

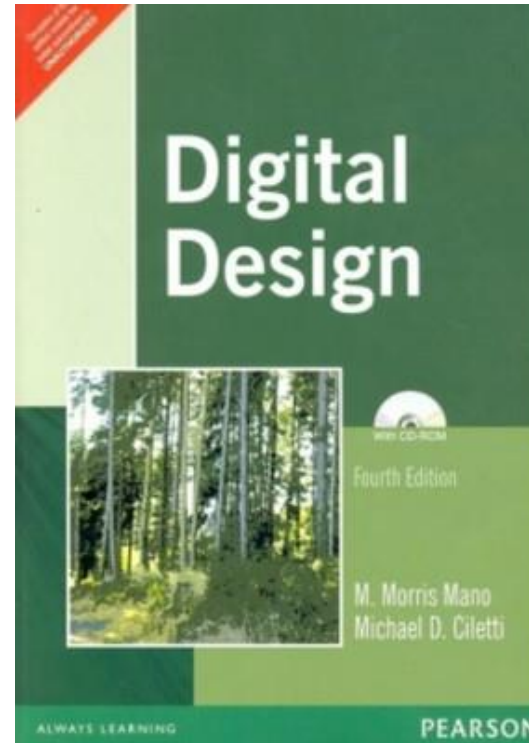
Sayısal Sistemler-H7CD1

Vize Öncesi Genel Tekrar-1

Dr. Meriç Çetin
versiyon251020

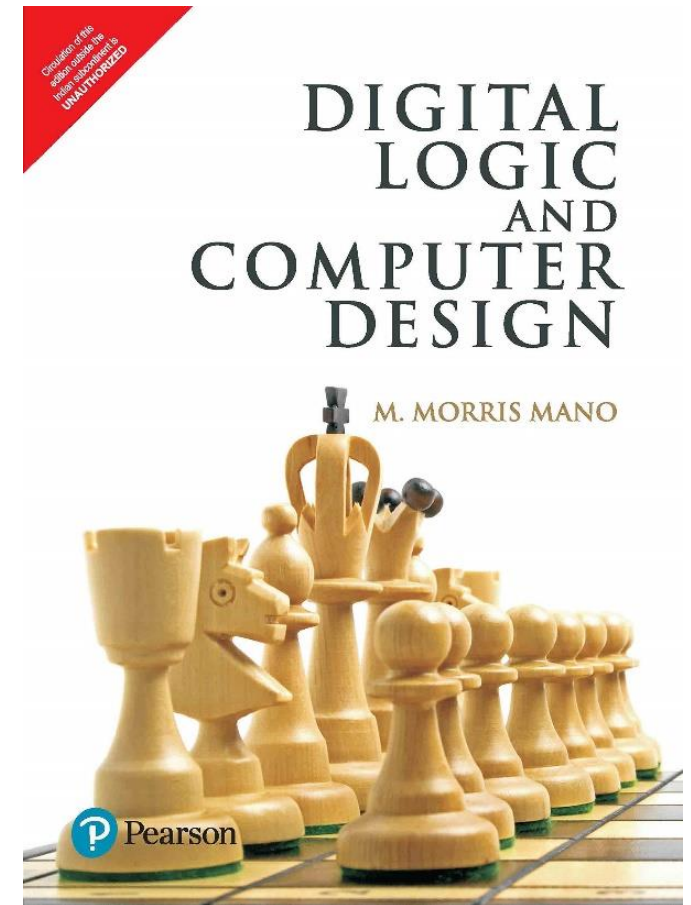
Kaynaklar

- “**Digital Design**”, M. Morris Mano
- “**Sayısal Tasarım**”, M. Morris Mano (Türkçe çevirisi)



Kaynaklar

- “Digital Logic and Computer Design”, M. Morris Mano



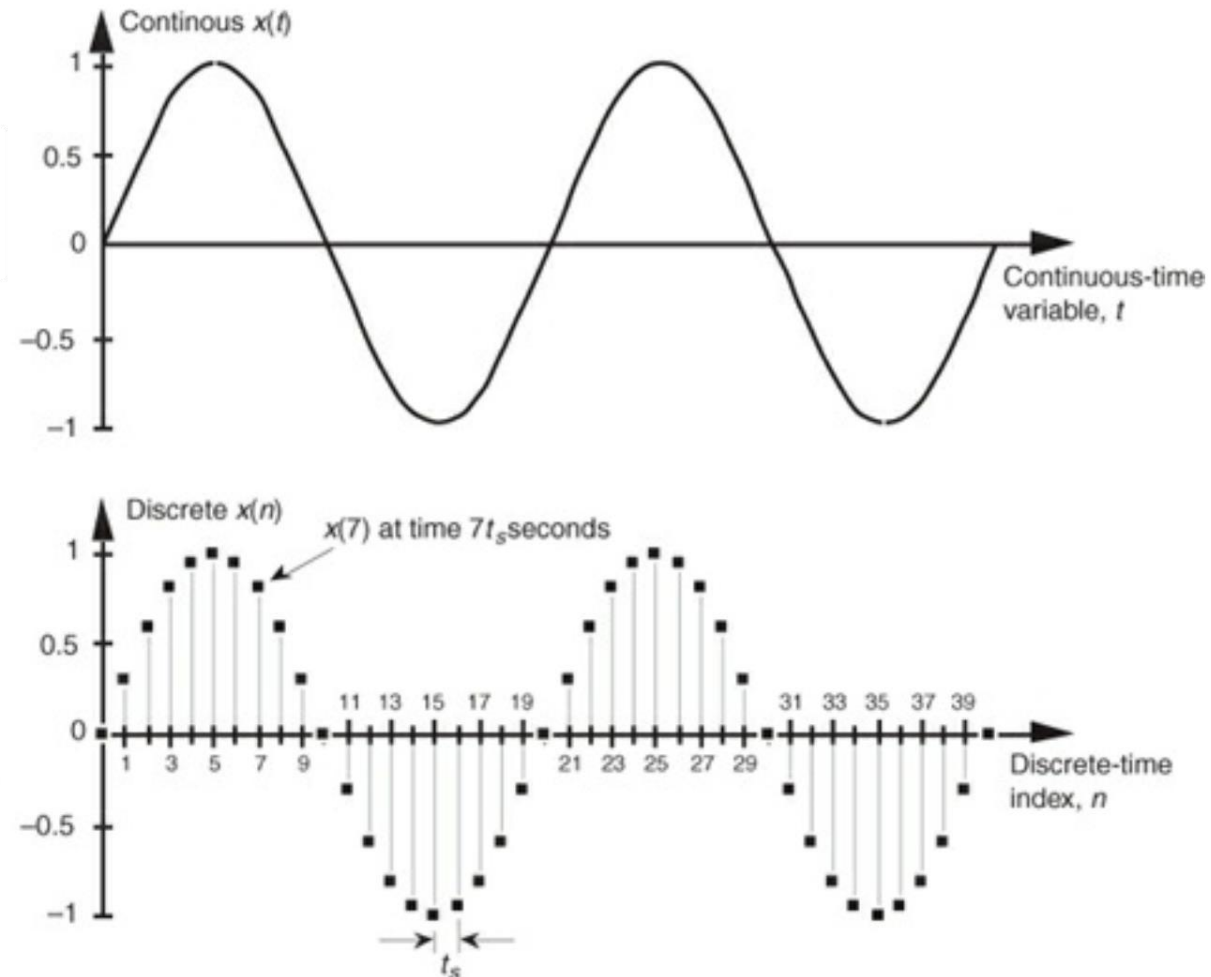
Sayısal Sistemler

- Dijital sistemlerin bir özelliği, **ayrık bilgi** öğelerini temsil etme ve kullanma yetenekleridir.



Sayısal Sistemler-devam

- Sınırlı sayıda elemanla sınırlı olan herhangi bir küme, ayırık bilgi içerir.
- Ayırık setlerin örnekleri 10 ondalık basamak, alfabenin 29 harfi, 52 oyun kartı ve bir satranç tahtasının 64 karesidir.
- Bilginin ayırık öğeleri, dijital bir sistemde sinyaller adı verilen fiziksel niceliklerle temsil edilir.
- Gerilim ve akım gibi elektrik sinyalleri en yaygın olanıdır.



1.2 İkili Sayılar

Sayısal Sistemler ve İkili Sayılar

- Genel olarak, **ondalık** noktalı bir sayı aşağıdaki gibi bir dizi katsayı ile temsil edilir:

$$a_5a_4a_3a_2a_1a_0 \cdot a_{-1}a_{-2}a_{-3} \quad \rightarrow$$

$$10^5a_5 + 10^4a_4 + 10^3a_3 + 10^2a_2 + 10^1a_1 + 10^0a_0 + 10^{-1}a_{-1} + 10^{-2}a_{-2} + 10^{-3}a_{-3}$$

- Birçok farklı sayı sistemi vardır. Genel olarak, bir **temel-r sayı sisteminde** ifade edilen bir sayının katsayıları r'nin katlarıyla şu şekilde çarpılır:

$$a_n \cdot r^n + a_{n-1} \cdot r^{n-1} + \dots + a_2 \cdot r^2 + a_1 \cdot r + a_0 + a_{-1} \cdot r^{-1} \\ + a_{-2} \cdot r^{-2} + \dots + a_{-m} \cdot r^{-m}$$

Sayısal Sistemleri → Örnek

$$(4021.2)_5 = 4 \times 5^3 + 0 \times 5^2 + 2 \times 5^1 + 1 \times 5^0 + 2 \times 5^{-1} = (511.4)_{10}$$

$$(127.4)_8 = 1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 4 \times 8^{-1} = (87.5)_{10}$$

$$(B65F)_{16} = 11 \times 16^3 + 6 \times 16^2 + 5 \times 16^1 + 15 \times 16^0 = (46,687)_{10}$$

$$(110101)_2 = 32 + 16 + 4 + 1 = (53)_{10}$$

1.3 Sayı-Taban Dönüşümleri

Sayı Tabanlarında Dönüşüm

$$\begin{array}{ccccccccc} (10 & 110 & 001 & 101 & 011 & \cdot & 111 & 100 & 000 & 110)_2 & = & (26153.7406)_8 \\ 2 & 6 & 1 & 5 & 3 & & 7 & 4 & 0 & 6 \end{array}$$

$$\begin{array}{ccccccc} (10 & 1100 & 0110 & 1011 & \cdot & 1111 & 0010)_2 & = & (2C6B.F2)_{16} \\ 2 & C & 6 & B & & F & 2 \end{array}$$

$$(673.124)_8 = \begin{array}{ccccccc} (110 & 111 & 011 & \cdot & 001 & 010 & 100)_2 \\ 6 & 7 & 3 & & 1 & 2 & 4 \end{array}$$

$$(306.D)_{16} = \begin{array}{ccccccc} (0011 & 0000 & 0110 & \cdot & 1101)_2 \\ 3 & 0 & 6 & & D \end{array}$$

1.5 Tmleyen iřlemi

Tümleme (Complement)

- Sayısal bilgisayarlar tüm aritmetik işlemleri toplama işlemine çevirerek gerçekleştirirler. Sayısal bilgisayarlardaki tümleyiciler genellikle çıkarma işlemini basitleştirmek ve mantıksal işlemler için kullanılır.
- Her r -tabanlı sayı sistemi için iki tür tümleme işleminden bahsedilir:
 - (r) 'ye tümleyen
 - $(r-1)$ 'e tümleyen

(r)'ye göre tümleme işlemi

- **n** haneli **r**-tabanlı **N** pozitif sayı için N sayısının **r'**ye tümleyeni:

$$r^n - N \text{ for } N \neq 0$$

$(.)_2$ 'ye göre tümleyen alırken;

- **Kural1:**

- Bütün en düşük değerlikli sıfırları ve sıfırdan farklı ilk haneyi olduğu gibi bırakıp diğer bütün bitlerdeki
 - **1'leri \rightarrow 0**
 - **0'ları \rightarrow 1**
- yaparak 2'ye tümleyen bulunabilir.

- **Kural2:**

- Önce 1'e göre tümleyen alınır daha sonra bu sonuca 1 eklenir.

(r-1)'ye göre tümleme işlemi

- **n** haneli bir tam sayı ve **m** haneli bir kesirli kısmı bulunan **r**-tabanlı **N** pozitif sayı için N sayısının (r-1)'e tümleyeni:

$$r^n - r^m - N.$$

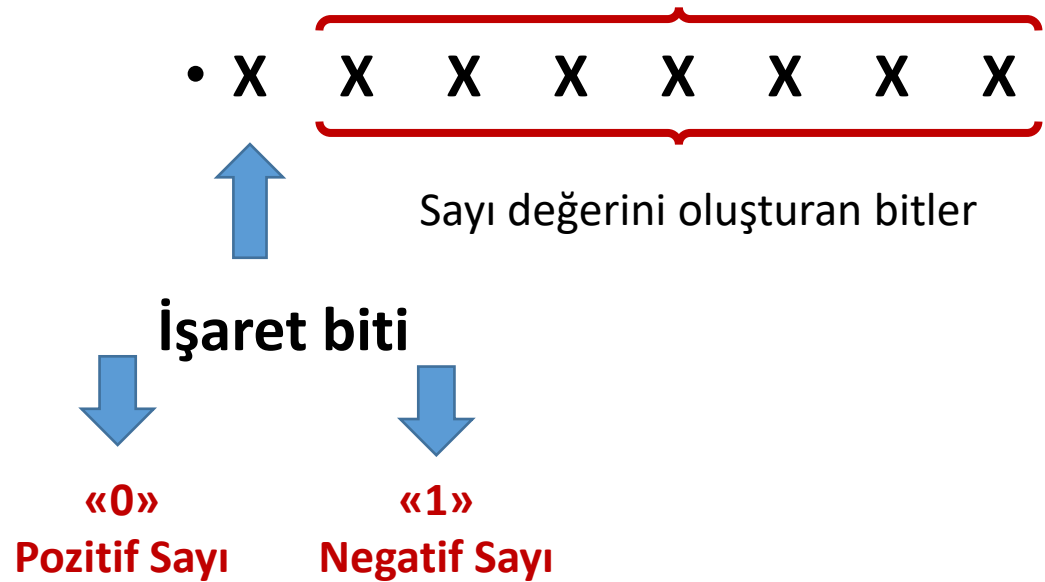
$(.)_1$ 'e göre tümleyen alırken;

- **Kural3:**
 - 1'e tümleyeni bulmak için tümleyeni alınacak olan ikili (binary) sayıdaki bütün
 - **1'ler \rightarrow 0**
 - **0'lar \rightarrow 1**
 - yapılır.

1.6 İşaret Bitli Sayılar

İşaret Bitli Sayılar

- Bir sayı için işaret bitli denilirse, bu sayıdaki en son MSB biti işaret biti olarak adlandırılır.
- İşaret bitli 8 bitlik bir sayıdan bahsedilmişse;



İşaret bitli sayılarda tümleyen

- **Kural:**

- İşaret bitli sayılarda tümleyen alınırken **işaret bitinin tümleyeni alınmaz.**

- **Örnek**

- $(10101100)_2$ işaret bitli sayının 2'ye tümleyeni nedir?



1.9 İkili Mantık

İkili Mantık (Binary Logic)

- İkili mantık, iki ayrı değer alan değişkenlerle ve mantıksal anlam üstlenen işlemlerle ilgilenir.
- Değişkenlerin aldığı iki değer farklı isimlerle çağrılabilir (örneğin, doğru ve yanlış, evet ve hayır, vb.).
- Amacımız için bit cinsinden düşünmek ve 1 ve 0 değerlerini atamak daha uygundur.
- İkili mantık, ikili bilginin işlenmesini matematiksel bir şekilde tanımlamak için kullanılır. Özellikle dijital sistemlerin analizi ve tasarımı için uygundur.
- Burada tanıtılacak ikili mantık, **Boole cebri** adı verilen bir cebire eşdeğerdir.
- Bu bölümün amacı, Boole cebirini sezgisel bir şekilde tanıtmak ve bunu dijital mantık devreleri ve ikili sinyallerle ilişkilendirmektir.

Lojik Kapılar

Lojik Kapılar – devam

- Mantık kapıları, bir çıkış sinyali üretmek için bir veya daha fazla giriş sinyali üzerinde çalışan elektronik devrelerdir.
- Gerilim veya akım gibi elektrik sinyalleri, 0 ila 3 V gibi belirli bir sürekli aralıkta değerlere sahip analog sinyaller olarak bulunur, ancak dijital bir sistemde bu voltajlar, iki tanınabilir değerden biri olarak yorumlanır, 0 veya 1.
- Mantık devreleri, lojik 1 veya lojik 0'a eşit bir ikili değişkeni temsil eden iki ayrı voltaj düzeyine yanıt verir.
- Örneğin, belirli bir sayısal sistem, lojik 0'ı 0 V'ye eşit bir sinyal ve lojik 1'i 3 V'a eşit bir sinyal olarak tanımlayabilir.

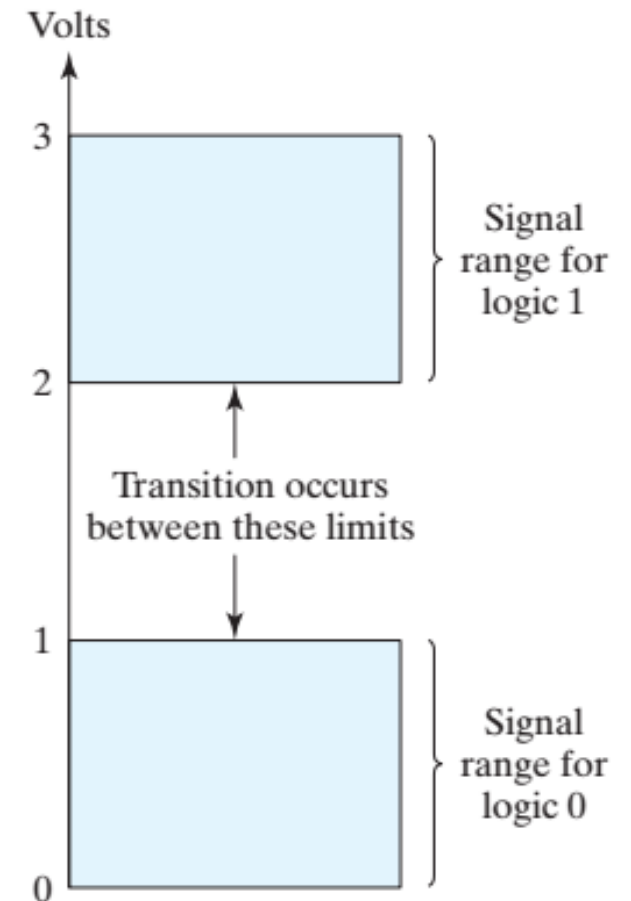










FIGURE 1.3
Signal levels for binary logic values



Özet

Name	Graphic symbol	Algebraic function	Truth table															
AND		$F = x \cdot y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = x + y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	1
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
Inverter		$F = x'$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	F	0	1	1	0									
x	F																	
0	1																	
1	0																	
Buffer		$F = x$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	x	F	0	0	1	1									
x	F																	
0	0																	
1	1																	

NAND		$F = (xy)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = (x + y)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	0
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

Nor ve Nand Kapıları ile temel mantık kapıları tasarımı

- Nor ile Nand kapıları kullanılarak Değil, Ve, Veya, Nor ve Nand kapıları elde etmek mümkündür.
- Her iki kapının doğruluk tabloları incelendiğinde girişler aynı iken çıkış girişlerin tersi şeklindedir.

NAND		$F = (xy)'$	x	y	F
			0	0	1
			0	1	1
			1	0	1
			1	1	0
<hr/>					
NOR		$F = (x + y)'$	x	y	F
			0	0	1
			0	1	0
			1	0	0
			1	1	0

Temel teoremler ve Boolean cebrinin özellikleri

Table 2.1

Postulates and Theorems of Boolean Algebra

Postulate 2	(a)	$x + 0 = x$	(b)	$x \cdot 1 = x$
Postulate 5	(a)	$x + x' = 1$	(b)	$x \cdot x' = 0$
Theorem 1	(a)	$x + x = x$	(b)	$x \cdot x = x$
Theorem 2	(a)	$x + 1 = 1$	(b)	$x \cdot 0 = 0$
Theorem 3, involution		$(x')' = x$		
Postulate 3, commutative	(a)	$x + y = y + x$	(b)	$xy = yx$
Theorem 4, associative	(a)	$x + (y + z) = (x + y) + z$	(b)	$x(yz) = (xy)z$
Postulate 4, distributive	(a)	$x(y + z) = xy + xz$	(b)	$x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$	(b)	$(xy)' = x' + y'$
Theorem 6, absorption	(a)	$x + xy = x$	(b)	$x(x + y) = x$

Operator Precedence

The operator precedence for evaluating Boolean expressions is (1) parentheses, (2) NOT, (3) AND, and (4) OR. In other words, expressions inside parentheses must be evaluated before all other operations. The next operation that holds precedence is the complement, and then follows the AND and, finally, the OR. As an example, consider the truth table for one of DeMorgan's theorems. The left side of the expression is $(x + y)'$. Therefore, the expression inside the parentheses is evaluated first and the result then complemented. The right side of the expression is $x'y'$, so the complement of x and the complement of y are both evaluated first and the result is then ANDed. Note that in ordinary arithmetic, the same precedence holds (except for the complement) when multiplication and addition are replaced by AND and OR, respectively.

Lojik Devre Tasarımına Giriş

- Bu bölümde geri besleme içermeyen ve kombinasyonel lojik olarak adlandırılan lojik devrelerin tasarımı anlatılacaktır. Lojik devre tasarımı aşağıdaki 4 adım gerçekleştirilerek yapılır.
 1. Lojik devrenin yapacağı tüm işlemler detaylarıyla belirlenir.
 2. İşlem doğruluk tablosu hazırlanır.
 3. İşlemin lojik ifadesi yani matematiksel modeli yazılır.
 4. Lojik devre basitleştirilebiliyorsa sadeleştirilir.
- Lojik ifadeler iki formda belirtilir.
 - 1. Form: Temel işlem toplama, ara işlemler çarpma
 - 2. Form: Temel işlem çarpma, ara işlemler toplama

Kanonik ve Standart Formlar

Minterm & Maxterm

Bir fonksiyonun çarpımların toplamı veya toplamların çarpımı şeklinde gösterilmesine uygun olarak minterm veya maxterm'ler ifade edilir.

İkili bir değişken, normal formunda (x) veya tümleyen formunda (x') yazılabilmektedir.

Minterm ve Maxterm'ler birbirinin değili/tümleyenidir.

Minterm & Maxterm

- **Minterm:**
- Değişkenlerin «1» değeri alanları kendisiyle, «0» değeri alanları değişkenin değili/tümleyeni ile sembolize edilerek **çarpım** formunda yazılırsa ve genel ifade **çarpımların toplamı** ile verilirse bu gösterime ‘**Minterm**’ adı verilir. Buna 1. Kanonik Açılım da denir. Minterm’ler küçük harfle sembolize edilir ($m_0, m_1, \dots, m_7, \dots$).

$$f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$$

Minterm & Maxterm

- **Maxterm:**
- Değişkenlerin «0» değeri alanları kendisiyle, «1» değeri alanları değişkenin değili/tümleyeni ile sembolize edilerek **toplama** formunda yazılırsa ve genel ifade **toplamların çarpımı** ile verilirse bu gösterime ‘**Maxterm**’ adı verilir. Buna 2. Kanonik Açılım da denir. Maxterm’ler büyük harfle sembolize edilir ($M_0, M_1, \dots, M_7, \dots$).

$$\begin{aligned} f_2 &= (x + y + z)(x + y + z')(x + y' + z)(x' + y + z) \\ &= M_0 M_1 M_2 M_4 \end{aligned}$$

Minterm & Maxterm

Table 2.3
Minterms and Maxterms for Three Binary Variables

x	y	z	Minterms		Maxterms	
			Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7