William Stallings
Data and Computer Communications

Chapter 6

Digital Data Communications Techniques

Digital Data Communications Techniques

Synchronization

Standard Interchange Codes

American Standards Committee for Information Interchange (ASCII):

Bit Positions			7	0	0	0	0	1	1	1	1
			6	0	0	1	1	0	0	1	1
			5	0	1	0	1	0	1	0	1
4	3	2	1								
0	0	0	0	NUL	DLE	SP	0	@	P	\	p
0	0	0	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	STX	DC2	“	2	B	R	b	r
0	0	1	1	ETX	DC3	#	3	C	S	c	s
0	1	0	0	EOT	DC4	$	4	D	T	d	t
0	1	0	1	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	ACK	SYN	&	6	F	V	f	v
0	1	1	1	BEL	ETB	’	7	G	W	g	w
1	0	0	0	BS	CAN	(	8	H	X	h	x
1	0	0	1	HT	EM	)	9	I	Y	i	y
1	0	1	0	LF	SUB	*	:	J	Z	j	z
1	0	1	1	VT	ESC	+	;	K	[	k	{
1	1	0	0	FF	FS	,	<	L	\	l	|
1	1	0	1	CR	GS	-	=	M	]	m	}
1	1	1	0	SO	RS	.	>	N	^	n	~
1	1	1	1	SI	US	/	?	O		o	DEL

Standard Interchange Codes

American Standards Committee for Information Interchange (ASCII):

Bit Positions			7	0	0	0	0	1	1	1	1
			6	0	0	1	1	0	0	1	1
			5	0	1	0	1	0	1	0	1
4	3	2	1								
0	0	0	0	NUL	DLE	SP	0	@	P	\	p
0	0	0	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	STX	DC2	“	2	B	R	b	r
0	0	1	1	ETX	DC3	#	3	C	S	c	s
0	1	0	0	EOT	DC4	$	4	D	T	d	t
0	1	0	1	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	ACK	SYN	&	6	F	V	f	v
0	1	1	1	BEL	ETB	’	7	G	W	g	w
1	0	0	0	BS	CAN	(	8	H	X	h	x
1	0	0	1	HT	EM	)	9	I	Y	i	y
1	0	1	0	LF	SUB	*	:	J	Z	j	z
1	0	1	1	VT	ESC	+	;	K	[	k	{
1	1	0	0	FF	FS	,	<	L	\	l	|
1	1	0	1	CR	GS	-	=	M	]	m	}
1	1	1	0	SO	RS	.	>	N	^	n	~
1	1	1	1	SI	US	/	?	O		o	DEL

Standard Interchange Codes

Extended Binary Coded Decimal Interchange Code (EBCDIC):

Bit Positions			8	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
			7	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
			6	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
			5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
4	3	2	1																
0	0	0	0	NUL	DLE	DS		SP	&	-									0
0	0	0	1	SOH	DC1	SOS				/		a	j			A	J		1
0	0	1	0	STX	DC2	FS	SYN					b	k	s		B	K	S	2
0	0	1	1	ETX	DC3							c	l	t		C	L	T	3
0	1	0	0	PF	RES	BYP	PN					d	m	u		D	M	U	4
0	1	0	1	HT	NL	LF	RS					e	n	v		E	N	V	5
0	1	1	0	LC	BS	EOB	UC					f	o	w		F	O	W	6
0	1	1	1	DEL	IL	PRE	EOT					g	p	x		G	P	X	7
1	0	0	0		CAN							h	q	y		H	Q	Y	8
1	0	0	1		EM							i	r	z		I	N	Z	9
1	0	1	0	SMM	CC	SM		¢	!		:								
1	0	1	1	VT				.	$	’	#								
1	1	0	0	FF	IFS		DC4	<	*	%	@								
1	1	0	1	CR	IGS	ENQ	NAK	(	)	-	,								
1	1	1	0	SO	IRS	ACK		+	;	>	=								
1	1	1	1	SI	IUS	BEL	SUB	|	¬	?	”								□

Standard Interchange Codes

Extended Binary Coded Decimal Interchange Code (EBCDIC):

Bit Positions			8	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
			7	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
			6	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
			5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
4	3	2	1																
0	0	0	0	NUL	DLE	DS		SP	&	-									0
0	0	0	1	SOH	DC1	SOS				/		a	j			A	J		1
0	0	1	0	STX	DC2	FS	SYN					b	k	s		B	K	S	2
0	0	1	1	ETX	DC3							c	l	t		C	L	T	3
0	1	0	0	PF	RES	BYP	PN					d	m	u		D	M	U	4
0	1	0	1	HT	NL	LF	RS					e	n	v		E	N	V	5
0	1	1	0	LC	BS	EOB	UC					f	o	w		F	O	W	6
0	1	1	1	DEL	IL	PRE	EOT					g	p	x		G	P	X	7
1	0	0	0		CAN							h	q	y		H	Q	Y	8
1	0	0	1		EM							i	r	z		I	N	Z	9
1	0	1	0	SMM	CC	SM		¢	!		:								
1	0	1	1	VT				.	$	’	#								
1	1	0	0	FF	IFS		DC4	<	*	%	@								
1	1	0	1	CR	IGS	ENQ	NAK	(	)	-	,								
1	1	1	0	SO	IRS	ACK		+	;	>	=								
1	1	1	1	SI	IUS	BEL	SUB	|	¬	?	”								□

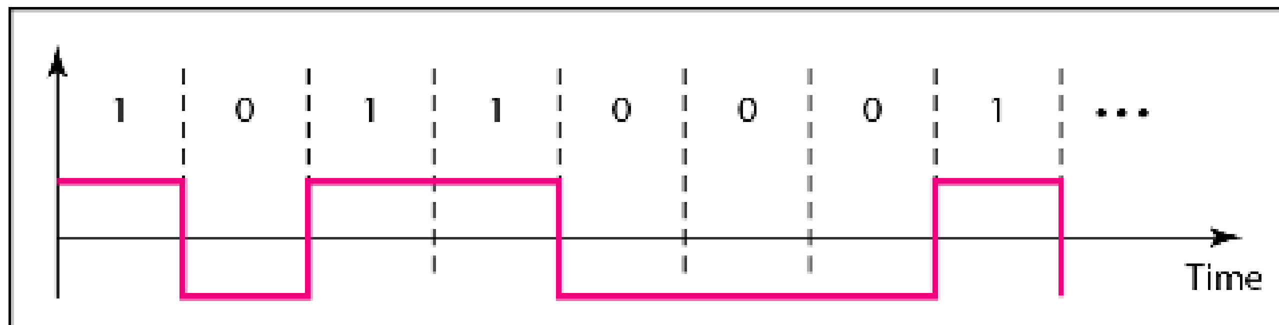
Synchronization

- To correctly interpret the signals received, the **receiver's bit intervals must correspond exactly to the sender's bit intervals** (the same clock rate).

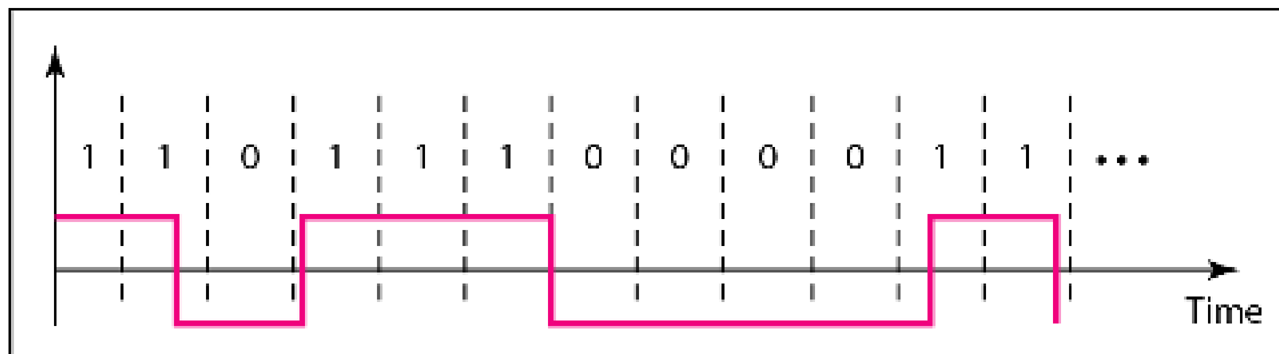
- timing problems require a mechanism to synchronize the transmitter and receiver
  - **if clocks not aligned and drifting will sample at wrong time after sufficient bits are sent**

Lack of Synchronization

- Example: Faster receiver clock



a. Sent

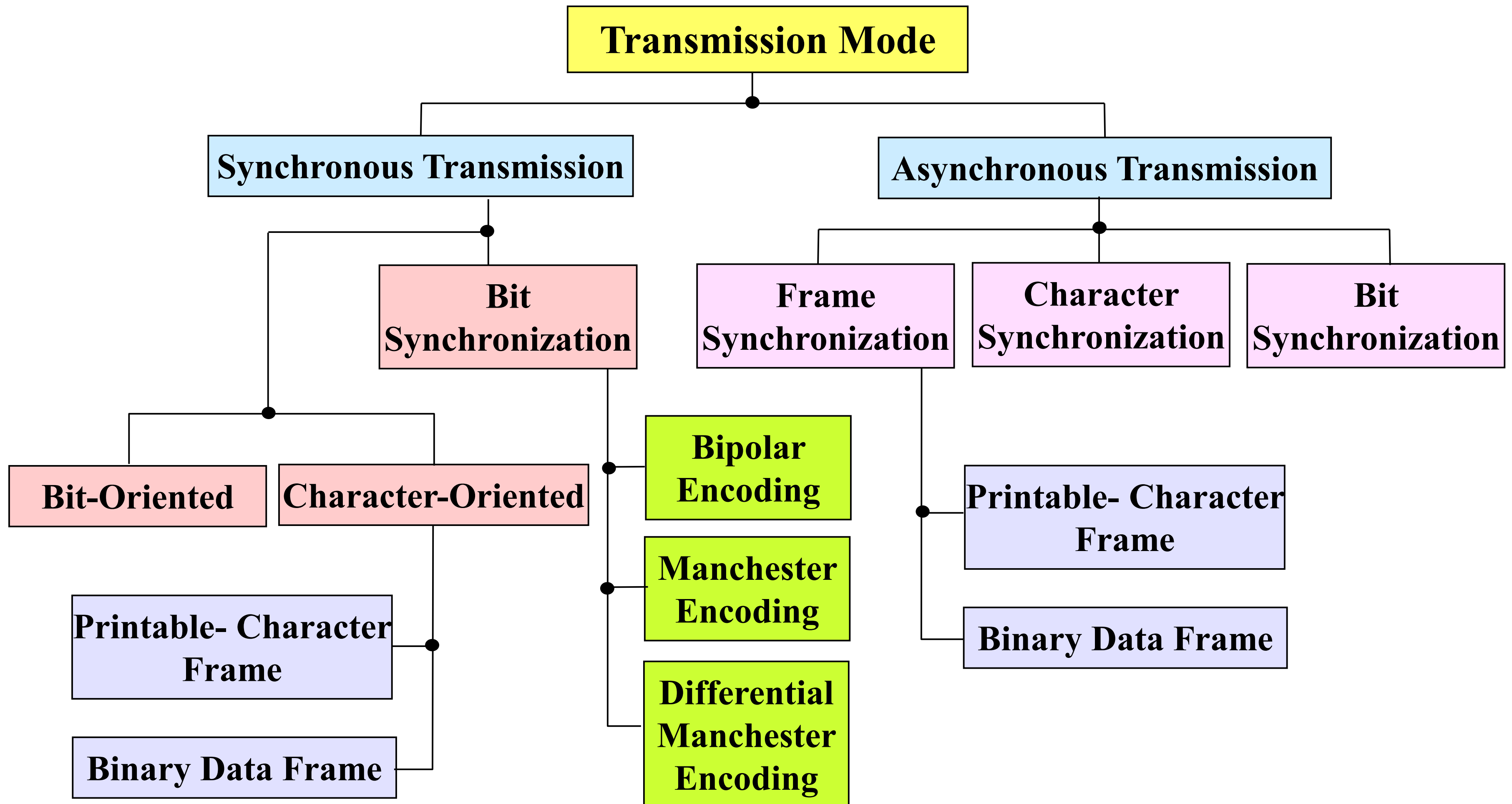


b. Received

Synchronization

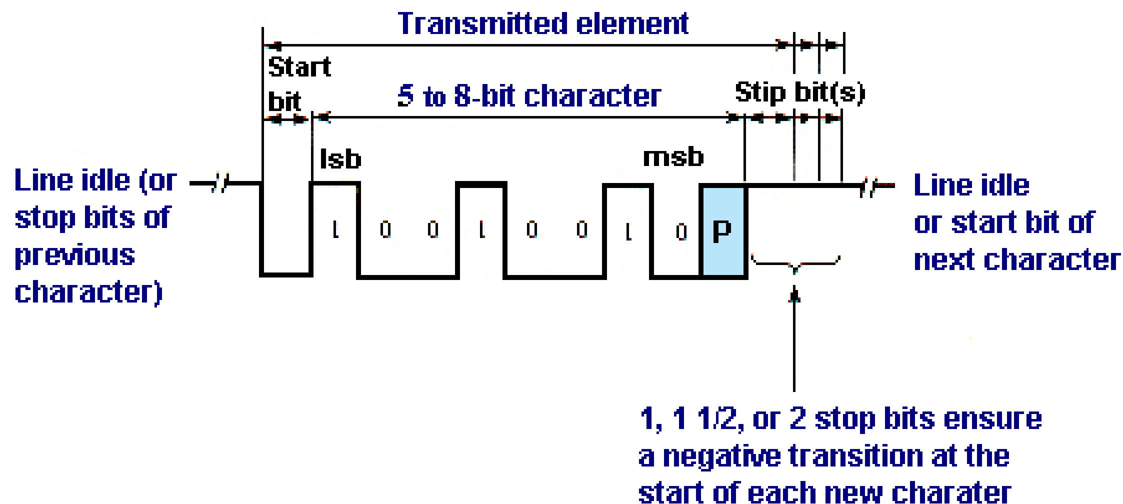
- Two solutions to synchronizing clocks
  - **Asynchronous transmission**
  - **Synchronous transmission**

Transmission Modes



Asynchronous Transmission

- In **asynchronous transmission**, the receiver clock (R×C) runs unsynchronized with respect to the incoming signal (R×D).
- Each character (byte) is encapsulated between an additional **start bit** and one or more **stop bits**.
- The state of the signal on the transmission line between characters is **idle** state.



Asynchronous Transmission

- send one character (e.g. 8 bits) at a time
- no data: idle state (usually negative volt)
- start bit (usually zero)
- data: usually use NRZ-L
- party bit can be added for error control
- stop bit, 1-2 bit time, same as idle
- timing requirement is modest (within 1 char)

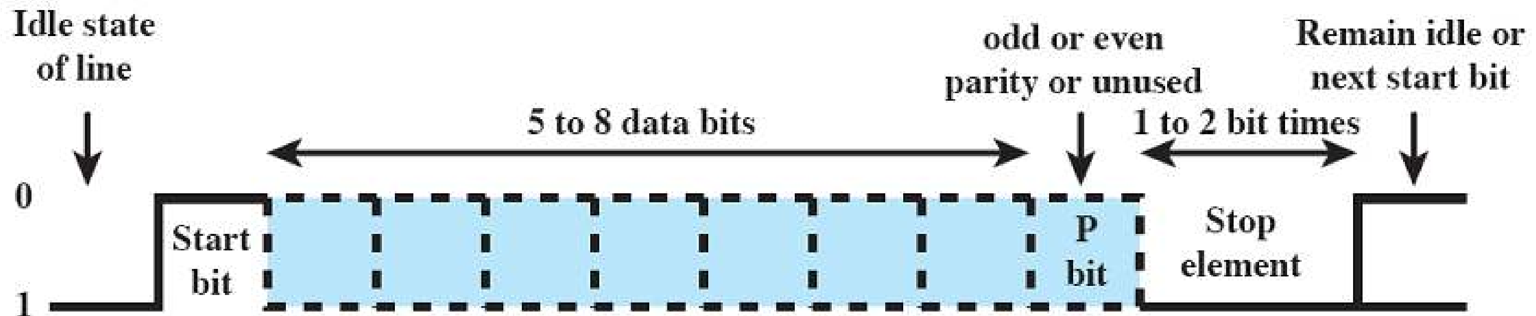
Asynchronous Transmission

- **Parity**
  - parity bit set so character has even (even parity) or odd (odd parity) number of ones
  - Error Detection

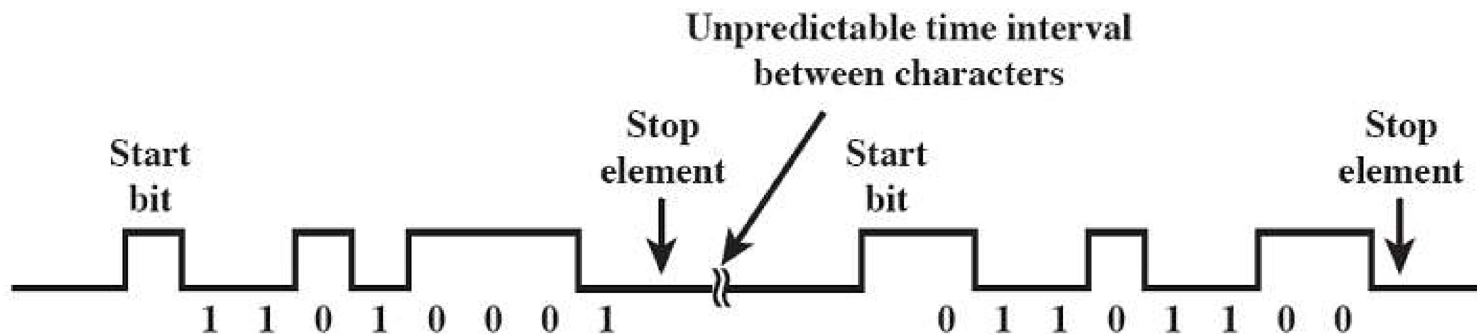
Character							Odd Parity
0	1	0	1	0	1	1	1
1	1	1	1	1	0	0	0
0	0	0	0	0	0	0	1

Character							Even Parity
0	1	0	1	0	1	1	0
1	1	1	1	1	0	0	1
0	0	0	0	0	0	0	0

Asynchronous Transmission

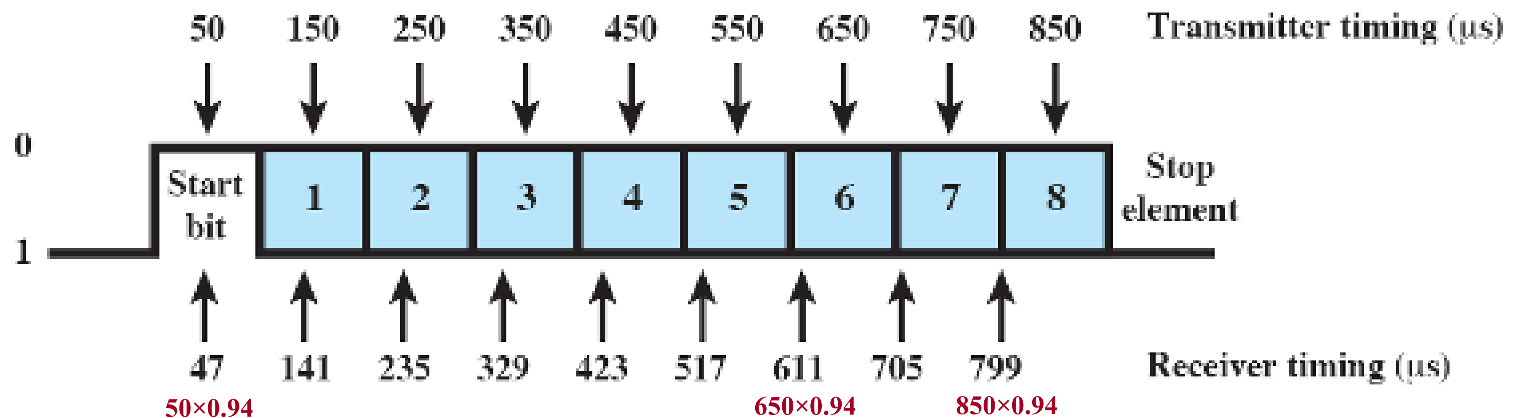


(a) Character format **(NRZ-L)**



(b) 8-bit asynchronous character stream **(NRZ-L)**

Asynchronous Transmission Example



- Data rate = 10 kbps, bit time = 0.1 ms = 100 μs
- Receiver is faster by 6% of bit time = 6 μs
- Receiver samples bit every 94 μs
- Last bit is in error

Asynchronous Transmission

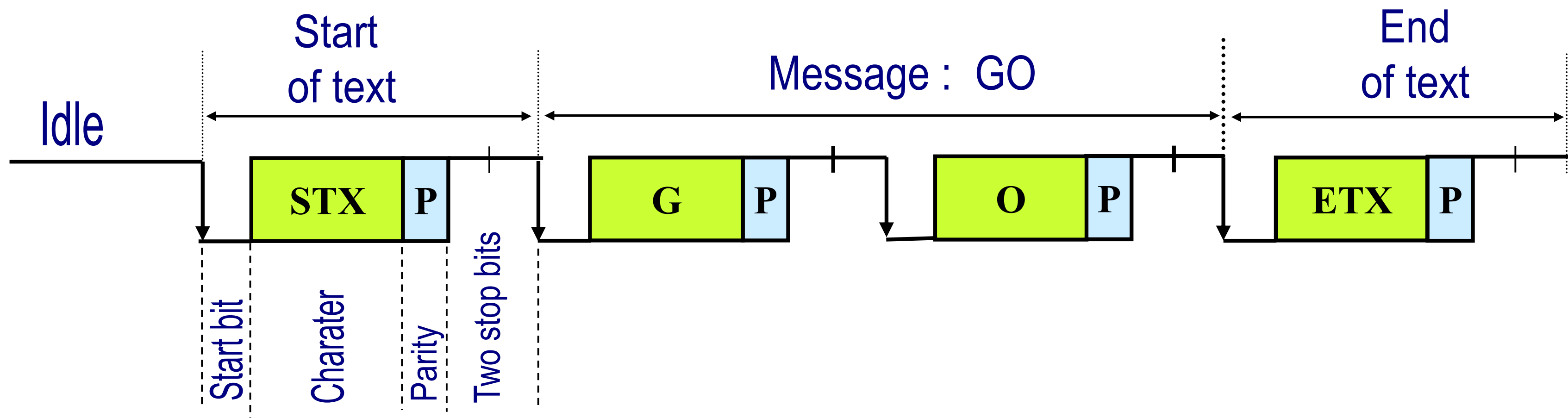
- In previous example, actually two errors
  - **last bit is incorrect**
  - When **a valid stop bit is not detected at end of each character** (i.e., it is supposed to be logic 1 and found logic 0) → **Framing Error**

Asynchronous Transmission

- Advantage
  - **simple and cheap**
- Disadvantage
  - **requires overhead 2 to 4 bits / character**
  - for 8 bit char, no parity, 1 stop bit, 20% overhead

Asynchronous Transmission – Example 1

Construct the transmitted frame using **asynchronous transmission mode** which contains the following data: **GO**. Assume that the number of bits per character is 8, the number of stop bits is 2, and parity bit is used.



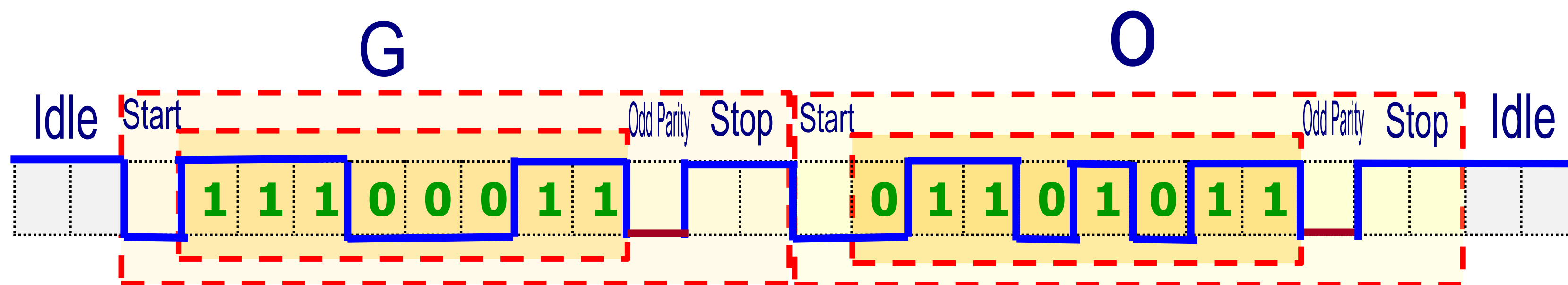
Efficiency (η)= 8/(1+8+1+2) = 8/12 = 66.67%

Asynchronous Transmission – Example 1

Construct the transmitted frame using **asynchronous transmission mode** which contains the following data: **GO**. Assume that the number of bits per character is 8, the number of stop bits is 2, and odd parity bit is used.

Hints:

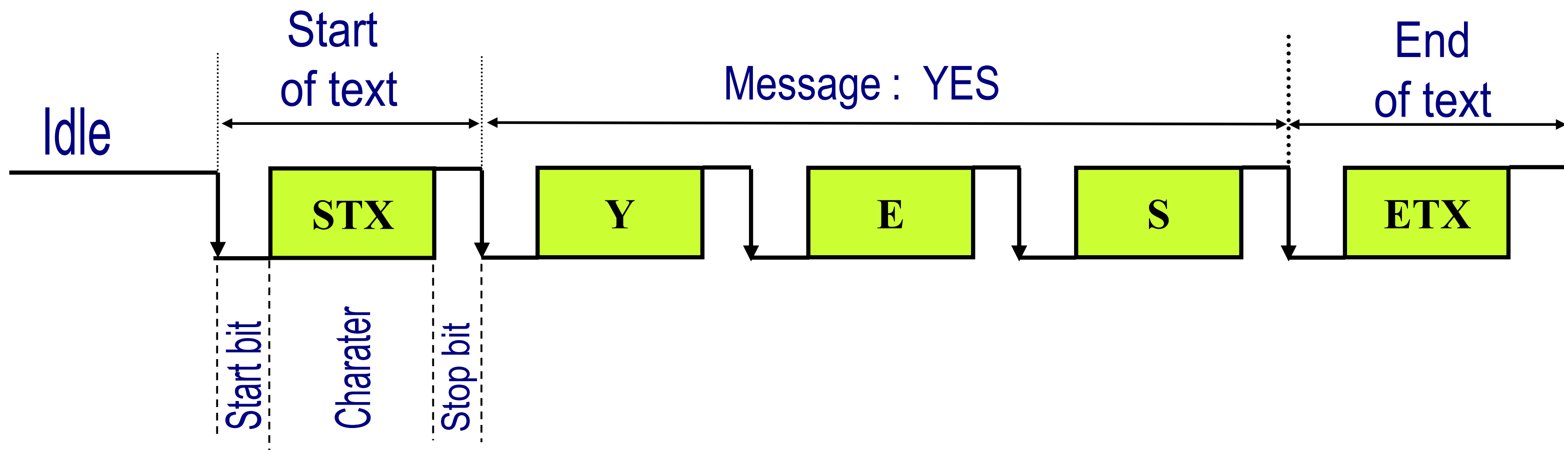
1. No need to include framing characters such as STX and ETX
2. G=**1100 0111**, O=**1101 0110**, STX=**1101 0110** , ETX=**1101 0110**
3. The least significant bit (**lsb**) is transmitted first



Efficiency (η)= 8/(1+8+1+2) = 8/12 = 66.67%

Asynchronous Transmission – Example 2

Construct the transmitted frame using **asynchronous transmission mode** which contains the following data: **YES**. Assume that the number of bits per character is 7, the number of stop bits is 1 and no parity bit is used.



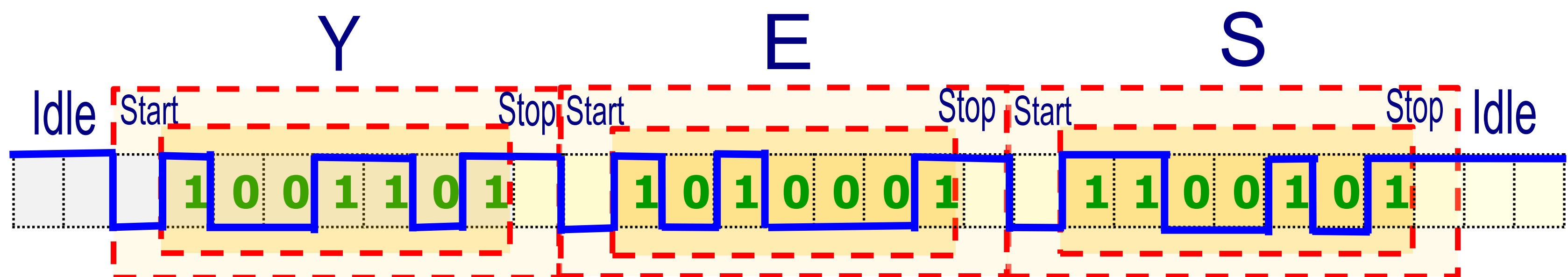
Efficiency (η) = 7/(1+7+1) = 7/9 = 77.78%

Asynchronous Transmission – Example 2

Construct the transmitted frame using **asynchronous transmission mode** which contains the following data: **YES**. Assume that the number of bits per character is 7, the number of stop bits is 1 and no parity bit is used.

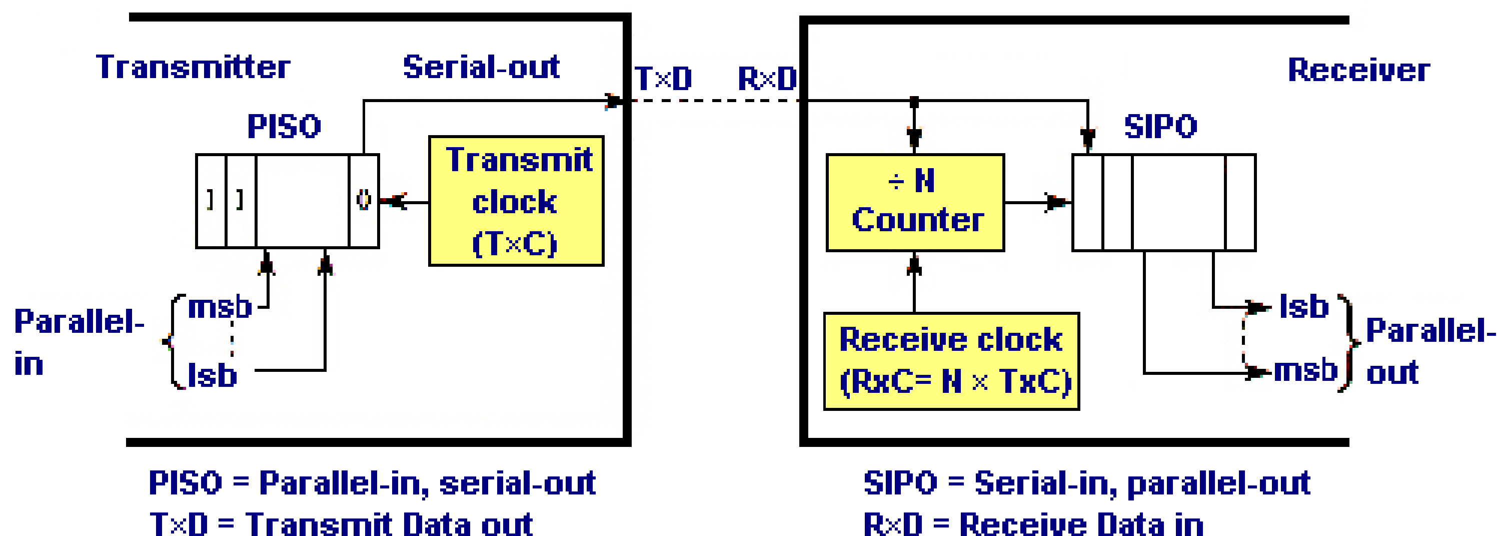
Hints:

1. No need to include framing characters such as STX and ETX
2. Y=**101 1001**, E=**100 0101**, S=**101 0011**
3. The least significant bit (lsb) is transmitted first



Efficiency (η) = 7/(1+7+1) = 7/9 = 77.78%

Asynchronous Transmission



Asynchronous Transmission

TxC
Time
lsb
msb
TxD
Idle
0
Start bit
5 to 8-bit per character
Stop bit(s)

lsb
msb
Idle
RxD
0
RxC
Time
Actual edge within
one clock cycle

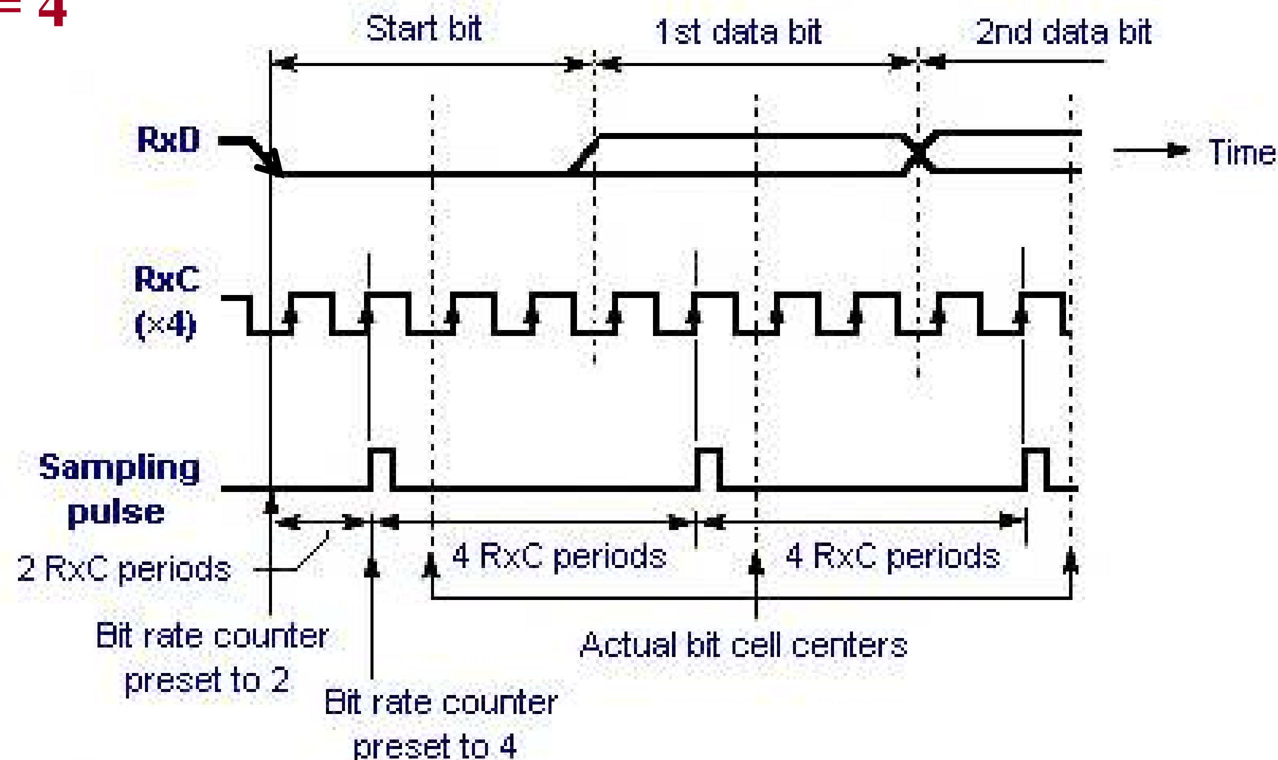
Asynchronous Transmission

Bit Synchronization using Asynchronous Transmission:

- The local receiver clock is *N* times the transmitted bit rate (***N*=16 is common**).
- The first **1→0 transition** is associated with the start bit.
- **Each bit is sampled at the center to avoid delay distortion problem.**
- **After the first transition is detected, the signal is sampled after *N*/2 clock cycles and then subsequently after *N* clock cycles for each bit in the character.**

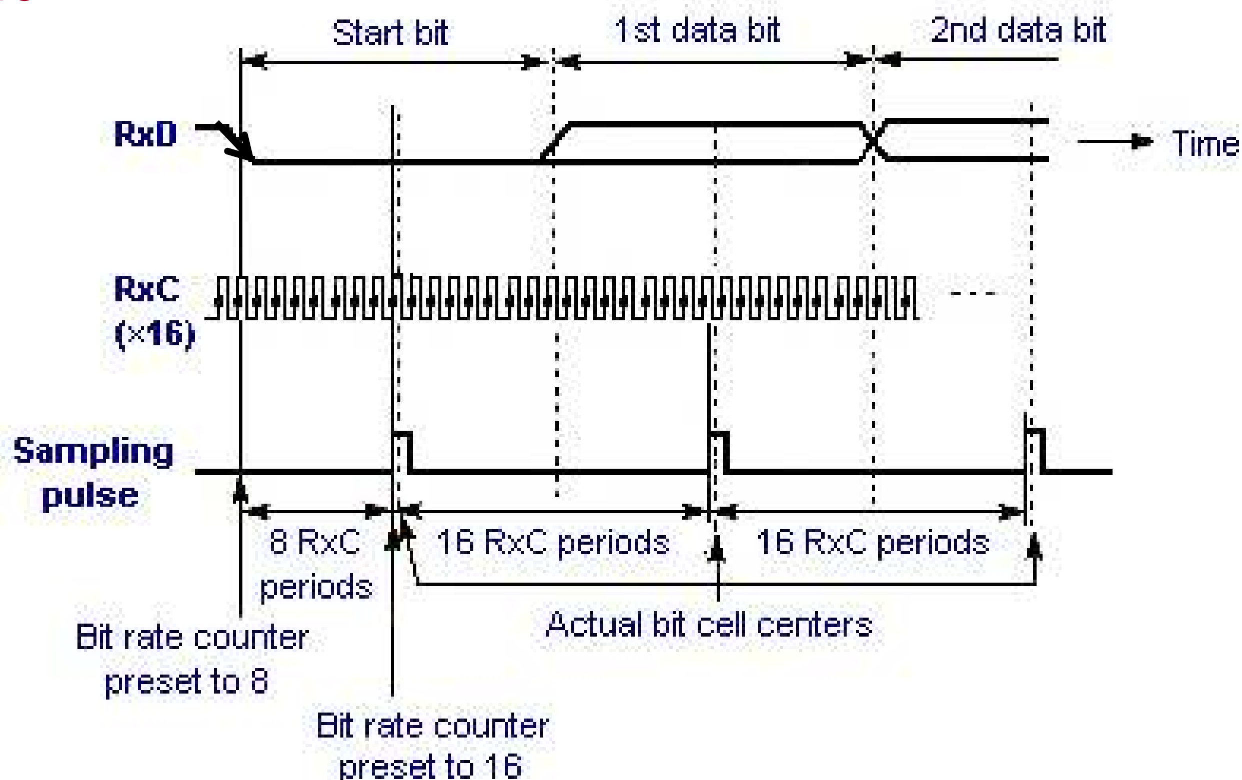
Asynchronous Transmission

$N = 4$

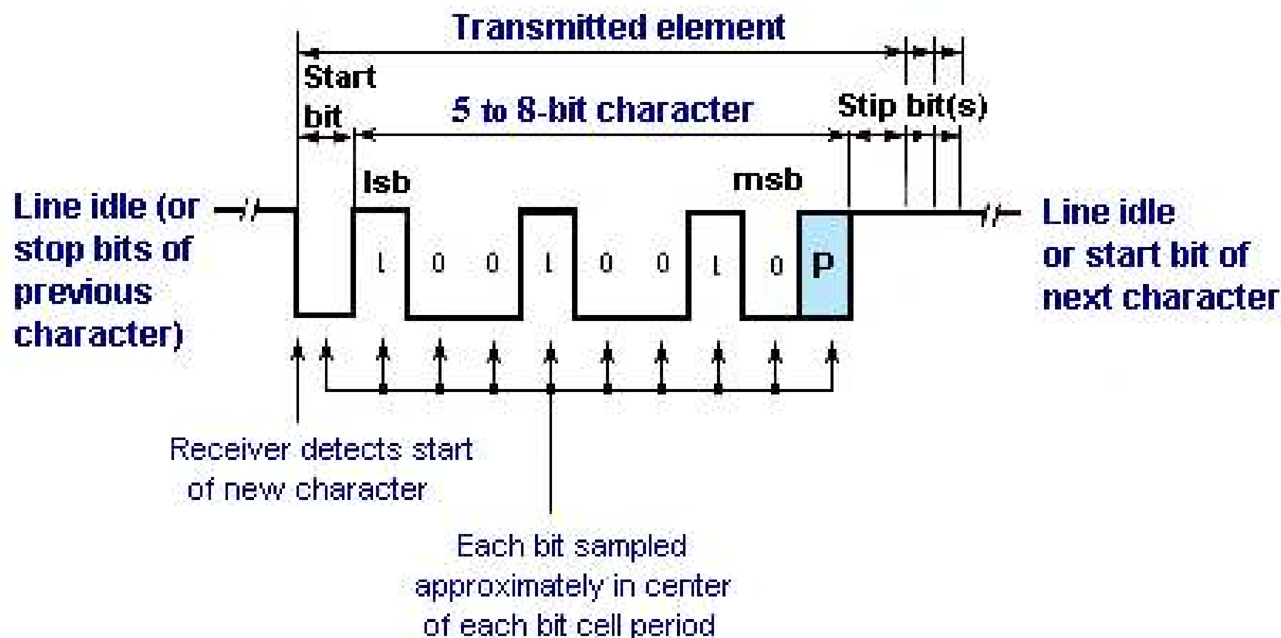


Asynchronous Transmission

$N = 16$



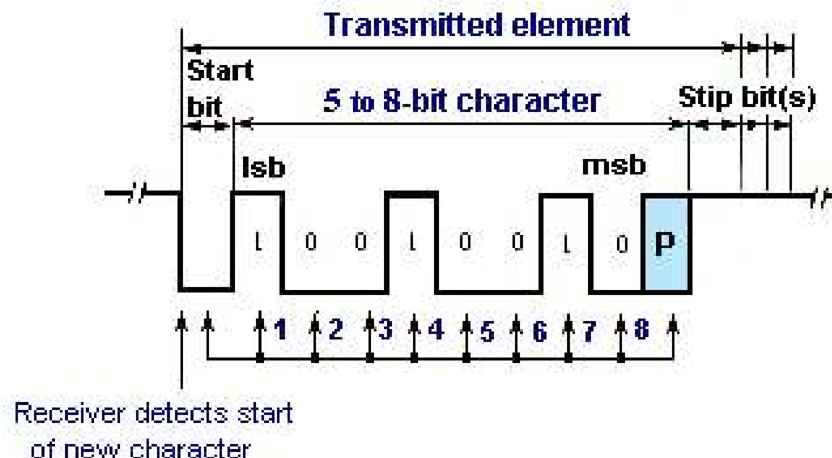
Asynchronous Transmission



Asynchronous Transmission

Character Synchronization using Asynchronous Transmission:

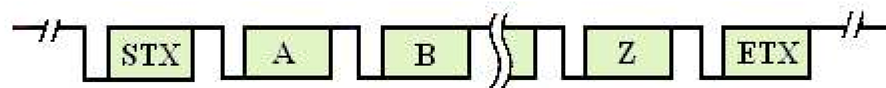
After the start bit is detected, the receiver achieves character synchronization simply **by counting the programmed number of bits.**



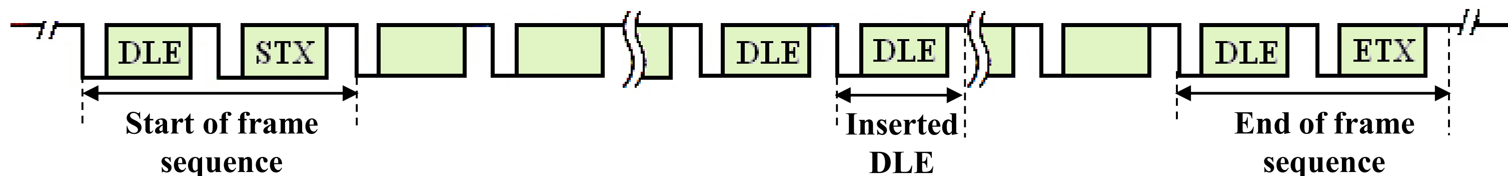
Asynchronous Transmission

Frame Synchronization using Asynchronous Transmission:

- Frame synchronization is used to determine the start and end of frame.
- 1. Printable Characters
  - Encapsulate the complete block between two non-printable characters: **STX** (start-of-text) and **ETX** (end-of-text).



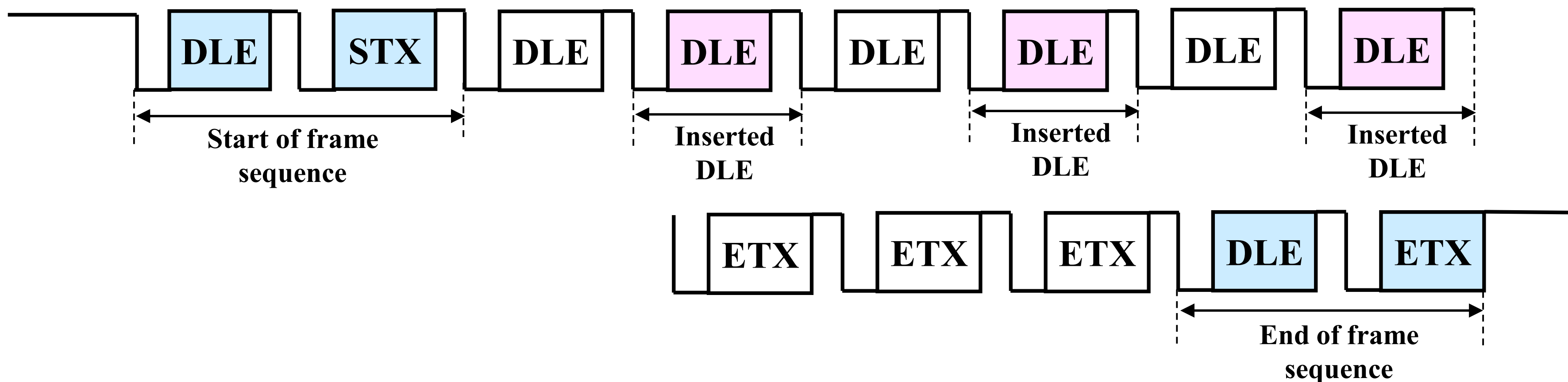
- 2. Binary Data



Asynchronous Transmission - Example

Construct the transmitted frame using asynchronous transmission mode which contains the following binary data: **DLE DLE DLE ETX ETX ETX.**

Assume that the number of stop bits is 1 and no parity bit is used.

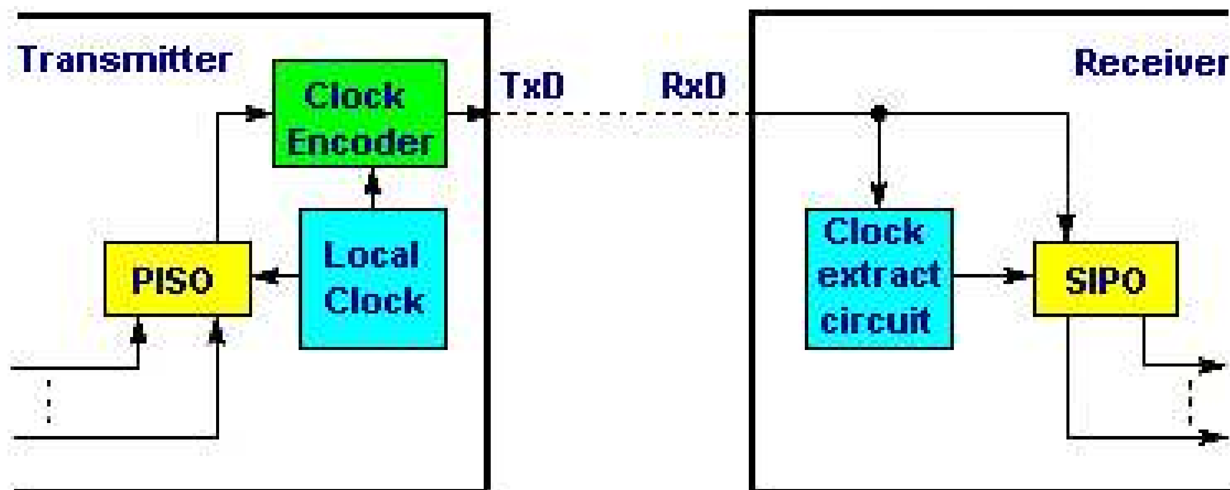


Synchronous Transmission

- block of data transmitted sent as a frame
- **No start or stop bits**
- **clocks must be synchronized**
  - can use separate clock line
  - or embed clock signal in data
- **need to indicate start and end of block**
  - use preamble and postamble
- **more efficient** (lower overhead) than async

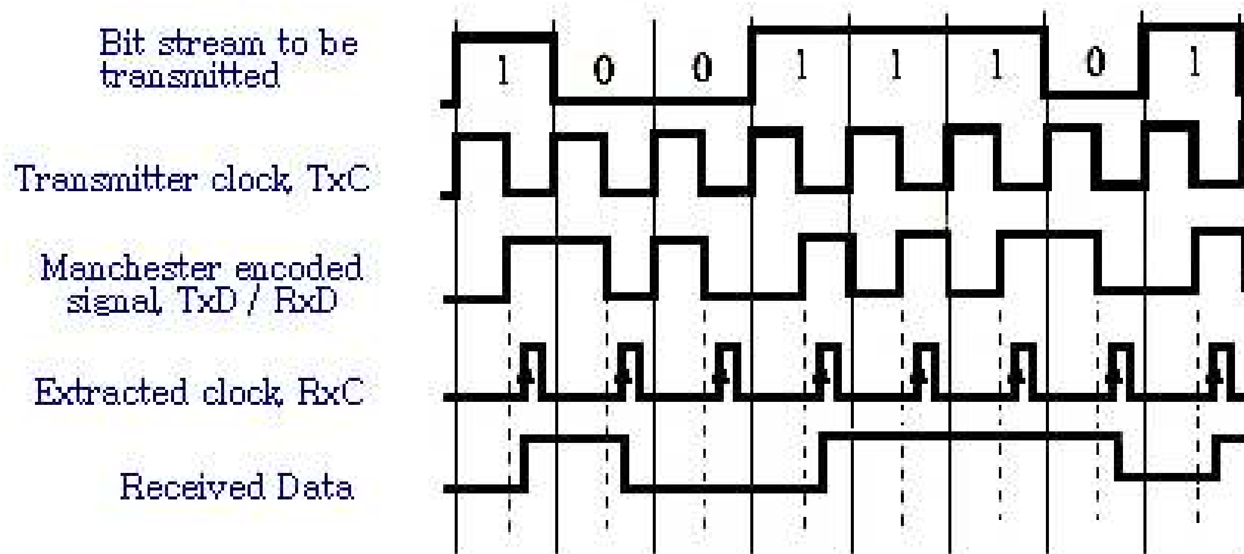
Synchronous Transmission

Clock Encoding and Extraction:



Synchronous Transmission

Clock Encoding and Extraction:



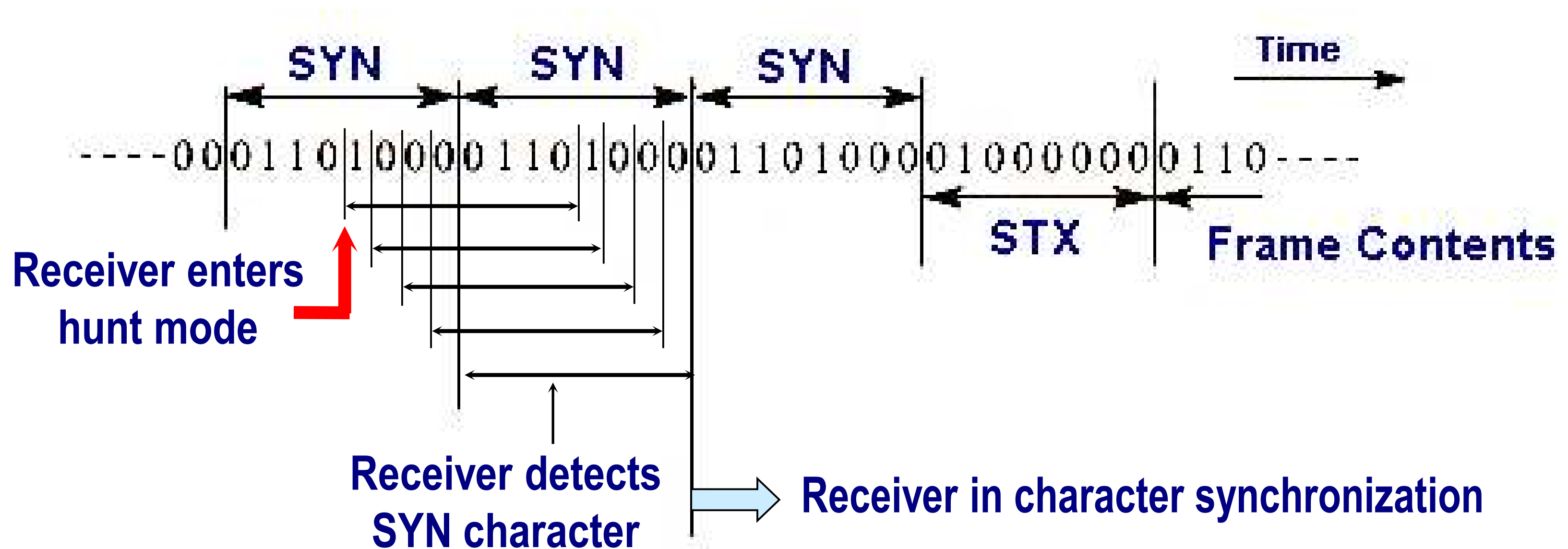
Synchronous Transmission

Character-Oriented Synchronous Transmission:

- The **SYN** character enables the receiver to achieve **character synchronization** before the **STX** character.
- Once the receiver has obtained **bit synchronization**, it enters **Hunt Mode**.
- When the receiver enters hunt mode, it starts to interpret the received bit stream in a window of 8 bits as each new bit is received**. It checks whether the last eight bits were equal to SYN character. If they are not**, **it receives the next bit and repeats the check. If they are, then the correct character boundary is found** and hence the following characters are then read after each subsequent eight bits have been received.

Synchronous Transmission

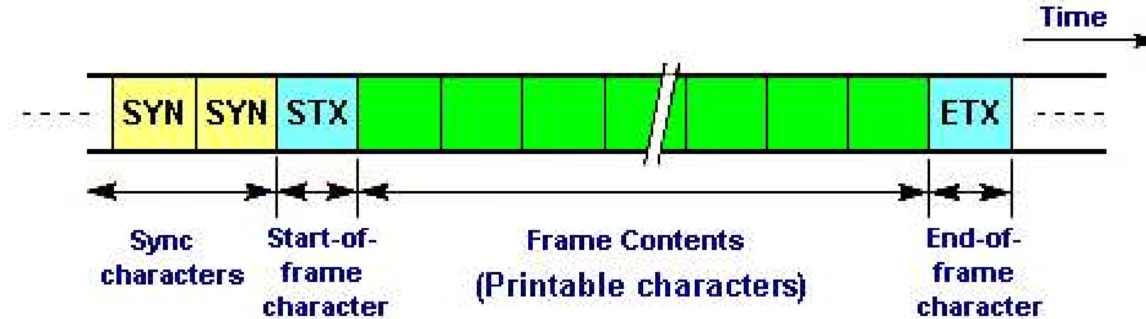
Character-Oriented Synchronous Transmission:



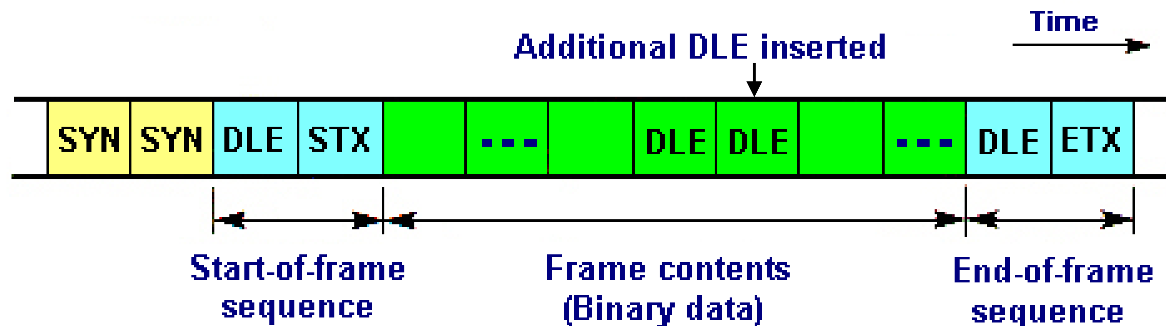
Synchronous Transmission

Character-Oriented Synchronous Transmission:

1. Printable characters:



2. Binary data:



Digital Data Communications Techniques

Error Detection & Correction

Types of Error

- an error occurs when a bit is altered between transmission and reception
- **Single bit errors**
  - **only one bit altered**
  - caused by white noise
- **Burst errors**
  - **contiguous sequence of *B* bits** in which first and last and any number of intermediate bits in error
  - caused by impulse noise or by fading in wireless
  - effect greater at higher data rates

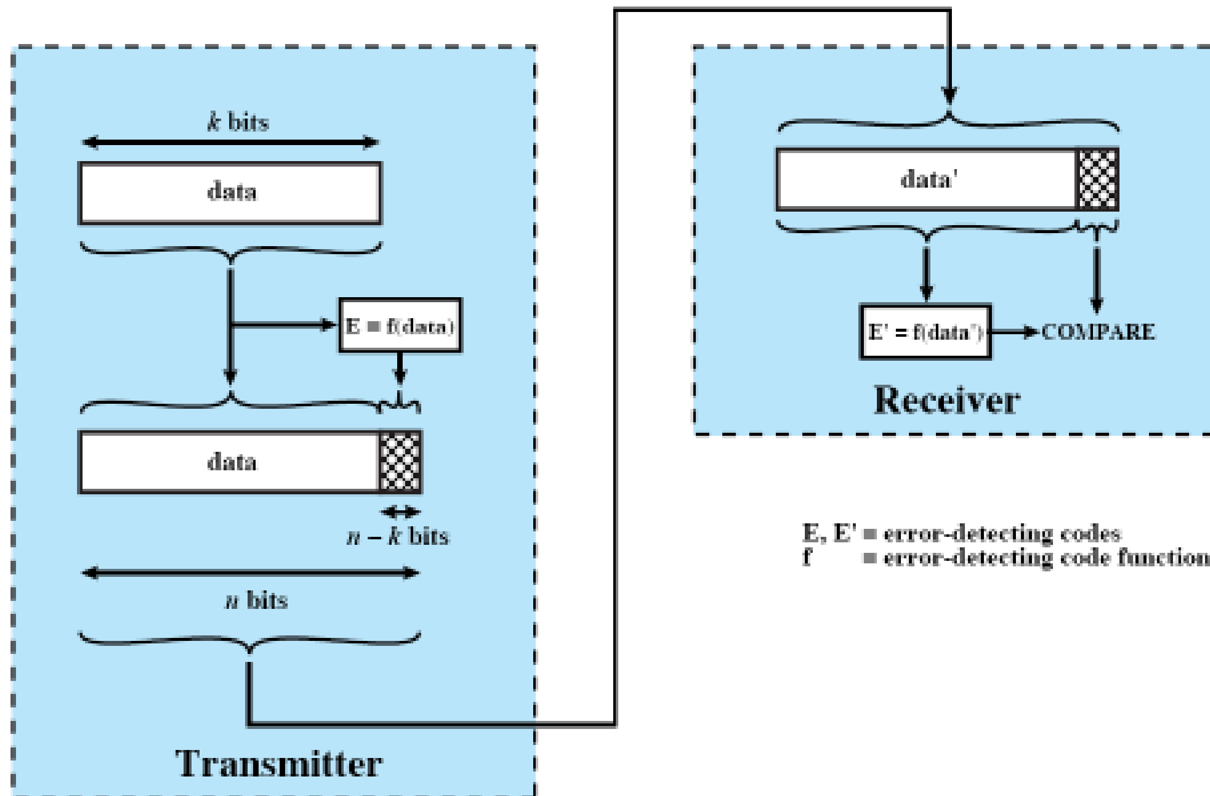
Error Detection

Error Detection

Error Detection

- Error detecting code added by transmitter
- k **data bits** + (n–k) **check bits** = n-bit frame
- Receiver separates k, n–k bits
- Receiver calculates error-detecting code
  - match received code: **assume no error**
  - mismatch: **error**
- **Some errors will be undetected**

Error Detection Process



Error Detection

- **Parity**
  - parity bit set so character has even (even parity) or odd (odd parity) number of ones
  - even number of bit errors goes undetected

Character							Odd Parity
0	1	0	1	0	1	1	1
1	1	1	1	1	0	0	0
0	0	0	0	0	0	0	1

Character							Even Parity
0	1	0	1	0	1	1	0
1	1	1	1	1	0	0	1
0	0	0	0	0	0	0	0

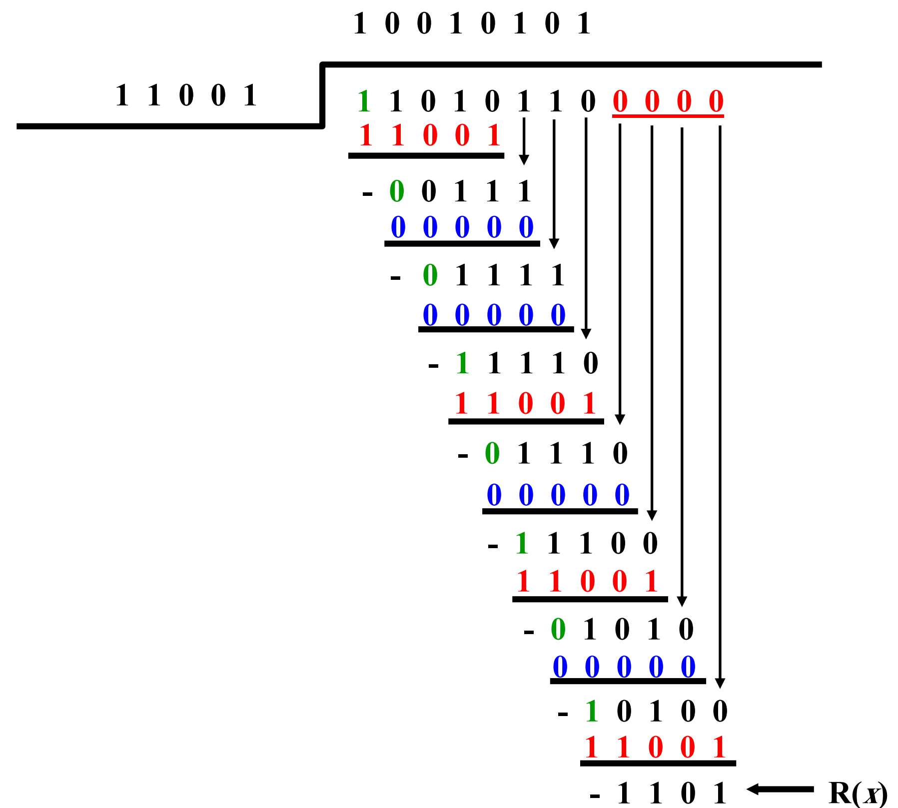
Cyclic Redundancy Check

- one of **most common** and **powerful checks**
- for block of *k* bits transmitter generates an *n* bit **frame check sequence (FCS)**
- **transmits *k+n* bits which is exactly divisible by some number**
- receiver divides frame by that number
  - **if no remainder, assume no error**

Cyclic Redundancy Check Example 1

An 8-bit message **1 1 0 1 0 1 1 0** is to be transmitted across a data link using CRC for error detection.

A generator polynomial $x^4 + x^3 + 1$ is to be used. Find the contents of the **transmitted frame T(x)**.



T(x) = 1 1 0 1 0 1 1 0 1 1 0 1

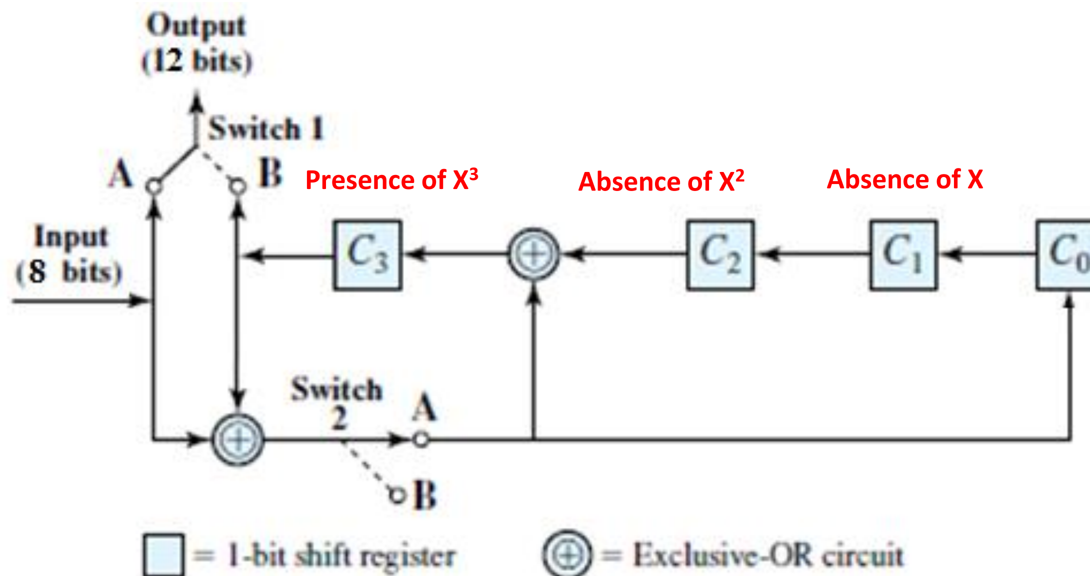
M(x) = 1 1 0 1 0 1 1 0, R(x) = 1 1 0 1

Cyclic Redundancy Check Example 1 – Circuit Implementation

Generator Polynomial $x^4 + x^3 + 1$

The circuit is implemented as follows:

1. The register contains $n - k$ bits, equal to the length of the FCS.
2. There are up to $n - k$ XOR gates.
3. The presence or absence of a gate corresponds to the presence or absence of a term in the divisor polynomial, $P(X)$, excluding the terms 1 and X^{n-k}.

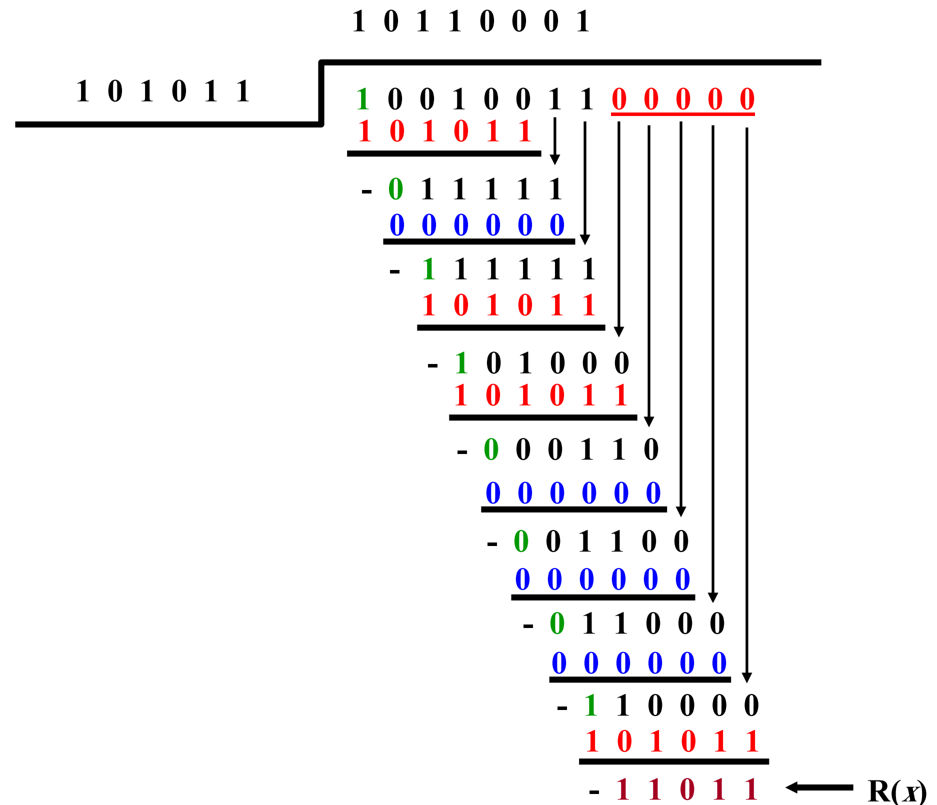


Shift-register implementation

Cyclic Redundancy Check Example 2

An 8-bit message **1 0 0 1 0 0 1 1** is to be transmitted across a data link using CRC for error detection.

A generator polynomial $x^5 + x^3 + x + 1$ is to be used. Find the contents of the **transmitted frame T(x)**.



T(x) = 1 0 0 1 0 0 1 1 1 1 0 1 1

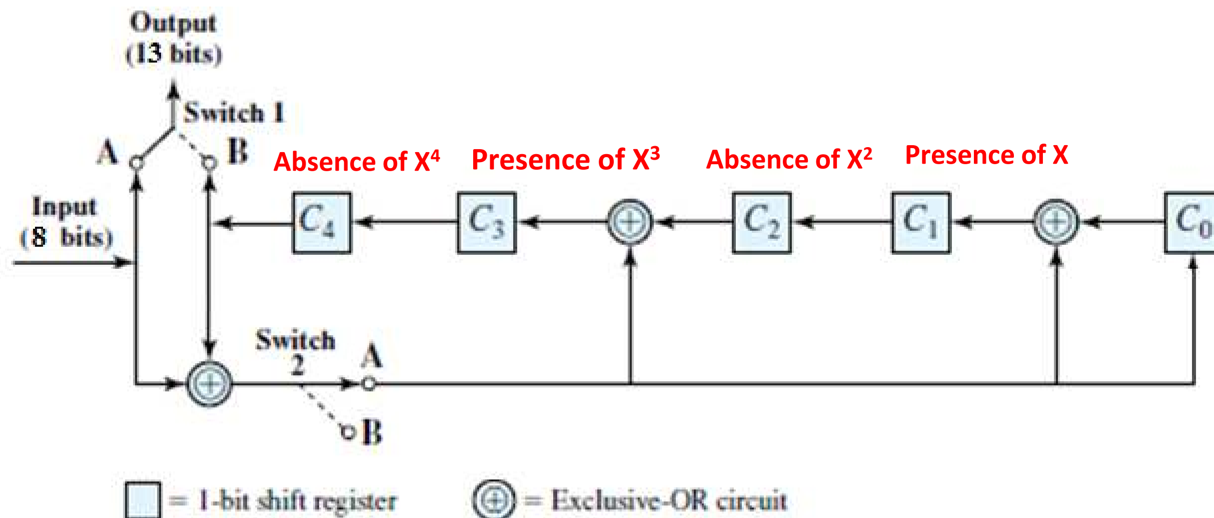
M(x) = 1 0 0 1 0 0 1 1, R(x) = 1 1 0 1 1

Cyclic Redundancy Check Example 2 – Circuit Implementation

Generator Polynomial $x^5 + x^3 + x + 1$

The circuit is implemented as follows:

1. The register contains $n - k$ bits, equal to the length of the FCS.
2. There are up to $n - k$ XOR gates.
3. The presence or absence of a gate corresponds to the presence or absence of a term in the divisor polynomial, $P(X)$, excluding the terms 1 and X^{n-k}.



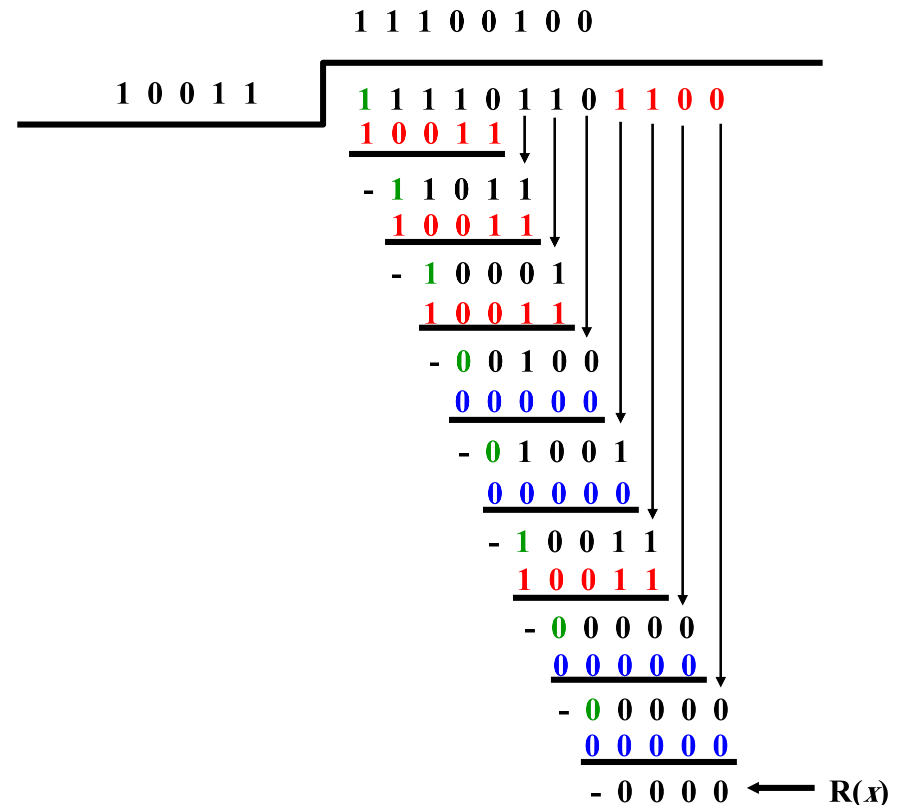
Shift-register implementation

Cyclic Redundancy Check Example 3

Assume that the following frame:

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

is **received** across a data link using CRC for error detection. A generator polynomial $x^4 + x + 1$ is to be used. Check whether the received frame is correct or incorrect. If the frame is correct, then deduce the original message.



R(x) = 0 → The Message is correct

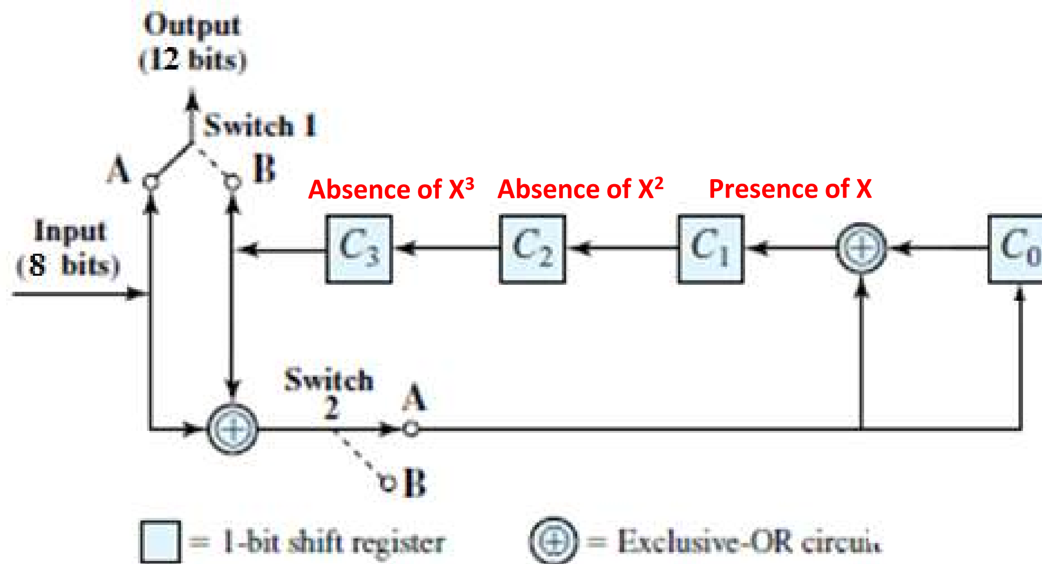
→ M(x) = 1 1 1 1 0 1 1 0

Cyclic Redundancy Check Example 3 – Circuit Implementation

Generator Polynomial $x^4 + x + 1$

The circuit is implemented as follows:

1. The register contains $n - k$ bits, equal to the length of the FCS.
2. There are up to $n - k$ XOR gates.
3. The presence or absence of a gate corresponds to the presence or absence of a term in the divisor polynomial, $P(X)$, excluding the terms 1 and X^{n-k}.



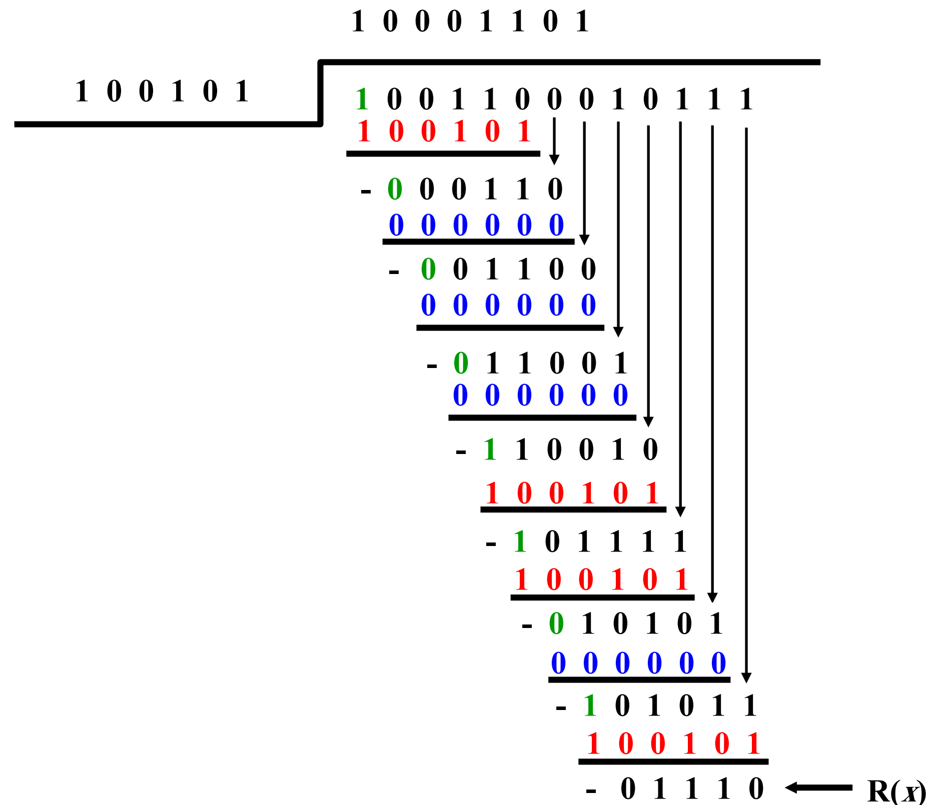
Shift-register implementation

Cyclic Redundancy Check Example 4

Assume that the following frame:

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

is **received** across a data link using CRC for error detection. A generator polynomial $x^5 + x^2 + 1$ is to be used. Check whether the received frame is correct or incorrect. If the frame is correct, then deduce the original message.

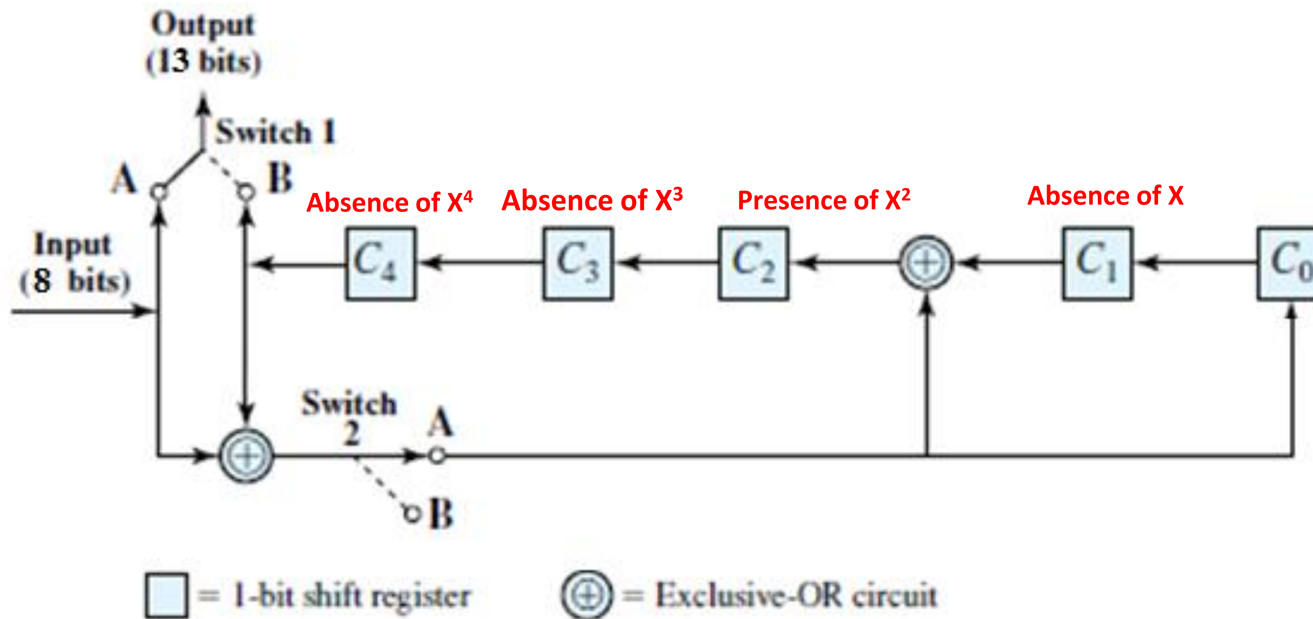


Cyclic Redundancy Check
Example 4 – Circuit Implementation

Generator Polynomial $x^5 + x^2 + 1$

The circuit is implemented as follows:

1. The register contains $n - k$ bits, equal to the length of the FCS.
2. There are up to $n - k$ XOR gates.
3. The presence or absence of a gate corresponds to the presence or absence of a term in the divisor polynomial, $P(X)$, excluding the terms 1 and X^{n-k}.



Shift-register implementation

Cyclic Redundancy Check

- Four versions of **Polynomial Division** P(X) are widely used:
- **CRC-12 = $X^{12} + X^{11} + X^3 + X^2 + X + 1$**
- **CRC-16 = $X^{16} + X^{15} + X^2 + 1$**
- **CRC-CCITT = $X^{16} + X^{12} + X^5 + 1$**
- **CRC-32 = $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$**
- **CRC-32 is used in IEEE 802 LAN standards.**

Cyclic Redundancy Check Implementation

- The CRC process can be represented by, and indeed implemented as, a dividing circuit consisting of **XOR gates** and a **shift register**.
- The **shift register** is a string of 1-bit storage devices. Each device has an output line, which indicates the value currently stored, and an input line.
- At discrete time instants, known as clock times, the value in the storage device is replaced by the value indicated by its input line.
- The entire register is clocked simultaneously, causing a 1-bit shift along the entire register. The circuit is implemented as follows:
  1. **The register contains *n-k* bits, equal to the length of the FCS.**
  2. **There are up to *n-k* XOR gates.**
  3. **The presence or absence of a gate corresponds to the presence or absence of a term in the divisor polynomial, *P(X), excluding the terms 1 and* X^{n-k}.**

Cyclic Redundancy Check Implementation

EXAMPLE 6.8 The architecture of a CRC circuit is best explained by first considering an example, which is illustrated in Figure 6.5. In this example, we use

$$\text{Data } D = 1010001101; \qquad D(X) = X^9 + X^7 + X^3 + X^2 + 1$$
$$\text{Divisor } P = 110101; \qquad P(X) = X^5 + X^4 + X^2 + 1$$

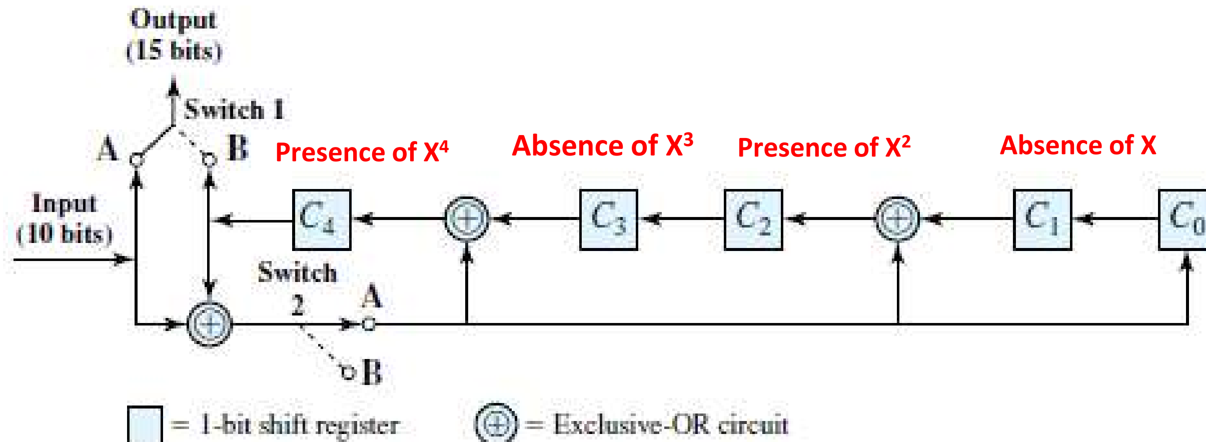
which were used earlier in the discussion.

Figure 6.5a shows the shift register implementation. The process begins with the shift register cleared (all zeros). The message, or dividend, is then entered, one bit at a time, starting with the most significant bit. Figure 6.5b is a table that shows the step-by-step operation as the input is applied one bit at a time. Each row of the table shows the values currently stored in the five shift-register elements. In addition, the row shows the values that appear at the outputs of the three XOR circuits. Finally, the row shows the value of the next input bit, which is available for the operation of the next step.

Note that the XOR operation affects C_4, C_2, and C_0 on the next shift. This is identical to the binary long division process illustrated earlier. The process continues through all the bits of the message. To produce the proper output, two switches are used. The input data bits are fed in with both switches in the A position. As a result, for the first 10 steps, the input bits are fed into the shift register and also used as output bits. After the last data bit is processed, the shift register contains the remainder (FCS) (shown shaded). As soon as the last data bit is provided to the shift register, both switches are set to the B position. This has two effects: (1) All of the XOR gates become simple pass-throughs; no bits are changed, and (2) as the shifting process continues, the 5 CRC bits are output.

At the receiver, the same logic is used. As each bit of M arrives, it is inserted into the shift register. If there have been no errors, the shift register should contain the bit pattern for R at the conclusion of M. The transmitted bits of R now begin to arrive, and the effect is to zero out the register so that, at the conclusion of reception, the register contains all 0s.

Circuit with Shift Registers for Dividing by the Polynomial $X^5 + X^4 + X^2 + 1$



(a) Shift-register implementation

	C_4	C_3	C_2	C_1	C_0	$C_4 \oplus C_3 \oplus I$	$C_4 \oplus C_1 \oplus I$	$C_4 \oplus I$	I = input
Initial	0	0	0	0	0	1	1	1	1
Step 1	1	0	1	0	1	1	1	1	0
Step 2	1	1	1	1	1	1	1	0	1
Step 3	1	1	1	1	0	0	0	1	0
Step 4	0	1	0	0	1	1	0	0	0
Step 5	1	0	0	1	0	1	0	1	0
Step 6	1	0	0	0	1	0	0	0	1
Step 7	0	0	0	1	0	1	0	1	1
Step 8	1	0	0	0	1	1	1	1	0
Step 9	1	0	1	1	1	0	1	0	1
Step 10	0	1	1	1	0				

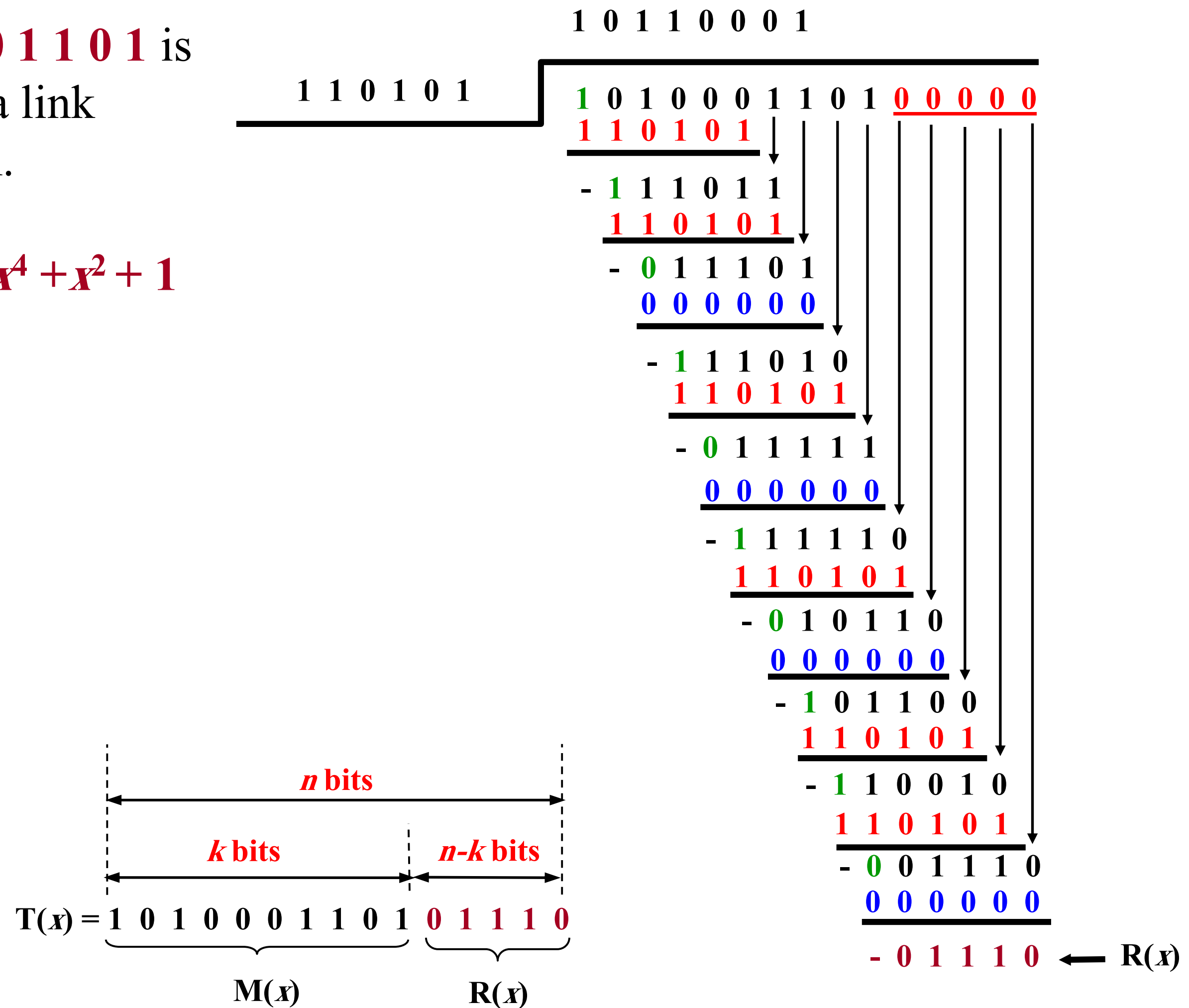
Message to be sent (the input bits, Initial through Step 9)

(b) Example with input of 1010001101

Circuit with Shift Registers for Dividing by the Polynomial $X^5 + X^4 + X^2 + 1$

An 10-bit message **1 0 1 0 0 0 1 1 0 1** is to be transmitted across a data link using CRC for error detection.

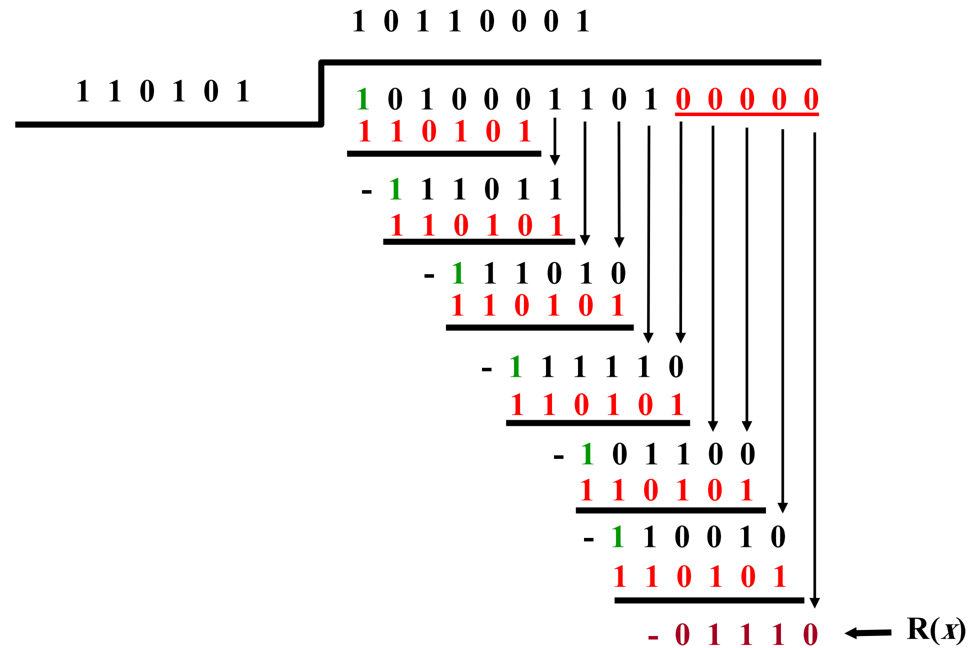
A generator polynomial $x^5 + x^4 + x^2 + 1$ is to be used.

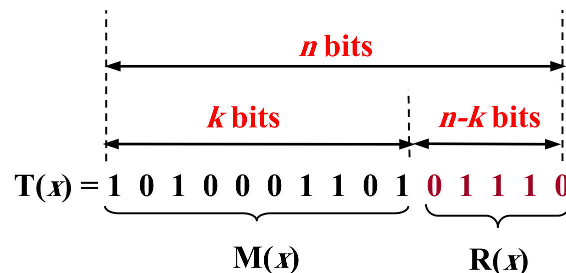


Circuit with Shift Registers for Dividing by the Polynomial $X^5 + X^4 + X^2 + 1$

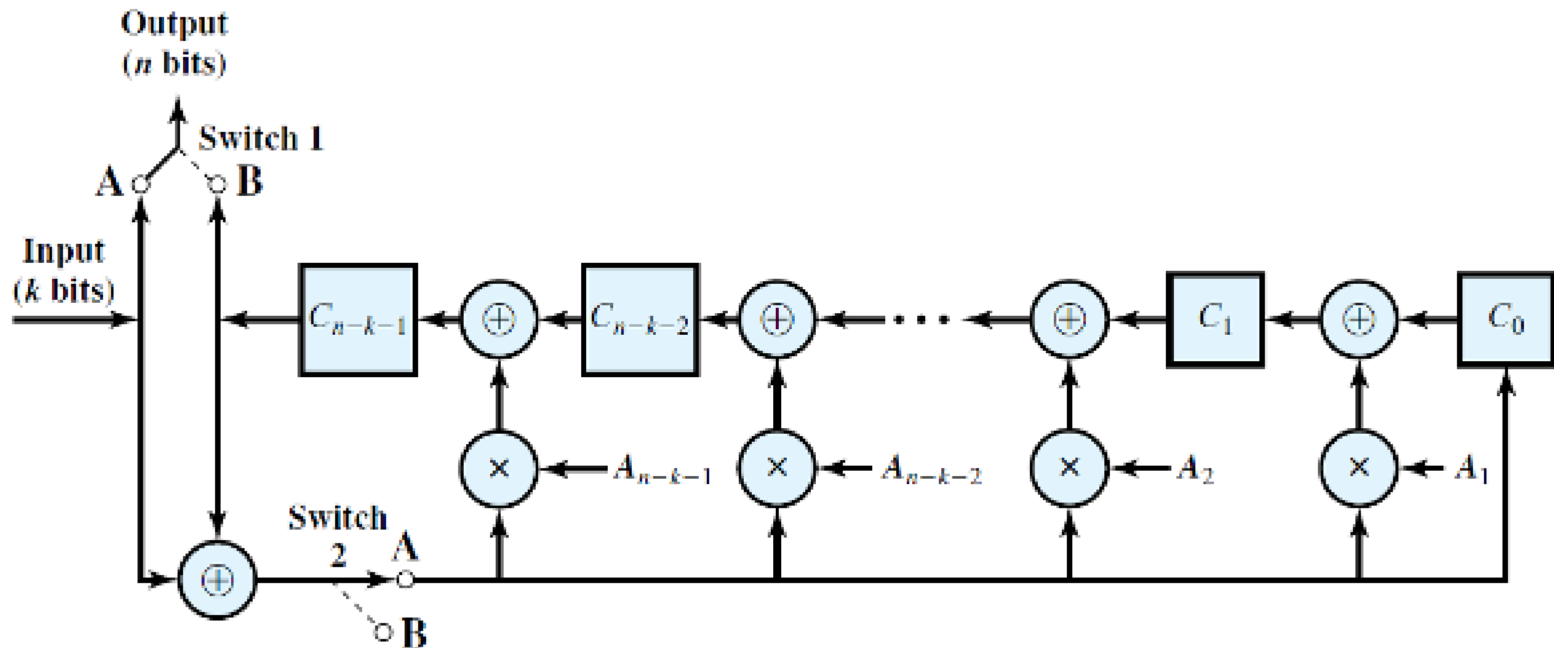
An 10-bit message **1 0 1 0 0 0 1 1 0 1** is to be transmitted across a data link using CRC for error detection.

A generator polynomial $x^5 + x^4 + x^2 + 1$ is to be used.





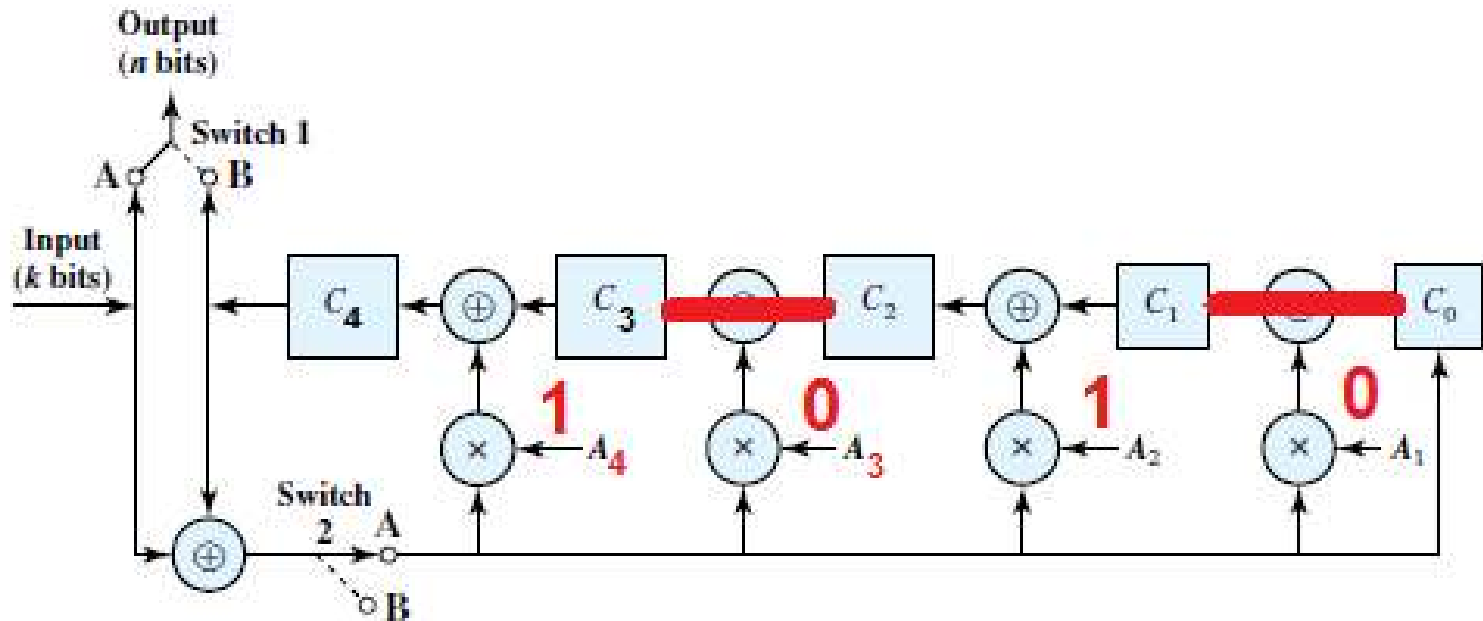
Cyclic Redundancy Check General Implementation



General CRC Architecture to Implement Divisor $(1 + A_1X + A_2X^2 + \cdots + A_{n-1}X^{n-k-1} + X^{n-k})$

Cyclic Redundancy Check General Implementation - Example

Generator Polynomial $x^5 + x^4 + x^2 + 1$



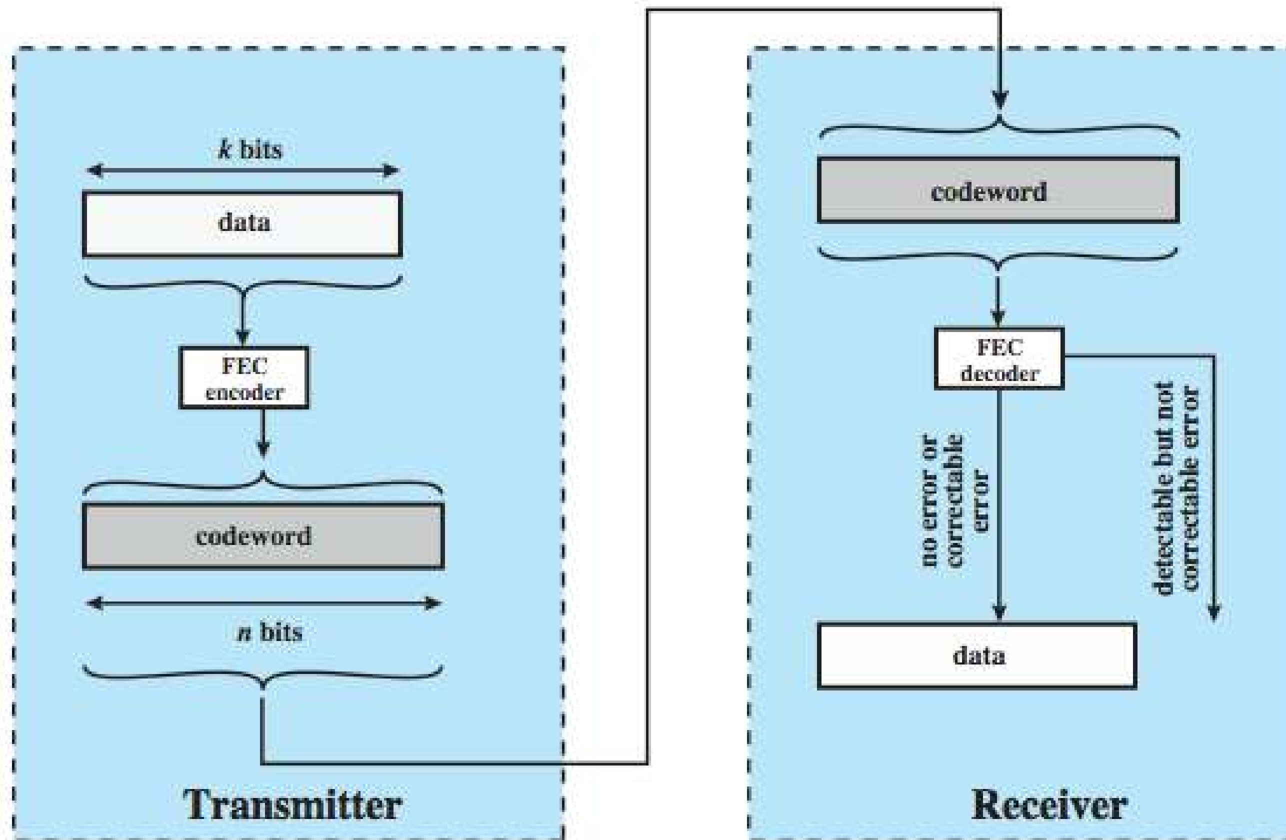
Error Correction

Error Correction

Error Correction

- Error detection is not always sufficient
  - **wireless links have many errors**
  - **satellite links have long propagation delay**
  - in both cases, **retransmissions are expensive**
- **Error-correcting codes**
  - add redundant bits to transmitted message
  - also known as forward error correction (FEC)

Error Correction Process



Error Correction – FEC Decoder

- This block is passed through an **FEC decoder**, with one of four possible outcomes:
- **1.** If **there are no bit errors**, the input to the FEC decoder is identical to the original codeword, and the decoder produces the original data block as output.
- **2.** For certain error patterns, it is possible for the **decoder to detect and correct those errors.** Thus, even though the incoming data block differs from the transmitted codeword, the FEC decoder is able to map this block into the original data block.
- **3.** For certain error patterns, the decoder can **detect but not correct the errors.** In this case, the decode simply reports an uncorrectable error.
- **4.** For certain, typically rare, error patterns, **the decoder does not detect that any errors have occurred** and maps the incoming n-bit data block into a k-bit block that differs from the original k-bit block.

How Error Correction Works

- adds redundancy to transmitted message
- **can deduce original despite some errors**
- eg. block error correction code
  - **map *k* bit input onto an *n* bit codeword**
  - **each distinctly different**
  - **if get error assume codeword sent was closest to that received**
- means have reduced effective data rate

Codewords

- **n bits total: k data bits, (n–k) redundant bits**
- 2^n possible codewords
- 2^k valid codewords represent data
- The ratio of redundant bits to data bits, **(n-k)/k** is called the **redundancy** of the code
- The ratio of data bits to total bits, ***k/n***, is called the **code rate**.
- The **code rate** is a measure of how much additional bandwidth is required to carry data at the same data rate as without the code.
- **For example, a code rate of 2/5 requires 2.5 times the capacity of an uncoded system. For example, if the data rate input to the encoder is 1 Mbps, then the output from the encoder must be at a rate of 2.5 Mbps to keep up.**

Codewords Example:

EXAMPLE 6.9

For $k = 2$ and $n = 5$, we can make the following assignment:

Data Block	Codeword
00	00000
01	00111
10	11001
11	11110

d =1
d =2
d =4
d =3

Now, suppose that a codeword block is received with the bit pattern 00100. This is not a valid codeword, and so the receiver has detected an error. Can the error be corrected? We cannot be sure which data block was sent because 1, 2, 3, 4, or even all 5 of the bits that were transmitted may have been corrupted by noise. However, notice that it would require only a single bit change to transform the valid codeword 00000 into 00100. It would take two bit changes to transform 00111 to 00100, three bit changes to transform 11110 to 00100, and it would take four bit changes to transform 11001 into 00100. Thus, we can deduce that the most likely codeword that was sent was 00000 and that therefore the desired data block is 00. This is error correction. In terms of Hamming distances, we have

$$d(00000, 00100) = 1; \quad d(00111, 00100) = 2;$$
$$d(11001, 00100) = 4; \quad d(11110, 00100) = 3$$

So the rule we would like to impose is that if an invalid codeword is received, then the valid codeword that is closest to it (minimum distance) is selected. This will only work if there is a unique valid codeword at a minimum distance from each invalid codeword.

Codewords Example:

For our example, it is not true that for every invalid codeword there is one and only one valid codeword at a minimum distance. There are $2^5 = 32$ possible codewords of which 4 are valid, leaving 28 invalid codewords. For the invalid codewords, we have the following:

Invalid Codeword	Minimum Distance	Valid Codeword	Invalid Codeword	Minimum Distance	Valid Codeword
00001	1	00000	10000	1	00000
00010	1	00000	10001	1	11001
00011	1	00111	10010	2	00000 or 11110
00100	1	00000	10011	2	00111 or 11001
00101	1	00111	10100	2	00000 or 11110
00110	1	00111	10101	2	00111 or 11001
01000	1	00000	10110	1	11110
01001	1	11001	10111	1	00111
01010	2	00000 or 11110	11000	1	11001
01011	2	00111 or 11001	11010	1	11110
01100	2	00000 or 11110	11011	1	11001
01101	2	00111 or 11001	11100	1	11110
01110	1	11110	11101	1	11001
01111	1	00111	11111	1	11110

Codewords Example:

There are eight cases in which an invalid codeword is at a distance 2 from two different valid codewords. Thus, if one such invalid codeword is received, an error in 2 bits could have caused it and the receiver has no way to choose between the two alternatives. An error is detected but cannot be corrected. However, in every case in which a single bit error occurs, the resulting codeword is of distance 1 from only one valid codeword and the decision can be made. This code is therefore capable of correcting all single-bit errors but cannot correct double bit errors. Another way to see this is to look at the pairwise distances between valid codewords:

$$d(00000, 00111) = 3; \quad d(00000, 11001) = 3; \quad d(00000, 11110) = 4;$$
$$d(00111, 11001) = 4; \quad d(00111, 11110) = 3; \quad d(11001, 11110) = 3;$$

The minimum distance between valid codewords is 3. Therefore, a single bit error will result in an invalid codeword that is a distance 1 from the original valid codeword but a distance at least 2 from all other valid codewords. As a result, the code can always correct a single-bit error. Note that the code also will always detect a double-bit error.

Hamming Distance

- Number of different bits between two codewords
- Calculated using bitwise XOR, counting 1s

- Example
  - $v_1 = 011011, \quad v_2 = 110001$
  - $v_1 \oplus v_2 = 101010, \; d(v_1, v_2) = 3$

- Minimum distance
  - for code consisting of $w_1, w_2, \ldots, w_s, \; s = 2^n$
  - $d_{min} = \min_{i \neq j} [d(w_i, w_j)]$

Hamming Distance

- **Maximum number of errors (t) that can be detected satisfies:**

$$t = d_{\min} - 1$$

  - t single bit errors will not produce a valid codeword

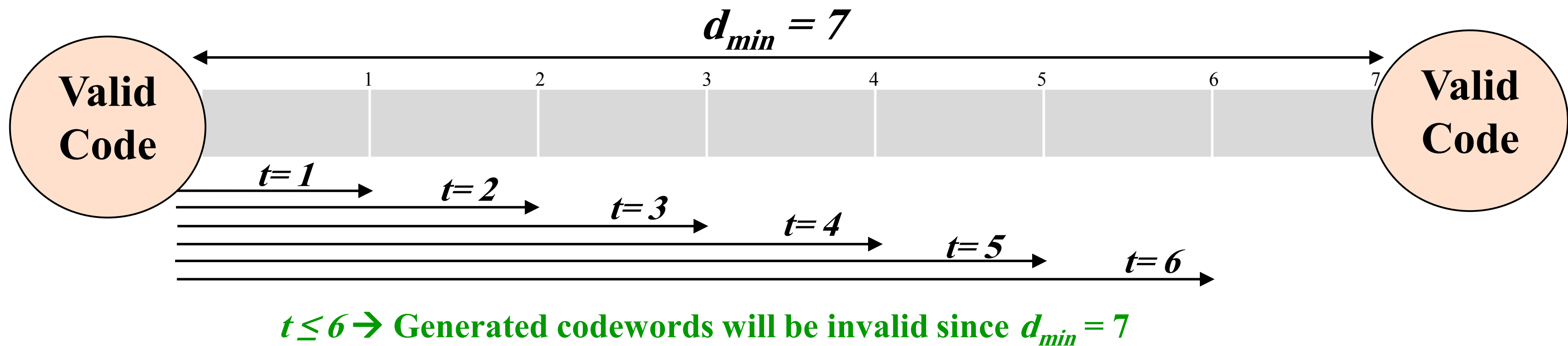
- **Maximum number of guaranteed correctable errors per codeword satisfies:**

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor$$

  - with t single bit errors, resulting codeword is still closer to one of the valid codewords

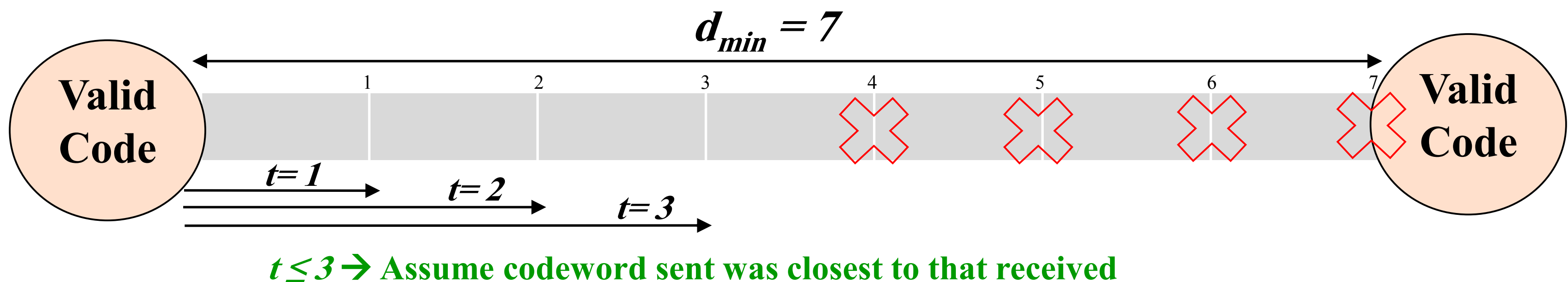
Hamming Distance - Example

- **Maximum number of errors (t) that can be detected satisfies:** $t = d_{\min} - 1$



$t \leq 6$ → Generated codewords will be invalid since $d_{min} = 7$

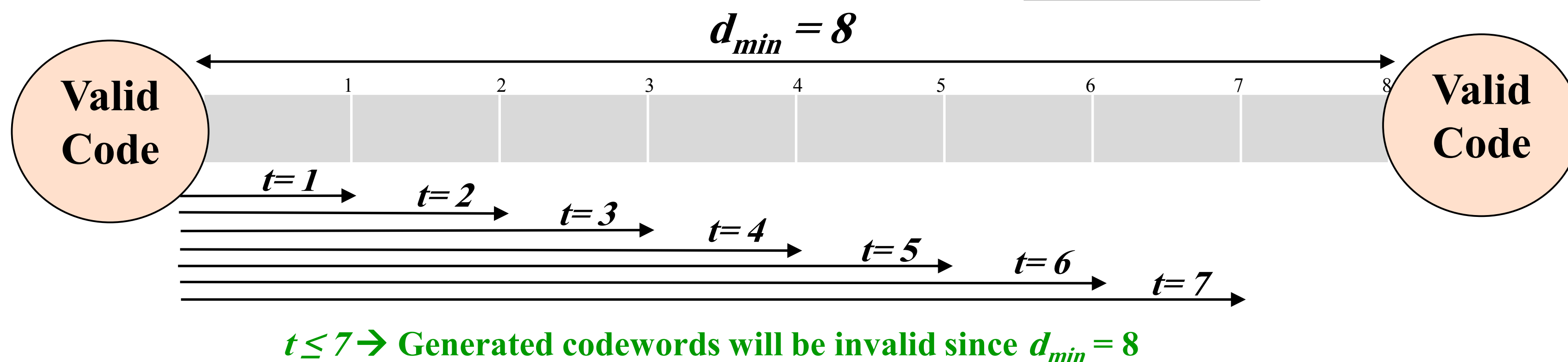
- **Maximum number of guaranteed correctable errors per codeword satisfies:** $t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor$



$t \leq 3$ → Assume codeword sent was closest to that received

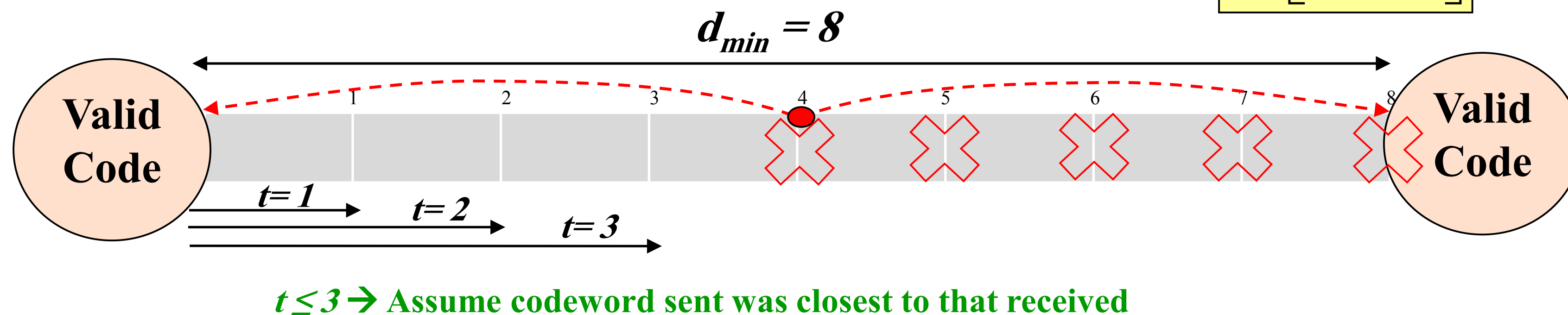
Hamming Distance - Example

⌘ Maximum number of errors (t) that can be **detected** satisfies: $t = d_{\min} - 1$



$t \leq 7$ → Generated codewords will be invalid since $d_{min} = 8$

⌘ Maximum number of guaranteed **correctable** errors per codeword satisfies: $t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor$



$t \leq 3$ → Assume codeword sent was closest to that received

Hamming Distance

⌘ The design of a block code involves a number of considerations:

1. For given values of *n* and *k,* we would like the **largest possible value of d_{min}.**

2. The code should be relatively easy to encode and decode, requiring minimal memory and processing time.

3. We would like the number of extra bits, **(*n* - *k*), to be small, to reduce bandwidth.**

4. We would like the number of extra bits , **(*n* - *k*), to be large, to reduce error rate.**

Clearly, the last two objectives are in conflict, and **tradeoffs** must be made.

Hamming Distance - Example 1

- ⌘ Valid codewords:
  000000, 000111, 111000, 111111
- ⌘ Hamming distance = 3

$$t = d_{\min} - 1$$

- ⌘ Can **detect** up to 2 errors
- ⌘ 000011, 000110, 100100, 100001
- ⌘ 001111, 000110, 110111, 101111
- ⌘ 110011, 011110, 101101, 111100

Discard

Hamming Distance - Example 2

- Valid codewords:
  0000000000, 0000011111, 1111100000, 1111111111

- Hamming distance = 5

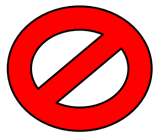
$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor$$

- Can **correct** up to 2 errors

- 0000000011, 0000000101 → 0000000000

- 0000000111, 0001111111 → 0000011111

Single Error Correction Hamming Code



- It is a code with *m* message bit and *r* parity bits that will allow all single errors to be corrected.

Transmitter:

- The bits of a codeword are numbered consecutively, starting with bit 1 at the left end.
- The bits that are **powers of** 2 (1, 2, 4, 8, 16, etc.) are **parity bits**. The rest (3, 5, 6, 7, 9, etc.) are filled up the *m* **data bits**.
- The parity bits are calculated such that we make (c1=c2=c4=…=0) according to the following equations:

$$c_1 = p_1 \oplus m_3 \oplus m_5 \oplus m_7 \oplus m_9 \oplus m_{11} \oplus \ldots$$

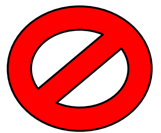
$$c_2 = p_2 \oplus m_3 \oplus m_6 \oplus m_7 \oplus m_9 \oplus m_{10} \oplus \ldots$$

$$c_4 = p_4 \oplus m_5 \oplus m_6 \oplus m_7 \oplus m_{12} \oplus m_{13} \oplus \ldots$$

$$c_8 = p_8 \oplus m_9 \oplus m_{10} \oplus m_{11} \oplus m_{12} \oplus m_{13} \oplus \ldots$$

$$\ldots\ldots$$

Single Error Correction Hamming Code



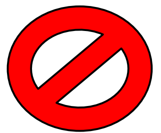
Receiver:

- The receiver calculates the check bits (c1, c2, c4, c8, etc.) using the same previous equations.

$$
\begin{aligned}
c_1 &= p_1 \oplus m_3 \oplus m_5 \oplus m_7 \oplus m_9 \oplus m_{11} \oplus \ldots \\
c_2 &= p_2 \oplus m_3 \oplus m_6 \oplus m_7 \oplus m_9 \oplus m_{10} \oplus \ldots \\
c_4 &= p_4 \oplus m_5 \oplus m_6 \oplus m_7 \oplus m_{12} \oplus m_{13} \oplus \ldots \\
c_8 &= p_8 \oplus m_9 \oplus m_{10} \oplus m_{11} \oplus m_{12} \oplus m_{13} \oplus \ldots \\
&\ldots\ldots
\end{aligned}
$$

- If (c1=c2=c4=c8=…=0), then the codeword is received correctly. Otherwise, the values of the check bits are used to determine the location of the error in the codeword.

Single Error Correction Hamming Code – Example 1:



Assuming that the transmitted character **'C'**, generate the Hamming codeword.

'C' = 11000011

P_1	P_2	m_3	P_4	m_5	m_6	m_7	P_8	m_9	m_{10}	m_{11}	m_{12}
?	?	1	?	1	0	0	?	0	0	1	1
0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100

Solution of Example 1:

$C_1 = P_1 \oplus m_3 \oplus m_5 \oplus m_7 \oplus m_9 \oplus m_{11}$

$\rightarrow C_1 = ? \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1$

$C_1 = 0 \rightarrow P_1 = 1$

$C_2 = P_2 \oplus m_3 \oplus m_6 \oplus m_7 \oplus m_{10} \oplus m_{11}$

$\rightarrow C_2 = ? \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1$

$C_2 = 0 \rightarrow P_2 = 0$

$C_4 = P_4 \oplus m_5 \oplus m_6 \oplus m_7 \oplus m_{12}$

$\rightarrow C_4 = ? \oplus 1 \oplus 0 \oplus 0 \oplus 1$

$C_4 = 0 \rightarrow P_4 = 0$

$C_8 = P_8 \oplus m_9 \oplus m_{10} \oplus m_{11} \oplus m_{12}$

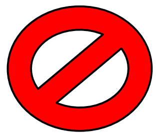
$\rightarrow C_8 = ? \oplus 0 \oplus 0 \oplus 1 \oplus 1$

$C_8 = 0 \rightarrow P_8 = 0$

→ **The transmitted Hamming codeword is:**

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

Single Error Correction Hamming Code – Example 2:



Assuming that the transmitted character **'M'**, generate the Hamming codeword.

'M' = 11010100

P_1	P_2	m_3	P_4	m_5	m_6	m_7	P_8	m_9	m_{10}	m_{11}	m_{12}
?	?	0	?	0	1	0	?	1	0	1	1
0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100

Solution of Example 2:

$C_1 = P_1 \oplus m_3 \oplus m_5 \oplus m_7 \oplus m_9 \oplus m_{11}$

→ $C_1 = ? \oplus 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1$

$C_1 = 0$ → $P_1 = 0$

$C_2 = P_2 \oplus m_3 \oplus m_6 \oplus m_7 \oplus m_{10} \oplus m_{11}$

→ $C_2 = ? \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1$

$C_2 = 0$ → $P_2 = 0$

$C_4 = P_4 \oplus m_5 \oplus m_6 \oplus m_7 \oplus m_{12}$

→ $C_4 = ? \oplus 0 \oplus 1 \oplus 0 \oplus 1$

$C_4 = 0$ → $P_4 = 0$

$C_8 = P_8 \oplus m_9 \oplus m_{10} \oplus m_{11} \oplus m_{12}$

→ $C_8 = ? \oplus 1 \oplus 0 \oplus 1 \oplus 1$

$C_8 = 0$ → $P_8 = 1$

→ **The transmitted Hamming codeword is:**

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

Single Error Correction Hamming Code – Example 3:

The received Hamming codeword is:

P_1	P_2	m_3	P_4	m_5	m_6	m_7	P_8	m_9	m_{10}	m_{11}	m_{12}
1	0	1	0	0	0	1	1	0	1	1	1
0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100

Deduce the correct character from the above codeword after correction any error, if any.

Conclusion:

→ **The codeword is correct**

→ **The codeword is 1 0 1 0 0 0 1 1 0 1 1 1**

→ **The message is 1 1 1 0 1 0 0 1**

→ **The EBCDIC character is *Z***

Solution of Example 3:

$C_1 = P_1 \oplus m_3 \oplus m_5 \oplus m_7 \oplus m_9 \oplus m_{11}$

$\rightarrow C_1 = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1$

$\rightarrow C_1 = 0$

$C_2 = P_2 \oplus m_3 \oplus m_6 \oplus m_7 \oplus m_{10} \oplus m_{11}$

$\rightarrow C_2 = 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1$

$\rightarrow C_2 = 0$

$C_4 = P_4 \oplus m_5 \oplus m_6 \oplus m_7 \oplus m_{12}$

$\rightarrow C_4 = 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1$

$\rightarrow C_4 = 0$

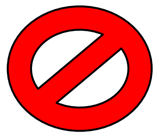
$C_8 = P_8 \oplus m_9 \oplus m_{10} \oplus m_{11} \oplus m_{12}$

$\rightarrow C_8 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1$

$\rightarrow C_8 = 0$

C_8	C_4	C_2	C_1
0	0	0	0

Single Error Correction Hamming Code – Example 4:



The received Hamming codeword is:

P_1	P_2	m_3	P_4	m_5	m_6	m_7	P_8	m_9	m_{10}	m_{11}	m_{12}
0	0	1	1	0	0	0	0	0	0	0	1
0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100

Deduce the correct character from the above codeword after correction any error, if any.

Solution of Example 4:

$C_1 = P_1 \oplus m_3 \oplus m_5 \oplus m_7 \oplus m_9 \oplus m_{11}$

→ $C_1 = 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 0$

→ $C_1 = 1$

$C_2 = P_2 \oplus m_3 \oplus m_6 \oplus m_7 \oplus m_{10} \oplus m_{11}$

→ $C_2 = 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 0$

→ $C_2 = 1$

$C_4 = P_4 \oplus m_5 \oplus m_6 \oplus m_7 \oplus m_{12}$

→ $C_4 = 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1$

→ $C_4 = 0$

$C_8 = P_8 \oplus m_9 \oplus m_{10} \oplus m_{11} \oplus m_{12}$

→ $C_8 = 0 \oplus 0 \oplus 0 \oplus 0 \oplus 1$

→ $C_8 = 1$

C_8	C_4	C_2	C_1
1	0	1	1

Conclusion:

→ m_{11} is incorrect

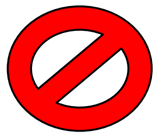
→ m_{11} must be 1

→ The codeword is 0 0 1 1 0 0 0 0 0 0 1 1

→ The message is 1 1 0 0 0 0 0 1

→ The EBCDIC character is A

Single Error Correction Hamming Code – Example 5:



The received Hamming codeword is:

P_1	P_2	m_3	P_4	m_5	m_6	m_7	P_8	m_9	m_{10}	m_{11}	m_{12}
0	0	1	0	1	0	1	0	1	1	1	1
0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100

Deduce the correct character from the above codeword after correction any error, if any.

Conclusion:

→ m_5 is incorrect

→ m_5 must be 0

→ The codeword is 0 0 1 0 0 0 1 0 1 1 1 1

→ The message is 1 1 1 1 1 0 0 1

→ The EBCDIC character is 9

Solution of Example 5:

$C_1 = P_1 \oplus m_3 \oplus m_5 \oplus m_7 \oplus m_9 \oplus m_{11}$

→ $C_1 = 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1$

→ $C_1 = 1$

$C_2 = P_2 \oplus m_3 \oplus m_6 \oplus m_7 \oplus m_{10} \oplus m_{11}$

→ $C_2 = 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1$

→ $C_2 = 0$

$C_4 = P_4 \oplus m_5 \oplus m_6 \oplus m_7 \oplus m_{12}$

→ $C_4 = 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1$

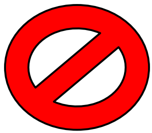
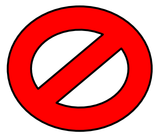
→ $C_4 = 1$

$C_8 = P_8 \oplus m_9 \oplus m_{10} \oplus m_{11} \oplus m_{12}$

→ $C_8 = 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1$

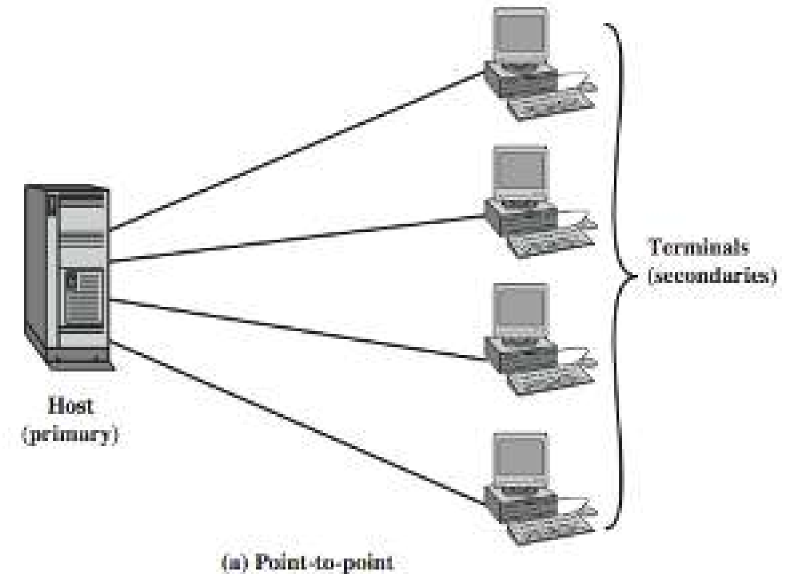
→ $C_8 = 0$

C_8	C_4	C_2	C_1
0	1	0	1

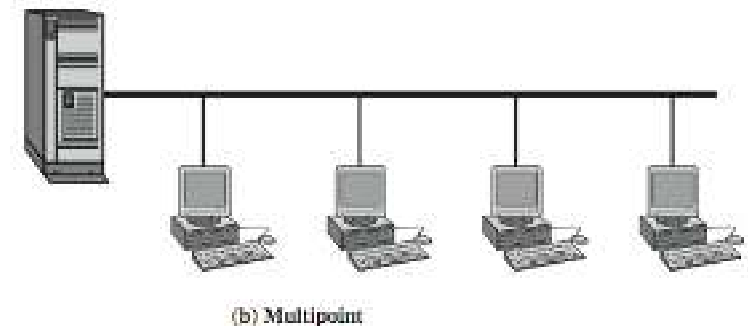



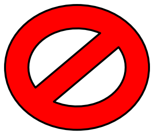
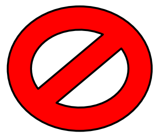
Line Configuration - Topology

- physical arrangement of stations on medium
  - point to point - two stations
    - such as between two routers / computers
  - multi point - multiple stations
    - traditionally mainframe computer and terminals



(a) Point-to-point


(b) Multipoint

Line Configuration - Duplex

- classify data exchange as half or full duplex
- half duplex (two-way alternate)
  - only one station may transmit at a time
  - requires one data path
- full duplex (two-way simultaneous)
  - simultaneous transmission and reception between two stations
  - requires two data paths
    - separate media or frequencies used for each direction

Summary

- asynchronous verses synchronous transmission
- error detection and correction
- line configuration issues