CAN 1011: Data Communication

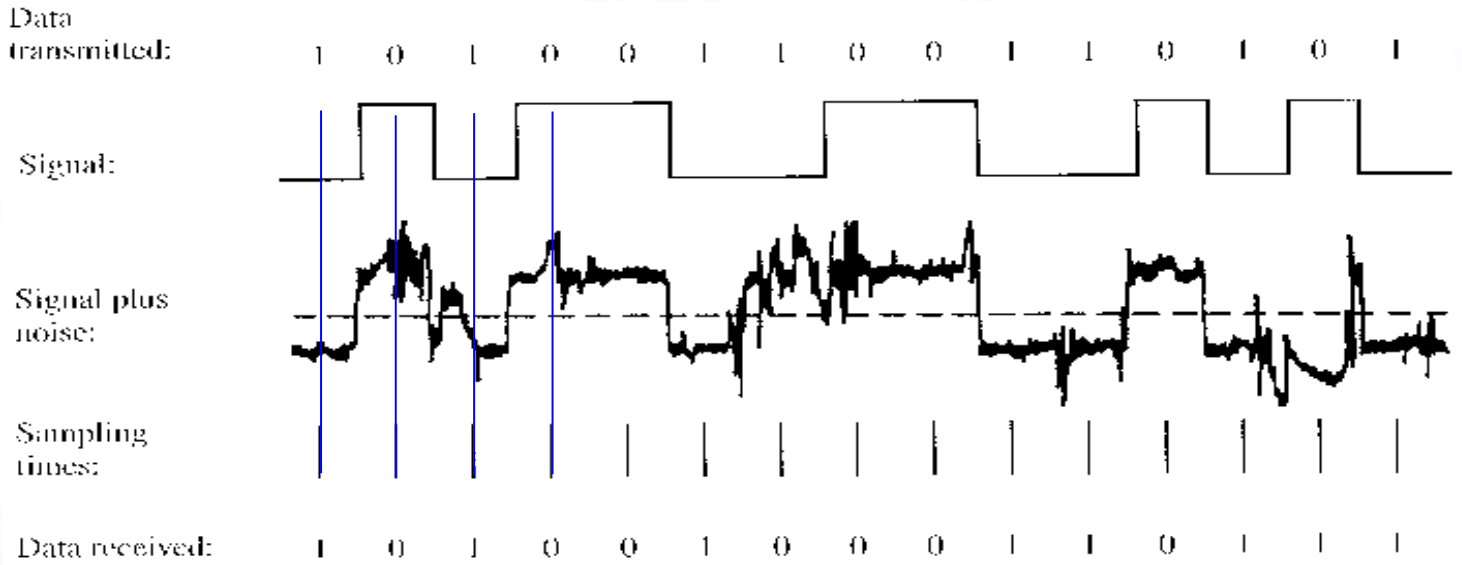
- Digital Data Communication Techniques

Contents

- Digital Data Communications Techniques
  - Asynchronous and Synchronous Transmission
  - Types of Errors
  - Error Detection
    - Parity Checks
    - Cyclic Redundancy Checks

Asynchronous and Synchronous Transmission:

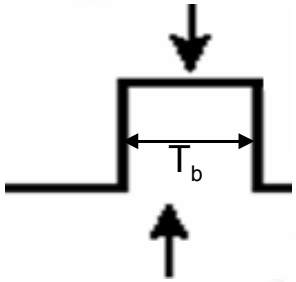
- To communicate meaningful data serially between TX and RX, signal timing should be the same at both
- Timing considerations include:
  - Rate,
  - Duration,
  - Spacing,
  - Etc.



- We need to achieve some synchronism between RX & TX
- Two ways to achieve this:
  - Asynchronous Transmission
  - Synchronous Transmission

- RX needs to sample the received data at mid-points
- So it needs to establish:
  - Bit arrival time
  - Bit duration

Need for RX and TX Synchronization:



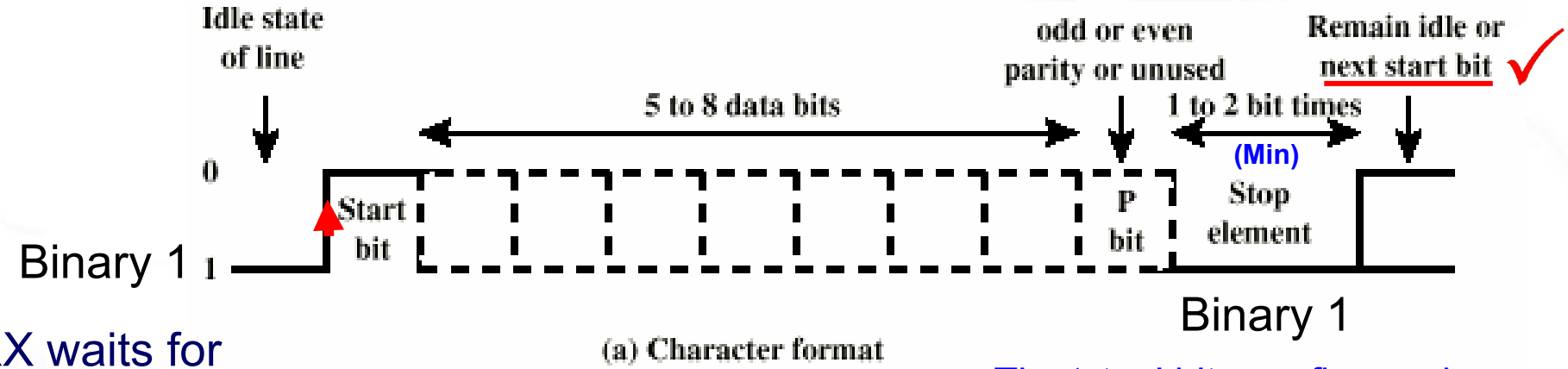
- Clock drift (example):
  - If the receiver clock drifts by 1% every bit sample time,
  - Total drift after 50 bit intervals = 50 X 0.01 = 0.5 T_b
  - i.e. instead of sampling at the middle of the bit, the receiver will sample bit # 50 at the edge of the bit – Bit 51 will be wrongly sampled
  - RX and TX clocks become out-of-synch ⇒ Communication Error!
- In general, No. of correctly sampled bits = 0.5 T_b/(n/100)T_b = 50/n, where n is the % timing error between TX and RX clocks
- Two approaches for correct reception:
  - Send only a few bits (e.g. a character) at a time (that RX can sample correctly before losing sync) → Asynchronous Transmission
  - Keep receiver clock properly synchronized with the transmitter clock all the time → send as many bits as you want…Synchronous Transmission

Asynchronous Transmission: Character-Level Synchronization

- Avoids the timing problem by NOT sending long, uninterrupted streams of bits
- So data is transmitted only one short character at a time: (so drift will not be a serious problem). Characters consists of:
  - A distinct start bit,
  - Say 5 to 8 data/parity bits
  - A distinct stop bit
- Character is *delimited* (at start & end) by known signal elements: start bit – stop element
- Sync needs to be maintained only for the short duration of the character (easier to achieve, allows some clock drift)
- The receiver has a new opportunity to resynchronize at the beginning of next character (Start bit)

$\Rightarrow$ i.e. Timing errors do not accumulate from character to character

Asynchronous Transmission

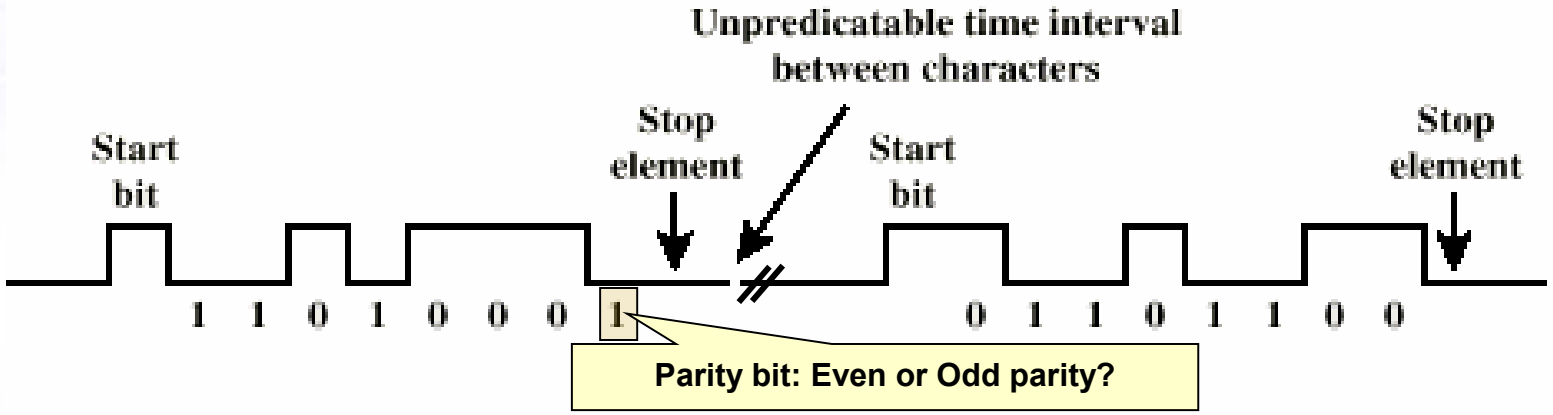


(a) Character format

RX waits for A character following the end of the previous character

RX "knows" how many bits To expect in a character, and keeps counting them following the 'start' bit

- The 'stop' bits confirm end of character → otherwise: Framing Error
- Stop bits continue (idling) till next character

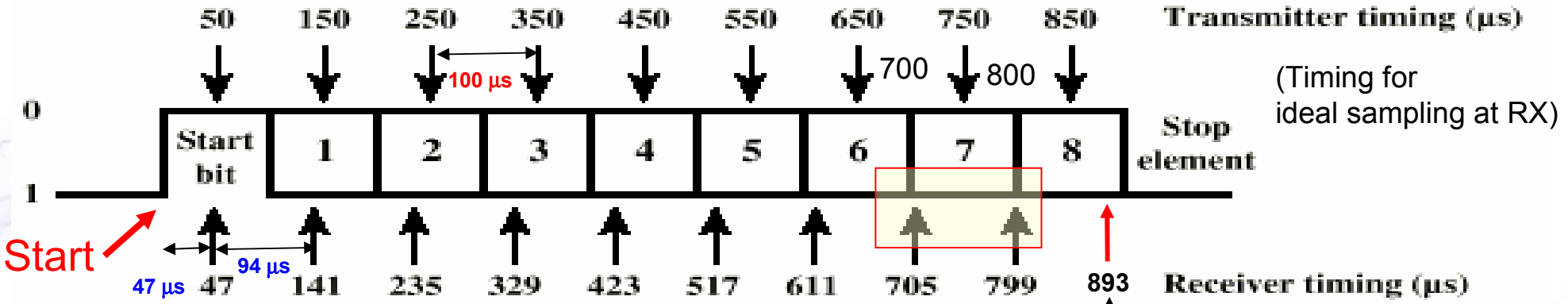


(b) 8-bit asynchronous character stream

Asynchronous Transmission

Errors due to lack of sync for an 8-bit system

- Let data rate = baud rate = 10 kbps
- Bit interval = signal element width = 1/10k = 100 µs
- Clock Drift: Let RX's clock is *faster* than TX by 6% (10.6 KHz)
  (So RX *thinks* that the bit interval is 1/10.6 KHz = 94 µs)
- RX checks mid-bit data: after 47 µs and then at 94 µs intervals
- Data bit 8 is wrongly sampled within bit 7 (bit 7 is read twice!)
- Actual data bit 8 is missed and is seen by RX as a stop bit!
- A framing error occurs if bit 8 is 1 or 0?



Half the bit interval from the 'start' rising edge

(c) Effect of timing error

8th data bit is taken as the stop bit!- If 1 error not Detected!- if 0 framing error occurs

Asynchronous Transmission: Efficiency

What are we paying to compensate for lack of proper TX to RX synchronization?

- Each Char uses 1 start bit & 2 stop bits: (3 non-data bits)

  with 8-bit data and no parity:

  $\Rightarrow$ Efficiency = Useful Data / Total Data = 8/(8+3) = 72%

  $\Rightarrow$ Overhead = Non Data bits / Total Data = 3/(8+3) = 28%

Asynchronous Transmission - Pros & Cons

Pros:

- Simple
- Cheap
- Good for data with large gaps in between (e.g. terminal to a computer like keyboard or mouse)

Cons:

- Overhead of 2 or 3 bits per short character (~20%)
- Limit on character size
- Timing errors accumulate within large characters

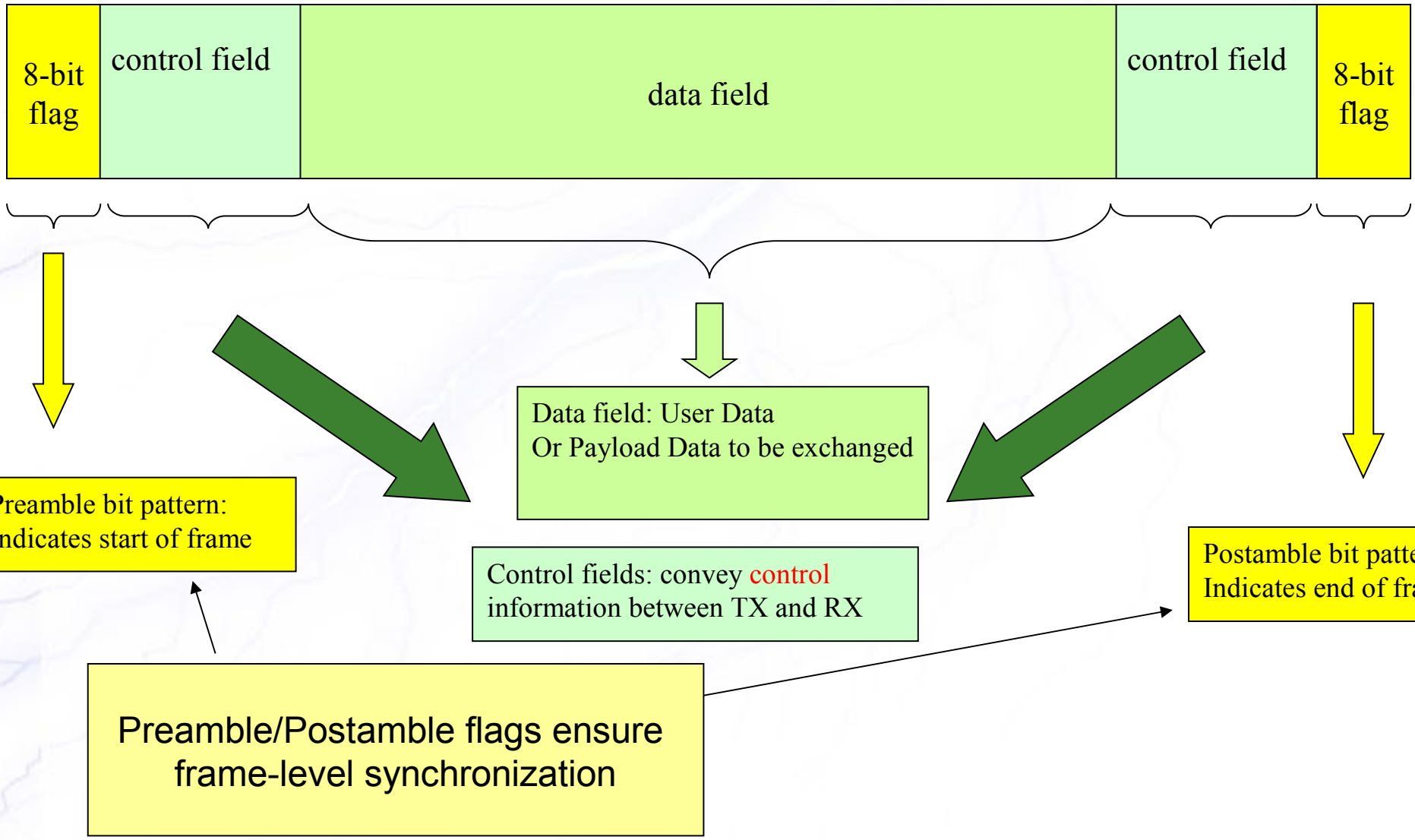
Synchronous Transmission: Bit-Synchronization

- Allows transmission of large blocks of data (frames)
- Need both bit-level and frame-level synchronization
- Bit-level synchronization (to prevent timing drift):
  - Use a separate clock line between TX and RX
    - OK over short distances
    - Subject to transmission impairments over long distances
  - Or: Embed clock signal in data using:
    - Self-clocking codes, e.g. Manchester or Differential Manchester encoding
    - Or carrier frequency for analog signals (shift keying)
- Frame-level synchronization:

  Preamble & Postamble flags

Synchronous Frame Format

Typical Frame Structure



Synchronous Transmission: Efficiency

- Example: HDLC data link protocol uses a total of 48 bits for control, preamble, and postamble fields per frame:

  With a data block consisting of 1000 characters (8-bits each),

  $\Rightarrow$ Efficiency = 8000/(8000+48) = 99.4%

  $\Rightarrow$ Overhead = 48/8048 = 0.6% (compare 20% for async)

- Note higher efficiency and lower overhead compared to asynchronous transmission

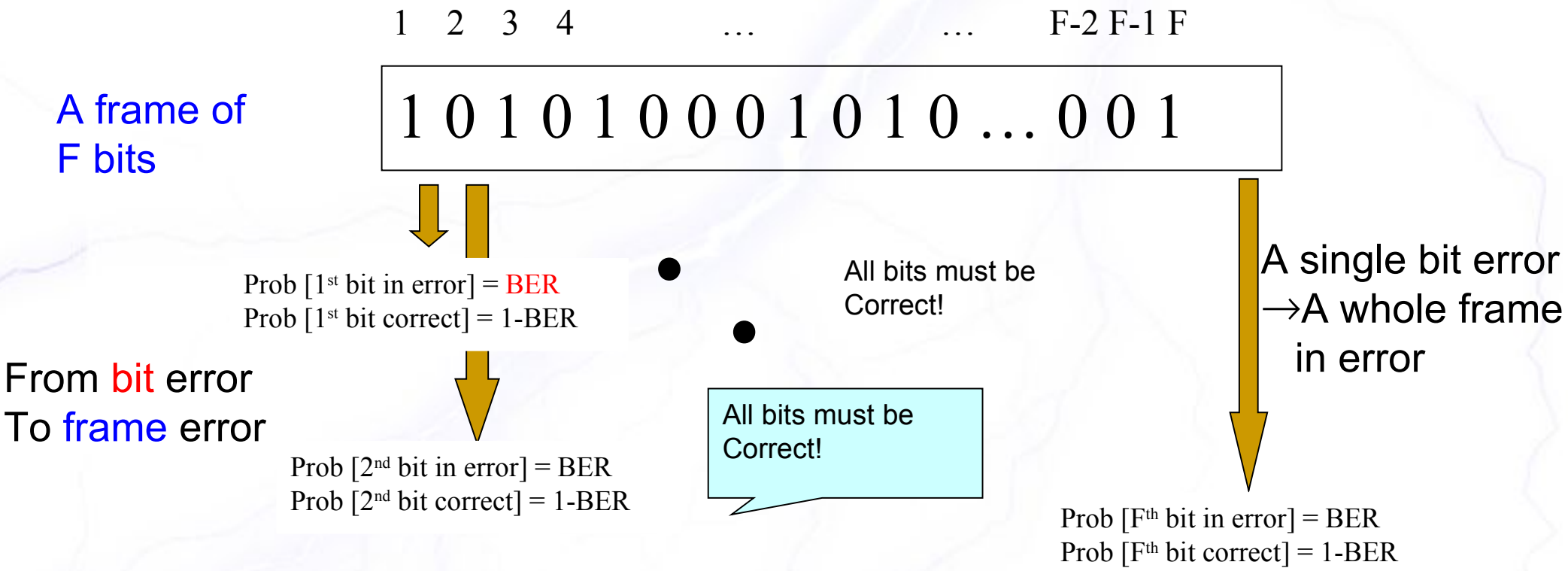
Errors in Digital Transmission

- Error occurs when a bit is altered between transmission and reception ($0 \Rightarrow 1$ or $1 \Rightarrow 0$)
- Two types of errors:
  - Single bit errors
    - One bit altered
    - Isolated incidence, adjacent bits not affected
    - Typically caused by white noise
  - Burst errors
    - Contiguous sequence of bits in which first, last, and any number of intermediate bits are in error
    - Caused by impulse noise or fading (in wireless communication)
    - More common, and more difficult to handle
    - Effect is greater at higher data rates
- What to do about these errors?:
  - Do nothing? (Is this acceptable?)
  - Detect them (at least, so we can ask TX to retransmit!)
    - and Correct them (if we can)

- Will show that - Without error detection/correction - rate of erroneous **frames** received would be unacceptably large!

Frame Error Rate

- We know about the bit error rate (BER)
- But we send data as large frames → We are more interested in frame error rate (FER)
- How does BER affect FER?



- Hence, for a frame of F bits,
  Prob [frame is correct] = $(1\text{-BER})^F$: Decreases with increasing BER & F (bad)
- Prob [frame is erroneous] = $1 - (1\text{-BER})^F$ = Frame Error Rate (FER) Increases with increasing BER & F (bad)

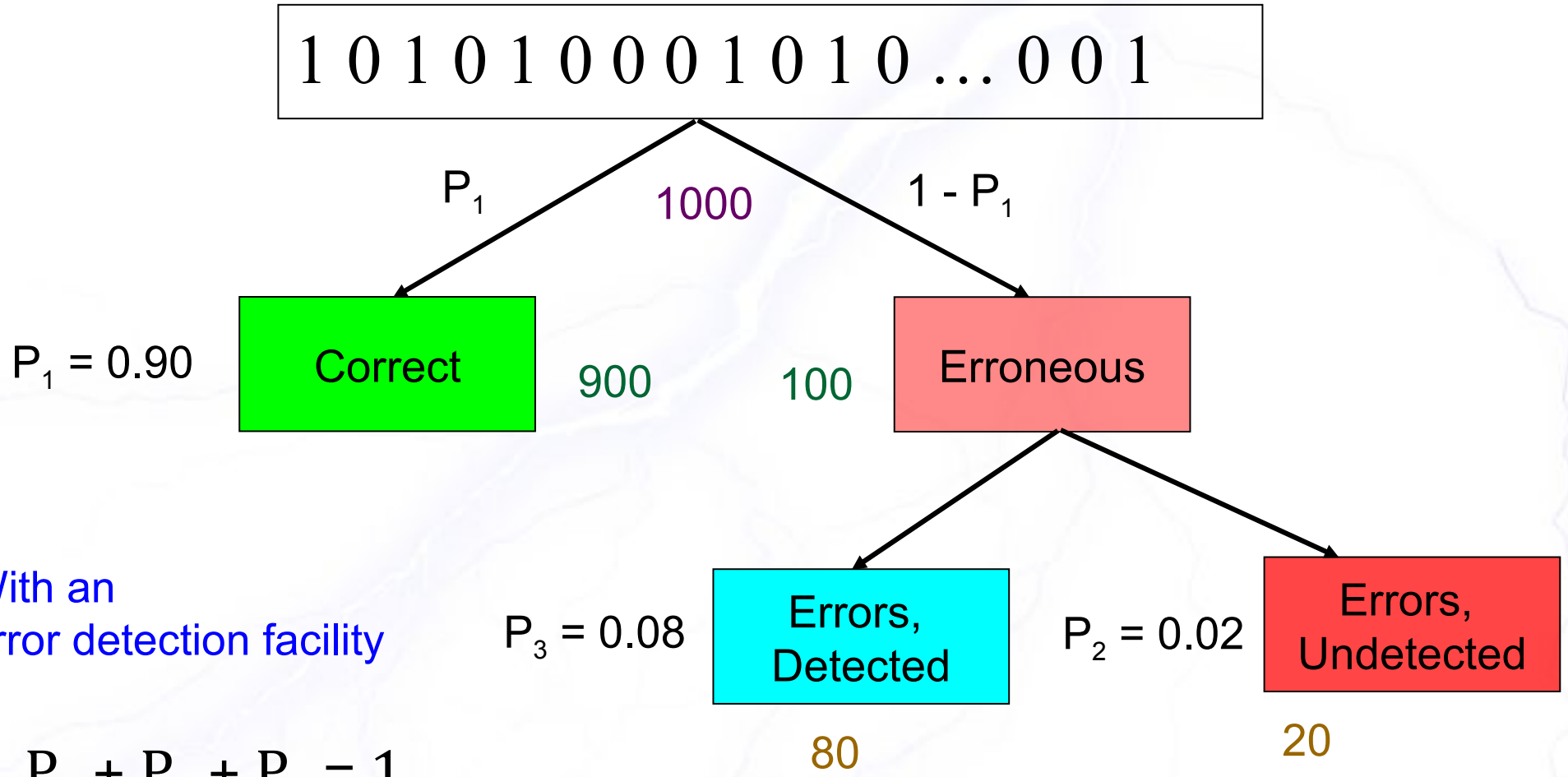
Motivation for Error Detection & Correction: Example

- ISDN specifies a BER = 10^{-6} for a 64kbps channel
- Let frame size F = 1000 bits
- What is the FER?
  - FER = $1 - (1 - BER)^F = 1 - (0.999999)^{1000} = 10^{-3}$
- Assume a continuously used channel…
  - How many erroneous frames in one day ?
- Number of frames sent/day = (64,000/1000 frames/s) × (24 × 3600 s/day)

  = 5.5296×10^6 Frames/day
- Number of erroneous frames/day:
  - = $5.5296 \times 10^6 \times 10^{-3} = 5.5296 \times 10^3$
- Typical requirement: Maximum of 1 erroneous frame /day!
- i.e. frame error rate is too high to be tolerated!

  ⇒ We definitely need error detection & correction!

Frame Error Probabilities:



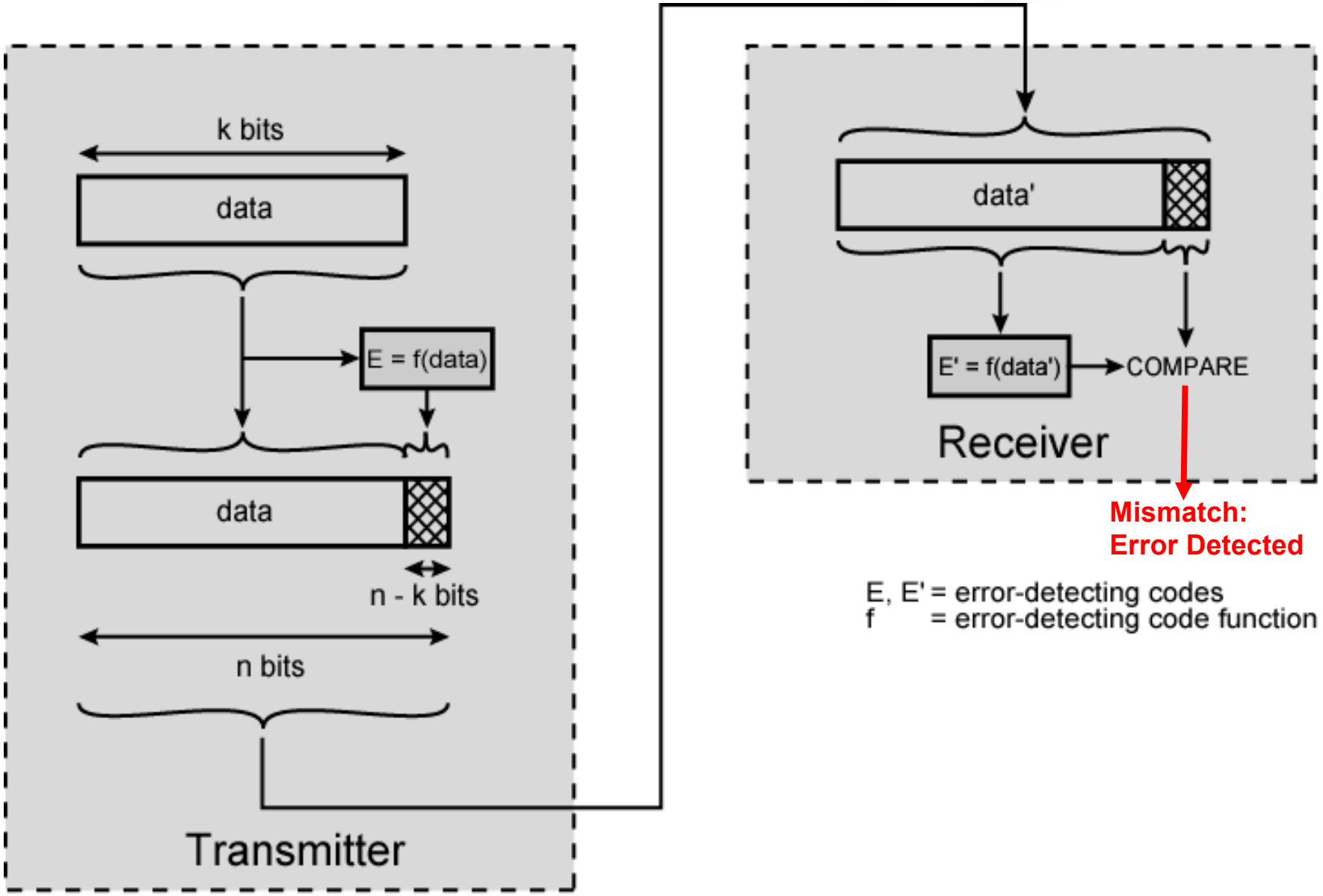
- $P_1 + P_2 + P_3 = 1$
- Without error detection facility: $P_3 = 0$, and:
  $P_2 = 1 - P_1$ (all errors are undetected)

Error Detection Techniques

- Two main error detection techniques:
  - Parity Check
  - Cyclic Redundancy Check (CRC)

- Both techniques use *additional bits* that are appended to the "payload data" by the transmitter for the purpose of error detection at the receiver

Error Detection: Implementation

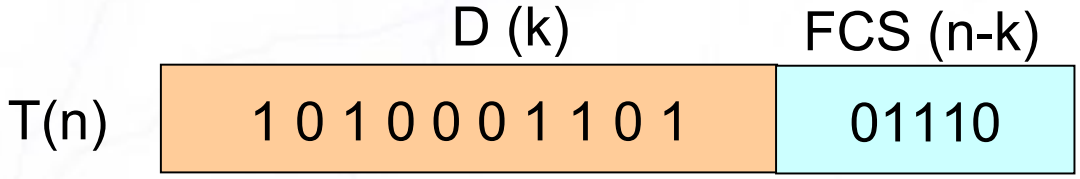


Parity Check

- Simplest error-detection scheme
- Appending one extra bit:
  - Even Parity: Will append "1" such that the total number of 1's is even
  - Odd Parity: Will append "1" such that the total number of 1's is odd
- Example: If an even-parity is used, RX will check if the total number of 1's is even
  - If it is not ⇒ error occurred
- Problem: even number of bit errors go undetected!

Cyclic Redundancy Check (CRC)

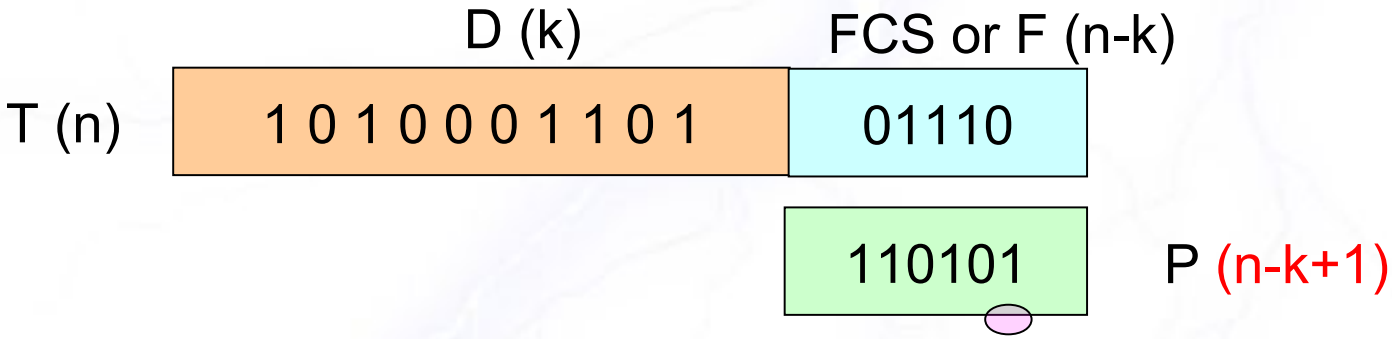
- Burst errors will most likely go undetected by a simple parity check scheme
- Instead, we use a more elaborate technique: Cyclic Redundancy Check (CRC)
- CRC appends **redundant** bits to the frame trailer called Frame Check Sequence (FCS)
  - The FCS bits are used at RX for error detection
- In a given frame containing a total of n bits, we define:
  - k = the number of original data bits
  - (n – k) = the number of added bits in the FCS field
  - So, that the total frame length is k + (n – k) = n bits



CRC Generation

- CRC generation at TX is all about finding the FCS, given the data (D) and a divisor (P)

  that makes T exactly divisible by P (i.e. with 0 remainder)



- There are three equivalent ways to see how the CRC code is generated:
  - Modulo-2 Arithmetic Method
  - Polynomial Method (not covered)
  - Digital Logic Method (not covered)

What is F that makes T divide P exactly? i.e. with no remainder

Modulo-2 Arithmetic

- Binary arithmetic *without* carry
- Equivalent to XOR operation
- i.e.:

$$0 \pm 0 = 0;\ 1 \pm 0 = 1;\ 0 \pm 1 = 1;\ 1 \pm 1 = 0$$

- Examples:

$$\begin{array}{r} 1111 \\ +1010 \\ \hline 0101 \end{array} \qquad \begin{array}{r} 1111 \\ -1010 \\ \hline 0101 \end{array} \qquad \begin{array}{r} 11001 \\ \times\ 11 \\ \hline 11001 \\ 11001 \\ \hline 101011 \end{array}$$

Subtraction is the same as addition!

$$\begin{array}{r} 1010 \\ +1010 \\ \hline 0000 \end{array}$$

A+A = A-A = 0!

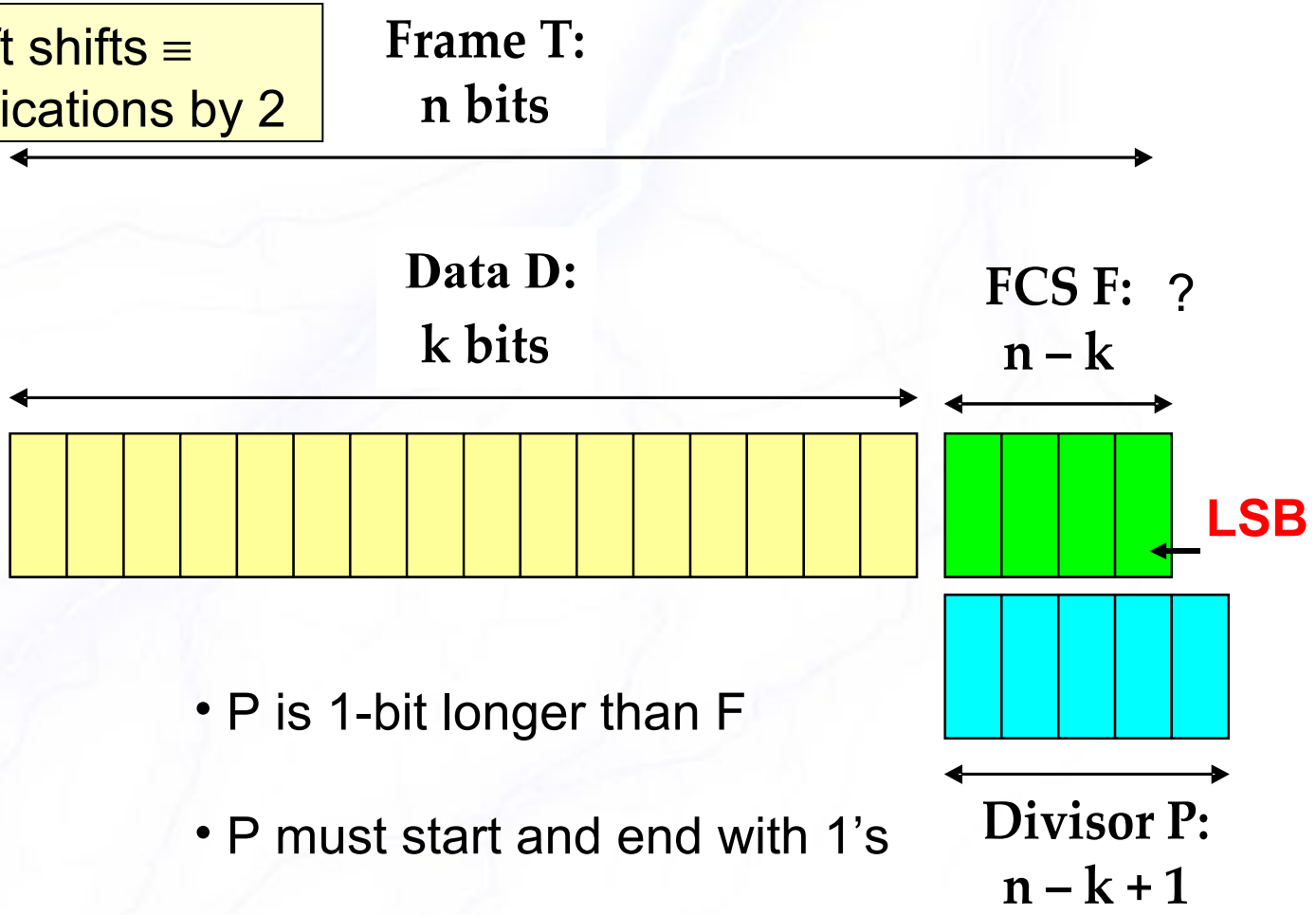
CRC Error Detection Process

- Given k-bit data (D), the TX generates an (n – k)-bit FCS field (F) such that the *total* n-bit frame (T) is exactly divisible by a predefined (n-k+1) bit divisor (P) (i.e. gives a zero remainder)
- In general, the received frame may or may not be identical to the sent frame
  - Let the received frame be (T')
  - Only in error-free transmissions that we have T' = T
- RX divides (T') by the same known divisor (P) and checks if there is any remainder
  - If division yields a remainder then the frame is erroneous
  - If the division yields zero remainder then the frame is error-free unless many erroneous bits in T' resulted in a new exact division by P (we now know what **cyclic** means!)
    - This is unlikely but possible, causing an **undetected** error!

CRC Generation

$$T = 2^{(n-k)} \times D + F$$

(n-k) left shifts ≡
(n-k) multiplications by 2



- P is 1-bit longer than F
- P must start and end with 1's

CRC Generation

- $T = 2^{(n-k)} \times D + F$, What is F that makes T divide P exactly ?
- Claim: F is the **remainder** obtained from dividing $\{2^{(n-k)} \times D\}$ by divisor P

$$\frac{2^{(n-k)}\,D}{P} = Q + \frac{F}{P} \qquad (1)$$

where Q is the quotient and F is the remainder

- If this is the correct F, T should now divide P with Zero remainder

$$\frac{T}{P} = \frac{2^{(n-k)}\,D}{P} + \frac{F}{P}, \text{ Substituting for the first term from (1)}$$

$$= Q + \frac{F}{P} + \frac{F}{P} = Q + \frac{F+F}{P}$$

$$= Q + \frac{0}{P}; \qquad (F + F = 0 \text{ in modulo 2 Arithmetic, XOR Operation})$$

It does!... T divides P exactly!

- Note: For F to be a remainder when dividing by P (in step 1), P should be 1-bit longer, that is why P is (n-k)+1 bits....

CRC Generation:
1. Modulo-2 Arithmetic Method

- At TX: CRC Generation (using the rules):
  1. Multiply: $2^{(n-k)} \times D$ (left shift by (n-k) bits)
  2. Divide: $\{2^{(n-k)} \times D\} / P$
  3. Use the resulting (n – k)-bit remainder as the FCS
- At RX: CRC Checking: RX divides the received T (i.e. T') by the known divisor (P) and checks if there is any remainder:
  - Non-zero remainder $\Rightarrow$ Error (for sure)
  - Zero Remainder $\Rightarrow$ Assume no error. You could be wrong- (undetected error) but with a small probability… see slide 41)

Example - Modulo-2 Arithmetic Method

- Given
  - D = 1 0 1 0 0 0 1 1 0 1 At the Transmitter (source) side
  - P = 1 1 0 1 0 1
- Find the FCS field
- Solution:
  - First we note that:
    - The size of the data block D is k = 10 bits
    - The size of P is (n – k + 1) = 6 bits

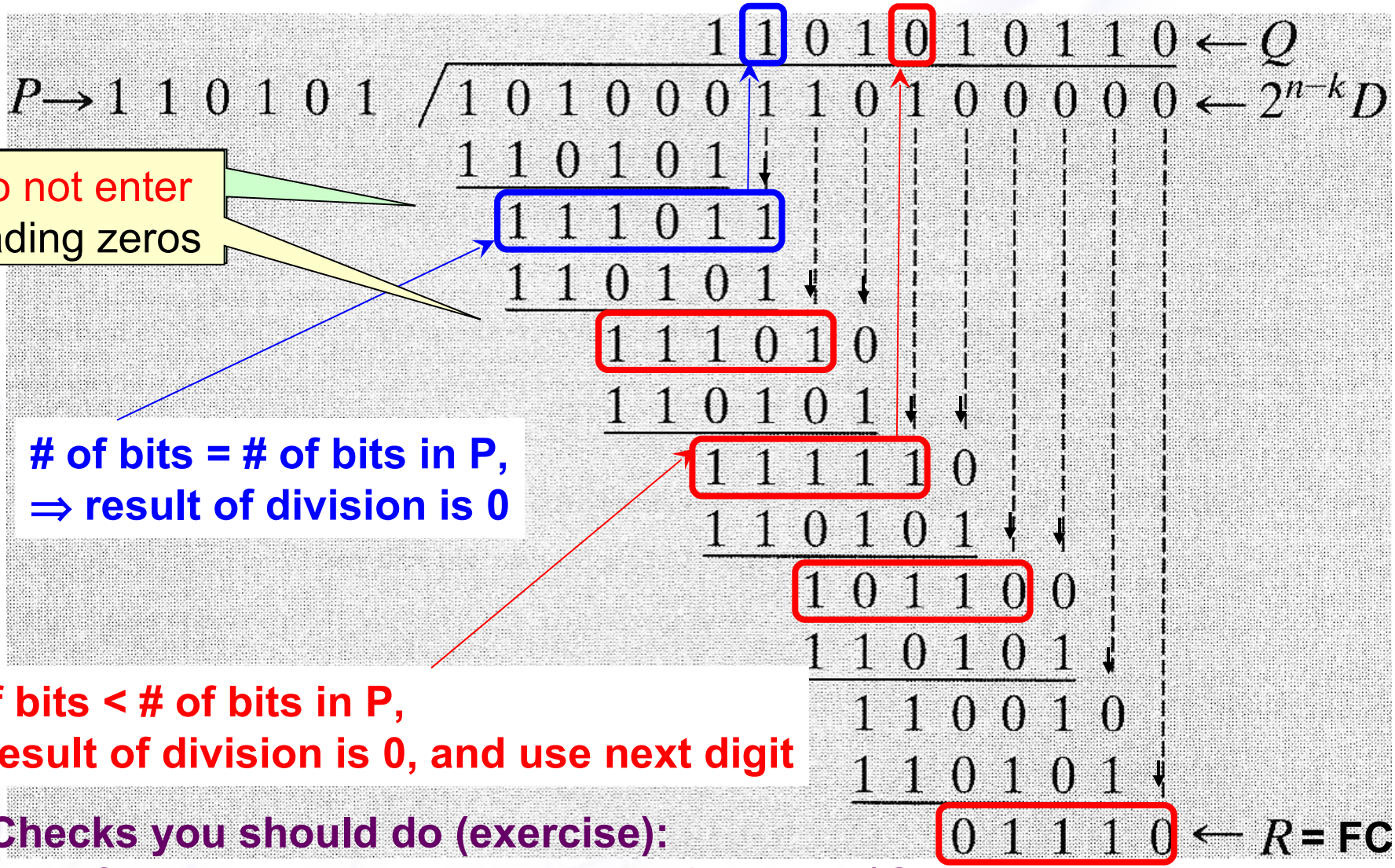
      ⇒ Hence the FCS length is n – k = 5

      ⇒ Total size of the frame T is n = 15 bits

Example - Modulo-2 Arith. Method

- Solution (continued):
  - Multiply $2^{(n-k)} \times D$
    - $2^{(5)} \times 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1 = 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 0$
    - This is a simple shift to the left by five positions
  - Divide $2^{(n-k)} \times D$ / P (see next slide for details)
    - $1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 0 \div 1\ 1\ 0\ 1\ 0\ 1$ yields:
      - **Quotient Q = 1 1 0 1 0 1 0 1 1 0**
      - **Remainder R = 0 1 1 1 0**
  - So, FCS = R = 0 1 1 1 0: Append it to D to get the full frame T to be transmitted
  - T = $\underbrace{1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1}_{D}\ \underbrace{0\ 1\ 1\ 1\ 0}_{FCS}$

Example - Modulo-2 Arith. Method



Do not enter leading zeros

# of bits = # of bits in P,
⇒ result of division is 0

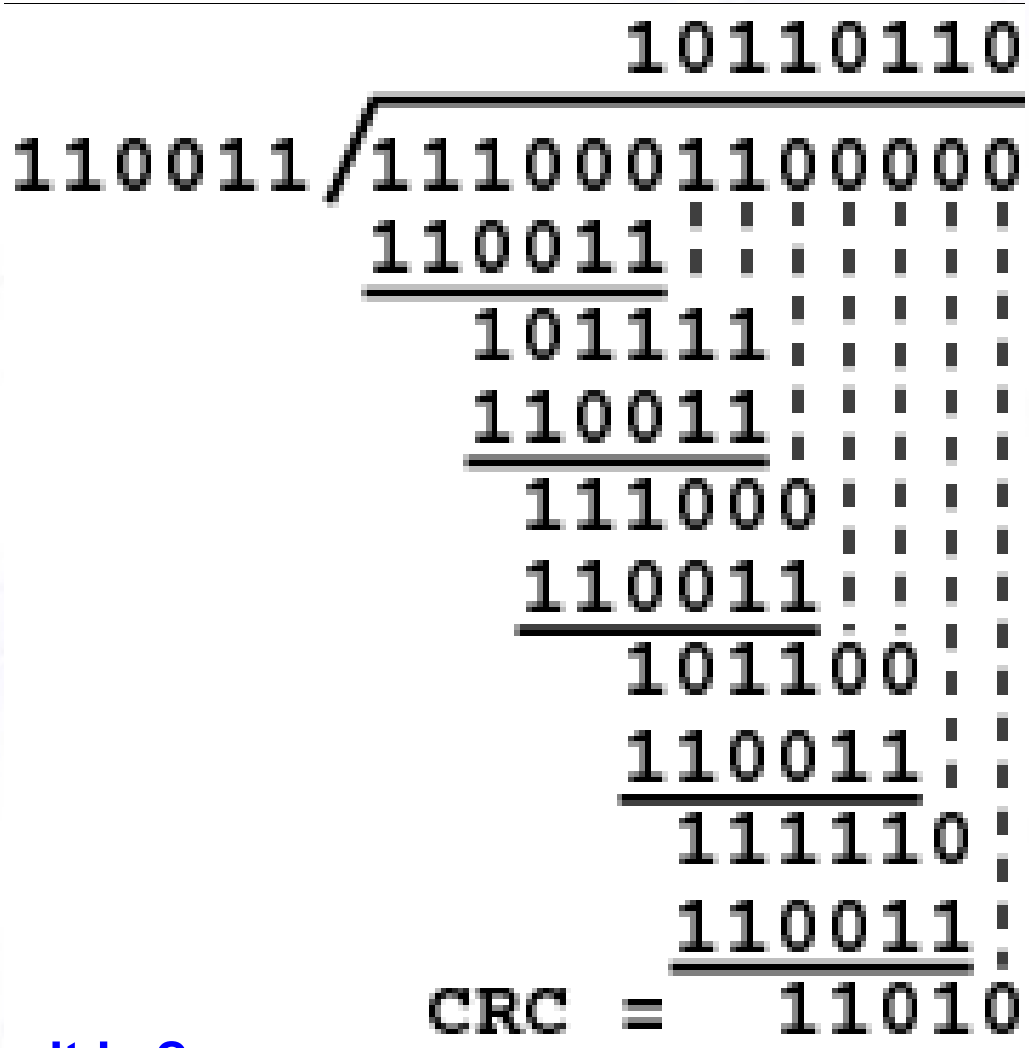
# of bits < # of bits in P,
⇒ result of division is 0, and use next digit

Checks you should do (exercise):

- **Verify correct operation, i.e. that $2^{(n-k)}D = P*Q + R$**
- **Verify that the obtained T (=10100011010111 0) divides P (110101) exactly (i.e. with zero remainder)**

Example

- For P = 110011 & D = 11100011, find the CRC



T to transmit is ?

Chances of missing an error by CRC error detection

- Let E be an n-bit number with a bit = 1 at the position of each error bit error occurring in T
- Error occurring in T causes bit reversal
- Bit reversal is obtained by XORing the bit with 1
- So, received $T_r = T \oplus E$
- Error is missed (not detected) if T_r is divisible by P
- Since T is made divisible by P, this requires E also to be divisible by P ! (can be proven)
- That is a 'bit' unlikely! But it can happen- causing a missed error that we could not detect…

Choice of P(X)

- How should we choose the polynomial P(X) (or equivalently the divisor P)?
- The answer depends on the types of errors that are likely to occur in our communication link
- As seen before, an error pattern E(X) will be undetectable only if it is divisible by P(X)
- It can be shown that the following error types are detectable :
  - All single-bit errors, if P(X) has two terms or more
  - All double-bit errors, if P(X) has three terms or more
  - Any odd number of errors, if P(X) contains the factor (X+1)
  - Any burst error whose length is less than the FCS length (n – k)
  - A fraction ($=1-2^{-(n-k-1)}$) of error bursts of length (n-k+1)
  - A fraction ($=1-2^{-(n-k)}$) of error bursts of length > (n-k+1)

Choice of P(X): Probability of undetected errors

- If all error-patterns are equally likely, and n - k = length of the FCS, then:
  - For a burst error of length (n-k+1), the probability of undetected error is $1/2^{(n-k-1)}$
  - For a longer burst error i.e. length > (n-k+1), the probability of undetected error is $1/2^{(n-k)}$

To improve error delectability use long divisors → (n-k+1) is large …. but this increases FCS overhead, (n-k) large, and processing time…

FCS is 1-bit shorter than P:

P(X) in practical systems

FCS is 1-bit shorter than P:

- There are four widely-used versions of P(X)
  - CRC-12: $P(X) = X^{12} + X^{11} + X^3 + X^2 + X + 1$ (13 bits)
    (r = 13 -1 = 12)
  - CRC-16: $P(X) = X^{16} + X^{15} + X^2 + 1$ (r = 17 -1 = 16)
  - CRC-CCITT: $P(X) = X^{16} + X^{12} + X^5 + 1$
  - CRC-32: $P(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$

FCS Size, not P size

(r = 33 -1 = 32)

- CRC-32 is used for the IEEE 802.3 (Ethernet) LAN standard

Note: P(X) always starts (& ends) with 1

Error Correction

- Once an error is detected, what action can the RX take?:
- Two alternatives:
  - RX asks for a retransmission of the erroneous frame
    - Adopted by data-link protocols such as HDLC (later) and transport protocols such as TCP
    - i.e. A **Backward Error Correction** (BEC) strategy
  - RX attempts to correct the errors if enough redundancy exists in the received data
    - TX uses **Block Coding** to allow RX to correct potential errors
    - i.e. A **Forward Error Correction** (FEC) strategy
    - Used in applications that do not tolerate the extra time required for retransmission, e.g. VoIP.

Error Correction vs. Error Control

- Backward error correction by retransmission is *not* recommended in the following cases:
  - Error rate is high (e.g. wireless communication)
    - Will cause too much retransmission traffic → network congestion
  - Transmission distance is long (e.g. satellite, submarine optical fiber cables)
    - Network becomes very inefficient (Not utilized properly)
- Usually:
  - **Error Correction** methods: Those that use FEC techniques
  - **Error Control** methods: Those that use BEC (retransmission)