Chap. 6 Registers and Counters

6-1 Registers

Register : a group of FF’s or a group of binary cells F-F을 모아놓은 것

suitable for holding binary information

*n-bit register = a group of n FF’s + control gates

control when and how new information

is transferred into the register

Counter : a register that goes a predetermined sequence of states upon

the application of input pulses

gates in a counter control counting sequence

module : Register를 모아놓은 것

paralel : 두개 이상
loading : 싣는다

Simple example :

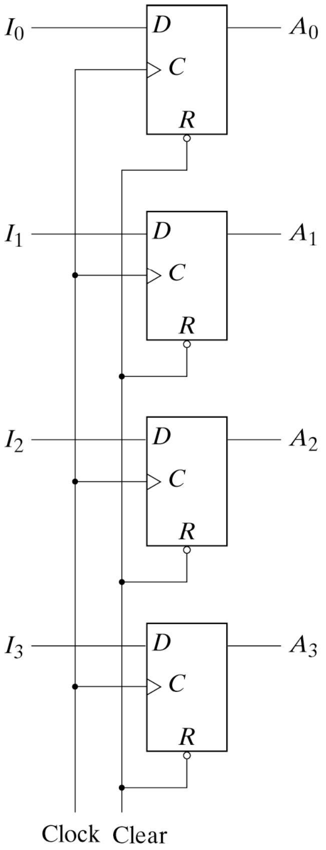
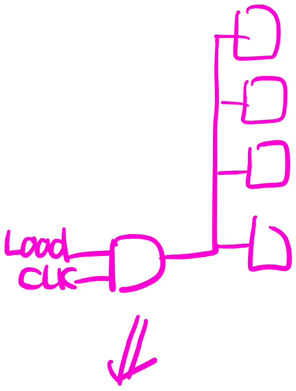
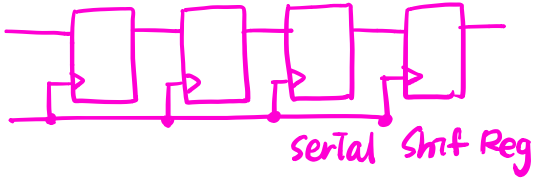


Fig. 6-1 4-Bit Register

Method I



→ 동기가 깨진다



Register with parallel load

옮기다. 전송. 적재. 싣다

loading : transfer of new information into a register

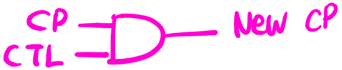
parallel loading : simultaneous loading to all the bits of register with a single CP

clock Pulse

loading control by

- master clock (CP) …….. *Method II*
- separate control signal (CTL) to specify a particular register ….. *Method I*

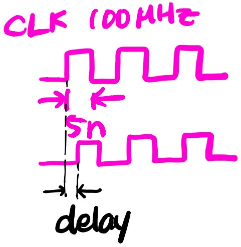
Method I



new_CP <= clk and CTL

gate produces t_{pd}

--> this may throw the system out of synchronism



Method II (recommended)

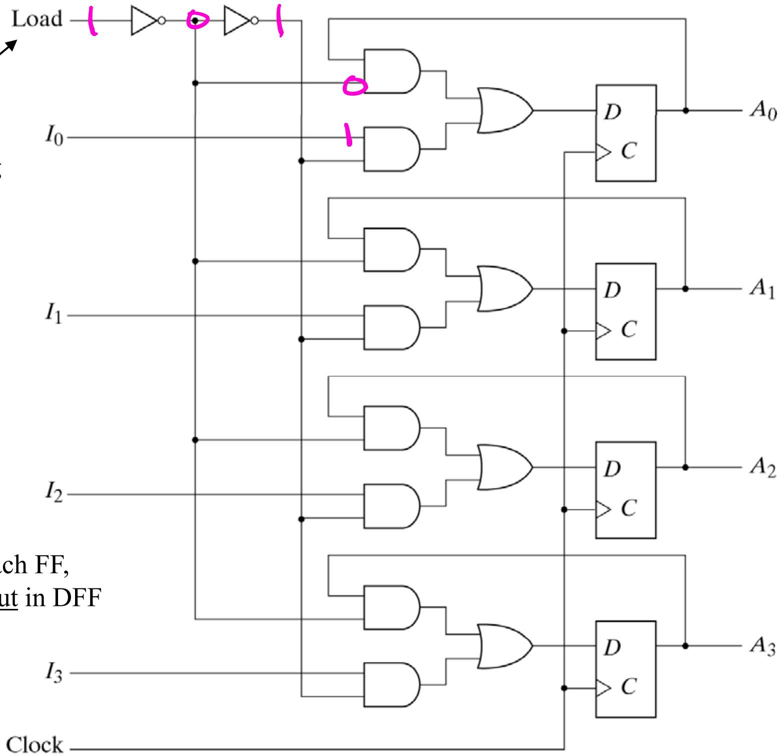
apply CP directly to all FF’s

and control the operation of the register with other inputs

Load high : 새로운 데이터 저장

Load low : 기존의 데이터 다시저장

Method II example



- not with CP
- controls loading

* Check the FB connection in each FF,
because of no "no change" input in DFF

Fig. 6-2 4-Bit Register with Parallel Load

6-2 Shift Registers

: registers capable of shifting its binary information to the right or to the left.

<u>The simplest</u>

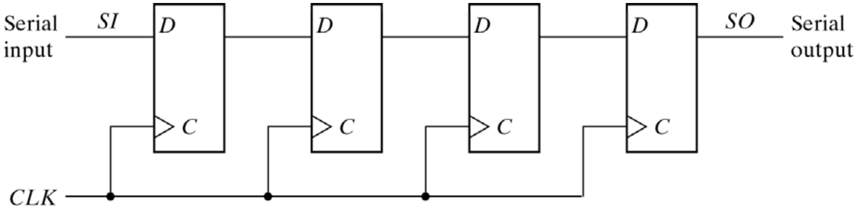


Fig. 6-3 4-Bit Shift Register

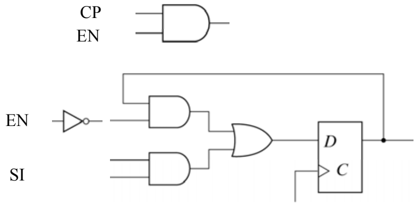
Shift right

Shift left ?

How to control shift ?

- shift only with certain pulses

method I or II of 2 pages before



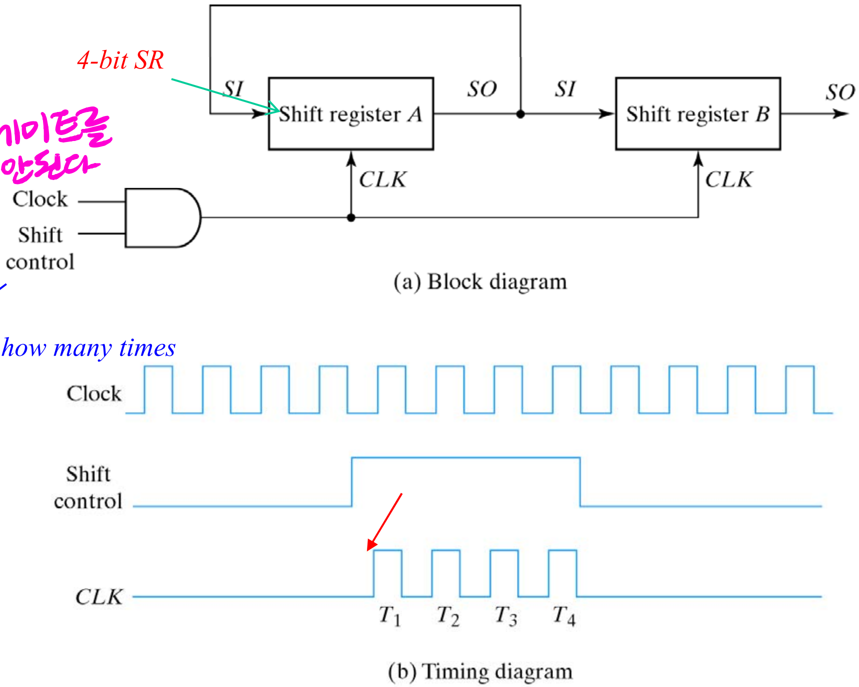
Serial Transfer or load

In a serial mode,
information is transferred and manipulated one bit at a time

4-bit SR

Clock이 게이트를 지나가면 안된다

Control when and how many times



(a) Block diagram

(b) Timing diagram

Fig. 6-4 Serial Transfer from Register A to register B

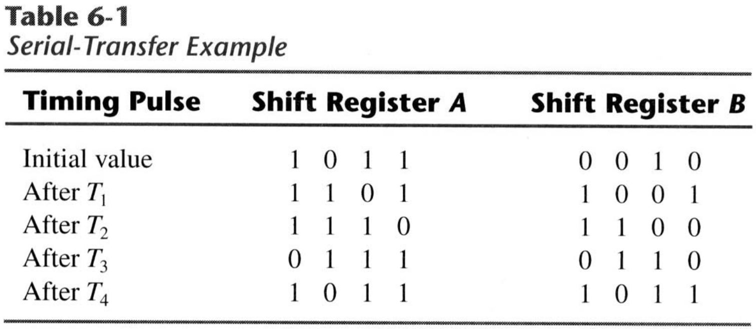


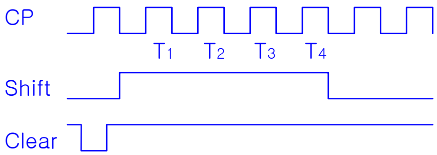
Table 6-1
Serial-Transfer Example

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 vs. parallel transfer

serial : one bit at a time, single input and output → less hardware
parallel : all bits at a time, several inputs and outputs → fast

Serial Adder



	A	B	x	y	z	S	C
Initial value	1011	0011	1	1	0	0	1
After T1	0101	x001	1	1	1	1	1
" 2							
" 3							
" 4	1110	xxxx	0	x	0	?	0

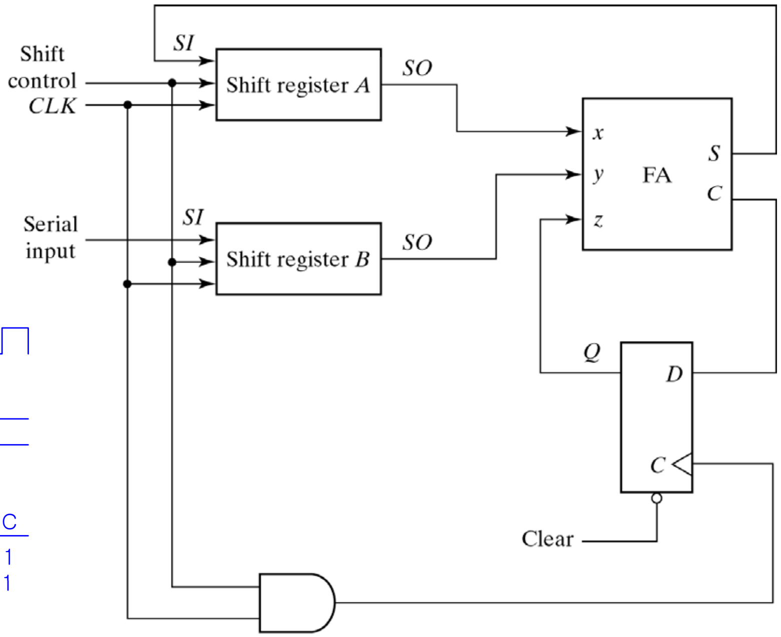


Fig. 6-5 Serial Adder

Parallel adder vs. serial adder

	parallel	serial
register	0 (or reg w/ p-load)	shift reg w/p-load
# of full adder	n (for n-bits)	1
carry FF	0	1
circuits	combinational	sequential

* “reg w/ p-load” stands for “register with parallel load”

Design of serial adder using a state table

- assume that we have registers for augend and addend
- Naming:

SO from augend register	…..	x
“ addend “	…..	y
FF for storing carry	…..	Q
output; sum	…..	S

- Excitation table (If DFF is used, the result is Fig. 6-5)

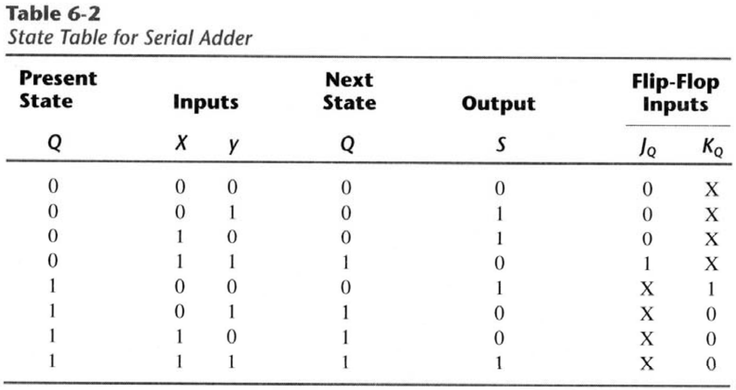


Table 6-2
State Table for Serial Adder

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

J_Q

xy

Q

K_Q

xy

Q

S

xy

Q

$J_Q = xy$

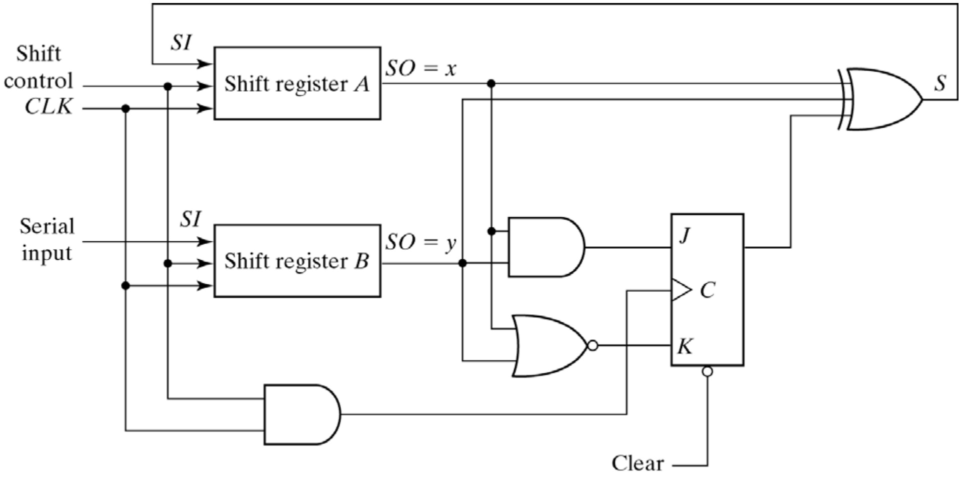


Fig. 6-6 Second form of Serial Adder

Universal Shift Register

Shift register with shift-left, shift-right, parallel load, parallel output capability and clear control

Operation mode selection

HDL coding HW

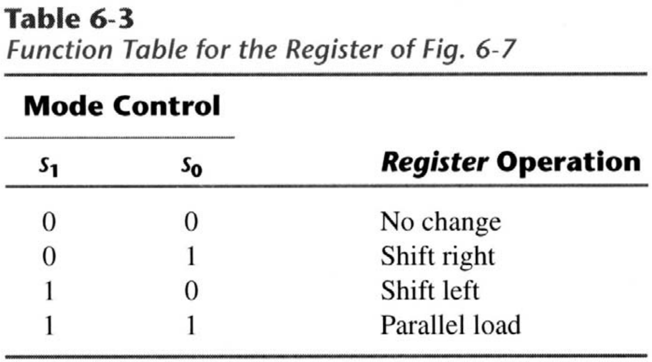


Table 6-3
Function Table for the Register of Fig. 6-7

Mode Control		
s_1	s_0	***Register* Operation**
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

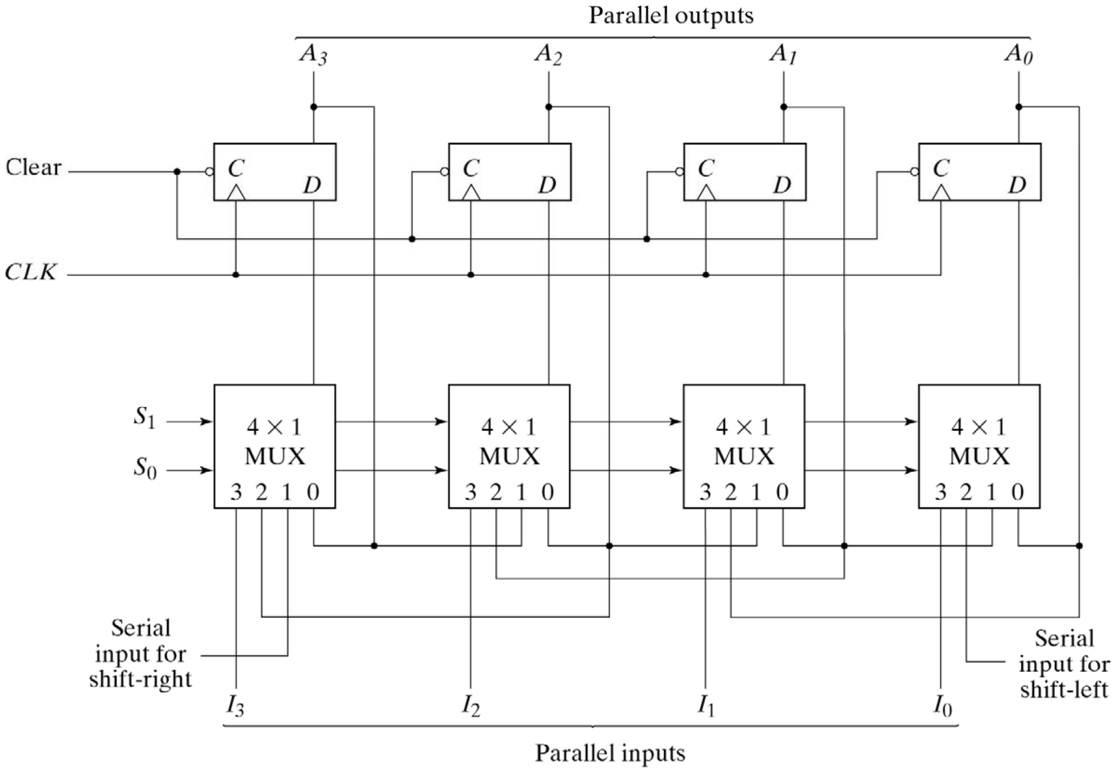


Fig. 6-7 4-Bit Universal Shift Register

6-3 Ripple Counter

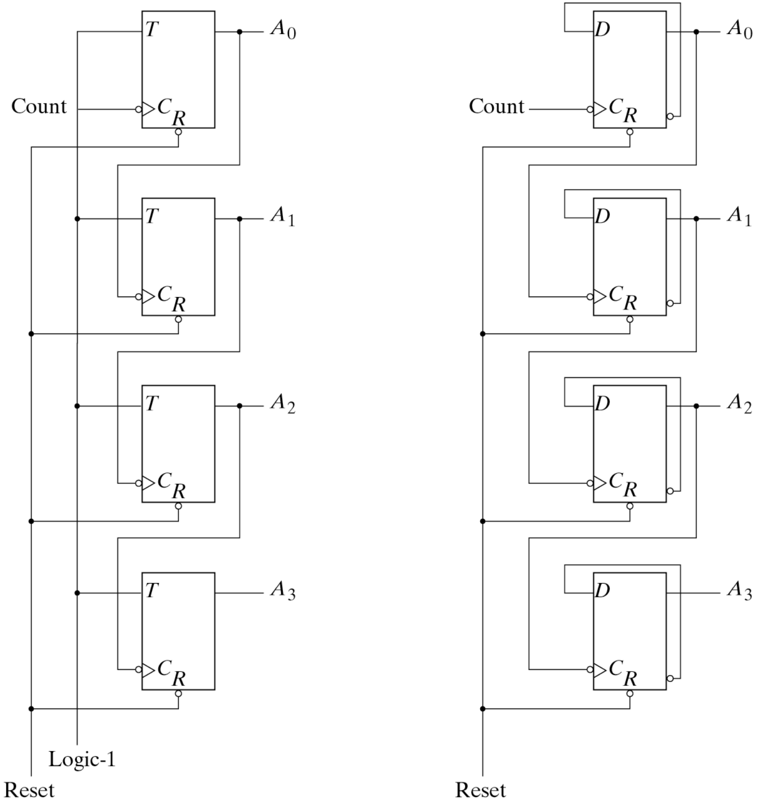
: register that goes through a prescribed sequence of states

ripple counter -- uses FF output as trigger signal for the next FF

synchronous counter

Binary ripple counter

Signal propagates in ripple fashion



(a) With T flip-flops

(b) With D flip-flops

Fig. 6-8 4-Bit Binary Ripple Counter

BCD ripple counter

decade counter

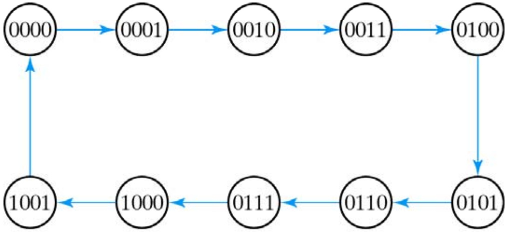


Fig. 6-9 State Diagram of a Decimal BCD-Counter

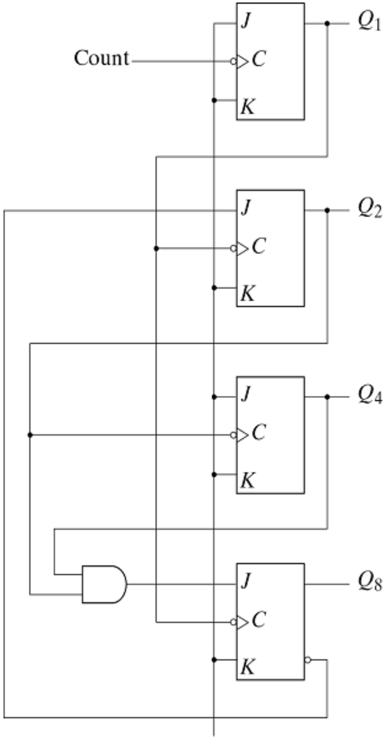
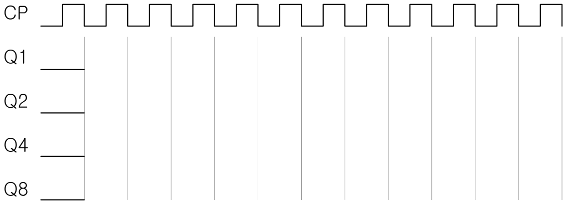


Fig. 6-10 BCD Ripple Counter

Timing diagram ?



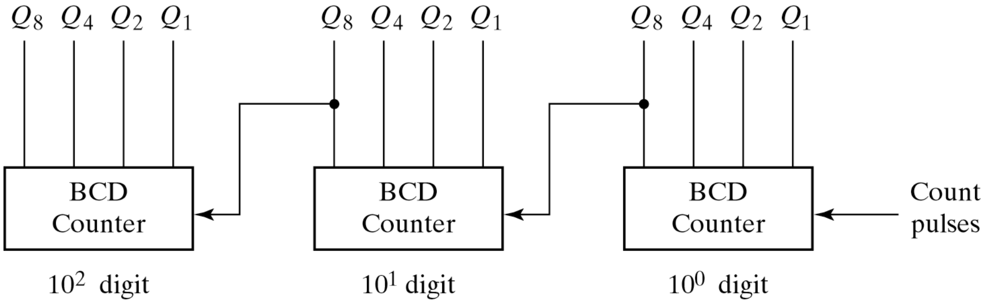
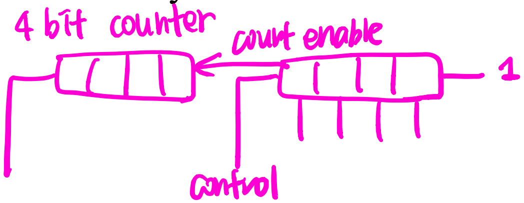


Fig. 6-11 Block Diagram of a Three-Decade Decimal BCD Counter

74160
161
163
164
168
169

6-4 Synchronous Counters



a common clock triggers all the flipflops

Binary counter

A4	A3	A2	A1
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
……….			
1	1	1	1
0	0	0	0

count up ↓

count down ↑

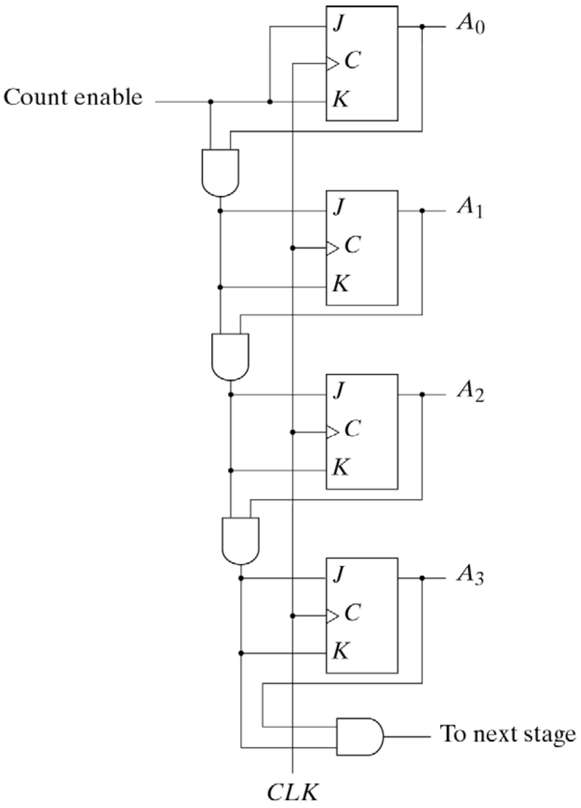


Fig. 6-12 4-Bit Synchronous Binary Counter

Binary counter

A4	A3	A2	A1
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	1	1	1
0	0	0	0

count up ↓

count down ↑

Up counting

LSB toggles with every pulse

The others toggle when all the bits in the lower order positions=‘1’

Down counting

LSB toggles with every pulse

The others toggle when all the bits in the lower order positions=‘0’

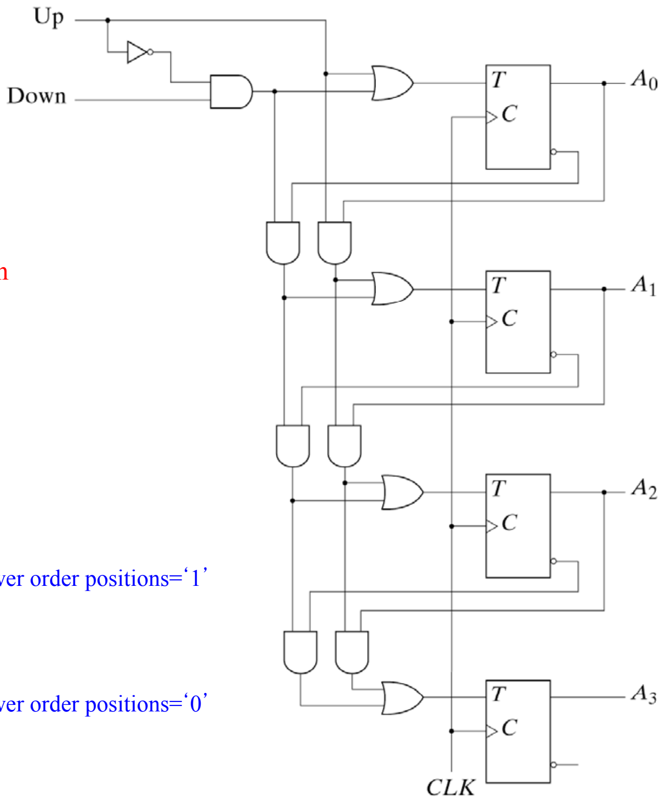


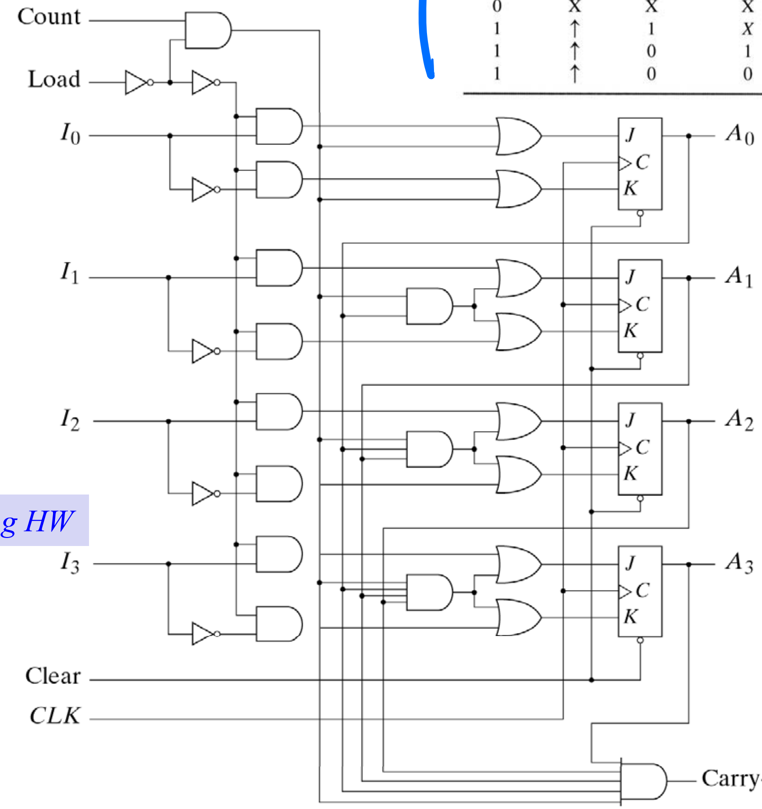
Fig. 6-13 4-Bit Up-Down Binary Counter

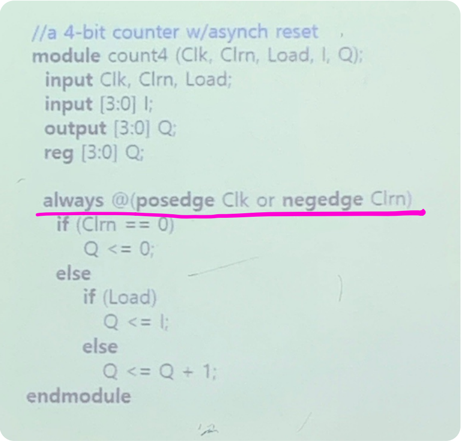
코딩 짜고
표만들기 or 표 짜고
코딩

positive edge triggred 4-bits counter
with asynchronous reset
and synchro load
with count controlled
input

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





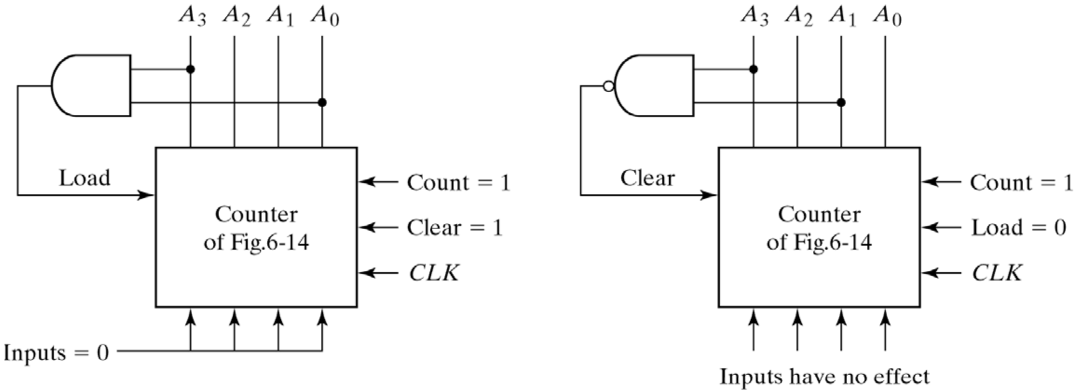
```
//a 4-bit counter w/asynch reset
module count4 (Clk, Clrn, Load, I, Q);
  input Clk, Clrn, Load;
  input [3:0] I;
  output [3:0] Q;
  reg [3:0] Q;

  always @(posedge Clk or negedge Clrn)
    if (Clrn == 0)
        Q <= 0;
    else
        if (Load)
          Q <= I;
        else
          Q <= Q + 1;
endmodule
```

HDL coding HW

Fig. 6-14 4-Bit Binary Counter with Parallel Load

BCD counters with the one in Fig. 6-14

- (b) generates a momentary spike as the count goes from 1001 to 0000
- Not recommended.



(a) Using the load input

(b) Using the clear input

Fig. 6-15 Two ways to Achieve a BCD Counter Using a Counter with Parallel Load

6-5 Other Counters

Modulo-N counter (divide-by-N counter)

a counter that goes through a repeated sequence of N states

Applications

- counting
- used to generate timing signals for digital systems

Counter with unused states

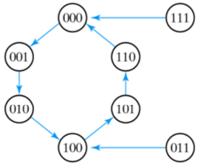
- may treat them as "*don't care*" or assign specific next states
- need to check whether it *eventually goes into one of the valid states*

Design Example (counter with unused states)

Table 6-7
State Table for Counter

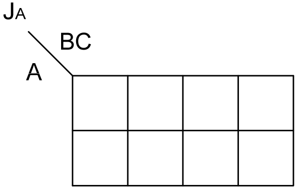
Present State			Next State			Flip-Flop Inputs					
A	*B*	*C*	*A*	*B*	*C*	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	1	0	0	1	X	X	1	0	X
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	0	0	0	X	1	X	1	0	X

HW 3, 2, 5, 1, 0, 7, 3, …의 순서를 반복하는 counter를 설계하라

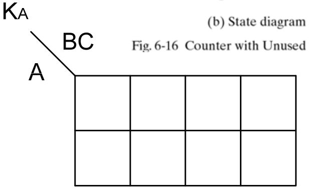


(b) State diagram

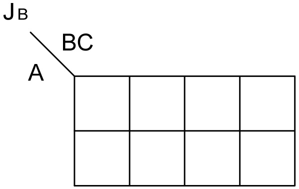
Fig. 6-16 Counter with Unused States



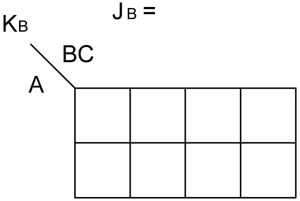
J_A =



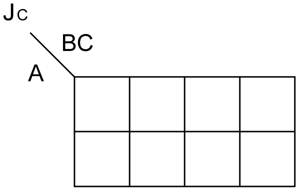
K_A =



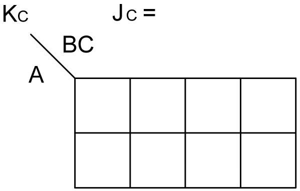
J_B =



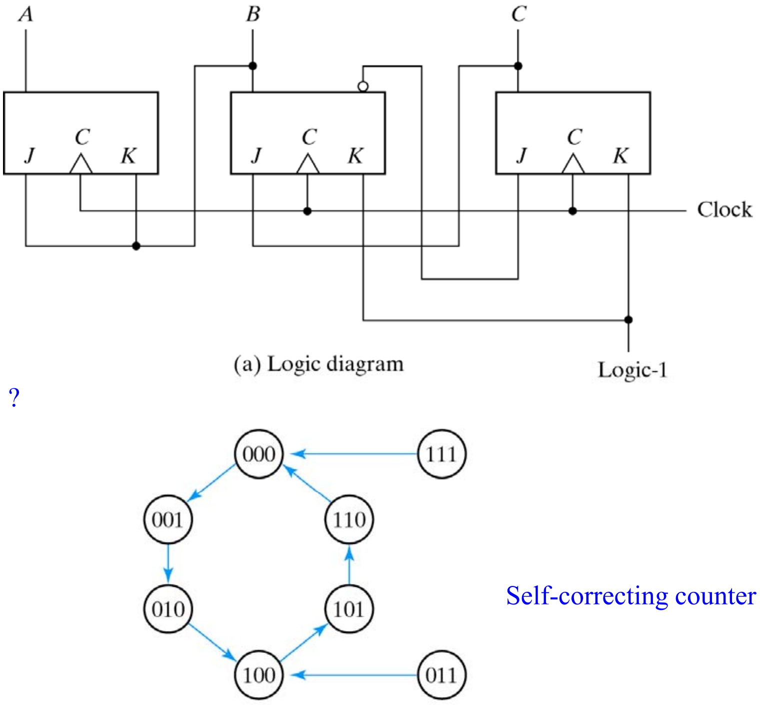
K_B =



J_C =



K_C =



(a) Logic diagram

Next states for unused states ?

(b) State diagram

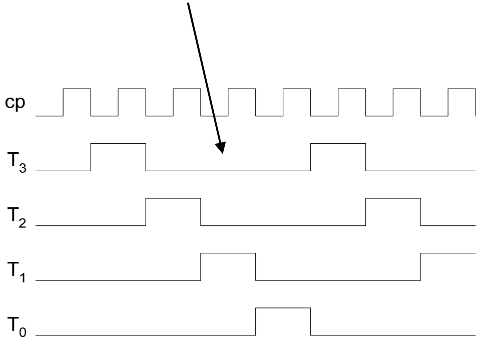
Self-correcting counter

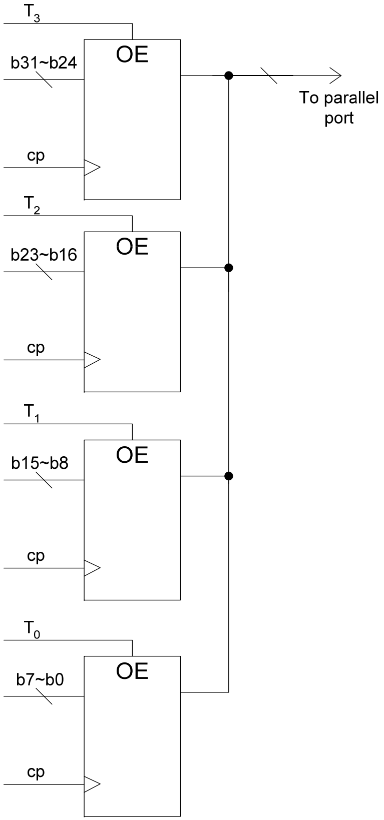
Fig. 6-16 Counter with Unused States

Timing signals

How to transfer 32 bits data
over 8 bits bus ?

Requires timing signals that make one enable at a time





Tristate buffer

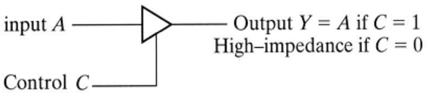
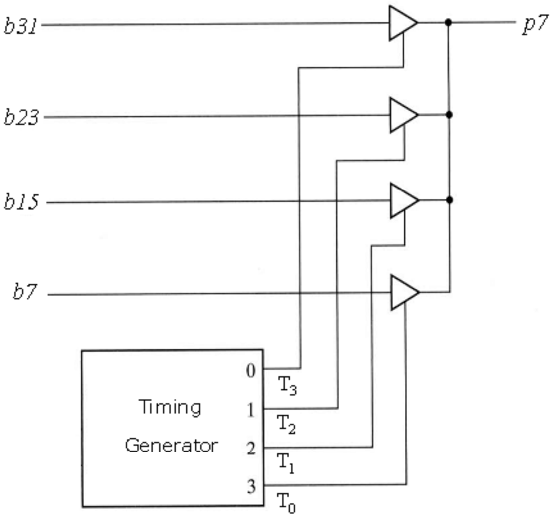


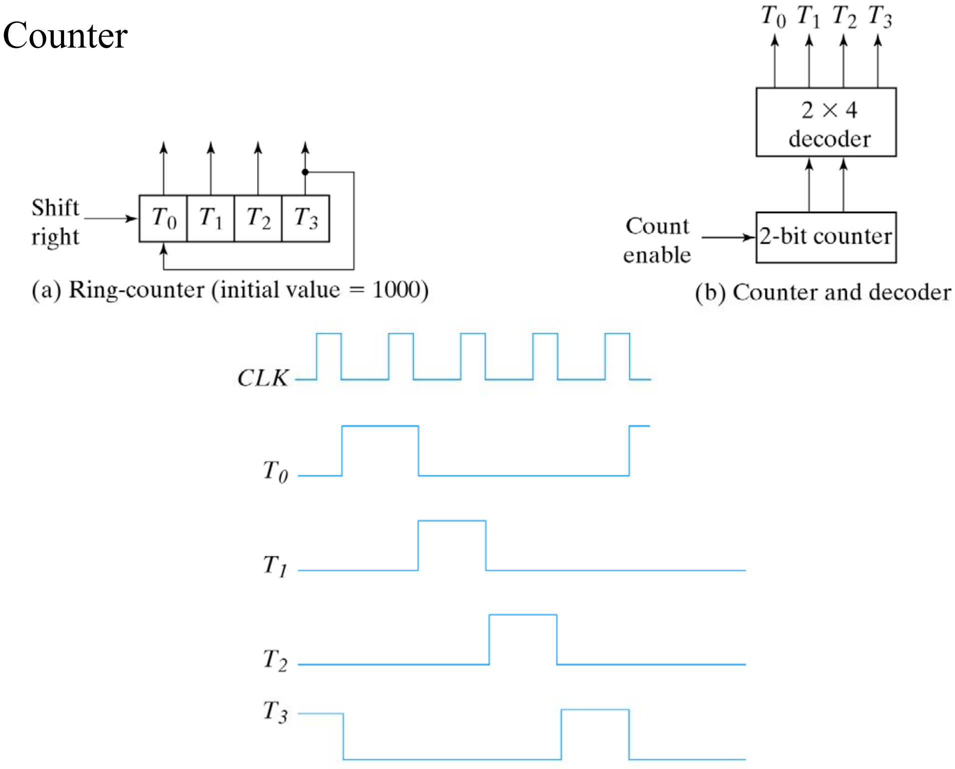
FIGURE 4-29 a Three-State Buffer



HW: Tristate buffer 대신 OR 또는 AND gate 를 사용하려면?

Multiplexers with Three-State Gates

Ring Counter



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

(b) Counter and decoder

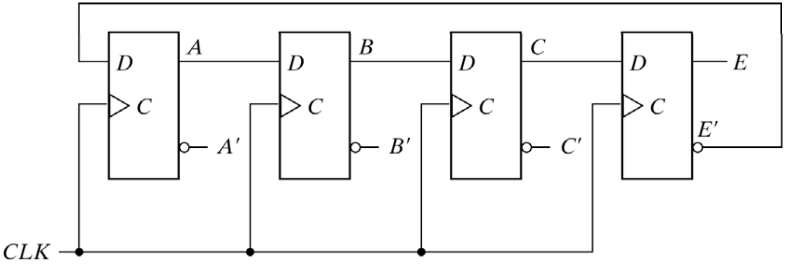
(c) Sequence of four timing signals

Fig. 6-17 Generation of Timing Signals

Johnson Counter

; useful in generating timing signals (2n distinguishable states with n FF's)

switch-tail ring counter



(a) Four-stage switch-tail ring counter

Disadvantage?
How to solve?
read textbook

Sequence number	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	$C'E$

Flip-flop outputs: A, B, C, E

(b) Count sequence and required decoding

Fig. 6-18 Construction of a Johnson Counter

Comparison

Assume that 2^n timing signals

1. Ring counter

   2^n FF's

2. Counter and decoder

   n bit counter and n-to2^n decoder (2^n n-input AND gates)

3. Johnson counter and decoder

   $2^{(n-1)}$ FF's + 2^n 2-input AND gates

   disadvantage : once it gets into an unused state, it will stay out of used states

   How to solve? $D_C=(A+C)B$

HW

CLK	Clr	S1	S0
		0	0
		0	1
		1	0
		1	1

Function Table 만들기.

+ Quartus waveform

4 4 4 4

6-6 HDL codes

```
//HDL Example 6-1
//---------------------
//Behavioral description of Universal shift register
// Fig. 6-7 and Table 6-3
module shftreg (s1,s0,Pin,lfin,rtin,A,CLK,Clr);
   input s1,s0;                  //Select inputs
   input lfin, rtin;             //Serial inputs
   input CLK,Clr;                   //Clock and Clear
   input [3:0] Pin;               //Parallel input
   output [3:0] A;                //Register output
   reg [3:0] A;
   always @ (posedge CLK or negedge Clr)
     if (~Clr) A = 4'b0000;
      else
        case ({s1,s0})
         2'b00: A = A;             //No change
         2'b01: A = {rtin,A[3:1]};  //Shift right
         2'b10: A = {A[2:0],lfin};  //Shift left
         2'b11: A = Pin;           //Parallel load input
        endcase
endmodule
```

이거 빼면 동기식 Clr

```
//HDL Example 6-3
//-------------------
//Binary counter with parallel load
//See Figure 6-14 and Table 6-6
module counter (Count,Load,IN,CLK,Clr,A,CO);
  input Count,Load,CLK,Clr;
  input [3:0] IN;                //Data input
  output CO;                     //Output carry
  output [3:0] A;                 //Data output
  reg [3:0] A;
  assign CO = Count & ~Load & (A == 4'b1111);
  always @ (posedge CLK or negedge Clr)
    if (~Clr) A = 4'b0000;
    else if (Load)  A = IN;
    else if (Count) A = A + 1'b1;
    else A = A;                // no change, default condition
endmodule
```