Chapter 6
Registers and Counters

Outline

- Registers
- Shift Registers
- Ripple Counters
- Synchronous Counters
- Other Counters
- HDL for Registers and Counters

Registers

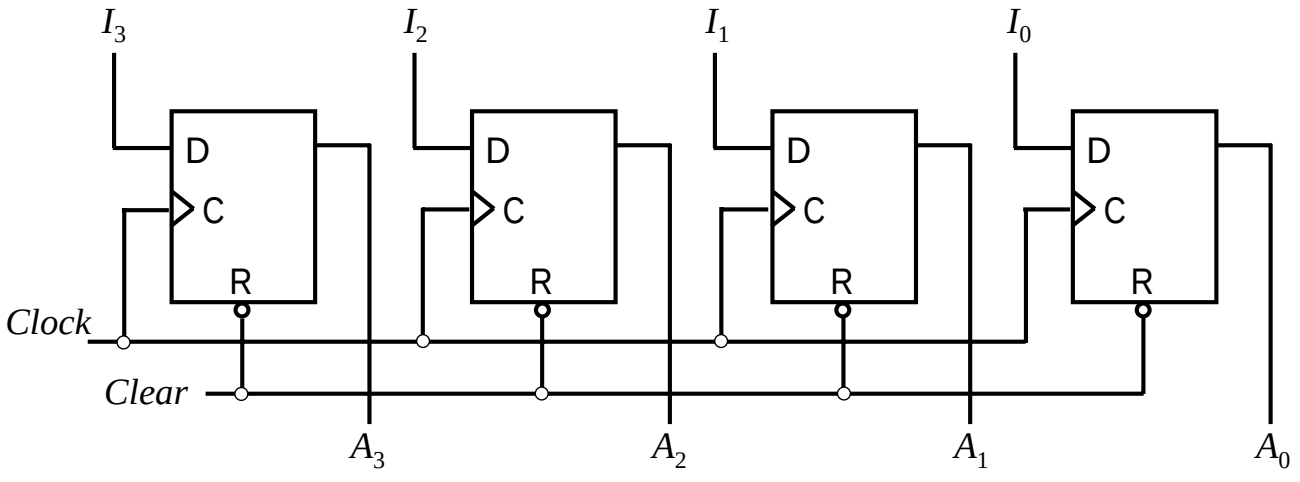
- Clocked sequential circuits
  - A group of flip-flops and combinational gates
  - Two typical clocked sequential circuits are *registers* and *counters*.

    Flip-flops + Combinational gates

    (essential) (optional)
- Register:
  - a group of flip-flops holding the binary information
  - gates that determine how the information is transferred into the register
- Counter:
  - a register that goes through a predetermined sequence of states

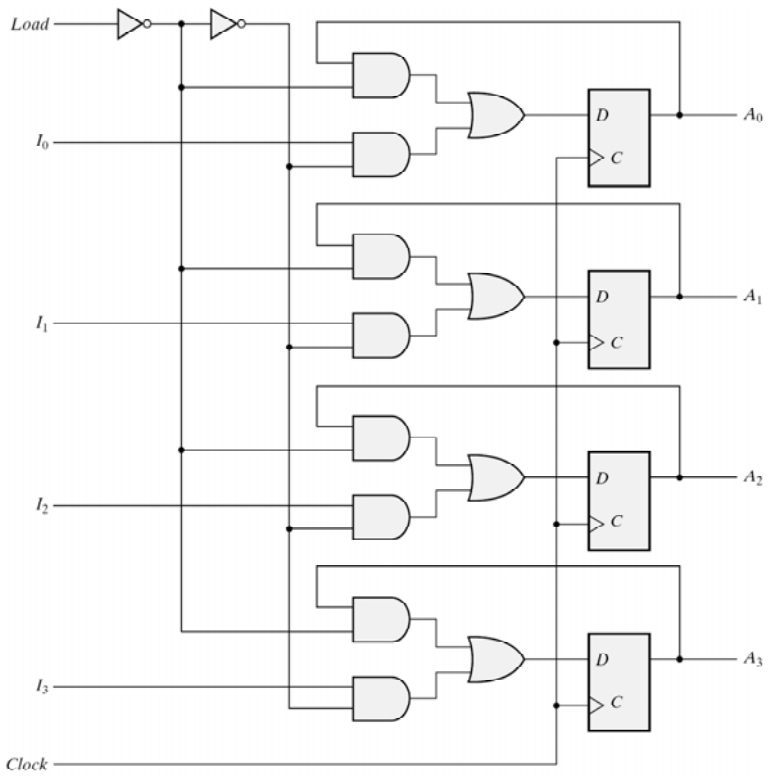
Registers

- An n-bit register
  - n flip-flops are capable of storing n bits of binary information
  - Four-bit register



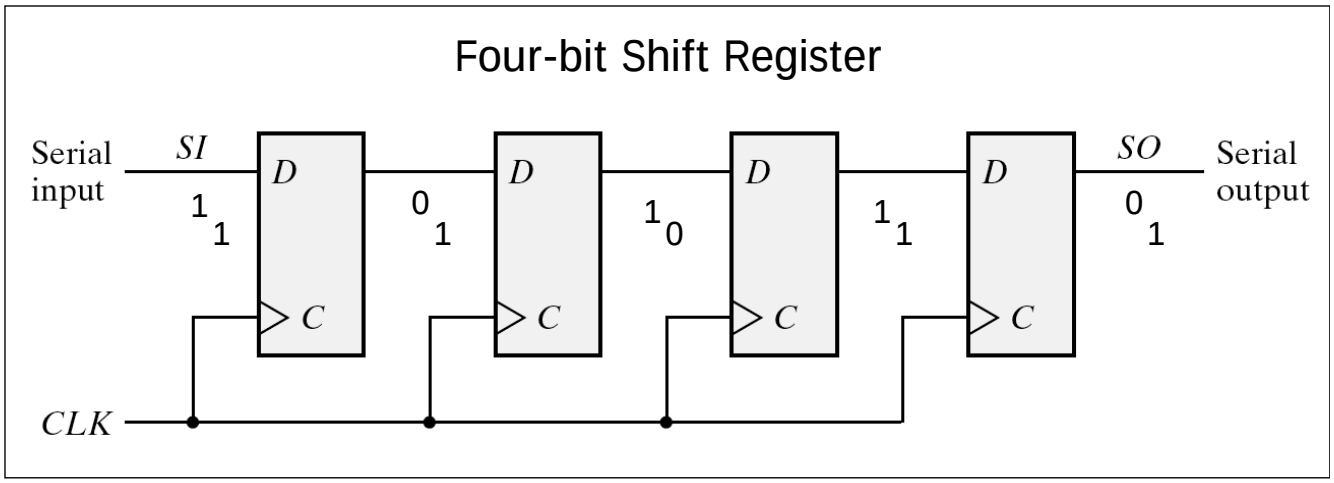
4-bit Register with Parallel Load

- Transfer (loading) of new information into the registers.
- All the bits all loaded to the flip-flops synchronously within one clock period.
- When load is disable, the information in the flip-flops must be left unchanged.



Shift Registers

- Shift register
  - a register capable of shifting its binary information in one or both directions
- Simplest shift register using flip-flops

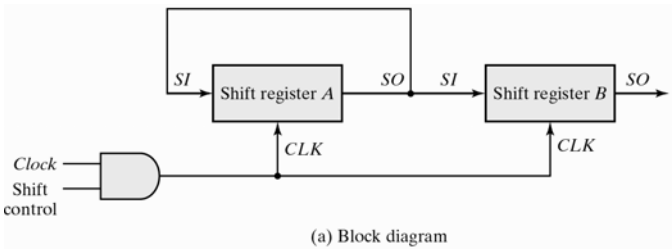


Serial Transfer

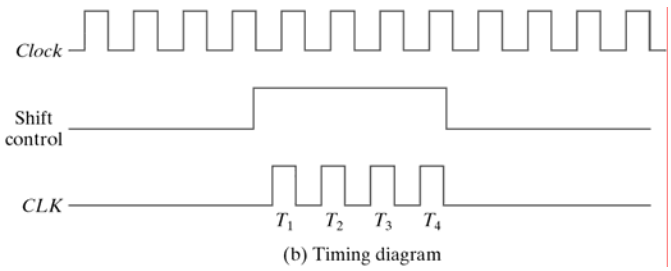
- Serial transfer vs. Parallel transfer
- Serial transfer:
  - Information is transferred and manipulated one bit at a time
  - shifts the bits out of the source register into the destination register
- Parallel transfer:
  - All the bits of the register are transferred and manipulated at the same time

Serial Transfer

- Example: Serial transfer from reg A to reg B
- Feedback of SO to SI in shift register A in order to prevent the loss of information.
- Information are shifted form register A to register B within n clock period.
- Information of shift register B will be lost unless it is connected to 3rd shift register.
- Shift control (negative edge triggered) determines when and how many times the registers are shifted.



(a) Block diagram



(b) Timing diagram

Table 6.1
Serial-Transfer Example

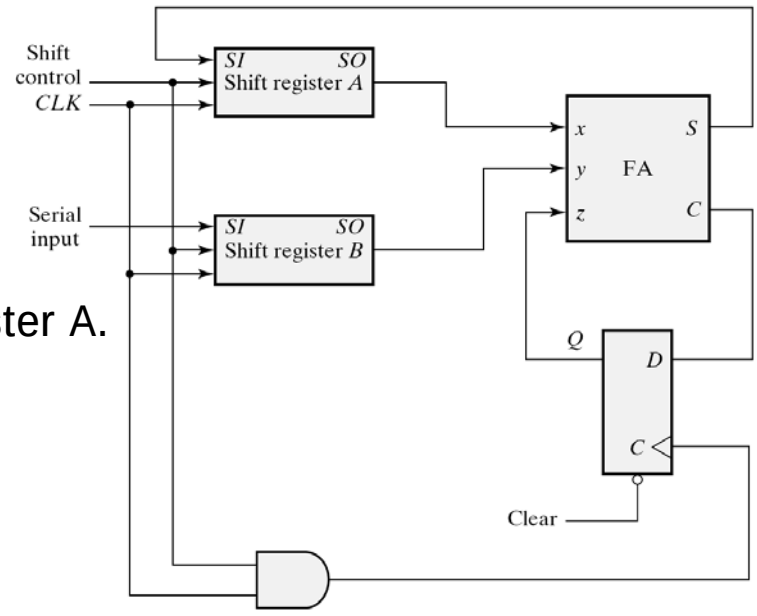
Timing Pulse	Shift Register *A*	Shift Register *B*
Initial value	1 0 1 1	0 0 1 0
After T_1	1 1 0 1	1 0 0 1
After T_2	1 1 1 0	1 1 0 0
After T_3	0 1 1 1	0 1 1 0
After T_4	1 0 1 1	1 0 1 1

Serial Addition vs. Parallel Addition

- Parallel mode vs. serial mode
  - Operation in parallel mode is faster, but requires more hardware.
  - Operation in serial mode is slower, but requires less hardware.
- Serial adder vs. Parallel adder
  - Serial adder transfer data using shift register while parallel adder transfer data using parallel load.
  - Serial adder uses only one full adder while parallel adder uses n full adders.
  - Serial adder is a sequential circuit while parallel adder is a combinational circuit.

Serial Addition using D Flip-Flops

- Initially, augend is in register A and addend is in register B.
- Shift control enables the triggering of clock and 1-bit addition of two operands from LSB to MSB.
- A new sum (S) bit is transferred to shift register A.
- A carry-out (C) of full adder is transferred to Q flip-flop as the z input of next addition.
- Finally, when the shift control is disabled, summation result is stored in shift register A.



Serial Adder using JK Flip-Flops

- Serial adder can be redesign using sequential circuit design method.

Table 6.2
State Table for Serial Adder

Present State	Inputs		Next State	Output	Flip-Flop Inputs	
Q	x	y	Q	S	J_Q	K_Q
0	0	0	0	0	0	X
0	0	1	0	1	0	X
0	1	0	0	1	0	X
0	1	1	1	0	1	X
1	0	0	0	1	X	1
1	0	1	1	0	X	0
1	1	0	1	0	X	0
1	1	1	1	1	X	0

- By K Map:

$J_Q = x\,y$

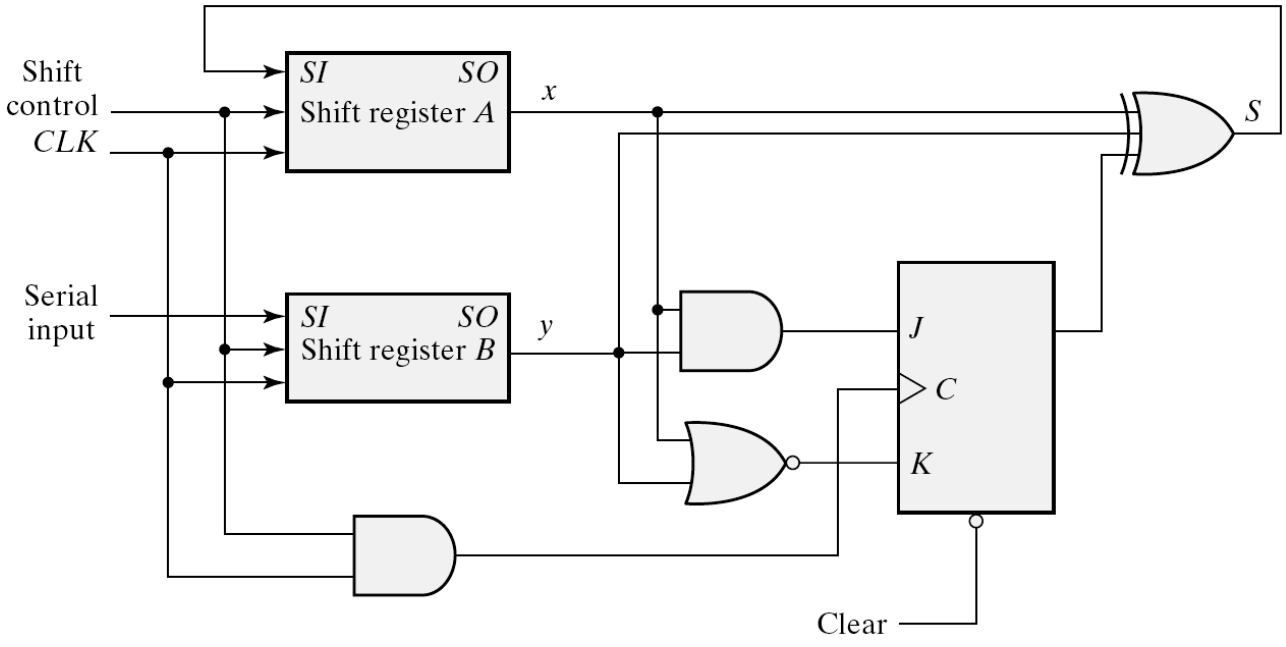
$K_Q = x'\,y' = (x + y)'$

$S = x \oplus y \oplus Q$

Flip-Flop Excitation Tables

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

Circuit Diagram



Universal Shift Register

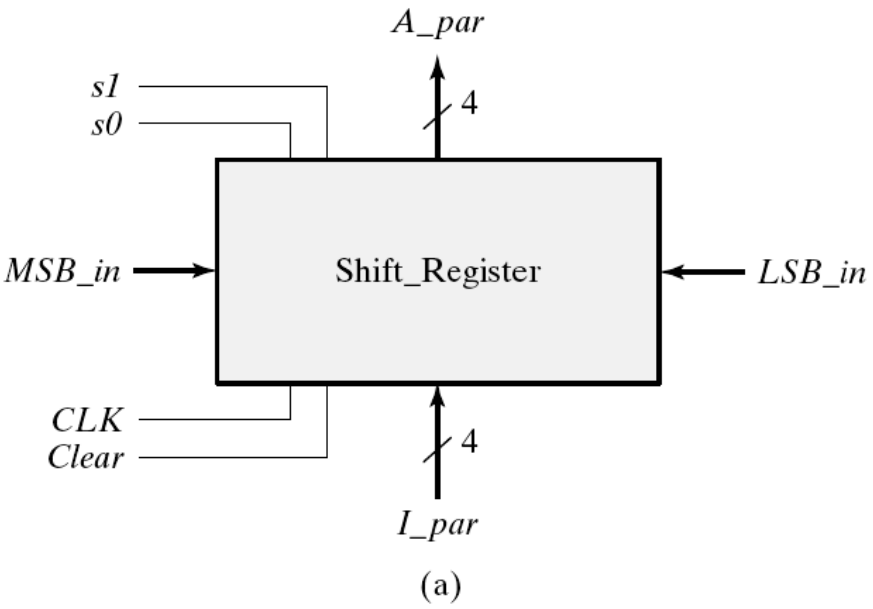
- Some Shift Register Types:
  - Unidirectional shift register
    - Shifting in one direction
  - Bidirectional shift register
    - Shifting in both direction
  - Universal shift register:
    - Having capabilities of both-direction shifting & parallel load/out.

Capability of a Universal Shift Register

- 1. A *clear* control to clear the register to 0.
- 2. A *clock* input to synchronize the operations.
- 3. A *shift-right* control to enable the shift right operation and the *serial input* and *output* lines associated with the shift right.
- 4. A *shift-left* control to enable the shift left operation and the *serial input* and *output* lines associated with the shift left.
- 5. A *parallel-load* control to enable a parallel transfer and the n *parallel input* lines associated with the parallel transfer.
- 6. n *parallel output* lines.
- 7. A control state that leaves the information in the register unchanged in the presence of the clock.

Universal Shift Register

- Four-bit universal shift register:



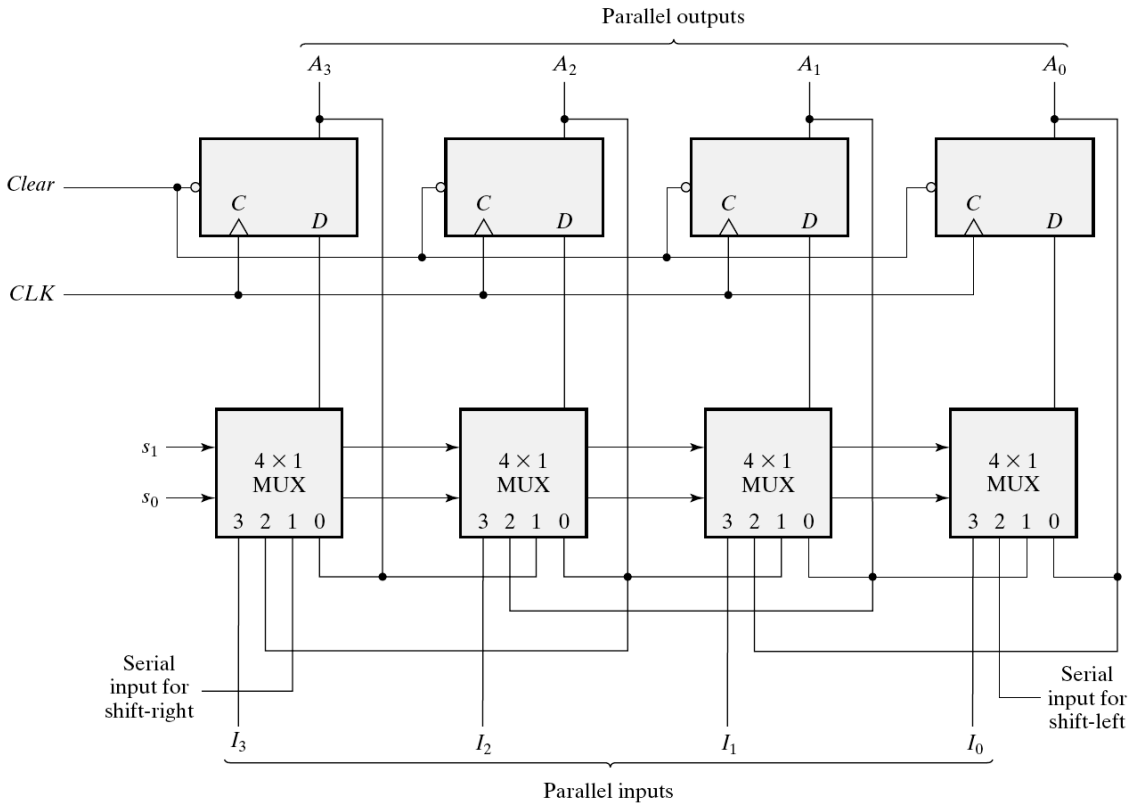
(a)

Function Table

- Function Table:

Clear	s_1	s_0	A_3^+	A_2^+	A_1^+	A_0^+	(operation)
0	x	x	0	0	0	0	Clear
1	0	0	A_3	A_2	A_1	A_0	No change
1	0	1	sri	A_3	A_2	A_1	Shift right
1	1	0	A_2	A_1	A_0	sli	Shift left
1	1	1	I_3	I_2	I_1	I_0	Parallel load

4-Bit Universal Shift Register



(b)

Ripple Counters

- Counter:
  - A register that goes through a prescribed sequence of states upon the application of input pulses
    - Input pulses: may be clock pulses or originate from some external source
    - The sequence of states: may follow the binary number sequence (⇒ Binary counter) or any other sequence of states

Categories of Counters

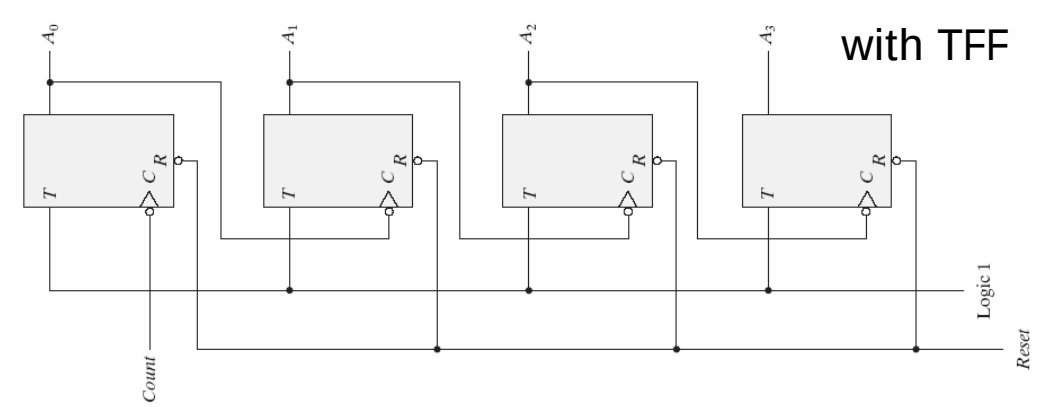
- 1. Ripple counters:
  - The flip-flop output transition serves as a source for triggering other flip-flops

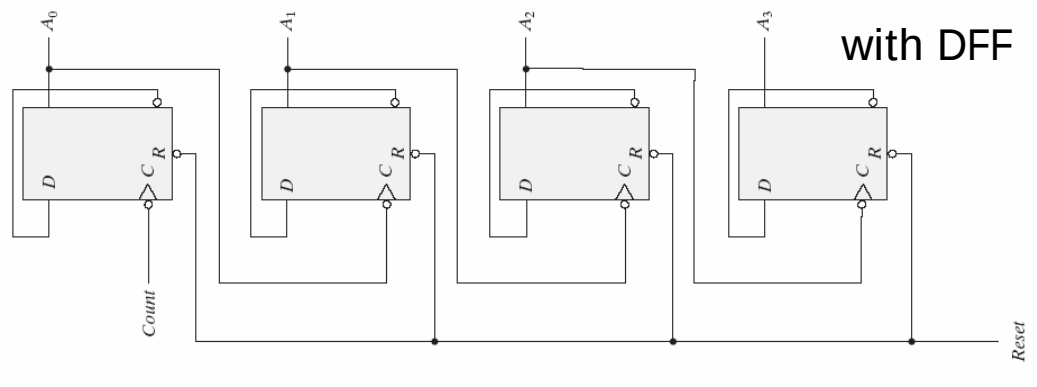
    ⇒ no common clock pulse (not synchronous)
- 2. Synchronous counters:
  - The CLK inputs of all flip-flops receive a common clock
- Example: 4-bit binary ripple counter
  - A complementing flip-flop is used in the least significant bit as the trigger source.
  - JK flip-flop, T-flip-flop, and D-flip-flop

Binary Counter Sequence

A_3	A_2	A_1	A_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

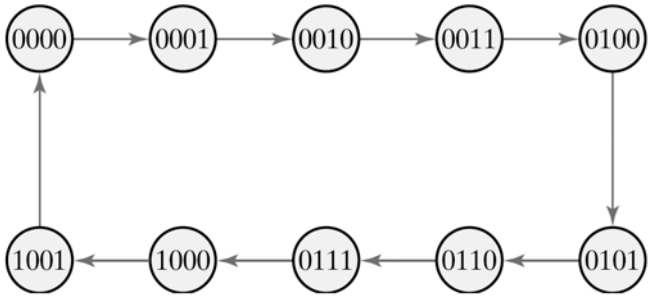
4-bit Binary Ripple Counter





BCD Ripple Counter

- State Diagram of a BCD counter

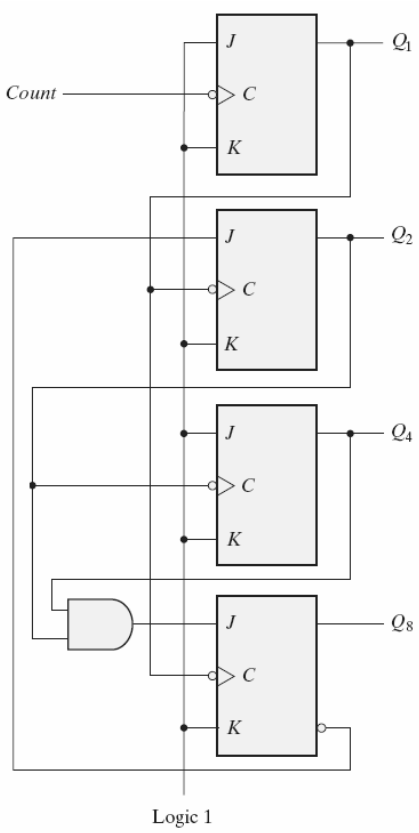


- State table for BCD ripple counter
- Excitation Table of JK flip-flop
- K-Map method

Present State				Next State			
Q_8	Q_4	Q_2	Q_1	Q_8	Q_4	Q_2	Q_1
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	0	0	0	0

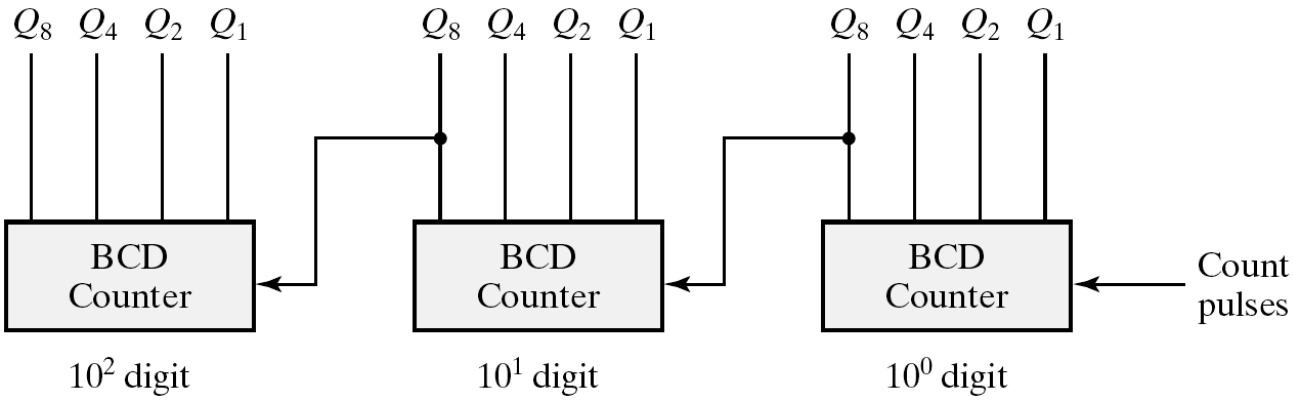
Logic Diagram of a BCD Ripple Counter

- The circuit for one-decade BCD counter
  - When J = 1 K = 1, JK is complementing flip-flop.
- Observation from the state table
  - Q_1: Complementing Q_1 for every negative count
  - Q_2: Complementing Q_2 at negative Q_1 when Q_8 is 0
  - Q_4: Complementing Q_4 at negative Q_2
  - Q_8: Complementing Q_8 at negative Q_4 when Q_2 =1 and Q_4 = 1.



Three-decade BCD Counter

- Block diagram
  - When Q_8 in one decade goes from 1 to 0, it triggers the count for the next higher-order decade while its own decade goes from 9 to 0.

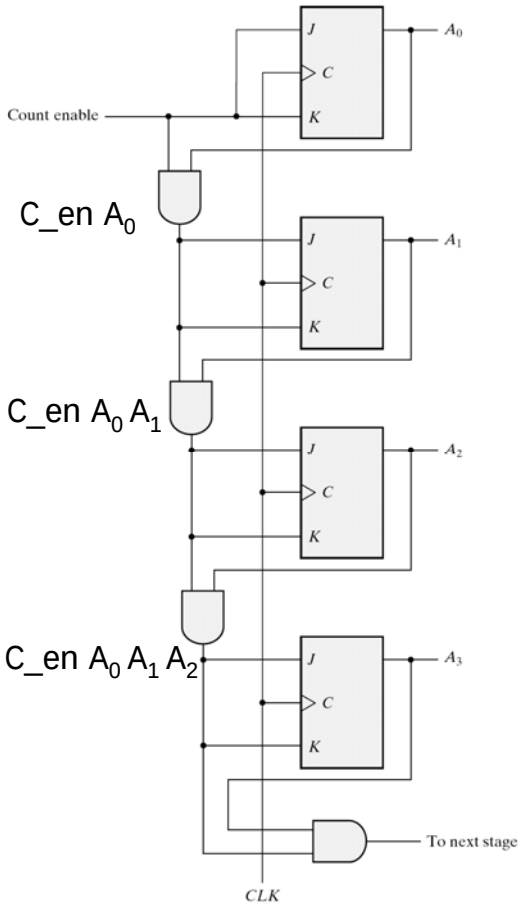


Synchronous Counters

- Synchronous counter
  - A common clock triggers all flip-flops simultaneously
- Design procedure
  - Apply the same procedure of synchronous sequential circuits.
  - Synchronous counter is simpler than general synchronous sequential circuit because of its regularity.

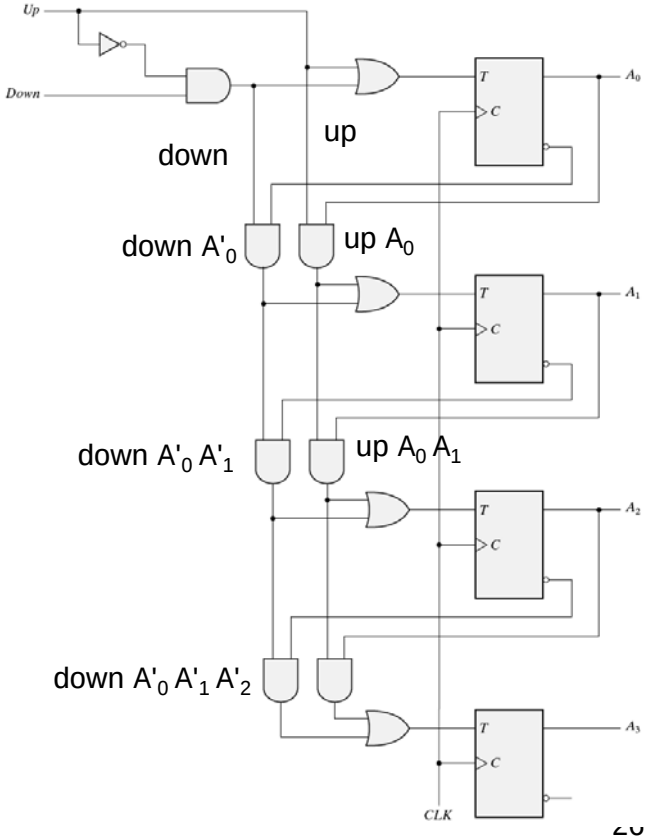
Four-bit Synchronous Binary Counter

- 4-bit synchronous binary counter
  - A_1 counts when C_en = 1 and A_0 = 1
  - A_2 counts when C_en = 1, A_0 = 1, and A_1 = 1.
  - A_3 counts when C_en = 1, A_0 = 1, A_1 = 1, and A_2 = 1.
- The synchronous counter can be triggered with either the positive or the negative clock edge



4-bit Up/Down Binary Counter

- 4-bit up/down binary counter
  - The bit in the least significant position is complemented with each pulse.
  - *Up counter*: A bit is complemented if all lower significant bits are equal to 1.
  - *Down counter*: A bit is complemented if all lower significant bits are equal to 0.



BCD Counters

- A BCD counter does not have a regular pattern as in a straight binary count, thus need to go through a sequential circuit procedure.

Table 6.5
State Table for BCD Counter

Present State				Next State				Output	Flip-Flop Inputs			
Q_8	Q_4	Q_2	Q_1	Q_8	Q_4	Q_2	Q_1	y	TQ_8	TQ_4	TQ_2	TQ_1
0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	1	0	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	0	1
0	0	1	1	0	1	0	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	0	1
0	1	0	1	0	1	1	0	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	0	1
0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	0	1
1	0	0	1	0	0	0	0	1	1	0	0	1

Excitation Function and Mapping

- K-map
  - $T_{Q1} = 1$
  - $T_{Q2} = Q_8'Q_1$
  - $T_{Q4} = Q_2Q_1$
  - $T_{Q8} = Q_8Q_1 + Q_4Q_2Q_1$
  - $y = Q_8Q_1$

$Q(t)$	$Q(t+1)$	T
0	0	0
0	1	1
1	0	1
1	1	0

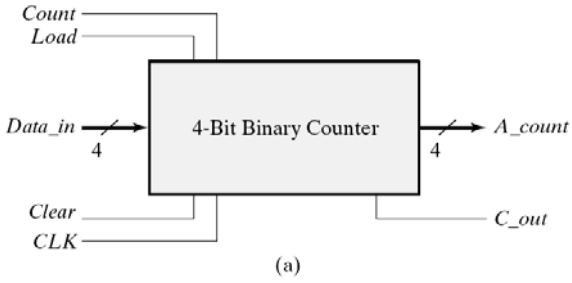
- • Circuit Diagram

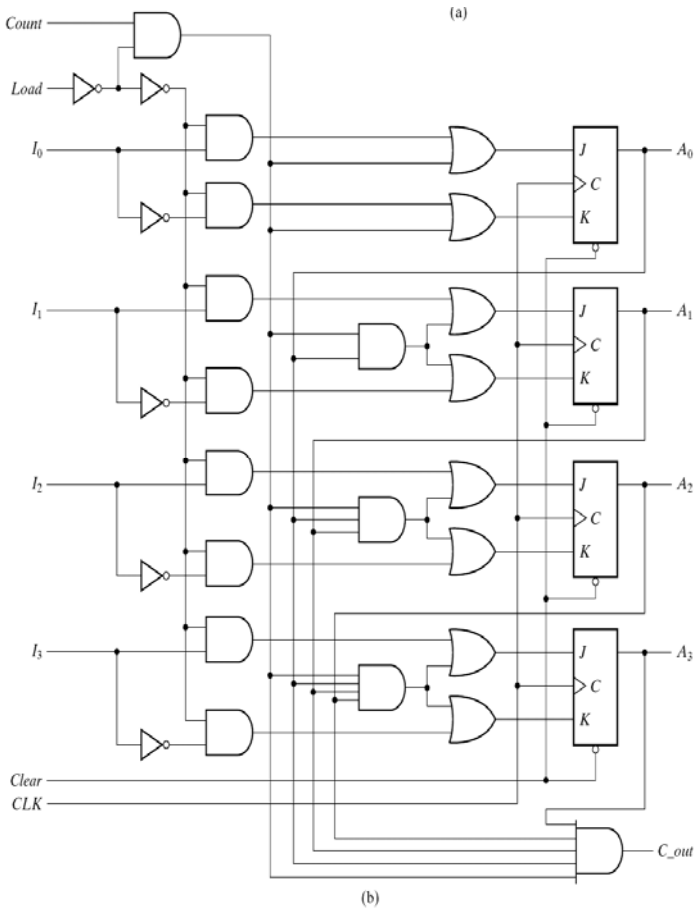
Table 6.6
Function Table for the Counter of Fig. 6.14

Clear	CLK	Load	Count	Function
0	X	X	X	Clear to 0
1	↑	1	X	Load inputs
1	↑	0	1	Count next binary state
1	↑	0	0	No change

4-bit Binary Counter with Parallel Load

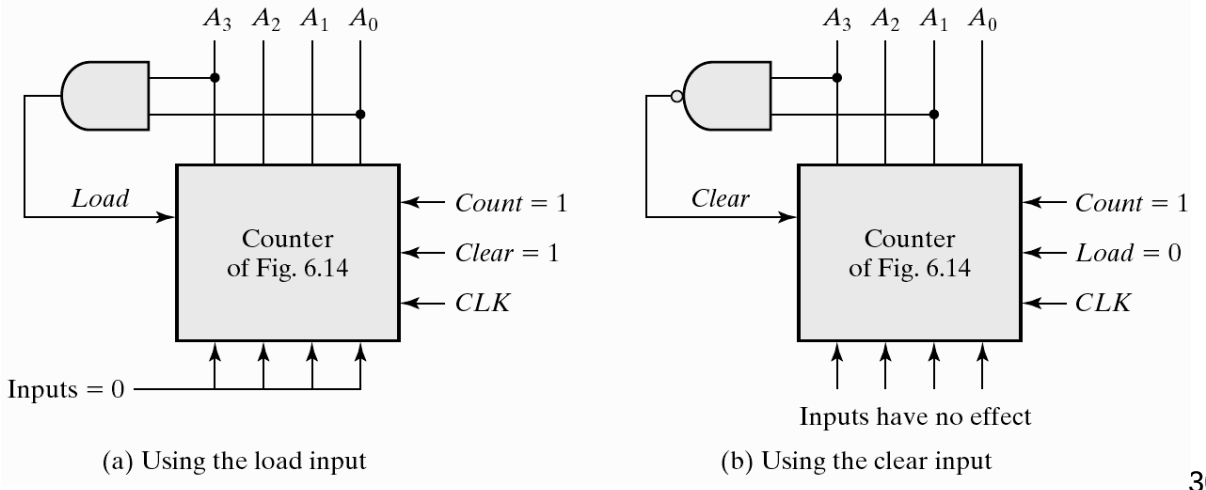
- 4-bit binary counter with parallel load
- For each JK flip-flop, complementing is executed when the least significant bits are all 1's.





BCD Counter Using a Binary Counter

- Generate any count sequence:
  - The AND gate in (a) detects the occurrence of state 1001, then the counter loads input 0000.
  - The NAND gate in (b) detects the occurrence of state 1001, then clears the counter and reset the counter to 0000.



Other Counters

- Counters can be designed to generate any desired sequence of states
- Divide-by-*N* counter (modulo-N counter)
  - A counter that goes through a repeated sequence of *N* states
  - The sequence may follow the binary count or may be any other arbitrary sequence

Counters with Unused States

- n flip-flops $\Rightarrow 2^n$ binary states
- Unused states
  - states that are not used in specifying the FSM and may be treated as don't-care conditions or may be assigned specific next states
- Self-correcting counter
  - Ensure that when a circuit enter one of its unused states, it eventually goes into one of the valid states after one or more clock pulses so it can resume normal operation.

    $\Rightarrow$ Analyze the circuit to determine the next state from an unused state after it is designed

Counters with Unused States

- An example:

Table 6.7
State Table for Counter

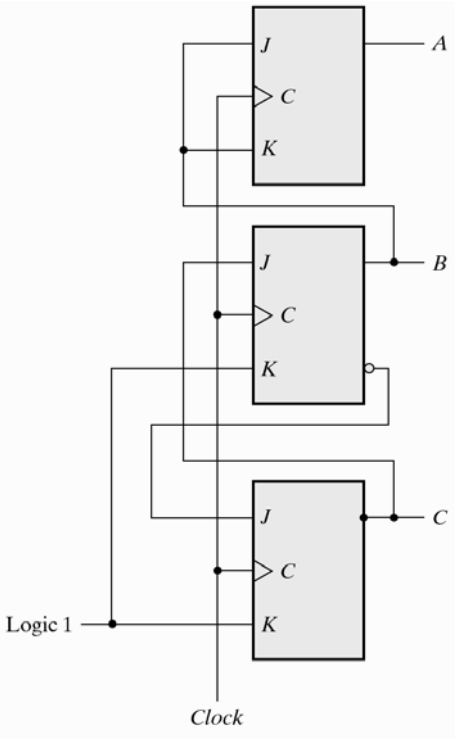
Present State			Next State			Flip-Flop Inputs					
A	B	C	A	B	C	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	1	0	0	1	X	X	1	0	X
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	0	0	0	X	1	X	1	0	X

Two unused states: 011 & 111
The simplified flip-flop input equations:

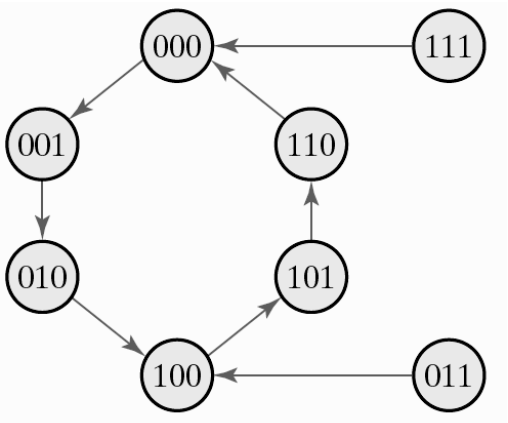
$J_A = B$, $K_A = B$
$J_B = C$, $K_B = 1$
$J_C = B'$, $K_C = 1$

Counter with Unused States

- The logic diagram and state diagram of the circuit



(a) Logic diagram

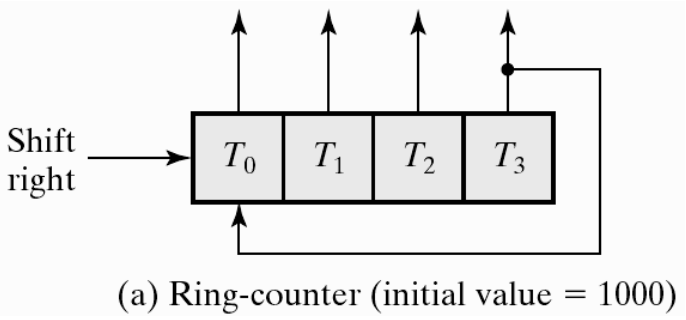


(b) State diagram

Ring Counter

- A circular shift register w/ only one flip-flop being set at any particular time, all others are cleared
  (initial value = 1 0 0 … 0)
- The single bit is shifted from one flip-flop to the next to produce the sequence of timing signals.
- An example: 4-bit counter

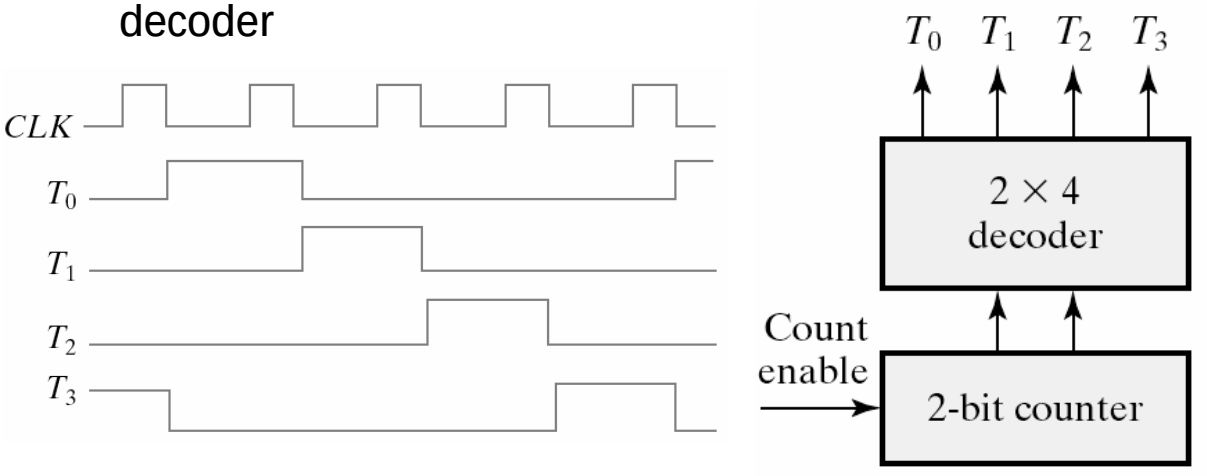
A_3	A_2	A_1	A_0
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0



(a) Ring-counter (initial value = 1000)

Ring Counter

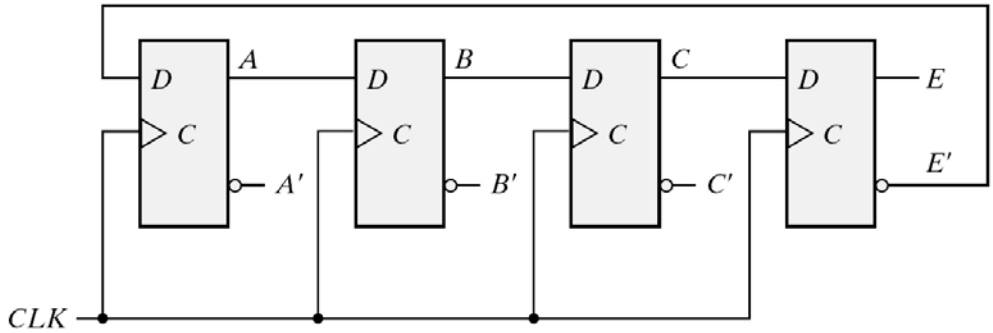
- Application of counters
  - Counters may be used to generate timing signals to control the sequence of operations in a digital system.
- Approaches for generation of 2^n timing signals
  1. a shift register w/ 2^n flip-flops
  2. an n-bit binary counter together w/ an n-to-2^n-line decoder



Johnson Counter

- Ring counter vs. Switch-tail ring counter
  - Ring counter
    - a k-bit ring counter circulates a single bit among the flip-flops to provide k distinguishable states.
  - Switch-tail ring counter
    - is a circular shift register w/ the complement output of the last flip-flop connected to the input of the first flip-flop
    - a k-bit switch-tail ring counter will go through a sequence of $2k$ distinguishable states. (initial value = 0 0 … 0)

Switch-Tail Ring Counter



(a) Four-stage switch-tail ring counter

Sequence number	Flip-flop outputs A	B	C	E	AND gate required for output
1	0	0	0	0	$A'E'$
2	1	0	0	0	AB'
3	1	1	0	0	BC'
4	1	1	1	0	CE'
5	1	1	1	1	AE
6	0	1	1	1	$A'B$
7	0	0	1	1	$B'C$
8	0	0	0	1	$C'E$

(b) Count sequence and required decoding

Johnson Counter

- Johnson counter
  - a k-bit switch-tail ring counter + $2k$ decoding gates
  - provide outputs for $2k$ timing signals
    - E.g.: 4-bit Johnson counter
  - The decoding follows a regular pattern:
    - 2 inputs per decoding gate

Summary

- Disadvantage of the switch-tail ring counter
  - If it finds itself in an unused state, it will persist to circulate in the invalid states and never find its way to a valid state.
  - One correcting procedure: $D_C = (A + C)\,B$
- Summary:
  - Johnson counters can be constructed for any # of timing sequences:

    # of flip-flops = 1/2 (the # of timing signals)

    # of decoding gates = # of timing signals

    2-input per gate

HDL for Registers and Counters

- Shift Register
- Statement: A_par <+ {MSB_in, A_par [3: 1]}
  - specifies a concatenation of serial data input for a right shift operation (MSB_in) with bits A_par[3 : 1] of the output data bus.

Behavioral Description of a 4-bit Universal Shift Register

```
// Fig. 6.7 and Table 6.3
module Shift_Register_4_beh (                          // V2001, 2005
  output reg     [3: 0]   A_par,                       // Register output
  input          [3: 0]   I_par,                       // Parallel input
  input                   s1, s0,                      // Select inputs
                          MSB_in, LSB_in,              // Serial inputs
                          CLK, Clear                   // Clock and Clear
);

 always @ (posedge CLK, negedge Clear) // V2001, 2005
    if (~Clear) A_par <= 4'b0000;
    else
      case ({s1, s0})
        2'b00: A_par <= A_par;                         // No change
        2'b01: A_par <= {MSB_in, A_par[3: 1]};         // Shift right
        2'b10: A_par <= {A_par[2: 0], LSB_in};         // Shift left
        2'b11: A_par <= I_par;                         // Parallel load of input
      endcase
endmodule
```

HDL for Universal Shift Register

```
// Structural description of a 4-bit universal shift register (see Fig. 6.7)
module Shift_Register_4_str (                          // V2001, 2005
  output [3: 0]   A_par,                               // Parallel output
  input  [3: 0]   I_par,                               // Parallel input
  input           s1, s0,                              // Mode select
  input           MSB_in, LSB_in, CLK, Clear           // Serial inputs, clock, clear
);

// bus for mode control
  wire   [1:0]    select = {s1, s0};

// Instantiate the four stages
  stage ST0 (A_par[0], A_par[1], LSB_in, I_par[0], A_par[0], select, CLK, Clear);
  stage ST1 (A_par[1], A_par[2], A_par[0], I_par[1],A_par[1], select, CLK, Clear);
  stage ST2 (A_par[2], A_par[3], A_par[1], I_par[2], A_par[2], select, CLK, Clear);
  stage ST3 (A_par[3], MSB_in, A_par[2], I_par[3], A_par[3], select, CLK, Clear);
endmodule
```


HDL for Universal Shift Register

```
// One stage of shift register
module stage (i0, i1, i2, i3, Q, select,
CLK, Clr);
  input  i0,        // circulation bit
selection
         i1,        // data from left
neighbor or serial input for shift-right
         i2,        // data from right
neighbor or serial input for shift-left
         i3;        // data from parallel
input
 output Q;
  input [1: 0]     select;   // stage
mode control bus
  input  CLK, Clr;           // Clock,
Clear for flip-flops
  wire   mux_out;

// instantiate mux and flip-flop
  Mux_4_x_1 M0 (mux_out, i0, i1, i2, i3,
select);
 D_flip_flop M1 (Q, mux_out, CLK, Clr);
endmodule

// 4x1 multiplexer
// behavioral model
module Mux_4_x_1 (mux_out, i0, i1, i2,
i3, select);
  output          mux_out;
  input           i0, i1, i2, i3;
  input [1: 0]    select;
  reg             mux_out;
  always @ (select, i0, i1, i2, i3)
    case (select)
      2'b00:      mux_out = i0;
      2'b01:      mux_out = i1;
      2'b10:      mux_out = i2;
      2'b11:      mux_out = i3;
    endcase
endmodule
```

HDL for Binary Counter with Parallel Load

```
module Binary_Counter_4_Par_Load (

  output reg      [3:0]     A_count, // Data output
  output                    C_out,   // Output carry
  input           [3:0]     Data_in, // Data input
  input                     Count,   // Active high to count
                            Load,    // Active high to load
                            CLK,     // Positive edge sensitive
                            Clear    // Active low
);

assign C_out = Count & (~Load) & (A_count == 4'b1111);
always @ (posedge CLK, negedge Clear)
  if (~Clear)               A_count <= 4'b0000;
  else if (Load)            A_count <= 4'b0000;
  else if (Count)           A_count <= A_count + 1'b1;
  else                      A_count <= A_count;       // redundant statement
endmodule
```


HDL for Ripple Counter

```
//Ripple Counter
`timescale 1ns / 100 ps
module Ripple_Counter_4bit (A3, A2, A1, A0, Count, Reset);
   output          A3, A2, A1, A0;
   input           Count,Reset;
//Instantiate complementing flip-flop
   Comp_D_flip_flop F0 (A0, Count, Reset);
   Comp_D_flip_flop F1 (A1, A0, Reset);
   Comp_D_flip_flop F2 (A2, A1, Reset);
   Comp_D_flip_flop F3 (A3, A2, Reset);
endmodule
//Complementing flip-flop with delay
//Input to D flip-flop = Q'
module Comp_D_flip_flop (Q, CLK, Reset);
   output          Q;
   input           CLK, Reset;
   reg             Q;
   always @ (negedge CLK, posedge Reset)
   if (Reset) Q <= 1'b0; else Q <= #2 ~Q;
endmodule
```

HDL for Ripple Counter

```
//Stimulus for testing ripple counter
module t_Ripple_Counter_4bit;
  reg    Count;
  reg    Reset;
  wire   A0, A1, A2, A3;
//Instantiate ripple counter
  Ripple_Counter_4bit M0 (A3, A2, A1, A0, Count, Reset);
  always
    #5 Count = ~Count;
  initial
    begin
        Count = 1'b0;
        Reset = 1'b1;
        #4 Reset = 1'b0;
    end

  initial #170 $finish;

endmodule
```


Simulation Output