



MODUL PERKULIAHAN

SISTEM DIGITAL

Sistem Bilangan

Fakultas
Ilmu Komputer

Program Studi
Teknik Informatika

Tatap Muka

Kode MK
15048

Disusun Oleh
Tim Dosen

01

Abstract

Modul ini membahas tentang jenis jenis system bilangan, dan konversi antar system bilangan

Kompetensi

- Mahasiswa diharapkan dapat mengetahui jenis jenis system bilangan
- Mahasiswa diharapkan dapat melakukan konversi antar system bilangan

Jenis jenis sistem bilangan

Ada 4 jenis system bilangan yang digunakan dalam teknologi digital yaitu :

1. Sistem Bilangan Desimal
2. Sistem Bilangan Biner
3. Sistem Bilangan Oktal
4. Sistem Bilangan Hexa Desimal

SISTEM BILANGAN DESIMAL

System decimal terdiri dari 10 bilangan yaitu 0,1,2,3,4,5,6,7,8,9. System decimal disebut dengan Base-10 karena system ini memiliki 10 digit. System decimal merupakan positional value system dimana nilai dari sebuah diit bergantung pada posisinya. Sebagai contoh angka decimal 453. Digit 4 menyatakan 4 ratus, 5 menyatakan 5 puluh dan 3 menyatakan satuan.

Desimal Point

Contoh : 27.35

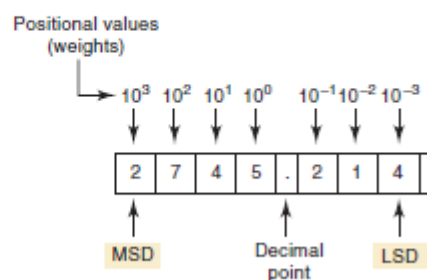
Bilangan ini sama dengan 2 puluhan ditambah 7 satuan ditambah 3 persepuluh ditambah 5 perseratus atau $2 \times 10 + 7 \times 1 + 3 \times 0.1 + 5 \times 0.01$

Decimal point digunakan untuk memisahkan bilangan bulat dan bilangan pecahan.

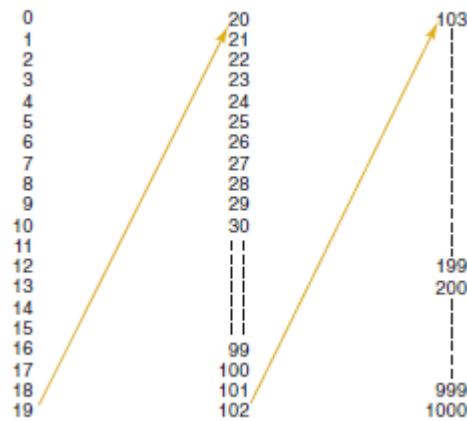
Gambar 1.1 memperlihatkan bilangan 2745.214

$$(2 \times 10^3) + (7 \times 10^2) + (4 \times 10^1) + (5 \times 10^0) + (2 \times 10^{-1}) + (1 \times 10^{-2}) + (4 \times 10^{-3})$$

FIGURE 1-3 Decimal position values as powers of 10.



Gambar 1.1 Posisi Nilai Desimal

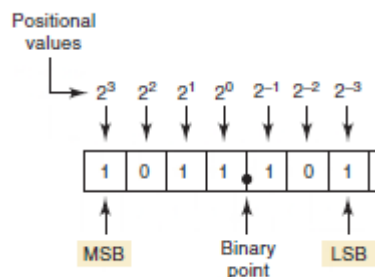


Gambar 1.2 Perhitungan nilai decimal

SISTEM BILANGAN BINER

Sistem bilangan decimal kurang tepat diimplementasikan dalam system digital. Hal ini akan sangat sulit merancang perangkat elektronik yang bekerja dengan 10 level tegangan berbeda dimana satu karakter decimal mewakili satu level tegangan). Akan sangat mudah untuk merancang rangkaian digital yang beroperasi hanya dengan 2 level tegangan. Karena alasan inilah hamper semua system digital menggunakan system bilangan biner (base 2) sebagai dasar system bilangan untuk operasinya.

Pada system bilangan biner terdapat 2 simbol atau nilai digit yaitu 0 dan 1. System bilangan biner juga merupakan sebuah system yang positional value, dimana setiap digit bilangan biner memiliki nilainya sendiri, yang dinyatakan sebagai kelipatan 2. Gambar 1.3 mengilustrasikan hal ini.



Gambar 1.3 Sistem bilangan Biner

Binary point merupakan pemisah antara pangkat 2 positif yang terletak dibagian kiri dan pangkat 2 negative yang terletak disebalah kanan. Contoh 1011.101

$$\begin{aligned}
 1011.101_2 &= (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\
 &\quad + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3}) \\
 &= 8 + 0 + 2 + 1 + 0.5 + 0 + 0.125 \\
 &= 11.625_{10}
 \end{aligned}$$

Dalam system bilangan biner, istilah digit bilangan biner disebut sebagai **bit**. Untuk bit dengan posisi paling kanan disebut sebagai LSB (Least Significat Bit) yang memiliki nilai paling kecil dan bit posisi paling kiri disebut sebagai MSB (Most Significant Bit) yang memiliki nilai paling besar.

Perhitungan Bilangan Biner

Weights →	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$	Decimal equivalent
	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	2
	0	0	1	1	3
	0	1	0	0	4
	0	1	0	1	5
	0	1	1	0	6
	0	1	1	1	7
	1	0	0	0	8
	1	0	0	1	9
	1	0	1	0	10
	1	0	1	1	11
	1	1	0	0	12
	1	1	0	1	13
	1	1	1	0	14
	1	1	1	1	15

↑
LSB

Gambar 1.4 Perhitungan Bilangan Biner

Pada Gambar 1.4 diatas bilangan biner dimulai dengan semua bit adalah 0, dan ini disebut zero count. Untuk perhitungan berikutnya, posisi satuan 2^0 berubah dari satu menjadi 0, berikutnya Posisi kedua 2^1 akan berubah dari 0 ke 1, berikutnya posisi ketiga 2^2 juga berubah dari 0 ke 1, dan posisi ke 4 2^3 berubah dari 0 ke 1, begitu seterusnya.

Bit LSB berubah dari 0 ke 1 atau dari 1 ke 0 setiap perhitungan. Bit kedua tetap berada pada 0 untuk 2 hitungan, kemudian berubah 1 untuk 2 hitungan, bit ketiga tetap pada bit 0 untuk 4 hitungan dan berubah 1 untuk 4 hitungan, begitu seterusnya.

SISTEM BILANGAN HEXADESIMAL

System bilangan ini menggunakan base 16, karena memiliki 16 digit symbol, yaitu menggunakan digit 0 – 9 ditambah dengan huruf A, B, C, D, E dan F.

Hexadecimal	Decimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15

SISTEM BILANGAN OKTAL

System bilangan ini menggunakan base 8, karena memiliki 8 digit symbol, yaitu menggunakan digit 0 – 7.



Konversi Antar Sistem Bilangan

Desimal ke Biner

Setiap unit bilangan biner merupakan kelipatan 2.

2^n	2^6	2^5	2^4	2^3	2^2	2^1	2^0
-------	-------	-------	-------	-------	-------	-------	-------	-------

Untuk melakukan konversi bilangan decimal ke biner dapat dilakukan dengan cara sebagai berikut .

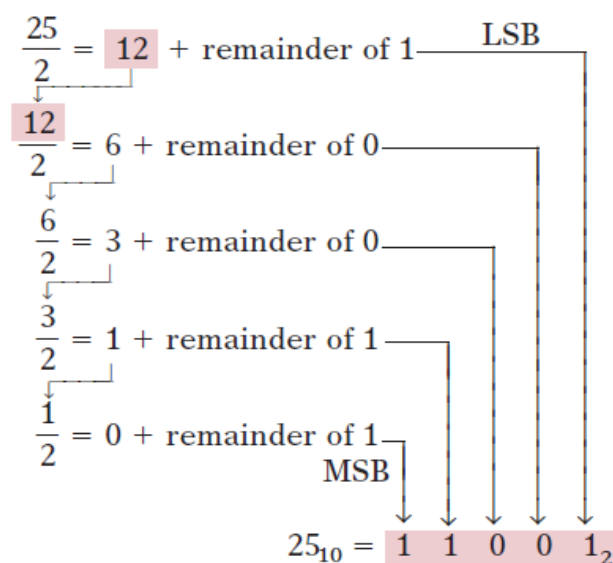
Contoh : 25 desimal

Cara 1 :

1. Nilai decimal adalah 25
2. Cari total nilai bit yang sama dengan 25 yaitu $16 + 8 + 1$.
3. Pada nilai bit yang ditotalkan, diberi bit 1, dan sisanya bit 0
4. Bit 0 pada sebelah kiri bisa diabaikan
5. 25 desimal = 11001 biner

$64=2^6$	$32=2^5$	$16=2^4$	$8=2^3$	$4=2^2$	$2=2^1$	$1=2^0$
0	0	1	1	0	0	1

Cara 2 : pembagian



Biner ke Desimal

Untuk melakukan konversi bilangan biner ke desimal maka cara yang dilakukan adalah kebalikan dari proses konversi decimal ke biner.

Contoh : 110010 biner

Cara :

1. Nilai biner adalah 110010
2. Tempatkan setiap bit bilangan biner dalam pola kelipatan 2
3. Cari total nilai bit yang bernilai 1 yaitu $32 + 16 + 2 = 50$

$64=2^6$	$32=2^5$	$16=2^4$	$8=2^3$	$4=2^2$	$2=2^1$	$1=2^0$
0	1	1	0	0	1	0

Biner ke Oktal

Konversi bilangan biner ke octal dilakukan per kelompok, dimana 3 bit / digit bilangan biner = 1 digit bilangan octal.

Contoh : 110010 biner

Cara :

1. Bilangan biner dibagi menjadi kelompok-kelompok, dimana 1 kelompok terdiri dari 3 digit bilangan biner
2. Kelipatan 2 dari bilangan biner dimulai dari 2^0 sampai 2^2
3. Hitung dan jumlahkan nilai bit untuk bit 1 per kelompok
4. Gabungkan setiap hasil perhitungan pada masing-masing kelompok
5. Untuk contoh $110010 = 62$ oktal

Kel 2			Kel 1		
$4=2^2$	$2=2^1$	$1=2^0$	$4=2^2$	$2=2^1$	$1=2^0$
1	1	0	0	1	0
4	2	0	0	2	0
$4+2+0=6$			$0+2+0=2$		
6			2		

110010

Oktal ke Biner

Konversi bilangan oktal ke biner dilakukan dengan cara sebaliknya, dimana 3 bit / digit bilangan biner = 1 digit bilangan oktal.

Contoh : 62 Oktal

Cara :

1. Bilangan oktal dibagi menjadi kelompok kelompok, dimana 1 kelompok terdiri dari 1 digit bilangan biner
2. Posisikan bit 1 pada setiap kelipatan 2 dari bilangan biner dimulai dari 2^0 sampai 2^2 untuk total nilai oktal.
3. Gabungkan setiap hasil perhitungan pada masing masing kelompok
4. Untuk contoh 62 oktal = 110010

Kel 2			Kel 1		
6			2		
$4=2^2$	$2=2^1$	$1=2^0$	$4=2^2$	$2=2^1$	$1=2^0$
4	2	0	0	2	0
1	1	0	0	1	0

Biner ke Hexadesimal

Konversi bilangan biner ke hexadesimal dilakukan per kelompok, dimana 4 bit / digit bilangan biner = 1 digit bilangan hexadesimal.

Contoh : 110010 biner

Cara :

1. Bilangan biner dibagi menjadi kelompok kelompok, dimana 1 kelompok terdiri dari 4 digit bilangan biner
2. Kelipatan 2 dari bilangan biner dimulai dari 2^0 sampai 2^3
3. Hitung dan jumlahkan nilai bit untuk bit 1 per kelompok
4. Gabungkan setiap hasil perhitungan pada masing masing kelompok
5. Untuk contoh 110010 = 31 hexadesimal

Kel 2				Kel 1			
$8=2^3$	$4=2^2$	$2=2^1$	$1=2^0$	$8=2^3$	$4=2^2$	$2=2^1$	$1=2^0$
		1	1	0	0	1	0
		2	1	0	0	1	0
2+1=3				1			
3				1			

110010

Hexadesimal ke Biner

Konversi bilangan hexadesimal ke biner dilakukan dengan cara sebaliknya, dimana 4 bit / digit bilangan biner = 1 digit bilangan hexadesimal.

Contoh : 31 Hexadesimal

Cara :

1. Bilangan hexadesimal dibagi menjadi kelompok kelompok, dimana 1 kelompok terdiri dari 1 digit bilangan hexadesimal
2. Posisikan bit 1 pada setiap kelipatan 2 dari bilangan biner dimulai dari 2^0 sampai 2^3 untuk total nilai hexadesimal.
3. Gabungkan setiap hasil perhitungan pada masing masing kelompok
4. Untuk contoh 31 hexadesimal = 110010

Kel 2				Kel 1			
3				1			
$8=2^3$	$4=2^2$	$2=2^1$	$1=2^0$	$8=2^3$	$4=2^2$	$2=2^1$	$1=2^0$
0	0	2	1	0	0	1	0
0	0	1	1	0	0	1	0

Daftar Pustaka

Ronald J. Tocci, Neal S. Widmer, Gregory L. Moss, Digital Systems Principles and Applications TENTH EDITION, 2007, Pearson Education International



MODUL PERKULIAHAN

SISTEM DIGITAL

Operasi Bilangan Biner

Fakultas
Ilmu Komputer

Program Studi
Teknik Informatika

Tatap Muka

02

Kode MK
15048

Disusun Oleh
Tim Dosen

Abstract

Modul ini membahas tentang operasi operasi pada bilangan biner serta operasi komplemen

Kompetensi

- Mahasiswa diharapkan dapat mengetahui operasi pada bilangan biner
- Mahasiswa diharapkan dapat melakukan operasi pada bilangan biner

Operasi Penjumlahan

Ada beberapa hal umum yang harus diketahui dalam penjumlahan bilangan biner yaitu sebagai berikut :

$0 + 0 = 0$
 $0 + 1 = 1$
 $1 + 0 = 1$
 $1 + 1 = 10 \rightarrow 0 + \text{carry } 1 \text{ ditempatkan pada posisi berikutnya}$
 $1 + 1 + 1 = 11 \rightarrow 1 + \text{carry } 1 \text{ ditempatkan pada posisi berikutnya}$

Contoh penjumlahan dalam bilangan biner

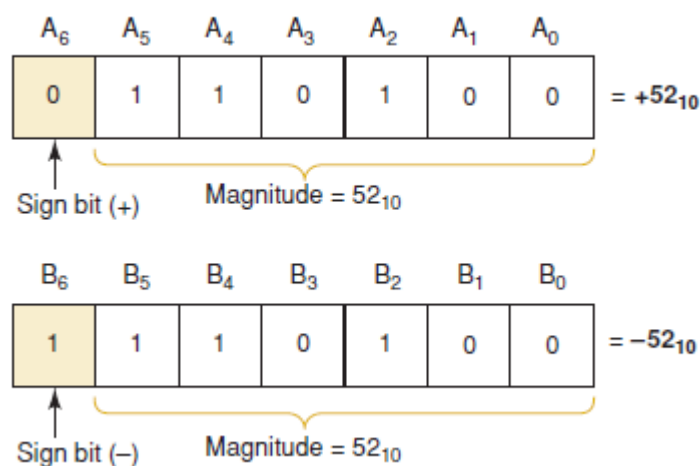
$$\begin{array}{r} 011 \text{ (3)} \\ + 110 \text{ (6)} \\ \hline 1001 \text{ (9)} \end{array}$$

$$\begin{array}{r} 1001 \text{ (9)} \\ + 1111 \text{ (15)} \\ \hline 11000 \text{ (24)} \end{array}$$

$$\begin{array}{r} 11.011 \text{ (3.375)} \\ + 10.110 \text{ (2.750)} \\ \hline 110.001 \text{ (6.125)} \end{array}$$

Bilangan Bertanda

Sebagian besar komputer digital menangani bilangan negative sebagai bilangan positif, sehingga diperlukan sign (tanda) bilangan + atau -. Tanda tersebut diwakili oleh satu bit yang disebut sebagai sign bit. Dimana 0 merupakan tanda positif dan 1 merupakan tanda negative. Bit tanda ini menempati posisi bit paling kiri atau pada bagian MSB seperti pada gambar 2.1 dibawah.



Gambar 2.1 Bilangan bertanda

Sign bit digunakan untuk menyatakan bilangan positif dan negative yang disimpan dalam bentuk bilangan biner. Pada Gambar 2.1 diatas terlihat bahwa bilangan tersebut terdiri dari 1 sign bit dan 6 magnitude bit. Magnitude bit merupakan bilangan biner yang nilainya sama dengan bilangan decimal yang mewakilinya. Prinsip ini dikenal dengan nama **sign magnitude system** untuk menyatakan bilangan biner bertanda.

System yang umum digunakan untuk menyatakan bilangan biner bertanda ini adalah 2's complement system. Komplemen 2 ini digunakan untuk menyatakan bilangan bertanda, karena untuk melakukan operasi pengurangan, sebenarnya operasi yang dilakukan adalah penjumlahan.

Langkah langkah melakukan komplemen 2 :

1's-Complement Form

Komplemen 1 dari sebuah bilangan biner merupakan diperoleh dari perubahan setiap 0 menjadi 1, dan 1 menjadi 0.

Contoh :

1 0 1 1 0 1	original binary number
↓ ↓ ↓ ↓ ↓ ↓	
0 1 0 0 1 0	complement each bit to form 1's complement

Komplemen 1 dari **101101** adalah **010010**

2's Complement Form

Komplemen 2 dari sebuah bilangan biner diperoleh dari hasil komplemen 1 ditambah dengan 1 pada posisi LSB.

1 0 1 1 0 1	binary equivalent of 45
0 1 0 0 1 0	complement each bit to form 1's complement
+ 1	add 1 to form 2's complement
0 1 0 0 1 1	2's complement of original binary number

Sehingga 010011 merupakan komplemen 2 dari 101101.

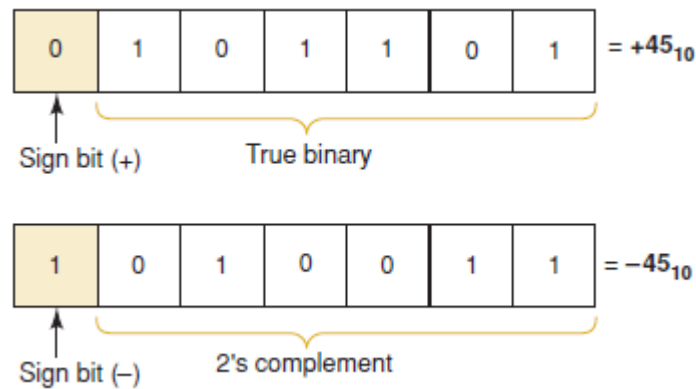
Representing Signed Numbers Using 2's Complement

Sistem komplemen 2 digunakan untuk menyatakan bilangan bertanda.

- Jika bilangan positif, magnitude dinyatakan dalam bentuk nilai bilangan biner asli dan sign bit adalah 0 ditempatkan pada bagian MSB.

- Jika bilangan negative, maka magnitude merupakan bentuk komplemen 2, dan sign bit adalah 1 ditempatkan pada bagian MSB.

2 ketentuan diatas dapat dilihat pada Gambar 2.2 dibawah.



Gambar 2.2 Bilangan bertanda dengan komplemen 2

PENJUMLAHAN COMPLEMENT DUA

Case I: Two Positive Numbers.

Menjumlahkan 2 bilangan positif sama seperti penjumlahan bilangan biner diatas,

Contoh +9 dan +4

$$\begin{array}{rcl}
 +9 \rightarrow & 0 & 1001 & \text{(augend)} \\
 +4 \rightarrow & 0 & 0100 & \text{(addend)} \\
 \hline
 & 0 & 1101 & \text{(sum = +13)} \\
 \uparrow & & & \text{sign bits}
 \end{array}$$

Case II: Positive Number and Smaller Negative Number.

Contoh : +9 dan -4

Langkah 1 : mencari nilai complemen 2 dari -4

$$\begin{array}{rcl}
 +4 & 0 & 1 & 0 & 0 \\
 C'1 & 1 & 0 & 1 & 1 \\
 \hline
 & & & & 1 \\
 C'2 & 1 & 1 & 0 & 0
 \end{array}$$

Langkah 2 : Menjumlahkan +9 dengan C'2 -4

$$\begin{array}{r}
 \text{sign bits} \\
 \downarrow \\
 \begin{array}{r}
 +9 \rightarrow \boxed{0} 1001 \quad (\text{augend}) \\
 -4 \rightarrow \boxed{1} 1100 \quad (\text{addend}) \\
 \hline
 1 \boxed{0} 0101 \\
 \uparrow
 \end{array}
 \end{array}$$

This carry is disregarded; the result is 00101 (sum = +5).

Pada kasus ini, sign bit dari addend (penambah) adalah 1. Hasil dari penjumlahan menghasilkan sebuah carry pada bagian akhir, dan carry ini diabaikan, sehingga hasil akhirnya adalah 0 0101 (+5)

Case III: Positive Number and Larger Negative Number.

Contoh : -9 dan +4

Langkah 1 : mencari nilai complemen 2 dari -9

$$\begin{array}{r}
 +9 \quad 1 \ 0 \ 0 \ 1 \\
 C'1 \quad 0 \ 1 \ 1 \ 0 \\
 \hline
 \quad \quad \quad 1 \\
 C'2 \quad 0 \ 1 \ 1 \ 1
 \end{array}$$

Langkah 2 menjumlahkan c'2 9 dengan +4

$$\begin{array}{r}
 -9 \rightarrow \quad 10111 \\
 +4 \rightarrow \quad 00100 \\
 \hline
 \quad 11011 \quad (\text{sum} = -5) \\
 \uparrow \text{negative sign bit}
 \end{array}$$

Dari hasil diperoleh 1 1011, dimana sign bit nya adalah 1 sehingga bilangannya adalah negative, dan magnitudenya merupakan hasil komplemen dua yaitu 1011, sehingga bilangan aslinya adalah :

$$\begin{array}{r}
 C'2 \quad 1 \ 0 \ 1 \ 1 \\
 C'1 \quad 0 \ 1 \ 0 \ 0 \\
 \hline
 \quad \quad \quad 1+ \\
 \text{Asli} \quad 0 \ 1 \ 0 \ 1 \quad \rightarrow 5
 \end{array}$$

Case IV: Two Negative Numbers

Contoh : -9 dan -4

Langkah 1 : mencari nilai complemen 2 dari -9

$$\begin{array}{r} +9 \quad 1001 \\ C'1 \quad 0110 \\ \hline \quad \quad 1 \\ C'2 \quad 0111 \end{array}$$

Langkah 2 : mencari nilai complemen 2 dari -4

$$\begin{array}{r} +4 \quad 0100 \\ C'1 \quad 1011 \\ \hline \quad \quad 1 \\ C'2 \quad 1100 \end{array}$$

Langkah 3 menjumlahkan

$$\begin{array}{r} -9 \rightarrow 10111 \\ -4 \rightarrow 11100 \\ \hline 1 \quad 10011 \\ \quad \uparrow \quad \uparrow \\ \quad \text{sign bit} \end{array}$$

This carry is disregarded; the result is 10011 (sum = -13).

Dari hasil diperoleh 1 1 0011, dimana 1 bit carry diabaikan, sign bit nya adalah 1 sehingga bilangannya adalah negative, dan magnitudenya merupakan hasil komplemen dua yaitu 0011, sehingga bilangan aslinya adalah :

$$\begin{array}{r} C'2 \quad 0011 \\ C'1 \quad 1100 \\ \hline \quad \quad 1+ \\ \text{Asli} \quad 1101 \end{array} \rightarrow 13$$

Case V: Equal and Opposite Numbers

Contoh : +9 dan -9

Langkah 1 : mencari nilai complemen 2 dari -9

$$\begin{array}{r} +9 \quad 1001 \\ C'1 \quad 0110 \\ \hline \quad \quad 1 \\ C'2 \quad 0111 \end{array}$$

Langkah 2 menjumlahkan

$$\begin{array}{r}
 -9 \rightarrow 10111 \\
 +9 \rightarrow 01001 \\
 \hline
 0 \quad 1 \quad 00000 \\
 \uparrow \\
 \text{Disregard; the result is } 00000 \text{ (sum} = +0\text{).}
 \end{array}$$

Perkalian Bilangan Biner

Perkalian bilangan biner dilakukan dengan cara yang sama dengan perkalian bilangan decimal. Proses menjadi lebih sederhana karena kita hanya mengalikan digit 1 atau 0, dan tidak melibatkan digit lainnya.

Contoh : 9×11

$$\begin{array}{r}
 1001 \quad \leftarrow \text{multiplicand} = 9_{10} \\
 1011 \quad \leftarrow \text{multiplier} = 11_{10} \\
 \hline
 1001 \\
 1001 \\
 0000 \\
 1001 \\
 \hline
 1100011
 \end{array}
 \left. \begin{array}{l} \\ \\ \\ \end{array} \right\} \text{partial products}$$

$1100011 \} \text{ final product} = 99_{10}$

Pada sebagian mesin digital, penjumlahan hanya dapat dilakukan antara 2 bilangan biner pada satu waktu, sehingga selama perkalian, maka penjumlahan tidak dilakukan seluruhnya pada satu waktu, tetapi penjumlahan dilakukan untuk 2 partial product per satuan waktunya. Pertama ditambah dengan kedua, hasilnya ditambah dengan ketiga, dan seterusnya seperti ilustrasi dibawah ini.

$$\begin{array}{r}
 \text{Add } \left\{ \begin{array}{l} 1001 \quad \leftarrow \text{first partial product} \\ 1001 \quad \leftarrow \text{second partial product shifted left} \end{array} \right. \\
 \hline
 \text{Add } \left\{ \begin{array}{l} 11011 \quad \leftarrow \text{sum of first two partial products} \\ 0000 \quad \leftarrow \text{third partial product shifted left} \end{array} \right. \\
 \hline
 \text{Add } \left\{ \begin{array}{l} 011011 \quad \leftarrow \text{sum of first three partial products} \\ 1001 \quad \leftarrow \text{fourth partial product shifted left} \end{array} \right. \\
 \hline
 1100011 \quad \leftarrow \text{sum of four partial products, which equals final total product}
 \end{array}$$

Pembagian Bilangan Biner

Proses Pembagian satu bilangan biner (dividend) dengan bilangan biner lainnya (divisor) sama dengan pembagian pada bilangan decimal. Prosesnya lebih sederhana dalam bilangan biner karena nilai yang dilibatkan hanya 0 atau 1.

Contoh :

$$\begin{array}{r} 0011 \\ 11 \overline{) 1001} \\ \underline{011} \\ 0011 \\ \underline{11} \\ 0 \end{array}$$

$$\begin{array}{r} 0010.1 \\ 100 \overline{) 1010.0} \\ \underline{100} \\ 100 \\ \underline{100} \\ 0 \end{array}$$

Daftar Pustaka

Ronald J. Tocci, Neal S. Widmer, Gregory L. Moss, Digital Systems Principles and Applications TENTH EDITION, 2007, Pearson Education International



MODUL PERKULIAHAN

SISTEM DIGITAL

Gerbang Logika Dasar

Fakultas
Ilmu Komputer

Program Studi
Teknik Informatika

Tatap Muka

03

Kode MK
15048

Disusun Oleh
Tim Dosen

Abstract

Modul ini membahas tentang gerbang gerbang logika dasar

Kompetensi

- Mahasiswa diharapkan dapat mengetahui gerbang gerbang logika dasar

Aljabar Boolean

Aljabar Boolean berbeda dengan aljabar lainnya karena konstanta dan variable Boolean hanya memiliki dua nilai yaitu 1 atau 0.

Variabel Boolean sering digunakan untuk menyatakan level tegangan pada kabel atau terminal input dan output dari sebuah rangkaian. Contoh dalam system digital, 0 mewakili range tegangan dari 0 – 0.8 V, sedangkan 1 mewakili range tegangan dari 2 – 5 V.

Karena Boolean 0 dan 1 tidak mewakili bilangan asli tetapi menyatakan posisi tegangan, maka disebut sebagai logic level.

Macam macam istilah yang digunakan untuk mewakili 0 dan 1 adalah seperti berikut:

Logic 0	Logic 1
False	True
Off	On
Low	High
No	Yes
Open switch	Closed switch

Aljabar Boolean merupakan sebuah persamaan yang menyatakan hubungan antara input dan output dari sebuah rangkaian logika.

Aljabar Boolean memiliki 3 operasi dasar yaitu OR, AND dan NOT. Tiga operasi dasar ini disebut operasi logika.

Rangkaian digital disebut sebagai gerbang logika yang dibangun dari diode, transistor, dan resistor yang dihubungkan sehingga output rangkaian merupakan hasil dari operasi logika (OR, AND, NOT) yang dilakukan pada input.



TABEL KEBENARAN

Tabel kebenaran merupakan gambaran bagaimana sebuah output dari rangkaian logika bergantung kepada logic level yang ada pada rangkaian inputnya, Tabel kebenaran ini berisi kombinasi dari logic level yang ada pada input. untuk jumlah kombinasi input adalah 2^n . Untuk 2 input maka kombinasinya adalah $2^n = 2^2$ yaitu 4 kombinasi input.

Tabel Kebenaran 2 input

Inputs		Output
A	B	x
0	0	1
0	1	0
1	0	1
1	1	0

Tabel Kebenaran 3 input

A	B	C	x
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

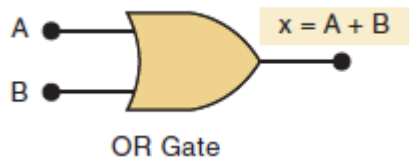
Table Kebenaran 4 input

A	B	C	D	x
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Operasi OR

Operasi OR merupakan operasi logika dimana akan menghasilkan nilai 1 jika salah satu atau semua inputnya bernilai 1. Dalam aljabar Boolean operasi OR dinyatakan sebagai tanda tambah (+)

Gerbang Logika OR 2 input



Gerbang Logika OR 3 Input



Tabel Kebenaran 2 Input

OR		
A	B	$x = A + B$
0	0	0
0	1	1
1	0	1
1	1	1

Tabel Kebenaran 3 input

A	B	C	$x = A + B + C$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Operasi OR

$$x = A + B$$

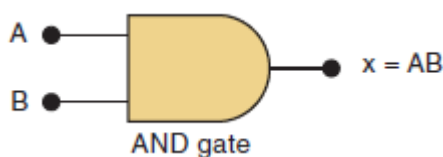
Operasi OR

$$x = A + \bar{B} + C$$

Operasi AND

Operasi AND merupakan operasi logika dimana akan menghasilkan nilai 1 jika semua inputnya bernilai 1. Dalam aljabar Boolean operasi AND dinyatakan sebagai tanda kali (.)

Gerbang Logika AND 2 input



Gerbang Logika AND 3 Input



Tabel Kebenaran 2 Input

AND		
A	B	$x = A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

Operasi OR

$$x = A \cdot B$$

Tabel Kebenaran 3 input

A	B	C	$x = ABC$
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

Operasi OR

$$x = ABC.$$

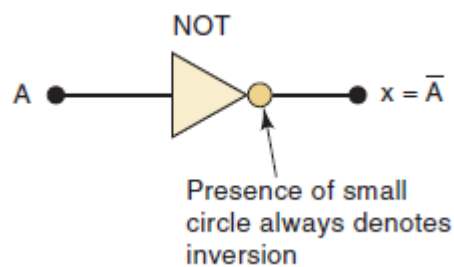
Operasi NOT

Operasi NOT berbeda dengan operasi OR dan AND, karena hanya memiliki 1 input. sebagai

contoh Variabel A dilakukan operasi NOT maka hasil x dinyatakan sebagai $x = \bar{A}$.

Operasi NOT memberikan output dengan nilai kebalikan dari inputnya. Jika inputnya 0, maka outputnya adalah 1, dan jika inputnya adalah 1 maka outputnya adalah 0.

Gerbang NOT



Tabel Kebenaran

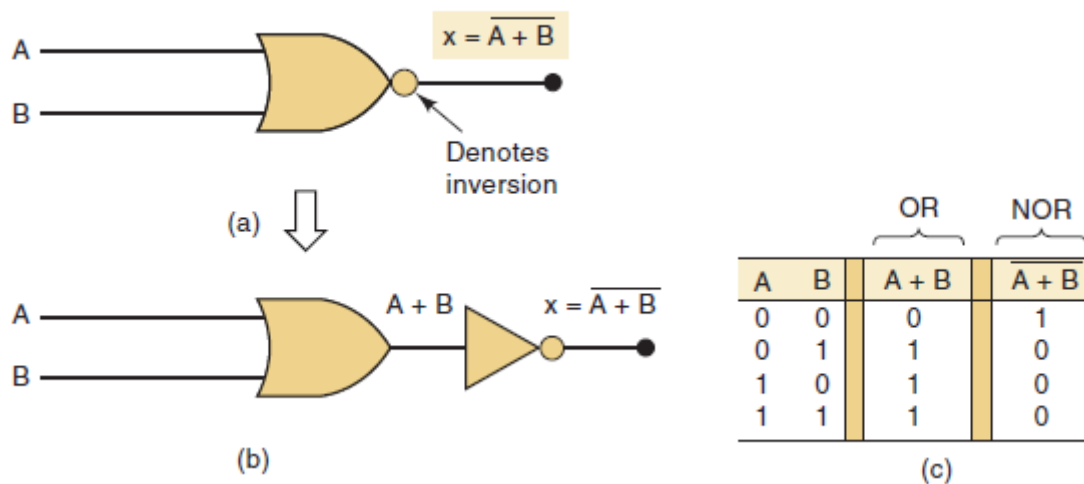
NOT	
A	$x = \bar{A}$
0	1
1	0

Gerbang logika NAND dan NOR

2 gerbang logika lainnya adalah Gerbang NOR dan gerbang NAND yang merupakan kombinasi dari gerbang dasar AND, OR, dan NOT.

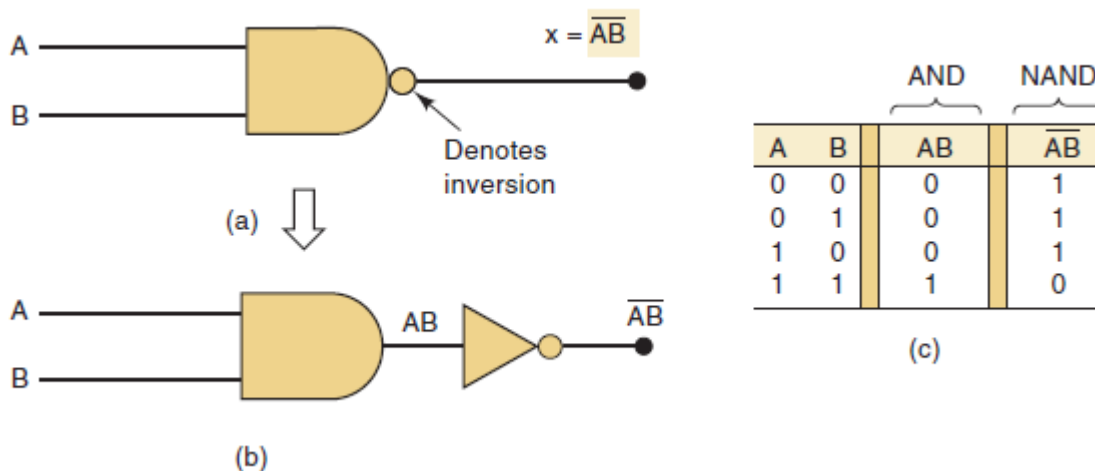
Gerbang NOR

Simbol dari Gerbang NOR mirip dengan symbol dari Gerbang OR dengan tambahan lingkaran kecil pada output. Lingkaran kecil ini menandakan operasi invers. Sehingga Gerbang NOR beroperasi seperti gerbang OR + sebuah inverter (NOT).



Gerbang NAND

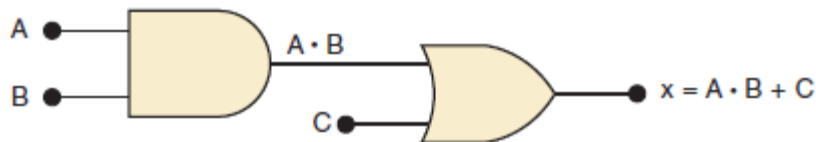
Simbol dari Gerbang NAND mirip dengan symbol dari Gerbang AND dengan tambahan lingkaran kecil pada output. Lingkaran kecil ini menandakan operasi invers. Sehingga Gerbang NAND beroperasi seperti gerbang AND + sebuah inverter (NOT).



RANGKAIAN DIGITAL → Persamaan Aljabar Boolean

Setiap rangkaian digital bagaimana pun kompleksnya, dapat digambarkan menggunakan 3 operasi dasar Boolean karena OR, AND dan NOT merupakan dasar dalam membangun system digital.

Contoh



Rangkaian ini memiliki 3 input A, B dan C dan satu buah output x. menggunakan persamaan Boolean, maka kita dapat dengan mudah membuat persamaan output.

Gambar a.

Gerbang 1 AND, dengan input A dan B → AB

Output Gerbang AND dan input C dihubungkan dengan Gerbang OR → $AB+C$

Analisa Tabel Kebenaran

A	B	C	AB	AB+C
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

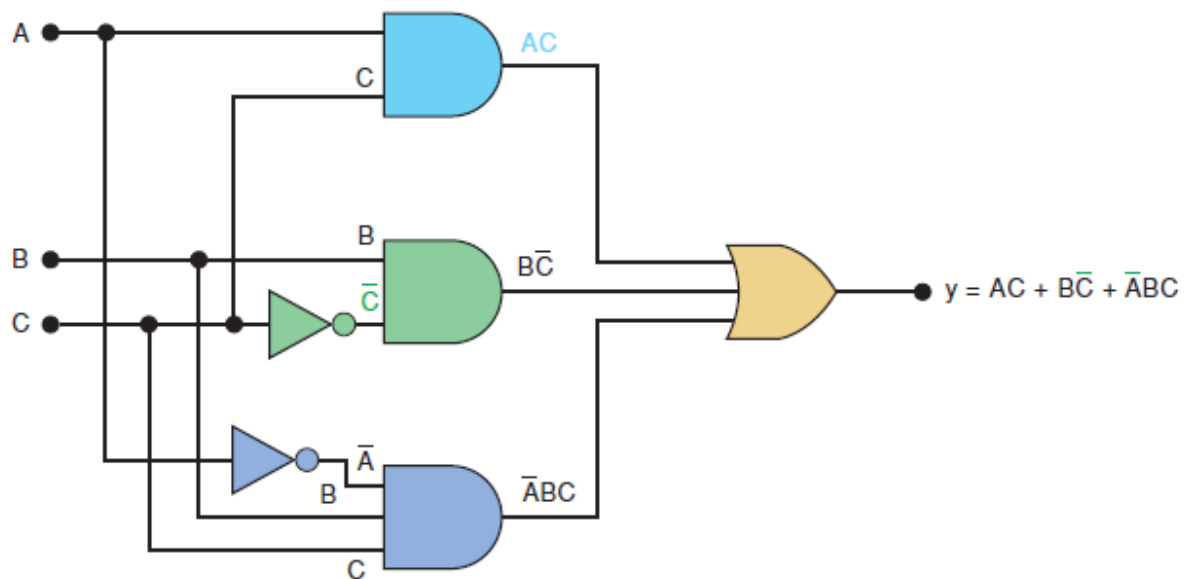
Implementasi Aljabar Boolean → Rangkaian Digital

Ketika operasi dari sebuah rangkaian dinyatakan dalam persamaan Boolean, maka kita dapat membuat rangkaian logikanya secara langsung berdasarkan persamaan tersebut.

Contoh

$$y = AC + B\bar{C} + \bar{A}BC$$

Persamaan Boolean diatas terdiri dari 3 terms yaitu $(AC, B\bar{C}, \bar{A}BC)$ yang di OR kan secara bersama. Persamaan diatas membutuhkan 3 gerbang AND, 2 gerbang NOT dan 1 gerbang OR.



Daftar Pustaka

Ronald J. Tocci, Neal S. Widmer, Gregory L. Moss, Digital Systems Principles and Applications TENTH EDITION, 2007, Pearson Education International



MODUL PERKULIAHAN

SISTEM DIGITAL

Aljabar Boolean

Fakultas

Ilmu Komputer

Program Studi

Teknik Informatika

Tatap Muka

04

Kode MK

15048

Disusun Oleh

Tim Dosen

Abstract

Modul ini membahas tentang hukum
hukum pada Aljabar Boolean

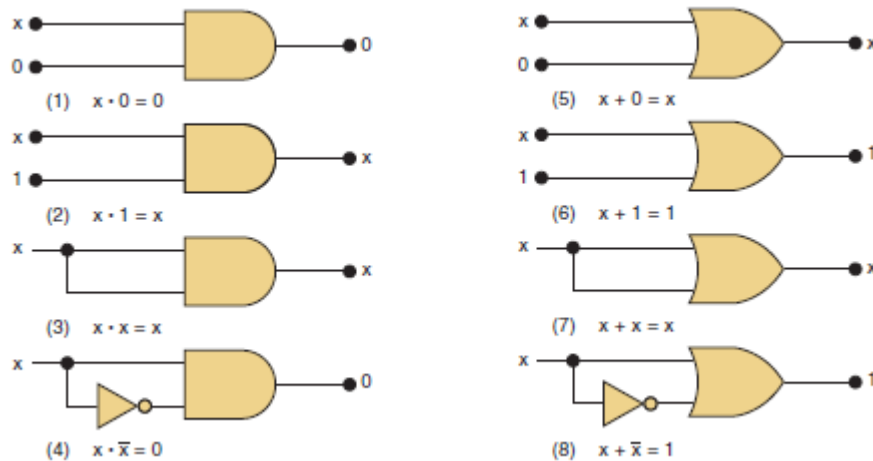
Kompetensi

- Mahasiswa diharapkan dapat
mengetahui hukum pada
aljabar boolea

Teorema Boolean

Teorema Boolean berisi aturan aturan yang dapat digunakan untuk menyederhanakan persamaan logika dan rangkaian logika.

Teori 1



Teori ini hanya melibatkan satu variable

1. Jika sebuah variabel di AND kan dengan 0, maka hasilnya adalah 0
2. Jika sebuah variable di AND kan dengan 1, maka hasilnya adalah variable itu sendiri.
3. Dapat diuji dengan kasus berikut
 $X = 0$, maka $0 \cdot 0 = 0$
 $X = 1$, maka $1 \cdot 1 = 1$
4. Jika setiap X di AND kan dengan invers nya maka akan menghasilkan 0
 $X = 0$, maka $0 \cdot 1 = 0$
 $X = 1$, maka $1 \cdot 0 = 0$
5. Jika 0 ditambahkan dengan apapun, maka tidak akan mempengaruhi hasil akhirnya, dan hasilnya akan sama dengan variable itu sendiri, baik dalam operasi biasa atau dalam OR.
6. Setiap variable yang di OR kan dengan 1, maka hasilnya akan selalu 1.
 $X = 0$, maka $0 + 1 = 1$
 $X = 1$, maka $1 + 1 = 1$
7. Dapat diuji dengan memeriksa kedua nilai dari X
 $X = 0$, maka $0 + 0 = 0$
 $X = 1$, maka $1 + 1 = 1$

8. Dapat diuji dengan cara yang sama

$X=0$, maka $0 + 1 = 1$

$X=1$, maka $1 + 0 = 1$

Teorema Multivariabel

Hukum Komutatif

$$(9) \quad x + y = y + x$$

$$(10) \quad x \cdot y = y \cdot x$$

Hukum komutatif mengindikasikan bahwa urutan dua variable dalam operasi OR atau AND tidak penting, karena hasilnya sama.

Hukum Asosiatif

$$(11) \quad x + (y + z) = (x + y) + z = x + y + z$$

$$(12) \quad x(yz) = (xy)z = xyz$$

Hukum asosiatif menyatakan bahwa kelompok variable dalam operasi OR atau AND tidak penting, karena hasilnya sama.

Hukum Distributif

$$(13a) \quad x(y + z) = xy + xz$$

$$(13b) \quad (w + x)(y + z) = wy + xy + wz + xz$$

Contoh :

$$A\bar{B}C + \bar{A}\bar{B}\bar{C} = \bar{B}(AC + \bar{A}\bar{C})$$

$$(14) \quad x + xy = x$$

$$(15a) \quad x + \bar{x}y = x + y$$

$$(15b) \quad \bar{x} + xy = \bar{x} + y$$

Pembuktian teorema 14

x	y	xy	x + xy
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Teorema Demorgan

Teorema ini ada 2 yaitu :

$$(16) \quad \overline{(x + y)} = \bar{x} \cdot \bar{y}$$

$$(17) \quad \overline{(x \cdot y)} = \bar{x} + \bar{y}$$

Teori 16 menyatakan bahwa penjumlahan dua variable (OR) yang diinvers, maka hasil nya sama dengan setiap variable diinvers dan di AND kan.

Teori 17 menyatakan bahwa penjumlahan dua variable (AND) yang diinvers, maka hasil nya sama dengan setiap variable diinvers dan di OR kan.

CONTOH

Example 1

$$\begin{aligned} z &= \overline{A + \overline{B \cdot C}} \\ &= \bar{A} \cdot \overline{\overline{B \cdot C}} \\ &= \bar{A} \cdot (\bar{\bar{B}} + \bar{\bar{C}}) \\ &= \bar{A} \cdot (B + \bar{C}) \end{aligned}$$

Example 2

$$\begin{aligned} \omega &= \overline{(A + BC) \cdot (D + EF)} \\ &= \overline{(A + BC)} + \overline{(D + EF)} \\ &= (\bar{A} \cdot \bar{BC}) + (\bar{D} \cdot \bar{EF}) \\ &= [\bar{A} \cdot (\bar{B} + \bar{C})] + [\bar{D} \cdot (\bar{E} + \bar{F})] \\ &= \bar{A}\bar{B} + \bar{A}\bar{C} + \bar{D}\bar{E} + \bar{D}\bar{F} \end{aligned}$$



UNIVERSALITY OF NAND GATES AND NOR GATES

NAND GATE

Semua persamaan Boolean terdiri dari berbagai kombinasi dari operasi dasar OR, AND dan INVERT. Setiap persamaan dapat diimplementasikan menggunakan gerbang OR, gerbang AND dan INVERTER. Sehingga memungkinkan untuk mengimplementasikan setiap persamaan logika menggunakan NAND gate, karena Gerbang NAND merupakan kombinasi yang dapat digunakan dalam operasi Boolean OR, AND dan INVERT

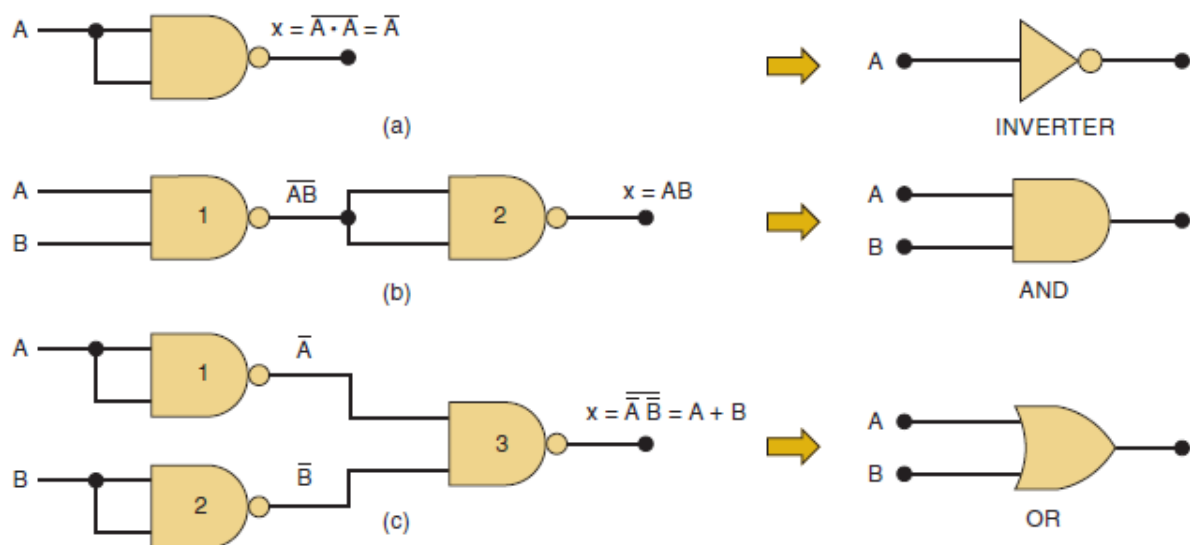


FIGURE 3-29 NAND gates can be used to implement any Boolean function.

Gambar a.

Gerbang NAND memiliki 2 input dimana kedua inputnya berasal dari variable A. dalam hal ini Gerbang NAND berindak sebagai INVERTER karena outputnya $x = \overline{A \cdot A} = \overline{A}$.

Gambar b.

2 Gerbang NAND yang terhubung sehingga menghasilkan operasi AND. Gerbang NAND kedua berfungsi sebagai INVERTER untuk merubah seperti fungsi AND yang diinginkan.

Gambar c.

Operasi OR dapat diimplementasikan menggunakan gerbang NAND yang dihubungkan seperti Gambar diatas. Gerbang NAND 1 dan 2 digunakan sebagai INVERTER untuk membalik input, sehingga hasil akhir output $x = \overline{\overline{A} \cdot \overline{B}}$, dapat disederhanakan menjadi $x = A + B$ menggunakan teori demorgan.

NOR GATE

Seperti halnya Gerbang NAND, dapat digunakan gerbang NOR.

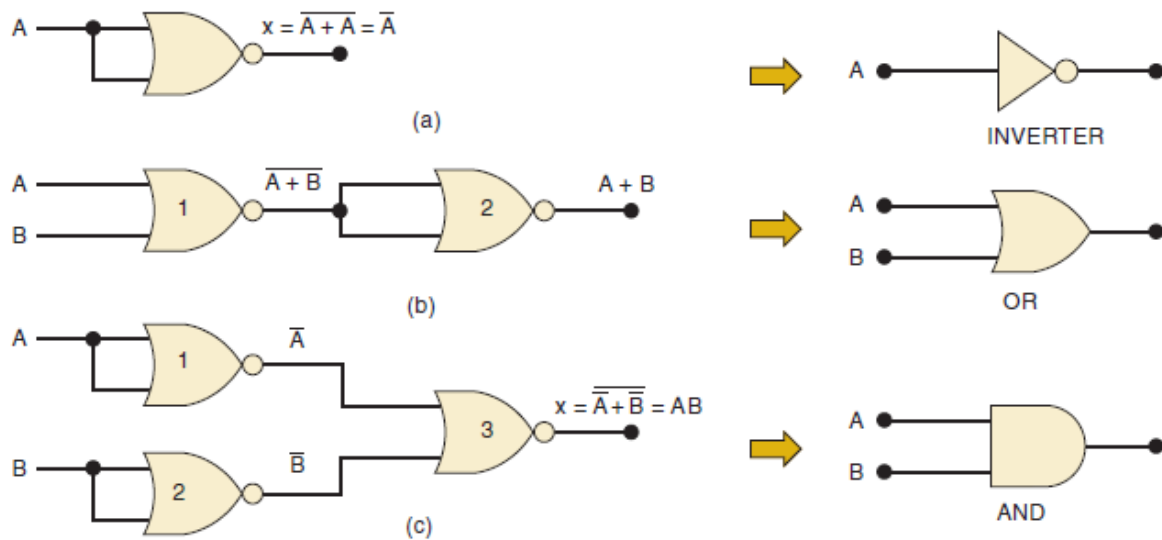


FIGURE 3-30 NOR gates can be used to implement any Boolean operation.

Gambar a.

Gerbang NOR memiliki 2 input dimana kedua inputnya berasal dari variable A. dalam hal ini Gerbang NOR bertindak sebagai INVERTER karena outputnya $x = \overline{A + A} = \overline{A}$.

Gambar b.

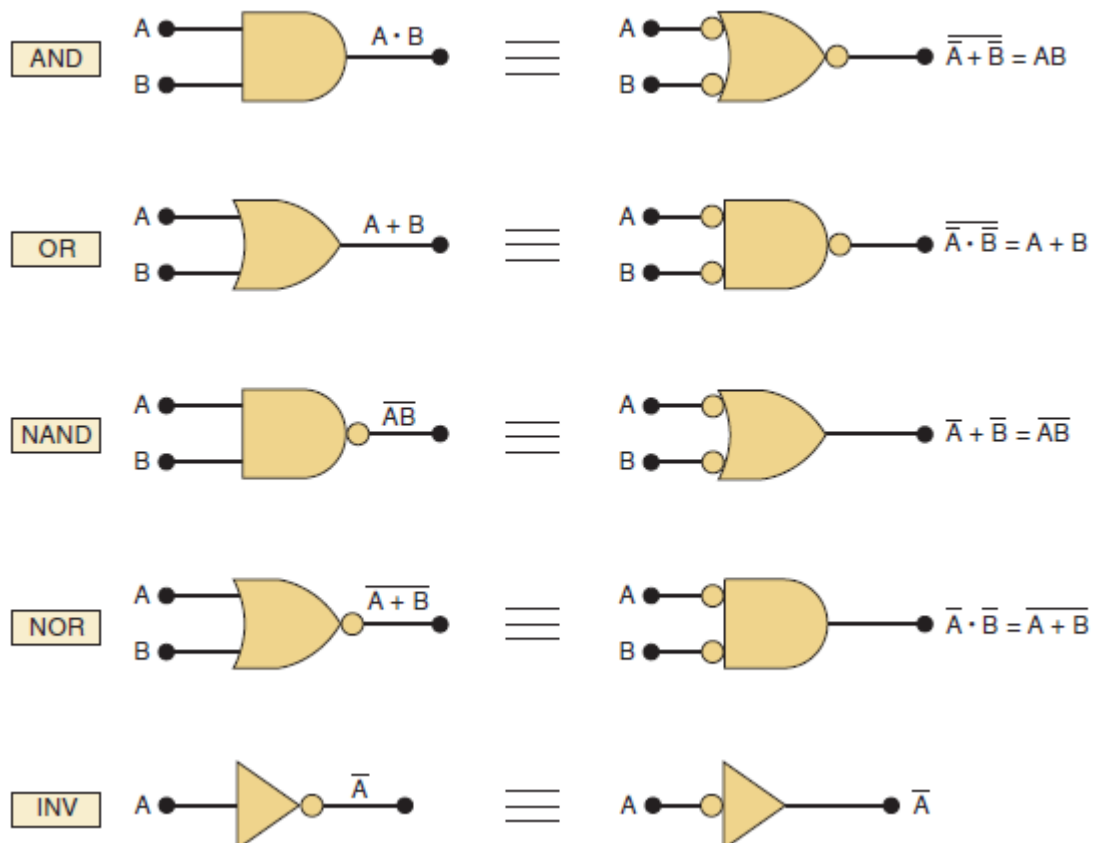
2 Gerbang NOR yang terhubung sehingga menghasilkan operasi OR. Gerbang NOR kedua berfungsi sebagai INVERTER untuk merubah seperti fungsi OR yang diinginkan.

Gambar c.

Operasi AND dapat diimplementasikan menggunakan gerbang NOR yang dihubungkan seperti Gambar diatas. Gerbang NOR 1 dan 2 digunakan sebagai INVERTER untuk membalik input, sehingga hasil akhir output $x = \overline{\overline{A} + \overline{B}}$, dapat disederhanakan menjadi $x = A \cdot B$ menggunakan teori demorgan.

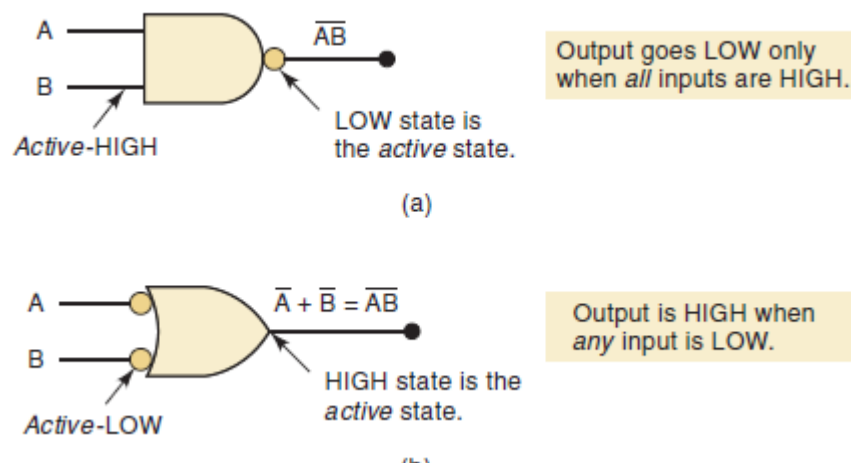
ALTERNATE LOGIC-GATE REPRESENTATIONS

5 gerbang logika dasar (AND, OR, INVERTER, NAND dan NOR) memiliki standar symbol yang digunakan pada diagram rangkaian logika. Tetapi terkadang kita menemukan beberapa rangkaian digital yang menggunakan symbol sendiri yang merupakan symbol logika alternative.



Logic-Symbol Interpretation

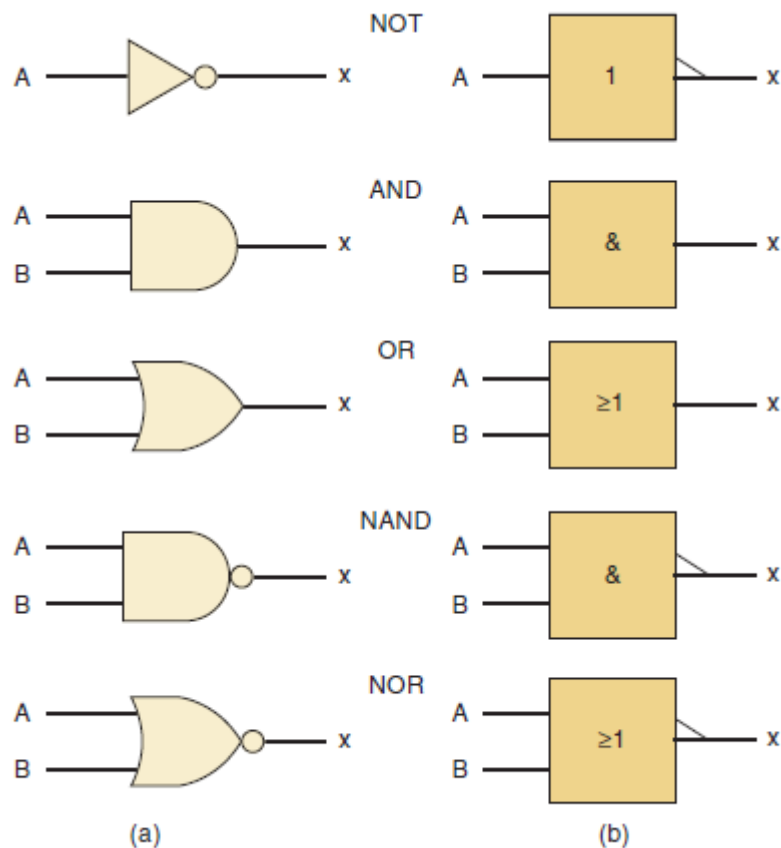
Ketika line input atau output pada sebuah rangkaian logika tidak memiliki **bubble** (lingkaran), maka line tersebut dikatakan aktif HIGH. Dan jika line input atau output memiliki bubble, maka line tersebut active LOW.




Gambar diatas menunjukkan symbol standard gerbang NAND. Symbol standar memiliki sebuah bubble pada outputnya dan tidak memiliki bubble pada inputnya. Gerbang NAND ini memiliki output active LOW dan input aktif HIGH. Operasi logika ini dinyatakan dengan symbol :

The output goes LOW only when all of the inputs are HIGH.

IEEE/ANSI STANDARD LOGIC SYMBOLS



- Simbol segi empat yang menggunakan sebuah segitiga () sama dengan bubble pada symbol tradisional yang menyatakan invers dari logic level. Kehadiran segitiga menunjukkan apakah input atau output aktif LOW atau aktif HIGH
- Notasi khusus yang berada didalam symbol segi empat menjelaskan relasi logika antara input dan output :
 - "1" → menunjukkan Inverter dan hanya memiliki 1 input.
 - "&" → symbol AND yang berarti output aktif HIGH, jika semua inputnya aktif HIGH
 - ≥ 1 → Gerbang OR, dimana output aktif HIGH jika salah satu atau semua inputnya aktif HIGH.
- Simbol segiempat NAND dan NOR sama dengan symbol AND dan OR, dengan menambahkan segitiga invers pada outputnya.

Daftar Pustaka

Ronald J. Tocci, Neal S. Widmer, Gregory L. Moss, Digital Systems Principles and Applications TENTH EDITION, 2007, Pearson Education International



MODUL PERKULIAHAN

SISTEM DIGITAL

SOP dan POS

Fakultas
Ilmu Komputer

Program Studi
Teknik Informatika

Tatap Muka

05

Kode MK
15048

Disusun Oleh
Tim Dosen

Abstract

Modul ini membahas tentang penyederhanaan dengan Sum of Product dan Product of SUM

Kompetensi

- Mahasiswa diharapkan melakukan penyederhanaan persamaan dengan menggunakan SOP dan POS

SUM-OF-PRODUCTS FORM

Metoda penyederhanaan rangkaian logika salah satunya adalah SOP (sum of products).

Contoh:

1. $ABC + \overline{A}\overline{B}\overline{C}$
2. $AB + \overline{A}\overline{B}\overline{C} + \overline{C}\overline{D} + D$
3. $\overline{A}B + \overline{C}\overline{D} + EF + GK + H\overline{L}$

Setiap pernyataan dalam bentuk sum of products terdiri dari dua atau lebih operasi AND yang semuanya di OR kan.

PRODUCTS OF SUM FORM

Bentuk persamaan logika juga menggunakan POS (products of sum). Terdiri dari 2 atau lebih operasi OR yang kemudian di AND kan.

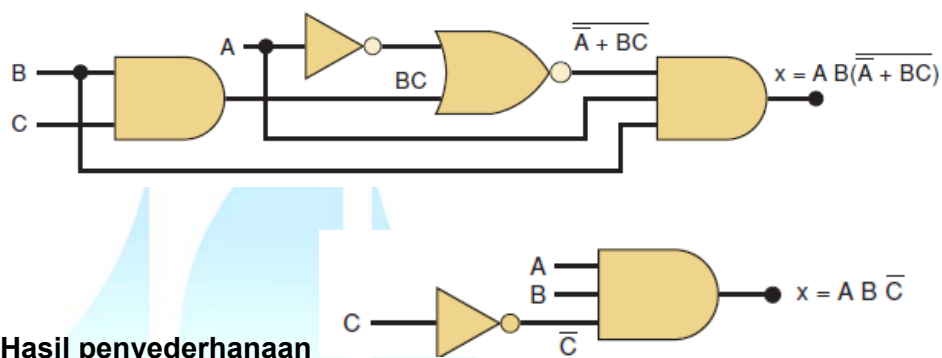
Contoh:

1. $(A + \overline{B} + C)(A + C)$
2. $(A + \overline{B})(\overline{C} + D)F$
3. $(A + C)(B + \overline{D})(\overline{B} + C)(A + \overline{D} + \overline{E})$

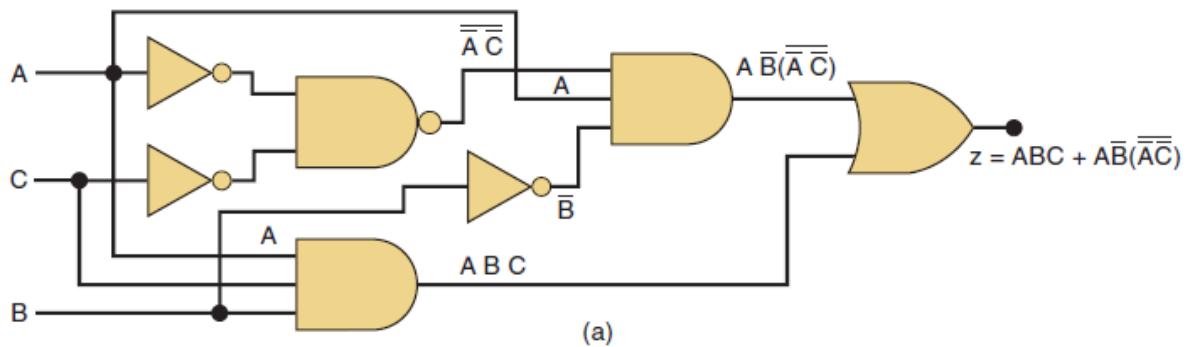
ALGEBRAIC SIMPLIFICATION

Sebuah pernyataan Boolean dapat kita sederhanakan bentuknya dengan menggunakan banyak cara, pernyataan yang baru digunakan untuk mengimplementasikan sebuah rangkaian yang sama dengan rangkaian asli, tetapi memiliki gerbang dan koneksi yang lebih sedikit.

Contoh



Contoh:



Persamaan : $z = ABC + AB\bar{A}\bar{C}$

Langkah langkah :

Menggunakan teori demorgan → menghasilkan bentuk SOP

$$\begin{aligned}
 z &= ABC + AB\bar{A}\bar{C} && [\text{theorem (17)}] \\
 &= ABC + AB(A + C) && [\text{cancel double inversions}] \\
 &= ABC + AB\bar{A} + AB\bar{C} && [\text{multiply out}] \\
 &= ABC + AB + AB\bar{C} && [A \cdot A = A]
 \end{aligned}$$

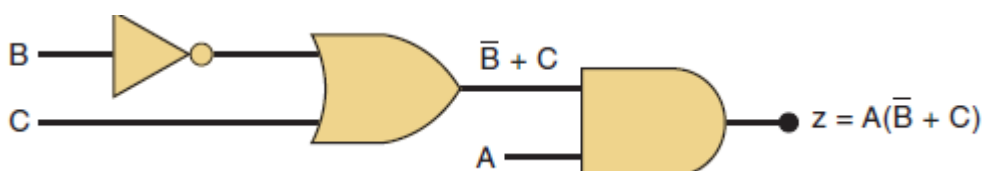
$$z = AC(B + \bar{B}) + AB$$

Karena $B + \bar{B} = 1$ maka

$$\begin{aligned}
 z &= AC(1) + AB \\
 &= AC + AB
 \end{aligned}$$

$$z = A(C + \bar{B})$$

Rangkaian Sederhana



Complete Design Procedure

Setiap masalah logika dapat diselesaikan dengan langkah langkah sebagai berikut :

1. Interpret the problem and set up a truth table to describe its operation.
2. Write the AND (product) term for each case where the output is 1.
3. Write the sum-of-products (SOP) expression for the output.
4. Simplify the output expression if possible.
5. Implement the circuit for the final, simplified expression.

Contoh :

Rancanglah sebuah rangkaian logika yang memiliki 3 input A, B dan C, dimana output akan menjadi HIGH hanya pada saat semua input mayoritas adalah HIGH.

Solusi

1. Menterjemahkan masalah dan membuat table kebenaran.

Output = 1 apabila mayoritas input adalah 1 (2/lebih input), dan yang lain output = 0

A	B	C	x
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

2. Membuat persamaan AND untuk output = 1

A	B	C	x
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$\rightarrow \overline{A}BC$

$\rightarrow A\overline{B}C$

$\rightarrow AB\overline{C}$

$\rightarrow ABC$

- Menulis persamaan sum of products untuk output

$$x = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

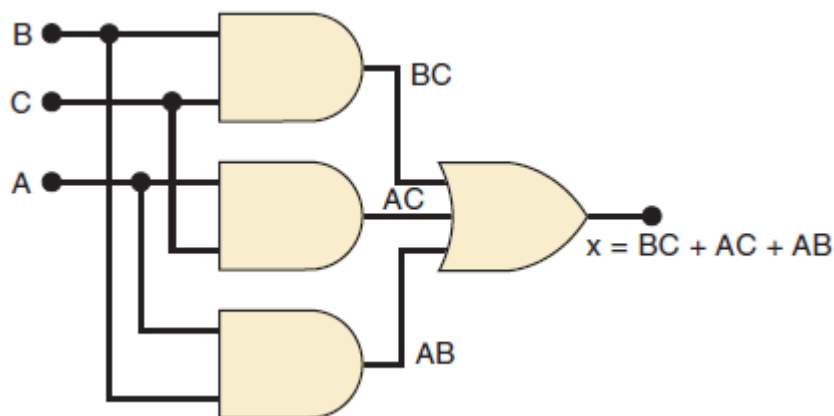
- Menyederhanakan persamaan

$$x = \overline{A}BC + ABC + A\overline{B}C + ABC + AB\overline{C} + ABC$$

$$x = BC(\overline{A} + A) + AC(\overline{B} + B) + AB(\overline{C} + C)$$

$$x = BC + AC + AB$$

- Implementasi rangkaian untuk persamaan terakhir



KARNAUGH MAP METHOD

Karnaugh map (K map) merupakan tool grafis yang digunakan untuk menyederhanakan persamaan logika atau untuk melakukan konversi sebuah table kebenaran menjadi rangkaian logika yang lebih sederhana.

Contoh

Peta karnaugh 2 variabel

A	B	X
0	0	1 → $\overline{A}\overline{B}$
0	1	0
1	0	0
1	1	1 → AB

$$\left\{ x = \overline{A}\overline{B} + AB \right\}$$

	\overline{B}	B
\overline{A}	1	0
A	0	1

Peta karnaugh 3 variabel

A	B	C	X
0	0	0	1 → $\overline{A}\overline{B}\overline{C}$
0	0	1	1 → $\overline{A}\overline{B}C$
0	1	0	1 → $\overline{A}B\overline{C}$
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1 → $AB\overline{C}$
1	1	1	0

$$\left\{ \begin{aligned} X = & \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C \\ & + \overline{A}B\overline{C} + AB\overline{C} \end{aligned} \right\}$$

(b)

	\overline{C}	C
$\overline{A}\overline{B}$	1	1
$\overline{A}B$	1	0
AB	1	0
$A\overline{B}$	0	0

Peta karnaugh 4 variabel

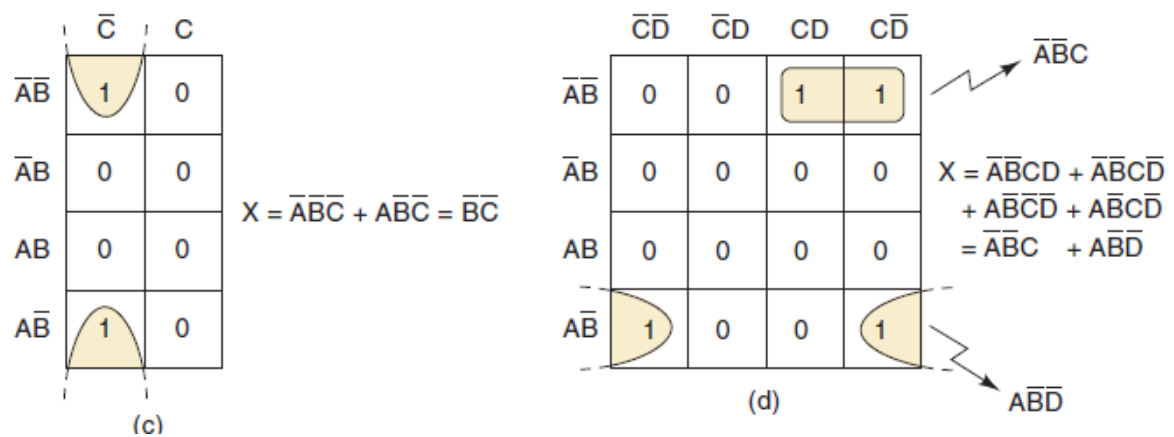
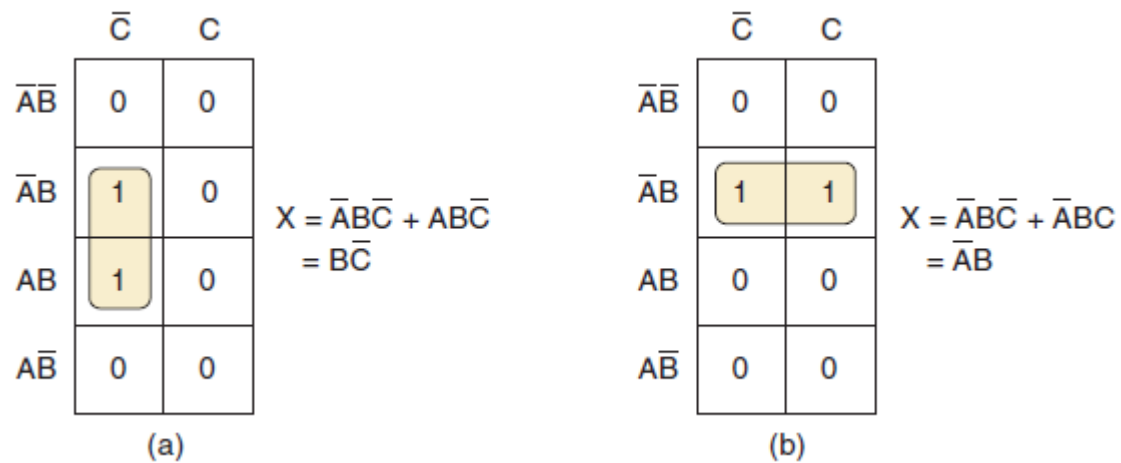
A	B	C	D	X
0	0	0	0	0
0	0	0	1	1 → $\overline{A}\overline{B}\overline{C}D$
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1 → $\overline{A}B\overline{C}D$
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1 → $AB\overline{C}D$
1	1	1	0	0
1	1	1	1	1 → $ABCD$

$$\left\{ \begin{aligned} X = & \overline{A}\overline{B}\overline{C}D + \overline{A}B\overline{C}D \\ & + AB\overline{C}D + ABCD \end{aligned} \right\}$$

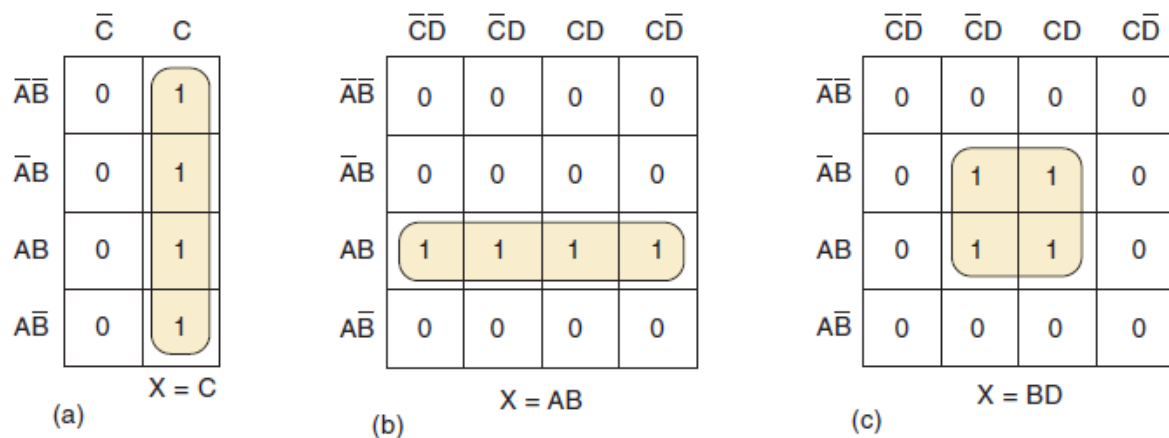
	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\overline{B}$	0	1	0	0
$\overline{A}B$	0	1	0	0
AB	0	1	1	0
$A\overline{B}$	0	0	0	0

Persamaan output X dapat disederhanakan dengan cara membuat kotak pada Kmap untuk yang berisi 1. Proses ini disebut dengan **looping**

Looping → 2



Looping → 4



	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	0	0	0	0
AB	1	0	0	1
$A\bar{B}$	1	0	0	1

(d) $X = A\bar{D}$

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	0	0	1
$\bar{A}B$	0	0	0	0
AB	0	0	0	0
$A\bar{B}$	1	0	0	1

(e) $X = \bar{B}\bar{D}$

Looping → 8

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	1	1	1	1
AB	1	1	1	1
$A\bar{B}$	0	0	0	0

(a) $X = B$

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	1	0	0
$\bar{A}B$	1	1	0	0
AB	1	1	0	0
$A\bar{B}$	1	1	0	0

(b) $X = \bar{C}$

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	1	1	1
$\bar{A}B$	0	0	0	0
AB	0	0	0	0
$A\bar{B}$	1	1	1	1

(c) $X = \bar{B}$

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	0	0	1
$\bar{A}B$	1	0	0	1
AB	1	0	0	1
$A\bar{B}$	1	0	0	1

(d) $X = \bar{D}$

Prosedure penyederhanan

- Step 1** Construct the K map and place 1s in those squares corresponding to the 1s in the truth table. Place 0s in the other squares.
- Step 2** Examine the map for adjacent 1s and loop those 1s that are *not* adjacent to any other 1s. These are called *isolated* 1s.
- Step 3** Next, look for those 1s that are adjacent to only one other 1. Loop *any* pair containing such a 1.
- Step 4** Loop any octet even if it contains some 1s that have already been looped.
- Step 5** Loop any quad that contains one or more 1s that have not already been looped, *making sure to use the minimum number of loops*.
- Step 6** Loop any pairs necessary to include any 1s that have not yet been looped, *making sure to use the minimum number of loops*.
- Step 7** Form the OR sum of all the terms generated by each loop.

Contoh

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0 ₁	0 ₂	0 ₃	1 ₄
$\bar{A}B$	0 ₅	1 ₆	1 ₇	0 ₈
AB	0 ₉	1 ₁₀	1 ₁₁	0 ₁₂
$A\bar{B}$	0 ₁₃	0 ₁₄	1 ₁₅	0 ₁₆

$$X = \underbrace{\bar{A}\bar{B}C\bar{D}}_{\text{loop 4}} + \underbrace{ACD}_{\text{loop 11, 15}} + \underbrace{BD}_{\text{loop 6, 7, 10, 11}}$$

(a)

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0 ₁	0 ₂	1 ₃	0 ₄
$\bar{A}B$	1 ₅	1 ₆	1 ₇	1 ₈
AB	1 ₉	1 ₁₀	0 ₁₁	0 ₁₂
$A\bar{B}$	0 ₁₃	0 ₁₄	0 ₁₅	0 ₁₆

$$X = \underbrace{\bar{A}B}_{\text{loop 5, 6, 7, 8}} + \underbrace{B\bar{C}}_{\text{loop 5, 6, 9, 10}} + \underbrace{\bar{A}CD}_{\text{loop 3, 7}}$$

(b)

	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\overline{B}$	0 1	1 2	0 3	0 4
$\overline{A}B$	0 5	1 6	1 7	1 8
AB	1 9	1 10	1 11	0 12
$A\overline{B}$	0 13	0 14	1 15	0 16

$$X = \underbrace{ABC}_{9, 10} + \underbrace{\overline{A}\overline{C}D}_{2, 6} + \underbrace{\overline{A}BC}_{7, 8} + \underbrace{ACD}_{11, 15}$$

(c)

Daftar Pustaka

Ronald J. Tocci, Neal S.Widmer, Gregory L. Moss, Digital Systems Principles and Applications TENTH EDITION, 2007, Pearson Education International



MODUL PERKULIAHAN

SISTEM DIGITAL

FLIP FLOP

Fakultas
Ilmu Komputer

Program Studi
Teknik Informatika

Tatap Muka

06

Kode MK
15048

Disusun Oleh
Tim Dosen

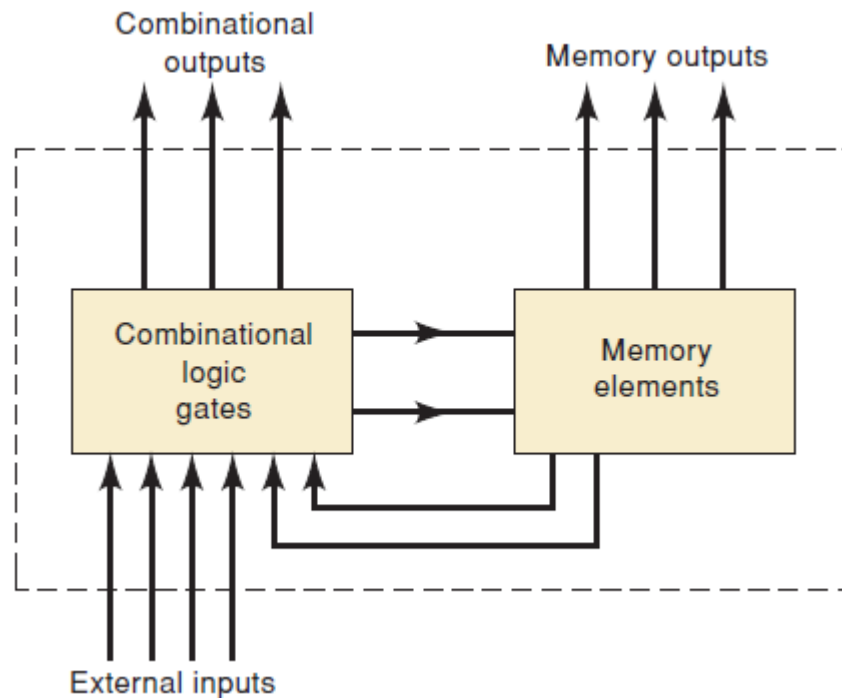
Abstract

Modul ini membahas tentang Flip Flop

Kompetensi

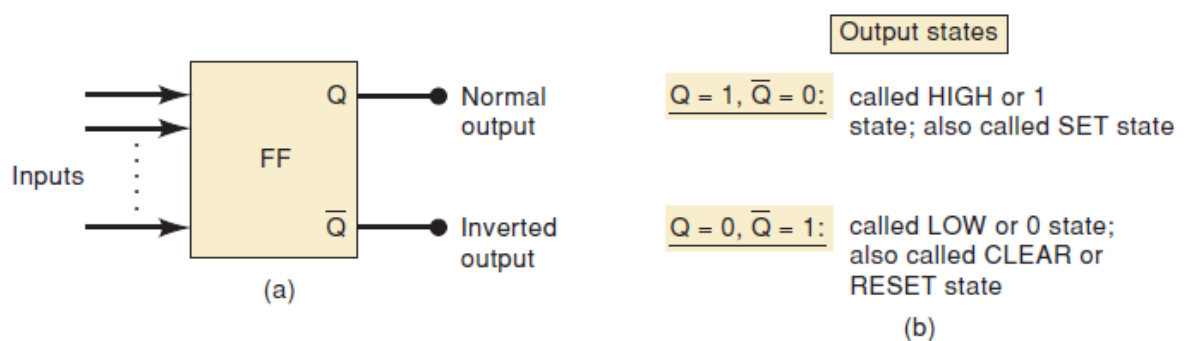
- Mahasiswa diharapkan memahami flip flop
-

Gambar 6.1 menunjukkan sebuah blok diagram dari system digital yang dikombinasikan dengan perangkat memory.



Gambar 6.1 Diagram system Digital

Elemen memory yang paling penting adalah flip flop, yang dibuat dari rangkaian gerbang gerbang logika. Walaupun sebuah gerbang logika tidak memiliki kemampuan penyimpanan, beberapa dapat dihubungkan sebagai cara untuk menyimpan informasi. Beberapa gerbang yang berbeda disusun dan digunakan untuk menghasilkan flip flop (FF).



Gambar 6.2 General Flip Flop

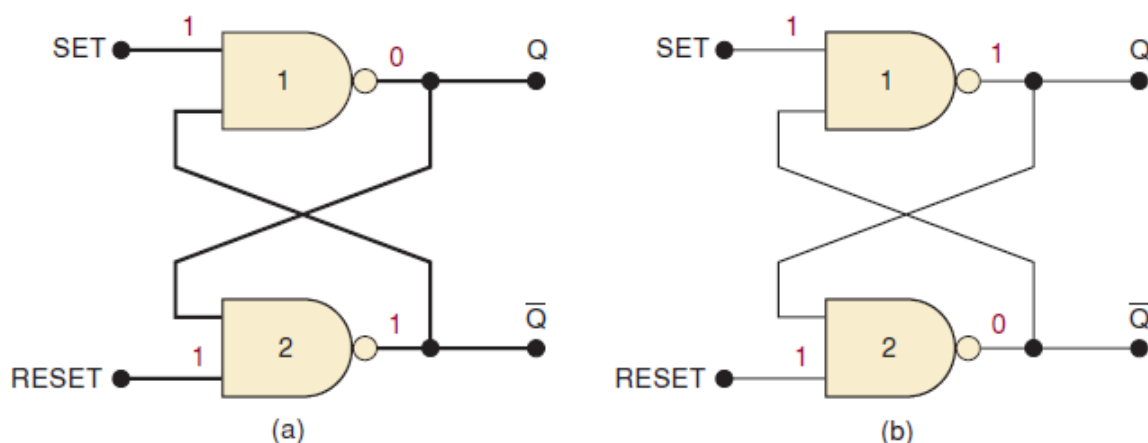
Gambar 6.2 a, merupakan symbol yang digunakan oleh flip flop. Menunjukkan 2 input yang diberi label Q dan \bar{Q} . Q/\bar{Q} merupakan output FF. Q output disebut sebagai normal output FF dan \bar{Q} merupakan invers output FF. Contoh Sebuah FF berada pada state HIGH (1), maka $Q=1$, jika FF pada state LOW (0) maka $Q=0$. Dan \bar{Q} memiliki nilai sebaliknya.

Gambar 6.2b merupakan 2 kemungkinan operasi pada FF. Untuk HIGH atau state 1 ($Q=1/\bar{Q}=0$) disebut sebagai SET state. Pada saat input ke FF menyebabkan $Q=1$, maka disebut setting FF, sehingga FF di set. Dengan cara yang sama LOW atau 0 state ($Q=0/\bar{Q}=1$) disebut sebagai CLEAR atau RESET state. Pada saat input ke FF menyebabkan $Q=0$, maka disebut clearing atau resetting FF, sehingga FF di reset. **FF memiliki SET input dan/atau CLEAR (RESET) input yang digunakan untuk drive FF menjadi state output.**

NAND GATE LATCH

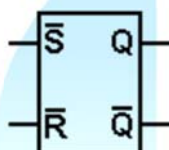
Rangkaian dasar FF dapat dibangun dari 2 gerbang NAND atau 2 gerbang NOR. Versi gerbang NAND disebut sebagai NAND gate latch atau secara sederhana latch (Gambar 6.3a). 2 gerbang NAND di cross-coupled sehingga output NAND-1 dihubungkan ke salah satu input dari NAND-2 dan sebaliknya. Output gerbang diberi label Q dan \bar{Q} dan disebut dengan latch output. Pada kondisi normal, output akan selalu di invers dengan yang lain. Ada 2 latch input : SET input yaitu input set Q sama dengan state 1; RESET input akan resets Q ke 0 state.

Versi kedua gerbang NAND (Gambar 6.3b). , dimana $Q = 1$ dan $\bar{Q} = 0$. HIGH dari gerbang NAND-1 menghasilkan LOW pada output NAND-2, dan output NAND-1 tetap HIGH. Ada 2 kemungkinan output yaitu SET = RESET = 1.



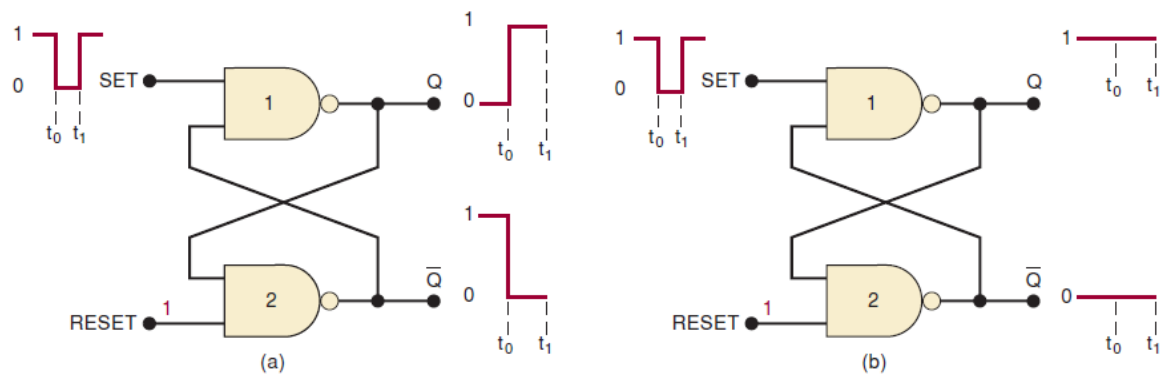
Gambar 6.3 NAND Latch

Simbol



Setting the Latch (FF)

Berdasarkan Gambar 6.4 dibawah, memperlihatkan proses SET dan RESET dari Latch



Gambar 6.4 SET dan RESET latch

Keterangan gambar 6.4a.

SET terjadi pada saat :

Pulsa LOW pada waktu t_0 , Q menjadi HIGH, dan \bar{Q} menjadi LOW dan NAND-1 memiliki 2 input LOW.

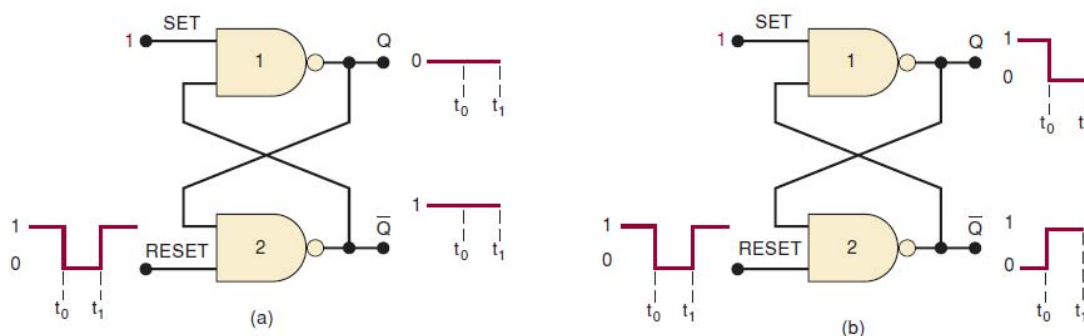
Pada saat SET kembali ke state 1 pada waktu t_1 , output NAND-1 menjadi HIGH, dan output NAND-2 tetap LOW.

Keterangan gambar 6.4b.

Jika $\bar{Q} = 0$ maka output NAND-1 HIGH karena pulsa LOW pada SET tidak akan merubah apapun. Sehingga pada saat SET = HIGH, latch output tetap berada pada $Q = 1$ dan $\bar{Q} = 0$

ReSetting the Latch (FF)

Jika input RESET = LOW dan SET = HIGH. Pada saat $Q = 0$ dan $\bar{Q} = 1$



Gambar 6.5 RESETTING LATCH

Tabel Kebenaran

Set	Reset	Output
1	1	No change
0	1	$Q = 1$
1	0	$Q = 0$
0	0	Invalid*

*Produces $Q = \bar{Q} = 1$.

1. SET = RESET = 1

Kondisi ini normal state, dan tidak membuat perubahan apapun terhadap state output. Q dan \bar{Q} akan tetap sama state nya dengan sebelumnya.

2. SET = 0, RESET = 1

Menyebabkan output berubah ke state $Q=1$, disebut sebagai setting latch

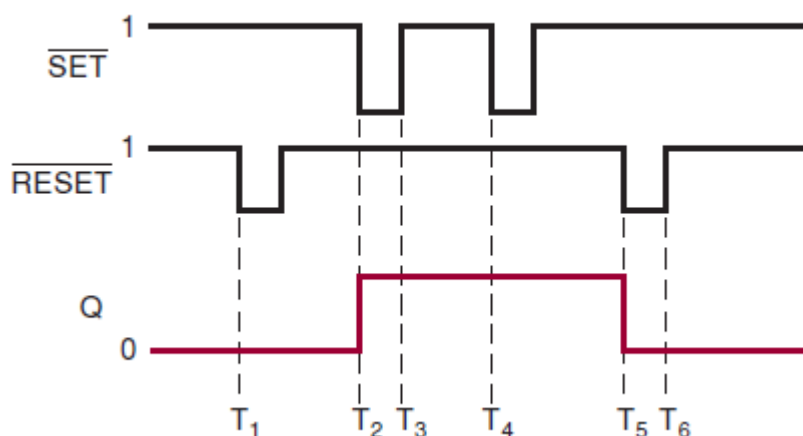
3. SET = 1, RESET = 0

Kondisi ini menghasilkan state $Q = 0$, dan disebut sebagai clearing/resetting latch

4. SET=RESET=0

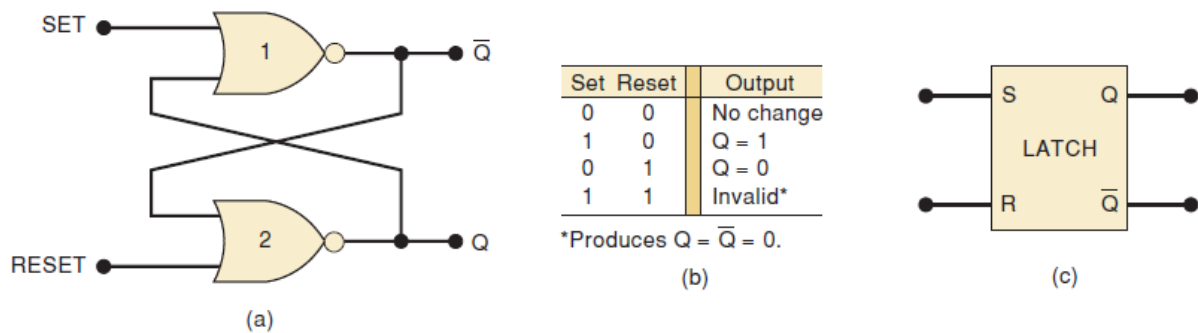
Kondisi ini mencoba untuk SET dan CLEAR latch pada waktu yang bersamaan, dan menghasilkan $Q = \bar{Q} = 1$. Jika input dikembalikan ke 1 secara bersamaan, maka hasilnya adalah tidak bisa diprediksi. Kondisi input ini tidak dipergunakan.

CONTOH



NOR GATE LATCH

2 gerbang NOR yang di cross-coupled dapat digunakan sebagai NOR gate latch. Gambar 6.6(a) mirip dengan NAND latch kecuali output Q dan \bar{Q} dengan posisi terbalik.



Gambar 6.6 NOR GATE LATCH

1. SET = RESET = 0

Kondisi ini normal state, dan tidak membuat perubahan apapun terhadap state output. Q dan \bar{Q} akan tetap sama state nya dengan sebelumnya.

2. SET = 1, RESET = 0

Menyebabkan output berubah ke state $Q=1$, disebut sebagai setting latch

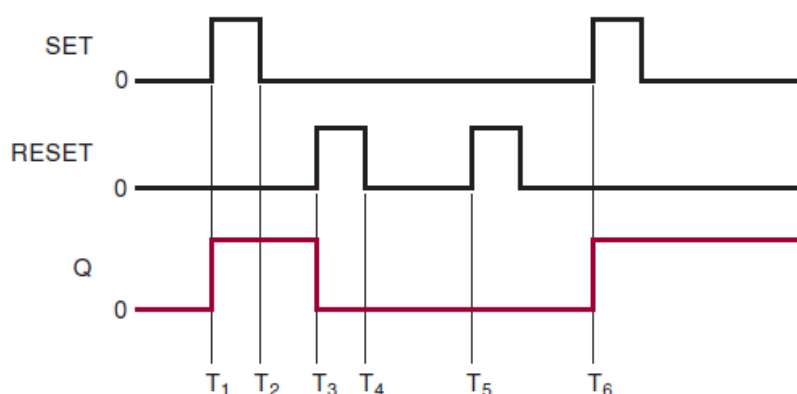
3. SET = 0, RESET = 1

Kondisi ini menghasilkan state $Q = 0$, dan disebut sebagai clearing/resetting latch

4. SET=RESET=1

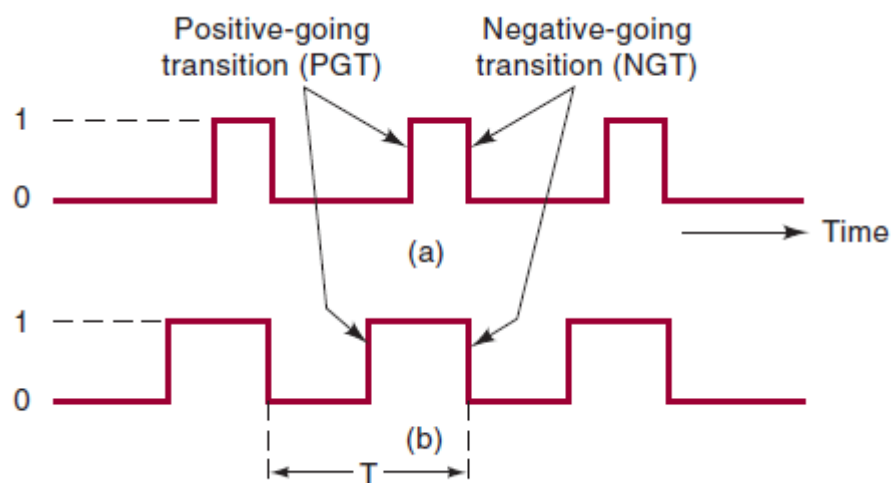
Kondisi ini mencoba untuk SET dan CLEAR latch pada waktu yang bersamaan, dan menghasilkan $Q = \bar{Q} = 0$. Jika input dikembalikan ke 0 secara bersamaan, maka hasilnya adalah tidak bisa diprediksi. Kondisi input ini tidak dipergunakan.

Contoh



CLOCK SIGNALS AND CLOCKED FLIP-FLOPS

System digital dapat beroperasi secara asinkron atau sinkron. Pada system asinkron, output rangkaian logika dapat merubah state setiap saat satu atau lebih input berubah. Pada system sinkron, waktu setiap output dapat merubah state ditentukan oleh sebuah sinyal yang disebut dengan **clock**. Sinyal clock merupakan sebuah pulsa segiempat seperti pada Gambar 6.7 berikut.



Gambar 6.7 Sinyal Clock

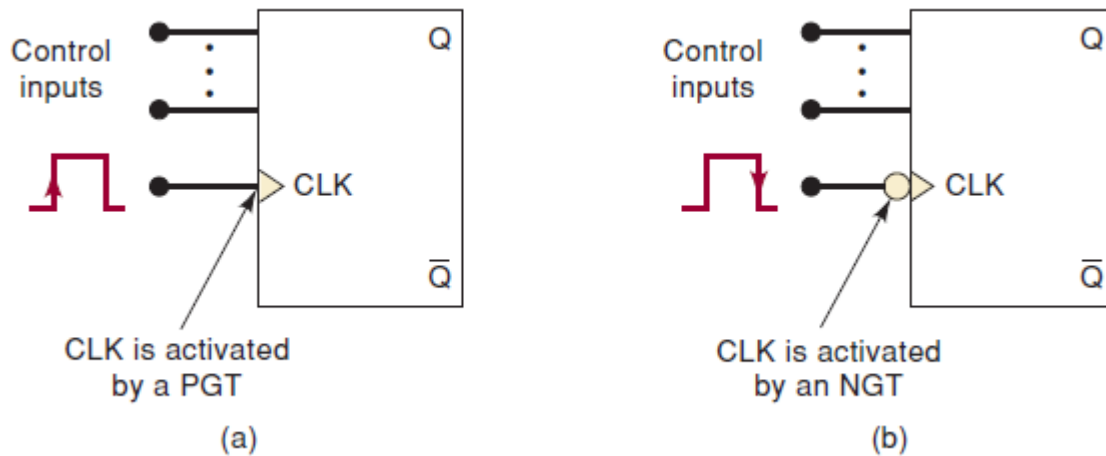
Sinyal clock didistribusikan ke semua bagian system, dan sebagian (if not all) output system dapat merubah state hanya pada saat clock membuat transisi. Transisi disebut sebagai *edges*.

Pada saat clock berubah dari 0 ke 1, maka disebut sebagai **positive-going transition (PGT)**; pada saat clock berubah dari 1 ke 0, maka disebut sebagai **negative-going transition (NGT)**.

Sebagian system digital merupakan sinkron karena rangkaian sinkron lebih mudah untuk melakukan perancangan dan troubleshoot. Hal ini disebabkan rangkaian output dapat berubah hanya pada waktu spesifik atau disinkronkan dengan transisi clock-signal.

Sinkronisasi dari sinyal clock juga digunakan pada **clocked flip-flops** yang digunakan untuk merubah states pada satu atau lebih clock's transitions.

Clocked Flip-Flops



Gambar 6.8 Clock FF

Karakteristik Clocked FF

1. Clock FF memiliki sebuah clock input yang diberi label CLK, CK atau CP. Normalnya menggunakan CLK (Gambar 6.8a). Pada sebagian Clock FF, CLK input merupakan edge triggered, yang akan diaktifkan oleh sebuah sinyal transisi. Hal ini diindikasikan oleh adanya segitiga kecil pada CLK input.

Sebuah FF dengan segitiga kecil pada CLK Input mengindikasikan bahwa input ini hanya diaktifkan pada saat positive going transition (PGT) terjadi, dan tidak ada bagian input pulse yang akan mempengaruhi CLK Input.

FF symbol memiliki sebuah bubble (Gambar 6.8b). seperti segitiga pada CLK Input. Yang berarti CLK input hanya diaktifkan pada saat Negative going Transition terjadi, dan juga tidak ada bagian input lainnya yang akan mempengaruhi CLK Input

2. Clock FF juga memiliki satu atau lebih input control yang dapat memiliki beberapa nama, tergantung kepada operasinya. Control input tidak memiliki efek pada Q sampai aktif transisi clock terjadi.

Contoh :

Control input pada FF (Gambar 6.8a) tidak memiliki efek pada Q sampai PGT clock signal terjadi, sedangkan untuk Gambar 6.8b tidak memiliki efek sampai NGT clock signal terjadi

Daftar Pustaka

Ronald J. Tocci, Neal S. Widmer, Gregory L. Moss, Digital Systems Principles and Applications TENTH EDITION, 2007, Pearson Education International



MODUL PERKULIAHAN

SISTEM DIGITAL

FLIP FLOP Bagian 2

Fakultas
Ilmu Komputer

Program Studi
Teknik Informatika

Tatap Muka

07

Kode MK
15048

Disusun Oleh
Tim Dosen

Abstract

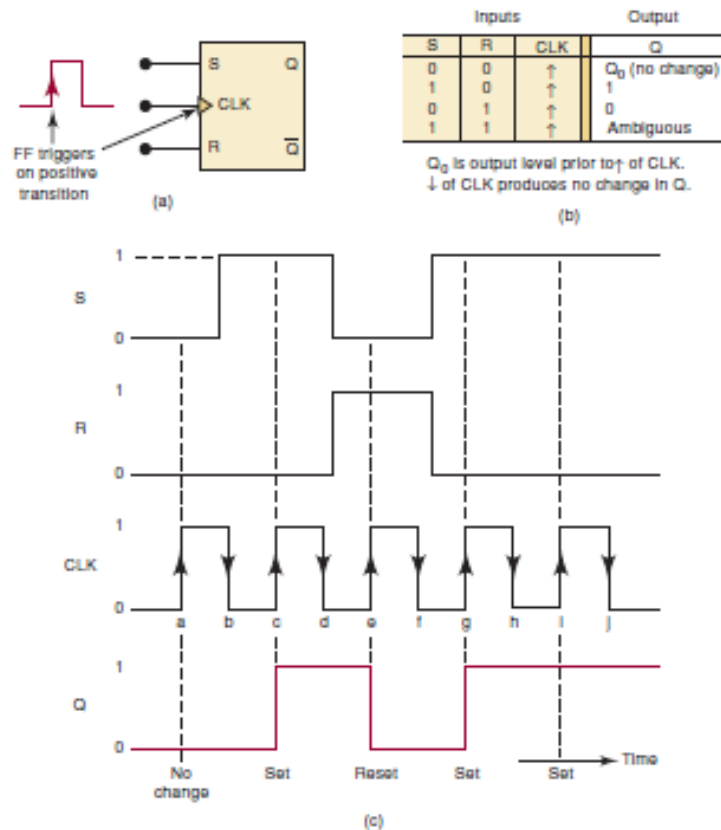
Modul ini membahas tentang Flip Flop

Kompetensi

- Mahasiswa diharapkan memahami flip flop
-

Jenis jenis Flip Flop

CLOCKED S-R FLIP-FLOP



Gambar 7.1 RS Flip Flop

Gambar 7.1 a

Merupakan symbol dari clocked S-R Flip Flop, ditrigger oleh Positive going edge dari sinyal clock. Berarti FF dapat merubah statenya hanya pada saat sebuah sinyal diberikan ke clock inputnya dan membuat transisi dari 0 ke 1.

Gambar 7.1 b

Tabel kebenaran yang menunjukkan bagaimana FF output akan memberikan respon ke PGT pada CLK Input untuk berbagai kombinasi input S dan R.

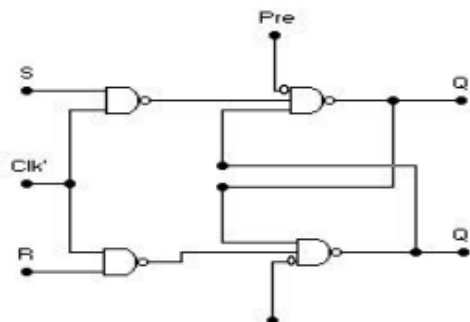
- Up arrow menunjukkan bahwa PGT diperlukan pada saat CLK
- Label menentukan level pada Q sebelumnya ke PGT

.Gambar 7.1 c

- Gelombang digital yang memperlihatkan operasi dari clocked RS flip flop.
- Analisa gelombang
 - Pada awalnya semua input adalah 0 dan output Q diasumsikan sebagai 0, sehingga $Q_0 = 0$
 - Pada saat PGT pulsa clock pertama terjadi (point a), input S dan R = 0, sehingga FF tidak mempengaruhi dan tetap pada posisi $Q=0$ ($Q = Q_0 = 0$)
 - Pada saat PGT terjadi pada pulsa clock kedua (point c), input S menjadi HIGH, dan R tetap LOW. FF menyet state 1 pada rising edge dari pulsa clock ini
 - Pada saat pulsa clock ketiga membuat transisi positif (point e), dan $S=0$ $R=1$, sehingga menyebabkan FF menjadi clear ke state 0.
 - Pulsa keempat menyet FF sekali lagi ke state $Q=1$ (point g) karena $S=1$, $R=0$ pada saat Positive edge occurs.
 - Pulsa kelima mendapatkan $S=1$, $R=0$ pada saat dia membuat positive going transition. $Q = \text{HIGH}$, maka tetap berada pada state yang sama.
 - $S=R=1$ tidak digunakan karena hasilnya adalah kondisi ragu.

Preset dan Clear pada R-S Flip-flop

Dengan penambahan Preset (Pre) dan Clear (Clr), seperti pada Gambar 7.2. yang pada ujungnya diberi tanda (inverter), rangkaian dapat dikendalikan dengan masukan tak sinkron. Masukan Pre dan Clr, dapat digunakan untuk penghapusan atau pengesetan data keluaran, sesuai Tabel 7.1 a. Pengesetan langsung $Q=1$ dapat dilakukan dengan memberi masukan $\text{Pre}=1$ dan $\text{Clr}=0$, tanpa memperdulikan masukan R dan S b. Penghapusan langsung $Q=0$ dilakukan dengan memberi masukan $\text{Pre}=0$ dan $\text{Clr}=1$, tanpa memperdulikan masukan R dan S c. Rangkaian dalam keadaan modus operasi, bila masukan $\text{Pre}=\text{Clr}=0$



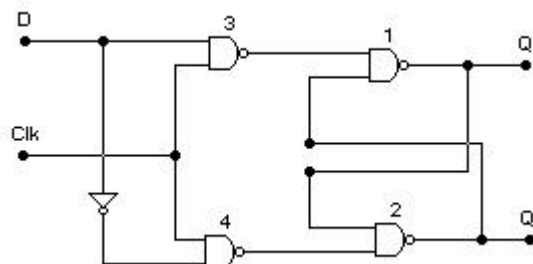
Gambar 7.2 Clocked FF

Tabel 7.1 Set dan Reset

Tabel Kebenaran							
Pre	Clr	R	S	Clk	Q	Q'	Mode
1	1	x	x	x	1	1	Invalid**
1	0	x	x	0	1	0	Set**
0	1	x	x	0	0	1	Reset**
0	0	x	x	0	Q_t	Q'_t	Memori
0	0	0	0		Q_t	Q'_t	Memori
0	0	0	1		1	0	Set
0	0	1	0		0	1	Reset
0	0	1	1		1*	1*	invalid
x = sembarang				Q_t = tetap			
* = invalid (larangan)				** = tak sinkron			

D-FLIP FLOP

Rangkaian D flip-flop yang dibentuk oleh gerbang NAND ditunjukkan dalam Gambar 10. Rangkaian ini sama seperti R-S flip-flop yang menggunakan NAND, tetapi antara masukan S dan R terpasang inverter yang membuat masukan R merupakan komplement masukan S



Gambar 7.3 Gerbang dan Tabel Kebenaran D FF

D Flip-flop Dengan Gerbang NAND

Tabel Kebenaran				
D	Clk	Q	Q'	Mode
x	0	Q_{t-1}	Q'_{t-1}	Memori
0	1	0	1	Data in
1	1	1	0	Data in
x = sembarang				
Q_{t-1} = keluaran sebelumnya				

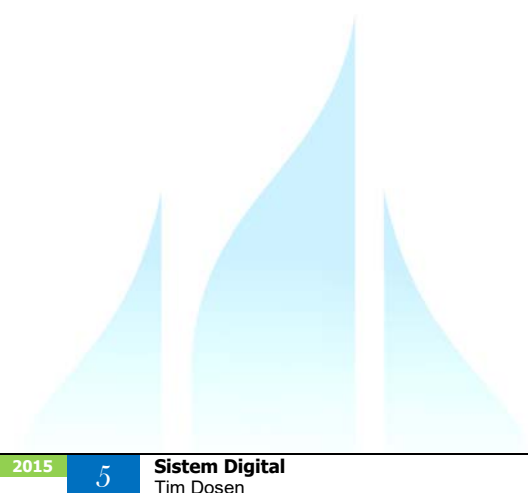
Pengeseetan masukan klok Clk pada level 0, berarti masukkan gerbang NAND 3 dan 4 berlogik 0, keadaan ini menyebabkan keluaran kedua gerbang NAND tersebut berlogik 1, yang tidak mengubah keadaan keluaran pengancing gerbang NAND 1 dan 2. Rangkaian ini dalam keadaan mode memori sepanjang klok Clk=0, lihat Gambar 7.3.

Pengeseetan masukan klok Clk pada level 1, terjadi perpindahan kontrol keluaran rangkaian D flip-flop, pada masukan D. Keluaran Q=1 bila masukan D=1, dan keluaran Q=0 bila masukan D=0. Keluaran Q rangkaian D flip-flop selalu sama dengan masukan D, sepanjang klok Clk=1. Sedang keluaran Q' selalu merupakan komplemen dari masukan D.

Dalam kenyataan pengeseetan klok Clk=1 membuat keluaran Q=D dan Q'=NOT D. Rangkaian D flip-flop tidak mempunyai mode masukan invalid sebagaimana terjadi pada R-S flip-flop. Dengan adanya inverter pada salah satu masukan S-R flip-flop, kondisi invalid tidak akan terjadi. Mode invalid terjadi pada R-S flip-flop saat keadaan kedua masukan R-S flip-flop berlevel 1 untuk waktu sama.

Keluaran Q selalu sesuai dengan masukan D selama Clk=1, dengan kata lain dalam rangkaian seperti ini masukan D berhubungan langsung dengan keluaran Q, atau melalui inverter dengan keluaran Q'. Mode memori R=S=0, ketika Clk=1 pada R-S flip-flop tidak terjadi dalam D flip-flop, keadaan memori dalam D flip-flop hanya dapat terjadi ketika Clk=0, lihat baris pertama pada Gambar 7.3

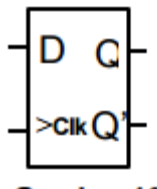
Kinerja dari D flip-flop dapat dirangkum sebagai berikut : 1. Keluaran Q selalu mengikuti masukan D sepanjang klok Clk=1 2. Flip-flop dalam keadaan mode memori sepanjang klok Clk=0 3. Rangkaian tidak mempunyai kondisi operasi invalid.



D flip-flop terpicu-sisi (Edge-Triggered)

D flip-flop jenis ini secara normal dalam keadaan mode memori baik klok pada logik 0 maupun pada logik 1. Hanya ada satu interval waktu yang sangat pendek yang dapat mengubah keadaan keluaran, yaitu masa perubahan dari 0 ke 1, atau perubahan dari 1 ke 0. Flip flop jenis ini hanya merespon pada sisi naik atau sisi turun dari sebuah bentuk gelombang masukan, selain itu D flip flop selalu dalam keadaan mode memori.

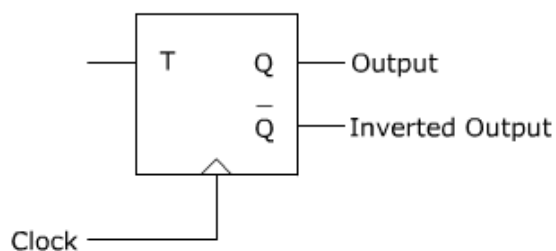
Tabel 10.
D Flip-flop Terpicu-Sisi



Tabel kebenaran				
D	Clk	Q	Q'	Mode
x	0	Q_t	Q'_t	Memory
x	1	Q_t	Q'_t	Memory
0	8	0	1	Data in
8	8	1	0	Data in

T FLIP FLOP

The T type flip-flop is a single input device: T (trigger). Two outputs: Q and Q' (where Q' is the inverse of Q).



The operation of the T type flip-flop is as follows: A '0' input to 'T' will make the next state the same as the present state (i.e. T = 0 present state = 0 therefore next state = 0). However a '1' input to 'T' will change the next state to the inverse of the present state (i.e. T = 1 present state = 0 therefore next state = 1).

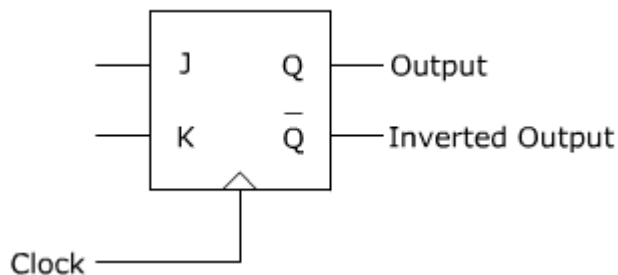
Knowing the above, we can now formalise the operating characteristics and the state change table:

Input T	Circuit Action $Q_{(Time\ t+1)}$
0	$Q_{(t)}$
1	$Q_{(t)}'$

Present State	Next State	Input T	Map Entry
0	0	0	0
0	1	1	T or 1
1	0	1	T or 1
1	1	0	0

JK FLIP FLOP

The JK type flip-flop consists of two data inputs: J and K, and one clock input. There are again two outputs Q and Q' (where Q' is the reverse of Q).



Input		Circuit Action $Q_{(Time\ t+1)}$
J	K	
0	0	$Q_{(t)}$
1	0	0
0	1	1
1	1	$Q_{(t)}'$

- When J=K=0, the current output will carry through to the next state. e.g. Current state Q = Next state Q
- When J=0 and K=1, the next state output will be put to 0. This happens regardless of the present state output.
- When J=1 and K=0, the next state output will be asserted (put to 1). This happens regardless of the present state output.
- When J=K=1, the next state output will be the inverse of the current state output. e.g. Current state Q' = Next state Q.

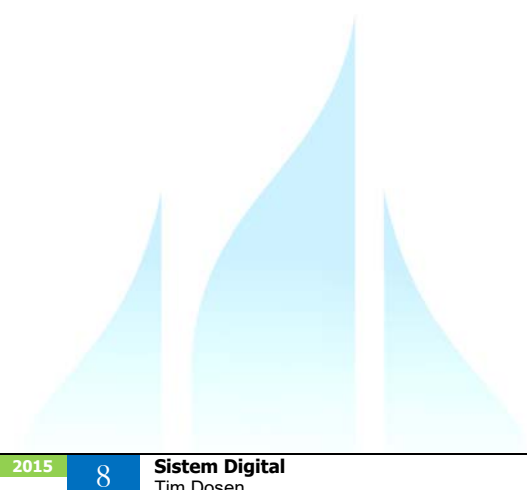
Knowing the above we can now construct the state change table:

Present State	Next State	Inputs		Map Entry	
		J	K	J	K
0	0	0	X	0	X
0	1	1	X	1	X
1	0	X	1	X	1
1	1	X	0	X	0

Lets discuss this state change table with respect to the operating characteristics diagram. There actually exists two operating characteristics that satisfy every possible output combination. This means there should be some 'don't care' terms with each output combination (as our diagram shows). In the list below we shall see how each of the terms

- i. Two conditions exist so that the next state is 0 while the present state is also 0. From the operating characteristics diagram, we can see that condition A and B would both satisfy this scenerio. The common term to make this scenerio true is $J=0$. We dont care about K, as $K=1$ or $K=0$ while $J=0$ will work. Hence the 'don't care' term is K,
- ii. Operating characteristics C and D both satisfy this scenerio. The common term is again J, as the situation is solved by $J=1$ and either $K=0$ or $K=1$, therefore the 'don't care' term is K as shown on the state change table.
- iii. When the output goes from 1 to 0, there are two characteristics that will allow this to happen; B and D. $K=1$ and J can be equal to 1 or 0. Therefore in this case, J is the 'don't care' term.
- iv. When the JK flip-flop remains at logic, it means that either A or C of the four operating characteristics have been applied. K must equal 0 in either case, but J could have been equal to 1 (A) or 0 (C). Because of this, J is the 'don't care' term.

The JK flip-flop can actually be reconfigured so that it can perform the operation of some of the other flip-flops that are discussed above. For example, if the two inputs J and K are tied together, then the output characteristics are fixed to A and D. This precisely matches the characteristics of a T type flip flop. Also to note, because the way a JK is made, you may replace an SR flip-flop with a JK flip-flop without a change in operation. However you cannot replace a JK flip-flop with an SR flip-flop as a $S=1$ $R=1$ condition is not allowed, but a $J=1$ $K=1$ condition is permitted



Daftar Pustaka

Ronald J. Tocci, Neal S.Widmer, Gregory L. Moss, Digital Systems Principles and Applications TENTH EDITION, 2007, Pearson Education International



MODUL PERKULIAHAN

SISTEM DIGITAL

REGISTER

Fakultas

Ilmu Komputer

Program Studi

Teknik Informatika

Tatap Muka

08

Kode MK

15048

Disusun Oleh

Tim Dosen

Abstract

Modul ini membahas tentang Register

Kompetensi

- Mahasiswa diharapkan dapat memahami bagaimana implementasi FF pada register
-

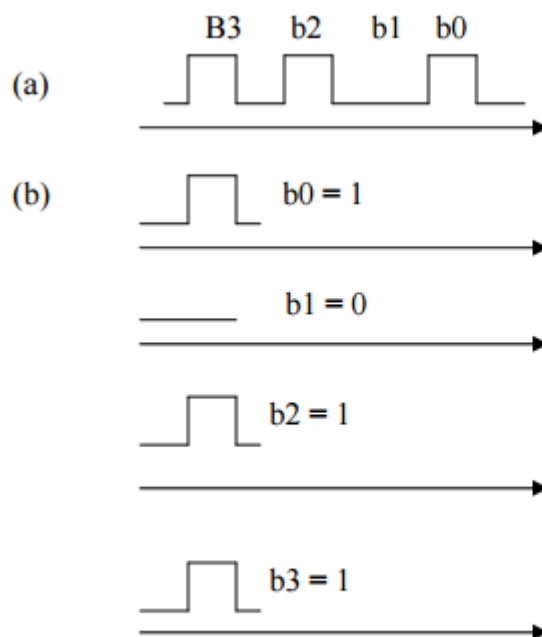
Register

Register Dalam elektronika digital seringkali diperlukan penyimpan data sementara sebelum data diolah lebih lanjut. Elemen penyimpan dasar adalah flip-flop. Setiap flip-flop menyimpan sebuah bit data. Sehingga untuk menyimpan data n-bit, diperlukan n buah flip-flop yang disusun sedemikian rupa dalam bentuk register. Suatu memori register menyimpan data 1001 dapat ditunjukkan secara blok diagram seperti gambar 8.1.



Gambar 8.1 Blok diagram register memori 4-bit

Data biner dapat dipindahkan secara seri atau paralel (lihat gambar 8.2)

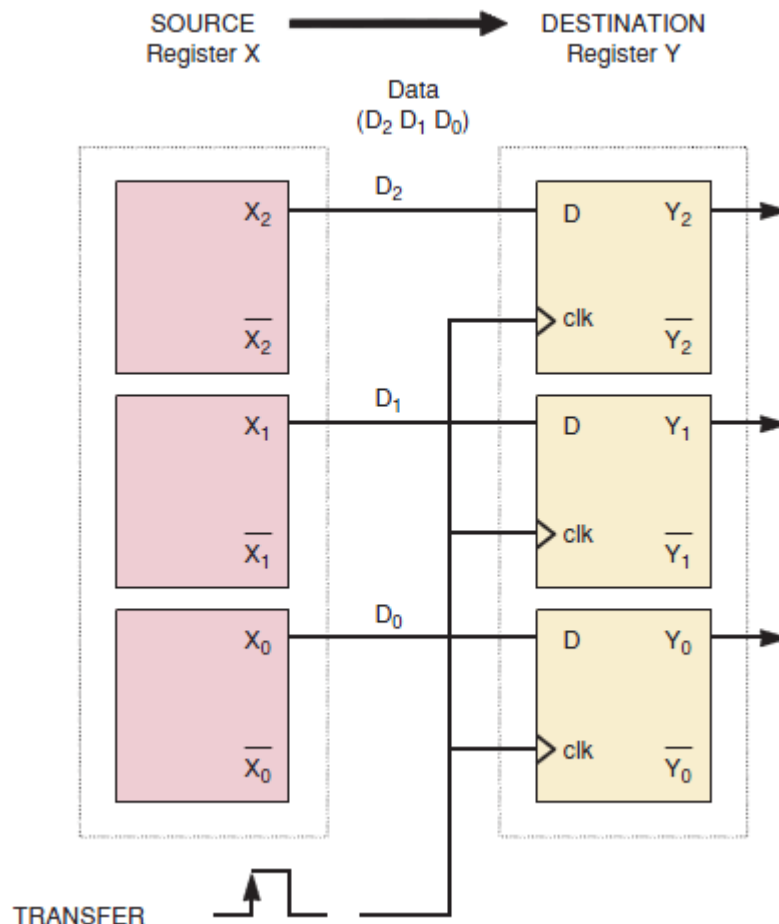


Gambar 8.2 Transfer data (a) mode seri , dan (b) mode paralel

Paralel data Transfer

Terdiri dari serangkaian memori 1 bit yang dapat ditulis atau dibaca secara bersamaan. Digunakan untuk menyimpan data.

- Contoh :



Gambar 8.3 paralel data transfer

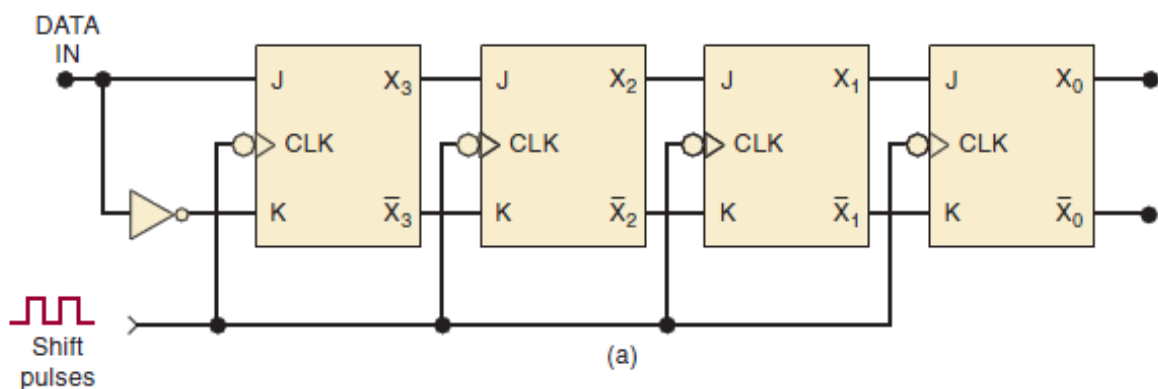
Pada gambar 8.3, merupakan register parallel data transfer dengan menggunakan D-FF. Terdapat dua buah register yaitu register X dan register Y, dimana register X akan mentransfer 3 bit data secara bersamaan ke register Y. Register X terdiri dari X₁, X₂, dan X₃. Sedangkan register Y terdiri dari Y₁, Y₂, dan Y₃. X₁ akan ditransfer ke Y₁, X₂ ke Y₂ dan X₃ ke Y₃. Pada saat clock aktif, maka seluruh flip flop D akan aktif, dan mengakibatkan data dari X akan sama dengan register Y secara bersamaan

Serial data Transfer / Shift

Serial data transfer register / Shift register terdiri dari sekelompok FF yang disusun sehingga bilangan biner dapat disimpan didalamnya dengan cara di shift / gesere dari satu FF ke FF berikutnya setiap Pulsa Clock.

Ilustrasi dari shift register ini adalah kalkulator. Ketika kita menekan satu tombol pada kalkulator, maka angka sebelumnya yang sudah tampil akan bergeser, sehingga setiap tombol ditekan, maka akan terjadi pergeseran angka pada layar monitor. Hal ini sama dengan konsep shift register, dimana setiap terjadi clock, maka akan ada 1 bit yang bergeser dari satu FF ke FF lainnya.

Contoh Shift register dengan JK FF



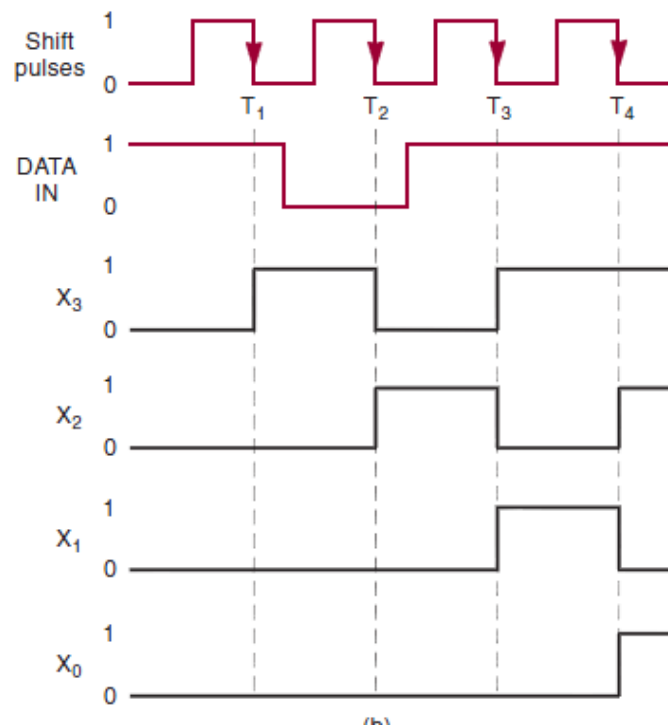
Gambar 8.4 JK Shift Register

Pada gambar 8.4 diatas, terlihat 4 buah JK FF yang disusun membentuk sebuah shift register 4 bit. Dimana data akan masuk pada **DATA IN** dan diteruskan ke FF pertama pada inputan J, sedangkan untuk inputan K berasal dari inverter DATA IN. untuk FF kedua dan seterusnya inputan J berasal dari output setiap FF dan inputn K berasal dari inverter output dari FF sebelumnya. Clock pada gambar diatas adalah aktif LOW

Contoh : data yang akan disimpan dalam shift register adalah 1001 atau 5 dalam bilangan decimal.

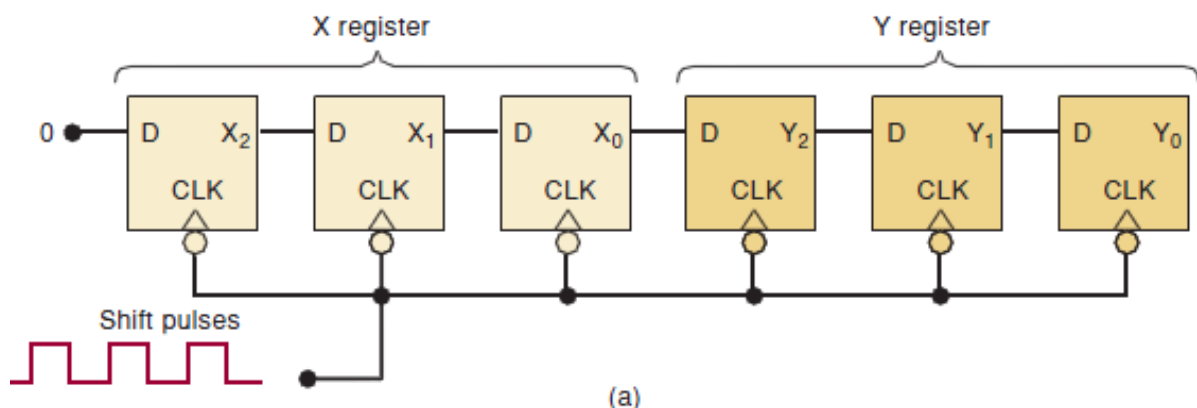
DATA	Clock	J	K	X3	J	K	X2	J	K	X1	J	K	X0
	-	1	0	0			0			0			0
1	Aktif	1	0	1			0			0			0
0	Aktif	0	1	0	1	0	1			0			0
0	Aktif	0	1	0	0	1	0	1	0	1			0
1	Aktif	1	0	1	0	1	0	0	1	0	1	0	1

Bentuk gelombang digital dari rangkaian pada gambar diatas dengan contoh data IN adalah 1011



Gambar 8.5 Gelombang Sinyal Digital

Contoh Shift Register menggunakan D FF



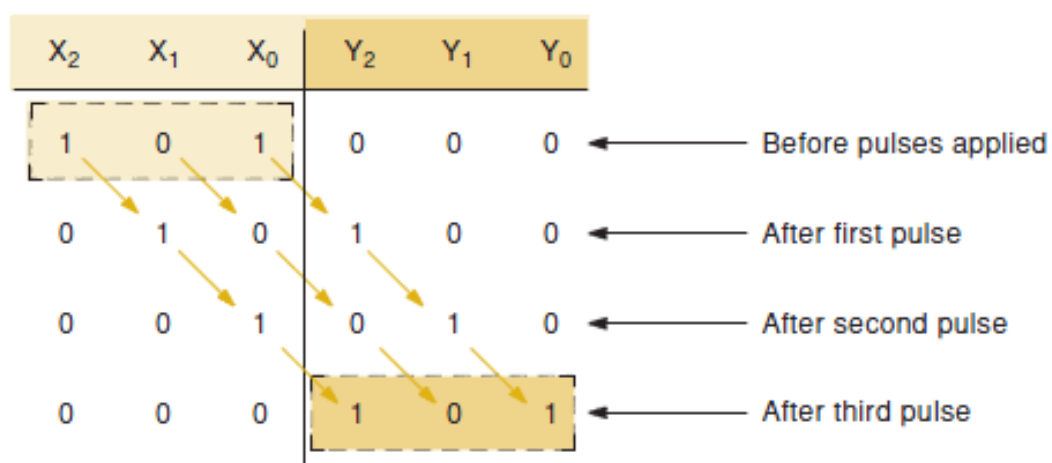
Gambar 8.6 Shift register menggunakan D FF

Pada gambar 8.6, merupakan register shift / serial data transfer dengan menggunakan D-FF. Terdapat dua buah register yaitu register X dan register Y, dimana register X akan mentransfer 3 bit data secara serial ke register Y. Register X terdiri dari X1, X2, dan X3, Sedangkan register Y terdiri dari Y1, Y2, dan Y3. Input FF yang pertama adalah data dan

input dari FF berikutnya adalah output dari FF sebelumnya. Pada saat clock aktif, maka seluruh flip flop D akan mendapatkan clock, tetapi yang aktif hanya FF yang sudah menerima input.

Contoh :

Register X sudah berisi data 101, dan akan dipindahkan ke register Y, setiap 1 kali clock, maka akan ada satu bit data yang akan digeser, sehingga diperlukan 3 kali clock agar seluruh data pada register X, berpindah ke Register Y. (Gambar 8.7)



Gambar 8.7 Ilustrasi perpindahan data

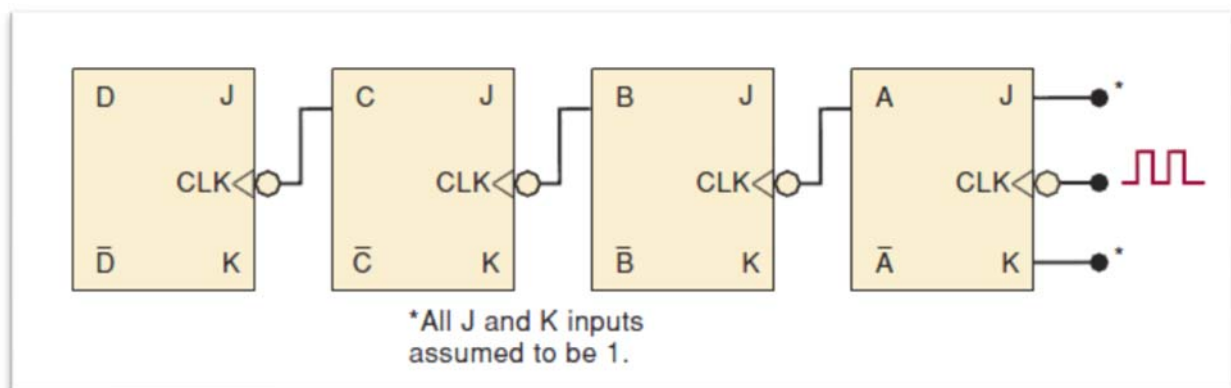
Counter

Counter merupakan sebuah Register yang nilainya di – increment dengan 1, dan juga dibangun dengan menggunakan n FF, contoh counter pada CPU adalah Program Counter. Program Counter berfungsi untuk menunjukkan alamat dari instruksi berikutnya yang akan di jemput oleh processor.

Counter didesain dengan menggunakan 2 model yaitu

- Asinkron → Model ini Lebih lambat, karena output dari sebuah FF merupakan trigger merubah status dari FF berikutnya
- Sinkron → Semua FF berubah state pada waktu yang sama, sehingga prosesnya lebih cepat. Impelementasinya adalah pada CPU

Counter Asinkron



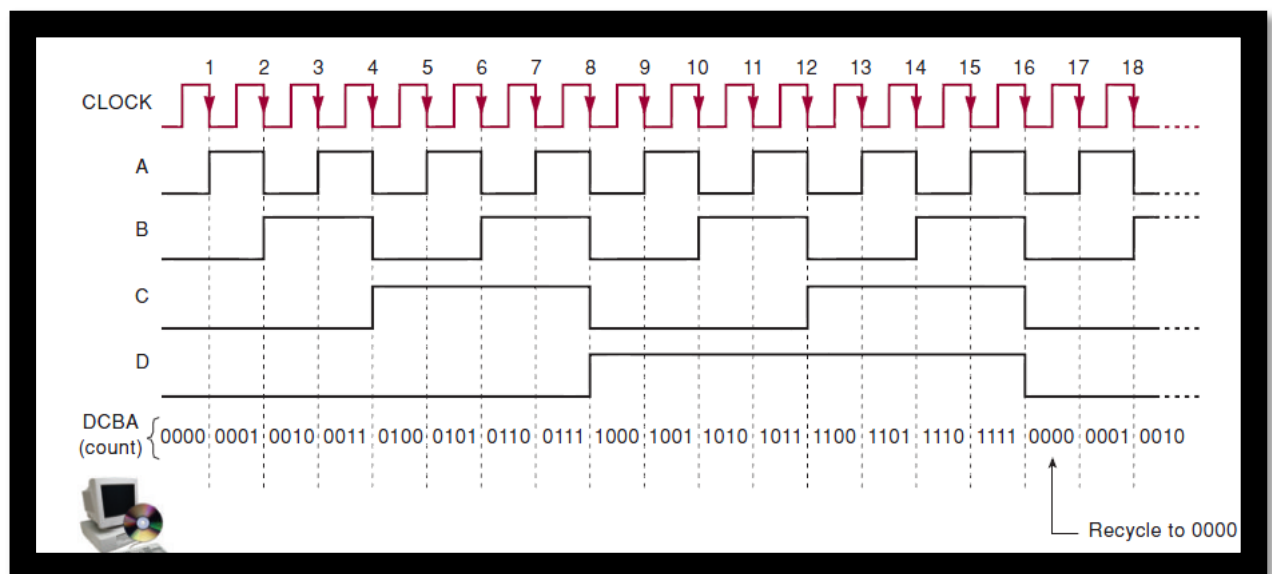
Gambar 8.8 Counter Asinkron JK FF

Pada gambar 8.8 diatas terlihat rangkaian 4 buah JK FF yang berfungsi sebagai counter dan bekerja dengan mode asinkron.

1. Semua Inputan dari FF diasumsikan bernilai 1.
2. Clock FF pertama berasal dari pulsa clock langsung, sedangkan clock FF berikutnya berasal dari output dari FF sebelumnya. Clock pada kasus ini adalah aktif LOW.

3. Proses nya dapat dilihat pada table berikut

Clock	FF4	D	FF3	C	FF2	B	FF1	A	Keterangan
		0		0		0		0	Nilai Awal
1	x	0	x	0	x	0	v	1	FF 2 tidak aktif karena output FF1 yaitu A = 1, sedangkan clock aktif rendah, sehingga tidak bisa memicu FF2
2	x	0	x	0	v	1	v	0	FF1 aktif, output A = 0, sehingga bisa mengaktifkan output FF2, dan menghasilkan output B=, Karena High, sehingga tidak bisa memicu FF3
3	x	0	x	0	x	1	v	1	FF1 aktif, output A = 1, Karena High, sehingga tidak bisa memicu FF berikutnya
4	Proses yang sama akan terjadi secara berulang								



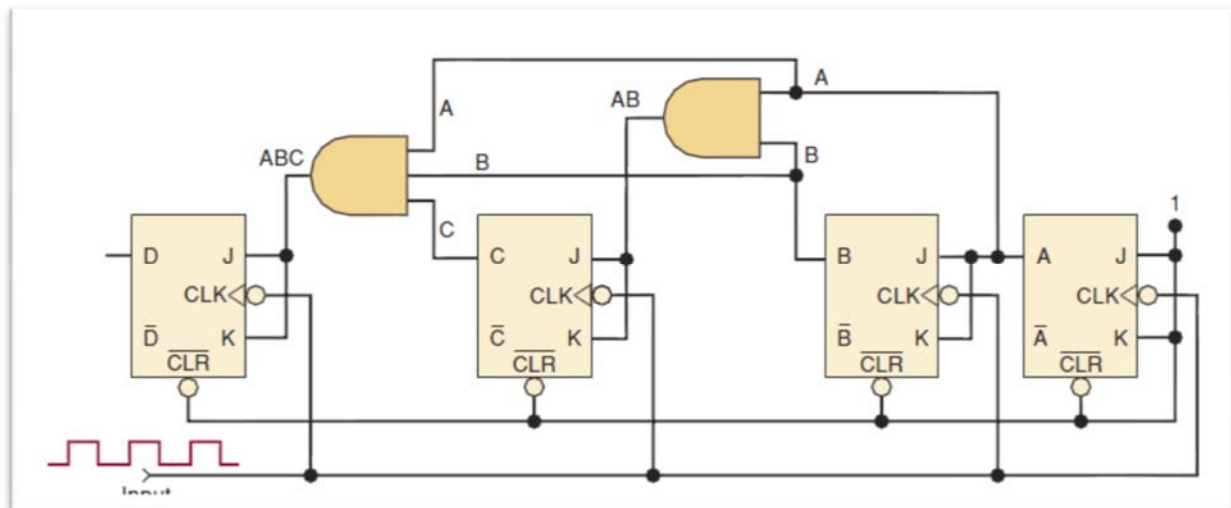
MOD NUMBER

Contoh sebelumnya memiliki 16 state (0000 – 1111), dan disebut sebagai MOD – 16 ripple counter. Hal ini diperoleh dengan cara:

$$\text{MOD number} = 2^N$$

Dimana N → jumlah FF yang dihubungkan

Sinkron Counter



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

Daftar Pustaka

Ronald J. Tocci, Neal S. Widmer, Gregory L. Moss, Digital Systems Principles and Applications TENTH EDITION, 2007, Pearson Education International



MODUL PERKULIAHAN

SISTEM DIGITAL

Rangkaian Aritmatika

Fakultas
Ilmu Komputer

Program Studi
Teknik Informatika

Tatap Muka

09

Kode MK
15048

Disusun Oleh
Tim Dosen

Abstract

Modul ini membahas tentang Half Adder dan Full Adder

Kompetensi

- Mahasiswa diharapkan dapat memahami bagaimana operasi aritmatika dengan menggunakan HA dan FA
-

ALU

All arithmetic operations take place in the **arithmetic/logic unit (ALU)** of a computer. Figure 9-1 is a block diagram showing the major elements included in a typical ALU. The main purpose of the ALU is to accept binary data that are stored in the memory and to execute arithmetic and logic operations on these data according to instructions from the control unit.

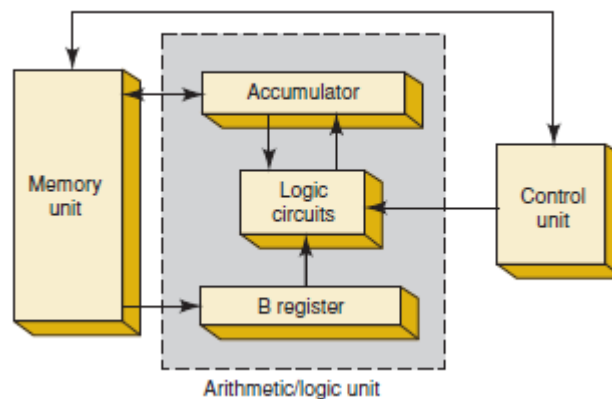


Figure 9.1 ALU

The arithmetic/logic unit contains at least two flip-flop registers: the *B register* and the **accumulator register**. It also contains combinational logic, which performs the arithmetic and logic operations on the binary numbers that are stored in the *B register* and the accumulator. A typical sequence of operations may occur as follows:

1. The control unit receives an instruction (from the memory unit) specifying that a number stored in a particular memory location (address) is to be added to the number presently stored in the accumulator register.
2. The number to be added is transferred from memory to the *B register*.
3. The number in the *B register* and the number in the accumulator register are added together in the logic circuits (upon command from the control unit). The resulting sum is then sent to the accumulator to be stored.
4. The new number in the accumulator can remain there so that another number can be added to it or, if the particular arithmetic process is finished, it can be transferred to memory for storage.

These steps should make it apparent how the accumulator register derives its name. This register “accumulates” the sums that occur when performing successive additions between new numbers acquired from memory and the previously accumulated sum. In fact, for any

arithmetic problem containing several steps, the accumulator usually contains the results of the intermediate steps as they are completed as well as the final result when the problem is finished.

Aritmatika Biner

Operasi aritmatika binari, merupakan operasi aritmatika yang melandasi tentang proses aritmatika dan logika pada sistem digital dan komputer modern. Namun pada kenyataannya, operasi aritmatika tersebut tidak dapat diwakili oleh operasi yang terdapat pada gerbang dasar, walaupun secara prinsipnya, tiap-tiap gerbang mewakili satu operasi aritmatika, terutama adalah operasi-operasi dasar penjumlahan dan pengurangan.

Contoh :

Pada saat kita akan menjumlahkan dua bilangan biner $1+1$, bila diwakilkan dengan operasi penjumlahan pada Gerbang OR, akan menghasilkan keluaran yang berbeda dengan hasil operasi aritmatika yang sesungguhnya, dimana jika diperasikan dengan Gerbang OR, akan diperoleh keluaran $1_2+1_2 = 1_2$, sedangkan pada operasi aritmatika idealnya output yang diperoleh adalah $1+1 = 10_2$, bagaimana mengimplementasikan operasi ini pada sistem komputer digital ? Maka untuk menjawab pertanyaan ini dibutuhkan sebuah rangkaian aritmatika yang dapat mewakili operasi aritmatika yang sesungguhnya.

Operasi penjumlahan aritmatika :

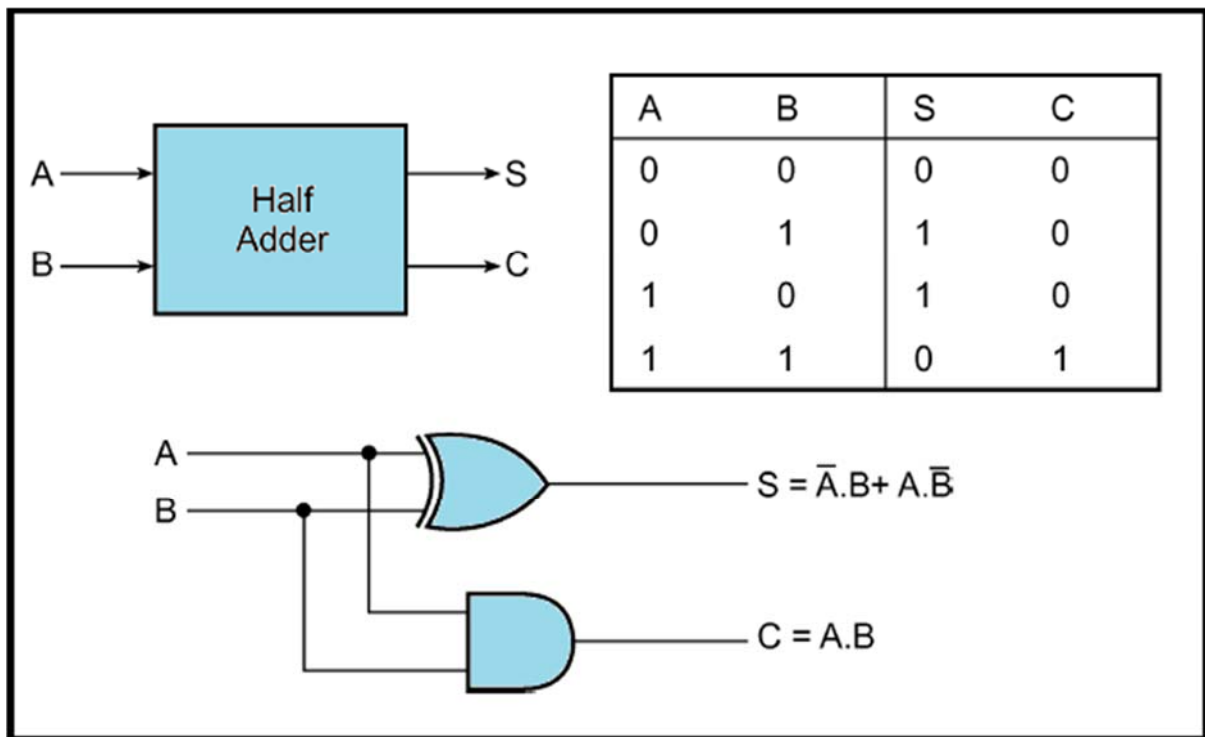
$$\begin{array}{r} 1 \\ 1 + \\ \hline 10 \end{array}$$

↑ Limpahan

← Hasil jumlah

Rangkaian Penjumlah Tak lengkap (Half Adder)

Penjumlah tak lengkap (*half adder*) menjumlahkan 2 angka biner pada satu operasi, yang akan menghasilkan keluaran dua digit biner, yaitu biner hasil jumlah dan biner limpahan. Rangkaian *half adder* (HA) ini mewakili operasi aritmatika penjumlahan dua bilangan biner, yang tidak bisa diwakili oleh operasi penjumlahan dengan gerbang OR. Sebuah HA, dapat digambarkan fungsinya seperti pada Gambar 9.2 berikut :



Gambar 9.2 Block Diagram, rangkaian dan table kebenaran half Adder

Untuk melakukan operasi penjumlahan dengan HA ini, diwakili oleh dua buah gerbang logika, yaitu gerbang EXOR dan AND, dengan bentuk rangkaian seperti pada Gambar 9.2. Operasi rangkaian ini, dapat dianalogikan dengan operasi persamaan :

- Sum = $AB + AB = A \oplus B$

- Carry = $A.B$

Prinsip kerja rangkaian ini dapat dibuktikan dengan tabel kebenaran pada Gambar 9.2

Seperti halnya untuk setiap rangkaian logika dengan 2 input, maka akan diperoleh variasi Masukan, Sehingga operasinya dapat diuraikan sebagai berikut :

1. Bila A = 0 dan B = 0, maka :

$$\text{Sum} = A \oplus B = 0 \oplus 0 = 0$$

$$\text{Carry} = A . B = 0 . 0 = 0$$

2. Bila $A = 0$ dan $B = 1$, maka :

Sum	$= A \oplus B$	$= 0 \oplus 1$	$= 1$
Carry	$= A . B$	$= 0 . 1$	$= 0$
3. Bila $A = 1$ dan $B = 0$, maka :

Sum	$= A \oplus B$	$= 1 \oplus 0$	$= 1$
Carry	$= A . B$	$= 1 . 0$	$= 0$
4. Bila $A = 1$ dan $B = 1$, maka :

Sum	$= A \oplus B$	$= 1 \oplus 1$	$= 0$
Carry	$= A . B$	$= 1 . 1$	$= 1$

Sehingga dengan rangkaian HA ini, kita dapat melakukan operasi penjumlahan biner dengan rangkaian logika, sehingga dengan demikian, akan dapat kita peroleh : $0+0 = 0$, $0+1 = 1$, $1+0=1$ dan $1+1 = 10$.

Kekurangan

Penjumlahan dengan HA ini hanya dapat melakukan operasi penjumlahan dua biner terhadap LSB (*least significant binary*)-nya saja, tetapi untuk nilai biner yang lebih berbobot, rangkaian ini tidak dapat melakukannya.

Rangkaian Penjumlah Lengkap (*Full Adder*)

Kelemahan yang dimiliki oleh rangkaian penjumlah tak penuh (HA), yang hanya dapat melakukan operasi penjumlahan terhadap 2 bilangan biner pada sisi LSB, diatasi dengan membangun rangkaian penjumlah yang lebih lengkap yang disebut dengan rangkaian penjumlah lengkap (*Full Adder*).

Rangkaian *Full Adder* (FA), merupakan sebuah rangkaian penjumlah yang mempunyai tiga input, termasuk masukan bawaan (*input carry*) dan menghasilkan keluaran hasil jumlah (*sum*) dan hasil bawaan (*output carry*).

Dalam menjumlahkan dua bilangan biner, mungkin terdapat bawaan dari satu kolom ke kolom berikutnya, contoh :

$$\begin{array}{r}
 111 \\
 +101 \\
 \hline
 1100
 \end{array}$$

Dalam kolom paling ringan (LSB – *least significant binary*) :

$$1 + 1 = 0, \text{ dengan bawaan } 1$$

Dalam kolom berikutnya, kita harus menjumlahkan 3 angka digit, akibat adanya bawaan dari kolom sebelumnya :

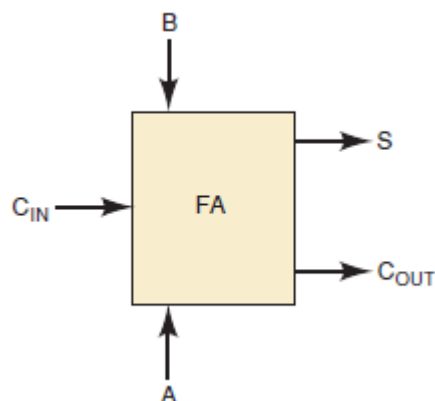
$$1 + 0 + 1 = 0, \text{ dengan bawaan } 1$$

Dalam kolom terakhir, kembali terjadi penjumlahan dengan 3 angka biner, akibat bawaan kolom kedua :

$$1 + 1 + 1 = 1, \text{ dengan bawaan } 1$$

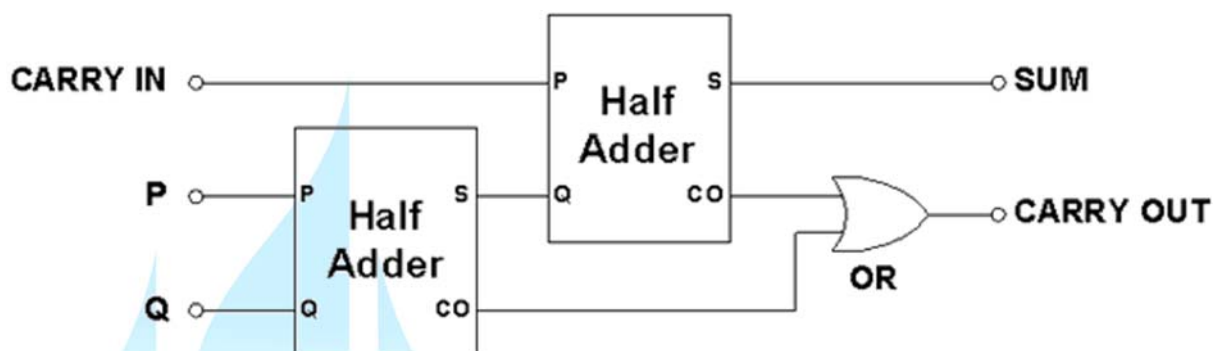
Oleh sebab itu, untuk menjumlahkan bilangan-bilangan biner dengan operasi bawaan yang menghasilkan penjumlahan 3 biner sekaligus, tidak bisa dilakukan lagi dengan HA, tetapi merupakan suatu rangkaian FA.

Sebuah FA pada dasarnya adalah gabungan dua buah HA, dengan kedua output *carry*-nya dijumlahkan, yang dilengkapi dengan sebuah masukan *carry* (C_{IN}), secara blok dapat dilihat pada Gambar 9.3.



Gambar 9.3 Block Diagram Full Adder

Full Adder Bisa dibangun dari gabungan 2 buah half adder seperti terlihat pada Gambar 9.4 dibawah

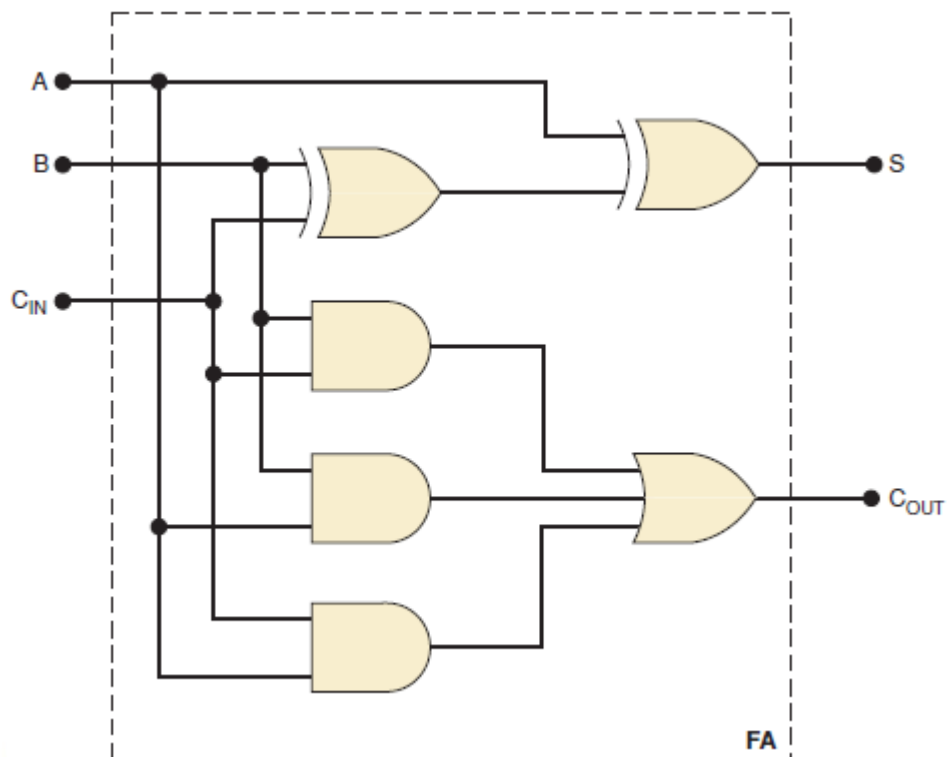
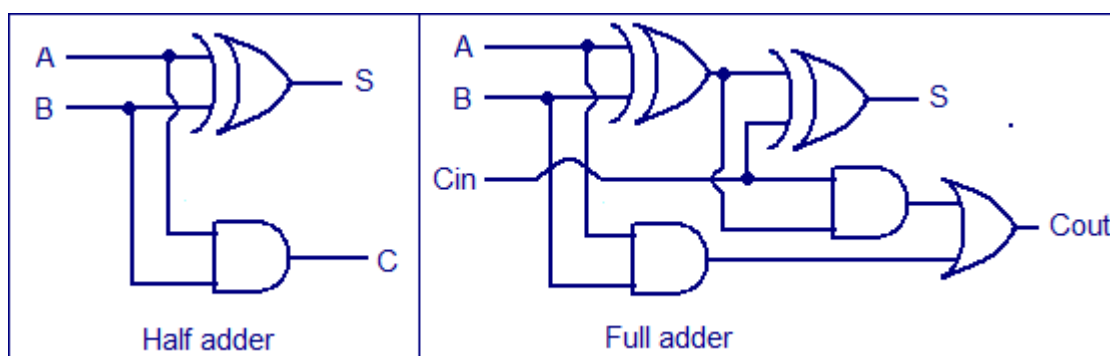


Gambar 9.4. Full adder dengan kombinasi 2 HA

Tabel Kebenarannya adalah sebagai berikut

Augend bit input	Addend bit input	Carry bit input	Sum bit output	Carry bit output
A	B	C _{IN}	S	C _{OUT}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Rangkaian Lengkapnya adalah seperti pada gambar 9.5



Gambar 9.5 Full adder

Dengan memperhatikan skema rangkaian FA, maka dapat dibuat suatu persamaan untuk output-outputnya sebagai berikut :

1. Keluaran HA1

$$\text{Sum} = A \oplus B, \text{ dan Carry} = A.B$$

2. Keluaran HA 2

$$\text{Sum} = (A \oplus B) \oplus \text{Cin}, \text{ dan Carry} = (A \oplus B). \text{Cin}$$

3. Keluaran lengkap Full Adder

$$\text{Sum} = (A \oplus B) \oplus \text{Cin}, \text{ dan Carry} = AB + (A \oplus B). \text{Cin}$$

Ilustrasi operasi adalah sebagai berikut

1. Saat masukan $A=0$, $B=0$ dan $\text{Cin}=0$, maka :

a. Keluaran HA 1

$$\text{Sum} = A \oplus B = 0 \oplus 0 = 0$$

$$\text{Carry} = A.B = 0 . 0 = 0$$

b. Keluaran HA 2

$$\text{Sum} = (A \oplus B) \oplus \text{Cin} = 0 \oplus 0 = 0$$

$$\text{Carry} = (A \oplus B). \text{Cin} = 0 . 0 = 0$$

c. Keluaran Full Adder

$$\text{Sum} = (A \oplus B) \oplus \text{Cin} = 0 \oplus 0 = 0$$

$$\text{Carry} = AB + (A \oplus B). \text{Cin} = 0 + 0 = 0$$

2. Saat masukan $A=0$, $B=0$ dan $\text{Cin}=1$, maka :

a. Keluaran HA 1

$$\text{Sum} = A \oplus B = 0 \oplus 0 = 0$$

$$\text{Carry} = A.B = 0 . 0 = 0$$

b. Keluaran HA 2

$$\text{Sum} = (A \oplus B) \oplus \text{Cin} = 0 \oplus 1 = 1$$

$$\text{Carry} = (A \oplus B). \text{Cin} = 0 . 1 = 0$$

c. Keluaran Full Adder

$$\text{Sum} = (A \oplus B) \oplus \text{Cin} = 0 \oplus 1 = 1$$

$$\text{Carry} = AB + (A \oplus B). \text{Cin} = 0 + 0 . 1 = 0$$

3. Saat masukan $A=0$, $B=1$ dan $\text{Cin}=0$, maka :

a. Keluaran HA 1

$$\text{Sum} = A \oplus B = 0 \oplus 1 = 1$$

$$\text{Carry} = A.B = 0 . 1 = 0$$

b. Keluaran HA 2

$$\text{Sum} = (A \oplus B) \oplus \text{Cin} = 1 \oplus 0 = 1$$

$$\text{Carry} = (A \oplus B) \cdot \text{Cin} = 1 \cdot 0 = 0$$

c. Keluaran Full Adder

$$\text{Sum} = (A \oplus B) \oplus \text{Cin} = 1 \oplus 0 = 1$$

$$\text{Carry} = AB + (A \oplus B) \cdot \text{Cin} = 0 + 1 \cdot 0 = 0$$

4. Saat masukan A= 0, B= 1 dan Cin = 1, maka :

a. Keluaran HA 1

$$\text{Sum} = A \oplus B = 0 \oplus 1 = 1$$

$$\text{Carry} = A \cdot B = 0 \cdot 1 = 0$$

b. Keluaran HA 2

$$\text{Sum} = (A \oplus B) \oplus \text{Cin} = 1 \oplus 1 = 0$$

$$\text{Carry} = (A \oplus B) \cdot \text{Cin} = 1 \cdot 1 = 1$$

c. Keluaran Full Adder

$$\text{Sum} = (A \oplus B) \oplus \text{Cin} = 1 \oplus 1 = 0$$

$$\text{Carry} = AB + (A \oplus B) \cdot \text{Cin} = 0 + 1 \cdot 1 = 1$$

5. Saat masukan A= 1, B= 0 dan Cin = 0, maka :

a. Keluaran HA 1

$$\text{Sum} = A \oplus B = 1 \oplus 0 = 1$$

$$\text{Carry} = A \cdot B = 1 \cdot 0 = 0$$

b. Keluaran HA 2

$$\text{Sum} = (A \oplus B) \oplus \text{Cin} = 1 \oplus 0 = 1$$

$$\text{Carry} = (A \oplus B) \cdot \text{Cin} = 1 \cdot 0 = 0$$

c. Keluaran Full Adder

$$\text{Sum} = (A \oplus B) \oplus \text{Cin} = 1 \oplus 0 = 1$$

$$\text{Carry} = AB + (A \oplus B) \cdot \text{Cin} = 0 + 1 \cdot 0 = 0$$

6. Saat masukan A= 1, B= 0 dan Cin = 1, maka :

a. Keluaran HA 1

$$\text{Sum} = A \oplus B = 1 \oplus 0 = 1$$

$$\text{Carry} = A \cdot B = 1 \cdot 0 = 0$$

b. Keluaran HA 2

$$\text{Sum} = (A \oplus B) \oplus \text{Cin} = 1 \oplus 1 = 0$$

$$\text{Carry} = (A \oplus B) \cdot \text{Cin} = 1 \cdot 1 = 1$$

c. Keluaran Full Adder

$$\text{Sum} = (A \oplus B) \oplus \text{Cin} = 1 \oplus 1 = 0$$

$$\text{Carry} = AB + (A \oplus B) \cdot \text{Cin} = 0 + 1 \cdot 1 = 1$$

7. Saat masukan A= 1, B= 1 dan Cin = 0, maka :

- a. Keluaran HA 1
 $\text{Sum} = A \oplus B = 1 \oplus 1 = 0$
 $\text{Carry} = A.B = 1 . 1 = 1$
 - b. Keluaran HA 2
 $\text{Sum} = (A \oplus B) \oplus \text{Cin} = 0 \oplus 0 = 0$
 $\text{Carry} = (A \oplus B). \text{Cin} = 0 . 0 = 0$
 - c. Keluaran Full Adder
 $\text{Sum} = (A \oplus B) \oplus \text{Cin} = 0 \oplus 0 = 0$
 $\text{Carry} = AB + (A \oplus B). \text{Cin} = 1 + 0 . 0 = 1$
8. Saat masukan A= 1, B= 1 dan Cin = 1, maka :
- a. Keluaran HA 1
 $\text{Sum} = A \oplus B = 1 \oplus 1 = 0$
 $\text{Carry} = A.B = 1 \oplus 1 = 1$
 - b. Keluaran HA 2
 $\text{Sum} = (A \oplus B) \oplus \text{Cin} = 0 \oplus 1 = 1$
 $\text{Carry} = (A \oplus B). \text{Cin} = 0 . 1 = 0$
 - c. Keluaran Full Adder
 $\text{Sum} = (A \oplus B) \oplus \text{Cin} = 0 \oplus 1 = 1$
 $\text{Carry} = AB + (A \oplus B). \text{Cin} = 1 + 1. 1 = 1$

Ronald J. Tocci, Neal S. Widmer, Gregory L. Moss, Digital Systems Principles and Applications TENTH EDITION, 2007, Pearson Education International

<http://www.circuitstoday.com/ripple-carry-adder>



MODUL PERKULIAHAN

SISTEM DIGITAL

Rangkaian Aritmatika 2

Fakultas
Ilmu Komputer

Program Studi
Teknik Informatika

Tatap Muka

10

Kode MK
15048

Disusun Oleh
Tim Dosen

Abstract

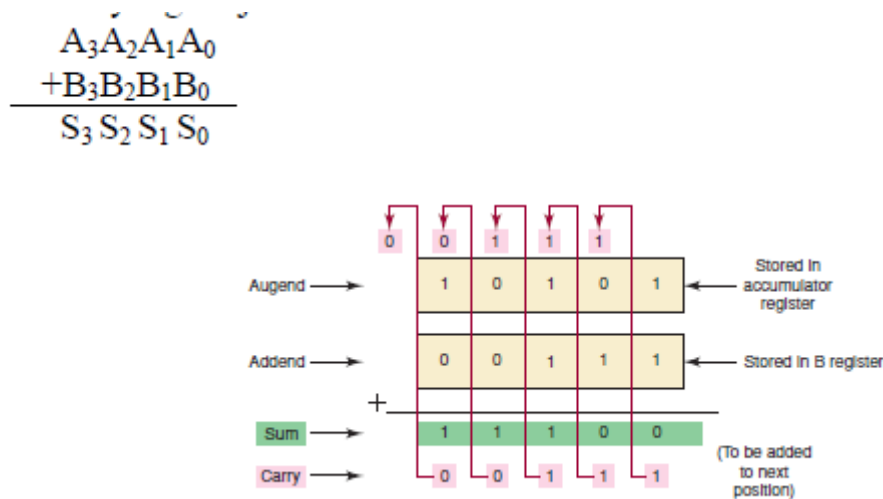
Modul ini membahas tentang Half Adder dan Full Adder

Kompetensi

- Mahasiswa diharapkan dapat memahami bagaimana operasi aritmatika dengan menggunakan HA dan FA
-

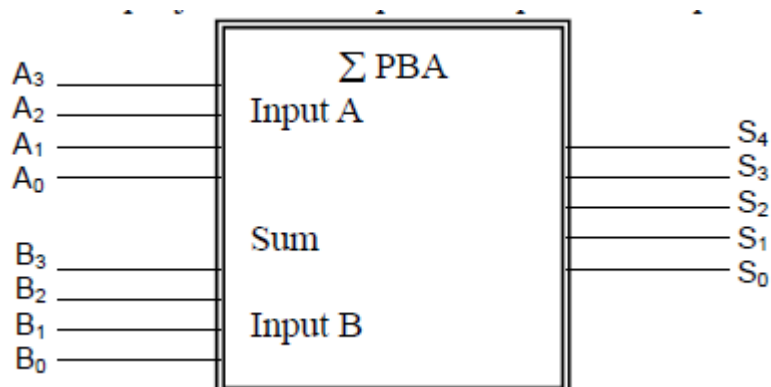
Paralel Binary Adder

Konsep dasar penjumlah lengkap (FA), telah dapat mewakili operasi penjumlahan yang menghasilkan suatu input *carry* pada penjumlahan kolom berikutnya. Tetapi dalam operasionalnya penjumlahan ini tidak hanya sebatas itu saja, tetapi kadang juga terdiri dari sejumlah bilangan biner yang berjajar (paralel), Misalnya :



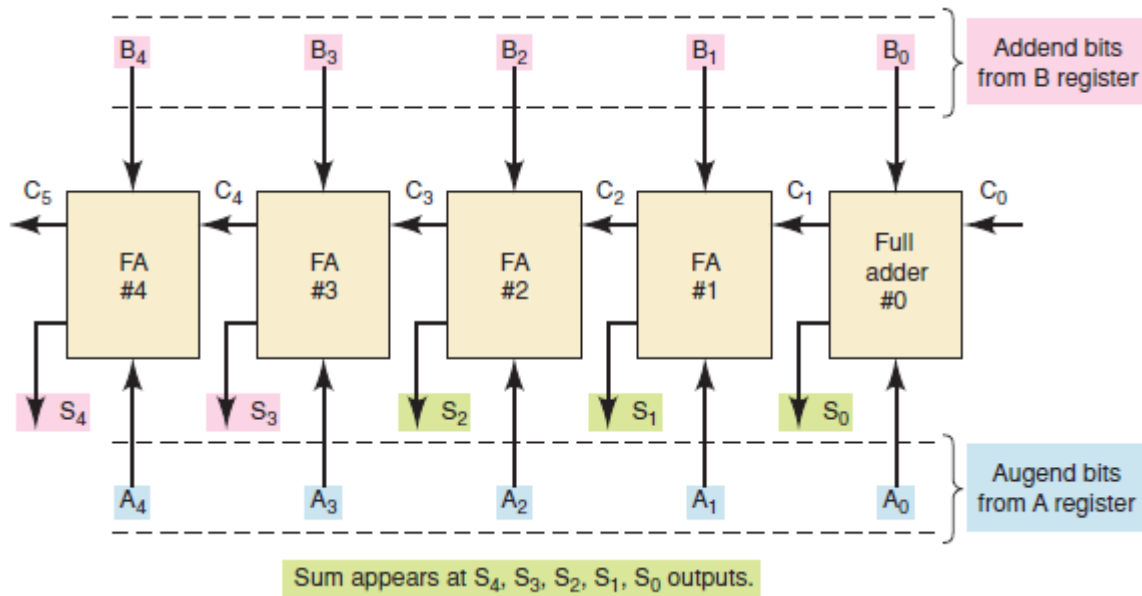
Gambar 10.1 Operasi bilangan biner

Maka untuk melakukan operasi tersebut, tidak dapat diwakilkan lagi dengan hanya menggunakan FA saja, tetapi menggunakan suatu metoda yang disebut *Parallel binary adder*. Blok diagram sebuah penjumlah biner paralel diperlihatkan pada Gambar 10.2



Gambar 10.2 Blok Diagram penjumlah biner paralel

Sebuah penjumlah biner paralel, pada dasarnya dibangun dari sejumlah FA dan HA, tergantung jumlah bit yang akan dijumlahkan secara paralel, selengkapnya untuk membangun sebuah penjumlah paralel biner 5 bit adalah :



Gambar 10.3. Penjumlah biner paralel 5 bit

Kelompok penjumlahan kolom pertama hanya membutuhkan sebuah penjumlah tak lengkap (*Half Adder*), karena belum ada terjadi penjumlahan terhadap *carry*, namun bagi setiap kolom setelah kolom pertama, mungkin akan terdapat bawaan dari kolom sebelumnya; oleh karena itu kita harus menggunakan sebuah penjumlah penuh bagi masing-masing kolom di atas kolom pertama.

Penjumlahan

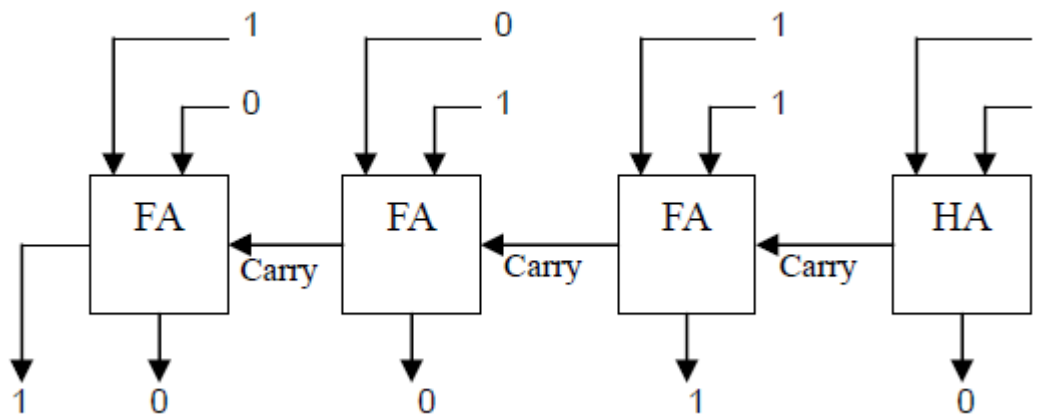
Sebagai contoh operasi penjumlahan biner paralel 4 bit, dapat dilihat pada Gambar 10.4. Misalkan kita akan menjumlahkan bilangan desimal 11 dan 7, maka hasilnya tentu 18, ekivalen 11 dalam biner adalah 1011, dan ekivalen 7 dalam biner adalah 0111, maka operasi

binernya adalah sebagai berikut :

$$\begin{array}{r} 1011 \\ +0111 \\ \hline 10010 \end{array}$$

Pada kolom biner pertama, hanya menggunakan penjumlah HA, karena hanya mempunyai dua masukan yaitu 1 dan 1, tetapi hasil dari operasi pada kolom pertama ini menghasilkan *carry* bagi kolom ke dua dan *carry* ini menjadi input operasi biner kolom kedua, maka pada kolom kedua digunakan sebuah FA, dan operasi pada kolom ke dua ini pun

menghasilkan carry bagi kolom ke tiga, dan kolom ketiga juga menghasilkan carry bagi kolom ke empat dan hasil dari operasi pada kolom ke empat menghasilkan sebuah output carry.



n-bit Carry Ripple Adder

An n-bit adder used to add two n-bit binary numbers can be built by connecting n full adders in series. Each full adder represents a bit position j (from 0 to n-1).

Each carry out C-out from a full adder at position j is connected to the carry in C-in of the full adder at higher position j+1. The output of a full adder at position j is given by: $S_j = X_j \oplus Y_j \oplus C_j$

$$C_{j+1} = X_j \cdot Y_j + X_j \cdot C_j + Y_j \cdot C_j$$

In the expression of the sum C_j must be generated by the full adder at lower position j. The propagation delay in each full adder to produce the carry is equal to two gate delays = 2 D. Since the generation of the sum requires the propagation of the carry from the lowest position to the highest position, the total propagation delay of the adder is approximately:

$$\text{Total Propagation delay} = 2 n D$$

4-bit Carry Ripple Adder

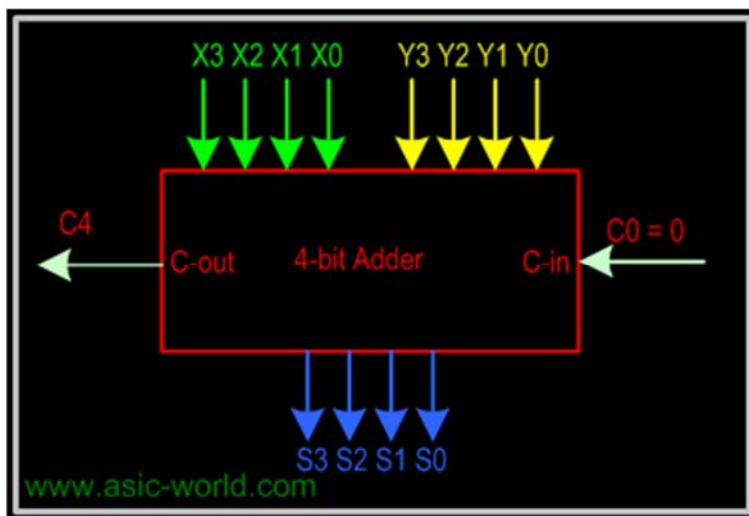
Adds two 4-bit numbers:

$X = X_3 X_2 X_1 X_0$

$Y = Y_3 Y_2 Y_1 Y_0$

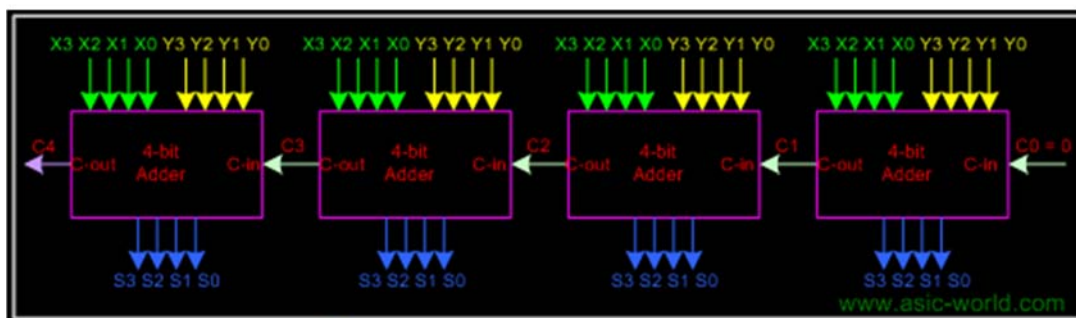
producing the sum $S = S_3 S_2 S_1 S_0$, $C\text{-out} = C_4$ from the most significant position $j=3$

Total Propagation delay = $2 nD = 8D$ or 8 gate delays



Larger Adder

Example: 16-bit adder using 4 4-bit adders. Adds two 16-bit inputs X (bits X_0 to X_{15}), Y (bits Y_0 to Y_{15}) producing a 16-bit Sum S (bits S_0 to S_{15}) and a carry out C_{16} from the most significant position.



Propagation delay for 16-bit adder = 4 x propagation delay of 4-bit adder

= $4 \times 2 nD = 4 \times 8D = 32 D$

or 32 gate delays

Carry Look-Ahead Adder

The delay generated by an N-bit adder is proportional to the length N of the two numbers X and Y that are added because the carry signals have to propagate from one full-adder to the next. For large values of N, the delay becomes unacceptably large so that a special solution needs to be adopted to accelerate the calculation of the carry bits. This solution involves a "look-ahead carry generator" which is a block that simultaneously calculates all the carry bits involved. Once these bits are available to the rest of the circuit, each individual three-bit addition ($X_i + Y_i + \text{carry_in}_i$) is implemented by a simple 3-input XOR gate. The design of the look-ahead carry generator involves two Boolean functions named Generate and Propagate. For each input bits pair these functions are defined as:

$$G_i = X_i \cdot Y_i$$

$$P_i = X_i + Y_i$$

The carry bit c-out(i) generated when adding two bits X_i and Y_i is '1' if the corresponding function G_i is '1' or if the c-out(i-1)='1' and the function $P_i = '1'$ simultaneously. In the first case, the carry bit is activated by the local conditions (the values of X_i and Y_i). In the second, the carry bit is received from the less significant elementary addition and is propagated further to the more significant elementary addition. Therefore, the carry_out bit corresponding to a pair of bits X_i and Y_i is calculated according to the equation:

$$\text{carry_out}(i) = G_i + P_i \cdot \text{carry_in}(i-1)$$

For a four-bit adder the carry-outs are calculated as follows

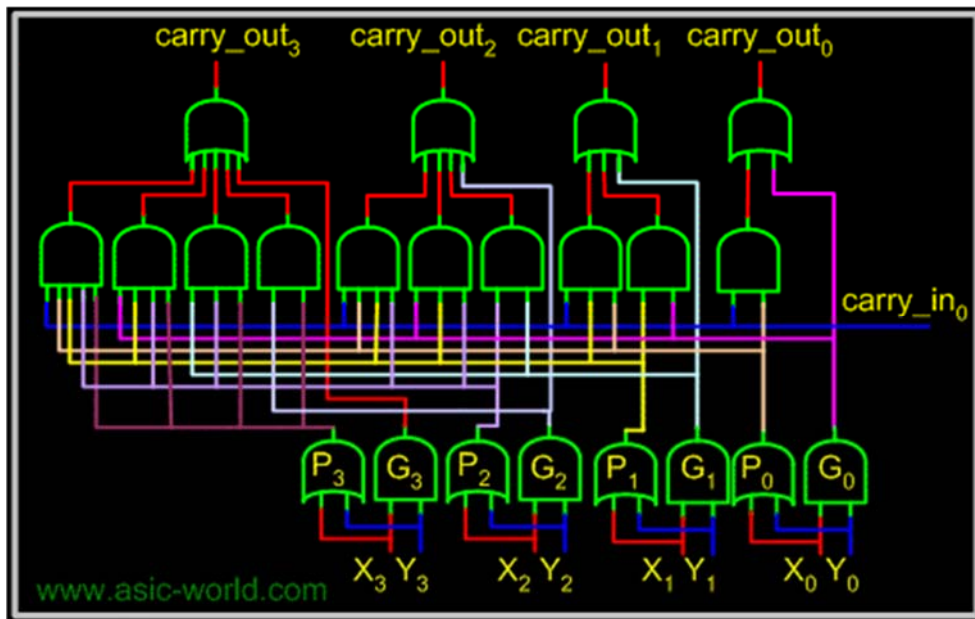
$$\text{carry_out}_0 = G_0 + P_0 \cdot \text{carry_in}_0$$

$$\text{carry_out}_1 = G_1 + P_1 \cdot \text{carry_out}_0 = G_1 + P_1 G_0 + P_1 P_0 \cdot \text{carry_in}_0$$

$$\text{carry_out}_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 \cdot \text{carry_in}_0$$

$$\text{carry_out}_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 \cdot \text{carry_in}_0$$

The set of equations above are implemented by the circuit below and a complete adder with a look-ahead carry generator is next. The input signals need to propagate through a maximum of 4 logic gate in such an adder as opposed to 8 and 12 logic gates in its counterparts illustrated earlier.



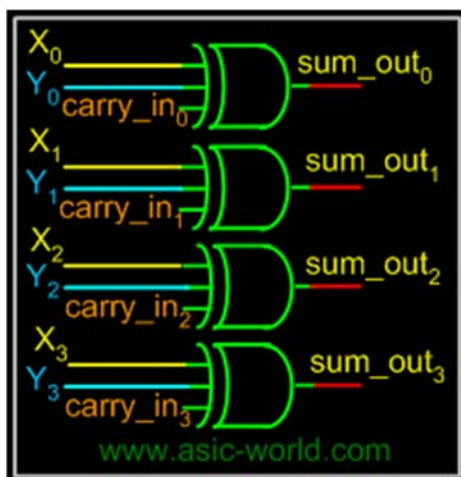
Sums can be calculated from the following equations, where carry_out is taken from the carry calculated in the above circuit.

$$\text{sum_out}_0 = X_0 \oplus Y_0 \oplus \text{carry_out}_0$$

$$\text{sum_out}_1 = X_1 \oplus Y_1 \oplus \text{carry_out}_1$$

$$\text{sum_out}_2 = X_2 \oplus Y_2 \oplus \text{carry_out}_2$$

$$\text{sum_out}_3 = X_3 \oplus Y_3 \oplus \text{carry_out}_3$$



BCD Adder

BCD addition is the same as binary addition with a bit of variation: whenever a sum is greater than 1001, it is not a valid BCD number, so we add 0110 to it, to do the correction. This will produce a carry, which is added to the next BCD position.

- Add the two 4-bit BCD code inputs.
- Determine if the sum of this addition is greater than 1001; if yes, then add 0110 to

this sum and generate a carry to the next decimal position.

Daftar Pustaka

Ronald J. Tocci, Neal S. Widmer, Gregory L. Moss, Digital Systems Principles and Applications TENTH EDITION, 2007, Pearson Education International

<http://www.circuitstoday.com/ripple-carry-adder>



MODUL PERKULIAHAN

SISTEM DIGITAL

Decoder & Encoder

Fakultas
Ilmu Komputer

Program Studi
Teknik Informatika

Tatap Muka

11

Kode MK
15048

Disusun Oleh
Tim Dosen

Abstract

Modul ini membahas tentang decoder dan encoder

Kompetensi

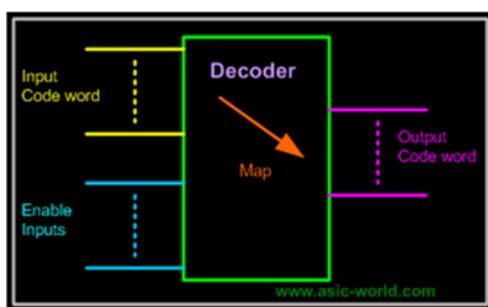
- Mahasiswa diharapkan dapat memahami bagaimana operasi decoder dan encoder
-

Pendahuluan

Dekoder merupakan suatu bentuk rangkaian logika yang berfungsi untuk menyederhanakan input yang masuk (berupa biner) supaya lebih mudah dipahami outputnya (dalam bentuk desimal).

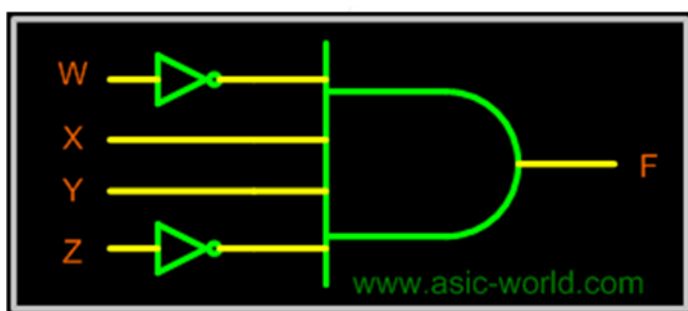
A decoder is a multiple-input, multiple-output logic circuit that converts coded inputs into coded outputs, where the input and output codes are different; e.g. n -to- 2^n , BCD decoders. Enable inputs must be on for the decoder to function, otherwise its outputs assume a single "disabled" output code word.

Decoding is necessary in applications such as data multiplexing, 7 segment display and memory address decoding. Figure below shows the pseudo block of a decoder



Basic Binary Decoder

AND gate can be used as the basic decoding element, because its output is HIGH only when all its inputs are HIGH. For example, if the input binary number is 0110, then, to make all the inputs to the AND gate HIGH, the two outer bits must be inverted using two inverters as shown in figure below



Binary n-to-2ⁿ Decoders

A binary decoder has n inputs and 2^n outputs. Only one output is active at any one time, corresponding to the input value. Figure below shows a representation of Binary n -to- 2^n decoder



Example - 2-to-4 Binary Decoder

A 2 to 4 decoder consists of two inputs and four outputs, truth table and symbols of which is shown below.

Truth Table

X	Y	F0	F1	F2	F3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

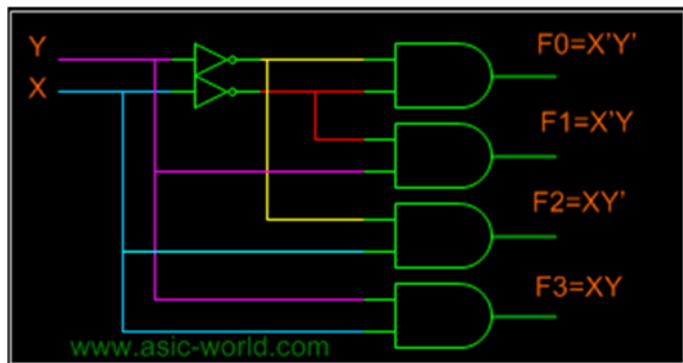
Symbol



To minimize the above truth table we may use kmap, but doing that you will realize that it is a waste of time. One can directly write down the function for each of the outputs. Thus we can draw the circuit as shown in figure below.

Note: Each output is a 2-variable minterm ($X'Y'$, $X'Y$, XY' , XY)

Circuit



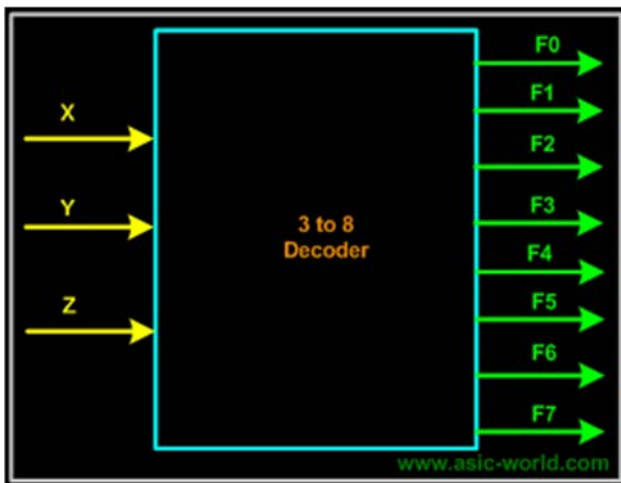
Contoh 3-8 binary decoder

A 3 to 8 decoder consists of three inputs and eight outputs, truth table and symbols of which is shown below.

Truth Table

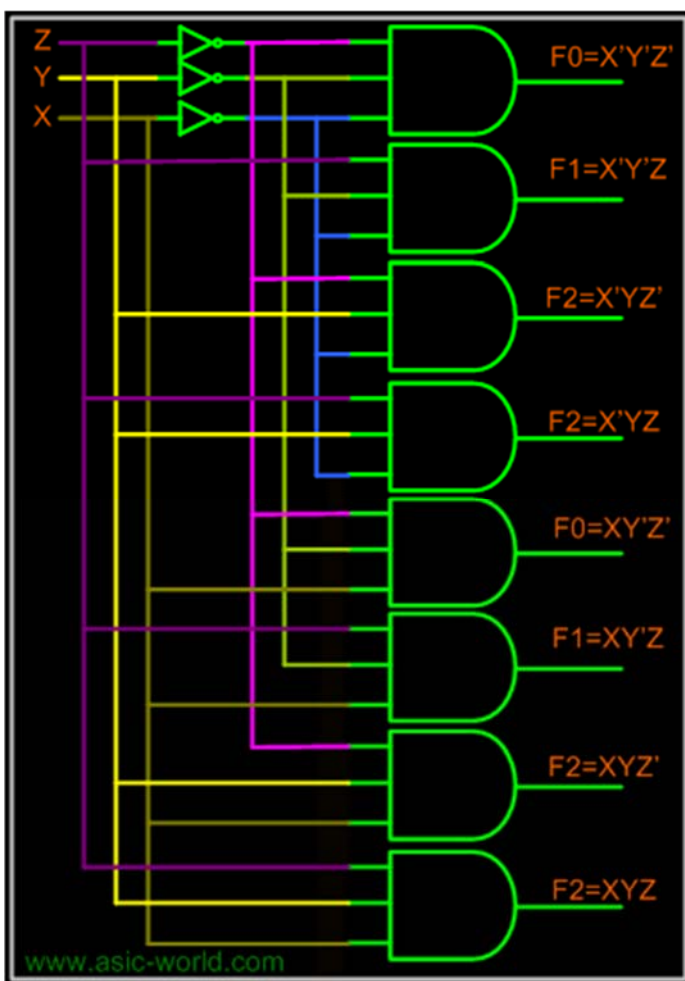
X	Y	Z	F0	F1	F2	F3	F4	F5	F6	F7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Symbol



From the truth table we can draw the circuit diagram as shown in figure below.

Circuit



Implementing Functions Using Decoders

- Any n-variable logic function, in canonical sum-of-minterms form can be implemented using a single n-to- 2^n decoder to generate the minterms, and an OR gate to form the sum.
 - The output lines of the decoder corresponding to the minterms of the function are used as inputs to the or gate.
- Any combinational circuit with n inputs and m outputs can be implemented with an n-to- 2^n decoder with m OR gates.
- Suitable when a circuit has many outputs, and each output function is expressed with few minterms.

Example - Full adder

Equation

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

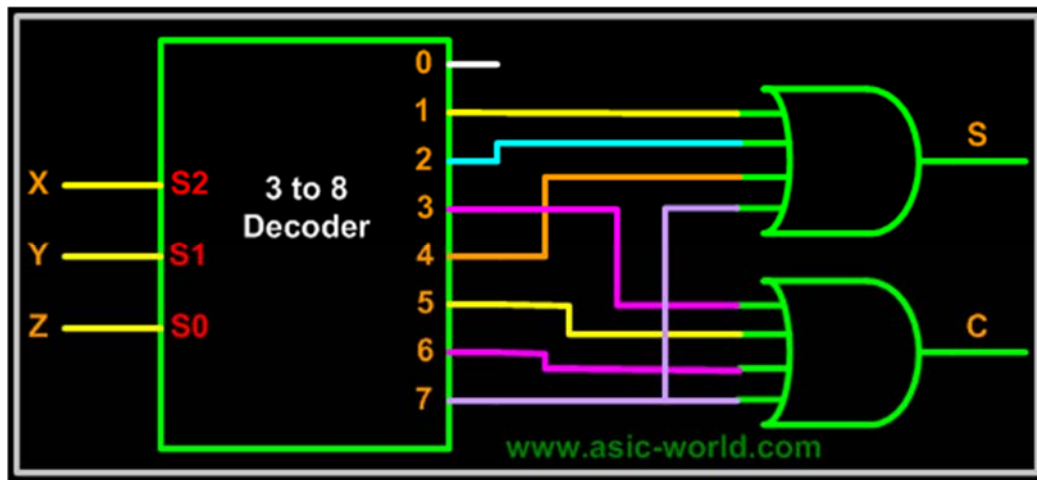
$$C(x, y, z) = \sum(3,5,6,7)$$

Truth Table

X	Y	Z	C	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

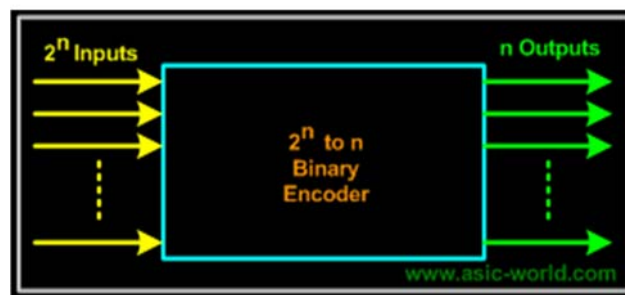
From the truth table we know the values for which the sum (s) is active and also the carry (c) is active. Thus we have the equation as shown above and a circuit can be drawn as shown below from the equation derived.

Circuit



An encoder is a combinational circuit that performs the inverse operation of a decoder. If a device output code has fewer bits than the input code has, the device is usually called an encoder. e.g. 2^n -to- n , priority encoders.

The simplest encoder is a 2^n -to- n binary encoder, where it has only one of 2^n inputs = 1 and the output is the n -bit binary number corresponding to the active input.



Example - Octal-to-Binary Encoder

Octal-to-Binary take 8 inputs and provides 3 outputs, thus doing the opposite of what the 3-to-8 decoder does. At any one time, only one input line has a value of 1. The figure below shows the truth table of an Octal-to-binary encoder.

Truth Table

I0	I1	I2	I3	I4	I5	I6	I7	Y2	Y1	Y0
----	----	----	----	----	----	----	----	----	----	----

1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

For an 8-to-3 binary encoder with inputs I0-I7 the logic expressions of the outputs Y0-Y2 are:

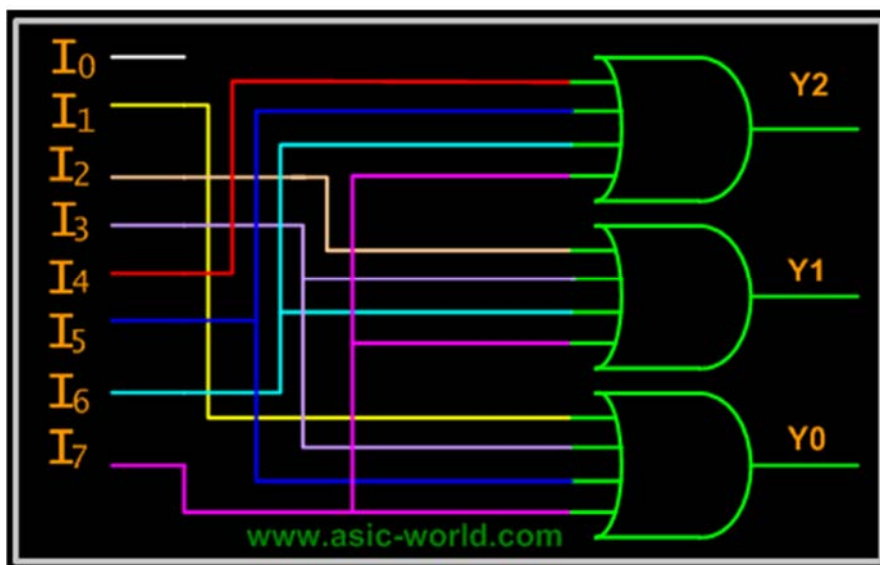
$$Y0 = I1 + I3 + I5 + I7$$

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

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

Based on the above equations, we can draw the circuit as shown below

Circuit



Example - Decimal-to-Binary Encoder

Decimal-to-Binary take 10 inputs and provides 4 outputs, thus doing the opposite of what the 4-to-10 decoder does. At any one time, only one input line has a value of 1. The figure below shows the truth table of a Decimal-to-binary encoder.

Truth Table

I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	Y3	Y2	Y1	Y0
----	----	----	----	----	----	----	----	----	----	----	----	----	----

1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1

From the above truth table , we can derive the functions Y3, Y2, Y1 and Y0 as given below.

$$Y3 = I8 + I9$$

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

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

$$Y0 = I1 + I3 + I5 + I7 + I9$$

Priority Encoder

If we look carefully at the Encoder circuits that we got, we see the following limitations. If more than two inputs are active simultaneously, the output is unpredictable or rather it is not what we expect it to be.

This ambiguity is resolved if priority is established so that only one input is encoded, no matter how many inputs are active at a given point of time.

The priority encoder includes a priority function. The operation of the priority encoder is such that if two or more inputs are active at the same time, the input having the highest priority will take precedence.

Example - 4to3 Priority Encoder

The truth table of a 4-input priority encoder is as shown below. The input D3 has the highest priority, D2 has next highest priority, D0 has the lowest priority. This means output Y2 and Y1 are 0 only when none of the inputs D1, D2, D3 are high and only D0 is high.

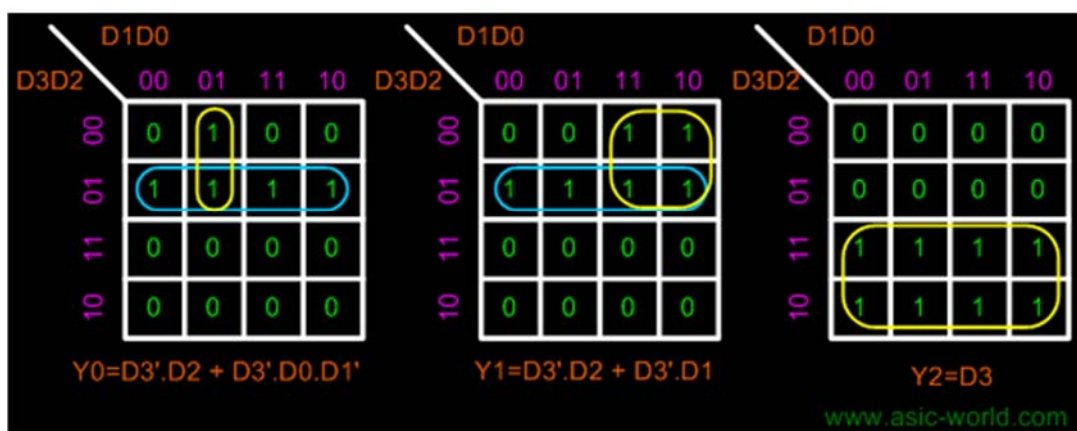
A 4 to 3 encoder consists of four inputs and three outputs, truth table and symbols of which is shown below.

Truth Table

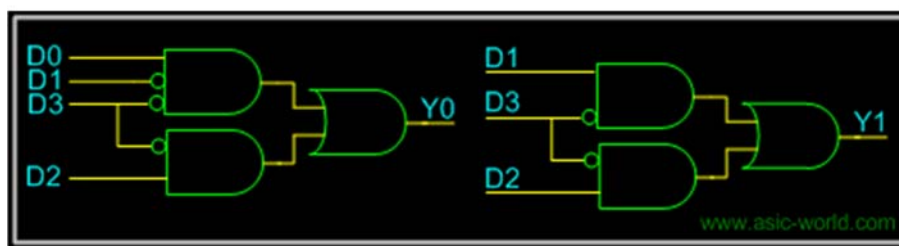
D3	D2	D1	D0	Y2	Y1	Y0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	x	0	1	0
0	1	x	x	0	1	1
1	x	x	x	1	0	0

Now that we have the truth table, we can draw the Kmaps as shown below.

Kmaps



From the Kmap we can draw the circuit as shown below. For Y2, we connect directly to D3.



We can apply the same logic to get higher order priority encoders.

Daftar Pustaka

Ronald J. Tocci, Neal S. Widmer, Gregory L. Moss, Digital Systems Principles and Applications TENTH EDITION, 2007, Pearson Education International

<http://www.circuitstoday.com/ripple-carry-adder>

<http://www.asic-world.com/digital/combo2.html>



MODUL PERKULIAHAN

SISTEM DIGITAL

Multiplexer

Fakultas
Ilmu Komputer

Program Studi
Teknik Informatika

Tatap Muka

12

Kode MK
15048

Disusun Oleh
Tim Dosen

Abstract

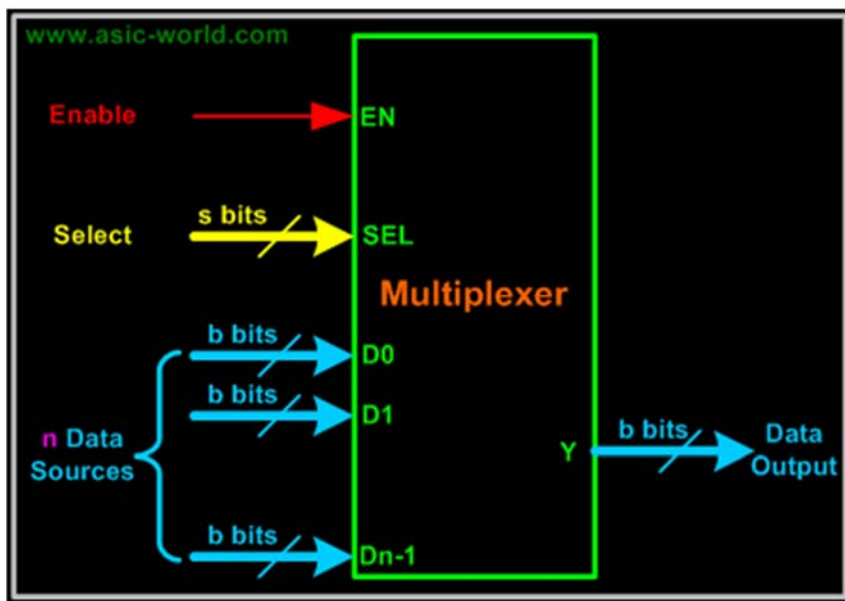
Modul ini membahas tentang Multiplexer

Kompetensi

- Mahasiswa diharapkan dapat memahami bagaimana operasi Multiplexer
-

Multiplexer

A multiplexer (MUX) is a digital switch which connects data from one of n sources to the output. A number of select inputs determine which data source is connected to the output. The block diagram of MUX with n data sources of b bits wide and s bits wide select line is shown in below figure.



MUX acts like a digitally controlled multi-position switch where the binary code applied to the select inputs controls the input source that will be switched on to the output as shown in the figure below. At any given point of time only one input gets selected and is connected to output, based on the select input signal.

Mechanical Equivalent of a Multiplexer

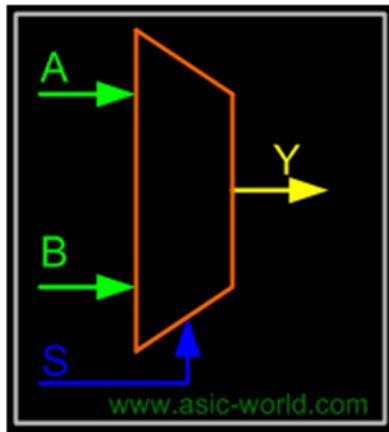
The operation of a multiplexer can be better explained using a mechanical switch as shown in the figure below. This rotary switch can touch any of the inputs, which is connected to the output. As you can see at any given point of time only one input gets transferred to output.



Example - 2x1 MUX

A 2 to 1 line multiplexer is shown in figure below, each 2 input lines A to B is applied to one input of an AND gate. Selection lines S are decoded to select a particular AND gate. The truth table for the 2:1 mux is given in the table below.

Symbol



Truth Table

S	Y
0	A
1	B

Design of a 2:1 Mux

To derive the gate level implementation of 2:1 mux we need to have truth table as shown in figure. And once we have the truth table, we can draw the K-map as shown in figure for all the cases when Y is equal to '1'.

Combining the two 1' as shown in figure, we can drive the output y as shown below

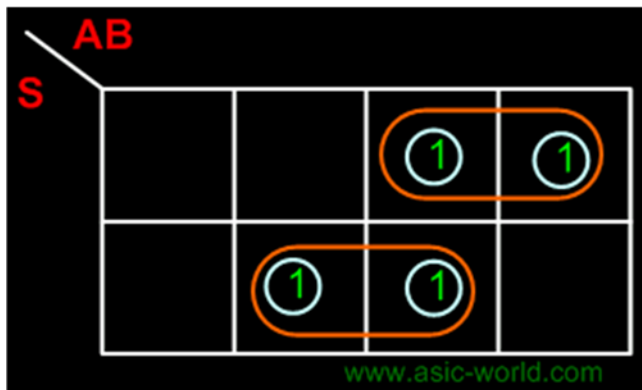
$$Y = A.S' + B.S$$

Truth Table

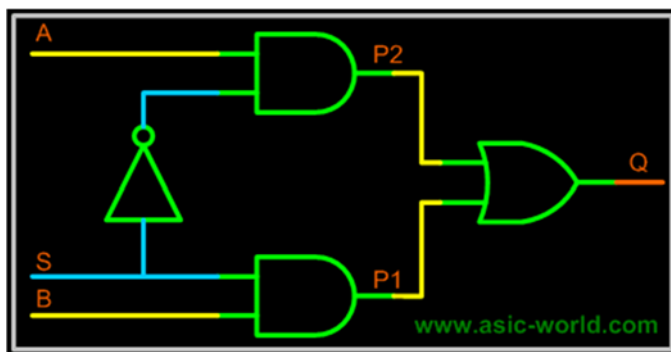
B	A	S	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0

1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Kmap



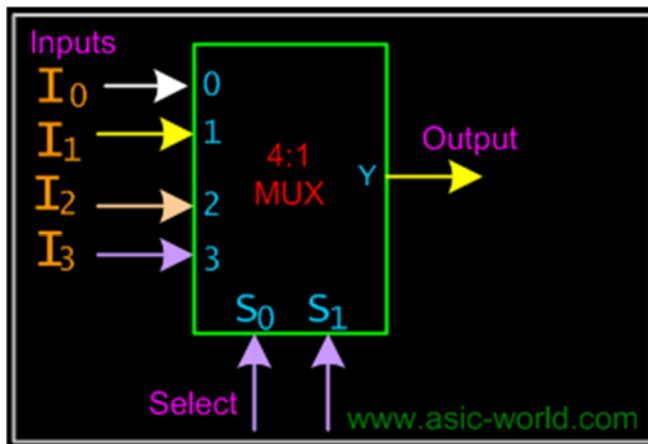
Circuit



Example : 4:1 MUX

A 4 to 1 line multiplexer is shown in figure below, each of 4 input lines I0 to I3 is applied to one input of an AND gate. Selection lines S0 and S1 are decoded to select a particular AND gate. The truth table for the 4:1 mux is given in the table below.

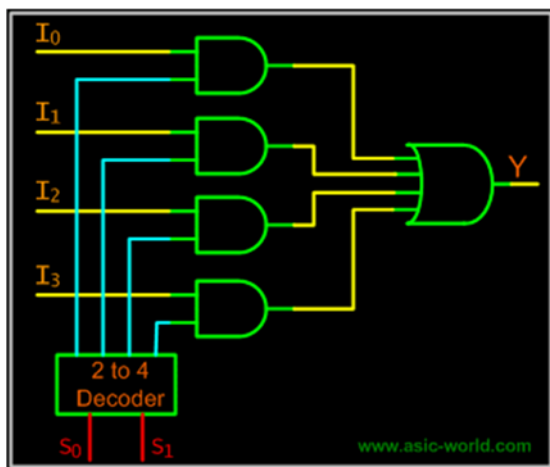
Symbol



Truth Table

S1	S0	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

Circuit



Larger Multiplexers

Larger multiplexers can be constructed from smaller ones. An 8-to-1 multiplexer can be constructed from smaller multiplexers as shown below.

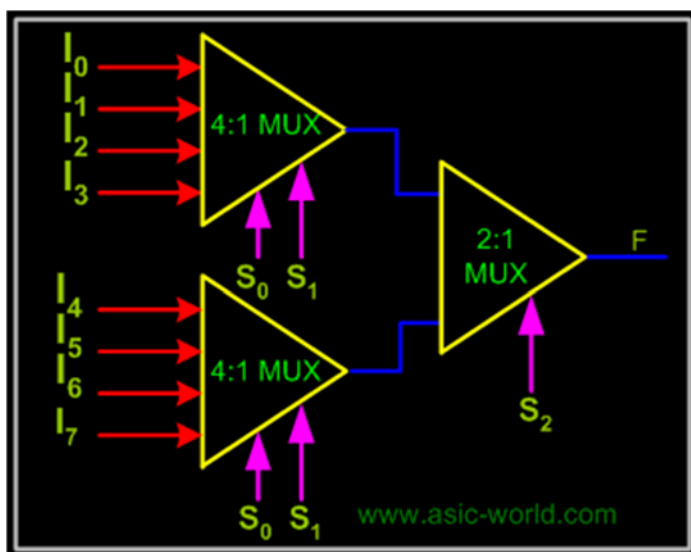
Example - 8-to-1 multiplexer from Smaller MUX

Truth Table

S2	S1	S0	F
0	0	0	I0

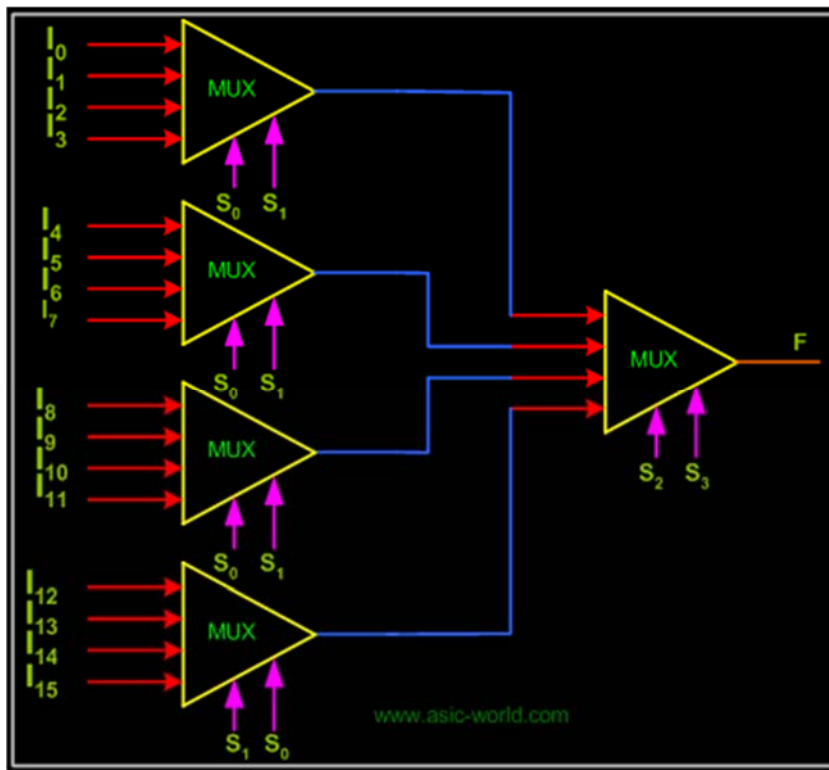
0	0	1	I1
0	1	0	I2
0	1	1	I3
1	0	0	I4
1	0	1	I5
1	1	0	I6
1	1	1	I7

Circuit



Example - 16-to-1 multiplexer from 4:1 mux



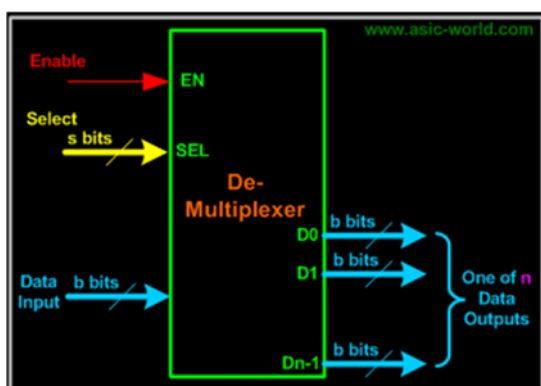


Demultiplexers

They are digital switches which connect data from one input source to one of n outputs.

Usually implemented by using n -to- 2^n binary decoders where the decoder enable line is used for data input of the de-multiplexer.

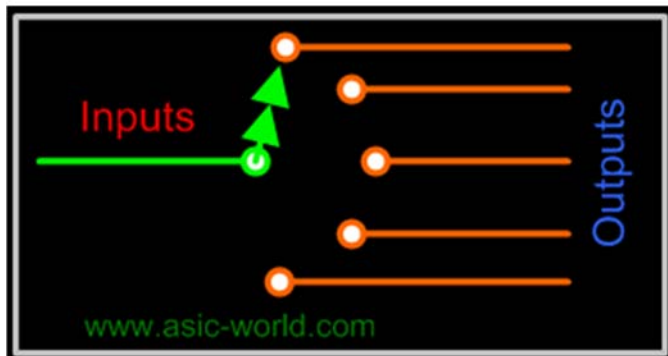
The figure below shows a de-multiplexer block diagram which has got s -bits-wide select input, one b -bits-wide data input and n b -bits-wide outputs.



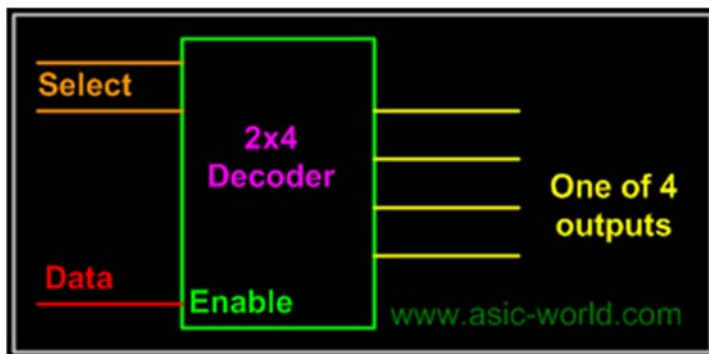
Mechanical Equivalent of a De-Multiplexer

The operation of a de-multiplexer can be better explained using a mechanical switch as shown in the figure below. This rotary switch can touch any of the outputs, which

is connected to the input. As you can see at any given point of time only one output gets connected to input.

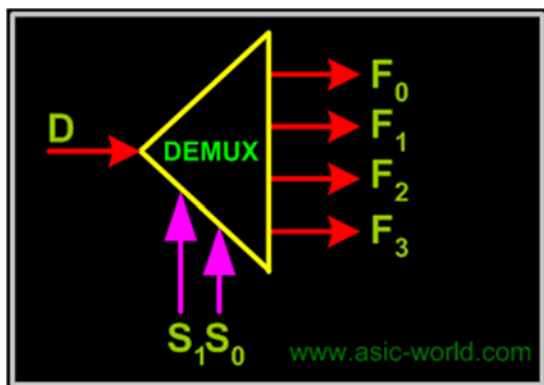


1-bit 4-output de-multiplexer using a 2x4 binary decoder.



Example: 1-to-4 De-multiplexer

Symbol



Truth Table

S1	S0	F0	F1	F2	F3
0	0	D	0	0	0
0	1	0	D	0	0

1	0	0	0	D	0
1	1	0	0	0	D

Daftar Pustaka

Ronald J. Tocci, Neal S. Widmer, Gregory L. Moss, Digital Systems Principles and Applications TENTH EDITION, 2007, Pearson Education International

<http://www.circuitstoday.com/ripple-carry-adder>

<http://www.asic-world.com/digital/combo2.html>



MODUL PERKULIAHAN

SISTEM DIGITAL

Rangkaian Aritmatika 3

Fakultas
Ilmu Komputer

Program Studi
Teknik Informatika

Tatap Muka

Kode MK
15048

Disusun Oleh
Tim Dosen

13

Abstract

Modul ini membahas tentang Multiplier, Divider, Comperator, dan error

Kompetensi

- Mahasiswa diharapkan dapat memahami bagaimana operasi perkalian, pembagian, perbandingan dan deteksi error
-

COMPARATOR

Comparators can compare either a variable number X ($x_n x_{n-1} \dots x_3 x_2 x_1$) with a predefined constant C ($c_n c_{n-1} \dots c_3 c_2 c_1$) or two variable numbers X and Y . In the first case the implementation reduces to a series of cascaded AND and OR logic gates. If the comparator answers the question ' $X > C$?' then its hardware implementation is designed according to the following rules:

- The number X has two types of binary figures: bits corresponding to '1' in the predefined constant and bits corresponding to '0' in the predefined constant.
- The bits of the number X corresponding to '1' are supplied to AND gates
- The bits corresponding to '0' are supplied to OR logic gates
- If the least significant bits of the predefined constant are '10' then bit X_0 is supplied to the same AND gate as bit X_1 .

If the least significant bits of the constant are all '1' then the corresponding bits of the number X are not included in the hardware implementation. All other relations between X and C can be transformed in equivalent ones that use the operator '>' and the NOT logic operator as shown in the table below.

Initial relationship to be tested	Equivalent relationship to be implemented
$X < C$	$\text{NOT } (X > C-1)$
$X \leq C$	$\text{NOT } (X > C)$
$X \geq C$	$X > C-1$

The comparison process of two positive numbers X and Y is performed in a bit-by-bit manner starting with the most significant bit:

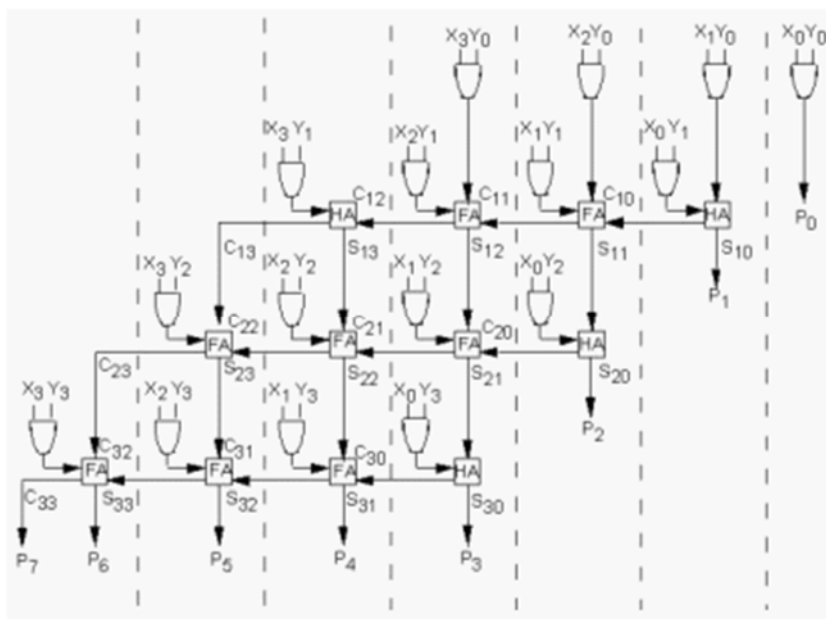
- If the most significant bits are $X_n = '1'$ and $Y_n = '0'$ then number X is larger than Y .
- If $X_n = '0'$ and $Y_n = '1'$ then number X is smaller than Y .
- If $X_n = Y_n$ then no decision can be taken about X and Y based only on these two bits.

If the most significant bits are equal then the result of the comparison is determined by the less significant bits X_{n-1} and Y_{n-1} . If these bits are equal as well, the process continues with the next pair of bits. If all bits are equal then the two numbers are equal.

MULTIPLIER

Multiplication is achieved by adding a list of shifted multiplicands according to the digits of the multiplier. An n-bit X n-bit multiplier can be realized in combinational circuitry by using an array of n-1 n-bit adders where each adder is shifted by one position. For each adder one input is the shifted multiplicand multiplied by 0 or 1 (using AND gates) depending on the multiplier bit, the other input is n partial product bits.

$$\begin{array}{r}
 10011 \times 1001 \\
 \hline
 10011 \leftarrow \\
 00000 \leftarrow \\
 00000 \leftarrow \\
 10011 \leftarrow \\
 \hline
 10101011
 \end{array}$$



Dividers

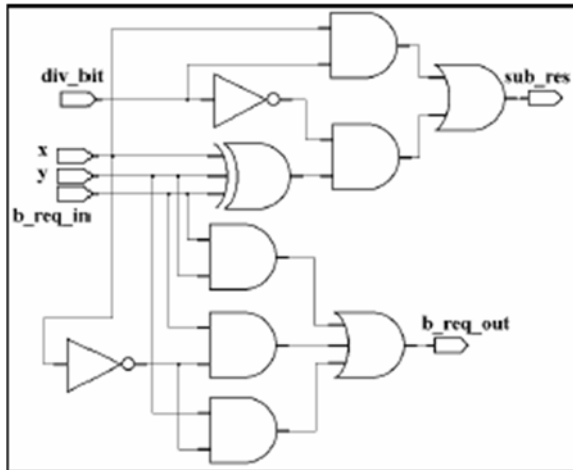
The binary divisions are performed in a very similar manner to the decimal

divisions, as shown in the below figure examples. Thus, the second number is repeatedly subtracted from the figures of the first number after being multiplied either with '1' or with '0'. The multiplication bit ('1' or '0') is selected for each subtraction step in such a manner that the subtraction result is not negative. The division result is composed from all the successive multiplication bits while the remainder is the result of the last subtraction step.

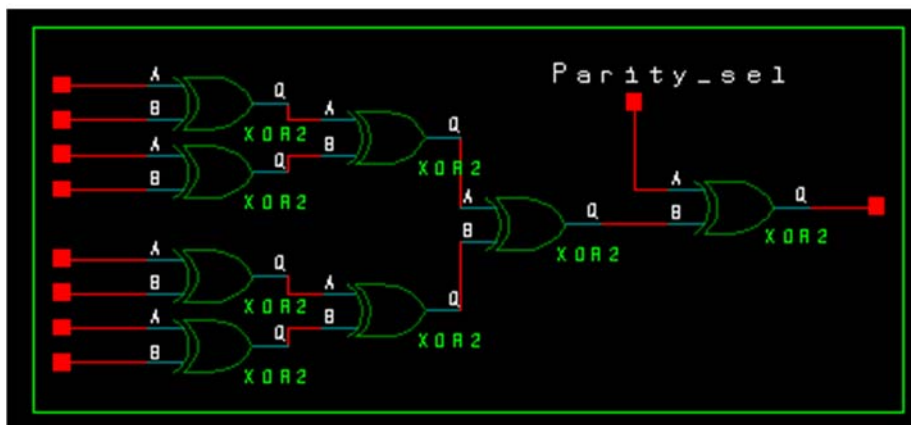
$$\begin{array}{r}
 1101011 : 101 = 10101 \\
 \underline{101 \times 1} \\
 11 \\
 \underline{101 \times 0} \\
 110 \\
 \underline{101 \times 1} \\
 11 \\
 \underline{101 \times 0} \\
 111 \\
 \underline{101 \times 1} \\
 10
 \end{array}$$

This algorithm can be implemented by a series of subtractors composed of modified elementary cells. Each subtractor calculates the difference between two input numbers, but if the result is negative the operation is canceled and replaced with a subtraction by zero. Thus, each divider cell has the normal inputs of a subtracter unit as in the figure below but a supplementary input ('div_bit') is also present. This input is connected to the b_req_out signal generated by the most significant cell of the subtracter. If this signal is '1', the initial subtraction result is negative and it has to be replaced with a subtraction by zero. Inside each divider cell the div_bit signal controls an equivalent 2:1 multiplexer that selects between bit 'x' and the bit included in the subtraction result X-Y. The complete division can therefore be implemented by a matrix of divider cells connected on rows and columns as shown in figure below. Each row performs one multiplication-and-subtraction cycle where the multiplication bit is supplied by the NOT logic gate at the end of each row. Therefor the NOT logic gates generate the bits of the division result.





Parity Circuit



SUBTRACTER

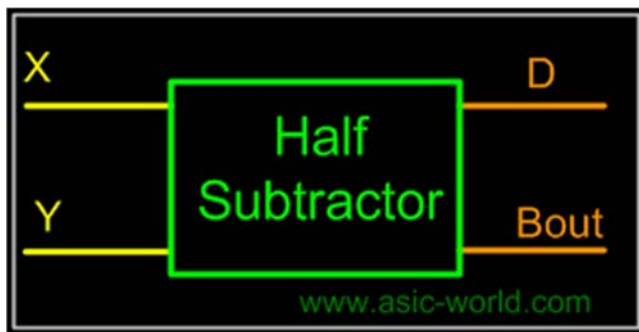
Subtractor circuits take two binary numbers as input and subtract one binary number input from the other binary number input. Similar to adders, it gives out two outputs, difference and borrow (carry-in the case of Adder). There are two types of subtracters.

- Half Subtractor.
- Full Subtractor.

Half Subtractor

The half-subtractor is a combinational circuit which is used to perform subtraction of two bits. It has two inputs, X (minuend) and Y (subtrahend) and two outputs D (difference) and B (borrow). The logic symbol and truth table are shown below.

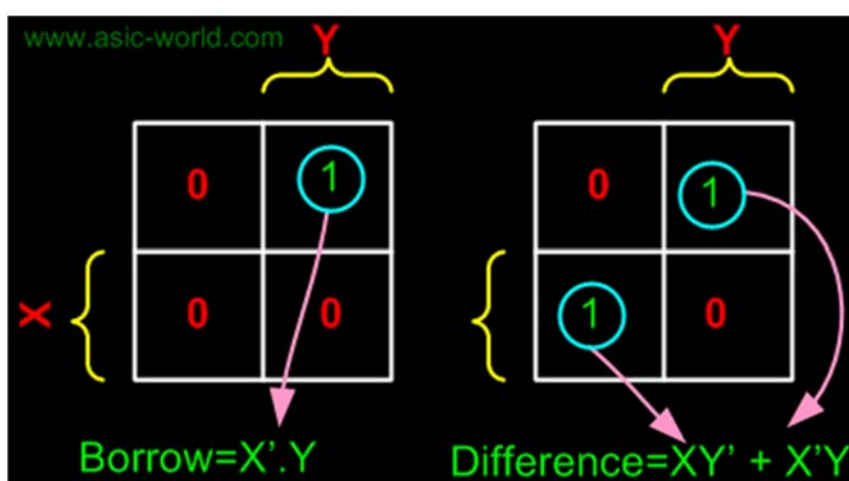
Symbol



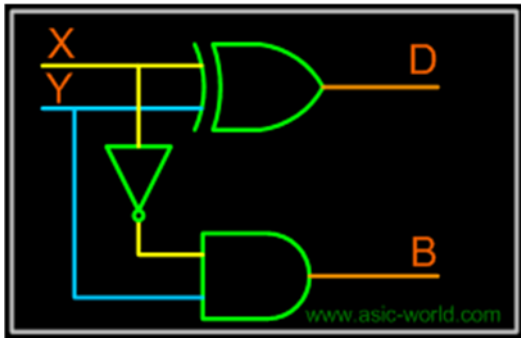
Truth Table

X	Y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

From the above table we can draw the Kmap as shown below for "difference" and "borrow". The boolean expression for the difference and Borrow can be written.



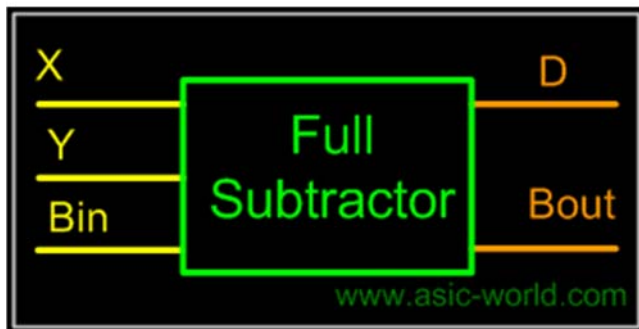
From the equation we can draw the half-subtractor as shown in the figure below.



Full Subtractor

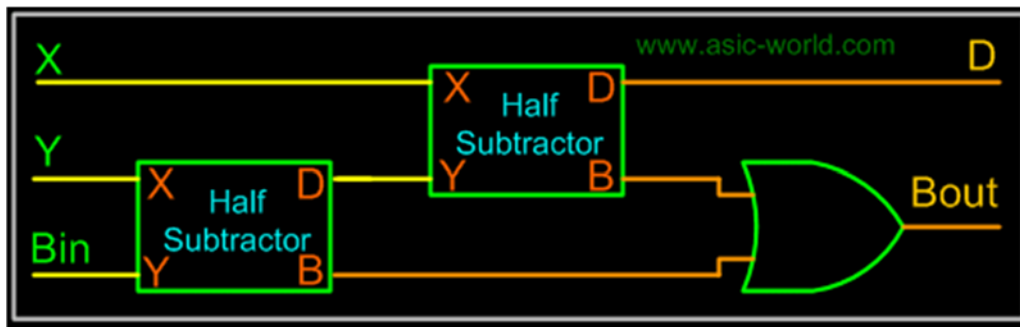
A full subtractor is a combinational circuit that performs subtraction involving three bits, namely minuend, subtrahend, and borrow-in. The logic symbol and truth table are shown below.

Symbol



Truth Table

X	Y	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



From above table we can draw the Kmap as shown below for "difference" and "borrow". The boolean expression for difference and borrow can be written.

$$D = X'Y'Bin + X'YBin' + XY'Bin' + XYBin$$

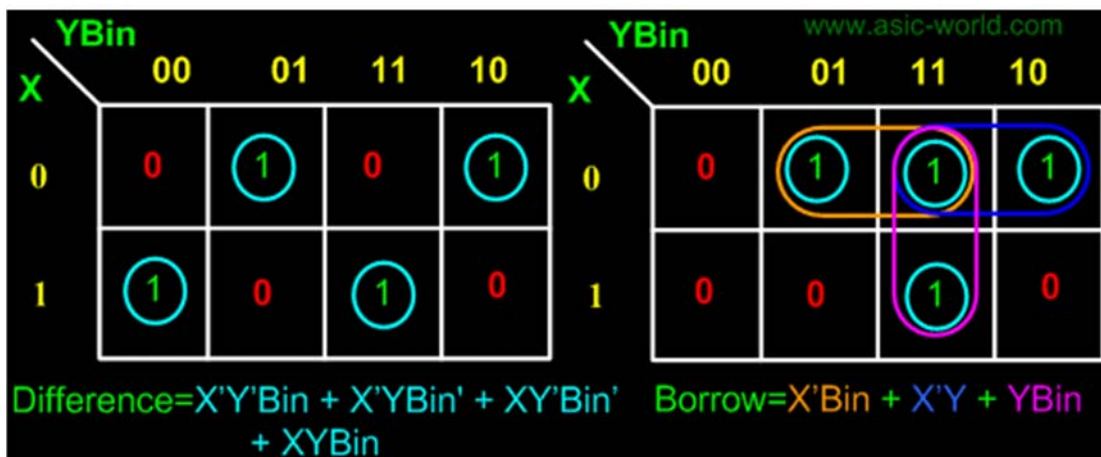
$$= (X'Y' + XY)Bin + (X'Y + XY')Bin'$$

$$= (X \oplus Y)'Bin + (X \oplus Y)Bin'$$

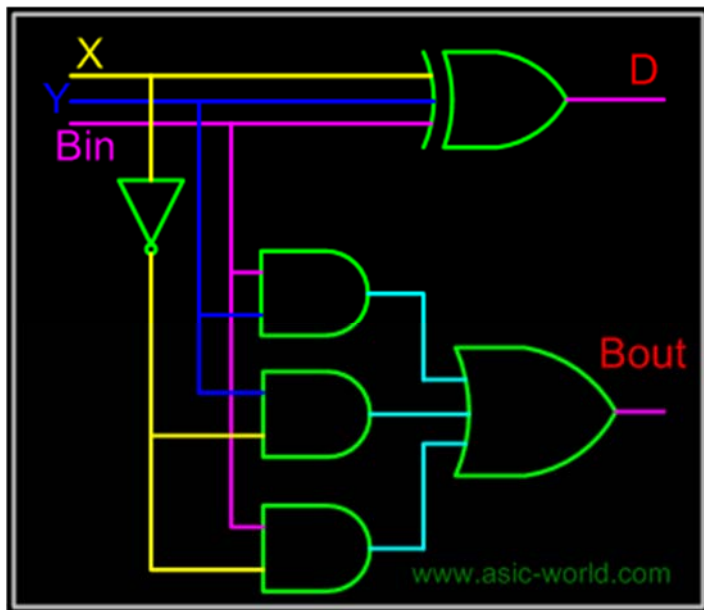
$$= X \oplus Y \oplus Bin$$

$$Bout = X'.Y + X'.Bin + Y.Bin$$

From the equation we can draw the half-subtractor as shown in figure below.



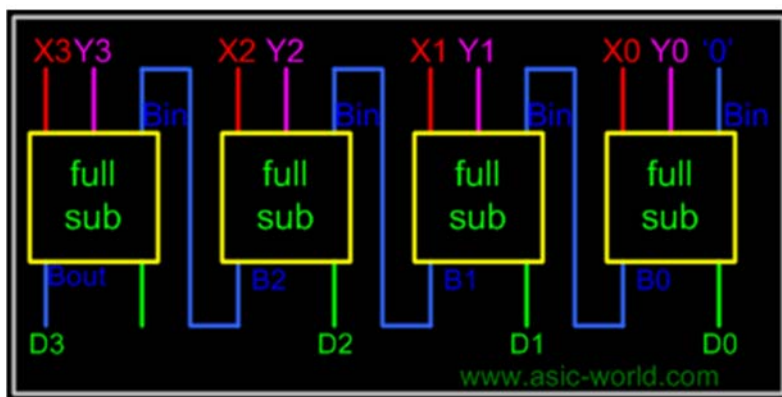
From the above expression, we can draw the circuit below. If you look carefully, you will see that a full-subtractor circuit is more or less same as a full-adder with slight modification.



Parallel Binary Subtractor

Parallel binary subtractor can be implemented by cascading several full-subtractors. Implementation and associated problems are those of a parallel binary adder, seen before in parallel binary adder section.

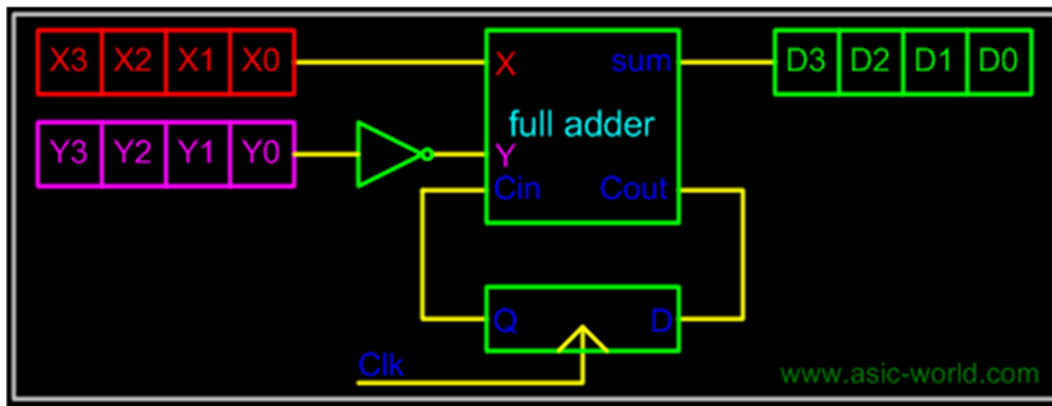
Below is the block level representation of a 4-bit parallel binary subtractor, which subtracts 4-bit $Y_3Y_2Y_1Y_0$ from 4-bit $X_3X_2X_1X_0$. It has 4-bit difference output $D_3D_2D_1D_0$ with borrow output Bout.



Serial Binary Subtractor

A serial subtractor can be obtained by converting the serial adder using the 2's complement system. The subtrahend is stored in the Y register and must be 2's complemented before it is added to the minuend stored in the X register.

The circuit for a 4-bit serial subtractor using full-adder is shown in the figure below.



Daftar Pustaka

Ronald J. Tocci, Neal S. Widmer, Gregory L. Moss, Digital Systems Principles and Applications TENTH EDITION, 2007, Pearson Education International

<http://www.circuitstoday.com/ripple-carry-adder>

<http://www.asic-world.com/digital/combo2.html>



MODUL PERKULIAHAN

SISTEM DIGITAL

MEMORY

Fakultas
Ilmu Komputer

Program Studi
Teknik Informatika

Tatap Muka

14

Kode MK
15048

Disusun Oleh
Tim Dosen

Abstract

Modul ini membahas tentang

Kompetensi

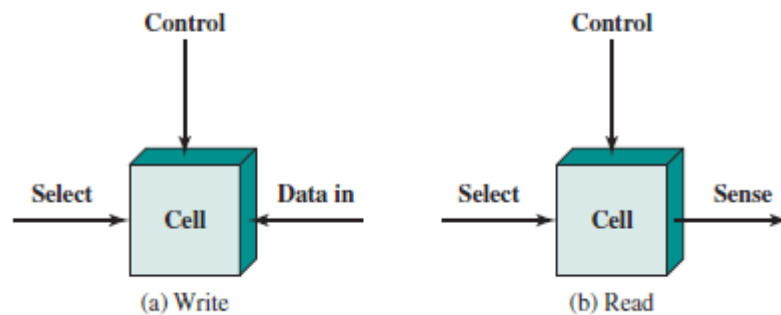
-

Main Memory Semi Konduktor

Organisasi

Elemen dasar dari sebuah memori semikonduktor adalah **memory cell**. **Memory cell** memiliki ciri ciri sebagai berikut:

- Memiliki 2 state yaitu 1 atau 0
- Mampu melakukan operasi write
- Mampu melakukan operasi read



Gambar 6.1 Operasi memory cell

Gambar 6.1 merupakan operasi pada memory cell. Pada umumnya memory cell memiliki 3 fungsi terminal yang mampu membawa sebuah sinyal elektrik.

- Select terminal → memilih sebuah memory cell untuk operasi baca atau tulis
- Control terminal → mengindikasikan read atau write. Untuk operasi write, terminal lain akan menyediakan sebuah sinyal elektrik yang di menset state dari cell ke 1 atau 0. Untuk operasi read, terminal digunakan untuk output cell state



DRAM dan SRAM

Tabel 6.1 jenis jenis memori semikonduktor

Memory Type	Category	Erasure	Write Mechanism	Volatility
Random-access memory (RAM)	Read-write memory	Electrically, byte-level	Electrically	Volatile
Read-only memory (ROM)	Read-only memory	Not possible	Masks	Nonvolatile
Programmable ROM (PROM)			Electrically	
Erasable PROM (EPROM)	UV light, chip-level			
Electrically Erasable PROM (EEPROM)	Electrically, byte-level			
Flash memory	Electrically, block-level			

Dynamic RAM (DRAM)

Teknologi RAM dibagi menjadi 2 yaitu dynamic dan static. Dynamic RAM (DRAM) dibuat dari cell yang menyimpan data seperti charge kapasitor. Ada atau tidak ada charge pada sebuah kapasitor diwakili oleh bilangan biner 1 atau 0. Karena kapasitor memiliki sebuah kecenderungan untuk discharge, maka DRAM memerlukan charge secara periodic untuk mempertahankan penyimpanan data. Istilah dynamic berasal dari kecenderungan charge melemah, walaupun power tersedia terus.

Gambar 6.2a merupakan struktur DRAM untuk individu cell yang menyimpan 1 bit. address line diaktifkan pada saat nilai bit dari cell ini di read atau write. Transistor bertindak sebagai switch yang menutup (close, mengizinkan ada aliran) jika tegangan diberikan ke address line dan open (tidak ada aliran) jika tidak ada tegangan pada address line.

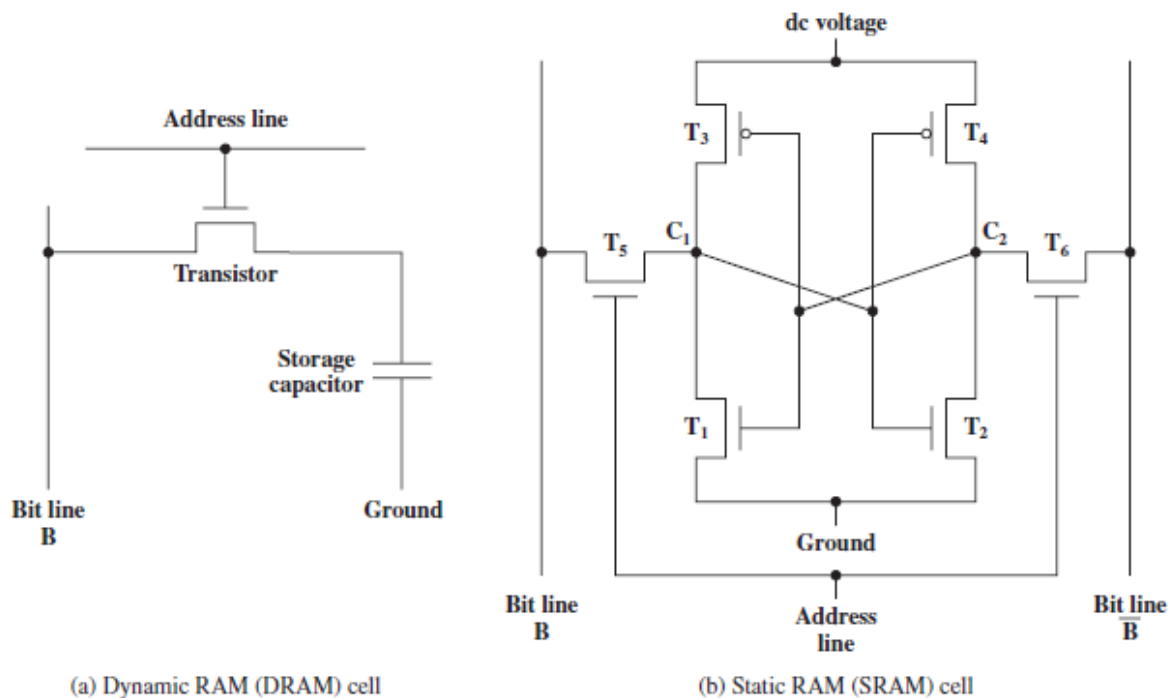
Operasi write

Sebuah sinyal tegangan diberikan ke bit line, tegangan tinggi diwakili 1 dan tegangan rendah diwakili 0. Sebuah sinyal diberikan ke address line, mengizinkan sebuah charge untuk ditransfer ke kapasitor.

Operasi read

Pada saat address line dipilih, transistor aktif dan charge disimpan kedalam kapasitor dan ke sebuah bit line serta ke sense amplifier. Sense amplifier membandingkan tegangan kapasitor dengan sebuah nilai referensi dan menentukan jika cell berisi

logic 0 atau logic 1. Readout cell menyebabkan kapasitor discharge, sehingga harus di restore untuk menyelesaikan operasi.



Gambar 6.2 Struktur Cell Memory

Static RAM

Static RAM merupakan sebuah perangkat digital yang menggunakan beberapa elemen logika yang digunakan dalam processor. Nilai bilangan biner disimpan menggunakan konfigurasi gerbang logika flip flop tradisional. Static RAM akan menyimpan data selama power ada.

Gambar 6.2b merupakan struktur RAM untuk sebuah cell individu. Ada 4 transistor (T1, T2, T3, T4) yang dihubungkan secara silang agar menghasilkan sebuah logic state yang stabil.

- Pada logic 1, Point C1 tinggi dan point C2 rendah. Pada state ini, T1 dan T4 off sedangkan T2 dan T3 on
- Pada Logic 0, point C1 low dan point C2 high, pada statet ini T1,T4 on dan T2 T3 off

Kedua state stabil jika tegangan DC (Direct Current) ada.

SRAM address line digunakan untuk open atau close sebuah switch. Pada saat sinyal diberikan ke line ini, 2 transistor di on kan, dan mengizinkan operasi read atau write. Untuk operasi write, nilai bit yang diinginkan di berikan ke line B, sementara komplemennya diberikan ke line B. Operasi read, nilai bit dibaca dari line B

Jenis Jenis ROM

Read Only Memory (ROM) berisi data permanen yang tidak dapat dirubah. ROM bersifat nonvolatile, dimana tidak memerlukan sumber listrik untuk mempertahankan nilai nilai bit dalam memory. ROM hanya bias dibaca (read) dan tidak bias ditulis (Write) data baru kedalamnya. Salah satu aplikasi penting dari ROM adalah microprogramming, dan aplikasi lainnya adalah :

- Library subroutine untuk fungsi fungsi yang seringkali diperlukan
- Program system
- Table table fungsi.

Keuntungan dari ROM adalah data atau program permanen didalam main memory dan tidak perlu di load dari sebuah perangkat penyimpanan eksternal. ROM merupakan sebuah chip dengan rangaian terintegrasi, dimana data dialirkan ke chip sebagai bagian dari proses pabrikasi. Ada 2 masalah terkait hal ini :

1. Langkah untuk memasukkan data memerlukan biaya besar.
2. Tidak boleh ada error, jika satu bit salah, maka keseluruhan proses ROM harus diulang.

Programmable ROM

PROM bersifat nonvolatile dan hanya ditulis satu kali. Proses penulisan dilakukan secara elektrik dan dapat dilakukan oleh supplier atau pelanggan pada waktu berikutnya selain dari chip yang asli pabrik. Diperlukan sebuah perangkat khusus untuk proses menulis atau “programming”. PROM lebih fleksibel dan nyaman.

Read Mostly Memory

Berfungsi untuk aplikasi dimana operasi read jauh lebih sering daripada operasi write, tetapi diperlukan penyimpanan yang bersifat nonvolatile. Ada 3 bentuk umum dari read mostly memory yaitu :

- EPROM (erasable programmable read only memory)
EPROM ditulis dan dibaca secara elektrik seperti halnya PROM. Sebelum operasi tulis, semua data dalam cell penyimpanan harus dihapus agar state inisialisasi nya sama dengan menyinari chip dengan radiasi ultra violet. Proses penghapusan dilakukan dengan menyinari sinar ultraviolet melalui sebuah jendela yang didesain ke chip memory. Proses penghapusan ini dapat dilakukan secara berulang, setiap

penghapusan memerlukan waktu sekitar 20menit. EPROM lebih mahal dibandingkan PROM, tetapi memiliki kemampuan multiple update.

- EEPROM (Electrically Erasable Programmable ROM)

Read Mostly Memory ini dapat ditulis setiap saat tanpa harus dihapus terlebih dahulu, hanya dengan memperbaharui byte atau byte address. Operasi write memerlukan waktu yang lebih lama dibandingkan operasi read (beberapa microsecond per byte). EEPROM menggabungkan keuntungan nonvolatile dengan fleksibilitas pembaharuan menggunakan bus control, address dan data line. EEPROM lebih mahal dibandingkan dengan EPROM.

- Flash memory

Diperkenalkan pertama kali pada pertengahan tahun 1980, flash memory merupakan gabungan dari EPROM dan EEPROM dari segi harga dan fungsionalitas. Seperti EEPROM flash memory menggunakan teknologi penghapusan elektrik. Seluruh flash memory dapat dihapus dalam waktu 1 atau 2 detik, lebih cepat dibandingkan EPROM. Flash memory dapat menghapus per block memory. Dinamakan Flash memory karena microchip disusun dari bagian cell memory yang dihapus dengan satu tindakan atau flash.

ERROR CORRECTING

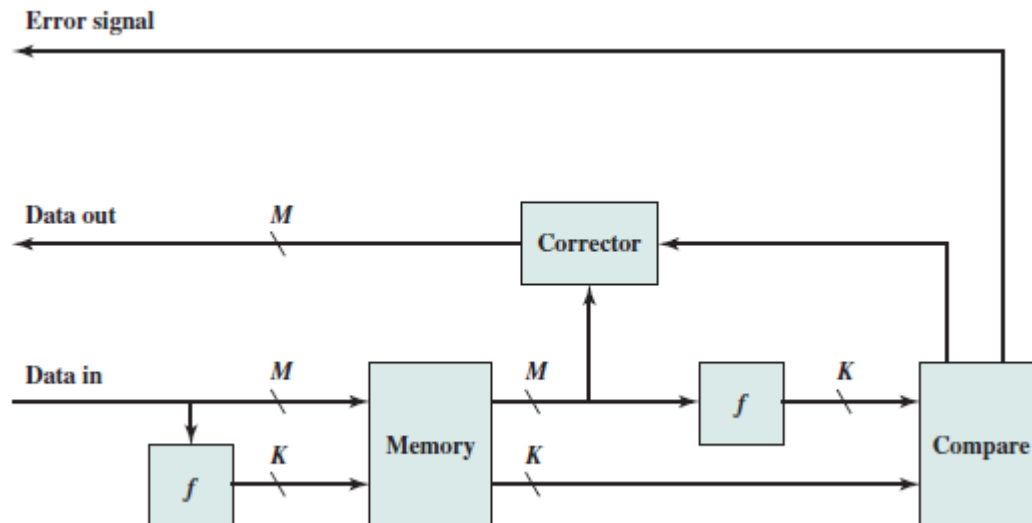
System memory semikonduktor merupakan subjek untuk terjadinya error. Error yang terjadi dikategorikan sebagai hard failure dan soft error.

Hard failure merupakan kerusakan fisik permanen sehingga cell memory tidak reliable menyimpan data karena menjadi stuck pada 0 atau 1 atau switch tak menentu antara 0 dan 1. Hard error dapat disebabkan oleh penyalahgunaan, proses pabrikasi dan penggunaan.

Soft error merupakan kejadian kejadian yang menyebabkan perubahan isi salah satu atau banyak cell memory tanpa merusak memory. Error ini dapat disebabkan karena masalah sumber daya atau partikel alpha. Partikel ini merupakan hasil dari kerusakan radioaktif.

Sebagian besar system memory memiliki logic untuk mendeteksi dan mengoreksi error tersebut.

Gambar 6.3 memperlihatkan proses pada saat data ditulis dalam memory, sebuah perhitungan (fungsi f) dilakukan pada data untuk menghasilkan sebuah kode. Kedua Kode dan data disimpan. Jika sebuah M bit word data disimpan dan kode dengan panjang K bit, maka ukuran dari word yang disimpan adalah $M+K$ bit.



Gambar 6.3 Fungsi error correcting code.

Pada saat word yang sebelumnya disimpan dibaca, kode digunakan untuk mendeteksi dan mengoreksi error. Sebuah kode K bit baru akan dihasilkan dari M bit data, dan dilakukan proses perbandingan dengan kode bit sebelumnya. Perbandingan ini menghasilkan 3 kesimpulan :

- Terdeteksi No error. Data bit yang tadi dijemput akan dikirim.
- Terdeteksi error, dan kemungkinan untuk diperbaiki. Data bit plus error correction bit akan dikirim ke sebuah corrector, menghasilkan serangkaian M bit yang sudah dikoreksi, dan baru dikirim
- Terdeteksi error, tetapi tidak bisa diperbaiki. Kondisi ini dilaporkan.

Kode kode yang beroperasi disebut sebagai error correcting codes. Sebuah kode dikarakteristikan dengan sejumlah bit error dalam sebuah word, dan dapat dikoreksi atau dideteksi.

Hamming Code

Error correcting code paling sederhana adalah hamming code yang ditemukan oleh Richard Hamming pada Bell Laboratories.

Tabel 6.2 , perbandingan jumlah cek bit dengan bit data

Data Bits	Single-Error Correction		Single-Error Correction/ Double-Error Detection	
	Check Bits	% Increase	Check Bits	% Increase
8	4	50	5	62.5
16	5	31.25	6	37.5
32	6	18.75	7	21.875
64	7	10.94	8	12.5
128	8	6.25	9	7.03
256	9	3.52	10	3.91

CONTOH

8 bit data + 4 cek bit, total word adalah 12 data, sehingga posisi nya adalah sebagai berikut:

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check bit					C8				C4		C2	C1

Setiap Cek bit akan diposisikan pada posisi yang memiliki jumlah bit 1 nya hanya satu. Yaitu posisi 1 (0001), posisi 2(0010), posisi 4(0100), posisi 8(1000). Dan sisanya adalah posisi bit data

Untuk memperoleh nilai cek bit maka dilakukan operasi logika X-OR seperti berikut

$$\begin{aligned}C1 &= D1 \oplus D2 \oplus D4 \oplus D5 \oplus D7 \\C2 &= D1 \oplus D3 \oplus D4 \oplus D6 \oplus D7 \\C4 &= D2 \oplus D3 \oplus D4 \oplus D8 \\C8 &= D5 \oplus D6 \oplus D7 \oplus D8\end{aligned}$$

CONTOH SOAL

Sebuah word data 8 bit 00111001, perhitungan error correction bit nya adalah sebagai berikut.

- Posisi bit data

Posisi bit	12	11	10	9	8	7	6	5	4	3	2	1
	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Posisi data	0	0	1	1		1	0	0		1		
Posisi cek bit												

- Menentukan nilai cek bit dengan cara melakukan operasi X-or berikut

$$\begin{aligned}
 C1 &= D1 \oplus D2 \oplus D4 \oplus D5 \oplus D7 \\
 C2 &= D1 \oplus D3 \oplus D4 \oplus D6 \oplus D7 \\
 C4 &= D2 \oplus D3 \oplus D4 \oplus D8 \\
 C8 &= D5 \oplus D6 \oplus D7 \oplus D8
 \end{aligned}$$

$$C1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$$

$$C2 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$$

$$C4 = 0 \oplus 0 \oplus 1 \oplus 0 = 1$$

$$C8 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$$

NOTE :

C1 berada pada posisi bit 1 (0001) dimana terdapat bit 1 pada bagian kanan (LSB=Least Significant Bit). Sehingga untuk mendapatkan nilai Cek bit C1, maka semua data pada posisi bit yang memiliki bit 1 pada bagian kanan dihitung dengan operasi X-or

C2 berada pada posisi bit 2 (0010), dimana terdapat bit 1 pada posisi kedua, sehingga untuk mendapatkan nilai Cek Bit C2, maka semua data pada posisi bit yang memiliki bit 1 pada posisi kedua dihitung dengan operasi X-or.

Posisi bit	12	11	10	9	8	7	6	5	4	3	2	1
	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Posisi data	D8	D7	D6	D5		D4	D3	D2		D1		
C1	0	0	1	1		1	0	0		1		
C2	0	0	1	1		1	0	0		1		
C3	0	0	1	1		1	0	0		1		
C4	0	0	1	1		1	0	0		1		

- Diperoleh data + cek bit adalah

Posisi bit	12	11	10	9	8	7	6	5	4	3	2	1
	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Posisi data	0	0	1	1		1	0	0		1		
Cek Bit					0				1		1	1
Data tersimpan	0	0	1	1	0	1	0	0	1	1	1	1

Jika misalnya data yang diambil (difetch) adalah 0011 0110 1111. Maka proses yang sama terjadi terhadap data yang baru di fetch.

1. Memisahkan data dan cek bit untuk data yang baru di ambil

Posisi bit	12	11	10	9	8	7	6	5	4	3	2	1
	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Posisi data	0	0	1	1		1	1	0		1		
Cek Bit					0				1		1	1

2. Melakukan proses perhitungan cek bit untuk data yang baru diambil

$$C1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$$

$$C2 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0$$

$$C4 = 0 \oplus 1 \oplus 1 \oplus 0 = 0$$

$$C8 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$$

3. Melakukan proses perbandingan terhadap cek bit data yang tersimpan dan cek data yang baru dijemput dengan operasi X-OR

$$\begin{array}{rcccc}
 & C8 & C4 & C2 & C1 \\
 & 0 & 1 & 1 & 1 \\
 \oplus & 0 & 0 & 0 & 1 \\
 \hline
 & 0 & 1 & 1 & 0
 \end{array}$$

Hasilnya adalah 0110 (6), maka bit posisi 6 error.

Ada 3 kemungkinan hasil yang akan diperoleh :

- Jika hasilnya adaah 0, maka tidak terjadi error
- Jika hasilnya terdiri dari 1 dan hanya 1 bit set ke 1, maka error yang terjadi pada salah satu cek bit, tidak diperlukan koreksi.
- Jika hasilnya terdiri dari lebih satu bit yang di set 1, maka hasilnya merupakan indikasi posisi bit error pada data. Data bit akan dikoreksi dengan cara invers (pembalikan nilai).

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check bit					C8				C4		C2	C1
Word stored as	0	0	1	1	0	1	0	0	1	1	1	1
Word fetched as	0	0	1	1	0	1	1	0	1	1	1	1
Position number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Check bit					0				0		0	1