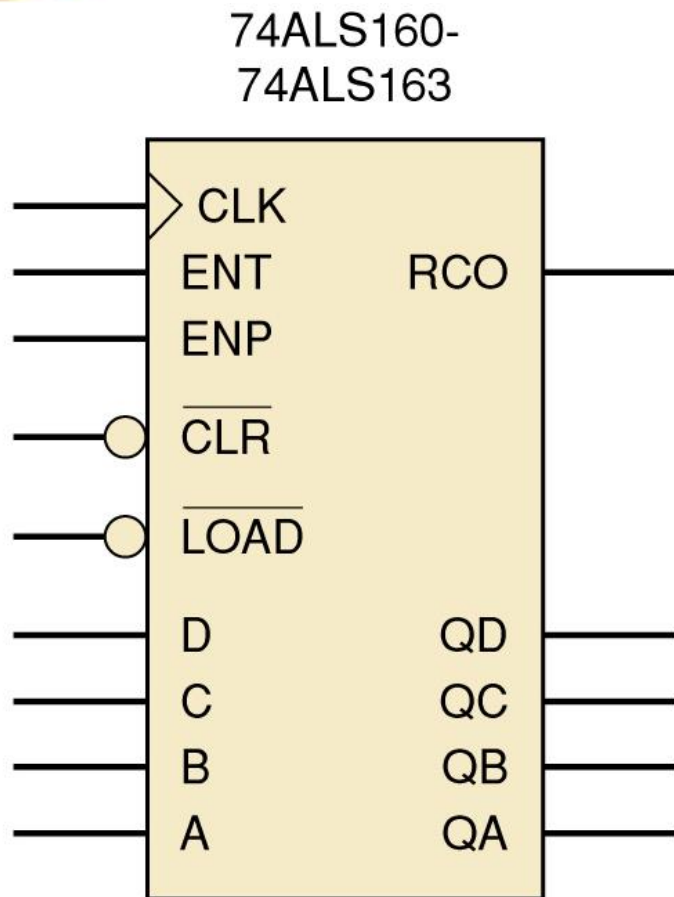
FFs are triggered by a PGT at the *CLK* input.

Active-low asynchronous CLEAR input.

Can be preset to any value by applying an active-low LOAD input to the *A*, *B*, *C*, and *D* inputs .

Part Number	Modulus
74ALS160	10
74ALS161	16
74ALS162	10
74ALS163	16

7-7 IC Asynchronous Counters



This series of IC counter chips has one more output pin—*RCO*—an active-HIGH output is to detect (*decode*) the last or terminal state of the counter.

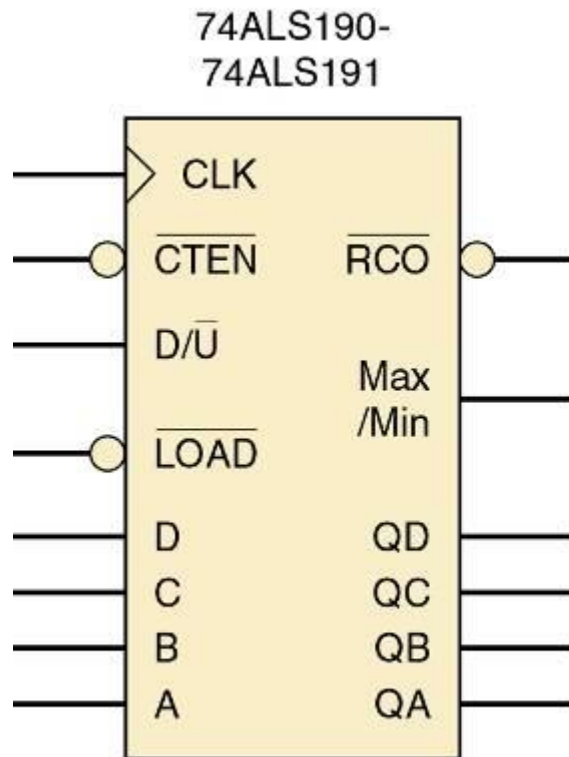
Very useful in connecting two or more counter chips in a multistage arrangement to create larger counters.

TTL 74ALS160 – 74ALS163 Function Table

$\overline{\text{CLR}}$	$\overline{\text{LOAD}}$	ENP	ENT	CLK	Function	Part Numbers
L	X	X	X	X	Asynch. Clear	74ALS160 & 74ALS161
L	X	X	X	\uparrow	Synchr. Clear	74ALS162 & 74ALS163
H	L	X	X	\uparrow	Synchr. Load	All
H	H	H	H	\uparrow	Count up	All
H	H	L	X	X	No change	All
H	H	X	L	X	No change	All

7-7 IC Asynchronous Counters

**74ALS190 and 74ALS191 series ICs
are recycling, four-bit counters.**



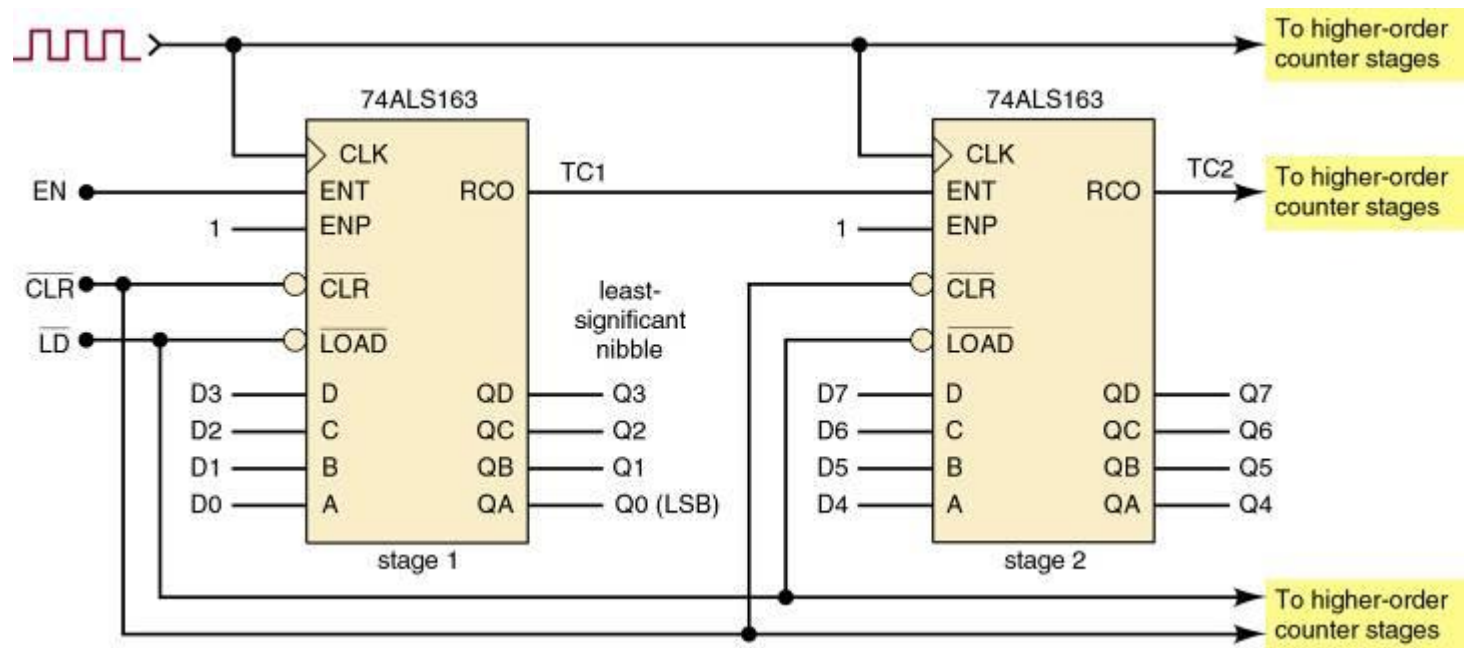
Part Number	Modulus
74ALS190	10
74ALS191	16

74ALS190-74ALS191 Function Table

LOAD	CTEN	D/U	CLK	Function
L	X	X	X	Asynch. Load
H	L	L	↑	Count up
H	L	H	↑	Count down
H	H	X	X	No change

7-7 IC Asynchronous Counters

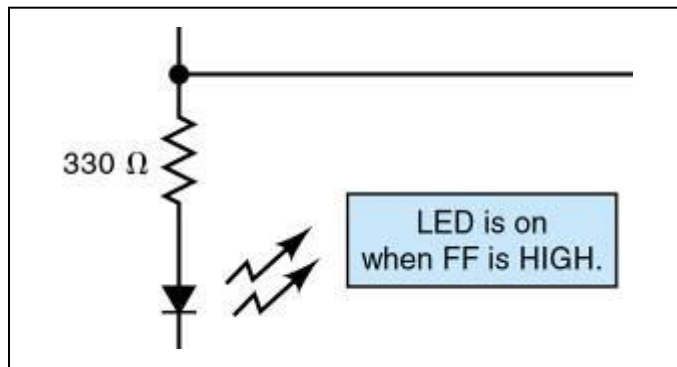
- Many standard IC counters have been designed to make it easy to connect multiple chips to create circuits with a higher counting range.
 - A **multistage** or **cascading** arrangement.



74ALS163s in a two-stage arrangement to extend the maximum counting range.

7-8 Decoding a Counter

- Digital counters are often used in applications where the count represented by FF states must somehow be determined or displayed.
 - One of the simplest means for displaying contents of a counter is connecting the FF output to an LED.

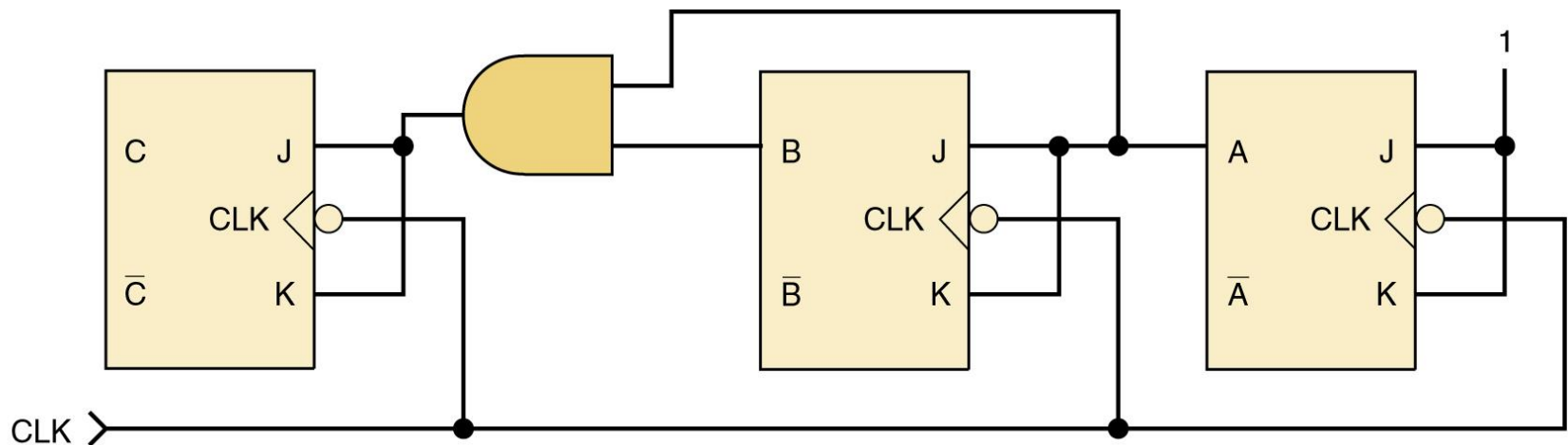


**Active-HIGH
decoder output.**

The LED method becomes inconvenient as the size (number of bits) of the counter increases. It is much harder to decode the display mentally.

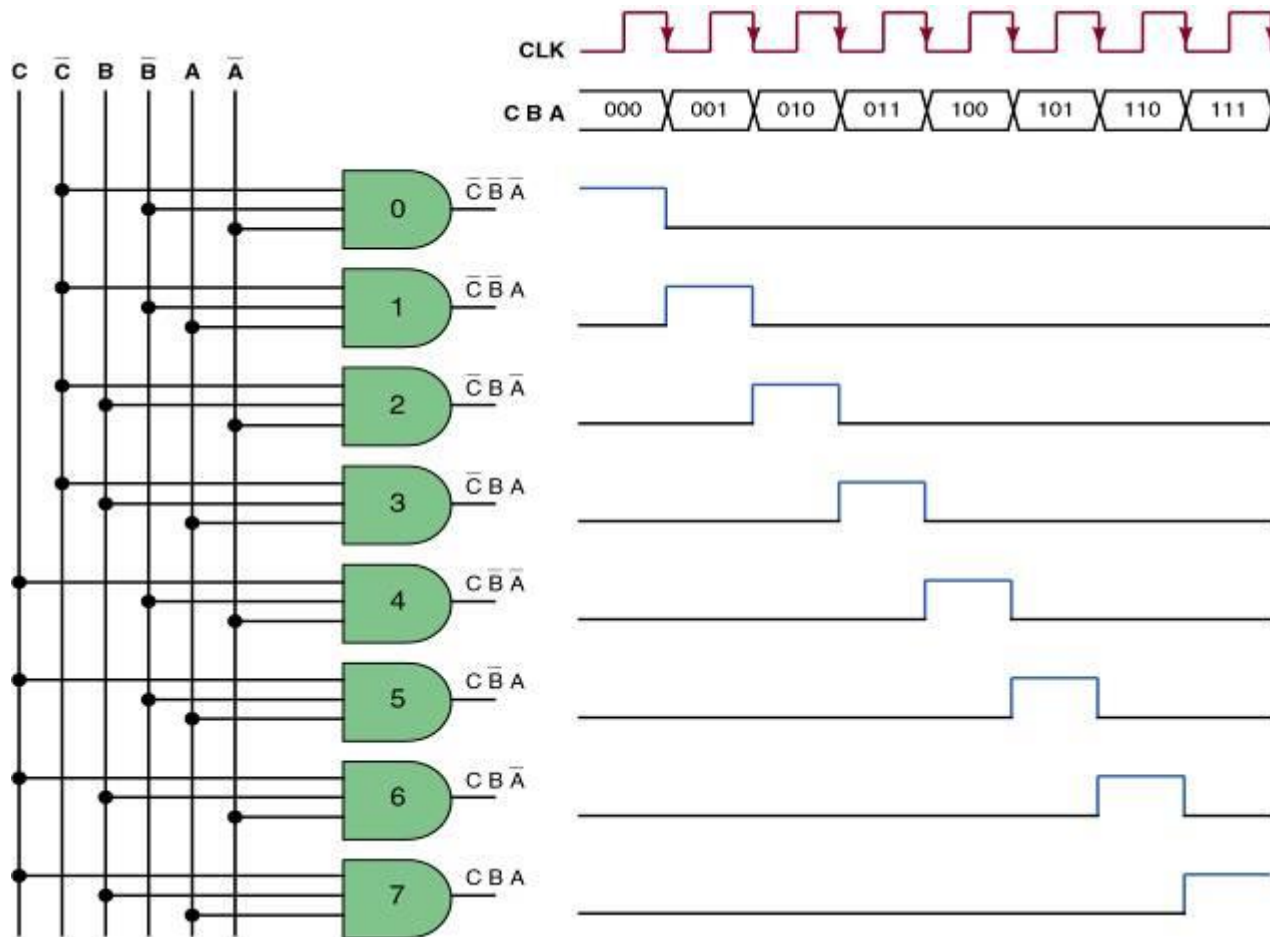
7-8 Decoding a Counter

- **Decoding** is the conversion of a binary output to a decimal value—a form immediately recognized.
 - The active-HIGH decoder shown can be used to light an LED representing each decimal number 0 to 7.



7-8 Decoding a Counter

- Decoding** is the conversion of a binary output to a decimal value—a form immediately recognized.

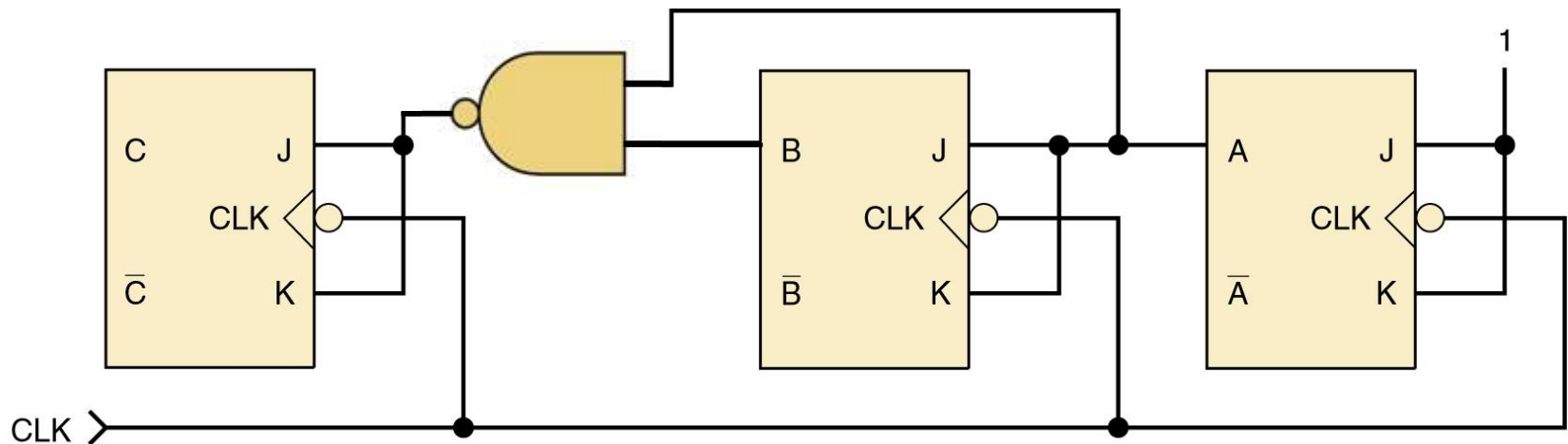


MOD-8
Active-HIGH
decoder output.

7-8 Decoding a Counter

The active-HIGH decoder shown can be changed to an active-LOW type.

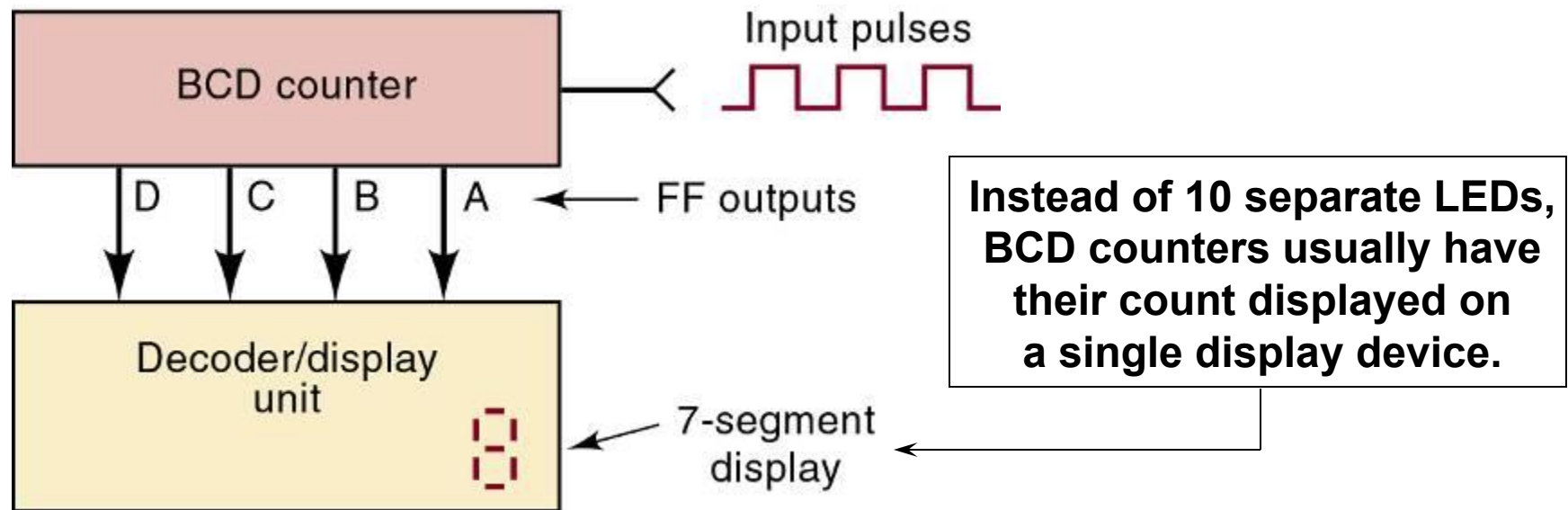
Active-LOW decoding is obtained by replacing the **AND** gates with **NAND** gates.



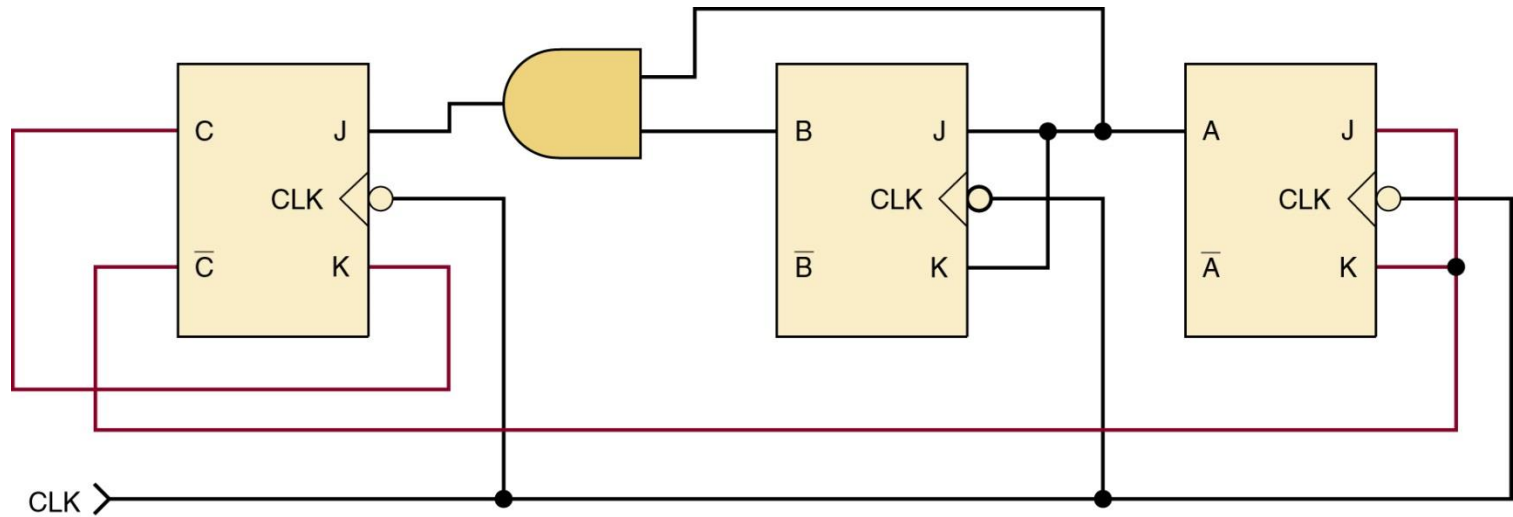
The decoder outputs now produce a normally HIGH signal, which goes LOW only when the number being decoded occurs.

7-8 Decoding a Counter

- A BCD counter has 10 states, decoded to provide 10 outputs corresponding to decimal digits 0 – 9.
 - Represented by the states of the counter FFs.
 - The 10 outputs can control 10 individual indicator LEDs for a visual display.



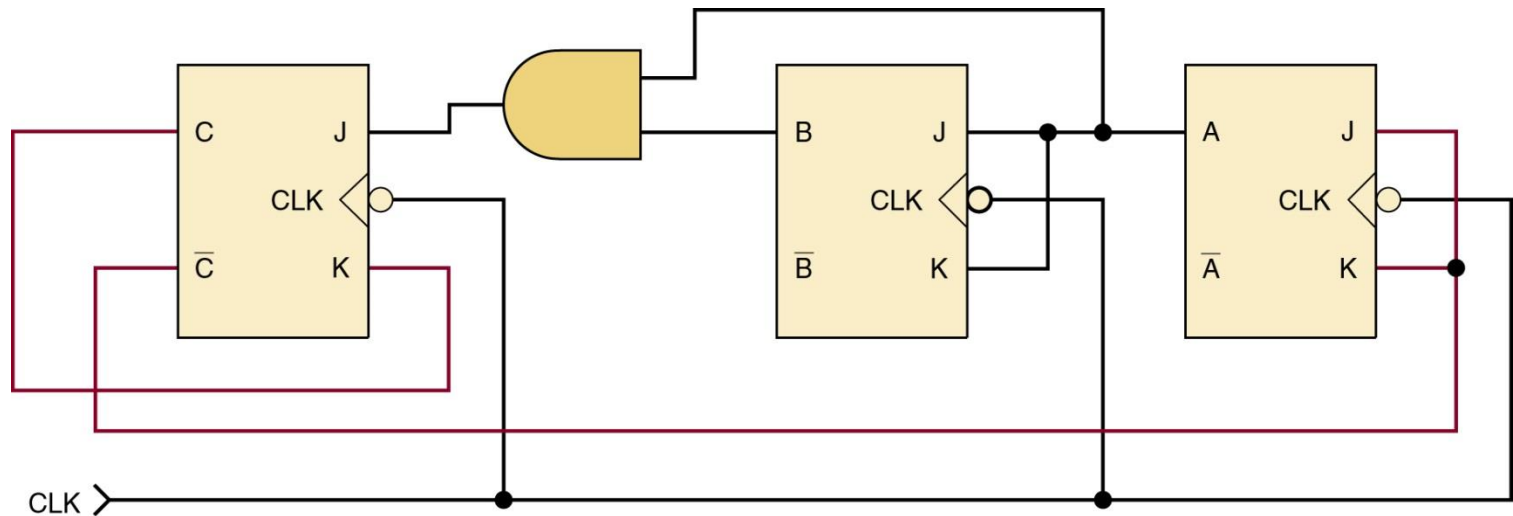
Synchronous up counter.



Control input expressions.

$$\begin{aligned} J_C &= A \cdot B \\ K_C &= C \\ J_B &= K_B = \overline{A} \\ J_A &= K_A = \overline{C} \end{aligned}$$

- Synchronous counters can be custom-designed to generate any desired count sequence.



Synchronous up-counter.

$$\begin{aligned} J_C &= A \cdot B \\ K_C &= C \\ J_B &= K_B = \overline{A} \\ J_A &= K_A = \overline{C} \end{aligned}$$

Control input expressions.

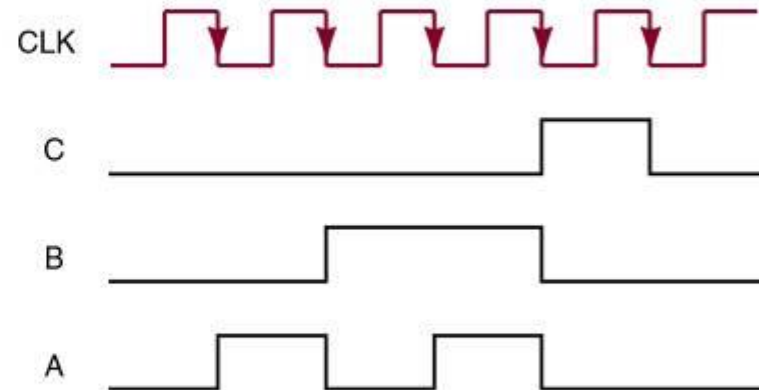
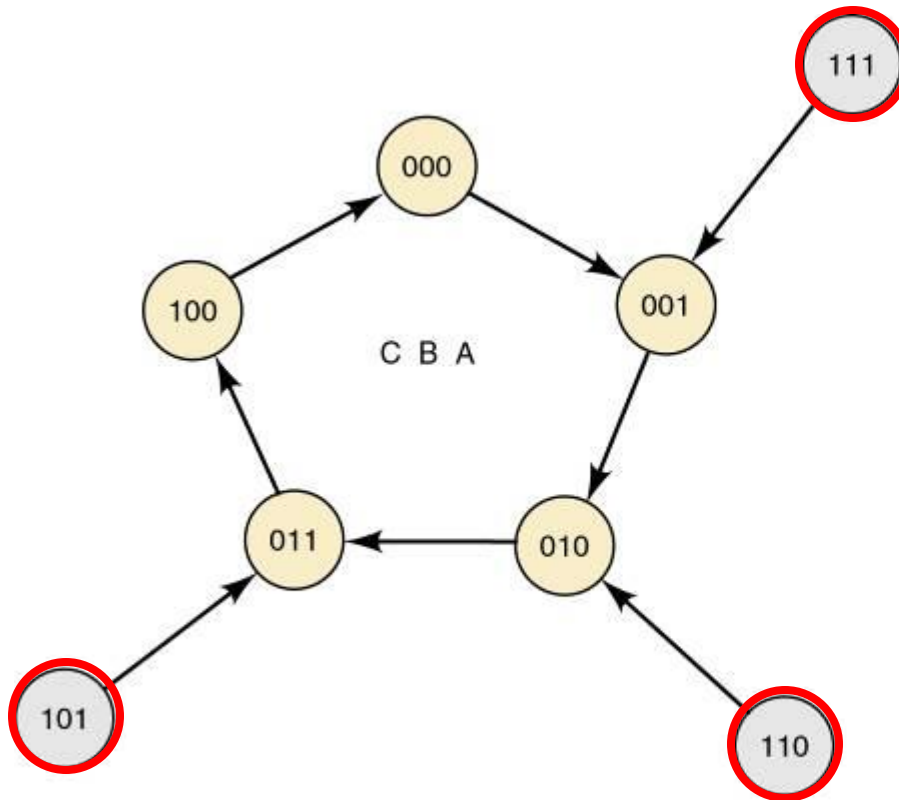
7-9 Analyzing Synchronous Counters

- Analyze counter designs of this type by predicting FF control inputs for each state of the counter.
 - A **PRESENT state/NEXT state table** is very useful for this purpose.

PRESENT State			Control Inputs						NEXT State		
C	B	A	J _C	K _C	J _B	K _B	J _A	K _A	C	B	A
0	0	0	0	0	0	0	1	1	0	0	1
0	0	1	0	0	1	1	1	1	0	1	0
0	1	0	0	0	0	0	1	1	0	1	1
0	1	1	1	0	1	1	1	1	1	0	0
1	0	0	0	1	0	0	0	0	0	0	0
1	0	1	0	1	1	1	0	0	0	1	1
1	1	0	0	1	0	0	0	0	0	1	0
1	1	1	1	1	1	1	0	0	0	0	1

7-9 Analyzing Synchronous Counters

Synchronous up counter state transition diagram and waveform.

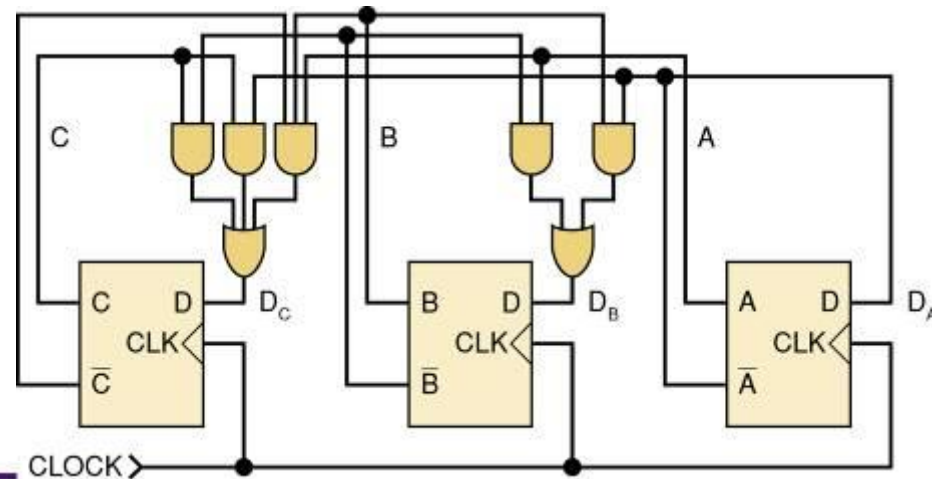


Highlighted information indicates this counter design is self-correcting.

7-9 Analyzing Synchronous Counters

Synchronous counter built using D-type FFs.

Control circuitry will typically be more complex than an equivalent JK-type counter,



PRESENT State			Control Inputs			NEXT State		
C	B	A	D _C	D _B	D _A	C	B	A
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	0
0	1	0	0	1	1	0	1	1
0	1	1	1	0	0	1	0	0
1	0	0	1	0	1	1	0	1
1	0	1	1	1	0	1	1	0
1	1	0	1	1	1	1	1	1
1	1	1	0	0	0	0	0	0

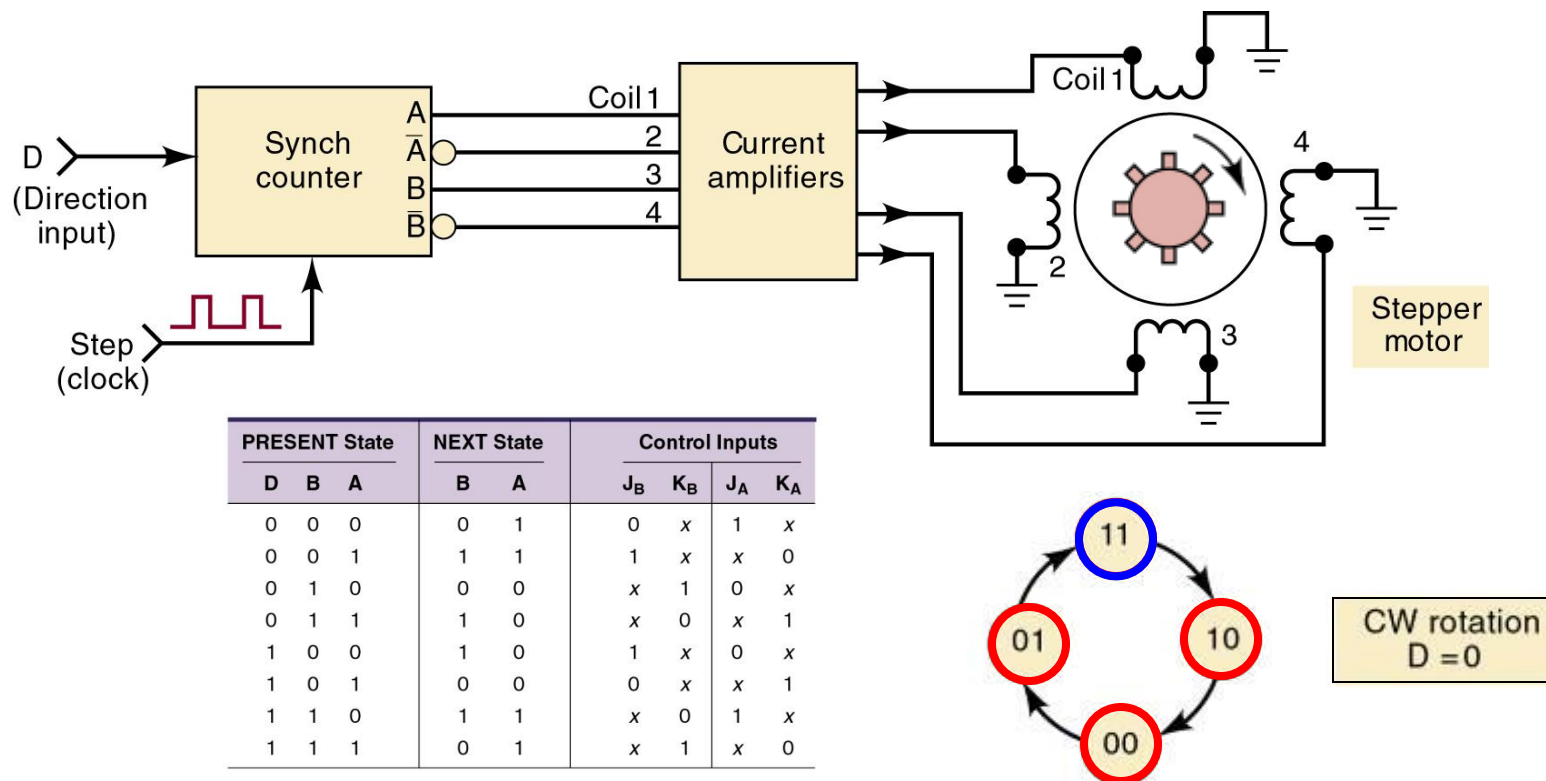
The number of synchronous inputs to control is reduced by half

7-10 Synchronous Counter Design

- Common design method using J-K flip-flops in a synchronous counter configuration.
 - Determine the desired number of bits (FFs) and the desired counting sequence.
 - Draw the state transition diagram showing *all* possible states—including those not part of the desired counting sequence.
 - Use the state transition diagram to set up a table that lists *all* PRESENT states and their NEXT states.
 - Add a column for each *J* and *K* input & indicate levels required to produce transition to the NEXT state.
 - Design the logic circuits to generate levels required at each input, and implement the final expressions.

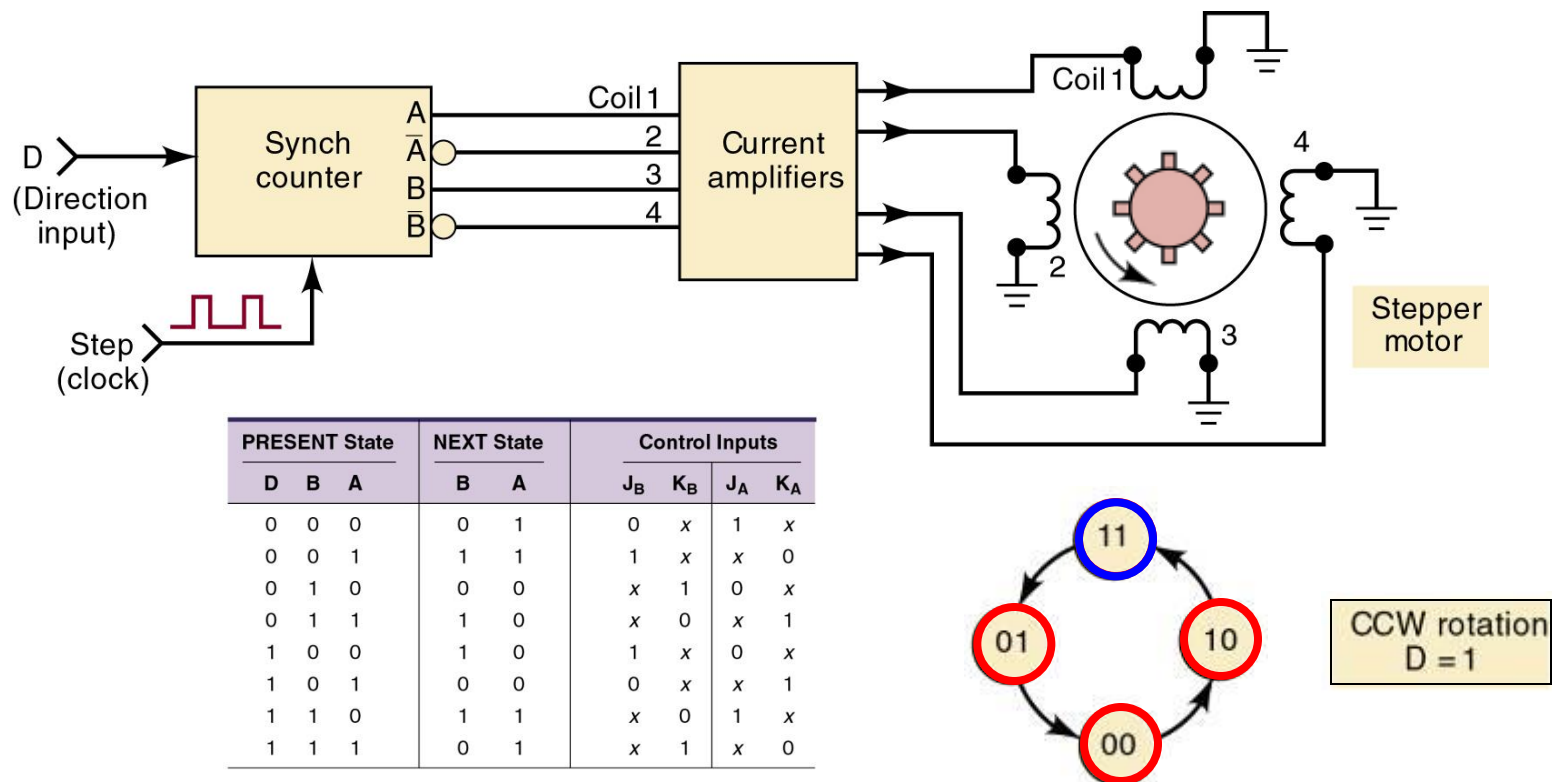
7-10 Synchronous Counter Design

- A stepper motor rotates in steps, not continuous motion—typically 15 degrees per step.
 - A practical application of synchronous counter design.
 - Reversible, depending on output level.



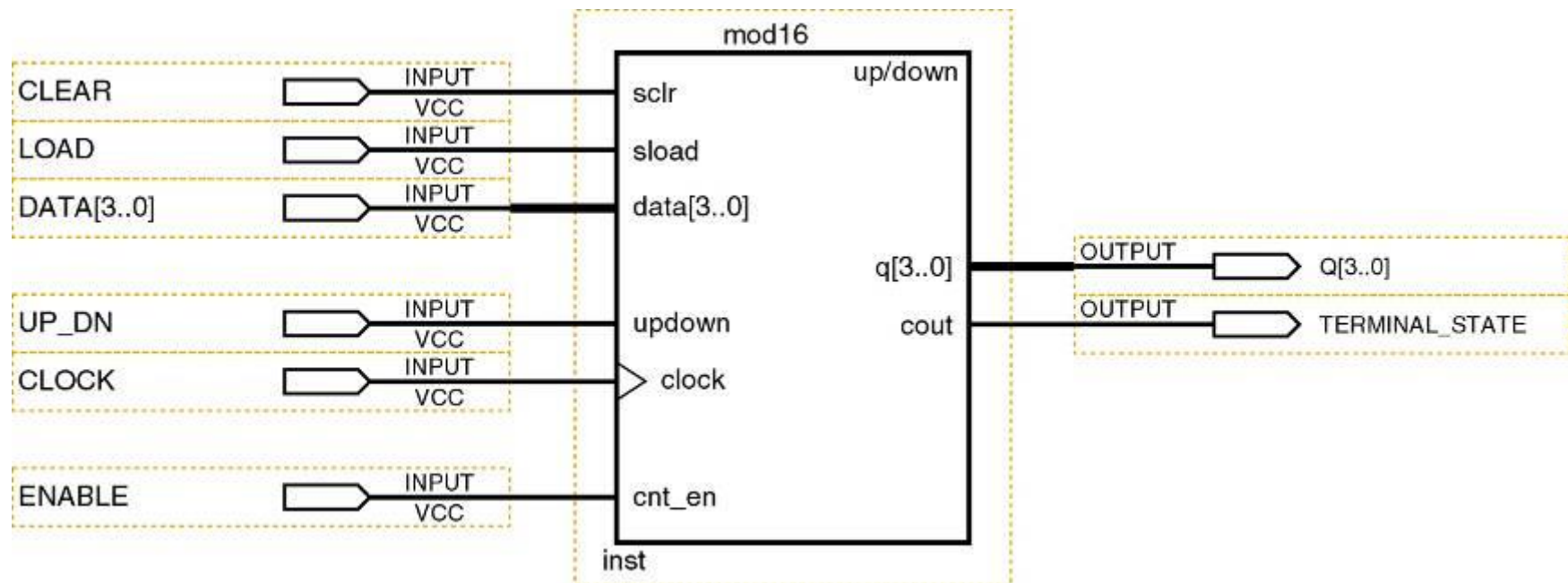
7-10 Synchronous Counter Design

- A stepper motor rotates in steps, not continuous motion—typically 15 degrees per step.
 - A practical application of synchronous counter design.
 - Reversible, depending on output level.



7-11 Altera Library Functions for Counters

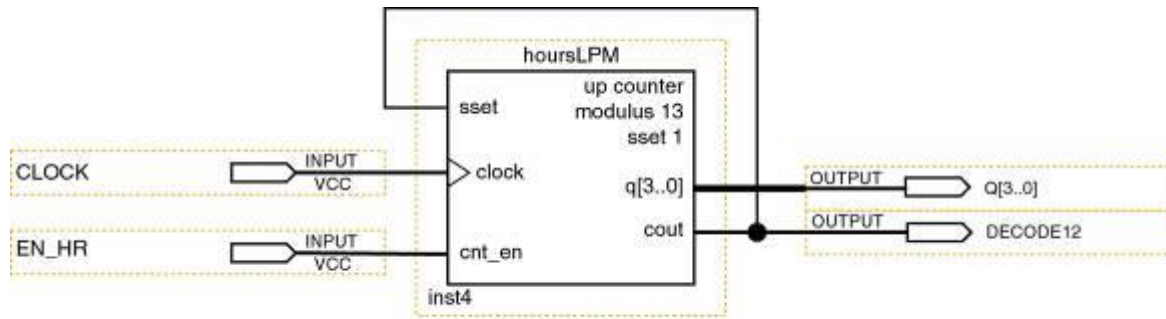
- The Quartus Block Editor can program a PLD with any counter using flip-flops and gates.
 - These macro-functions are in the *maxplus2* library.



A full-featured MOD-16 up/down counter.

7-11 Altera Library Functions for Counters

Digital clock hours counter—block diagram, MegaWizard settings & simulation results.



How wide should the 'q' output bus be? bits

What should the counter direction be?

- ☒ Up only
- ☐ Down only
- ☐ Create an 'updown' input port to allow me to do both (1 counts up; 0 counts down)

Which type of counter do you want?

- ☐ Plain binary
- ☒ Modulus, with a count modulus of

Do you want any optional additional ports?

- ☐ Clock Enable
- ☒ Count Enable
- ☐ Carry-in
- ☒ Carry-out

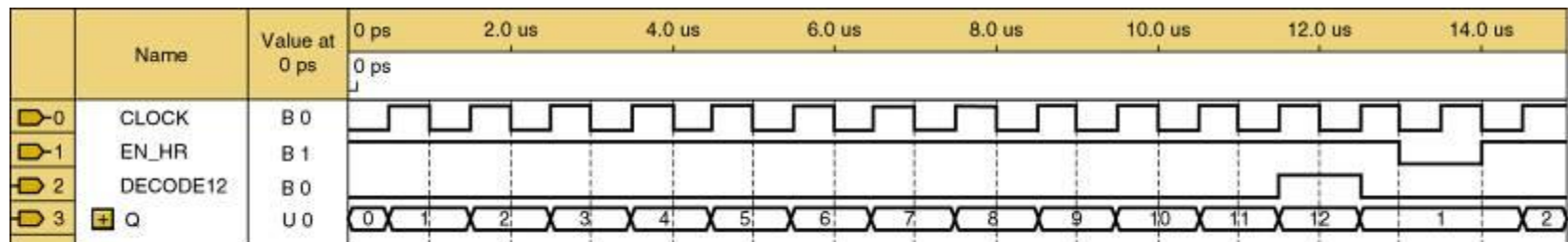
Do you want any optional inputs?

Synchronous inputs

- ☐ Clear
- ☐ Load
- ☒ Set
 - ☐ Set to all 1's
 - ☒ Set to

Asynchronous inputs

- ☐ Clear
- ☐ Load
- ☐ Set
 - ☒ Set to all 1's
 - ☐ Set to 0

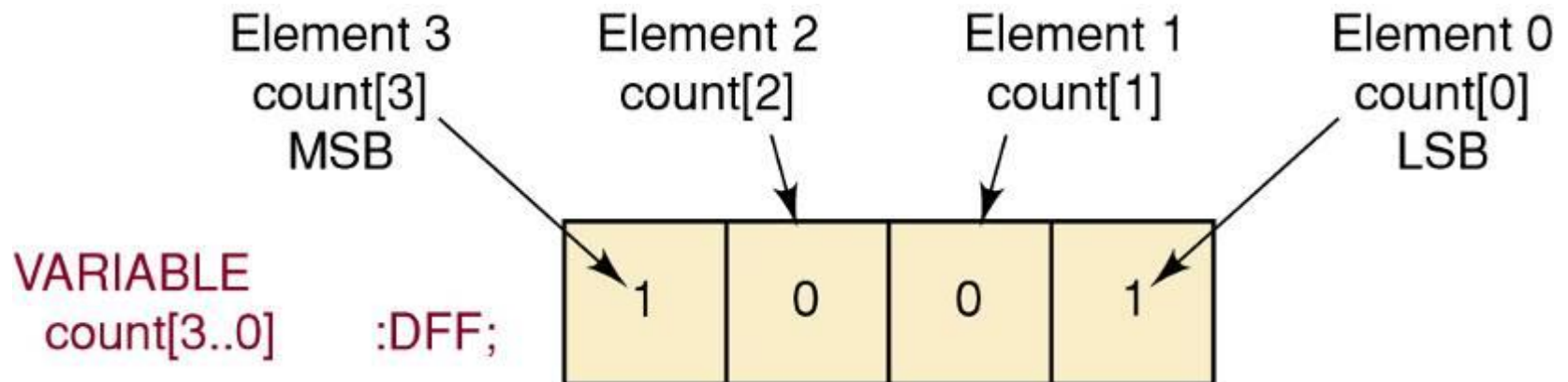


7-12 HDL Counters

- Methods to describe counter circuits using HDL primarily use synchronous techniques.
 - All flip-flops update simultaneously in response to the same clock event.
 - All bits in a count sequence go from their PRESENT state to their prescribed NEXT state simultaneously,
- State Transition Description Methods
 - The PRESENTstate/NEXT state table is the equivalent of the truth table.

7-12 HDL Counters

- State descriptions in AHDL
 - The first important step is to declare the counter output pins properly— as a bit array.



7-12 HDL Counters

- State descriptions in AHDL

```
1  SUBDESIGN fig7_43
2  (
3      clock      :INPUT;
4      q[2..0]    :OUTPUT;  -- declare 3-bit array of output bits
5  )
6  VARIABLE
7      count[2..0] :DFF; -- declare a register of D flip flops.
8
9  BEGIN
10     count[].clk = clock; -- connect all clocks to synchronous source
11     IF count[].q < 4 THEN -- note; count[] is the same as count[].q
12         count[].d = count[].q + 1; -- increment current value by one
13     ELSE count[].d = 0;           -- recycle to zero: force unused states to 0
14     END IF;
15     q[] = count[].q;             -- transfer register contents to outputs
16 END;
```

The current state of the counter is evaluated (*count[].q*) on line 11, and if it is less than the highest desired count value, it uses the description *count[].d count.q + 1* (line 12).

7-12 HDL Counters

- State descriptions in AHDL

```
1  SUBDESIGN fig7_43
2  (
3      clock      :INPUT;
4      q[2..0]    :OUTPUT;  -- declare 3-bit array of output bits
5  )
6  VARIABLE
7      count[2..0] :DFF; -- declare a register of D flip flops.
8
9  BEGIN
10     count[].clk = clock; -- connect all clocks to synchronous source
11     IF count[].q < 4 THEN -- note; count[] is the same as count[].q
12         count[].d = count[].q + 1; -- increment current value by one
13     ELSE count[].d = 0;           -- recycle to zero: force unused states to 0
14     END IF;
15     q[] = count[].q;             -- transfer register contents to outputs
16 END;
```

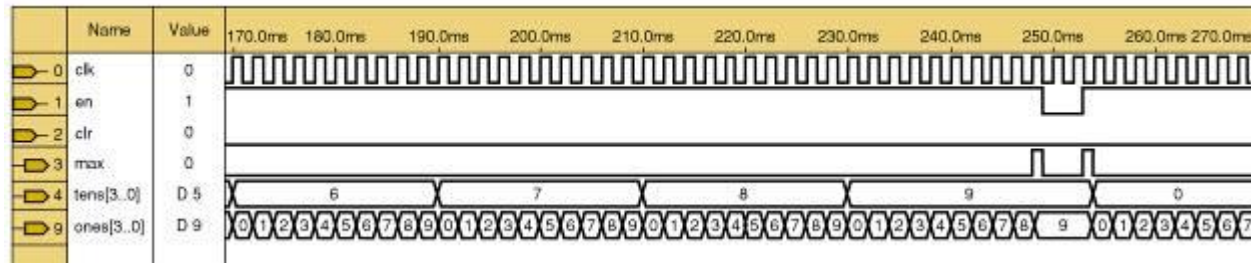
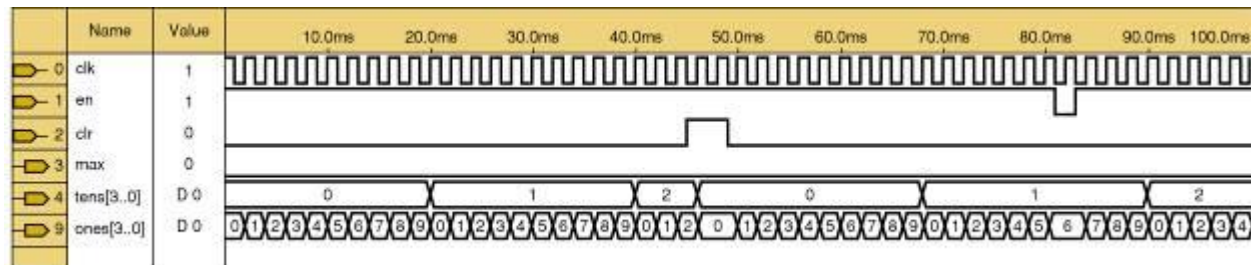
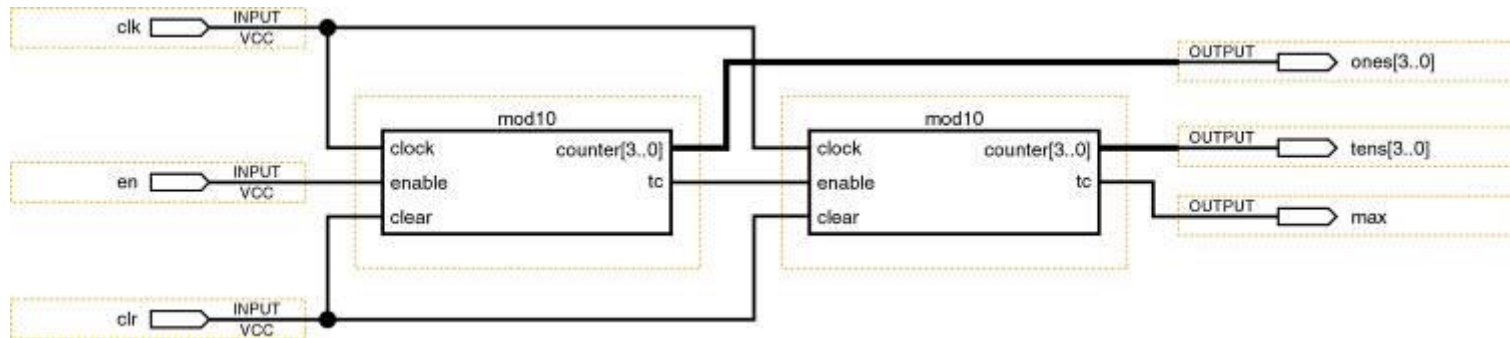
When the counter has reached the highest desired state (or higher), the *IF* statement test will be false, resulting in a NEXT-state input value of zero, which recycles the counter.

7-13 Wiring HDL Modules Together

- Designing large digital systems is much easier if the system is subdivided into smaller, manageable modules that are interconnected.
 - The essence of the concept of **hierarchical design**.
- Consider a recycling, MOD-100 BCD counter with a synchronous clear.
 - Creating a MOD-10 BCD counter module, cascading synchronously two of these in a higher-level design file is the easiest way to do this.

7-13 Wiring HDL Modules Together

Block diagram design and simulation results for MOD-100 BCD counter design.



7-13 Wiring HDL Modules Together

- Decoding the AHDL MOD-5 counter

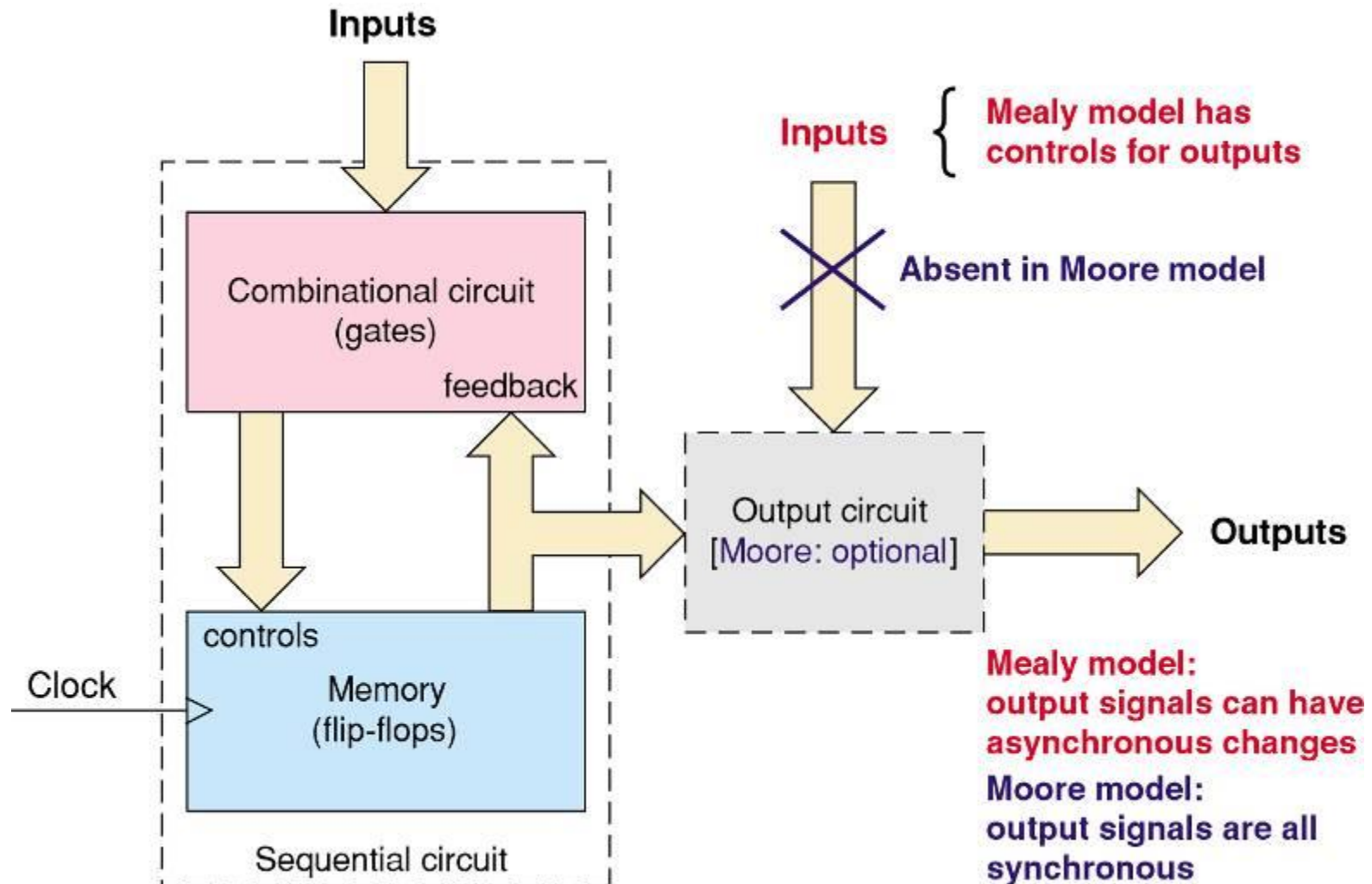
```
1  SUBDESIGN fig7_40
2  (
3      clock      :INPUT;
4      q[2..0]    :OUTPUT;
5  )
6  VARIABLE
7      count[2..0] :DFF;      --create a 3-bit register
8  BEGIN
9      count[].clk = clock;    --connect all clocks in parallel
10
11      CASE count[] IS
12  --          Present      Next
13  -----
14          WHEN 0    =>    count[].d = 1;
15          WHEN 1    =>    count[].d = 2;
16          WHEN 2    =>    count[].d = 3;
17          WHEN 3    =>    count[].d = 4;
18          WHEN 4    =>    count[].d = 0;
19          WHEN OTHERS =>    count[].d = 0;
20      END CASE;
21      q[] = count[];          -- assign register to output pins
22  END;
```


7-14 State Machines

- The term **state machine** refers to a circuit that sequences through a set of predetermined states.
 - Controlled by a clock and other input signals.
- The term *counter* is used for sequential circuits that have a regular numeric count sequence.
 - The things that are counted are actually clock pulses.
 - But the pulses may represent many kinds of events.
- The general distinction between the two terms:
 - A *counter* is commonly used to *count* events,
 - A *state machine* is commonly used to *control* events.

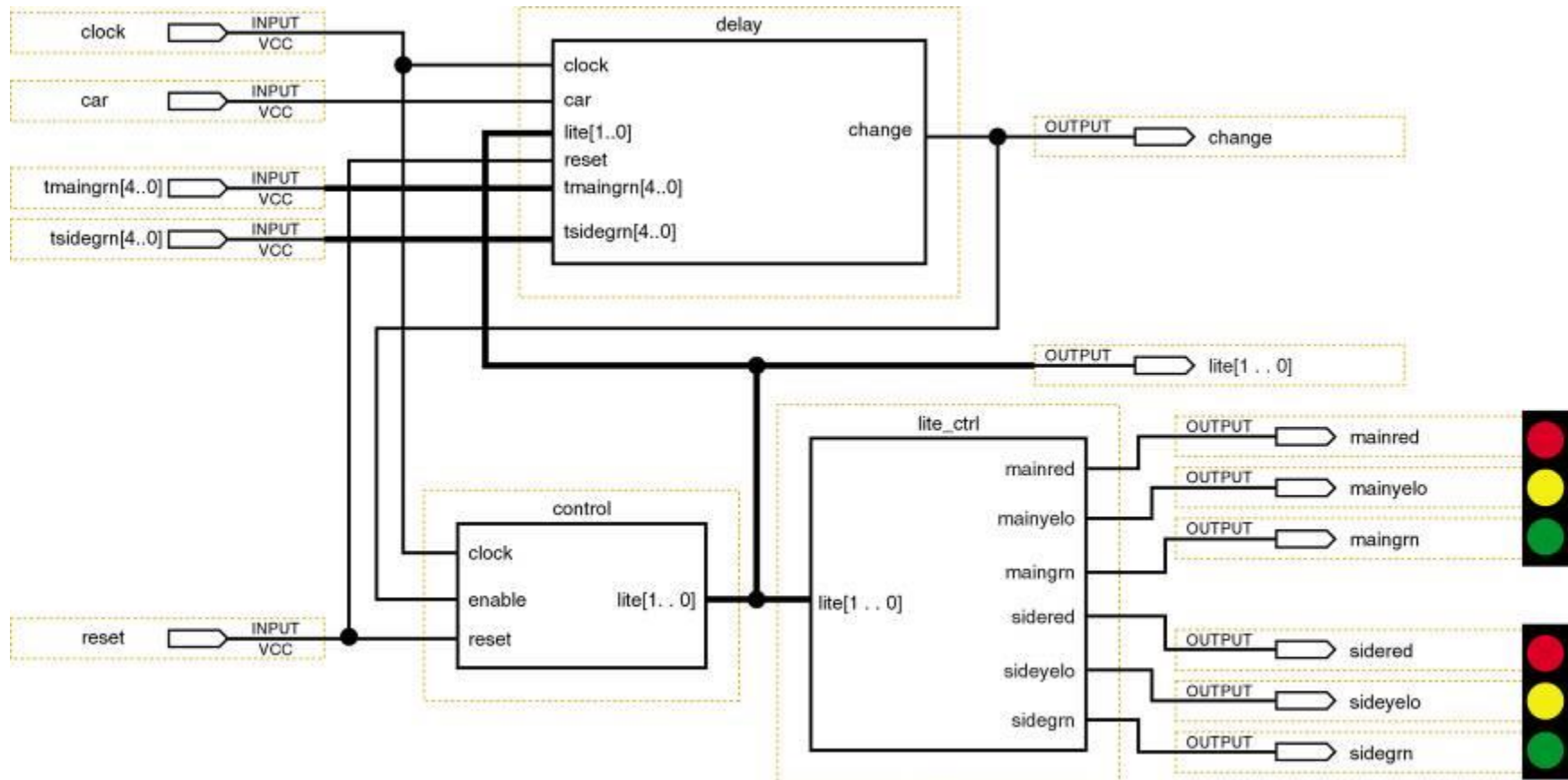
7-14 State Machines

Block diagram for counters and state machines.



7-14 State Machines

Traffic light controller state machine.

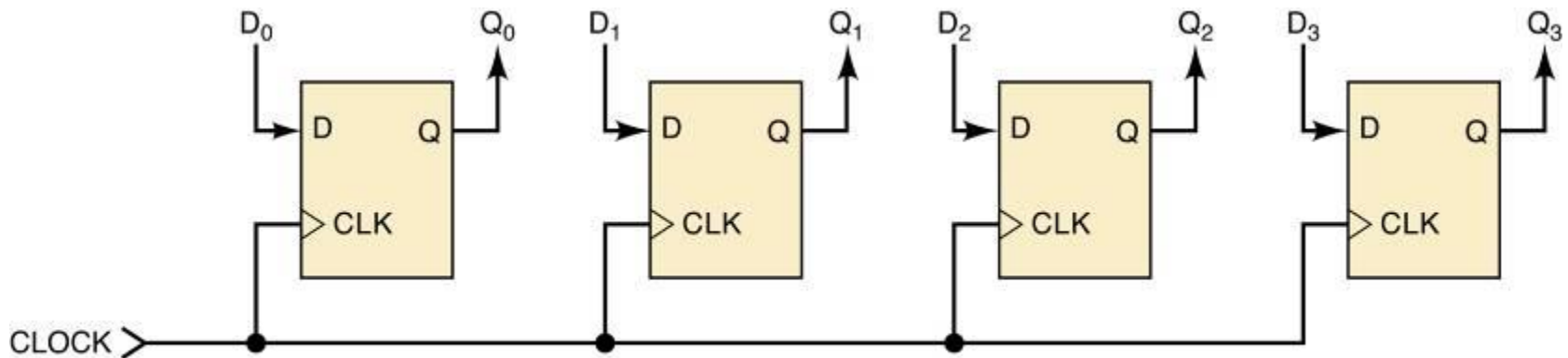


7-15 Register Data Transfer

- The various types of registers can be classified according to...
 - The manner in which data can be entered into the register for storage.
 - The manner in which data are outputted from the register.
- Serial data flow through a register is generally called *shifting*—either to the left or to the right.
 - Serial output data fed back to the input of the same register is called a *data rotate*.
- Parallel inputting of data is often described as a *register load*.

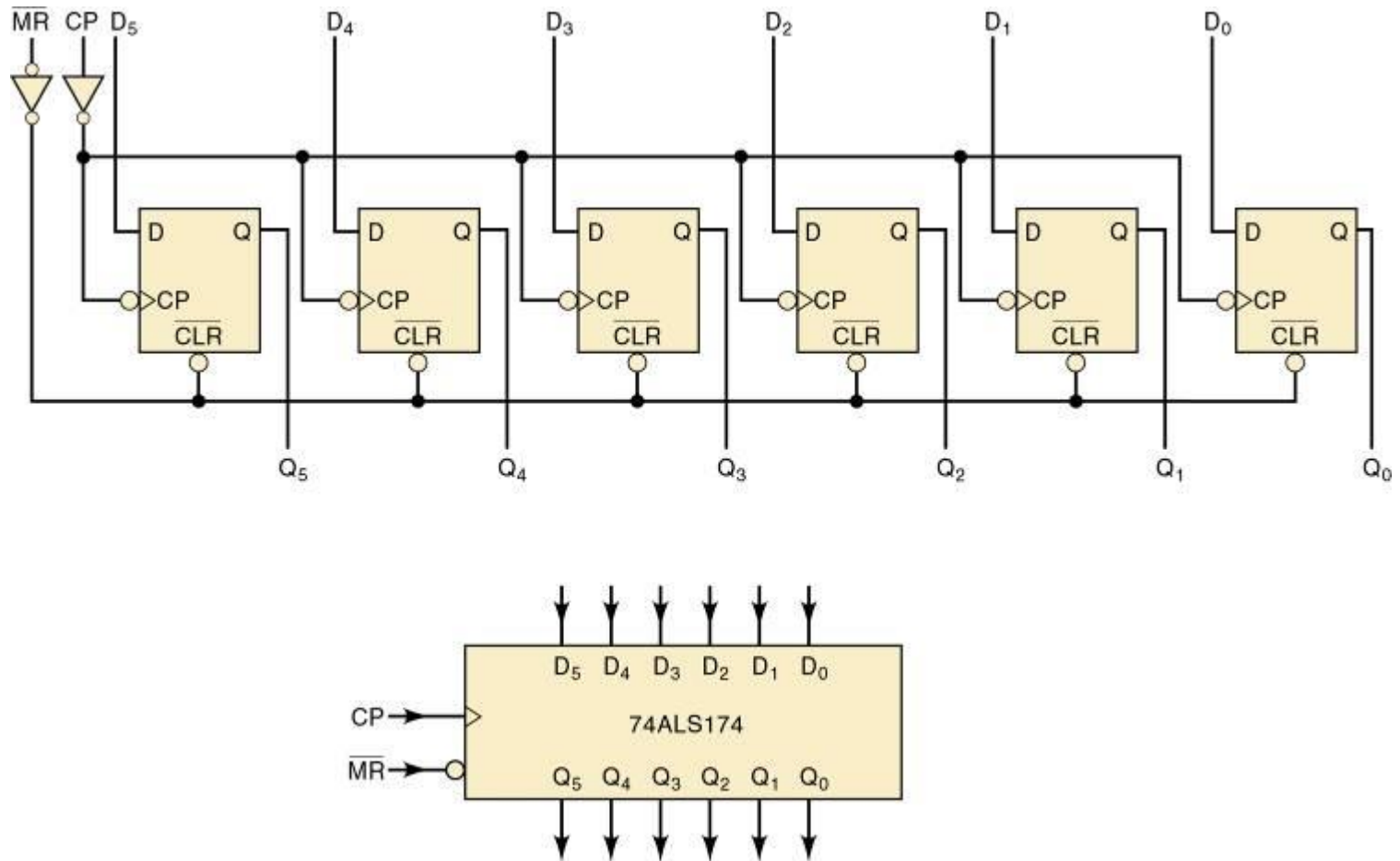
74ALS174/74HC174

Parallel in/parallel out (PIPO)

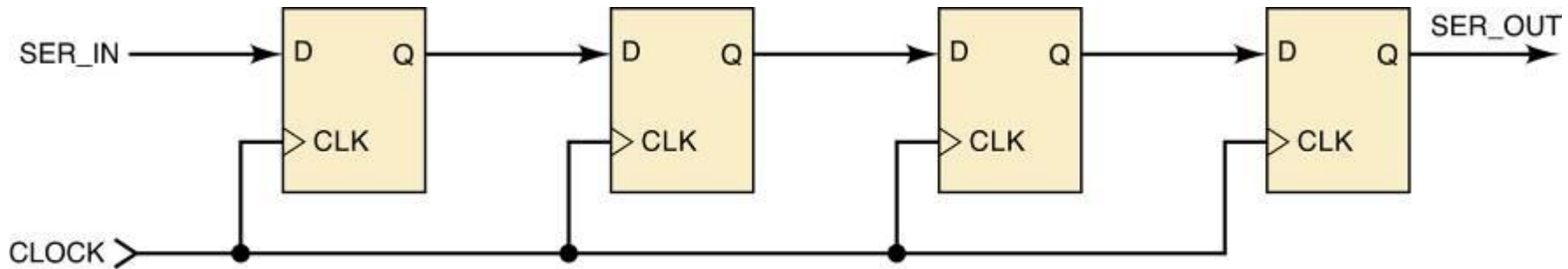


A group of flip-flops that can store multiple bits simultaneously and in which all bits of the stored binary value are directly available.

74ALS174/74HC174

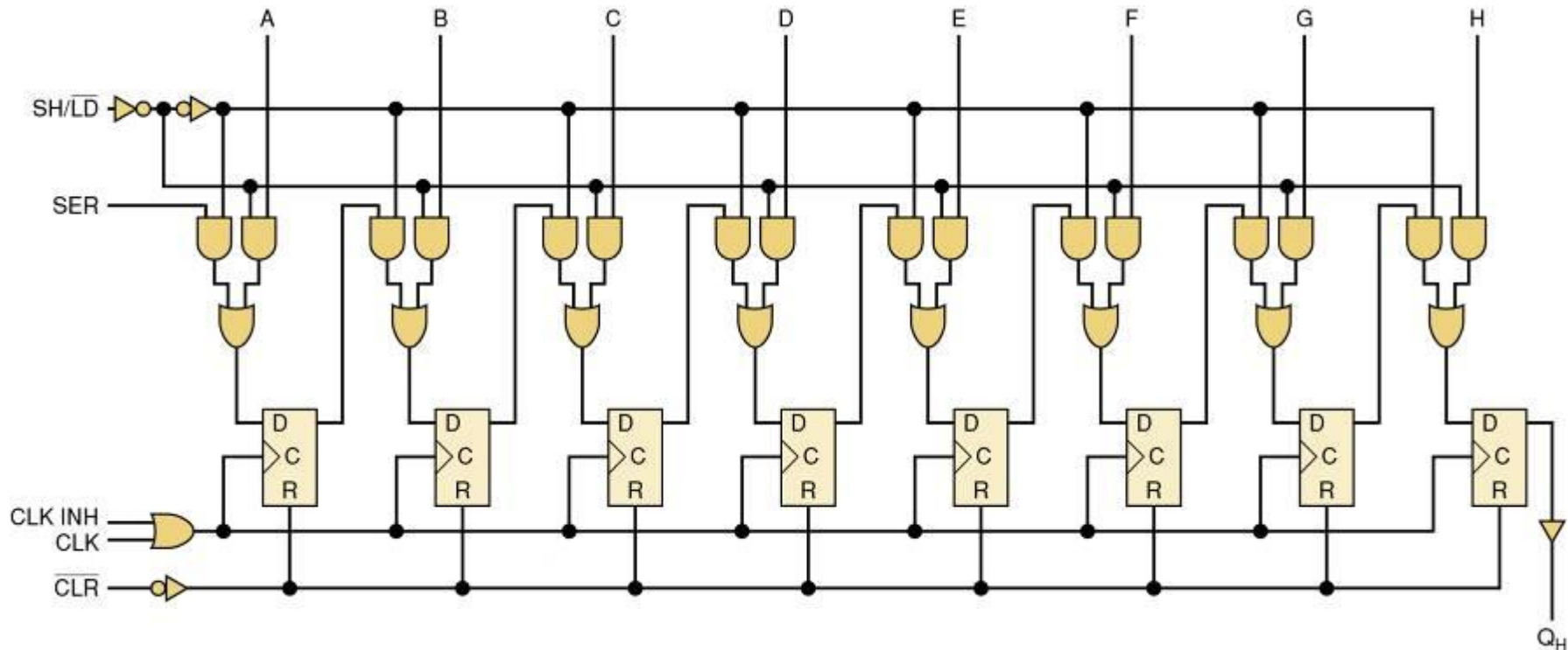


74ALS166/74HC166 Serial in/serial out (SISO)



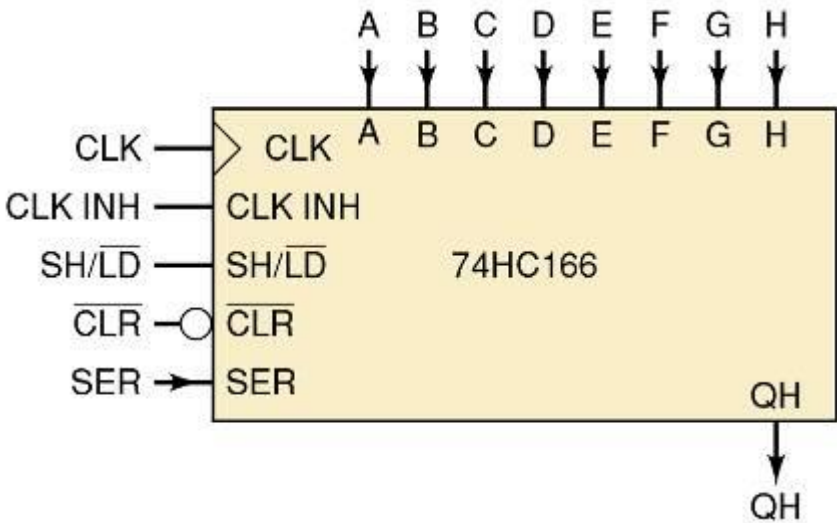
Data loaded one bit at a time moves one bit at a time, with each clock pulse through the flip-flops toward the other end of the register, and exit one bit at a time in the same order as originally loaded.

74ALS166/74HC166



Circuit Diagram

74ALS166/74HC166



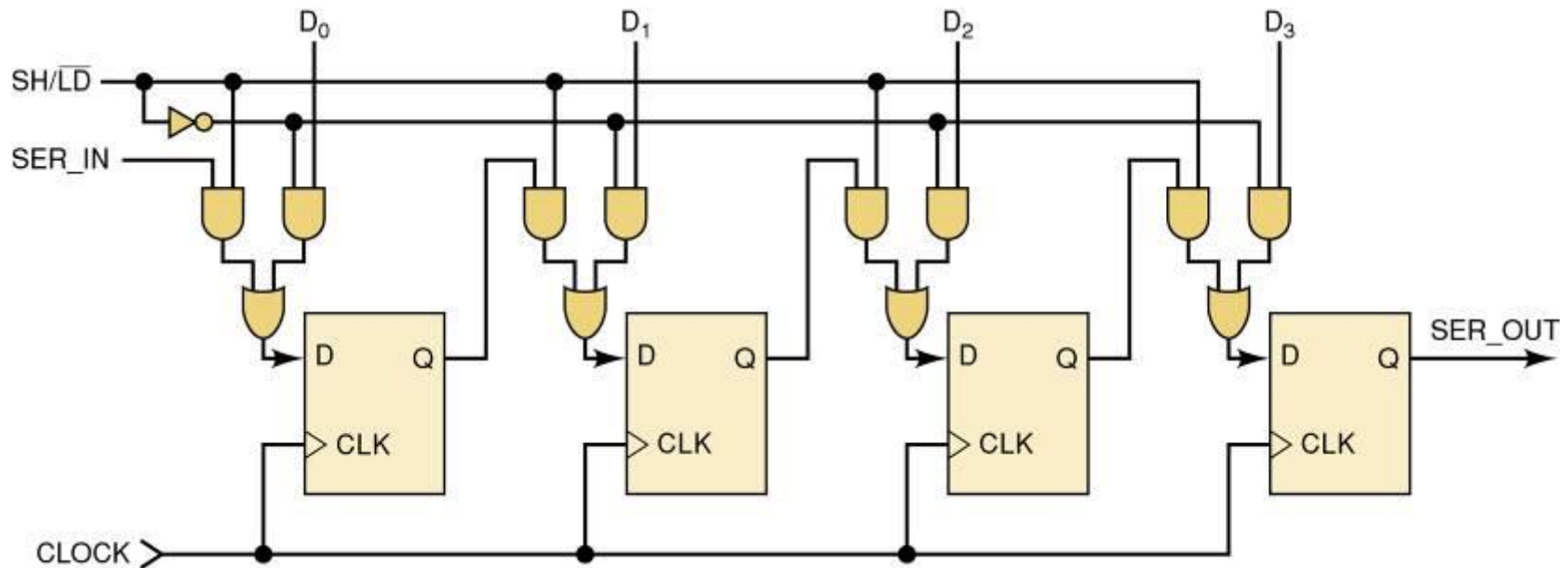
Logic
Symbol

Function
Table

INPUTS						OUTPUTS		
						INTERNAL		QH
CLR	SH/LD	CLK INH	CLK	SER	PARALLEL A . . . H	QA	QB	
L	X	X	X	X	X	L	L	L
H	X	L	L	X	X	QA0	QB0	QH0
H	L	L	↑	X	a . . . h	a	b	h
H	H	L	↑	H	X	H	QAn	QGn
H	H	L	↑	L	X	L	QAn	QGn
H	X	H	↑	X	X	QA0	QB0	QH0

74ALS165/74HC165

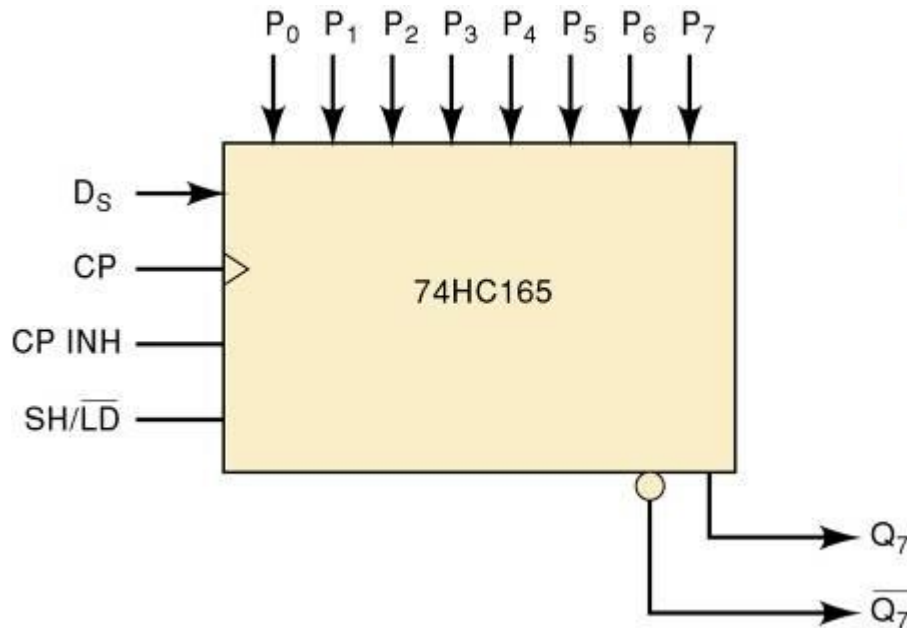
8-bit parallel in/serial out (PISO)



Serial shifting is always synchronous, as the clock is required to ensure the input data moves only one bit at a time with each appropriate clocking edge.

74ALS165/74HC165

8-bit parallel in/serial out (PISO)



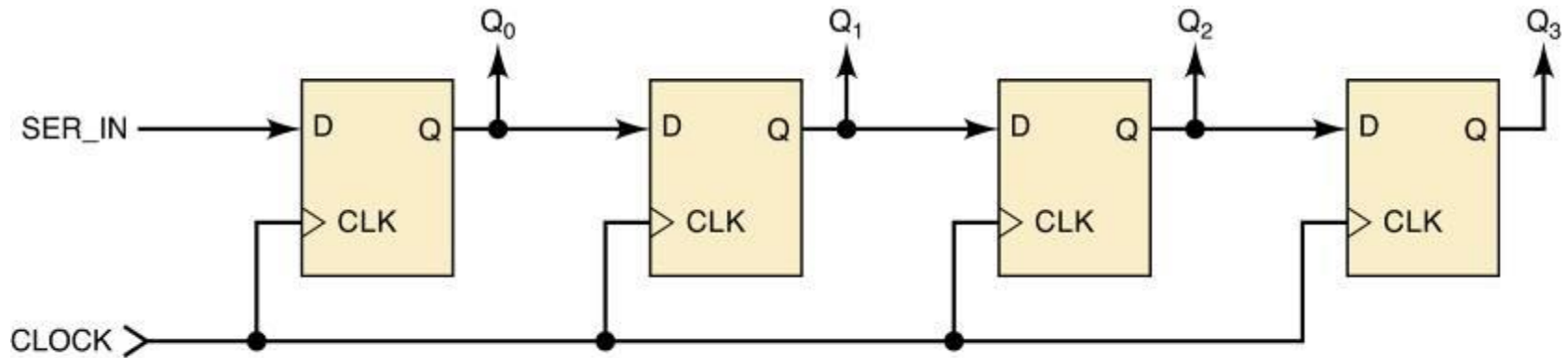
Function Table

Inputs			Operation
SH/LD	CP	CP INH	
L	X	X	Parallel load
H	H	X	No change
H	X	H	No change
H	⌄	L	Shifting
H	L	⌄	Shifting

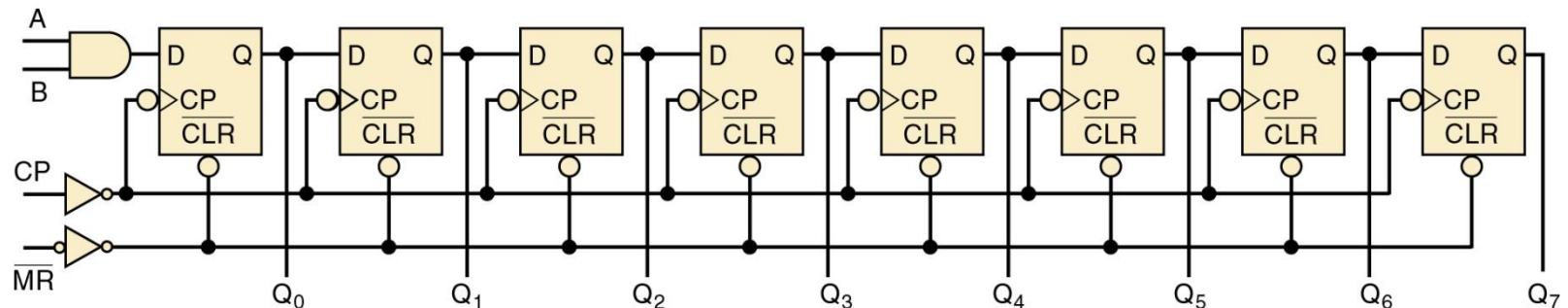
H = high level
L = low level
X = immaterial
⌄ = PGT

74ALS164/74HC164

8-bit serial in/parallel out (SIPO)

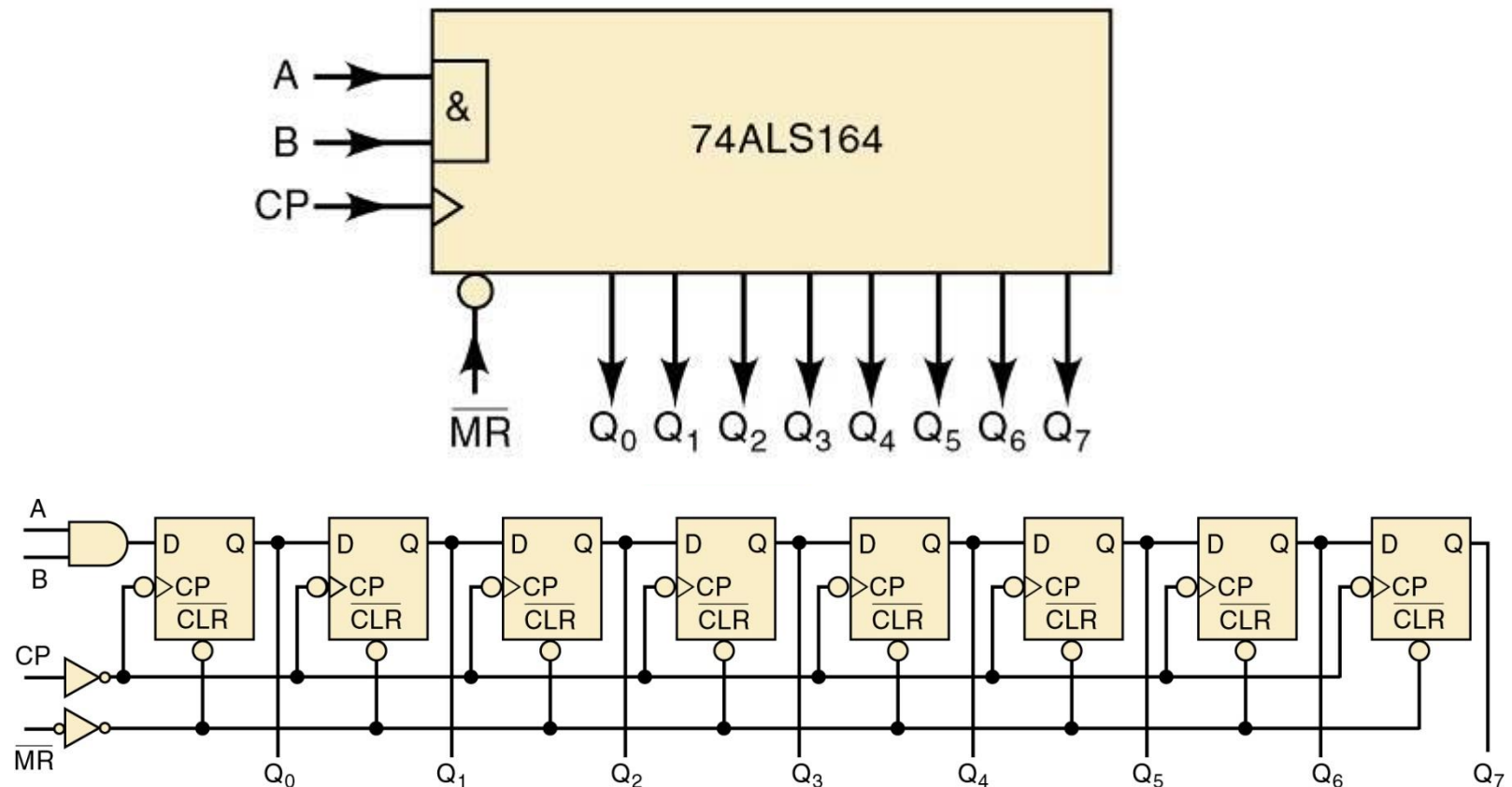


8-bit
shift register
74ALS164



74ALS164/74HC164

8-bit serial in/parallel out (SIPO)

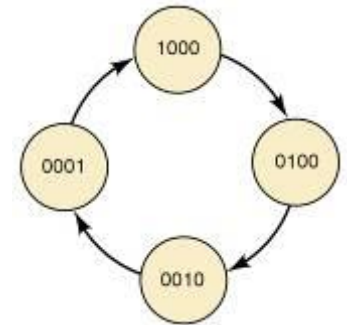
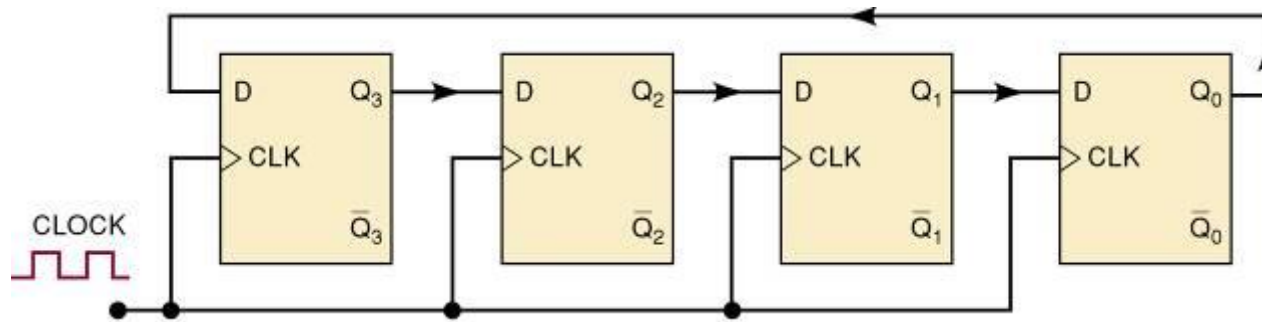


7-17 Shift Register Counters

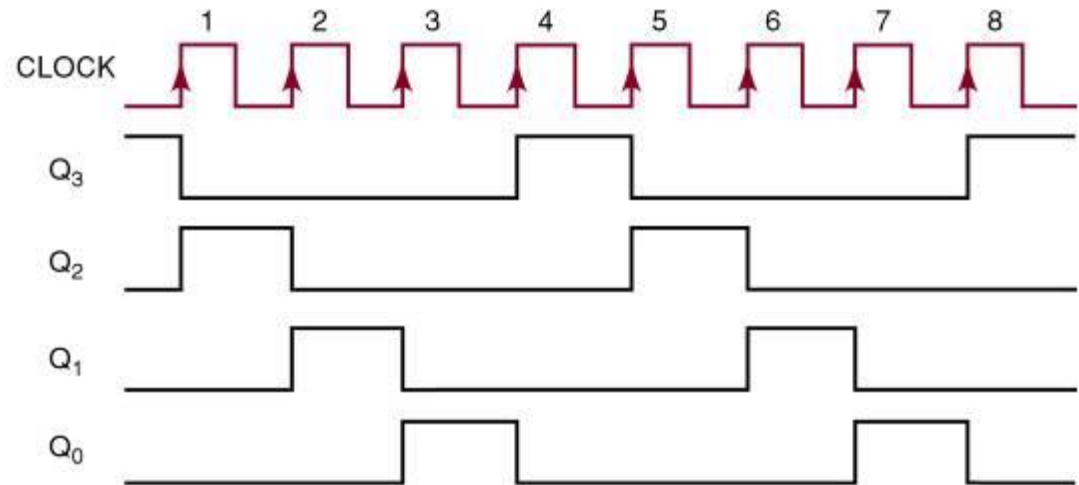
- Shift-register counters use *feedback*—the output of the last FF in the register is connected back to the first FF in some way.

7-17 Shift Register Counters

- A **ring counter** is a **circulating shift register** connected so the last FF shifts its value to the first.



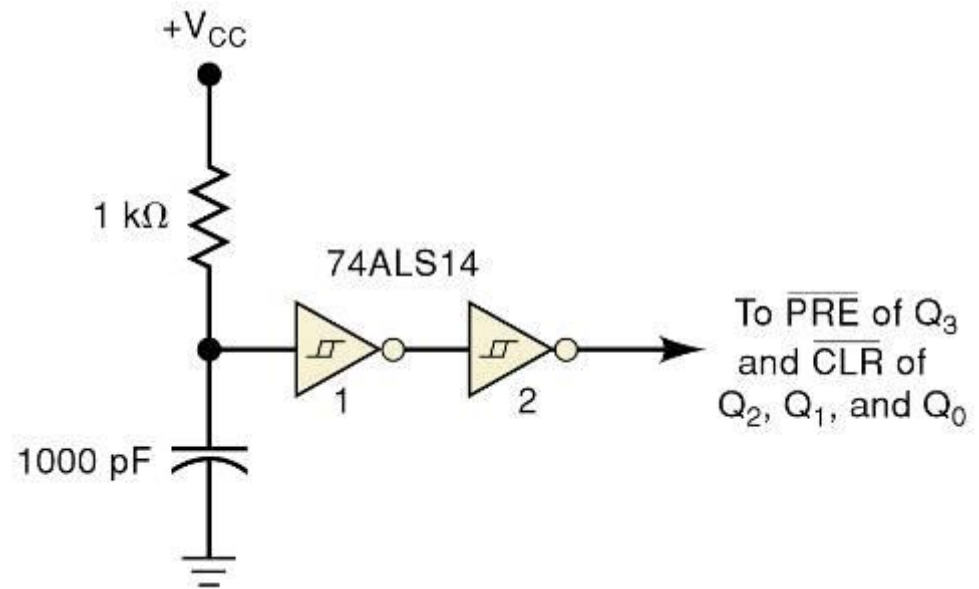
Q ₃	Q ₂	Q ₁	Q ₀	CLOCK pulse
1	0	0	0	0
0	1	0	0	1
0	0	1	0	2
0	0	0	1	3
1	0	0	0	4
0	1	0	0	5
0	0	1	0	6
0	0	0	1	7
⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮



7-17 Shift Register Counters

- To operate properly, a ring counter must start off with only one FF in the 1 state and all the others in the 0 state.
 - As power-up starting states will be unpredictable, the counter is preset *before* clock pulses are applied.

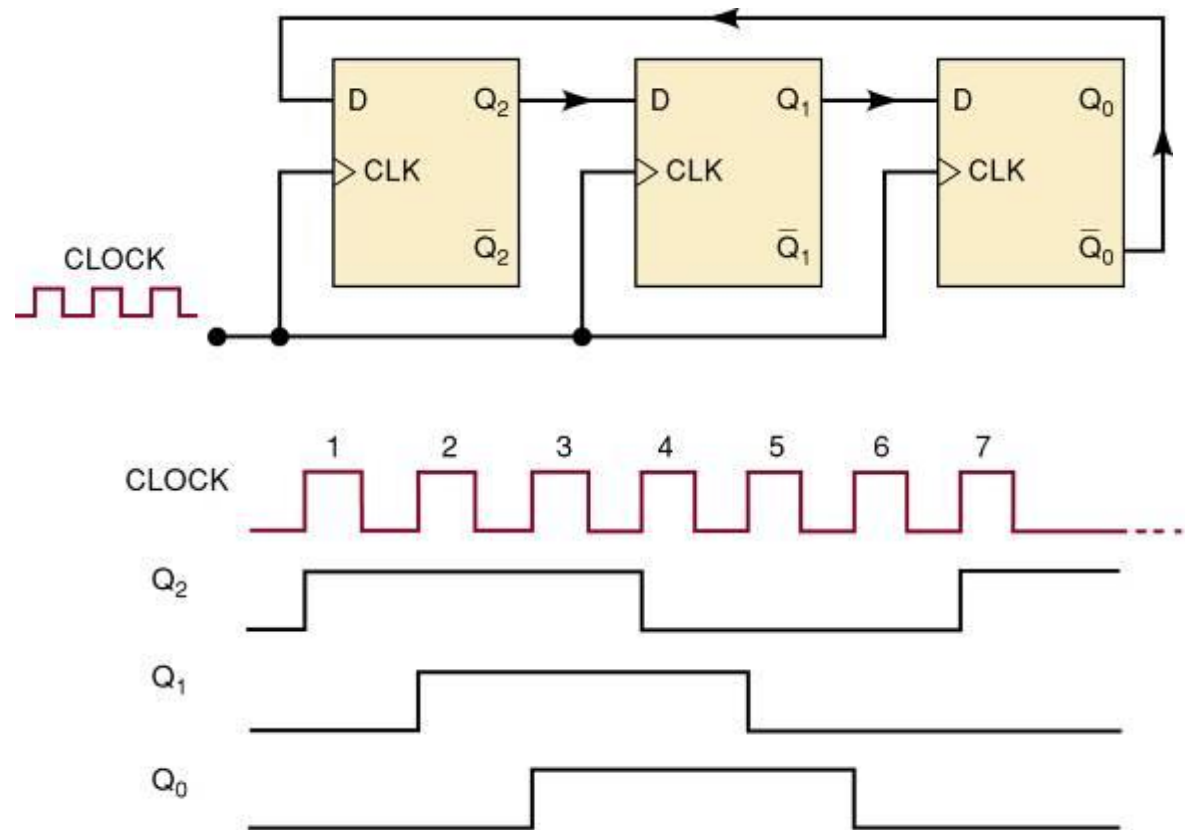
Circuit for ensuring the ring counter starts in the 1000 state on power-up.



7-17 Shift Register Counters

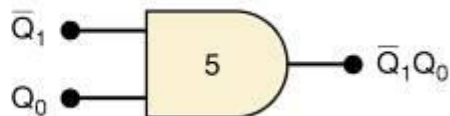
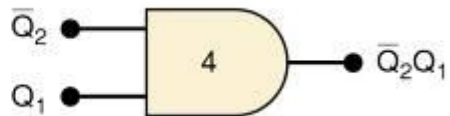
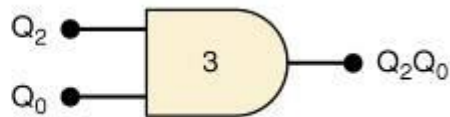
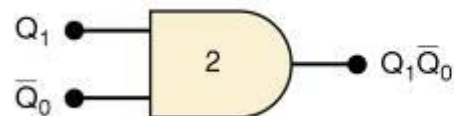
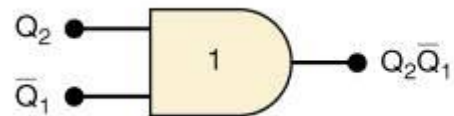
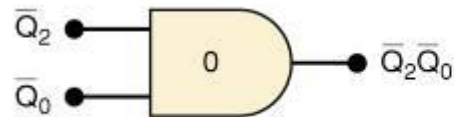
- In the **Johnson** or **twisted-ring counter** *inverted* output of the last FF is connected to the input of the first FF.

Q ₂	Q ₁	Q ₀	CLOCK pulse
0	0	0	0
1	0	0	1
1	1	0	2
1	1	1	3
0	1	1	4
0	0	1	5
0	0	0	6
1	0	0	7
1	1	0	8
.	.	.	.
.	.	.	.
.	.	.	.



7-17 Shift Register Counters

- For a given MOD number, a Johnson counter requires only half the FFs a ring counter requires.
 - It requires decoding gates—a ring counter does not.
- A Johnson counter uses one logic gate to decode for each count.
 - Each gate requires only two inputs, regardless of the number of FFs in the counter.



Q_2	Q_1	Q_0	Active gate
0	0	0	0
1	0	0	1
1	1	0	2
1	1	1	3
0	1	1	4
0	0	1	5

7-17 Shift Register Counters

- Very few ring counters or Johnson counters are available as ICs.
 - It is relatively simple to wire a shift-register IC as either a ring counter or a Johnson counter.
- Some of the CMOS Johnson-counter ICs include the complete decoding circuitry on the same chip as the counter.
 - (74HC4017, 74HC4022)

7-18 Troubleshooting

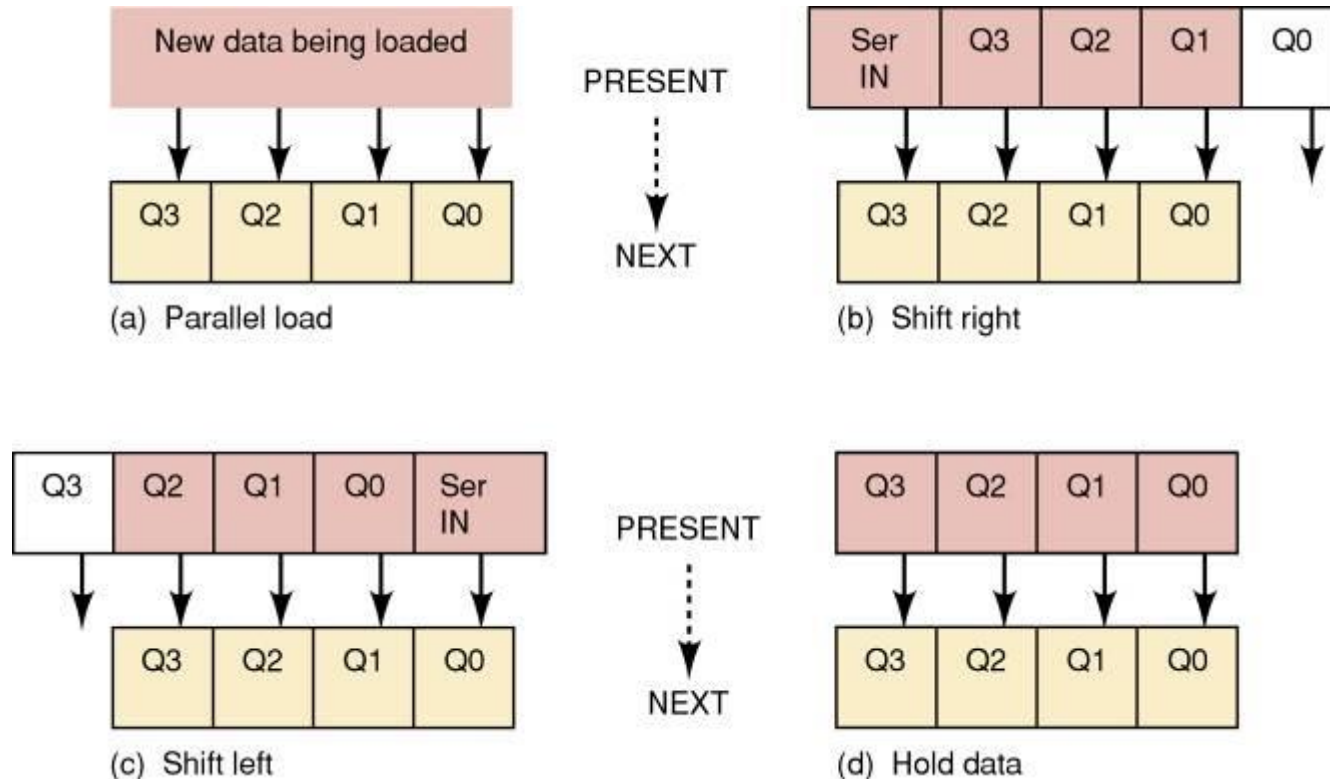
- Flip-flops, counters, and registers are the major components in sequential logic system.
 - Sequential systems suffer from the same types of failures as do combinational systems.
 - Open circuits, shorts, internal IC faults, etc.
- Basic troubleshooting procedures still apply.
 - Observe system operation.
 - Use analytical reasoning to determine possible causes.
 - Use test equipment to isolate the exact fault.

7-20 HDL Registers

- HDL techniques use bit arrays to describe register data & transfer the data in parallel or serial format.
- **Concatenation**—grouping together in a specific sequence—of desired set of data bits can be used to describe data movement for serial shifting.

7-20 HDL Registers

Four flip-flops performing transfer operations of parallel load, shift right, shift left, and hold data.



Bits are transferred synchronously, which means they all move simultaneously on a single clock edge.

7-21 HDL Ring Counters

- A ring counter is a shift register that circulates a single active logic level through all its FFs.
 - Modulus is equal to the number of FFS in the register.
 - There are always many unused and invalid states.
- With planning, we can ensure the counter reaches the desired sequence—no matter the initial state.
 - Regardless of the state to which the counter initializes, it eventually fills with zeros.
 - At which time, logic shifts in a HIGH to start the ring sequence.

7-21 HDL Ring Counters

- In the AHDL code shown, lines 11 & 12 control the serial input using the strategy just described.
 - The (==) operator evaluates whether the expressions on each side are equal or not.

```
1  SUBDESIGN fig7_92
2  (
3      clk          :INPUT;
4      q[3..0]      :OUTPUT;
5  )
6  VARIABLE
7      ff[3..0]     :DFF;
8      ser_in       :NODE;
9  BEGIN
10     ff[].clk = clk;
11     IF ff[3..1].q == B"000" THEN ser_in = VCC;  -- self start
12     ELSE ser_in = GND;
13     END IF;
14     ff[3..0].d = (ser_in, ff[3..1].q);          -- shift right
15     q[] = ff[].q;
16 END;
```

- The concept of a counter can be applied to implement a **digital one-shot** using HDL.
 - A *nonretriggerable* one-shot ignores the trigger input pulse output is still active.
 - A *retriggerable* one-shot starts a pulse in response to a trigger.
 - Internal pulse timer restarts each time a subsequent trigger edge occurs *before* the pulse is complete.

END

ELEVENTH EDITION

Digital Systems

Principles and Applications

Ronald J. Tocci

Monroe Community College

Neal S. Widmer

Purdue University

Gregory L. Moss

Purdue University

PEARSON