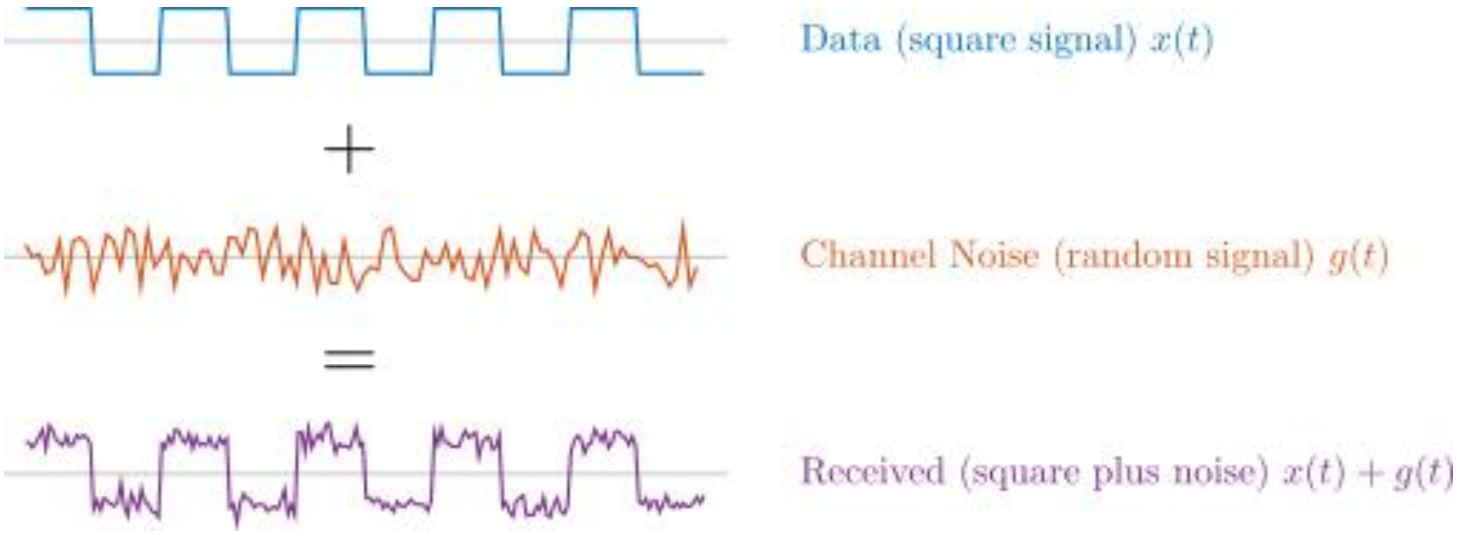
Konu 6
Data Transmission & Integrity

Text: Communication Systems Principles using MATLAB®

Text, drawings and images copyright © John Wiley & Sons 2018

Alıcıda gürültü vardır

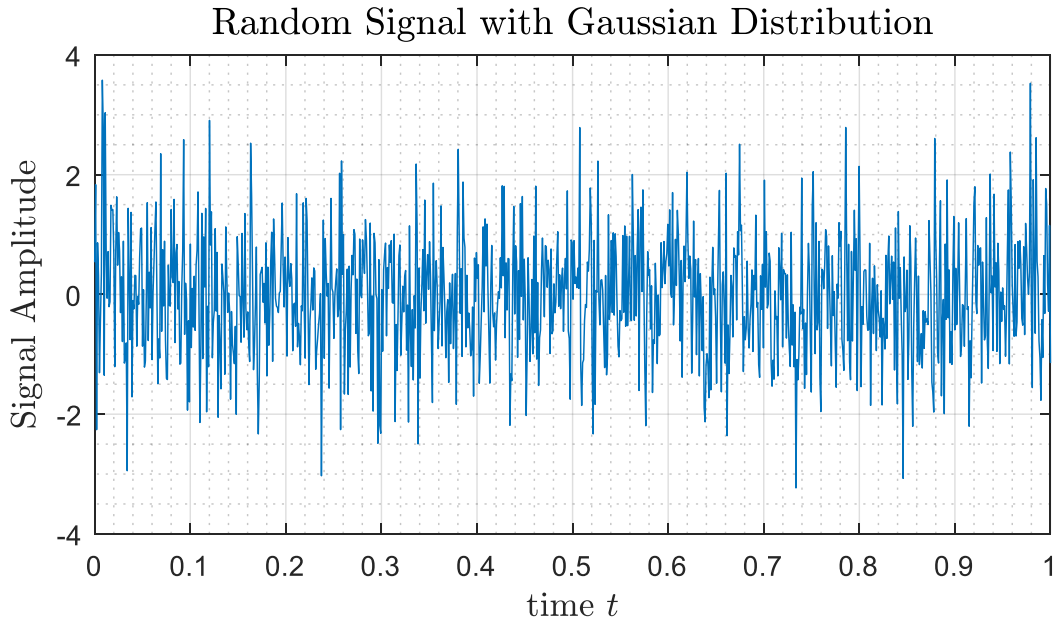
- Rastgele (tahmin edilemeyen veya edilebilir) gürültü.
- Alıcının hata oranını etkiler
- Sayısal iletişimde
  - Alıcı hata yapmazsa bu gürültüyü tamamen temizlemiş olur
  - Hata yapılsa bile bu bit hataları tespit edilebilir
  - Hatta bu bit hataları düzeltilebilir.



© John Wiley & Sons 2018

Gürültü

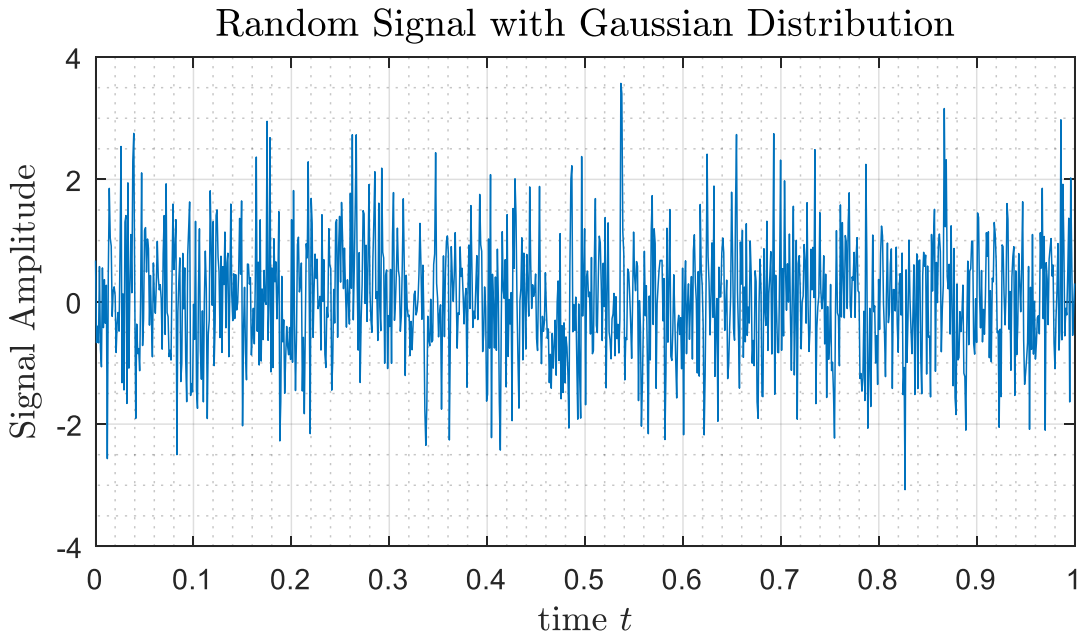
- Gürültü sinyal örnekleri Gauss dağılımlıdır.
  - Sıfır ortalamalı $E[n(t)] = 0, \forall t$
- $E[n(t_1)n(t_2)] = E[n(t_1)]E[n(t_2)] = 0, \forall t_1 \neq t_2$
  - Her hangi iki andaki gürültü birbirinden bağımsızdır
  - $R(t_1 - t_2) = E[n(t_1)n(t_2)] = \delta(t_1 - t_2) = \delta(\tau)$ →White noise

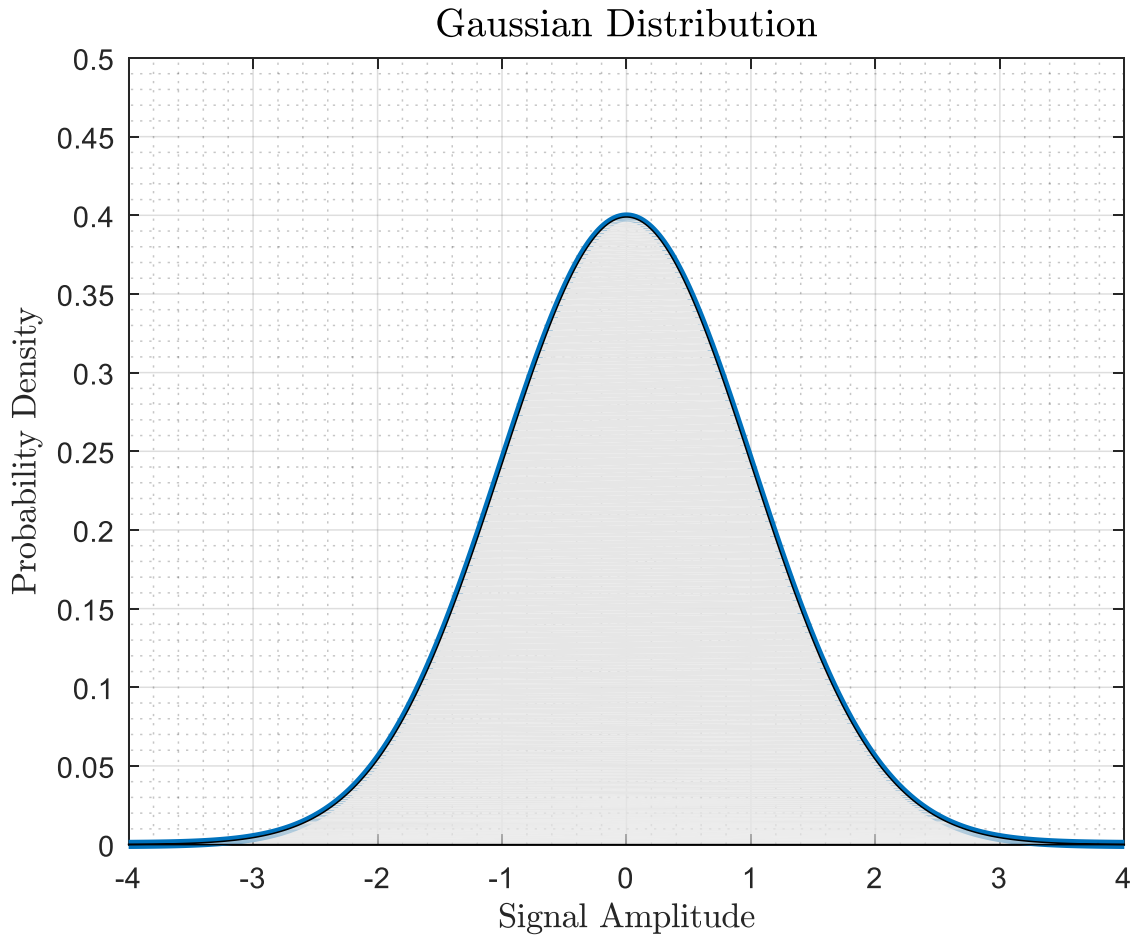


© John Wiley & Sons 2018

Olasılık yoğunluk fonksiyonu

- Gauss dağılımı





© John Wiley & Sons 2018

Olasılık yoğunluk fonksiyonu

- pdf

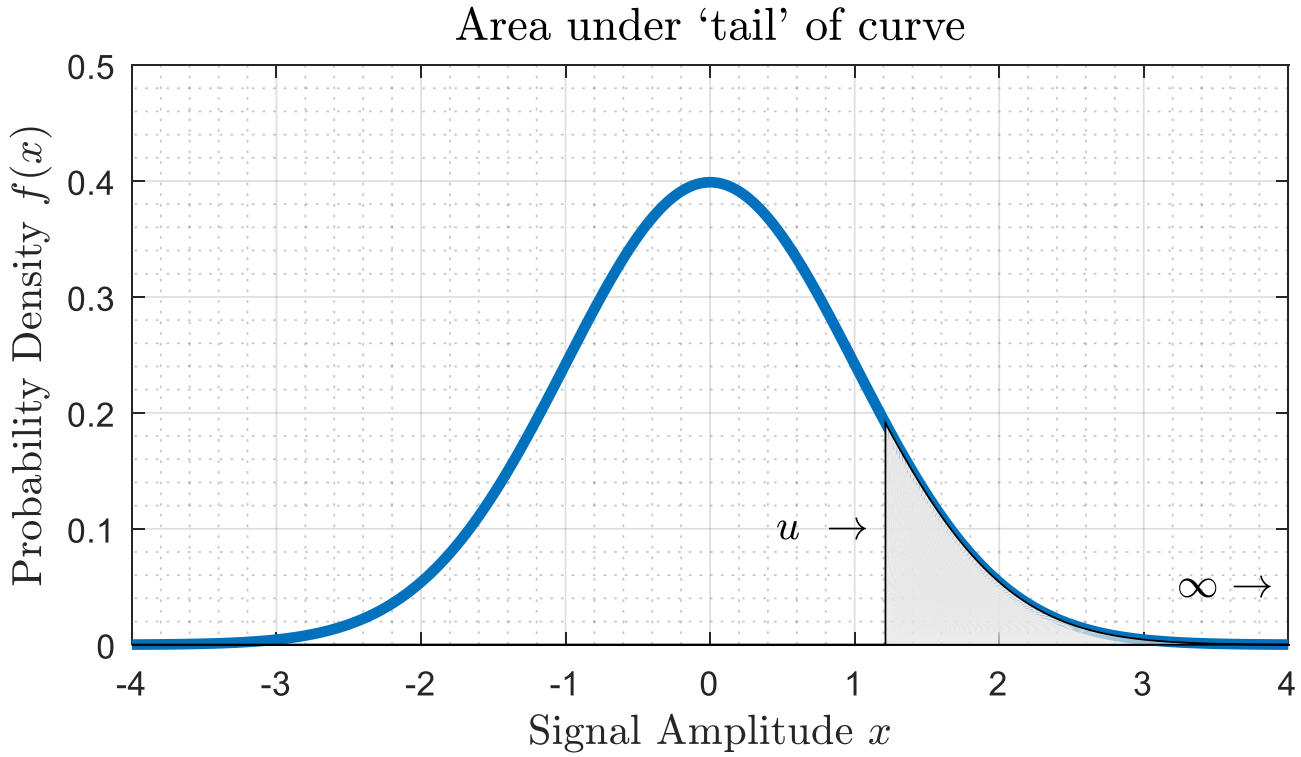
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

  - Ortalama: $E[X] = \mu$.
  - Varyans : $Var[X] = \sigma^2$
  - Matlab
    - Rand
    - Sigma*rand+mu

© John Wiley & Sons 2018

Noise Signal

- Sayısal iletişimde hata, gürültünün belli bir değerin üzerinde olma olasılığı ile yakından ilişkilidir.
- GaussArea.m



© John Wiley & Sons 2018

PDF Areas

- Complimentary error function: erfc()

$$\mathrm{erfc}(u) = \frac{2}{\sqrt{\pi}} \int_{u}^{\infty} e^{-x^2}\ dx$$

- Q fonksiyonu (Matlab qfunc(), qfuncinv())

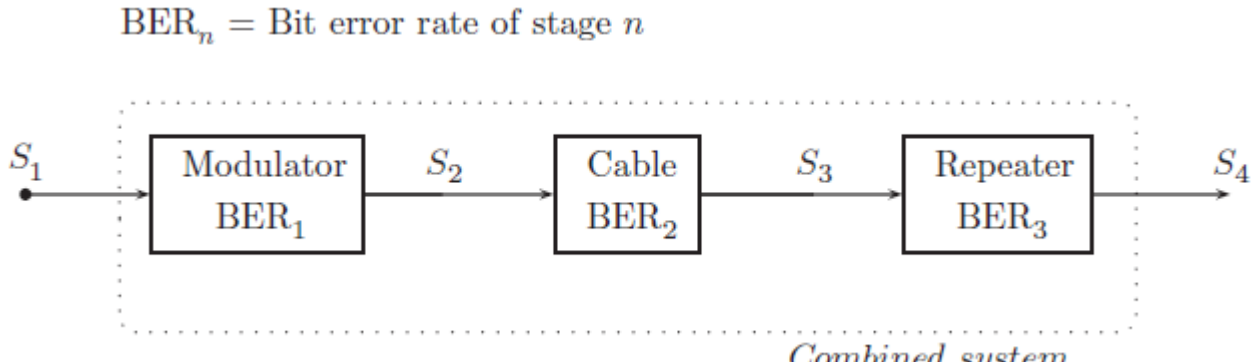
$$\mathrm{Q}(u) = P(X > u) = \frac{1}{\sqrt{2\pi}} \int_{u}^{\infty} e^{-\frac{x^2}{2}}\ dx$$

- Aradaki ilişki
- Standart olmayan Gauss dağılımlı ise?

© John Wiley & Sons 2018

6.4 Sayısal Sistemlerde Bit Hataları

- Hata kabul etmeyen uygulamalar
  - E-posta, dosya iletimi
- Hataya toleranslı uygulamalar
  - Telekonferans, videokonferans, radyo, TV
- Bit hata oranı (BER)
  - 10^{-6}
  - 100 MB'lık dosyadaki ortalama hatalı bit sayısı?
- Kaskat sistemler (örnek?)
- Birleşik sistemin bit hata oranı nasıl hesaplanır?



6.4 Sayısal Sistemlerde Bit Hataları

- Bit hataları nasıl olur?
  - Gürültü
  - Sönümlenme
  - Bozulma
- Önceki konuda gördüğümüz kipçözücü çıktısında bir nokta elde ediyorduk
  - Bu bozulmalar nedeniyle nokta yanlış bölgeye düşer ve yanlış karar verilir.
- Gürültü olasılık yoğunluğu hatayı nasıl etkiler?
- Basit bir iki örnek (BPSK)

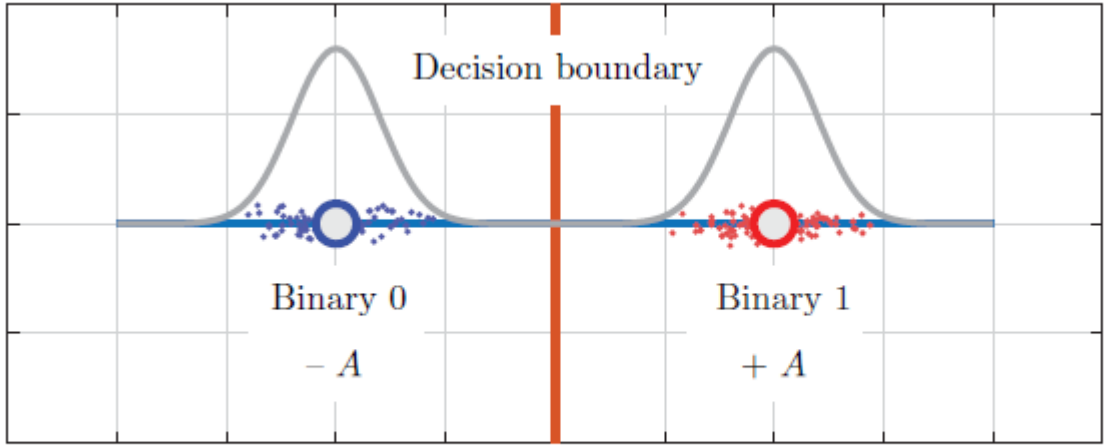


Figure 6.6 Two polar values $+A$ and $-A$, with additive noise. The probability density shows the likelihood for each level at the receiver.

© John Wiley & Sons 2018

6.4.2 Bit hatalarının analizi

- QPSK örneği

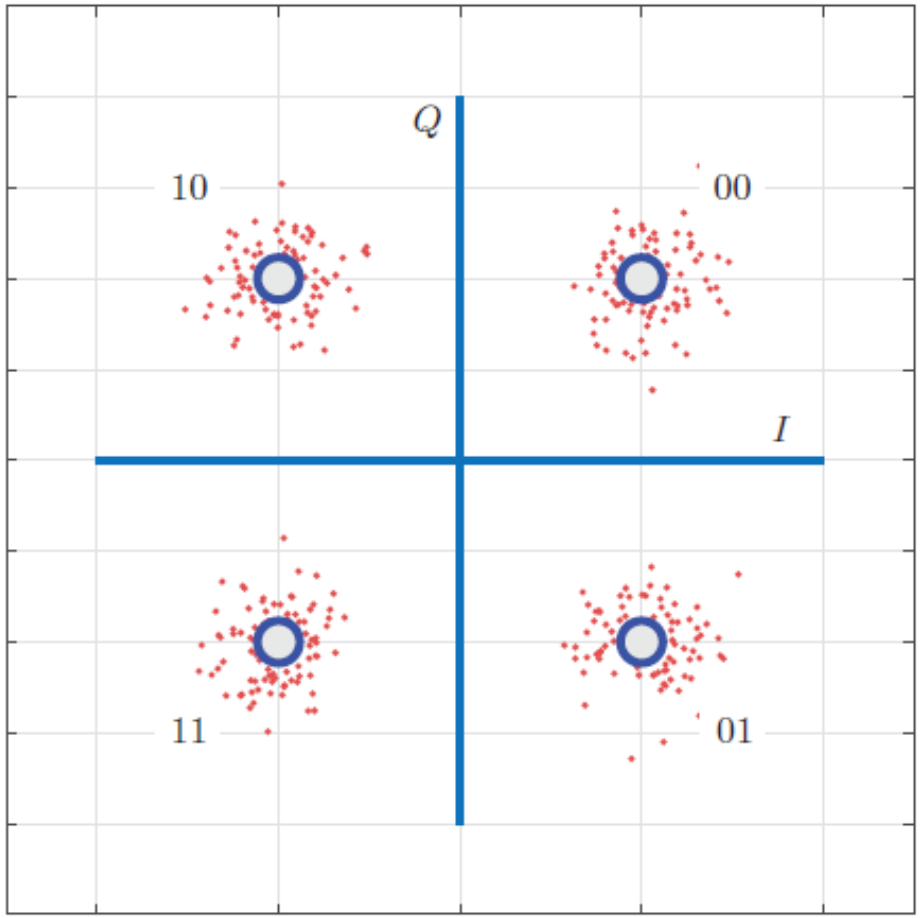
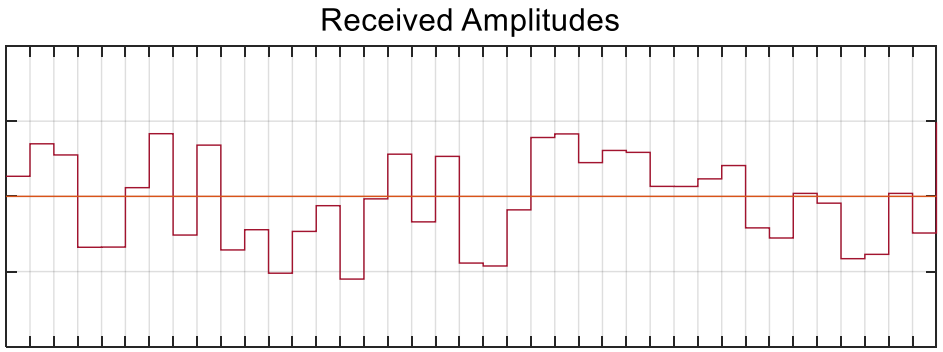


Figure 6.7 Extending the concept of received points to two orthogonal axes. Two bits are transmitted at a time, with the decision boundary being the axes themselves.

© John Wiley & Sons 2018

6.4.2 Bit hatalarının analizi

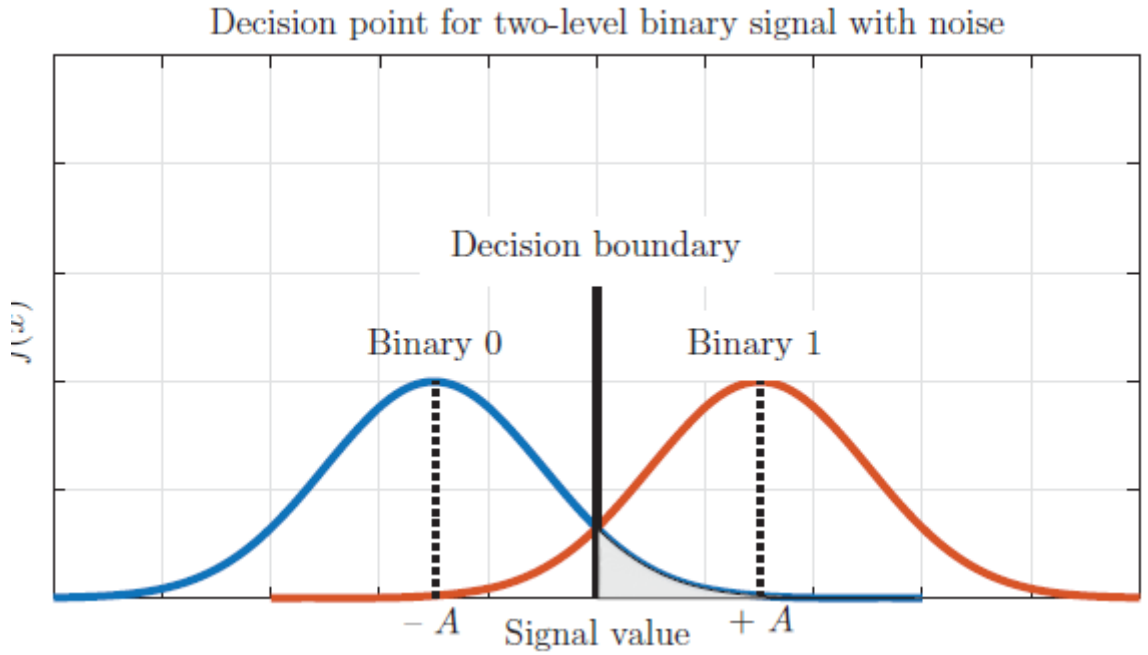
- Sinyal $+A$ üzerine gürültü eklendiğinde eşik değerinin altına düşerek $-A$ kararına (yanlış karara) yol açabilir.
  - Tam tersi de olabilir.



© John Wiley & Sons 2018

6.4.2 Bit hatalarının analizi

- $r(t) = \pm 1 s(t) + n(t)$
- Alıcıda $s(t)$ ile çarpıp integralini al.$r = \pm A + n$
- $P(hata) = P(r < 0|A)P(A) + P(r > 0|-A)P(-A)$
- Taralı alan

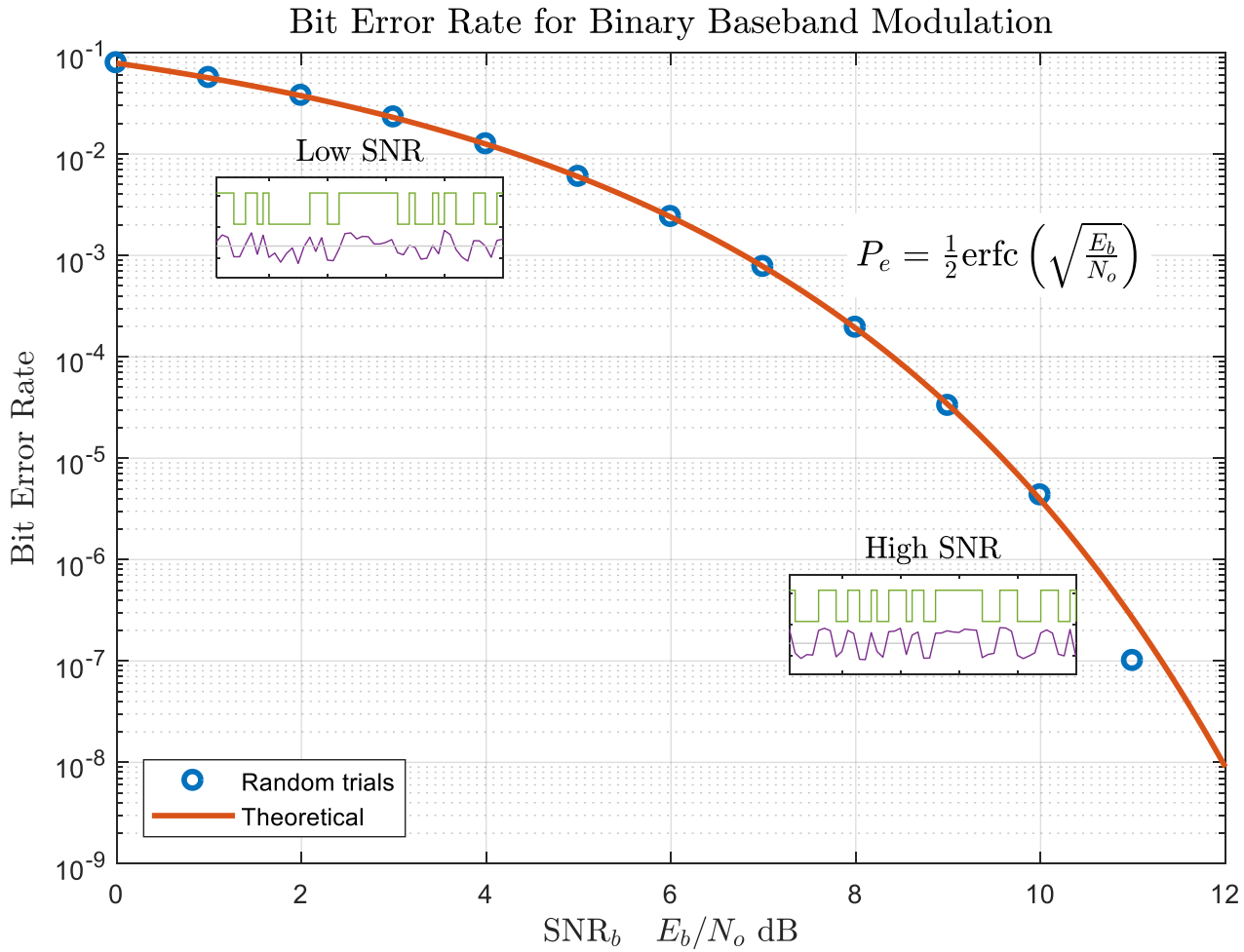


© John Wiley & Sons 2018

6.4.2 Bit hatalarının analizi

- $BER = P(n < -A)\frac{1}{2} + P(n > A)\ \frac{1}{2}$
- Simetriden dolayı
- $BER = P(n > A)$
- $n \sim \mathcal{N}(0, \sigma^2)$
- $BER =$

- $BER = Q\left(\sqrt{SNR}\right)$
- SNR cinsinden yazalım
- BerPlots.m



© John Wiley & Sons 2018

6.5 Hata Tespiti

- Alıcı, gönderilen bit dizisinde hata olup olmadığını tespit edebilir
- Hatta, bu hatayı düzeltebilir
- Bunun için gönderilen bit dizisine fazladan bilgi (redundancy) eklenmesi gerekir.
  - Bu iş sistematik olarak yapılmalıdır.
- Bu işlemlerde XOR (Exclusive-OR) fonksiyonu sıkça karşımıza çıkar
- $A \oplus B = A.\bar{B} + B.\bar{A}$
- İki bitin farklı olup olmadığını tespit eder

Input A	Input B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

© John Wiley & Sons 2018

6.5 Hata Tespiti

- Eşlik biti (parity bit)
- Bir bit dizisinin sonuna eklenir
  - Ör. 1'ler çift sayıda olacak şekilde
- Ardışık XOR işlemleriyle üretilebilir
- Bir bit hatalı giderse alıcıda ne olur?
- İki bit hatalı giderse??
  - Bit hataları genelde art arda gelir
- İki boyutlu parity işlemi

Table 6.2 Examples of computation of an even parity bit.

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	b_{parity}
1	0	1	1	1	0	1	1	0
0	0	1	1	0	0	1	0	1
1	1	0	1	1	0	0	1	1
1	0	1	1	1	1	0	1	0

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

© John Wiley & Sons 2018

6.5.1 Hata Düzeltme

- İlk akla gelen yöntem: Tekrar etmek (repetition)
- Çok fazla ek yük getiriyor.
- Repetition + Majority Rule
- Aşağıdaki örnek
  - Mesaj kelimeleri
  - Bunlara karşılık gelen kod kelimeleri

Input Bits		Coded Output Bits								
0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	1	1
1	0	0	0	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

© John Wiley & Sons 2018

Tekrar Kodu: Repetition Code

- Suppose one bit changes, can we **detect** the error?
- *Yes.. Because 000 000 010 is not a valid codeword.*
- Can we **correct** the error?
- *Yes…if receiver assumes the 1 should be 0 and codeword sent was 000 000 000.*

Input Bits		Coded Output Bits								
0	0	0	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	1	1
1	0	0	0	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

© John Wiley & Sons 2018

Repetition Code

- Suppose two bits change, can we **detect** the error?
- *Yes.. Because 000 000 011 is not a valid codeword.*
- Can we **correct** the error?
- *Yes…if receiver assumes the valid codeword sent was 000 000 111. But this is wrong!*
- *2 bit hatayı düzeltemedik*

Input Bits		Coded Output Bits								
0	0	0	0	0	0	0	0	0	1	1
0	1	0	0	0	0	0	0	1	1	1
1	0	0	0	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

© John Wiley & Sons 2018

Hamming Uzaklığı

- İki kod kelimesinin farklı olduğu bit sayısı:
- Bütün kod kitabındaki en ufak uzaklık: Hamming Uzaklığı d_{min}.
  - Kodun performansını en yakın iki kod kelimesi belirler
    - Zincirdeki en zayıf halka
- Önceki örnekte, $d_{min} = \cdots$
- d bit hata sezebilmek için $d_{min} = d + 1$
- d bit hata düzeltebilmek için $d_{min} = 2d + 1$

© John Wiley & Sons 2018

6.5.1 Hamming Kodları

- Hata DÜZELTEBİLEN kodlar
- Her bit için bir kontrol değeri sağlarlar
  - Böylece bütün bir bitlik hataları düzeltebilirler.
- Örnek: Hamming (N, M) kodu:
  - N bit kod kelimesi, M bit mesaj kelimesi.
- Önceki tekrarlama kodu: $(9,2)$'dir. Çünkü: 2 bit mesaja 9 bit kod kelimesi karşılık gelir. Yani $9 - 2 = 7$ kontrol biti var. Çok verimsiz.
- En bilinen örnek: Hamming $(7,4)$ kodu. M=4 mesaj kelimesine karşılık 7 bit kod kelimesi (yani $\mathrm{C} = 3$ artık (kontrol) biti).
- Kontrol bitlerine eşlik biti de denir

© John Wiley & Sons 2018

6.5.1 Hamming Kodları

- $(N, M) = (9,2)$ kod
  - $2^M = 2^2 = 4$ mesaj
  - $2^N = 2^9 = 512$ adet kod kelimesi adayı
  - Bu adaylardan 4 tanesi kullanılıyor, diğerleri ($2^N - 2^M = 508$ adet) geçersiz oluyor
- Her biri N bit olan 2^M kod kelimesinde,
  - Bir bitlik hatalarla oluşabilecek kod kelimesi sayısı $N \times 2^M$
  - Bir bitlik hataların düzeltilebilmesi için $(N + 1) \times 2^M \leq 2^N$ olmalıdır.
  - $C = N - M$ artık bit sayısı
  - $2^C - 1 \geq M + C$

M	C	$2^C - 1$	$N = M + C$
4	1	1	5
4	2	3	6
4	3	7	7
4	4	15	8
7	3	7	10
7	4	15	11
8	1	1	9
8	2	3	10
8	3	7	11
8	4	15	12
8	5	31	13
16	4	15	20
16	5	31	21
32	6	63	38

© John Wiley & Sons 2018

6.5.1 Hamming Kodlarının Üretimi

- H(7,4) kodu
- Bit pozisyonları 1(LSB) – 7(MSB)
- $(m_3, m_2, m_1, c_2, m_0, c_1, c_0)$
  - Artık bitler 2'nin kuvvetlerinde bulunur
- Her artık bit in altında bir adet 1 var.
- O 1'e karşılık gelen mesaj bitleri XOR'lanır
- Alıcıda

	7	6	5	4	3	2	1
	m_3	m_2	m_1	c_2	m_0	c_1	c_0
MSB	1	1	1	1	0	0	0
	1	1	0	0	1	1	0
LSB	1	0	1	0	1	0	1

$$c_0 = m_3 \oplus m_1 \oplus m_0$$

$$c_1 = m_3 \oplus m_2 \oplus m_0$$

$$c_2 = m_3 \oplus m_2 \oplus m_1$$

$$\begin{aligned} s_0 &= c_0 \oplus c_0 \\ &= c_0 \oplus m_3 \oplus m_1 \oplus m_0 \end{aligned}$$

- Sendromun onluk düzendeki değeri hatalı bit pozisyonunu verir
- Ör. Mesaj bitleri 0101 olsun…

© John Wiley & Sons 2018

6.5.2 Checksum (Sağlama toplamı)

- Bit hatası olup olmadığını anlamaya yarar
- Veri kendi içinde toplanır
- Gerçeklenmesi kolaydır
- Veri 8 veya 16 bitlik parçalara ayrılır ve toplanır
  - Alıcı ve vericide aynı sonucu vermeli
- A few extensions to make it easier (faster) to calculate.
- TCP/IP protokollerinde kullanılır
  - IP Header (paket başlığı, checksum içindedir)
- Hataları sezer ama düzeltemez.
- Geçersiz sağlama olan paketleri TCP yeniden iletir.

© John Wiley & Sons 2018

Internet checksum

From Wikipedia, the free encyclopedia

The **Internet checksum**,[1][2] also called the **IPv4 header checksum** is a checksum used in version 4 of the Internet Protocol (IPv4) to detect corruption in the header of IPv4 packets. It is carried in the IP packet header, and represents the 16-bit result of summation of the header words.[3]

The IPv6 protocol does not use header checksums. Its designers considered that the whole-packet link layer checksumming provided in protocols, such as PPP and Ethernet, combined with the use of checksums in upper layer protocols such as TCP and UDP, are sufficient.[4] Thus, IPv6 routers are relieved of the task of recomputing the checksum whenever the packet changes, for instance by the lowering of the *Hop limit counter* on every hop.

The Internet checksum is mandatory to detect errors in IPV6 UDP packets (including data payload).

The Internet checksum is used to detect errors in ICMP packets (including data payload).

© John Wiley & Sons 2018

Computation

- The checksum calculation is defined in RFC 791:[5]

- The checksum field is the 16-bit ones' complement of the ones' complement sum of all 16-bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero.

- If there is no corruption, the result of summing the entire IP header, including checksum, should be zero.

- At each hop, the checksum is verified. Packets with checksum mismatch are discarded. The router must adjust the checksum if it changes the IP header (such as when decrementing the TTL).[6]

- The procedure is explained in detail in RFC 1071 "Computing the Internet Checksum". [1]
- Optimisations are presented in RFC 1624 "Computation of the Internet Checksum via Incremental Update" [2] (with errata), to cover the case in routers which need to recompute the header checksum during packet forwarding when only a single field has changed.

1. *Computing the Internet Checksum*. *doi*:*10.17487/RFC1071*. *RFC* *1071*.
2.^ Jump up to:[a] [b] *Computation of the Internet Checksum via Incremental Update*. *doi*:*10.17487/RFC1624*. *RFC* *1624*.

© John Wiley & Sons 2018

- **Examples**

- **Calculating the IPv4 header checksum**

- Take the following truncated excerpt of an IPv4 packet. The header is shown in bold and the checksum is underlined.
- **4500 0073 0000 4000 4011 b861 c0a8 0001**
  c0a8 00c7 0035 e97c 005f 279f 1e4b 8180

- For ones' complement addition, each time a carry occurs, we must add a 1 to the sum.[7]
- A carry check and correction can be performed with each addition or as a post-process after all additions.
- If another carry is generated by the correction, another 1 is added to the sum.
- To calculate the checksum, we can first calculate the sum of each 16 bit value within the header, skipping only the checksum field itself.
- Note that these values are in hexadecimal notation.

- `4500 + 0073 + 0000 + 4000 + 4011 + c0a8 + 0001 + c0a8 + 00c7 = 2479c`

- The first digit is the carry count and is added to the sum:
- `2 + 479c = 479e` (if another carry is generated by this addition, another 1 must be added to the sum)

- To obtain the checksum we take the ones' complement of this result: `b861` (as shown underlined in the original IP packet header).

- **Verifying the IPv4 header checksum**[edit]

- When verifying a checksum, the same procedure is used as above, except that the original header checksum is not omitted.
  `4500 + 0073 + 0000 + 4000 + 4011 + b861 + c0a8 + 0001 + c0a8 + 00c7 = 2fffd`

- Add the carry bits:
- `fffd + 2 = ffff`
- Taking the ones' complement (flipping every bit) yields 0000, which indicates that no error is detected. IP header checksum does not check for the correct order of 16 bit values within the header.

© John Wiley & Sons 2018

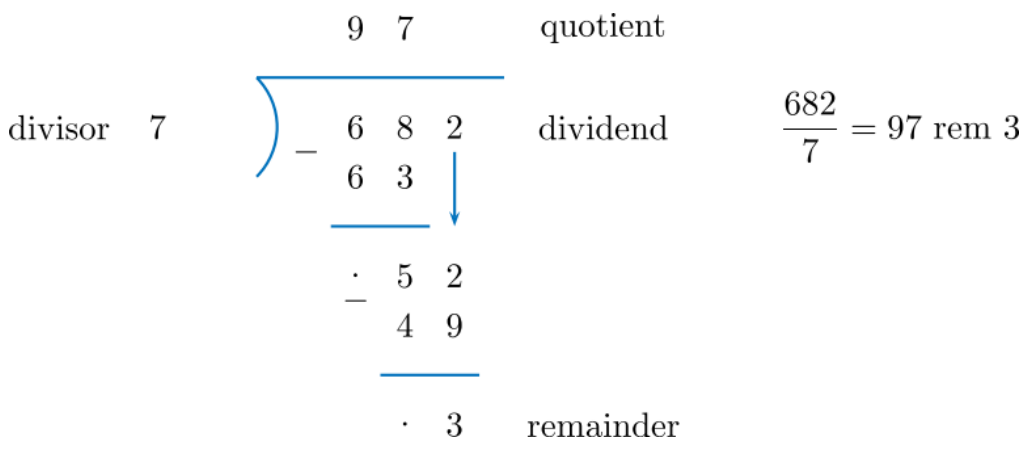
6.5.3 Cyclic Redundancy Checks (CRC)

- Checksum: Yazılımda daha kolay, CRC: donanımda daha kolay
  - Ör. Ethernet
- Sıralı bitler üzerinde işlem yaparak bir değer üretilir
  - Shift register, XOR kapıları
- Mesaj bit dizisi önceden belirlenmiş bir bit dizisine BÖLÜNÜR
  - Kalan değer bit dizisinin sonuna eklenir
- Gerçek bir bölüme işlemi değil, aslında XOR işlemi
- Hataları sezer ama yine düzeltemez
- 1000'lerce bitlik mesajda sadece 16-32 bit eklenerek, olabilecek hataların büyük kısmı tespit edilebilir

© John Wiley & Sons 2018

6.5.3 Cyclic Redundancy Checks

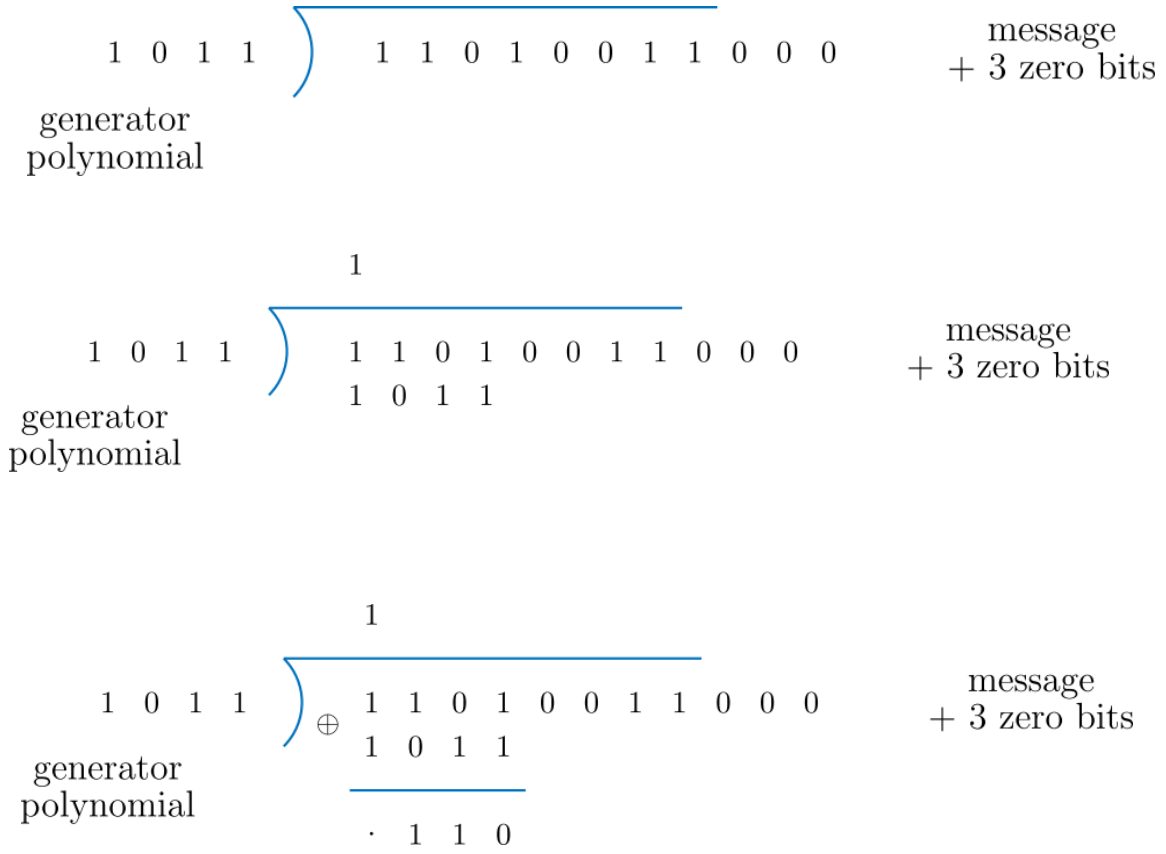
- 10luk tabanda bölme işlemi örneği
- Procedure is to add redundant bits to message, successively multiply by 1 or 0 and XOR.
- Decimal example shown to fix ideas, but not actually done this way.
- 682 is the “message”, the remainder of 3 is the error check.
- If message corrupted, receiver will obtain a different remainder.



© John Wiley & Sons 2018

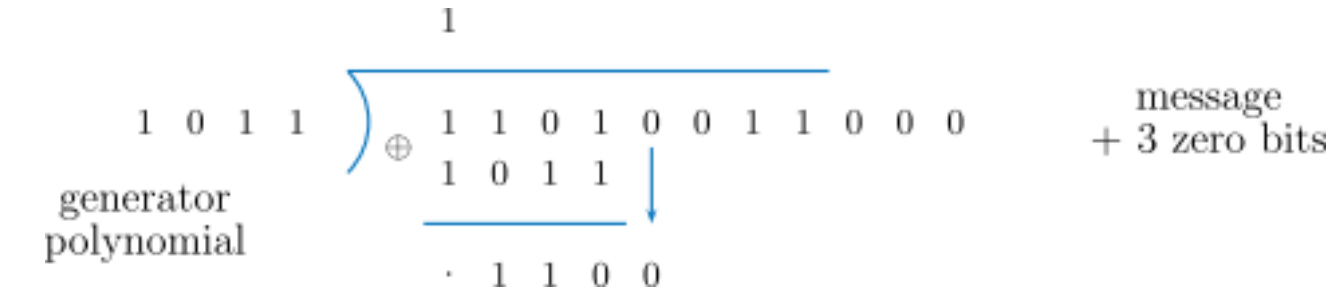
6.5.3 Cyclic Redundancy Checks

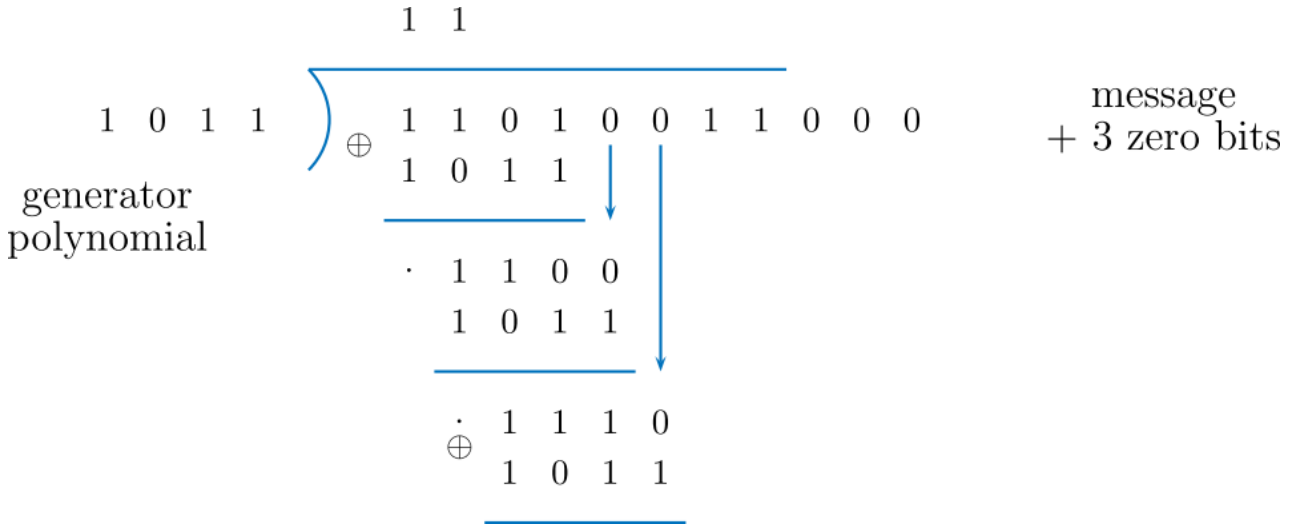
- İkilik tabanda bölme
- Üreteç polinomu 4 basamaklı
- Mesaj dizisinin sonuna 3 bit sıfır eklenir
- Her adımda:
  1. 1 veya 0 ile çarp
  2. XOR yap
  3. Yeni bir bit aşağı indir
  4. Sonuna kadar devam et



© John Wiley & Sons 2018

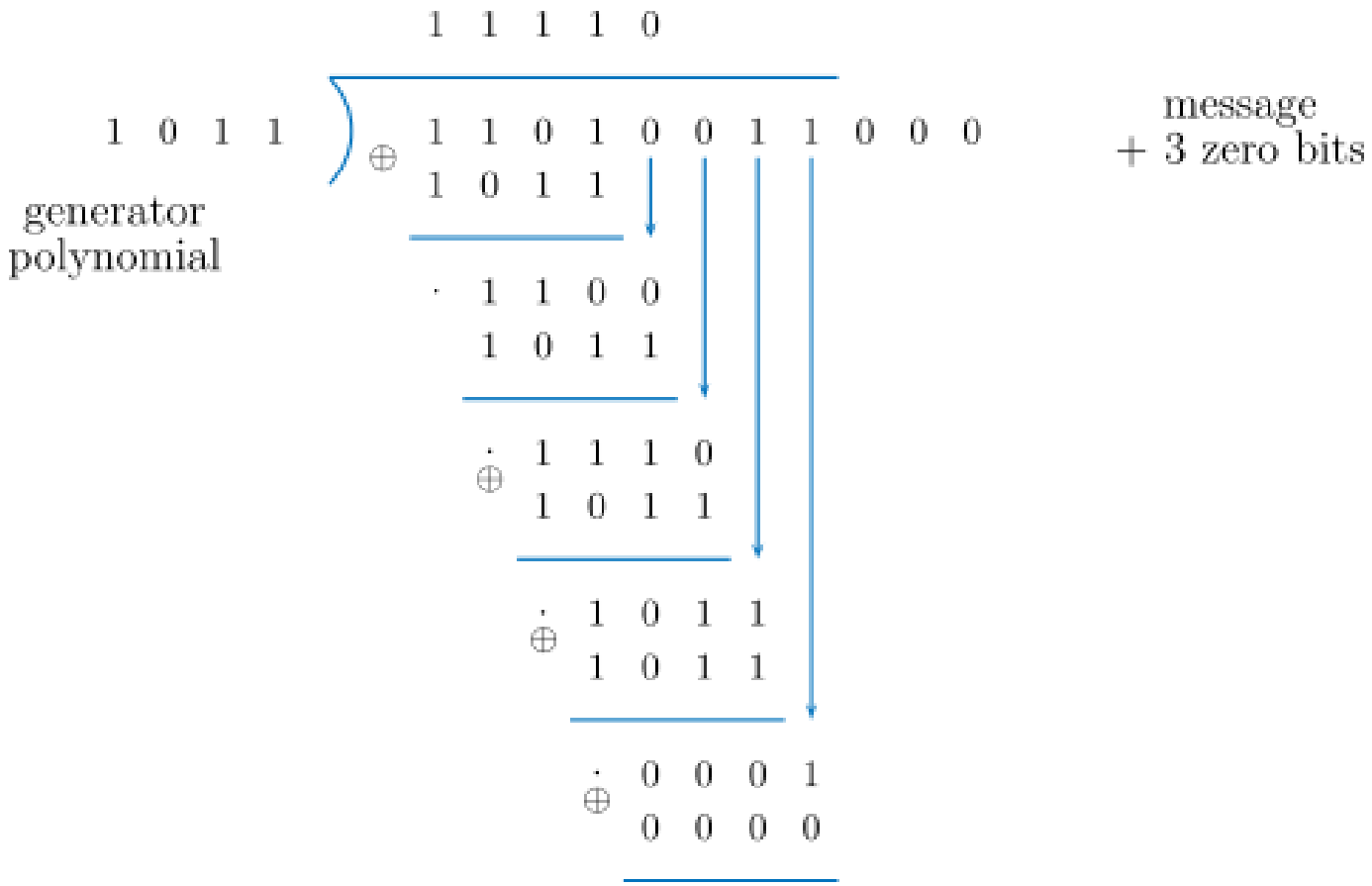
6.5.3 Cyclic Redundancy Checks





© John Wiley & Sons 2018

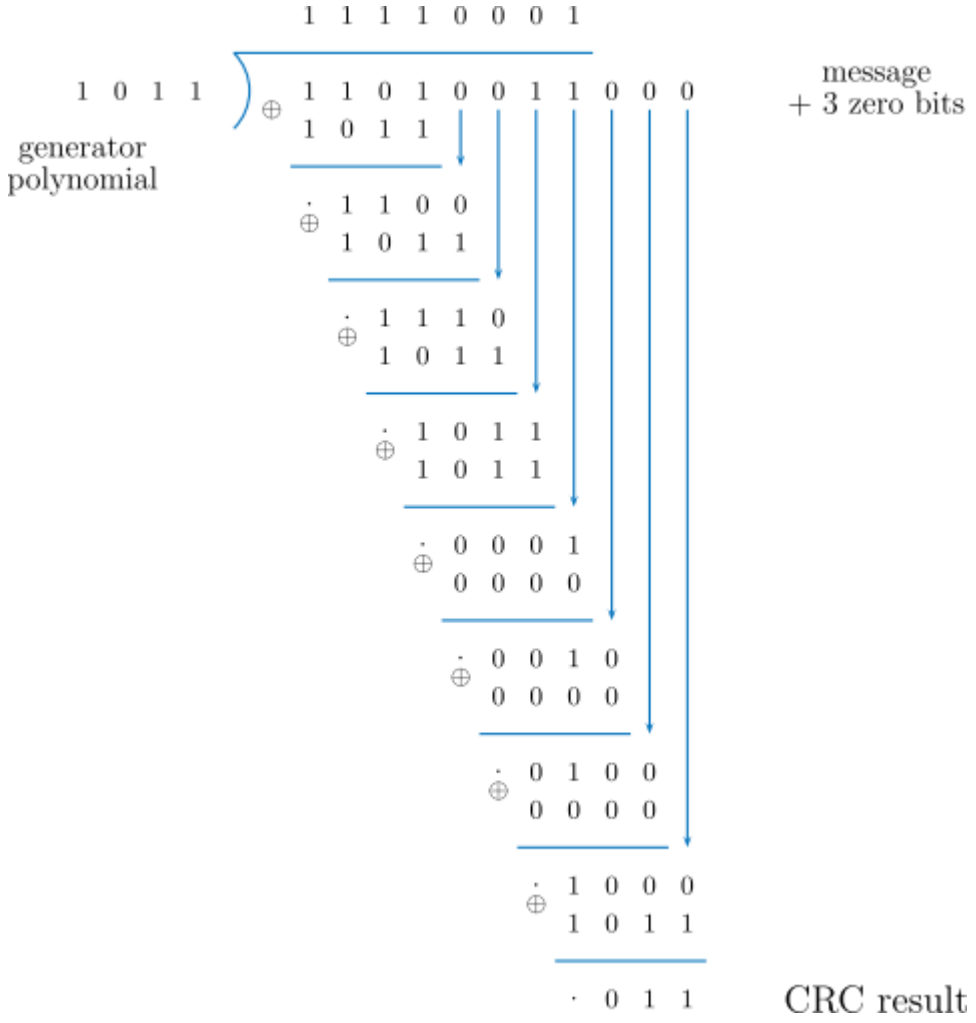
Cyclic Redundancy Checks



© John Wiley & Sons 2018

6.5.3 Cyclic Redundancy Checks

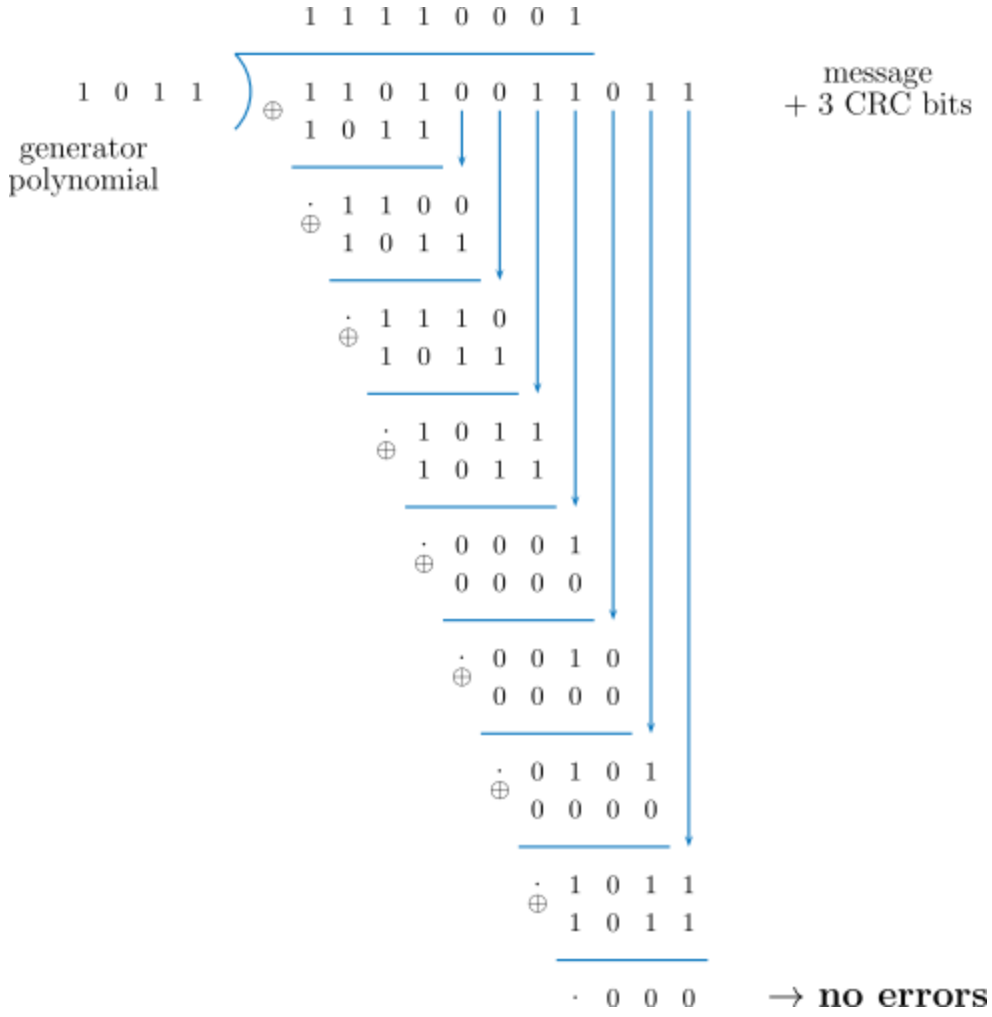
- Sonunda bir kalan elde edilir (bu örnekte 3 bit)
- Bu kalan , sondaki 3 bit sıfırın yerine konur
- Artık bölme işlemi kalansız olacaktır.
- Kaydırma ve çıkarma işlemleri kaydıran yazmaç ve XOR kapıları ile yapılır.



© John Wiley & Sons 2018

Cyclic Redundancy Checks

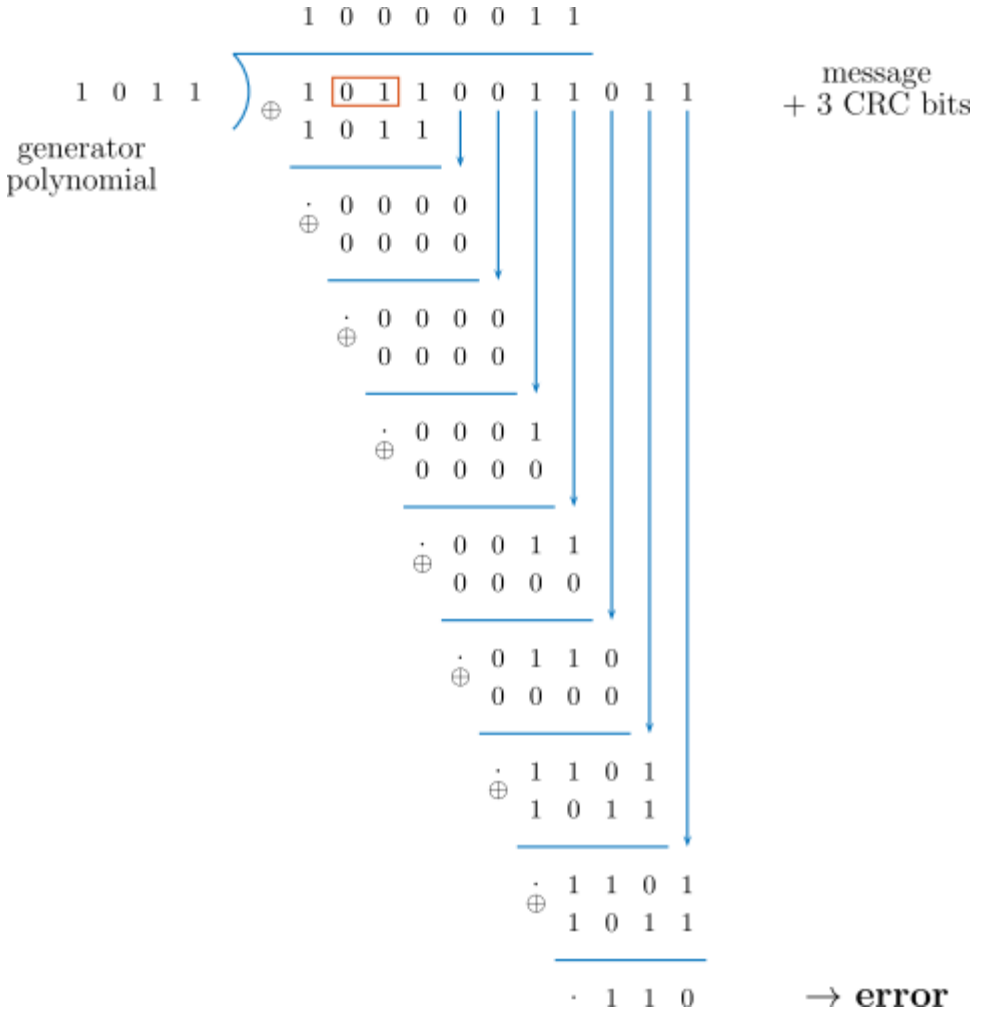
- Alıcı aynı işi tekrar eder
- Kalan sıfır olursa hata yoktur



© John Wiley & Sons 2018

Cyclic Redundancy Checks

- Alıcı aynı işi tekrar eder
- Kalan sıfır olursa hata yoktur
- Bazen o kadar çok hata olur ki, bölme işlemi yine kalansız olur ve bu tespit edilemez. Ancak bu ihtimal çok çok düşüktür



© John Wiley & Sons 2018

6.1 Explain the difference between *error detection* and *error correction*. In relation to these concepts:

a) Define the terms *error distance* and *Hamming distance*.
b) Explain how two-dimensional parity could be used to correct errors.
c) To detect d errors, what Hamming distance is required in the code-words? Explain how this comes about.
d) To correct d errors, what Hamming distance is required in the code-words? Explain how this comes about.

6.2 Hamming codes are able to detect, and in many cases correct, errors.

a) Derive the Boolean check bit generating equations for a Hamming (7,4) error-correcting code.
b) Calculate the check bits for the 4-bit message block `1110`.
c) Calculate the syndrome bits if the message is correctly received.
d) Calculate the syndrome bits if the message is erroneously received as `1111`. Does the syndrome correctly identify the error?
e) Calculate the syndrome bits if the message is erroneously received as `1101`. Does the syndrome correctly identify the error?

© John Wiley & Sons 2018

6.3 Create a table similar to Table 6.4 and show that a Hamming (15, 11) code is feasible.

© John Wiley & Sons 2018

6.4 With reference to error-detecting codes:

a) Explain what is meant by a checksum. What practical systems use a checksum for error checking?

b) Explain what is meant by a CRC. What practical systems use a CRC for error checking?

© John Wiley & Sons 2018

6.5 Using the CRC generator 1011 and message 1010 1110,

a) Calculate the CRC remainder if there are no errors. Check your CRC by replacing it and performing the CRC division process a second time to show that the remainder is all zero.

b) If there is a 3-bit error burst 111 starting at the third bit transmitted, show that the bit sequence becomes 1001 0110 101. Then show that the error is detected.

c) If there is an error burst that happens to be identical to the generator, starting at bit position 4, show that the bit sequence becomes 1011 1000 101. Then show that this error is not detected.

© John Wiley & Sons 2018