Error Detection and Correction

1) see lecture slides

2) Parity Check

a) 01101010 11010100 10101111 11001100 10001000 01110010 11100111

b)
01101010
11010100
10101111
11001100
10001000
01110010
11100111
11000000

c) Parity checks are simple to compute but can only detect single-bit errors.

3) Cyclic Redundancy Check (CRC)

a) Sender Side

Divisor in polynomial form:
x^8+x^2+x+1

Divisor in binary form:
100000111

Data:
1011001001001011

Sender calculates:
101100100100101100000000 (Data extended by m zeros where m is the power of the polynomial form) divided by 100000111

CRC:
10010011

Sender sends:
101100100100101110010011

Receiver calculates:
101100100100101110010011 divided by 100000111

Remainder of this calculation after error-free transmission should be 0.

b) Receiver Side

If the left-most bit of the data is inverted the calculation will return a result other than 0. This will indicate to the receiver that the data have been corrupted.

```
Sender Calculation:              Receiver Calculation

10110010010010110000000         10110010010010111010010011
100000111                        100000111
---------                        ---------
 011000111                        011000111
 000000000                        000000000
 ---------                        ---------
  110001110                        110001110
  100000111                        100000111
  ---------                        ---------
   100010010                        100010010
   100000111                        100000111
   ---------                        ---------
    000101011                        000101011
    000000000                        000000000
    ---------                        ---------
     001010110                        001010110
     000000000                        000000000
     ---------                        ---------
      010101101                        010101101
      000000000                        000000000
      ---------                        ---------
       101011011                        101011011
       100000111                        100000111
       ---------                        ---------
        010111000                        010111001
        000000000                        000000000
        ---------                        ---------
         101110000                        101110010
         100000111                        100000111
         ---------                        ---------
          011101110                        011101010
          000000000                        000000000
          ---------                        ---------
           111011100                        111010101
           100000111                        100000111
           ---------                        ---------
            110110110                        110100100
            100000111                        100000111
            ---------                        ---------
             101100010                        101000110
             100000111                        100000111
             ---------                        ---------
              011001010                        010000011
              000000000                        000000000
              ---------                        ---------
               110010100                        100000111
               100000111                        100000111
               ---------                        ---------
                10010011                         00000000
```

4) Checksum – Sender Side

```
       1001  0011  0110  1100
       1100  1001  1010  1101
       1001  1000  0011  1101
                            1
                         1
                      1 0
                   1 0
              1
              1
          1
        1
     1
       _____
       1111  0101  0101  0110
                            1
       _____
Sum    1111  0101  0101  0111
       0000  1010  1010  1000


       1001  0011  0110  1100
       1100  1001  1010  1101
       1001  1000  0011  1101
       0000  1010  1010  1000
                            1
                         1
                      1 0
                   1 0
               1
             1
            1
          1
        1
     1
       _____
       1111  1111  1111  1110
                            1
       _____
Sum    1111  1111  1111  1111
       0000  0000  0000  0000
```

5) Checksum – Receiver Side

1	0	0	1	0	0	1	1	0	1	1	0	1	1	0	0	
1	1	0	0	1	1	0	0	1	0	0	0	1	1	0	1	
1	0	0	1	1	0	0	0	0	0	1	1	1	1	0	1	
0	0	0	0	1	0	1	0	1	0	1	0	1	0	0	0	Checksum
													1			
											1					
									1	0						
							1	0								
					1											
				1												
		1														
	1															
1																
1	1	1	1	1	0	0	0	1	1	0	1	1	1	1	0	
															1	Carry-Over
1	1	1	1	1	0	0	0	1	1	0	1	1	1	1	1	
0	0	0	0	0	1	1	1	0	0	1	0	0	0	0	0	Inverted sum

6) Hamming Code – Sender Side

```
1001011 1101100 1001001

100?101?1?? 110?110?0?? 100?100?1??

10011010110 11001100000 10011001111
```

7) Hamming Code – Receiver side

```
11101010000  0
11101010000  1
11101010000  0
11101010000  1

1010 = 9 => Single-bit error in bit 9
```

8)

a) The general layout of a HDLC frame includes a flag byte, an address, a control byte, the payload of the frame, a CRC and a flag byte. The flag byte marks the beginning and end of a frame. The address is either indicates the destination if the frame is being send by a master node or the sender if it being send by a slave node. The control byte defines the type of the frame and may include a sequence of the frame and/or a sequence number for an acknowledgement. The CRC provides a mechanism for error detection.

b)
Flow control mechanisms are implemented through the I-frames and S-frames in HDLC. The control byte of the I-frame contains a sequence number for the sender and a sequence number for acknowledgements that indicates which frame is expected next. The control byte of the S-frame have a field for a code that indicates the type of S-frame and a field for a sequence number that can be used to indicate the sequence number of

the next expected frame e.g. for ready-receive (RR) frames that represent acknowledgements or the sequence number of a frame that was not received.
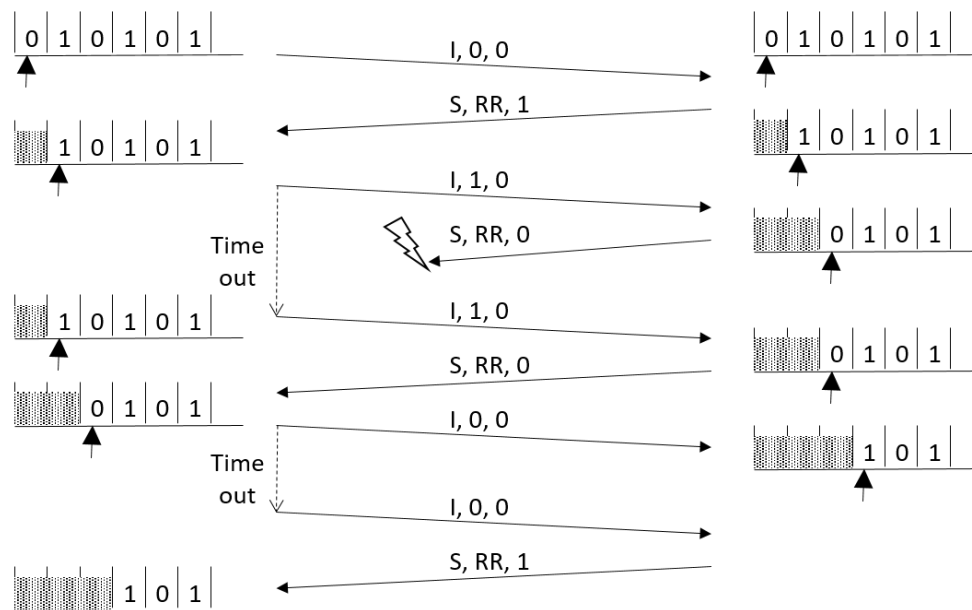

Figure 1: Stop & Wait - The diagram shows the progression of the sender – the arrow indicates the active frame - whenever it receives an acknowledgement and the progression of the receiver when it receives a frame with an expected sequence number. Whenever a timeout occurs, the sender will retransmit the current frame. **Important:** The receiver is passive and only acts when a frame has been received.
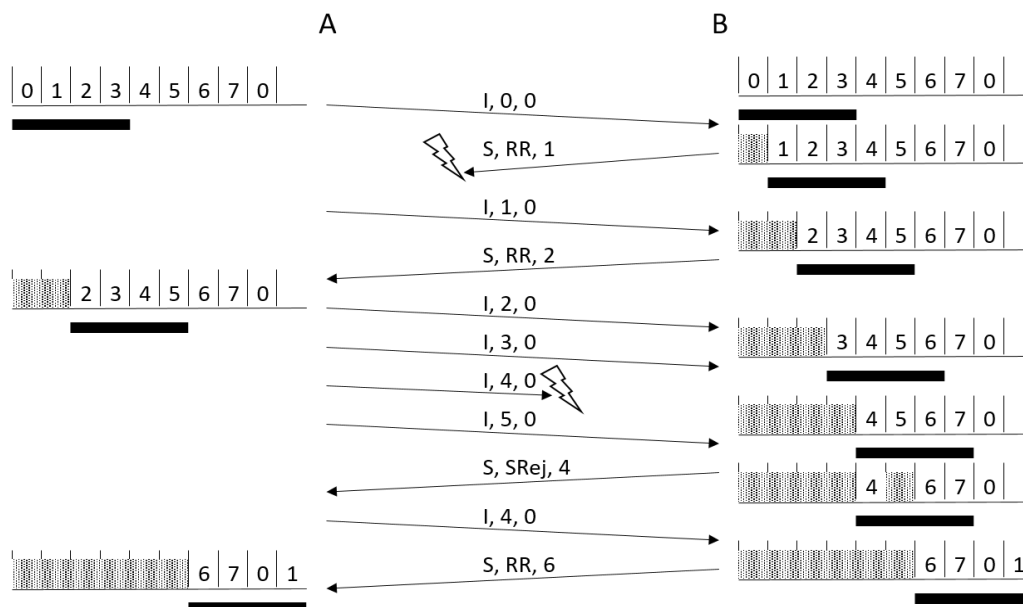
Figure 2: Selective-Repeat: The diagram shows the progression of the windows at the sender and the receiver – indicated by the bar underneath the list of frames. The window at the sender progresses whenever an acknowledgement is being received. An acknowledgement carries the sequence number that the receiver expects next; this indicates that the receiver has received all frames with sequence numbers up to the one indicated in the acknowledgement. The window at the receiver progresses when the first frame in the window has been received. When station B receives a frame without having received a predecessor, it will transmit a selective-repeat request for the frame that is missing.

(The above is also a good example for how to answer exam questions: Give an introduction, state any assumptions that you making, then – if the question asks for diagrams – draw diagrams and then include a description of the diagrams. A number of books and publications only provide very short captions e.g. "Stop & Wait", leaving it to the reader to interpret the figure – which is error-prone. If you provide a figure, provide an explanation to assure that the reader understands the figure.)

9)
The frame should look like the following – starting with a flag byte, then the address, followed by a control byte with assuming that the sequence number for both sender and receiver are 1, followed by the payload, a checksum and finally another flag byte:

```
01111110 00001100 00010001   <200 bytes> 011001110 01111110
Flag     Address  Control byte  Payload Checksum  Flag
```