เนื้อหา

วงจรคอมไบเนชั่น

วงจรดิจิตอลสามารถออกแบ่งตามลักษณะการทำงานได้ 2 แบบคือ

1. วงจรคอมไบเนชั่น (Combinational Logic Circuit) เป็นวงจรที่นำเอาอุปกรณ์ลอจิกหลายตัวมาต่อเข้าด้วยกัน และจะต้องไม่มีส่วนของการป้อนกลับสัญญาณจากเอาต์พุตมาสู่อินพุต เป็นผลให้การทำงานของวงจรประเภทนี้ มีเอาต์พุตขึ้นอยู่กับอินพุตที่ป้อนเข้ามาเท่านั้น

2. วงจรซีเควนเชียล (Sequential Logic Circuit) เป็นวงจรที่นำเอาสัญญาณเอาต์พุตป้อนกลับมาเป็นอินพุตของวงจร เพื่อจะได้มีสภาวะที่สัมพันธ์ต่อเนื่องกัน จึงทำให้การทำงานของวงจรประเภทนี้ มีเอาต์พุตที่ขึ้นอยู่กับอินพุตที่ป้อนเข้ามาและเอาต์พุตก่อนหน้าด้วย

ขั้นตอนการออกแบบวงจรคอมไบเนชั่น

วงจร Combination สามารถออกแบบได้โดยใช้ขั้นตอน 4 ขั้นตอน เรียกว่า 4-Step Problem Solving Procedure (PSP) คือ

Step 1 : Problem Statement / Specification : บอกรายละเอียดของวงจรที่ต้องการ ประกอบด้วย Inputs, Outputs, และฟังก์ชั่นการทำงานของวงจร

Step 2 : Conceptualization : นำรายละเอียดการทำงานของวงจรมาเขียนเป็น Function Table หรือ Truth Table

Step 3 : Solution / Simplification : หาฟังก์ชั่นของแต่ละ O/P ในรูปของ Sum Of Product (SOP) หรือ Product Of Sum (POS)

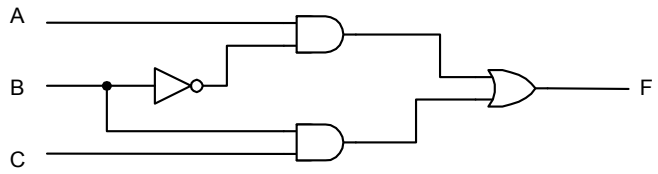
Step 4 : Realization : เขียนวงจรจากฟังก์ชั่นที่ได้ เพื่อนำไปประกอบวงจรและทดสอบว่าสามารถทำงานได้ตามที่กำหนดหรือไม่

วงจร Combination สามารถออกแบบได้โดยใช้อุปกรณ์ได้หลายประเภท ได้แก่

1. ใช้ Basic Gate (AND, OR , NOT, NAND, NOR, XOR, XNOR)
2. ใช้ MSI Ics เช่น Decoder, Multiplexer
3. ใช้ Programmable Logic Device

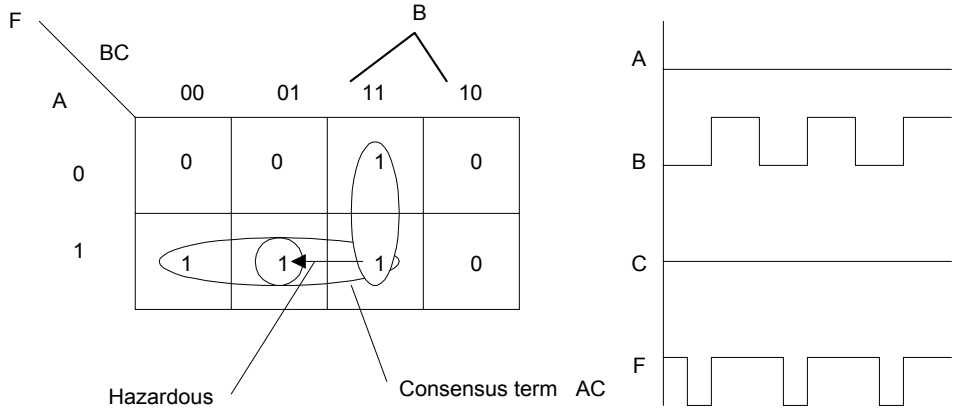
Hazards in Combinational Logic Circuits

Hazard เป็นภาวะที่เกิดขึ้นในวงจร Combination ที่สร้างจากเกตพื้นฐาน (Basic Gate) ซึ่งจะทำให้เกิดการทำงานที่ไม่เสถียร ถ้ามีสัญญาณที่อินพุตของเกตมาจากส่วนที่มี propagation delay ต่างกัน เมื่อค่าของอินพุตเปลี่ยนแปลง อาจจะปรากฏ Output transient (การเปลี่ยนแปลงชั่วคราวของสัญญาณ) ที่ไม่ต้องการขึ้น โดย Hazard จะทำให้เกิด Single Transient ที่เอาต์พุตซึ่งอาจยังเปลี่ยนค่าตามอินพุตไม่เสร็จ ดังตัวอย่างในรูปที่ 1



รูปที่ 1 ตัวอย่างวงจร Combination ที่มี Static Hazard

ถ้า เดิม A, B, C = 1, 1, 1 แล้วค่า B ถูกเปลี่ยนเป็น 0 เอาต์พุตของ B.C จะเปลี่ยนเป็น 0 ก่อนที่เอาต์พุตของ $\overline{A.B}$ จะเปลี่ยนเป็น 1 ทำให้เอาต์พุต F เป็น 0 ชั่วครู่ ก่อนที่จะได้เอาต์พุตที่แท้จริง ดังรูปที่ 2



รูปที่ 2 แสดงให้เห็น Static Hazard (a) K-Map (b) Timing Diagram

Hazard สามารถกำจัดออกไปได้โดยเพิ่มเทอมที่เหลือ (AC) ลงไปเพื่อให้ติดต่อระหว่างทั้ง 2 กลุ่ม นั่นคือ เปลี่ยนจาก

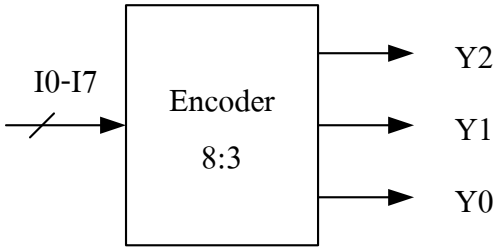
$$F = A\overline{B} + BC$$

เป็น

$$F = A\overline{B} + BC + AC$$

วงจรเข้ารหัส (Encoder)

หมายถึงวงจรที่เปลี่ยนรหัสใดๆ มาเป็นรหัสเลขฐานสอง เพื่อนำเข้าสู่ขั้นตอนของการประมวลผลในระบบดิจิตอล ดูตัวอย่างต่อไปนี้ เป็นวงจรเข้ารหัสจากเลขฐานแปดเป็นเลขฐานสอง ขนาด 3 บิต ซึ่งอินพุต8 เส้นที่เปรียบเหมือนเลขฐานแปด

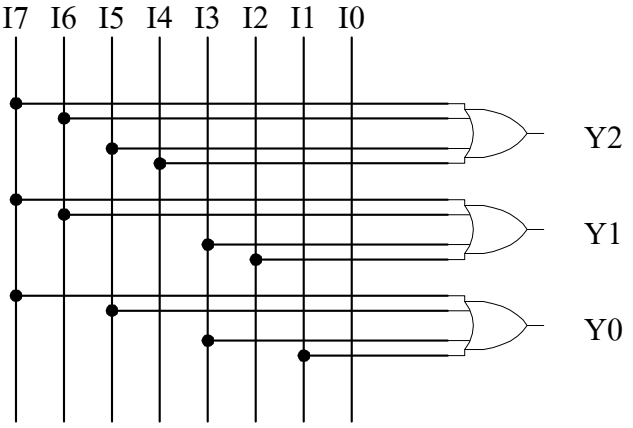


Input								Output		
I7	I6	I5	I4	I3	I2	I1	I0	Y2	Y1	Y0
1	X	X	X	X	X	X	X	1	1	1
0	1	X	X	X	X	X	X	1	1	0
0	0	1	X	X	X	X	X	1	0	1
0	0	0	1	X	X	X	X	1	0	0
0	0	0	0	1	X	X	X	0	1	1
0	0	0	0	0	1	X	X	0	1	0
0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	0	0	1	0	0	0

$$Y2 = I7+I6+I5+I4$$

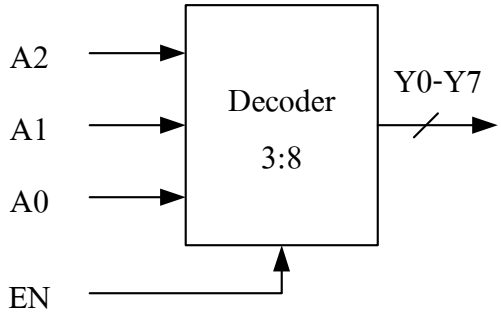
$$Y1 = I7+I6+I3+I2$$

$$Y0 = I7+I5+I3+I1$$



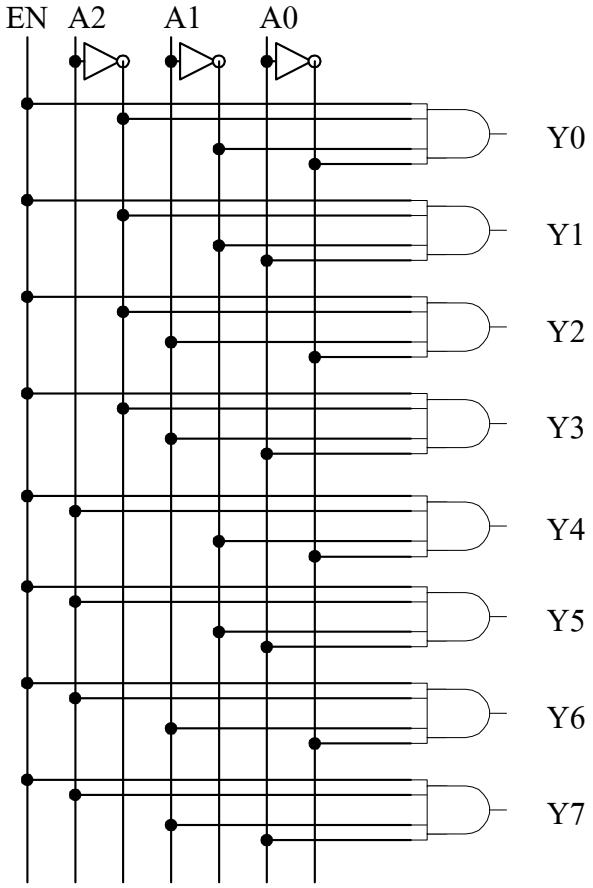
วงจรถอดรหัส (Decoder)

หมายถึงวงจรที่เปลี่ยนรหัสเลขฐานสอง ไปเป็นรหัสในรูปแบบอื่นๆ เพื่อใช้ในการแสดงผล หรือการสื่อสารระหว่างระบบเป็นต้น ดูตัวอย่างต่อไปนี้เป็นตารางความจริงของวงจรถอดรหัสเลขฐานสองเป็นฐานแปด ซึ่งมีเอาต์พุต 8 เส้นเปรียบเหมือนเลขฐานแปด



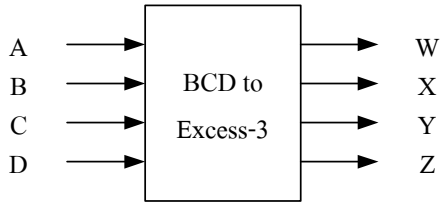
EN	A_2	A_1	A_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0		
0	X	X	X	0	0	0	0	0	0	0	0		
1	0	0	0	0	0	0	0	0	0	0	1	Y_0 =	$A'_2A'_1A'_0EN$
1	0	0	1	0	0	0	0	0	0	1	0	Y_1 =	$A'_2A'_1A_0EN$
1	0	1	0	0	0	0	0	0	1	0	0	Y_2 =	$A_2'A_1A'_0EN$
1	0	1	1	0	0	0	0	1	0	0	0	Y_3 =	$A'_2A_1A_0EN$
1	1	0	0	0	0	0	1	0	0	0	0	Y_4 =	$A_2A'_1A'_0EN$
1	1	0	1	0	0	1	0	0	0	0	0	Y_5 =	$A_2A'_1A_0EN$
1	1	1	0	0	1	0	0	0	0	0	0	Y_6 =	$A_2A_1A'_0EN$
1	1	1	1	1	0	0	0	0	0	0	0	Y_7 =	$A_2A_1A_0EN$

ตารางความจริงแสดงการทำงานของวงจร Decoder 3 อินพุต



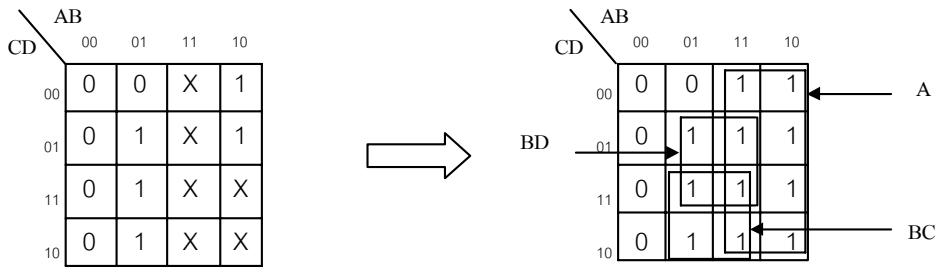
วงจรแปลงรหัส (Code Converter)

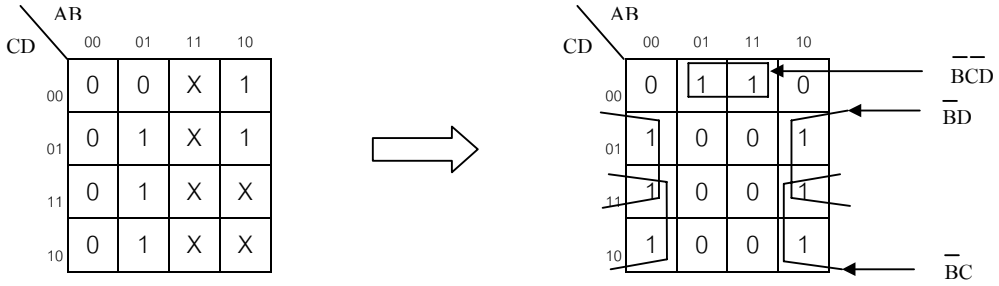
หมายถึงวงจรที่เปลี่ยนรหัสในรูปแบบหนึ่งไปเป็นรหัสอีกรูปแบบหนึ่ง เพื่อให้เกิดความเหมาะกับการทำงานในระบบดิจิตอลที่ใช้รหัสต่างกัน ดูตัวอย่างต่อไปนี้เป็นตารางความจริงของวงจรแปลงรหัส BCD-8421 เป็น Excess-3



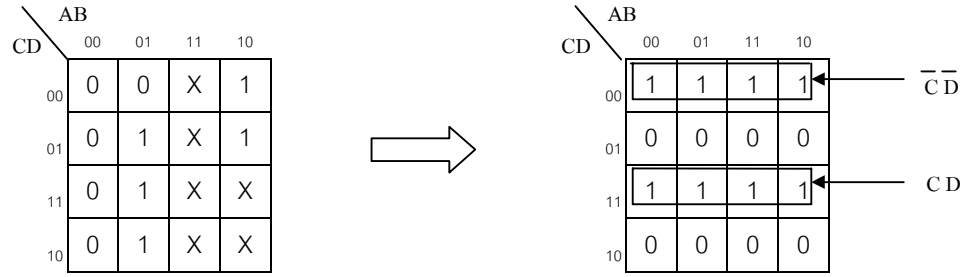
Input BCD-8421				Output Excess-3			
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

พิจารณาเอาต์พุตโดยใช้ min-term

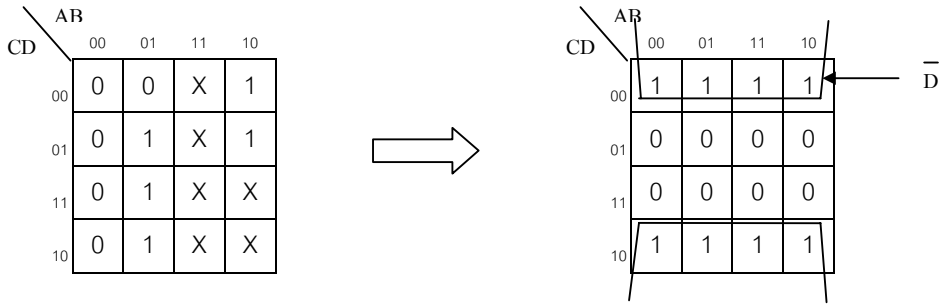




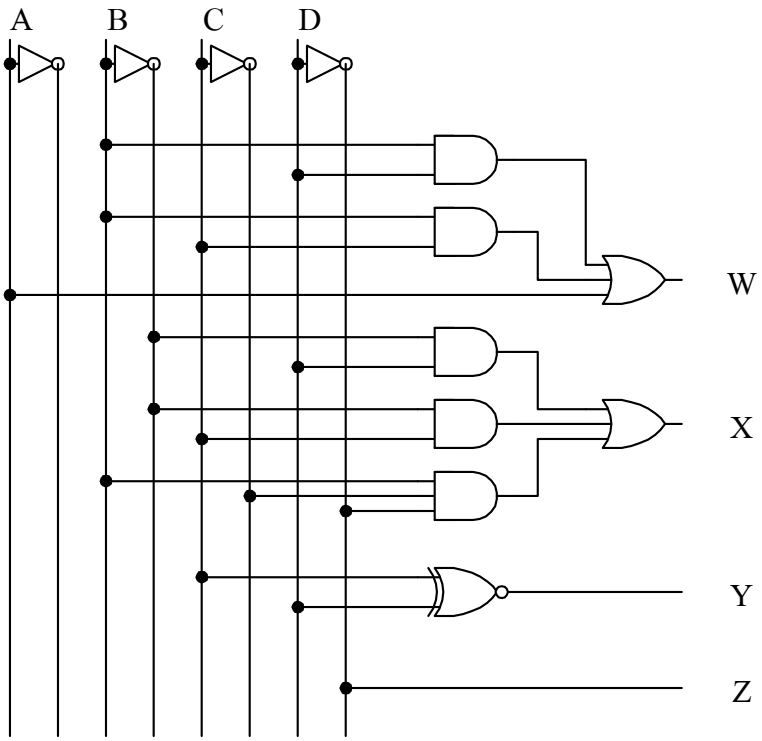
$$X = \overline{B}D + \overline{B}C + \overline{B}\overline{C}\overline{D}$$



$$Y = CD + \overline{C}\overline{D} = \overline{C \oplus D}$$

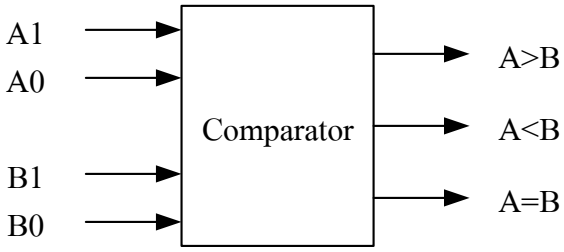


$$Z = \overline{D}$$



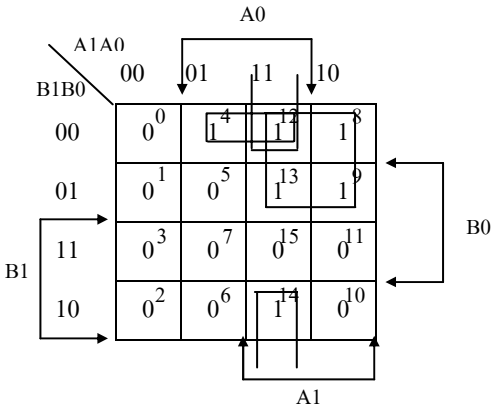
วงจรเปรียบเทียบ (Comparator)

เป็นวงจรที่ทำหน้าที่เปรียบเทียบอินพุตจำนวน 2 ชุดว่ามีค่าแตกต่างกันอย่างไร ดูตัวอย่างต่อไปนี้ เป็นตารางความจริงของวงจรเปรียบเทียบขนาด 2 บิต

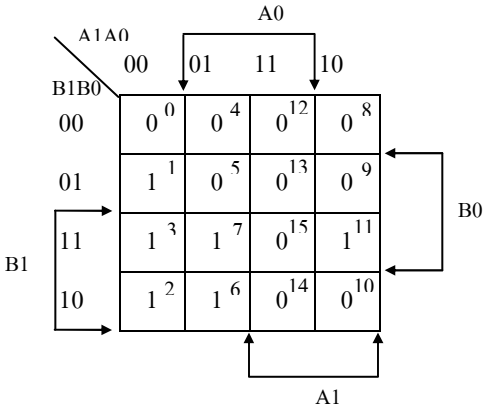


ลำดับ		อินพุต				เอาต์พุต		
A	**B**	A_1	A_0	B_1	B_0	**A>B**	**A<B**	**A=B**
0	0	0	0	0	0	0	0	1
0	1	0	0	0	1	0	1	0
0	2	0	0	1	0	0	1	0
0	3	0	0	1	1	0	1	0
1	0	0	1	0	0	1	0	0
1	1	0	1	0	1	0	0	1
1	2	0	1	1	0	0	1	0
1	3	0	1	1	1	0	1	0
2	0	1	0	0	0	1	0	0
2	1	1	0	0	1	1	0	0
2	2	1	0	1	0	0	0	1
2	3	1	0	1	1	0	1	0
3	0	1	1	0	0	1	0	0
3	1	1	1	0	1	1	0	0
3	2	1	1	1	0	1	0	0
3	3	1	1	1	1	0	0	1

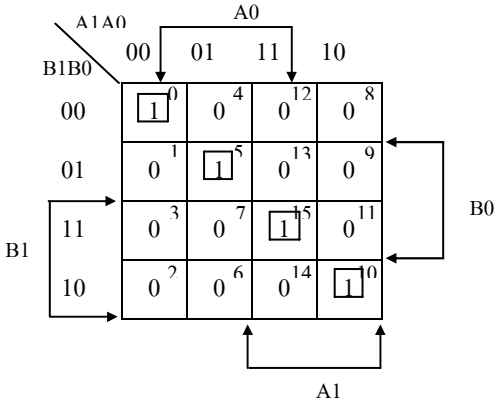
ตารางความจริงเปรียบเทียบระหว่างจำนวน 2 จำนวน



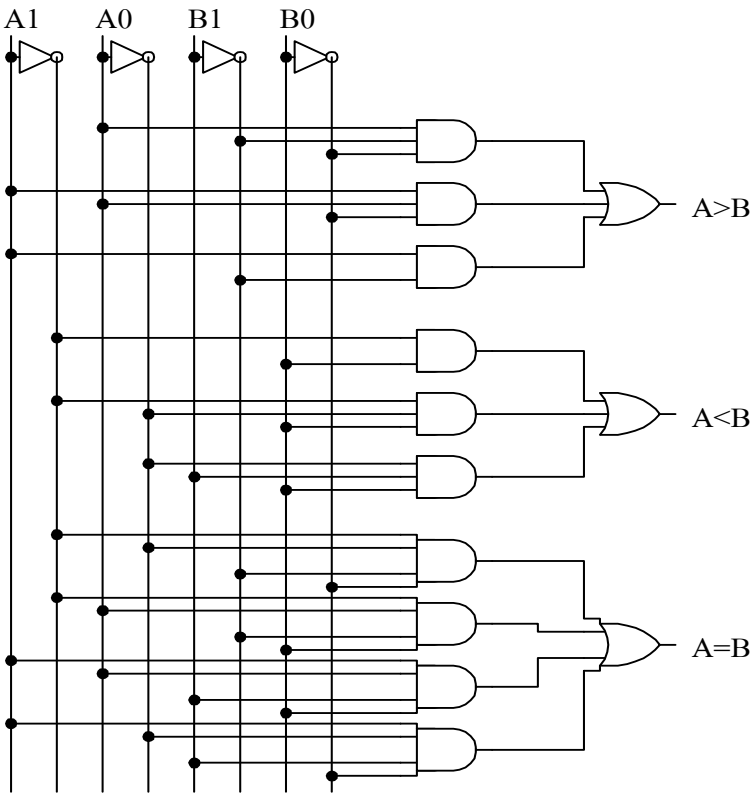
$(A>B) = A_0\overline{B}_1\overline{B}_0 + A_1A_0\overline{B}_0 + A_1\overline{B}_1$



$(A<B) = \overline{A}_0B_0 + \overline{A}_1\overline{A}_0B_0 + \overline{A}_1B_0B_1$

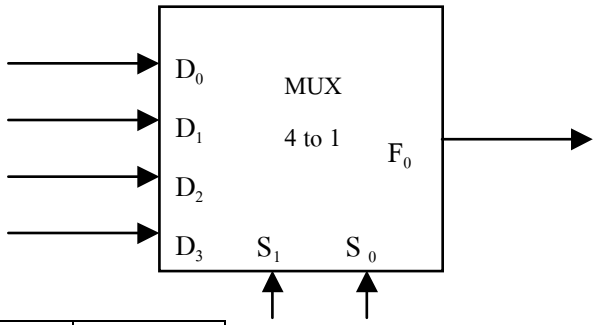


$(A=B) = \overline{A}_1\overline{A}_0\overline{B}_0\overline{B}_1 + \overline{A}_1A_0B_1\overline{B}_0 + A_1A_0B_0B_1 + A_1\overline{A}_0B_0B_1$



วงจรมัลติเพล็กซ์ (Multiplexer)

เป็นวงจรที่ทำหน้าที่สวิตช์เลือกสัญญาณจากอินพุตหลายเส้นไปออกยังเอาต์พุตเพียงเส้นเดียว เพื่อใช้ประโยชน์ในการสื่อสารระหว่างระบบดิจิตอล ที่ต้องการประหยัดสายส่งสัญญาณ ดูตัวอย่างต่อไปนี้เป็นวงจรมัลติเพล็กซ์ 4 อินพุต (D0-D4)



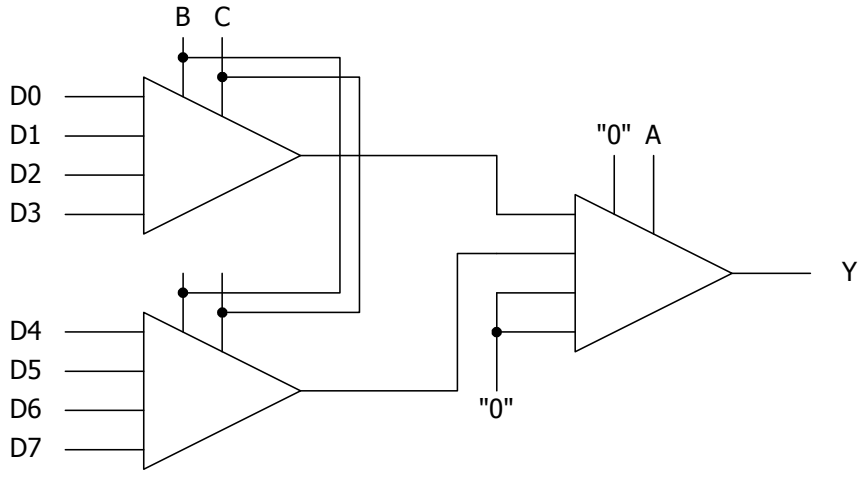
S_1	S_0	F_0
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

$$F0 = D0\overline{S1}\,\overline{S0} + D1\overline{S1}S0 + D2S1\overline{S0} + D3S1S0$$

$$= (D0\overline{S0} + D1S0)\overline{S1} + (D2\overline{S0} + D3S0)S1$$

เราสามารถนำวงจรมัลติเพล็กซ์หลายตัวมาต่อกันเพื่อให้รับอินพุตที่มากขึ้นได้ ดังตัวอย่างต่อไปนี้

A	B	C	Y
0	0	0	D0
0	0	1	D1
0	1	0	D2
0	1	1	D3
1	0	0	D4
1	0	1	D5
1	1	0	D6
1	1	1	D7



วงจรดีมัลติเพล็กซ์ (Demultiplexer)

เป็นวงจรที่ทำงานตรงกันข้ามกับวงจรมัลติเพล็กซ์ เพื่อใช้เป็นตัวรับสัญญาณในระบบสื่อสารดิจิตอลที่ทำหน้าที่แยกสัญญาณจากสายเส้นเดียวเป็นเอาต์พุตหลายเส้น ซึ่งระหว่างฝ่ายรับและฝ่ายส่งจะต้องทำงานเข้าจังหวะกันจึงจะได้ข่าวสารที่ถูกต้องตรงกัน ดูตัวอย่างต่อไปนี้เป็นตารางความจริงของวงจรดีมัลติเพล็ก 8 เอาต์พุต (Y0-Y7)

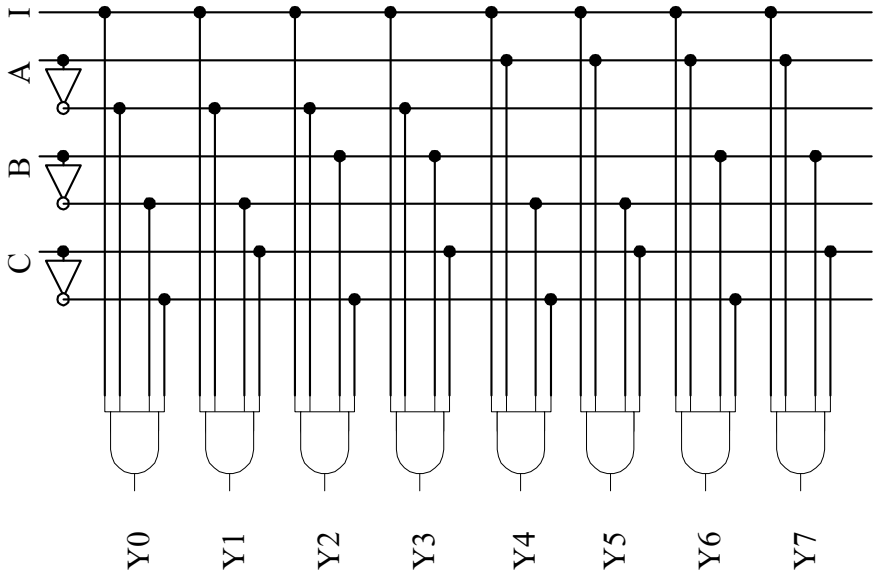
A	B	C	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0	0	0	0	0	0	0	0	I
0	0	1	0	0	0	0	0	0	I	0
0	1	0	0	0	0	0	0	I	0	0
0	1	1	0	0	0	0	I	0	0	0
1	0	0	0	0	0	I	0	0	0	0
1	0	1	0	0	I	0	0	0	0	0
1	1	0	0	I	0	0	0	0	0	0
1	1	1	I	0	0	0	0	0	0	0

$$Y0 = \overline{A}\,\overline{B}\,\overline{C}I \qquad Y4 = A\overline{B}\,\overline{C}I$$

$$Y1 = \overline{A}\,\overline{B}CI \qquad Y5 = A\overline{B}CI$$

$$Y2 = \overline{A}B\overline{C}I \qquad Y6 = AB\overline{C}I$$

$$Y3 = \overline{A}BCI \qquad Y7 = ABCI$$



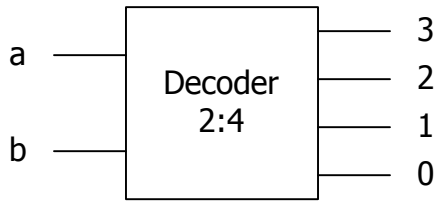
Implementation of Combinational – Logic Function

Using MSI ICs

การทำงานของ Combinational - Logic Function สามารถที่จะสร้างขึ้นได้โดยใช้อุปกรณ์ MSI เช่น Decoder หรือ Multiplexer เนื่องจากอุปกรณ์ทั้งสองอย่างสามารถหา เอาต์พุต ของ Minterm ทั้ง 2^n เทอมได้

Implementation Using Decoder

Decoder ขนาด n : 2^n จะมีอินพุตจำนวน n บิตและมีเอาต์พุตซึ่งแต่ละเอาต์พุตจะ Active เมื่อมีการป้อนค่าอินพุตที่สอดคล้องกับ Minterm ของเอาต์พุตนั้นดังรูป Block Diagram และ Truth Table ในรูปที่ 1



a	b	0	1	2	3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

รูปที่ 1 Block Diagram และ Truth Table ของ 2 : 4 Decoder

การใช้ Decoder ในการสร้างวงจรคือจะต้องพิจารณาจากฟังก์ชั่นของเอาต์พุตซึ่งอยู่ในรูปของ Sum of product สามารถสร้างวงจรโดยใช้ วงจร Decoder ที่มีจำนวนอินพุตของวงจรที่ต้องการ และดึงเอาเอาต์พุตของ Decoder ซึ่งแทน Minterm (หรือ Product term) ที่มีอยู่ในฟังก์ชั่นมาทำการ Or กัน (Sum) เพื่อให้ได้การทำงานตามฟังก์ชั่นที่ต้องการ

Implementation of 1 bit Full Adder Using 3 : 8 Decoder

Step 1 : Specification : ออกแบบวงจร Full Adder ใช้สำหรับบวกเลขขนาด 1 บิต x , y ร่วมเข้ากับตัวทดเข้า Ci เพื่อให้ได้ เอาต์พุต เป็นผลบวก Sum, S และตัวทดออก Carry Out, Co

Step 2 : Conceptualization : การทำงานของวงจร Full Adder ที่ต้องการ สามารถเขียนเป็น Truth Table ได้ดังต่อไปนี้

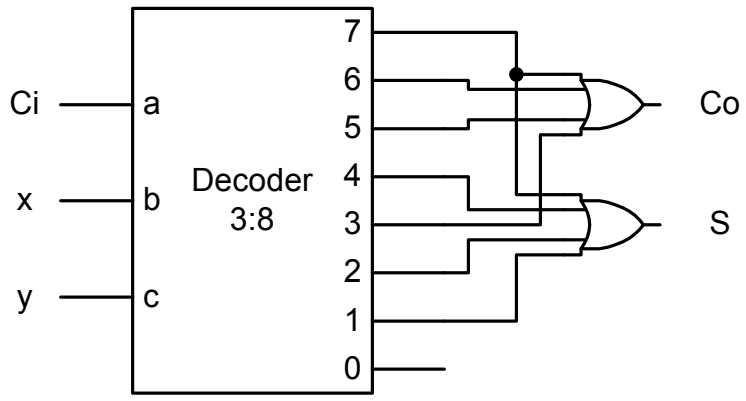
Ci	x	y	Co	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Step 3 : Simplification : จาก Truth Table สามารถเขียนเป็นฟังก์ชั่นของ S และ CO ในรูปของ Canonical Sum Of Product ได้ดังนี้

$$S = \sum(1,2,4,7)$$

$$Co = \sum(3,5,6,7)$$

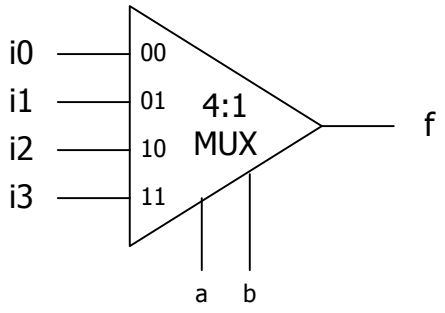
Step 4 : Realization : จากฟังก์ชั่นที่ได้ สามารถสร้างวงจรโดยใช้ 3:8 Decoder ได้ดังรูปที่ 2



รูปที่ 2 การสร้างวงจร Full Adder โดยใช้ 3:8 Decoder

Implementation Using Multiplexer

Multiplexer ขนาด 2^n : 1 จะมี Data Input จำนวน 2^n อินพุต และมี เอาต์พุต เพียง 1 เอาต์พุต โดยจะมี Selector จำนวน n บิต ที่ใช้สำหรับเลือกเอา Data Input เส้นใดเส้นหนึ่งงออกไปทาง เอาต์พุตดังตัวอย่างในรูปที่ 3 ซึ่งจะเห็นได้ว่า Selector จำนวน n บิตนั้น สามารถจะแทน Minterm ได้จำนวน2^n เทอม เป็นการเลือกอินพุตที่สอดคล้องกับ Minterm นั้นออกไปทาง เอาต์พุต



a	b	f
0	0	i0
0	1	i1
1	0	i2
1	1	i3

รูปที่ 3 Block Diagram และ Truth Table ของ 4 : 1 Multiplexer

การใช้ MUX ในการสร้างวงจรได้ โดยจะต้องใช้ MUX ที่มีจำนวน Selector เท่ากับจำนวน อินพุต ของวงจรที่ต้องการและป้อนค่าของเอาต์พุตตามฟังก์ชั่นที่ต้องการจาก Function Table เข้าที่ Data Input ที่สอดคล้องกับ Minterm แต่ละเทอมดังตัวอย่าง

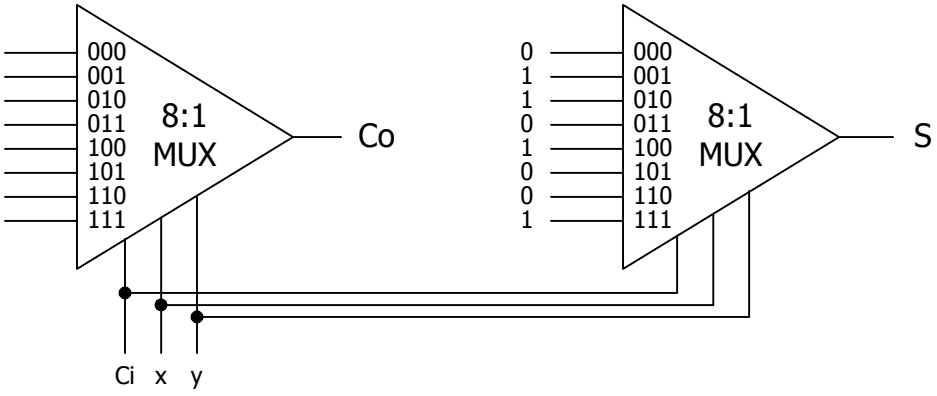
Implementation of 1 bit Full Adder Using 8 : 1 Multiplexer

Step 3 : Simplification : จาก Truth Table สามารถเขียนเป็นฟังก์ชั่นของ S และ Co ในรูป ของ Canonical Sum Of Product ได้ดังนี้

$$S = \sum (1,2,4,7)$$

$$Co = \sum (3,5,6,7)$$

Step 4 : Realization : จากฟังก์ชั่นที่ได้ สามารถสร้างวงจรโดยใช้ 8:1 MUX ได้ดังรูปที่ 4



รูปที่ 4 การสร้างวงจร Full Adder โดยใช้ 8: 1 MUX

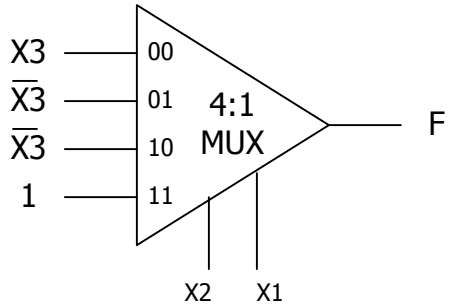
เราสามารถสร้างวงจรโดยใช้ MUX ที่มีจำนวน Selector น้อยกว่าจำนวนอินพุตของวงจรที่ต้องการได้ โดยใช้ อินพุตส่วนหนึ่งเป็น Selector และใช้ อินพุต อีกส่วนหนึ่งเป็นฟังก์ชั่นที่ Data Input ดังตัวอย่างของวงจรดัง Truth Table ต่อไปนี้

X3	X2	X1	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

ถ้าเขียนเป็น Variable – Entering Map จะได้ดังนี้

F	X2X1			
	00	01	11	10
	X3	$\overline{X3}$	1	$\overline{X3}$

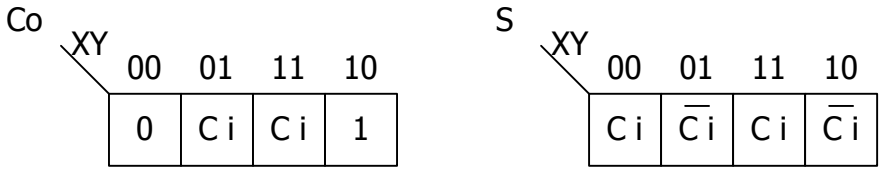
จะสามารถสร้างวงจรโดยใช้ 4:1 MUX ได้ดังรูปที่ 5



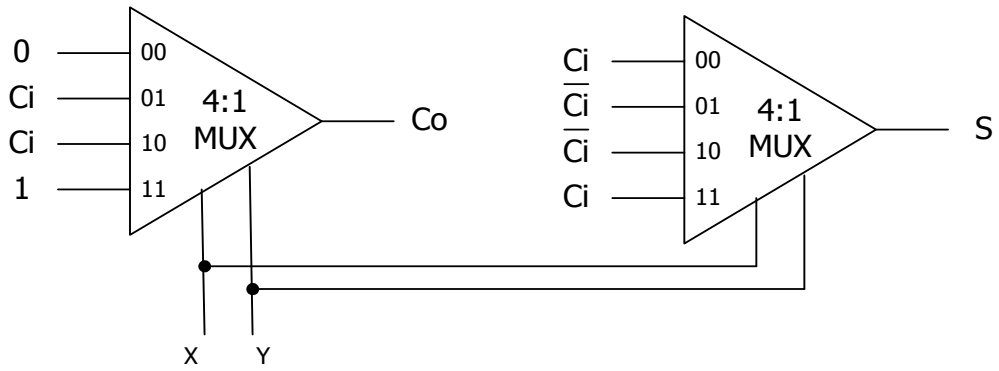
รูปที่ 5 วงจร 3 อินพุตสร้างจาก 4:1 MUX

Implementation of 1 bit Full Adder Using 4 : 1 MUX

Step 3: Simplification : จาก Truth Table สามารถเขียน Variable – Entering Map ของ S และ Ci ได้ดังนี้



Step 4 : Realization : สามารถสร้างวงจรโดยใช้ 4:1 MUX ได้ดังรูปที่ 6



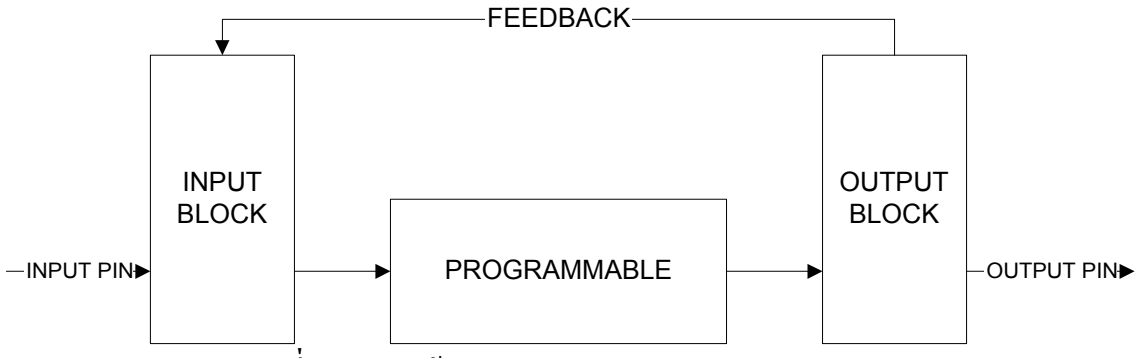
รูปที่ 6 การสร้างวงจร Full Adder โดยใช้ 4: 1 MUX

การสร้างวงจรนั้น ควรจะพิจารณาว่าฟังก์ชั่นเอาต์พุตที่เราต้องการนั้น เป็นฟังก์ชั่นที่ยุ่งยากหรือไม่ เพื่อพิจารณาว่า ระหว่างการใช้ เกตพื้นฐานกับการใช้ MUX วิธีไหนจะได้วงจรที่ง่ายกว่า

Implementation of Combinational-Logic Function

Using Programmable Logic Devices

อุปกรณ์ทางลอจิกที่สามารถโปรแกรมได้ (Programmable Logic Devices : PLDs) เป็นอุปกรณ์ที่ผู้ใช้สามารถกำหนดฟังก์ชั่นภายในได้ตามต้องการโครงสร้างภายในของ PLD จะเป็นดังรูปที่ 7



รูปที่ 7 โครงสร้างของ Programmable Logic Device

โครงสร้างหลักของ PLD ในส่วนที่โปรแกรมได้ จะอยู่ในรูปของเกต AND และ OR ซึ่งต่อกันไว้เพื่อให้ผู้ใช้สามารถโปรแกรมฟังก์ชั่นในรูป Sum Of Product ลงไปได้

- ในส่วนของอินพุตจะมีการ Latch โดย Buffer หรืออาจมี Option เพิ่มเติมในPLD บางชนิด

- ในส่วนของ Output Block ใน PLD บางตัวอาจมี Buffer เพื่อ Latch Output ก่อนส่งออกไป หรือใน PLD บางตัวอาจมี Control Register ซึ่งก็คือ Flip-Flop เป็นส่วนประกอบ สำหรับการ Drive Output และ เอาต์พุตอาจถูก Feedback กลับไปเป็นฟังก์ชั่น ของอินพุตด้วย

ส่วนของ Input Block และ Output Block นั้น ขึ้นอยู่กับ PLD แต่ละตัว ซึ่งจะมีส่วนประกอบไม่เหมือนกัน แต่สิ่งที่ PLD ทุกตัวมีเหมือนกันคือ ส่วนของ Programmable ‘AND’ and ‘OR’ Array คือส่วนที่ PLD ทุกตัวจะสามารถโปรแกรมฟังก์ชั่นลงไปได้ แต่วิธีในการโปรแกรมส่วนของ Programmable ‘AND’ and ‘OR’ Array นี้จะมีความแตกต่างกันไปใน PLD แต่ละชนิด

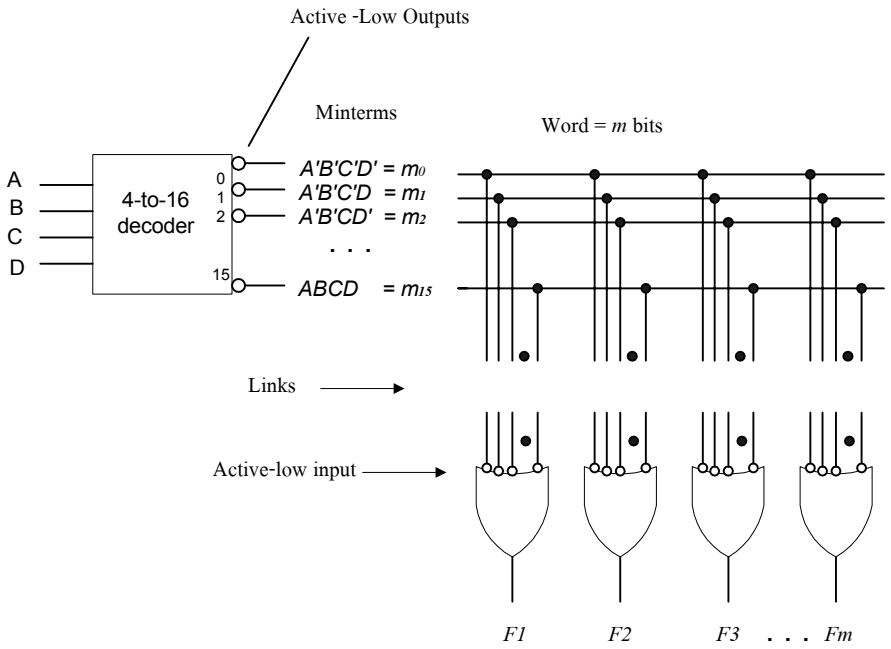
PLD แบ่งตามโครงสร้างในส่วนของ Programmable ‘AND’ and ‘OR’ Array เป็น 3 แบบ ดังตารางต่อไปนี้

Type Of PLD	AND Array	OR Array
Programmable Read-only Memory (PROM)	Decode All Minterm (Fixed)	Programmable
Programmable Logic Array (PLA)	Programmable	Programmable
Programmable AND-Array Logic (PAL)	Programmable	Fixed

PLD ทั้ง 3 ประเภท ยังแบ่งออกเป็นอีกหลายๆ แบบ ซึ่งมีลักษณะโครงสร้างในส่วนของ Input Block และ Output Block ที่ต่างๆ กันไป ซึ่งสามารถนำไปออกแบบวงจรต่างๆ กัน ในการใช้ PLD นั้น ผู้ใช้จึงต้องพิจารณาว่าจะใช้ PLD แบบใด โดยสามารถเลือกใช้ได้ตามความเหมาะสมของวงจรที่ต้องการ

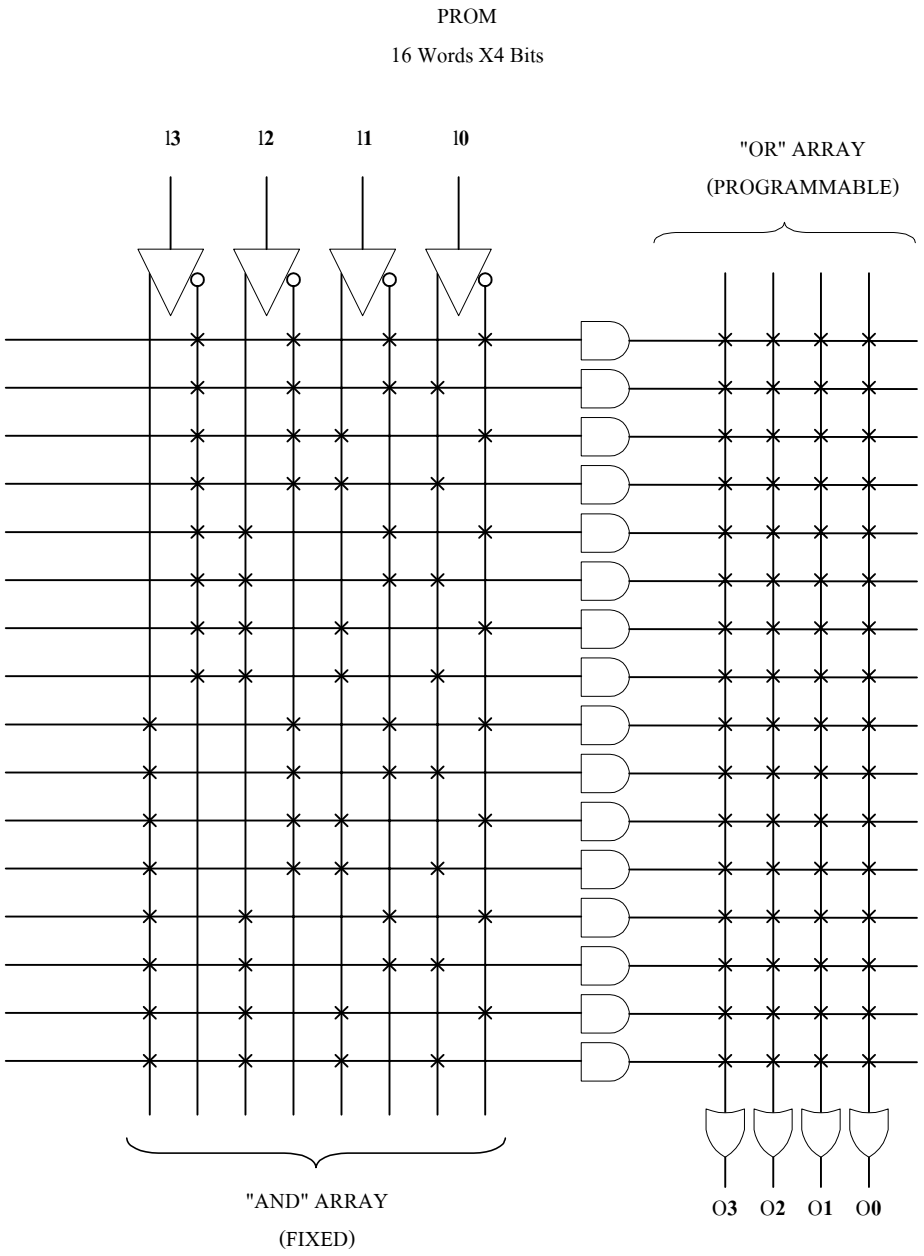
Programmable Read-Only Memory : PROM

PROM เรียกได้ว่าเป็น PLD แบบหนึ่ง PROM เป็นอุปกรณ์ประเภทหน่วยความจำ ในส่วนของ AND Array นั้นจะเป็น Decoder ซึ่งจะให้ผลเป็น 2^n Minterm ของ n Address lines และเอาต์พุตของ Decoder มักจะ Active Low ดังนั้น ส่วนของ OR Array จึงเป็น NAND gate ดังรูปที่ 8



รูปที่ 8 โครงสร้างภายในของ PROM

ดังนั้น ใน PROM ที่มี n อินพุต จะมี AND Array ของทั้ง 2^n Minterm จาก Decoder ซึ่งอินพุตที่ว่านี้ก็หมายถึง Address lines ทั้ง 2^n Minterm นี้จะจัดเตรียมไว้สำหรับนำไปสร้างฟังก์ชั่นแบบ Sum Of Product ทาง Output ส่วนทางด้าน Output จะเป็น OR Array ซึ่งสามารถที่จะโปรแกรมได้ว่า เอาต์พุตที่ OR แต่ละ เอาต์พุต จะได้จากการ OR กันของ Minterm ใดบ้าง สำหรับโครงสร้างทางลอจิกของ PROM จะเป็นดังรูปที่ 9



รูปที่ 9 โครงสร้างทางลอจิกของ PROM

Implement a Binary-to-Gray code Converter Using 16 ✕ 4 bit PROM

Step 1 : Specification : ออกแบบวงจรแปลงรหัสตัวเลข Binary ขนาด 4 บิต (B3, B2, B1, B0) เป็นรหัส Gray 4 บิต (G3, G2, G1, G0)

Step 2 : Conceptualization : การทำงานของ 4-bit Binary-to-Gray Code Converter สามารถเขียนเป็น Truth Table ได้ดังนี้

Binary Input				Gray Output			
B3	B2	B1	B0	G3	G2	G1	G0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Step 3 : Simplification : จาก Truth Table จะได้ฟังก์ชั่น SOP ของแต่ละเอาต์พุตดังนี้

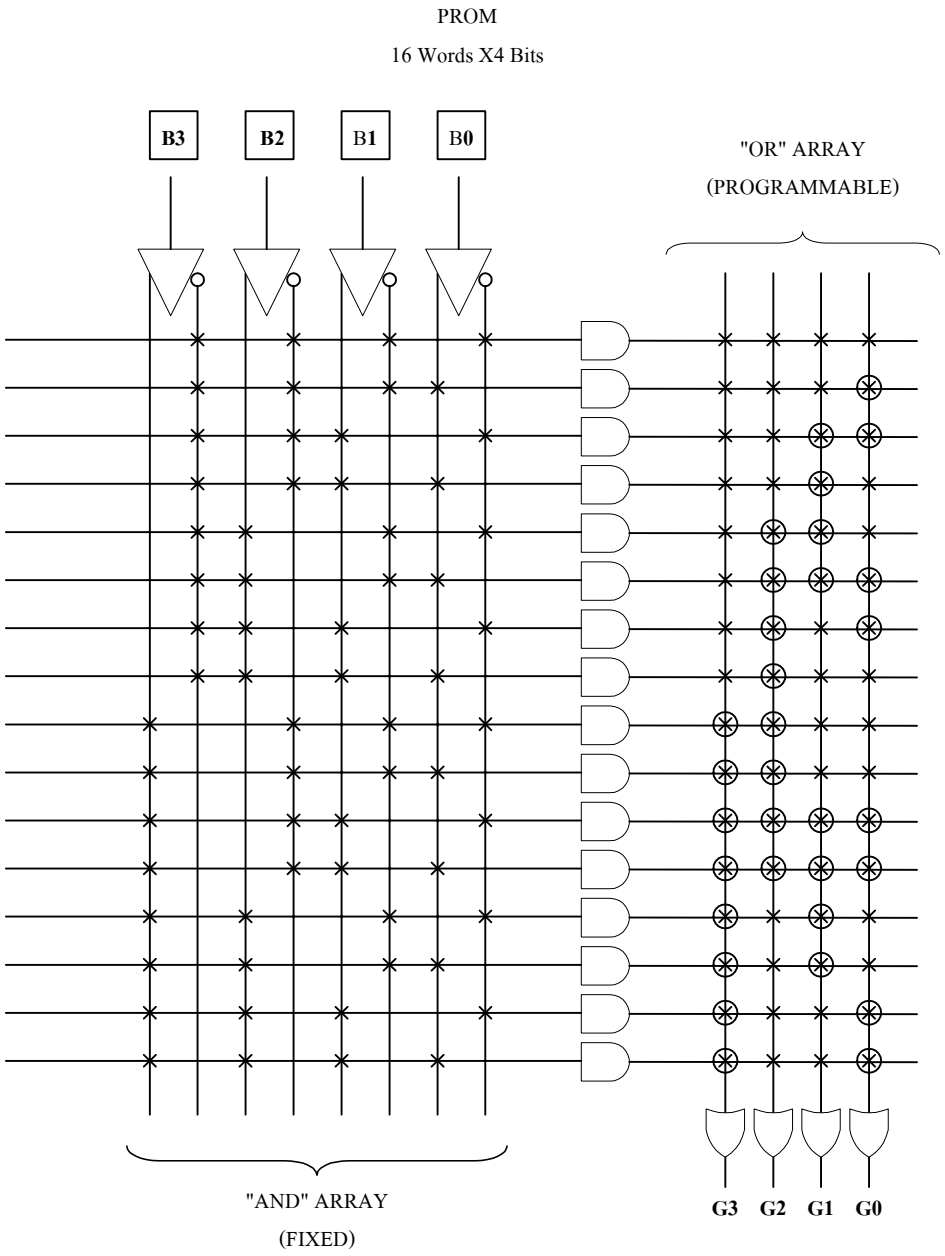
G3 = $\sum$m (8,9,10,11,12,13,14,15)

G2 = $\sum$m (4,5,6,7,8,9,10,11)

G1 = $\sum$m(2,3,4,5,10,11,12,13)

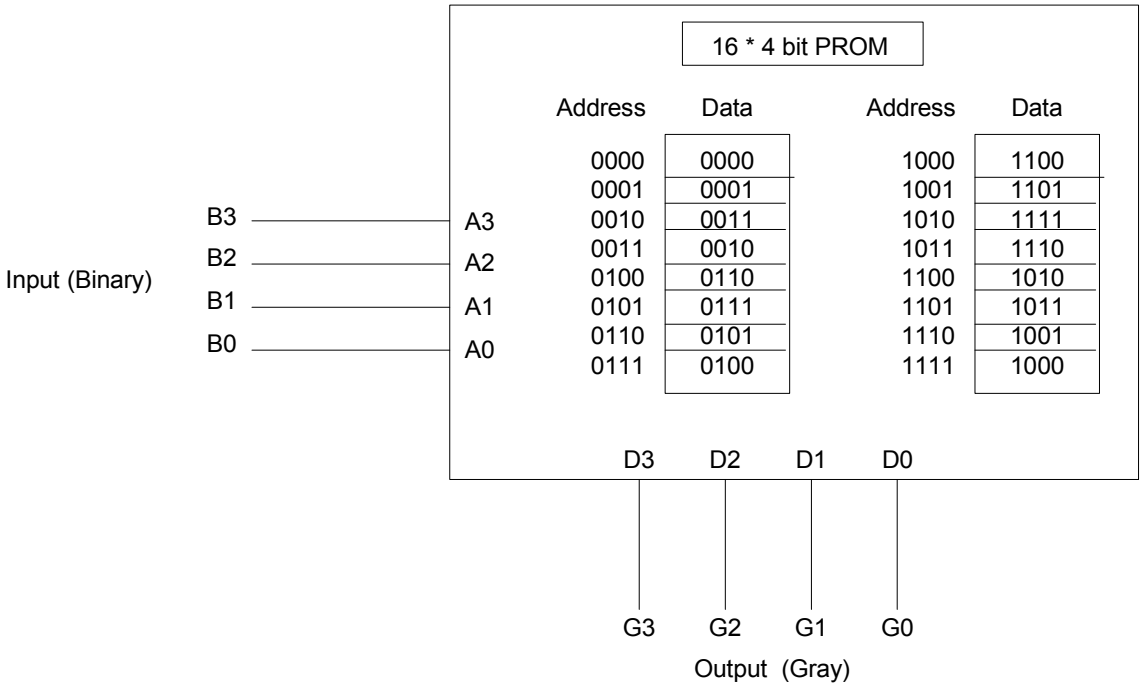
G0 = $\sum$m(1,2,5,6,9,10,13,14)

Step 4 : Realization : สร้างวงจรโดยใช้ PROM ขนาด 16 × 4 bit โดยวางแบบการโปรแกรม PROM ขนาด 16 × 4 bit ได้ดังรูปที่ 10



รูปที่ 10 โครงสร้างการโปรแกรม PROM ขนาด 16 × 4 bit ให้เป็นวงจร Binary to Gray Code Converter

PROM เป็นอุปกรณ์หน่วยความจำสามารถทำการโปรแกรมได้โดยการกำหนด Data ตามต้องการเข้าที่ Address ต่างๆ ดังนั้นสำหรับการโปรแกรมการทำงานของวงจร Binary-to-Gray Code Converter สามารถโปรแกรมได้ตาม Truth Table ใน Step 2 โดย อินพุต (B3, B2, B1, B0) เป็น Address line และ เอาต์พุต (G3, G2, G1, G0) เป็น Data line

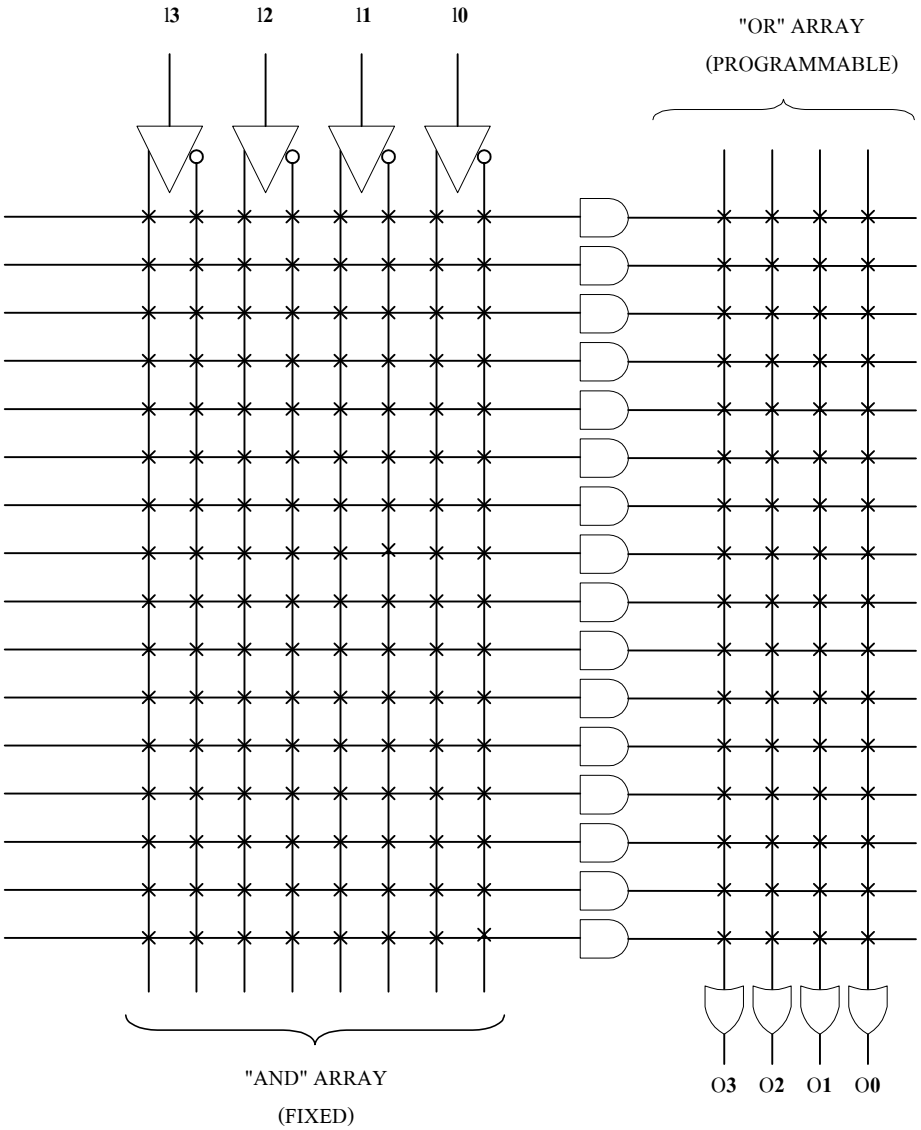


รูปที่ 11 Binary-to-Gray Code Converter สร้างจาก PROM ขนาด 16 × 4 bit

Programmable Logic Array : PLA

PLA นั้น ผู้ใช้สามารถโปรแกรมได้ทั้งส่วนของ AND Array และ OR Array นั่นคือ ส่วนของ AND Array ก็สามารถโปรแกรมได้ว่า AND gate แต่ละตัวจะใช้ อินพุต บ้าง และส่วนของ OR gate ก็สามารถโปรแกรมได้ว่า OR gate แต่ละตัวจะใช้ เอาต์พุต ของ AND gate ตัวใดบ้าง โครงสร้างทางลอจิกของ PLA เป็นดังรูปที่ 12

4 In .4 Out - 16 Products



รูปที่ 12 โครงสร้างทางลอจิกของ PLA

Implement 2-bit Multiplier Using 4 Input / 4 Output PLA

Step 1 : Specification : ออกแบบวงจร Multiplier ขนาด 2 บิต ซึ่งจะทำการคูณตัวเลข Binary 2 บิต AB เข้ากับ CD ซึ่งให้ เอาต์พุต ขนาด 4 บิต (f3, f2, f1, f0)

Step 2 : Conceptualization : วงจรที่ต้องการจะมีการทำงานดัง Truth Table ต่อไปนี้

A B C D	F3 F2 F1 F0
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 0
0 0 1 0	0 0 0 0
0 0 1 1	0 0 0 0
0 1 0 0	0 0 0 0
0 1 0 1	0 0 0 1
0 1 1 0	0 0 1 0
0 1 1 1	0 0 1 1
1 0 0 0	0 0 0 0
1 0 0 1	0 0 1 0
1 0 1 0	0 1 0 0
1 0 1 1	0 1 1 0
1 1 0 0	0 0 0 0
1 1 0 1	0 0 1 1
1 1 1 0	0 1 1 0
1 1 1 1	1 0 0 1

Step 3 : Simplification : จาก Truth Table สามารถเขียนหาฟังก์ชั่นในรูป SOP ได้ดังนี้

$F0 = B\,D$

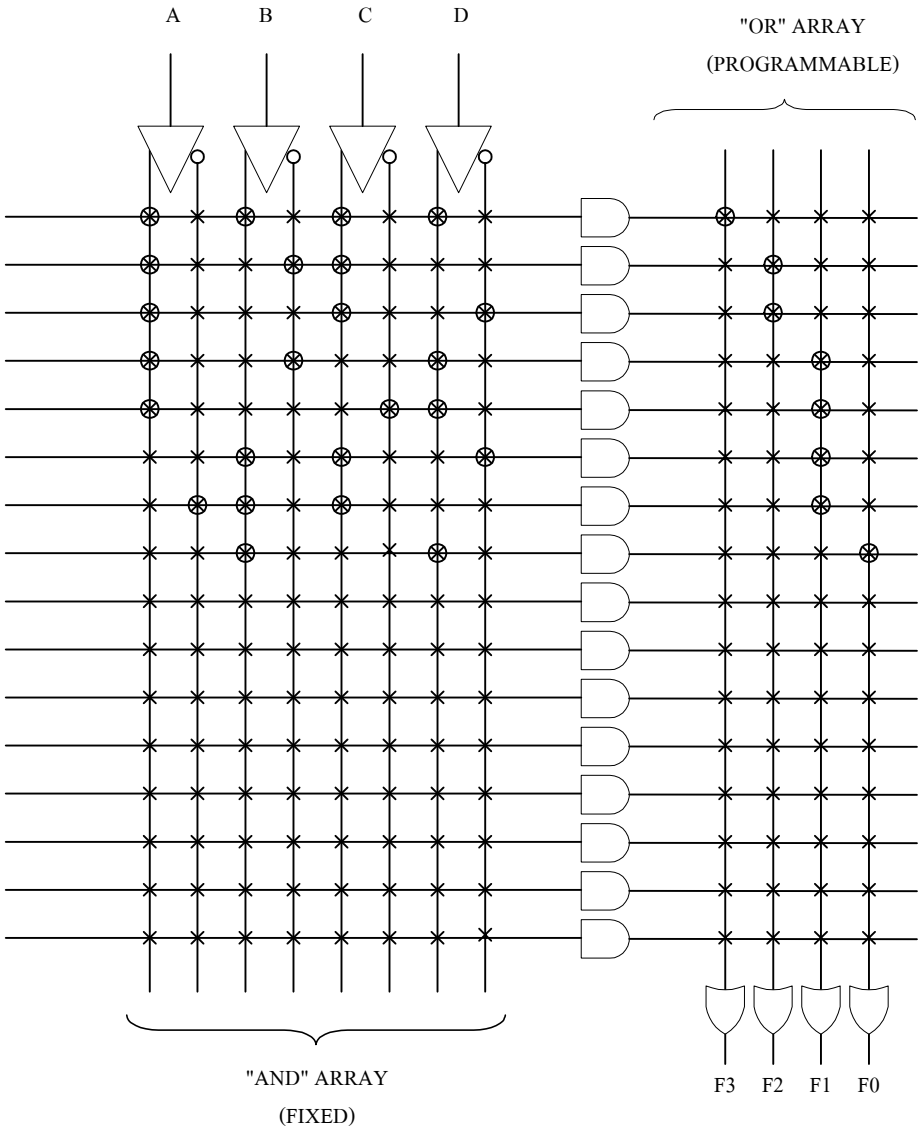
$F1 = A\,\overline{B}\,D + \overline{A}\,C\,D + B\,C\,\overline{D} + \overline{A}\,B\,C$

$F2 = A\,\overline{B}\,C + A\,C\,\overline{D}$

$F3 = A\,B\,C\,D$

Step 4 : Realization : จากฟังก์ชั่นที่ได้สามารถวางโครงสร้างการโปรแกรม PLA ได้ดังรูปที่ 13

4 In .4 Out - 16 Products

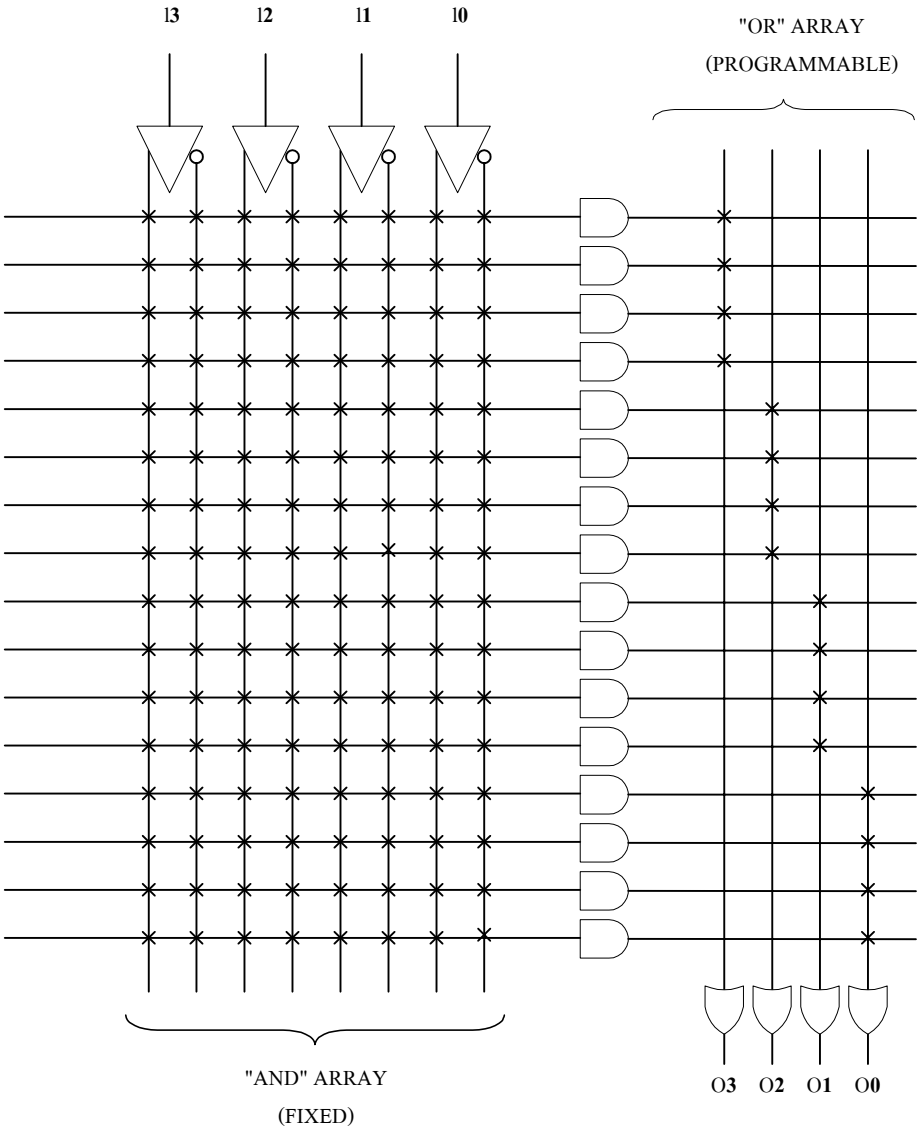


รูปที่ 13 โครงสร้างการโปรแกรม PLA เพื่อให้เป็นวงจร 2-bit Multiplier

Programmable AND-Array Logic : PAL

สำหรับ PAL นั้น ก็จะมี OR Array อยู่ เท่ากับจำนวน เอาต์พุต โดยที่ อินพุต ของ OR แต่ละตัวจะมี AND Array จำนวนหนึ่งต่อ Fixed อยู่ และผู้ใช้จะเป็นผู้โปรแกรมว่า AND แต่ละตัวนั้นจะเป็น Minterm ของอินพุตใดบ้าง โครงสร้างทางลอจิกของ PAL จะเป็นดังรูปที่ 14

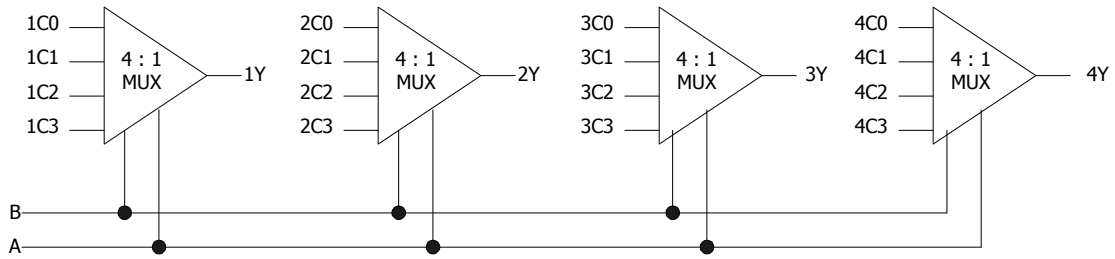
4 In .4 Out - 16 Products



รูปที่ 14 โครงสร้างทางลอจิกของ PAL

Implement Quad 4:1 MUX Using PAL18L4

Step 1 : Specification : ออกแบบวงจร 4:1 Multiplexer ซึ่งมี 4 อินพุต (nC0, nC1, nC2, nC3) , 1 เอาต์พุต (nY) และ 2 Selector (B, A) จำนวน 4 วงจร โดยทั้ง 4 วงจรใช้ Selector ร่วมกัน ดัง Block Diagram ในรูปที่ 15 โดยใช้ PAL18L4



รูปที่ 15 Block diagram ของ Quad 4:1 Multiplexer

Step 2 : Conceptualization : วงจร 4:1 MUX ทั้ง 4 วงจรมีการทำงานเหมือนกันดัง ต่อไปนี้

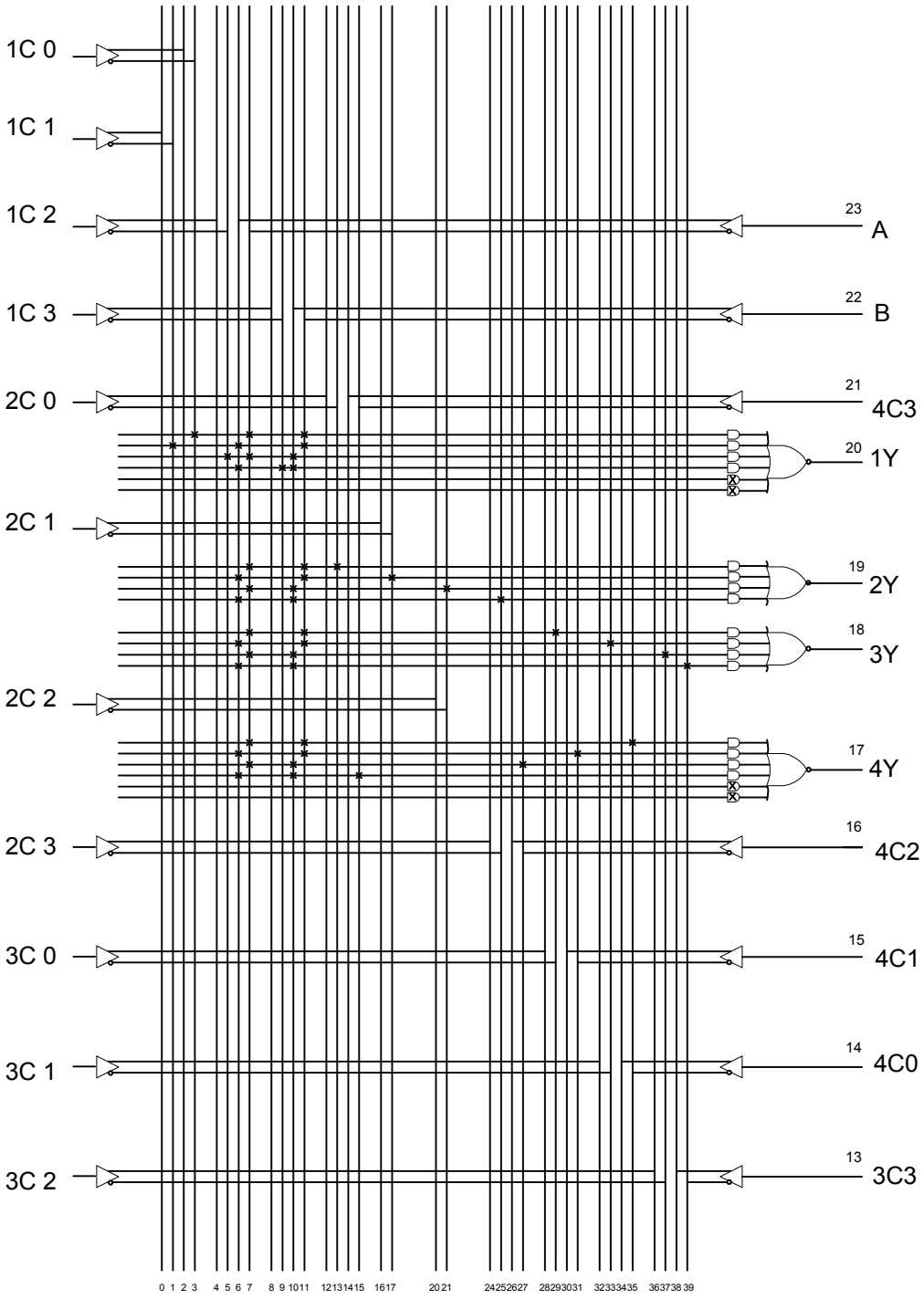
A B	nY	$\overline{nY}$
0 0	nC0	$\overline{nC0}$
0 1	nC1	$\overline{nC1}$
1 0	nC2	$\overline{nC2}$
1 1	nC3	$\overline{nC3}$

เนื่องจาก PAL18L4 มี เอาต์พุต Active Low คือเป็น NOR gate แต่ฟังก์ชั่นที่ใช้ต้องเป็นแบบ SOP ซึ่งใช้ OR gate ดังนั้นในการหาฟังก์ชั่น เราต้องหาฟังก์ชั่นที่ตรงข้ามกับ เอาต์พุต เพื่อให้เป็นฟังก์ชั่นของ AND-OR เมื่อเราต้องการเอาต์พุตเป็น $\overline{nY}$ ฟังก์ชั่นที่เราต้องหาคือฟังก์ชั่นของ nY

Step 3 : Simplification : เขียนฟังก์ชั่นของ nY ในรูปของ SOP ได้ดังนี้

$$NY = \overline{A}\,\overline{B}\,\overline{nCo} + \overline{A}\,B\,\overline{nC1} + A\,\overline{B}\,\overline{nC2} + A\,B\,\overline{nC3}$$

Step 4 : Realization : วางแบบการโปรแกรม PAL18L4 ได้ดังรูปที่ 16



รูปที่ 16 โครงสร้างการโปรแกรม PAL16L8 เป็นวงจร Quad 4 : 1 MUX