EEE6221 Typical Exam Solutions 15/16

Q.1

a) (3 marks)

Bandwidth: As low as possible with low or no DC component that can be problematic in ac-coupled channels.

Synchronisation: Efficient built-in mechanism for synchronisation with low overhead, good noise immunity and possibly built-in error detection.

Complexity: Cheap and simple implementation, in particular at the receiver end

b) (8 marks)

Need data encoding schemes that translate signal energy down the lower parts of the available spectrum

i.

From the timing diagram in Fig.Q.1.a, the following characteristics can be extracted for the code:

- A Polar Biphase code since it employs two voltage levels (polarities) and uses positive to negative and negative to positive transitions.
- Non-DC free as transitions are not uniformly compensated (even number of 1s between 0s)
- Middle bit transitions for 1s and end of bit transitions for any 2 consecutive zeros only, hence less transitions overall than comparable biphase codes such as Manchester codes.
- Less transitions results in less bandwidth requirements but also less noise immunity.
- Clock Synchronisation is difficult due to non-uniform change in position and number of transitions.
- Suitable for low bandwidth applications where clock synchronisation is not critical.

From the timing diagram in Fig.Q.1.b, the following characteristics can be extracted for the code:

- Biphase encoding, employs positive to negative and negative to positive transitions to represent the 1 and 0 bits.
- The transition at the start of the bit period is used for synchronisation, but the bits are represented by a presence (for a 1), or an absence (for a 0), of a transition at the middle of the bit period.
- Therefore incurring no latency in recovering the bits and offering improved noise immunity.
- Synchronisation and bit representation, using only 2 voltage levels, as opposed to 3 levels as in RZ type encoding. (less complex to implement)
- A degree of error-detection due to absence of a transition.
- Requires higher bandwidth (due to extra transitions)
- Suitable for applications where synchronisation is more important than bandwidth.

Bit pattern sequence is : 0 1 0 0 1 1 1 0 1 0

Select code in Fig.Q.1.a for its lower bandwidth.

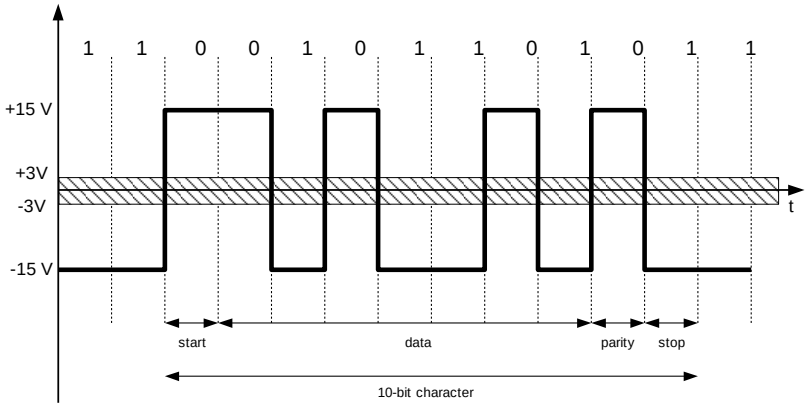
b. (3 marks)

As we know, Nyquist gives us the minimum (theoretical) bandwidth required to recover our data (transmitted at a data rate C) without distortion: B=C/2 Hz, so if B is limited, high frequencies beyond B need not, in theory be transmitted. Idea here to shape our pulses to meet Nyquist minimum bandwidth; in theory this can be achieved by putting our square pulses through an ideal Low-pass filter with bandwidth B. Convolving a pulse input with an ideal low-pass filter gives a *sinc* response whose nulls occur at multiples of the bit period; these nulls will correspond to the centre positions of the following pulses.

If we sample exactly at the bit centres then the sampled value at that point will not be affected by neighbouring values. The *sinc* response has infinite time (side lobes can still have an effect), in practice other pulse shapes with faster side lobe roll-off are used instead, most popular is Raised cosine shaping (Root-raised cosine shaping if filter at both transnitter and receiver) at the cost of increased bandwidth

c. 4 marks [5 marks]

The 7 bit-binary value of the character sent is **1011010** This is illustrated in the diagram below. The RS232 line is initially in the idle state (OFF state), logic 1 (a voltage between –3V and –15V) this signals to the receiver that the link is working in the absence of any transmission. A start bit is first transmitted resulting in a change from the idle state (logic1) to logic 0 (a voltage between +3V and +15V) signalling to the receiver to lock on to 7 bits of the incoming data, LSB first. There may be also an extra eighth bit, a parity bit, to act as a simple error detector (an extra 0 for even parity in this case). A stop bit (or bits), at logic 1, then follows, signalling to the receiver the end of the current character and be ready for the next character (start bit). The data rate in this example is therefore 19200/10=1920 characters per second



RS232 signal for Character 'Z'

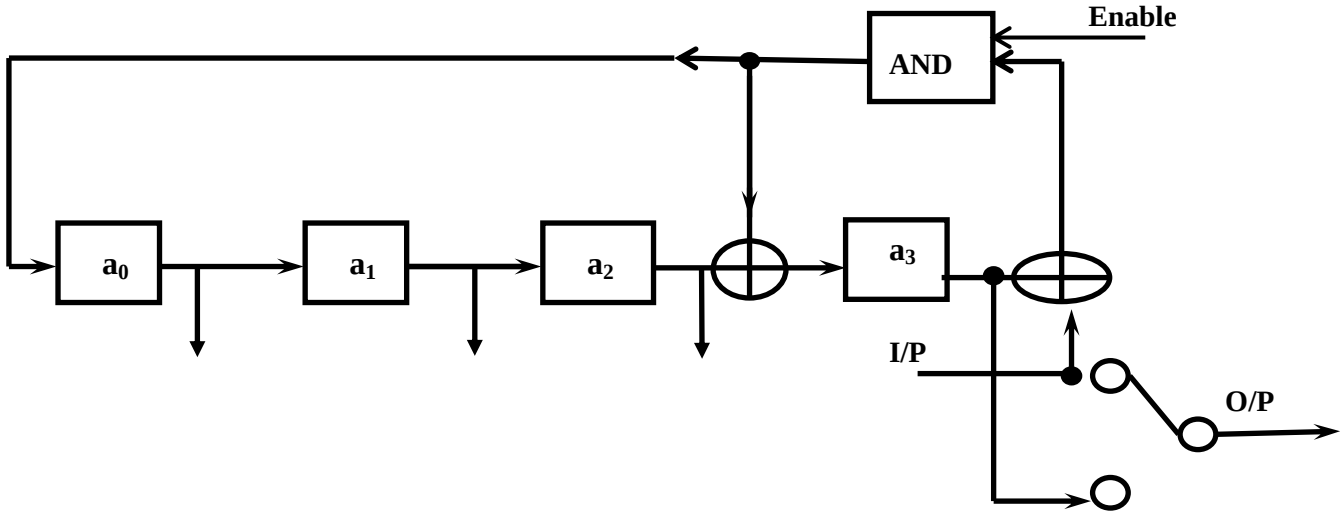
5marks

d. (7 marks)

Ftc = Fn(1+δ)= 1/Ttc (transmitter clock fast)
Frc=Fn(1- δ) = 1/Trc (receiver clock slow)
Tmid-start bit = (m/2+1)Trc (start-bit detected 1 cycle late)
T character = (N-1)xmxTrc + (m/2+1)Trc = (mN-m/2+1)Trc = NxmxTtc
(mN-m/2+1) [1/Fn(1- δ)] = mN [1/Fn(1+δ)]
mN (1- δ) = (mN-m/2+1) (1+ δ)
mN-mNδ = mN +mNδ –m/2 –(m/2)δ +1 + δ
11m-0.11m = 11m +0.11m –m/2 –0.005m +1.01
0.285m = 1.01 which gives m= 3.54
The first even multiple would be 4; in practice higher multiples would be more suitable.

2.
a)

i) (4 marks)



For the first *k* clock cycles, the information bits are fed into the circuit with the AND gate enabled, once these bits have been entered, the *(n-k=4)* remainder bits are held in the a_i registers. To access these, the bottom switch is inverted, the gate is disabled, and the 4 CRC parity bits are clocked out an appended to the k information bits.

ii) (3 marks) Need first to determine correct CRC bits using longhand division. ***(00110)*** = i(x) = $x^2+ x$. Next we need to find the CRC bits generated by the generator polynomial gx)= $x^4+ x^3+1$. This g(x) will generate a 4-bit CRC which we can find by calculating the remainder of division of $i(x)x^{n-k}$ =i(x) $x^4 = x^6+ x^5$ by g(x) using modulo-2 long hand division as follows:

$X^6+ x^5$	$x^4+ x^3+1$
$x^6+ x^5+ x^2$	x^2
x^2 (Remainder = 0100)	

Therefore the transmitted codeword is:00 1100100.

iii) (2 marks) For this 4-bit CRC, the error-detection reliability figure is (1-1/16)x100 = 93.75%

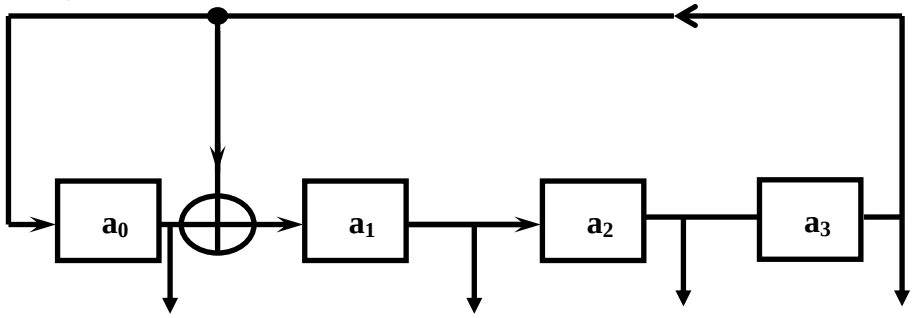
b) (5 marks)
(110100010) xor 110010000 = (000110010) is the erroneous received codeword (3 errors located as by the 1s in the error pattern)
To check for errors we divide the erroneous received codeword by g(x) and check the remainder.

erroneous codeword received is $x^5+ x^4+x$ working out the remainder as before:

$x^5+ x^4+x$	$x^4+ x^3+1$
$x^5+ x^4+ x$	$x+ x^3+ x$
0 (Remainder = 0000)	

A zero remainer for an erroneous codeword. An erroneous codeword that cannot be detected by the 4-bit CRC.This is because the error pattern is a multiple of g(x) - in this case $e(x) =x^8+x^7+ x^4= x^4g(x)$ which results in an undetectable pattern given that for the 4-bit CRC, the error-detection reliability figure is 93.75%

c. (5 marks)



Power of α	Polynomial Representation	Vector Representation $(a_3a_2a_1a_0)$
-	0	0000
0	1	0001
1	α	0010
2	α^2	0100
3	α^3	1000
4	$\alpha + 1$	0011
5	$\alpha^2 + \alpha$	0110
6	$\alpha^3 + \alpha^2$	1100
7	$\alpha^3 + \alpha + 1$	1011
8	$\alpha^2 + 1$	0101
9	$\alpha^3 + \alpha$	1010
10	$\alpha^2 + \alpha + 1$	1011
11	$\alpha^3 + \alpha^2 + \alpha$	1110
12	$\alpha^3 + \alpha^2 + \alpha + 1$	1111
13	$\alpha^3 + \alpha^2 + 1$	1101
14	$\alpha^3 + 1$	1001

- Pre-multiply the message *u(x)* by x^{n-k} giving $x^{12} + x^{10}+x^8$
- Divide $I(x)x^{n-k}$ by g(x) (modulo-2) and obtain the remainder $x^3 + x^2+1$
- Append the remainder to $u(x)x^{n-k}$ to form the BCH codeword $C(x)=x^{12} + x^{10}+ x^8 + x^3+ x^2+1$
  Check that $c(\alpha)=0$

d. **(6 marks)**

In contrast to Meggit decoding, algebraic decoding uses the the syndromes indirectly to locate the errors by defining an error locator polynomial, $\sigma(x)$ the roots, X_k, (or reciprocal roots) of which give the error locations

$$\sigma(x) = \prod_{k=1}^{t}(x + X_k) = x^t + \sigma_1 x^{t-1} + \sigma_2 x^{t-2} + ... + \sigma_t$$

Where X_k is a k^{th} error locator such that $X_k = \alpha^j$, $\alpha^j \in GF(2^m)$, $k= 1,2,...,t.$, denotes an error in j^{th} position(bit), $j=0,1,2,.....n-1$, of the received n-bit word $r(x)$ and hence

$$S_i = e(\alpha^i) = \sum_{k=1}^{t} X_k^i \quad , \quad i = 1,2,.....,2t \qquad (*)$$

using $\sigma(x)$ allows (*) to be translated into a set of linear equations
For t =1 we have $\sigma(x) = x+\sigma_1$ giving an error location $X_1 = \sigma_1$
and from (*) we have $S_1 = \sigma_1$ and $S_2=\sigma_1^{\,2}$, However for a binary code $S_2 = S_1^{\,2}$ which gives $S_1 = \sigma_1$
So the error location is simply $X_1= S_1$
In our case $S_1=r(\alpha)= \alpha^{14}+\alpha^{11}+\alpha^9+\alpha^7+\alpha = \alpha^2= X_1$
To correct r(x) invert the r_2 (the third LSB) to give $r(x)= x^{14}+x^{11}+x^9+x^7+x^2+x$ =(100101010000110)

Requires 7 cycles.
The result can be validated by calculation as follows: 1000 = $i(x) = x^3$; $i(x)x^{n-k} = i(x)\, x^3 = x^6$ dividing by g(x) using modulo-2 long hand division we have:

x^6	$x^3+ x+1$
$x^6+ x^4+ x^3$	$x^3+ x + 1$
$x^4+ x^3$	
$x^4 + x^2+ x$	
$x^3+ x^2+ x$	
$x^3 + x + 1$	

$x^2 + 1$ (Remainder = 101).

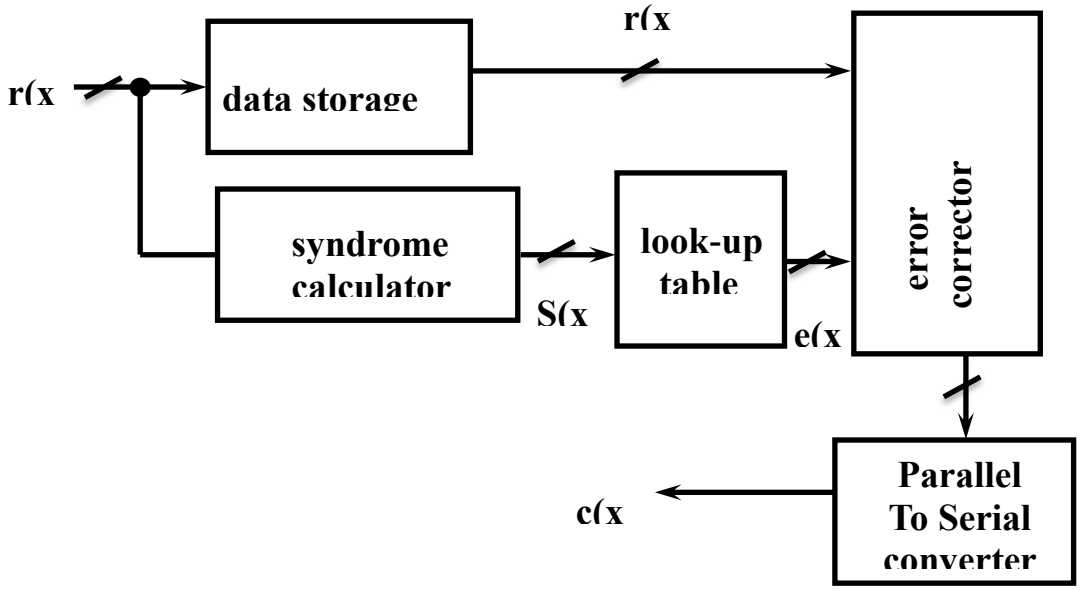
Therefore the transmitted codeword is $c(x) = x^6 + x^2 + 1 = 1000101$.

3.

a) (3 marks)
LUT decoding approach: based on relating the syndromes to their corresponding error patterns in a direct adhoc way; therefore requiring a pre-compuation of all the syndrome for all possible error patterns and then use these in a look up table is then used to to match a calculated syndrome to its error pattern. the steps involved are as follows:
- Calculate the syndrome, S(x)
- Relate a particular error pattern to this syndrome
- Add (XOR) this error pattern to the received word for correction.
-

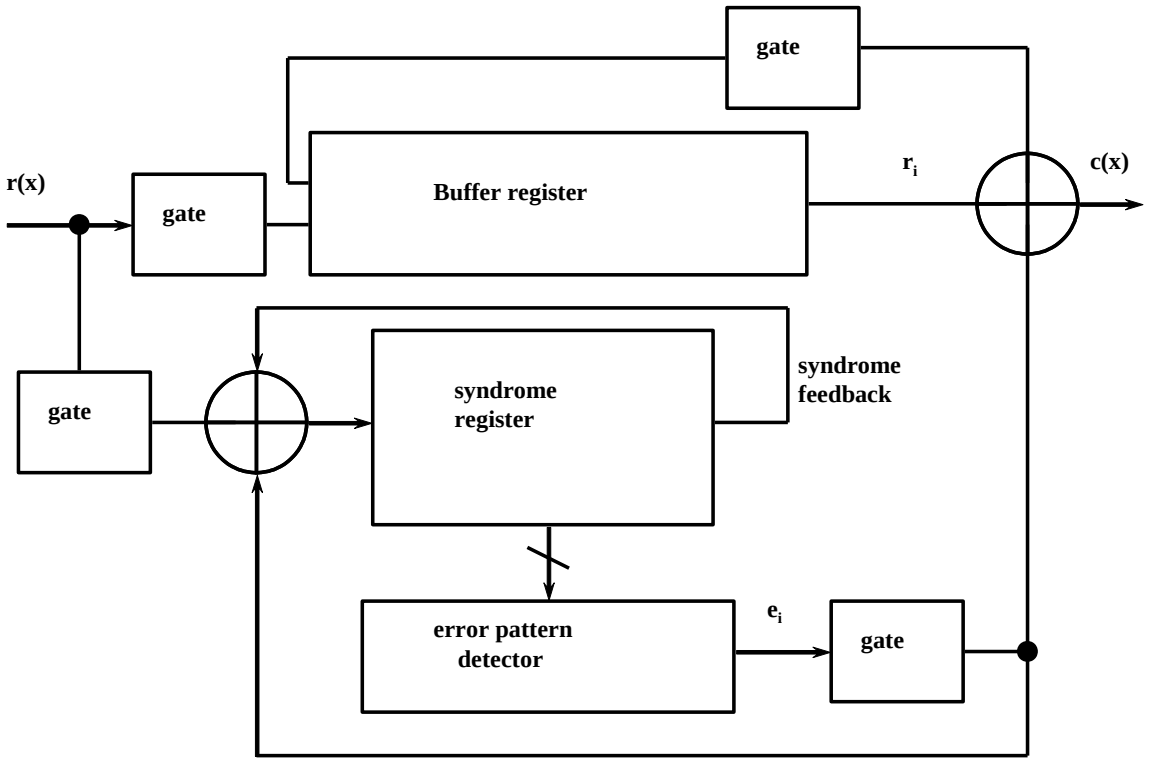
- The complexity of the LUT decoder grows with n and t; Meggitt decoding can be used to reduce this complexity by not looking up all possible error patterns against syndromes. The Idea is to 'look-up' only a single error pattern (MSB error pattern) at each cyclic shifting of the received message cycle till all bits have been shifted as MSBs and therefore all error patterns checked. This decoding strategy takes *n* clock cycles, if a correctable error pattern exists, after *n* clock cycles the syndrome registers will contain 0 (or a zon-zero syndrome if the effect of the errors is not removed - e_i is not added to the syndrome).

b.
i. (4 marks) This is a single-error correcting code therefore there are 15 possible error patterns
Each pattern corresponds to a unique syndrome. A look-up table can be used to match a calculated syndrome to its error pattern. The LUT can be simplified by considering errors in the message bits only. The block diagram for below depicts a LUT decoder.



ii. (4 marks)

- Generate the syndrome, determine from this if there is an error in the MSB (coefficient of x_{n-1}), if not, output rn-1
- If MSB is in error, invert rn-1 and output it, we also remove the effect of this error on the original syndrome by adding ei to the current syndrome
- The syndrome register is then shifted and the test for an error in the current MSB is made (coefficient of xn-2) and the procedure above is repeated till the LSB



iii. (7 marks)

To simplify the problem the syndrome corresponding to MSB being in is provided to the students in the question, that is $\boldsymbol{S(x)= x^3+1}$

$\boldsymbol{r(x)= x^{13}+x^5+x^2+x.}$ Using Meggitt decoding, we first divide r(x) by g(x) = $\boldsymbol{g(x = x^4+ x + 1}$ resulting in an non-zero syndrome that is different from the syndrome provided, this means that the first MSB is not wrong. $\boldsymbol{r(x)}$ is then cyclically shifted resulting in $\boldsymbol{r'(x)= x^{14}+x^6+ x^3+x^2+1}$ this is then divided by g(x) and the remainder compared to the provided syndrome, this is found to be S' = $\boldsymbol{S(x)= x^3+1}$ the same syndrome provided which points to the fact that the error pattern correspond to the current MSB (2nd MSB). Therefore the error is x^{13} and the correct codeord is: c(x)= $\boldsymbol{x^5+x^2+x}$

For validation divide corrected codeword by g(x) and check that syndrome is zero.

From above corrected r(x) = c(x)= $\boldsymbol{x^5+x^2+x}$ divided by g(x) gives remainder $\boldsymbol{0}$ and therefore a valid codeword**.**

c. (3 marks) for double error correction using Meggitt decoding, we need a more complex error pattern detector as in this case we need to 'pre-calculate and pre-store' all syndromes corresponding to the 2 –bit error patterns correspoding to the MSB + e^i where i= 0 to n-2 for n-bit codeword . This implies even more complexity for multiple error correction where algebraic decoding techniques offer a more elegant and far less complex implementation.

d. (4marks)

The complexity of both the LUT and Meggitt decoding methods becomes inpractical for when t is higher than 2 implying increased area and delay. Algebraic decoding techniques offer a more practical and far less complex implementation of cyclic block codes by using the syndromes indirectly to locate the errors. The steps involved are as follows:

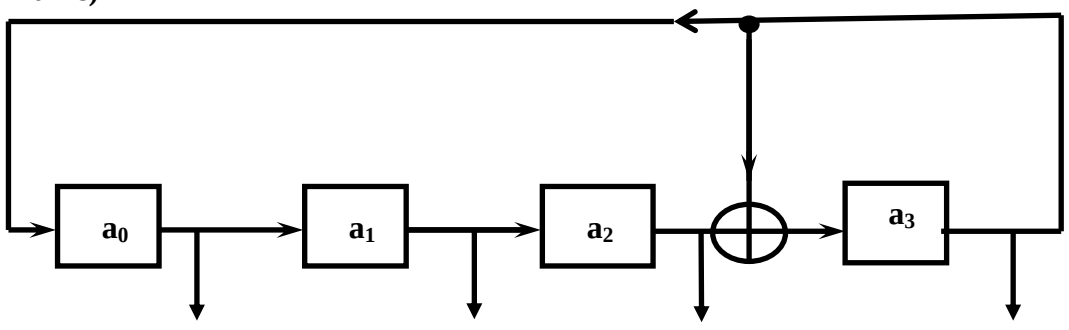
- Calculate Syndromes (by checking if received word still multiple of code generator polynomial)
- From the syndromes, define error locator polynomial $\sigma(x)$(find its coefficients); algebraic decoding uses the syndromes indirectly to locate the errors by defining an error locator polynomial, $\sigma(x)$ the roots, X_k, (or reciprocal roots) of which give the error locations.The coefficients of error locator polynomial $\sigma(x)$ can be found by by solving

$$S_{t+j} + \sigma_1 S_{t+j-1} + \sigma_2 S_{t+j-2} + ... + \sigma_t S_j = 0, j = 1,2,...t$$

- Once, $\sigma(x)$ is determined from the syndromes, its t roots X_k **,** $k=1,...t$ are then evaluated to give the error locations in the form $X_k = \alpha^j$ **,** $\alpha^j \in GF(2^m)$, this can be achieved by a trial of all possible elements of $GF(2m)$ as possible roots; a method of achieving this is by sequential substitution is known as the Chien search.
- For binary codes, correction is performed by inverting the particular j^{th} located bit

4.

a) (4marks)



Powers of α	Polynomial	Binary $(a_3a_2a_1a_0)$
0	0	0000
1	1	0001
α	α	0010
α^2	α^2	0100
α^3	α^3	1000
α^4	$\alpha^3 + 1$	1001
α^5	$\alpha^3 + \alpha + 1$	1011
α^6	$\alpha^3 + \alpha^2 + \alpha + 1$	1111
α^7	$\alpha^2 + \alpha + 1$	0111
α^8	$\alpha^3 + \alpha^2 + \alpha$	1110
α^9	$\alpha^2 + 1$	0101
α^{10}	$\alpha^3 + \alpha$	1010
α^{11}	$\alpha^3 + \alpha^2 + 1$	1101
α^{12}	$\alpha + 1$	0011
α^{13}	$\alpha^2 + \alpha$	0111
α^{14}	$\alpha^3 + \alpha^2$	1100

b.

i) (3marks)

A t-error correcting (n,k) RS code has : $d_{min}=2t+1$, $2t = n-k$, $n= 2^m -1$
In thiscase:m = 4 therefore n = 15; n-k=2t=4 therefore k=11, giving RS(15,11). Each symbol is 3-bits and therefore any 2-symbol error with errors up to 4-bits/symbol can be corrected, also bursts up to 2-symbols (8-bits can also be corrected).

ii) 6 marks

A t-error correcting (n,k) RS code has a generator polynomial g(x) with 2t consecutive powers of a primitive element $\alpha \in GF(2m)$, as roots : $g(x)=(x-\alpha)(x-\alpha^2)(x-\alpha^3)\ldots..(x-\alpha^{2t})$
For a (15,11) RS code over $GF(2^4)$, using
$p(x) = x^4 + x^3 + 1$, $g(x)=(x+\alpha)(x+\alpha^2)(x+\alpha^3)(x+\alpha^4) = x^4 + g_3x^3 + g_2x^2 + g_1x + g_0$
$g(x)=(x+\alpha)(x+\alpha^2)(x+\alpha^3)(x+\alpha^4) = x^4 + g_3x^3 + g_2x^2 + g_1x + g_0$ **$= x^4 + \alpha^7x^3 + \alpha^4 x^2 + \alpha^{12}x + \alpha^{10}$**

To find the codeword:
Pre-multiply the message *u(x)* by x^{n-k} giving $\mathbf{x^5 + x^4}$
Divide $I(x)x^{n-k}$ by g(x) (modulo-2) and obtain the remainder $\mathbf{\alpha x^3 + \alpha^7 x^2 + \alpha^8}$
Append the remainder to $u(x)x^{n-k}$ to form the RS codeword $\mathbf{C(x) = x^5 + x^4 + \alpha x^3 + \alpha^7 x^2 + \alpha^8}$

For validation, enough to check that $c(\alpha) = \alpha^5+\alpha^4+\alpha^4+\alpha^9 \ \alpha^8 = 0$ =

c) **(6marks)**

GF(2^3) elements representation using Irreducible polynomial $p(x) = x^3 + x^2 + 1$

Powers of α	Polynomial	Binary ($a_2a_1a_0$)
0	0	000
1	1	001
α	α	010
α^2	α^2	100
α^3	$1 + \alpha^2$	101
α^4	$1 + \alpha + \alpha^2$	111
α^5	$\alpha + 1$	011
α^6	$\alpha + \alpha^2$	110

Algebraic decoding uses the the syndromes indirectly to locate the errors by defining an error locator polynomial, $\sigma(x)$ the roots, X_k, (or reciprocal roots) of which give the error locations
Where X_k is a k^{th} error locator such that $X_k = \alpha^j$, $\alpha^j \chi$ $GF(2^m)$, $k = 1,2,...,t.$, denotes an error in j^{th}

$$\sigma(x) = \prod_{k=1}^{t}(x + X_k) = x^t + \sigma_1 x^{t-1} + \sigma_2 x^{t-2} + ... + \sigma_t$$

position(bit), $j=0,1,2,.....n-1$, of the received n-bit word $r(x)$, the magnitude of the error, Yj can then be evaluated using the equations above:

$$X_1 = \sigma_1 = \frac{S_2}{S_1}$$

Need to work out first 2 syndromes:
In our case $S_1 = r(\alpha) = \alpha^4$, and $S_2 = r(\alpha^2) = \alpha^4$ giving $X_1 = \alpha^0$ (need to correct r_0; LSB symbol)

$$Y_1 = \frac{S_1^2}{S_2} = \alpha^4$$

To correct r(x) add α^4 to r_0 to give: $\alpha^4 + 1 = \alpha^6$ resulting in a corrected
r(x) $= \alpha^2x^5 + \alpha x^4 + \alpha x^2 + \alpha^6 x + \alpha^6$

5.

a

(3 marks)

Losseless compression:

- Exploits statistical redundancy in data
- Must be reproducible without errors
- Gives limited compression ratios
- Requires less processing

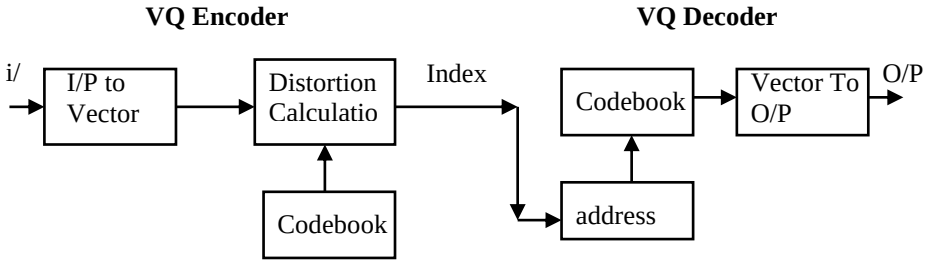
Lossy compression

- Exploits spatial/spectral redundancy and perception
- Must be reproducible within a specified measure
- Gives higher compression ratios
- Requires more processing

b. (4 marks)

VQ:

- Generalisation of scalar quantisation
- Compression achieved by comparison of an input vector to a finite set of re-produced (optimal) code-vectors to achieve minimum distortion.
- Codebook can be defined in a dynamic way (i.e for each image) or designed beforehand using training data (images) representative of the application.
- Subsampling and encoding result in size reduction (compression)
- Complex to implement (computationally)
- Efficiency dependent on efficiency of codebook generation algorithms and size of coodbook
- Exhibit high compression rates



c. (4 marks)

- Has good compaction efficiency of energy
- Is invertible and separable
- Has image independent basis
- Has fast algorithms for computation and implementation

In DCT domain compression samples are grouped into blocks which are transformed into a domain which allows a more compact representation. Compression is achieved by then discarding the less important information

Limitations:

- Blocking effects (block transform)
- Reduced Energy compaction efficiency for for weakly correlated data

d. 7 marks

Using row-column decomposition:
For first and fourth rows:
DCT0 = $1/\sqrt{2}$ ($10/\sqrt{2}$ cos 0 + $10/\sqrt{2}$ cos 0) = 10
DCT1 = $1/\sqrt{2}$ (10 cos Π/8 + 10 cos 5Π/8) = $1/\sqrt{2}$ (0.924x10 - 0.383x10) =3.82
DCT2 = $1/\sqrt{2}$ (10 cos Π/4 + 10 cos 5Π/4) = $1/\sqrt{2}$ (10x0.707 –0.707 x10) =0
DCT3 = $1/\sqrt{2}$ (10 cos 3Π/8 +10 cos 15Π/8) = $1/\sqrt{2}$ (10x0.383+0.924x10)= 9.24

resulting in 1-D DCT matrix:

10	3.82	0	9.24
0	0	0	0
0	0	0	0
10	3.82	0	9.24

The 2-D DCT is obtained by applying the 1-D DCT to this matrix along the columns. Since the resulting columns have the same pattern, we just need to calculate the DCT for the first column then derive the rest.
DCT0 = $1/\sqrt{2}$ ($10/\sqrt{2}$ cos 0 + $10/\sqrt{2}$ cos 0) = 10
DCT1 = $1/\sqrt{2}$ (10 cos Π/8 + 10cos 7Π/8) = $10/\sqrt{2}$ (0.924 –0.924) =0
DCT2 = $1/\sqrt{2}$ (10 cos Π/4 + 10 cos 7Π/4) = $10/\sqrt{2}$ (0.707 +0.707) =10
DCT3 = $1/\sqrt{2}$ (10 cos 3Π/8 + 10 cos 21Π/8) = $10/\sqrt{2}$ (0.383 - 0.383) = 0
Therefore the 2-D DCT matrix in this case is :

10	3.82	0	9.24
0	0	0	0
10	3.82	0	9.24
0	0	0	0

Significant energy at the higher-frequencies, with coefficients having relative high-values, if heavily quantised will result in significant loss of detail when recovering the data. Using appropriate quantisation matrix will normally result in this case in lower compression rate. (this reasoning assumes blocks of 4x4 are used instead of the usual 8x8 for simplification purposes)

e. 7 marks

First perform Zig-Zag encoding on the qunatised DCT coefficients to obtain

$$\begin{bmatrix} 47 & 28 & -22 & 12 & -10 & -3 & 0 & -2 \\ 10 & 9 & 0 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

AC codewords starting with first AC coefficient:

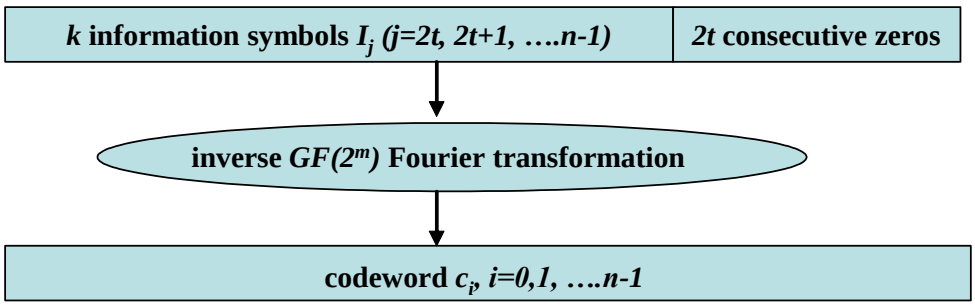
Value =28, run/size=0/5, Huffman code=11010, amplitude=11100, hence corresponding codeword=1101011100 (10 bits).
Value=-22, run/size=0/5, Huffman code=11010, amplitude= one's complement of 10110=01001, hence corresponding codeword=1101001001 (10bits)
Similarly the codewords for the remaining AC coefficients are therefore
12 →10111100 (8bits); -10 → 10110101 (8 bits); -3→run/size=0/2 →0100 (4 bits)
-2→run/size=1/2→1101101 (7 bits); 10→run/size=0/4 →10111010 (8bits);
9→run/size=0/4→10111001(8bits);-2→run/size=1/2 →1101101(7bits);
1→run/size=0/1 →001 (3 bits); EOB→run/size=0/0 →1010 (4 bits);

Number of bits taken by AC coefficients = 77 bits

Total number of bits (dc+ac) = 5 + 77 = 82 bits, therefore average bit rate = 82/64 = 1.28 bit/pixel giving a compression rate of 8/1.36 = 6.25

6.
a. (3 marks)
In block codes *k* information symbols are formed into a word $(I_1,I_2, ..., I_k)$. This information word is then encoded into *n* codeword symbols $(C_1,C_2, ..., C_n)$.($n > k$).Typically these symbols are strings of bits. So a block of *k* information bits is converted into a block of *n* code bits resulting in an *(n,k)* block codes where $R = k/n$ is the rate of the code. Block codes have no memory and so consecutive codewords are independent. However, because we are dealing with blocks of data, buffering memory and latency overheads are always associated with block codes.
Block codes , as opposed to convolutional codes, can be cross interleaved for reliable storage of data. Block codes can be concatenated with convolutional codes or mapped together onto an iterative (turbo) configuration for higher performance over some channels.
Hard decision decoding of block codes, although involves in cases significant latency and hardware overheads, requires well defined stages and hence the process is rather mechanical. Soft-decision decoding of block codes is rather difficult. Block codes are employed, and in some cases are standard, in satellite communications, CDs, DVB.
An (*n,k,m*) convolutional encoder takes, a continuous stream of *k* information bits and produces an encoded sequence of *n* bits at an $R = k/n$ code rate. But each *n* bit codeword depends not only on the *k* information bits but also on *m* previous message blocks. The encoder has therefore memory of order *m*, which is rarely of more than a few bits length, hence resulting in minimal buffering and latency overheads.
Convolutional Encoders are in essence simple state machines hence very easy to implement in hardware. However, decoding is complex as it involves searching for a best fit path to recover the sent information but is more amenable to soft-decision decoding compared to block codes, which can result in better coding performance.
Hence convolutional codes are suitable for very low SNR channels, and also where transmitters use simple low power devices.

b. [4 marks]
RS encoding can be performed in the frequency domain using a *GF* Fourier transform, this is possible because the *2t* consecutive powers of □ roots in time domain correspond to *2t* consecutive zeros in frequency domain. Therefore *k* information symbols *Ij (j=2t, 2t+1, ….n-1)* can be encoded by letting *Cj = 0, j=0,…2t-1* and *Cj = Ij, j=2t,2t+1, ….n-1 t*hen perform an inverse $GF(2^m)$ Fourier transformation on *Cj* to generate the codeword *ci, i=0,1, ….n-1*



RS decoding can also be performed in the frequency domain by recursive extension of the error spectrum as follows:

- Apply forward *GF* Fourier Transform to the received word to obtain the original frequency domain message
- If the message is not corrupted, *2t* consecutive zeros will appear in the transformed word and the information symbols is retrieved
- In the case of errors, some or all of the *2t* zero positions will be non-zero. The complete error spectrum is then obtained by recursive extension from the *2t* error spectrum.
- A time domain complete error polynomial is then obtained by inverse *GF* Fourier Transform of the error spectrum. The time domain error polynomial is added (*XORed*) with the received message for correction
- A forward Fourier Transform is then applied to the corrected message to obtain the original frequency domain message

c.

i. [5 marks]

from C'= (α^4, α^2, α, α, α^5, α, α^2), E= (α^4, α^2, α, α, ? , ?, ?) So knowing the first 2t=4 error spectral values Ej (j-0 to 3) , we want to find the rest of the error spectrum. This can be achieved simply by using the FSR circuit given that recursively calculates the remaining 3 error spectrum values from the already known 4 values of the spectrum.

The process starts by preloading, registers Ej+1 and Ej+2 with E0 (α^4) and E1(α^2), respectively such that on the first clock cycle E0 is output followed by E1 on the next cycle and such that E2 (α)and E3(α) are output in the next 2 clock cycles. Further clocking of the circuit will generate the remaining 3 error spectrum values.

recursively from above we complete the error spectrum E = (α^4, α^2, α, α, α^4 , 0, α^2)

ii. [6 marks]

Find inverse *GF(2^3)* Fourier Transform of the error spectrum E: e= (e_0…., e_6) as follows:

$$e_i = \sum_{j=0}^{6} E_j \alpha^{-ij} \quad \text{for } i=0,1,\ldots.6$$

$e_0 = E0\,\alpha^0 + E1\,\alpha^0 + E2\,\alpha^0 + E3\,\alpha^0 + E4\,\alpha^0 + E5\,\alpha^0 + E6\,\alpha^0 = E4\,\alpha 0 = 0$

$e_1 = E0\,\alpha^0 + E1\,\alpha^{-1} + E2\,\alpha^{-2} + E3\,\alpha^{-3} + E4\,\alpha^{-4} + E5\,\alpha^{-5} + E6\,\alpha^{-6} = \alpha^2$

$e_2 = 0$

$e_3 = 0$

$e_4 = \alpha$

$e_5 = 0$

$e_6 = 0$

Giving e = (0,α^2, 0, 0, α, 0, 0)

d. (7 marks)

Set up a cost table for Viterbi decoding of received sequence as follows (students can simplify the trellis by ignoring any paths with more than 3 errors):

Depth/Op code(Q_1Q_0)	11	01	11	00	01	01	10	00	01	11	
0/00	2	3	5	5/0	1/6	2/5	3/4	3/2	3/6	5/6	
1/11			0	6/5	4/5	3/6	2/6	6/5	4/4	3/5	
2/01		0	4	4/3	2/3	1/4	4/5	4/3	4/3	4/5	
3/10			3	5/4	5/6	6/5	3/3	3/6	5/5	6/4	
4/11	0	3	3	7/2	1/6	2/5	3/4	5/4	3/6	3/4	
5/00			2	4/3	4/5	3/6	2/6	4/3	4/4	5/7	
6/10		2	4	4/3	4/5	3/6	2/3	4/3	6/5	4/5	
7/01			3	5/4	3/4	4/3	5/5	3/6	3/3	6/4	

There are two paths with equal cost and therefore same likelihood of being correct. In this case, further information provided by the receiver can be used to advantage to determine the most likely code. This is known as soft-decision decoding.

Working through the trellis backwards we can determine that the transmitted data was either 1011110100 or 1000100001