Modeling Sequential Elements with Verilog

Prof. Chien-Nan Liu
TEL: 03-4227151 ext:34534
Email: jimmy@ee.ncu.edu.tw

Sequential Circuit

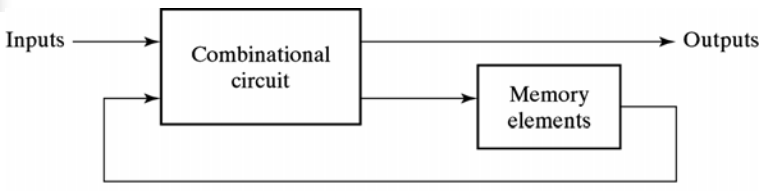


Fig. 5-1 Block Diagram of Sequential Circuit

- Outputs are functions of inputs and present states of storage elements
- Two types of sequential circuits
  - Synchronous (preferred !!)
  - Asynchronous

Synchronous vs. Asynchronous

- Avoid asynchronous and multi-cycle paths
- Register-based synchronous design is preferred
  - Accelerate synthesis and simulation
  - Ease static timing analysis
  - Use single (positive) edge triggered flip-flop
- Avoid to use latch as possible

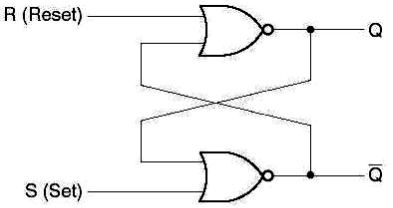
4-3

Memory Elements

- Allow sequential logic design
- Latch — a level-sensitive memory element
  - SR latches
  - D latches
- Flip-Flop — an edge-triggered memory element
  - Master-slave flip-flop
  - Edge-triggered flip-flop
- RAM and ROM — a mass memory element

4-4

SR Latch



(a) Logic diagram

S	R	Q	$\overline{Q}$	
1	0	1	0	Set state
0	0	1	0	
0	1	0	1	
0	0	0	1	Reset state
1	1	0	0	Undefined

(b) Function table

Fig. 4-4 *SR* Latch with NOR Gates

Should be very careful for this case

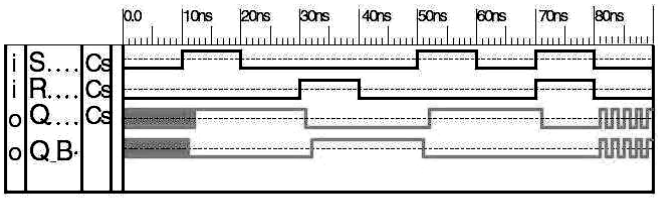
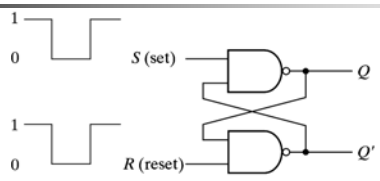


Fig. 4-5 Logic Simulation of *SR* Latch Behavior

4-5

Other SR Latches

S and R are active-low

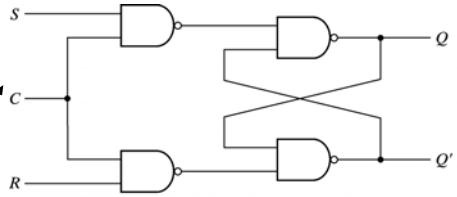


S	R	Q	Q′	
1	0	0	1	
1	1	0	1	(after $S = 1, R = 0$)
0	1	1	0	
1	1	1	0	(after $S = 0, R = 1$)
0	0	1	1	

(a) Logic diagram (b) Function table

Fig. 5-4 *SR* Latch with NAND Gates

C=0: disable all actions



C	S	R	Next state of Q
0	X	X	No change
1	0	0	No change
1	0	1	$Q = 0$; Reset state
1	1	0	$Q = 1$; set state
1	1	1	Indeterminate

(a) Logic diagram (b) Function table

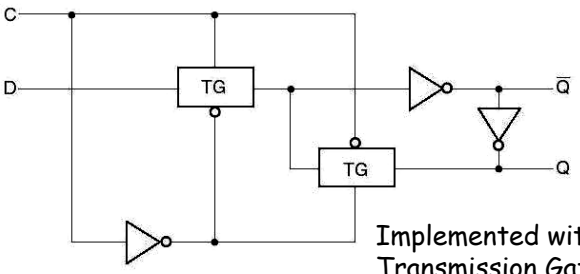
Fig. 5-5 SR Latch with Control Input

4-6

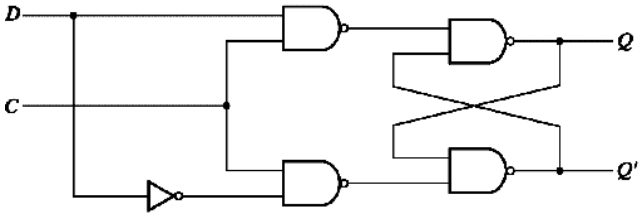
D Latch with Control

C	D	Next state of Q
0	X	No change
1	0	Q = 0; Reset state
1	1	Q = 1; Set state

Function Table



Implemented with Transmission Gates

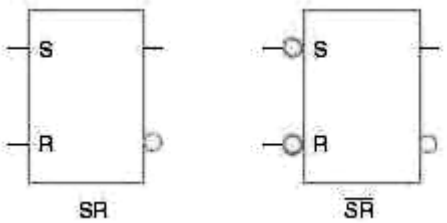


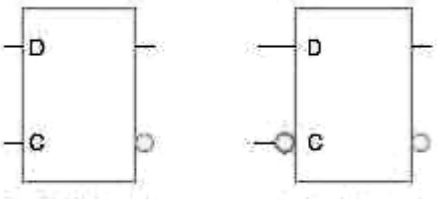
(a) Logic diagram

Logic Diagram

4-7

Symbols for Latches





4-8

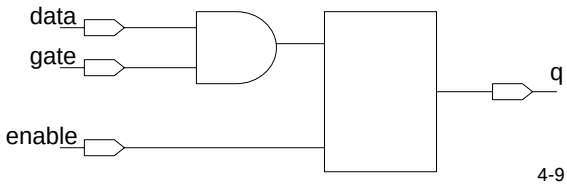
Latch Inference

- Incompletely specified wire in the synchronous section
- D latch

```
always @(enable or data)
  if (enable)
     q = data;
```

- D latch with gated asynchronous data

```
always @(enable or data or gate)
  if (enable)
     q = data & gate;
```

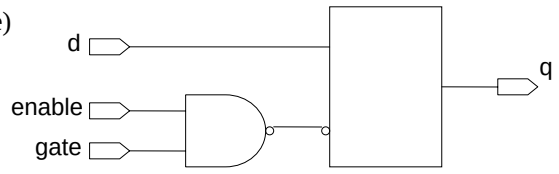


4-9

More Latches

- D latch with gated “enable”

```
always @(enable or d or gate)
  if (enable & gate)
     q = d;
```



- D latch with asynchronous reset

```
always @(reset or data or gate)
  if (reset)
     q = 1'b0;
  else if (enable)
     q = data;
```

4-10

Latch with Preset/Clear (1/2)

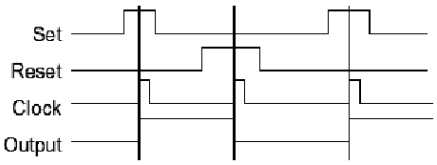
```
module LATCH_ASYNC_P_C (En1, Clear1, X1, En2, Preset2, X2, En3,
                        Preset3, Clear3, X3, Y1,Y2, Y3);
  input  En1, Clear1, X1, En2, Preset2, X2, En3, Preset3, Clear3, X3;
  output  Y1, Y2, Y3;
  reg  Y1, Y2, Y3;

  always @(En1 or Clear1 or X1 or En2 or Preset2 or X2 or En3 or
       Preset3 or Clear3 or X3)
  begin

    if (! Clear1)
       Y1 = 0;
    else if (En1)
       Y1 = X1;
```


Latch with Preset/Clear (2/2)

```
    if (Preset2)
       Y2 = 1;
    else if (En2)
       Y2 = X2;

    if (Clear3)
       Y3 = 0;
    else if (Preset3)
       Y3 = 1;
    else if (En3)
       Y3 = X3;

  end

endmodule
```

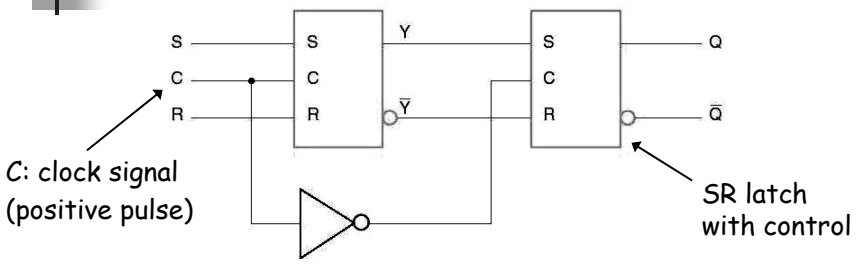
Latch vs. Flip-Flop

- Latch:
  - Change stored value under specific status of the control signals
  - Transparent for input signals when control signal is “on”
  - May cause combinational feedback loop and extra changes at the output
- Flip-Flop
  - Can only change stored value by a momentary switch in value of the control signals
  - Cannot “see” the change of its output in the same clock pulse
  - Encounter fewer problems than using latches



4-13

Master-Slave SR Flip-Flop



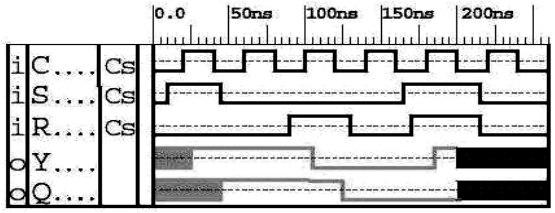
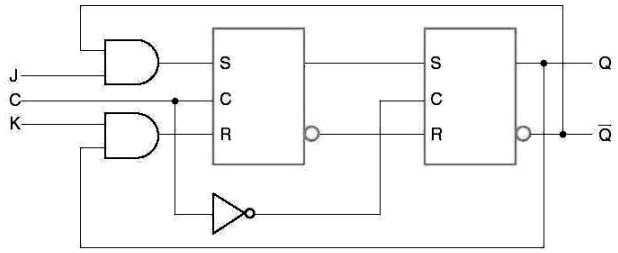


Fig. 4-11 Logic Simulation of a Master-Slave Flip-Flop

4-14

Master-Slave JK Flip-Flop



(a)

Reset

Set

J	K	Next State of Q
0	0	Q
0	1	0
1	0	1
1	1	$\overline{Q}$

Toggle

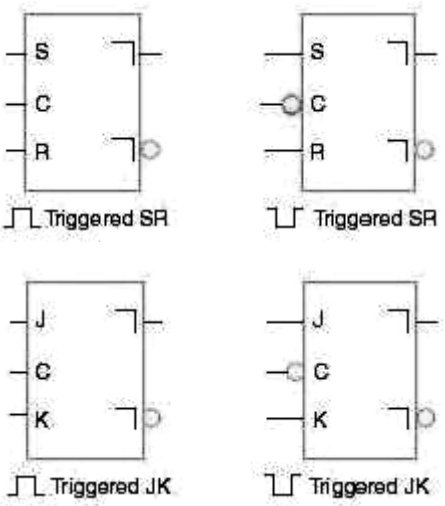
(b)

Solve the undefined problem in SR FF

Fig. 4-12 Master-Slave *JK* Flip-Flop

4-15

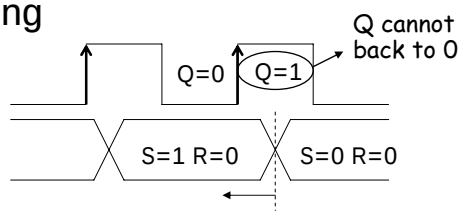
Symbols for Master-Slave F/F



4-16

Problems of Master-Slave F/F

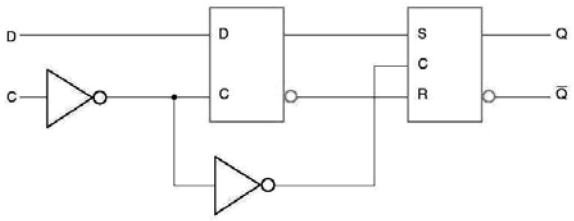
- Master-slave flip-flops are also referred as **pulse-triggered** flip-flops
- May cause errors when the delay of combinational feedback path is too long



- To solve:
  - Ensure the delay of combinational block is short enough
  - Use **edge-triggered** flip-flops instead

4-17

Positive-Edge-Triggered D F/F



- The master latch is a D latch
  - Transparent to input during C=0 without other control signals
  - The input value of the slave latch is decided just at the end of the period “C=0”
  - The delayed changes can still be stored if they can arrive before C has a 0->1 transition
- Solve the problems of master-slave flip-flops

4-18

Positive-Edge-Triggered D F/F

- If only SR latches are available, three latches are required
- Two latches are used for locking the two inputs (CLK & D)
- The final latch provides the output of the flip-flop

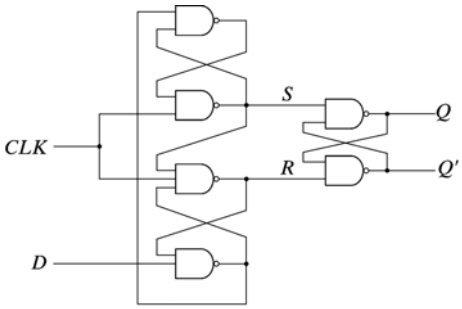
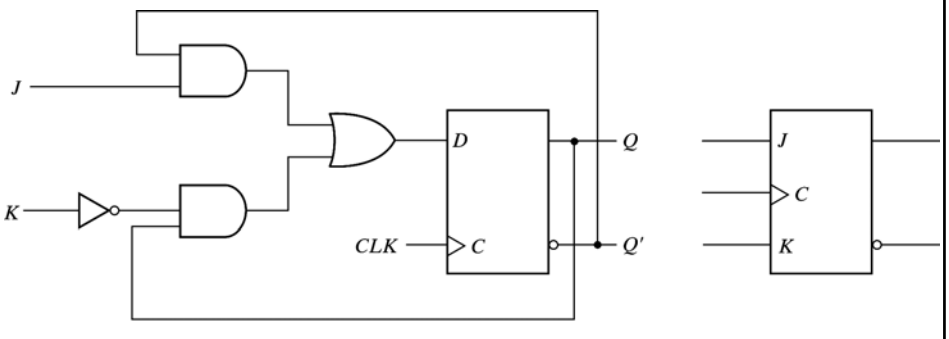


Fig. 5-10 *D*-Type Positive-Edge-Triggered Flip-Flop

4-19

Positive-Edge-Triggered JK F/F

$$D = JQ'+K'Q$$

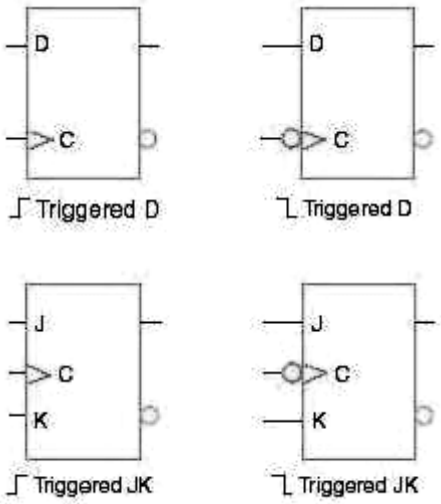


(a) Circuit diagram (b) Graphic symbol

Fig. 5-12 *JK* Flip-Flop

4-20

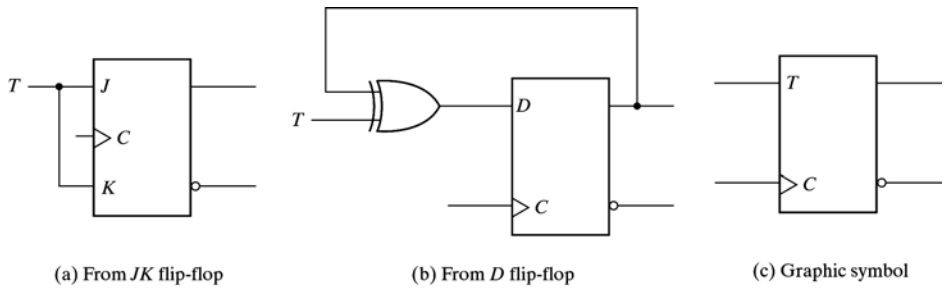
Symbols for Edge-Triggered FF



4-21

Positive-Edge-Triggered T F/F

$$D = T \oplus Q = TQ' + T'Q$$



(a) From *JK* flip-flop (b) From *D* flip-flop (c) Graphic symbol

Fig. 5-13 T Flip-Flop

4-22

Characteristic Tables

- Define the logical properties in tabular form

TABLE 4-1
Flip-Flop Characteristic Tables

(a) *JK* Flip-Flop				(b) *SR* Flip-Flop			
J	K	$Q(t+1)$	Operation	S	R	$Q(t+1)$	Operation
0	0	$Q(t)$	No change	0	0	$Q(t)$	No change
0	1	0	Reset	0	1	0	Reset
1	0	1	Set	1	0	1	Set
1	1	$\overline{Q}(t)$	Complement	1	1	?	Undefined

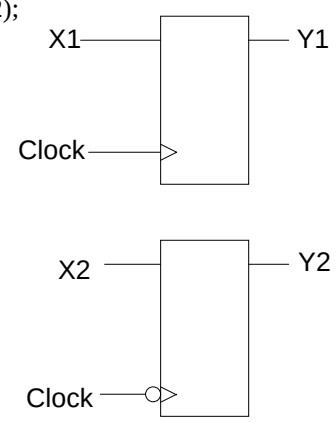
(c) *D* Flip-Flop			(d) *T* Flip-Flop		
D	$Q(t+1)$	Operation	T	$Q(t+1)$	Operation
0	0	Reset	0	$Q(t)$	No change
1	1	Set	1	$\overline{Q}(t)$	Complement

4-23

Flip-Flop Inference

- Wire (port) assigned in the synchronous section

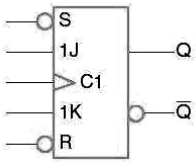
```
module FF_PN (Clock, X1, X2, Y1, Y2);
   input  Clock;
   input  X1, X2;
   output  Y1, Y2;
   reg  Y1, Y2;

  always @(posedge Clock)
    Y1 = X1;

  always @(negedge Clock)
    Y2 = X2;

endmodule
```



4-24

Direct Inputs

- Directly set or reset a flip-flop immediately
  - Independent of clock signal
- Often called *direct set (preset)* and *directly reset (clear)*
- Also called *asynchronous* set/reset



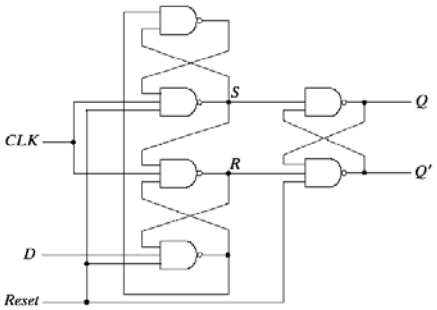
(a) Graphic symbols

S	R	C	J	K	Q	$\overline{Q}$
0	1	X	X	X	1	0
1	0	X	X	X	0	1
0	0	X	X	X	Undefined	
1	1	↑	0	0	No change	
1	1	↑	0	1	0	1
1	1	↑	1	0	1	0
1	1	↑	1	1	Complement	

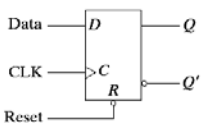
(b) Function table

Fig. 4-16 *JK* Flip-Flop with Direct Set and Reset

D F/F with Asynchronous Reset



(a) Circuit diagram



(b) Graphic symbol

R	C	D	Q	Q′
0	X	X	0	1
1	↑	0	0	1
1	↑	1	1	0

(b) Function table

Reset (1/2)

- Synchronous reset
  - Easy to synthesize
  - Requires a free-running clock, especially at power-up phase
  - Hard to deal with tri-state bus initialization at power on

4-27

Reset (2/2)

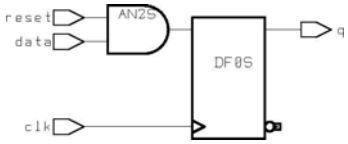
- Asynchronous reset
  - Do not require a free-running clock
  - Hard to implement the huge reset network
    - Clock tree synthesis (CTS) technique may also required for the reset signal
  - Synchronous de-assertion problem
  - Make STA and cycle-based simulation more difficult
- Asynchronous reset is more popular so far

4-28

Modeling Reset

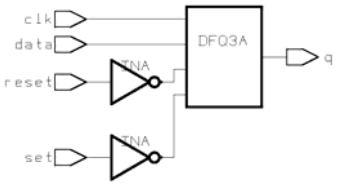
Synchronous reset

```
always@(posedge clk)
if (!reset) q=1'b0;
else q=data;
```



Asynchronous reset

```
always@(posedge clk or negedge reset )
  if (!reset) q=1'b0;
  else q=data;
```



4-29

Flip-Flop with Set/Reset (1/3)

- Asynchronous / synchronous reset / enable

```
module FFS (Clock, SReset, ASReset, En, Data1, Data2, Y1, Y2
 Y3, Y4, Y5, Y6);

   input  Clock, SReset, ASReset, En, Data1, Data2;
   output  Y1, Y2, Y3, Y4, Y5, Y6;
   reg  Y1, Y2, Y3, Y4, Y5, Y6;

 always @(posedge Clock)
  begin
   if (! SReset)   // Synchronous reset
     Y1 = 0;
   else
     Y1 = Data1 | Data2;
  end
```

4-30

Flip-Flop with Set/Reset (2/3)

```
// Negative active asynchronous reset
always @(posedge Clock or negedge ASReset)
  if (! ASReset)
    Y2 = 0;
  else
    Y2 = Data1 & Data2;

// One synchronous & one asynchronous reset
always @(posedge Clock or negedge ASReset)
  if (! ASReset)
    Y3 = 0;
  else if (SReset)
    Y3 = 0;
  else
    Y3 = Data1 | Data2;
```


Flip-Flop with Set/Reset (3/3)

```
// Single enable
always @(posedge Clock)
  if (En)
    Y4 = Data1 & Data2;

// Synchronous reset and enable
always @(posedge Clock)
  if (SReset)
    Y5 = 0;
  else if (En)
    Y5 = Data1 | Data2;

endmodule
```

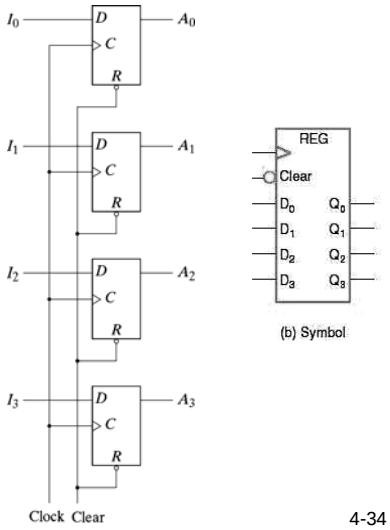
Registers and Counters

- Register:
  - A set of flip-flops, possibly with added combinational gates, that perform data-processing tasks
  - Store and manipulate information in a digital system
- Counter:
  - A register that goes through a predetermined sequence of states
    - A special type of register
  - Employed in circuits to sequence and control operations

4-33

The Simplest Register

- Consist of only flip-flops
- Triggered by common clock input
- The *Clear* input goes to the R (reset) input of all flip-flops
  - *Clear* = 0 → all flip-flops are reset ***asynchronously***
- The *Clear* input is useful for cleaning the registers to all 0's prior to its clocked operation
  - Must maintain at logic 1 during normal operations



4-34

Register with Parallel Load

- When *Load* = 1, all the bits of data inputs are transferred into the registers
  - Parallel loaded
- When *Load* =0, the outputs of the flip-flops are connected to their respective inputs
  - Keep no change

control data loading at the input side

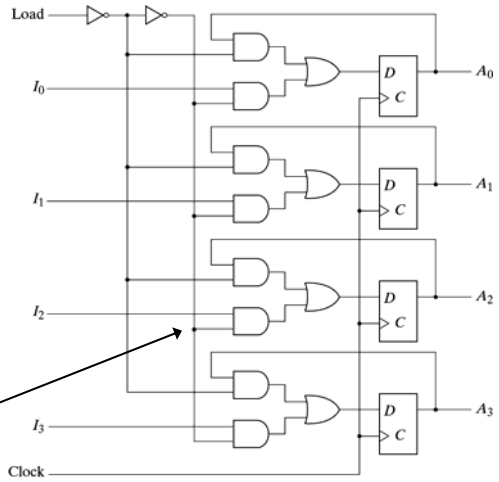


Fig. 6-2 4-Bit Register with Parallel Load

HDL Modeling for Registers

```
module Reg4 (CLK, CLR_N, Load, D, Q);
  input CLK, CLR_N, Load;
  input [3:0] D;
  output [3:0] Q;
  reg [3:0] Q;

  always @(posedge CLK) begin
    if (CLR_N == 0)
      Q <= 0;
    else if (Load == 1)
      Q <= D;
  end
endmodule
```

The Simplest Shift Register

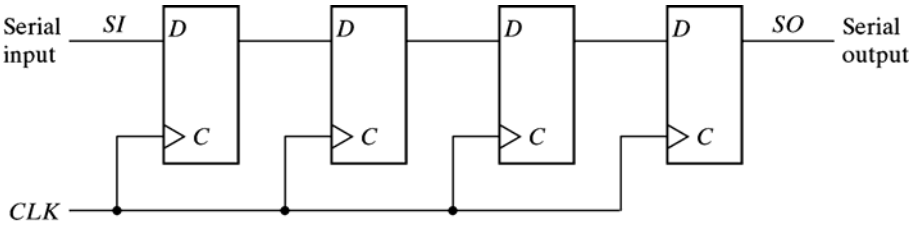
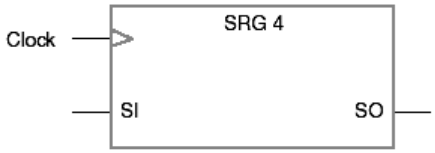


Fig. 6-3 4-Bit Shift Register



(b) Symbol

4-37

HDL Modeling for Shift Reg.

- Blocking assignment

```
assign SO = D;
always @(posedge CLK) begin
  if (reset) begin
    A = 0;  B = 0; C = 0; D = 0;
  end
  else if (shift) begin
    D = C;
    C = B;
    B = A;
    A = SI;
  end
end
```

order dependent

- Non-blocking assignment

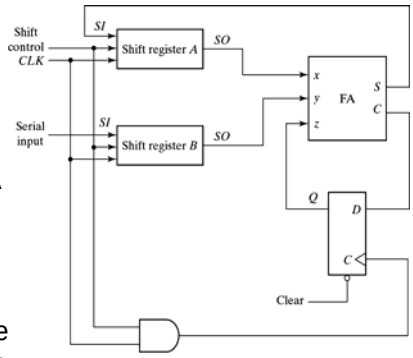
```
assign SO = D;
always @(posedge CLK) begin
  if (reset) begin
    A <= 0;  B <= 0; C <= 0; D <= 0;
  end
  else if (shift) begin
    A <= SI;
    C <= B;
    D <= C;
    B <= A;
  end
end
```

can be any order

4-38

Serial Addition

- The bits of two binary numbers are added ***one pair at a time*** through a single full adder (FA) circuit
- Initialization:
  - A = augend; B = addend
  - Carry flip-flop is cleared to 0
- For each clock pulse:
  - A new sum bit is transferred to A
  - A new carry is transferred to Q
  - Both registers are shifted right
- To add three or more numbers:
  - Shift in the next number from the serial input while B is shifted to the FA
  - A will accumulate their sum



Serial v.s. Parallel

- Serial adders:
  - Use shift registers
  - A sequential circuit
  - Require only one FA and a carry flip-flop
  - Slower but require less equipment
- Parallel adders:
  - Use registers with parallel load for sum
  - Basically a pure combinational circuit
  - ***n*** FAs are required
  - Faster

Shift Register with Parallel Load

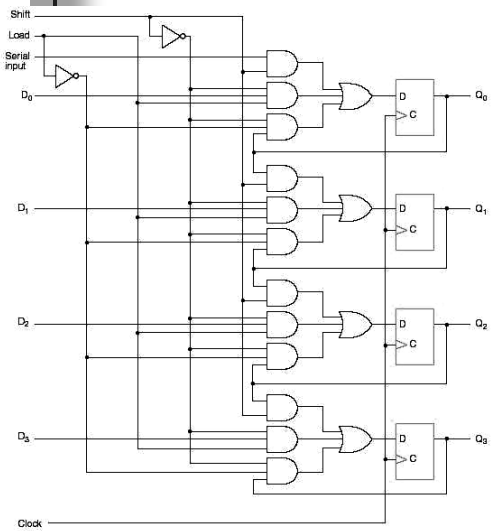
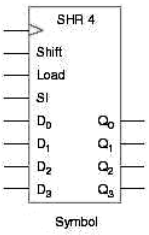


TABLE 5-2
Function Table for the Register of Figure 5-6

Shift	Load	Operation
0	0	No change
0	1	Load parallel data
1	×	Shift down from Q_0 to Q_3

Table 5-2 Function Table for the Register of Figure 5-6



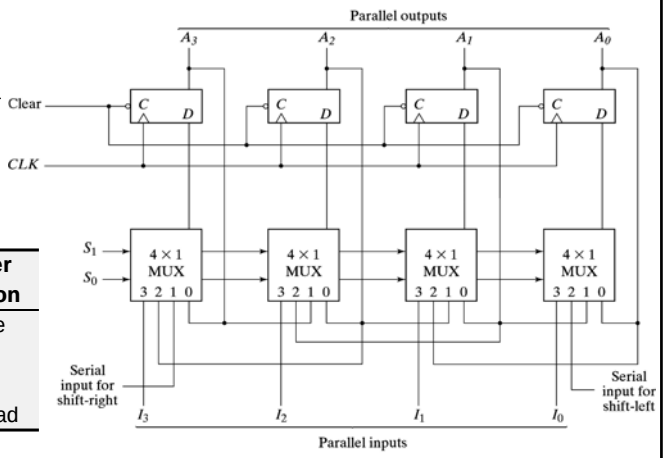
Modeling Shift Reg. with Load

```
always @(posedge CLK) begin
  if (Reset)
    Q <= 4`b0;
  else if (Shift)
    for (i=0; i<4; i=i+1)
      Q[i+1] <= Q[i];
  else if (Load)
    Q <= D;
end
```

should be careful about the priority

Universal Shift Register

clear all registers to zero



Mode Control		Register
S1	S0	Operation
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

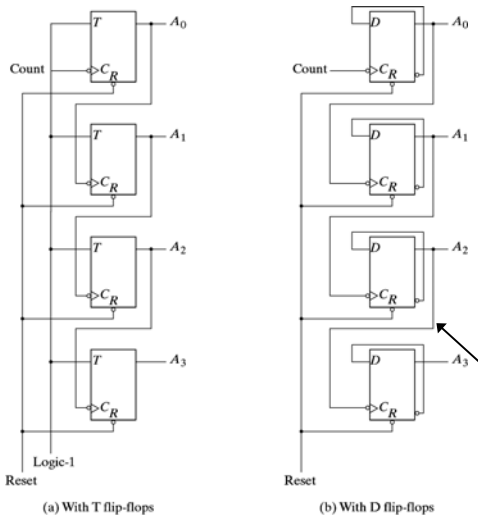
4-43

Counters

- A register that goes through a prescribed sequence of states
- Ripple counter
  - The flip-flop output transition serves as a source for triggering other flip-flops
  - Hardware is much simpler
  - Unreliable and delay dependent due to the asynchronous behaviors
- Synchronous counter
  - The common clock pulse triggers all flip-flops simultaneously
  - Hardware may be more complex but more reliable

4-44

Binary Ripple Counter



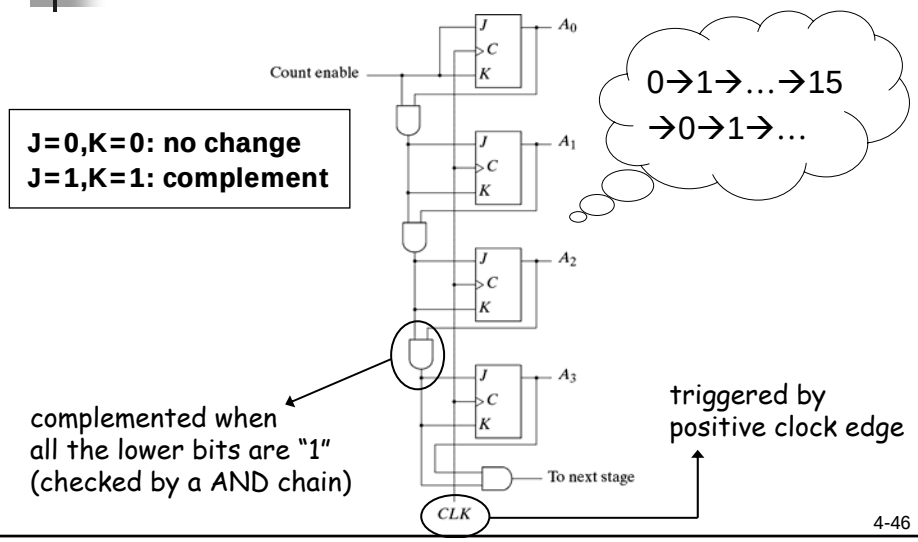
Binary Count Sequence

A3	A2	A1	A0
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

the output transition triggers the next flip-flop

4-45

Binary Synchronous Counter



J=0,K=0: no change
J=1,K=1: complement

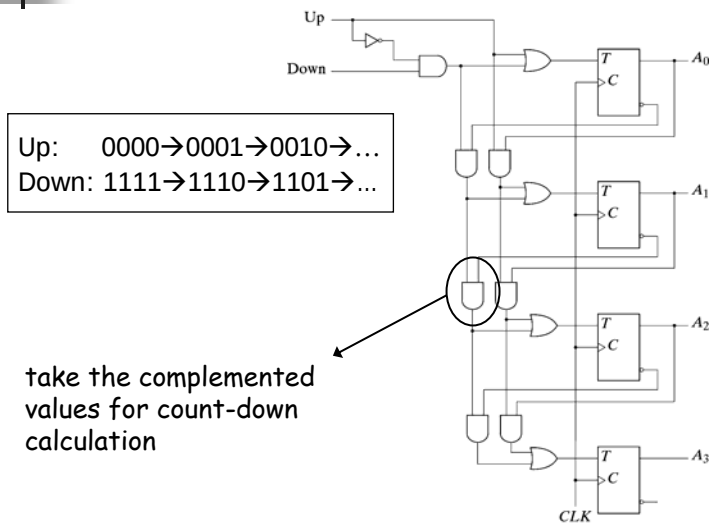
0→1→…→15
→0→1→…

complemented when all the lower bits are "1" (checked by a AND chain)

triggered by positive clock edge

4-46

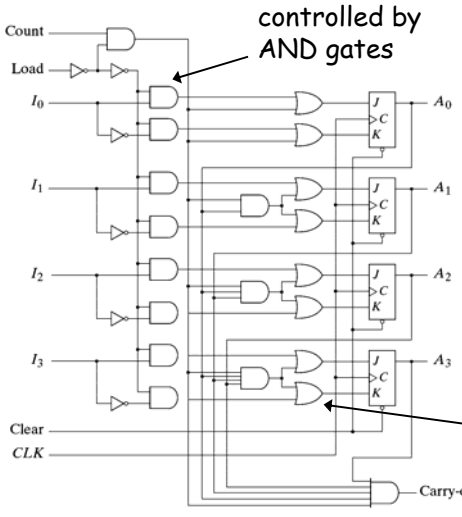
Up-Down Binary Counter



Up: 0000→0001→0010→…
Down: 1111→1110→1101→…

take the complemented values for count-down calculation

Binary Counter with Parallel Load



controlled by AND gates

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

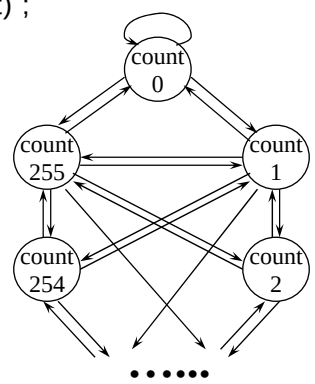
asynchronous input

merged by OR gates

Carry-output → set when count = 1111

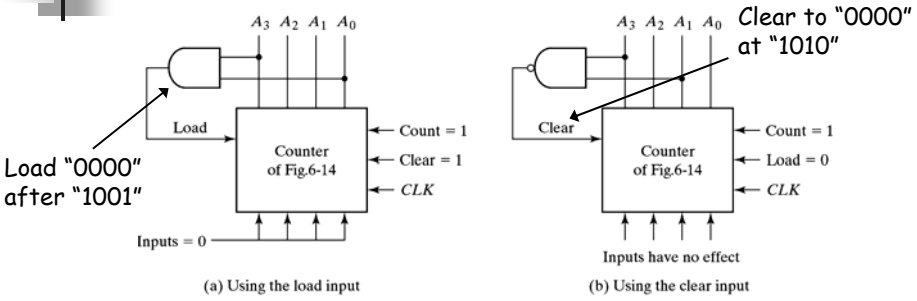
Modeling Sync. Counters

```
module counter (clk, reset, load, in, count) ;
input          clk, reset, load ;
input   [7:0] in ;
output [7:0] count ;
reg      [7:0] count ;

always @(posedge clk) begin
    if (reset) count = 0 ;
    else if (load) count = in ;
    else if (count == 255) count = 0 ;
    else count = count + 1 ;
end
endmodule
```



256 states 66047 transitions

BCD Counter



- Divide-by-N counter:
  - Go through a repeated sequence of N states
- BCD counter: only 0 to 9

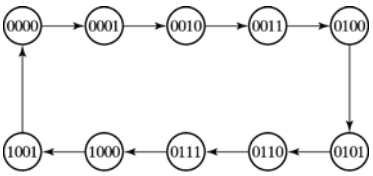


Fig. 6-9 State Diagram of a Decimal BCD-Counter

Modeling BCD Counters

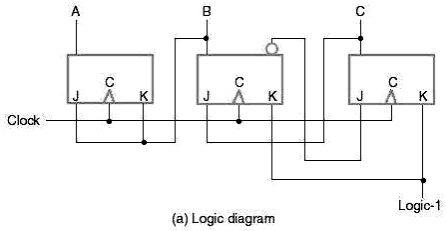
```
module BCDcounter (clk, reset, count) ;
input          clk, reset ;
output [3:0] count ;
reg      [3:0] count ;

always @(posedge clk) begin
    if (reset) count = 0 ;
    else if (count == 9) count = 0 ;
    else count = count + 1 ;
end
endmodule
```


Arbitrary Count Sequence

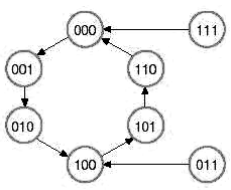
TABLE 5-8
State Table and Flip-Flop Inputs for Counter

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	×	0	×	1	×
0	0	1	0	1	0	0	×	1	×	×	1
0	1	0	1	0	0	1	×	×	1	0	×
1	0	0	1	0	1	×	0	0	×	1	×
1	0	1	1	1	0	×	0	1	×	×	1
1	1	0	0	0	0	×	1	×	1	0	×

Design it as a FSM



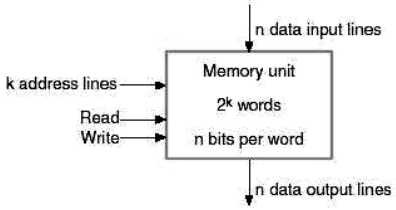
(a) Logic diagram



(b) State diagram

Mass Memory Elements

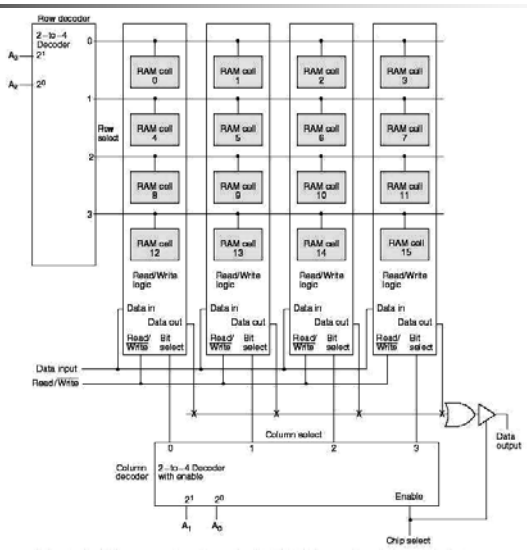
- Memory is a collection of binary cells together with associated circuits needed to transfer information to or from any desired location



- Two primary categories of memory:
  - Random access memory (RAM)
  - Read only memory (ROM)

4-53

RAM Cell Array



4-54

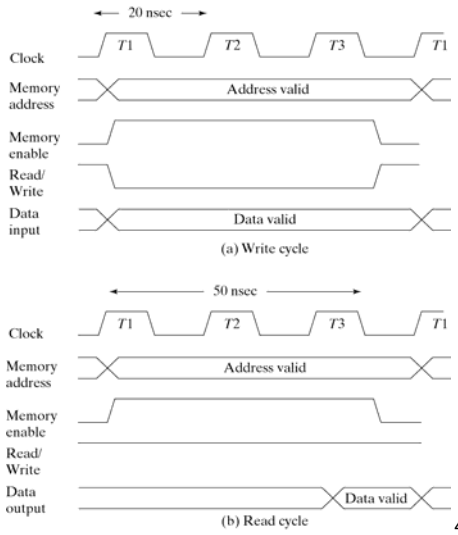
Write and Read Operations

- Write to RAM
  - Apply the binary address of the desired word to the ***address lines***
  - Apply the data bits that must be stored in memory to the ***data input lines***
  - Activate the ***write control***
- Read from RAM
  - Apply the binary address of the desired word to the ***address lines***
  - Activate the ***read control***

4-55

Timing Waveforms

- CPU clock = 50 MHz
  - cycle time = 20 ns
- Memory access time = 50 ns
  - The time required to complete a read or write operation
- The control signals must stay active for at least 50 ns
  - 3 CPU cycles are required



4-56

Coincident Decoding

- Address decoders are often divided into two parts
  - A two-dimensional scheme
- The total number of gates in decoders can be reduced
- Can arrange the memory cells to a square shape
- EX: 10-bit address
  404 = 0110010100
  X = 01100
  Y = 10100

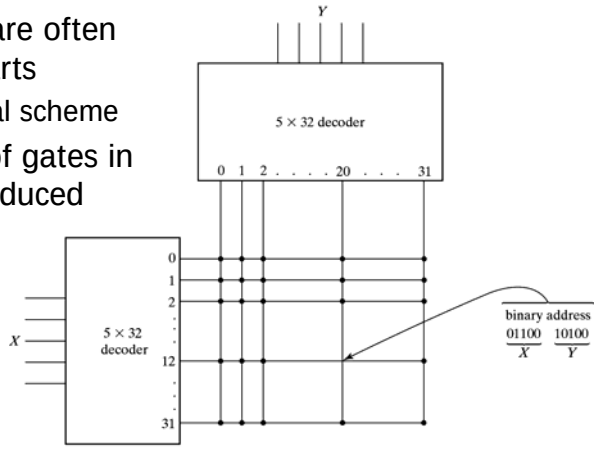


Fig. 7-7 Two-Dimensional Decoding Structure for a 1K-Word Memory

4-57

Address Multiplexing

- Memory address lines often occupy too much I/O pads
  - 64K = 16 lines
  - 256M = 28 lines
- Share the address lines of X and Y domains
  - Reduce the number of lines to a half
  - An extra register is required for both domain to store the address

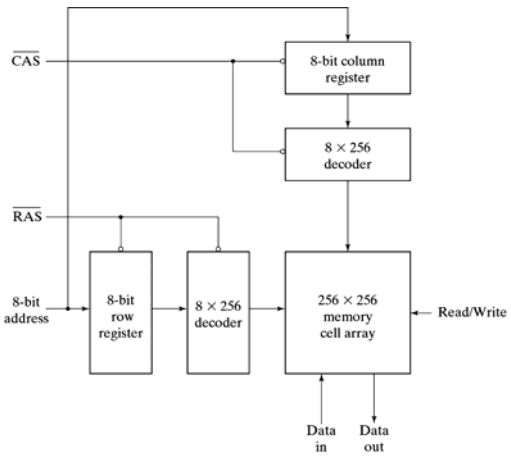


Fig. 7-8 Address Multiplexing for a 64K DRAM

4-58

SRAM vs. DRAM

- Static RAM:
  - Use internal latch to store the binary info.
  - Stored information remains valid as long as power is on
  - Shorter read and write cycles
  - Larger cell area and power consumption
- Dynamic RAM:
  - Use a capacitor to store the binary info.
  - Need periodically refreshing to hold the stored info.
  - Longer read and write cycles
  - Smaller cell area and power consumption

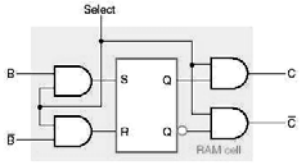


Fig. 6-5 Static RAM Cell

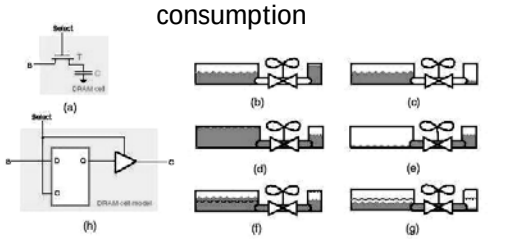


Fig. 6-12 Dynamic RAM Cell, Hydraulic Analogy of Cell Operation, and Cell Model

Read Only Memory (1/2)

ROM Truth Table (Partial) — **Table 7-3**

Inputs					Outputs							
I_4	I_3	I_2	I_1	I_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
		.							.			
		.							.			
		.							.			
1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1

4-60

Read Only Memory (2/2)

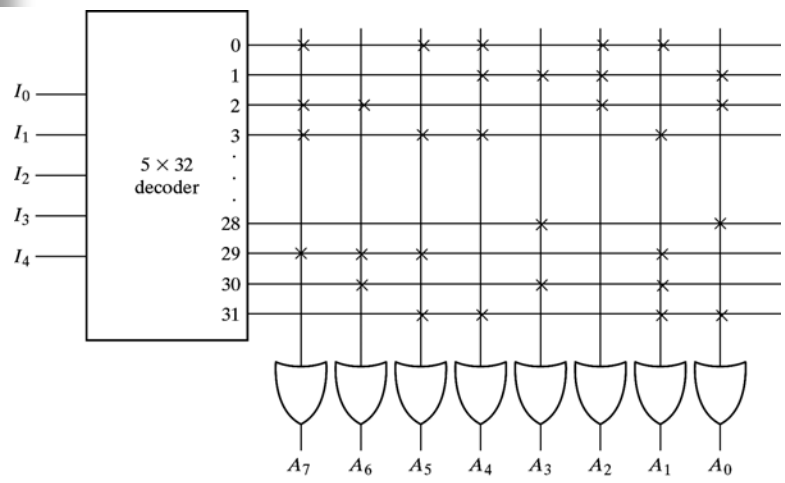


Fig. 7-11 Programming the ROM According to Table 7-3

4-61

Types of ROMs

- Mask programming
  - Program the ROM in the semiconductor factory
  - Economic for large quantity of the same ROM
- Programmable ROM (PROM)
  - Contain all fuses at the factory
  - Program the ROM by burning out the undesired fuses (irreversible process)
- Erasable PROM (EPROM)
  - Can be restructured to the initial state under a special ultra-violet light for a given period of time
- Electrically erasable PROM (EEPROM or E^2PROM)
  - Like the EPROM except being erased with electrical signals

4-62

Programmable Logic Devices

- ROM provides full decoding of variables
  - Waste hardware if the functions are given
- For known combinational functions, Programmable Logic Devices (PLD) are often used
  - Programmable read-only memory (PROM)
  - Programmable array logic (PAL)
  - Programmable logic array (PLA)
- For sequential functions, we can use
  - Sequential programmable logic device (SPLD)
  - Complex programmable logic device (CPLD)
  - Field programmable gate array (FPGA) ← most popular

Configurations of Three PLDs

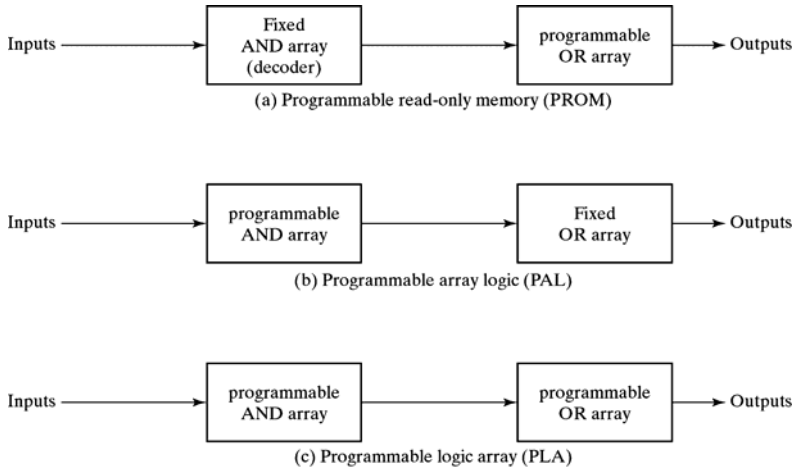


Fig. 7-13 Basic Configuration of Three PLDs

PLD Examples

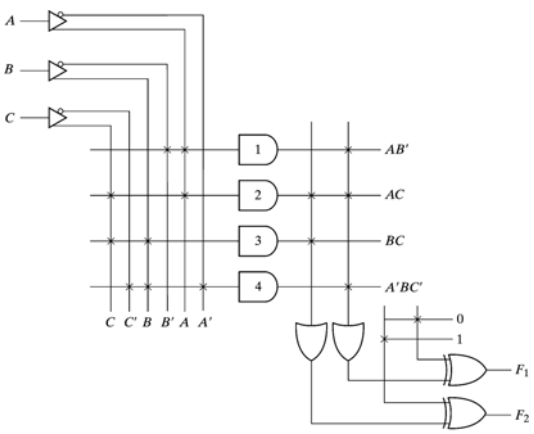


Fig. 7-14 PLA with 3 Inputs, 4 Product Terms, and 2 Outputs

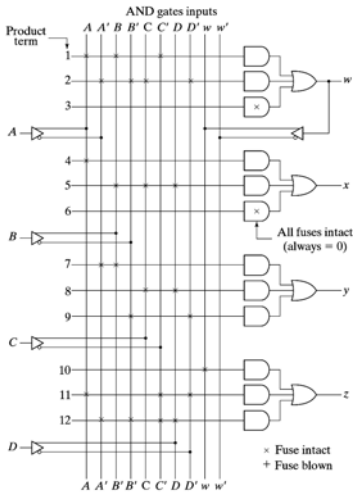


Fig. 7-17 Fuse Map for PAL as Specified in Table 7-6

4-65

HDL Modeling for Memory

- Modeling ROM and combinational PLDs
  - Similar to modeling a combinational code converter
- Modeling RAM
  - Use memory array declaration in Verilog
    ex: reg [3:0] MY_MEM [0:63]; // 64 4-bit registers
    MY_MEM[0] ← 4-bit variable
  - Can load memory by using a system task
    ex: $readmemb("mem_content", MY_MEM, 0, 63);
  - If synthesized, only SRAM (array of registers) will be generated
    - Use memory compiler or pre-designed layout instead

4-66