EEE6221 Typical Exam Solutions 14/15

Q.1

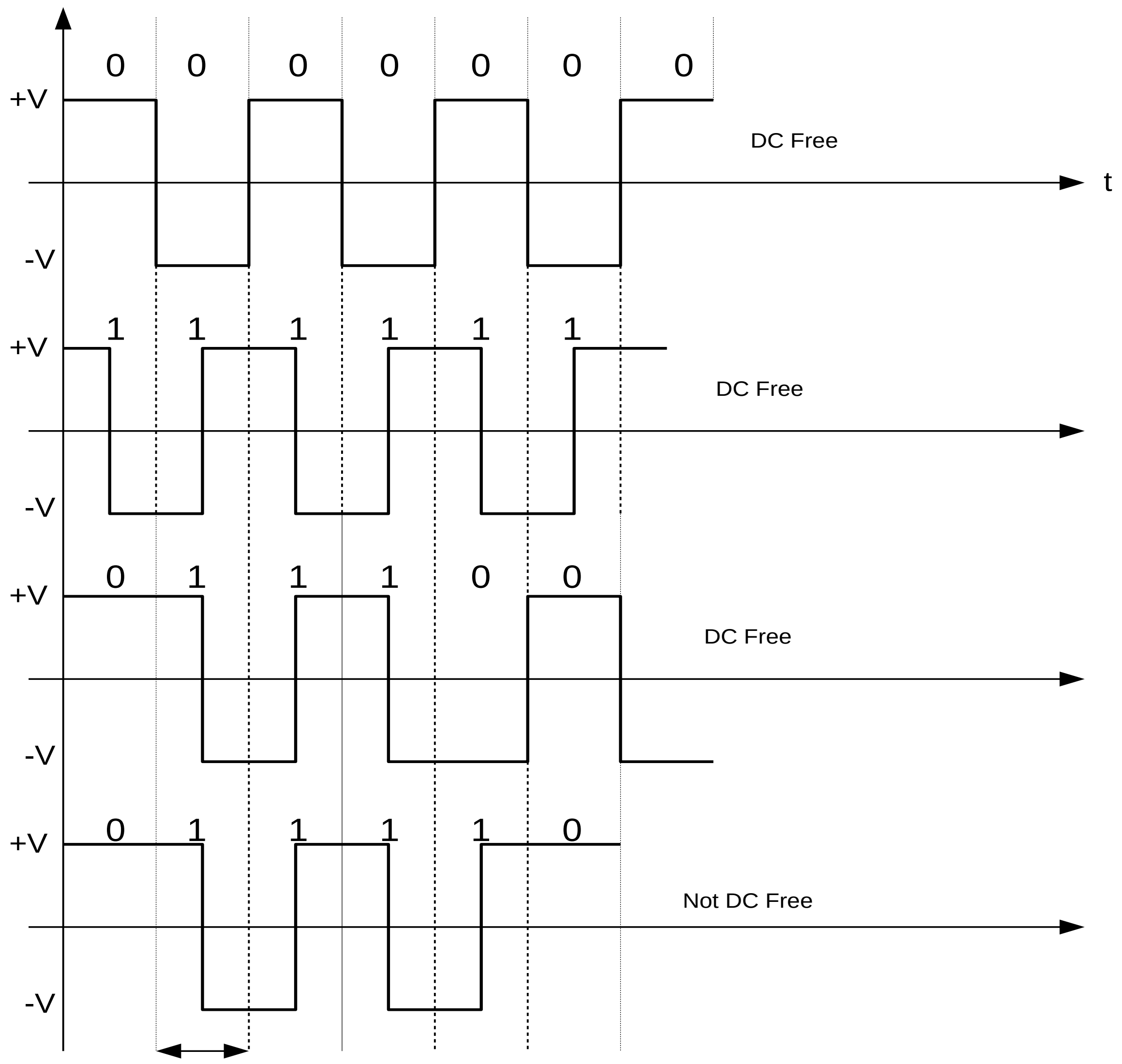
a) (2 marks)

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)



.

3 marks

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.
- 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.

The code can be modified to provide DC free operation by ensuring that no transition occurs for the last one bit whenever an even number of ones occurs between zeros. This would incur additional complexity and delay .

5marks

c) (8)

i) 2marks

Worst case conditions: Transmitter master clock fast, Receiver clock Slow and start bit detected one cycle late.

ii) 6marks

Ftc = Fn(1+δ)= 1/Ttc (transmitter clock fast)
Frc=Fn(1- δ) = 1/Trc (receiver clock slow)
Tmid-start bit = (8+1)Trc = 9Trc (start-bit detected 1 cycle late)
T character = (N-1)x16xTrc + 9Trc = (16N-7)Trc = Nx16xTtc
(16N-7) [1/Fn(1- δ)] = 16N [1/Fn(1+δ)]
16N (1- δ) = (16N-7)(1+ δ)
16N-16Nδ = 16N +16Nδ -7 -7δ
(32N-7)δ = 7 therefore δ = 7/(32N-7)

m=8, N=10 and δ ~ 2.2%
m=16, N=18 and δ ~ 1.2%
m=32, N=34 and δ ~ 0.65%

Increasing m puts a more stringent restriction on the clock. Correct detection becomes more severe as the number of bits gets higher that is why Synchronous Transmission is used most comms networks.

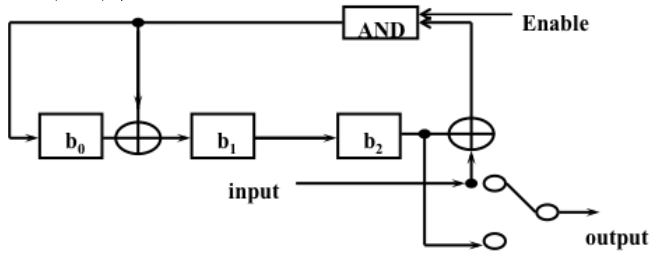
d. 2marks

Data scrambling allows long sequences of consecutive 0s or 1s to be avoided and no extra bandwidth. This is achived by using a feedback shift register in which data is XORed with a PRBS. However, dta scramblers are prone to errors due the lack of redundancy and care must be taken (interms of extra complexity) to avoid lock up.

2.

a)

i) (5)



CLK	En	I/P	FBK	b0	b1	b2	O/P
1	1	1	1	1	1	0	1
2	1	0	0	0	1	1	0
3	1	0	1	1	1	1	0
4	1	0	1	1	0	1	0
5	0	0	0	0	0	1	1
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	1

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:

$$
\begin{array}{l|l}
x^6 & x^3+x+1 \\
\hline
x^6+x^4+x^3 & x^3+x+1 \\
\hline
x^4+x^3 & \\
x^4+x^2+x & \\
\hline
x^3+x^2+x & \\
x^3+x+1 & \\
\hline
\end{array}
$$

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

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

ii)
(5 marks)
Two possibilities for the students:
One direct one is to work out the 4 unique remainders for the four possible single error patterns Only interested in the message bits), and find the error pattern that corresponds to the 101 ($x^2 + 1$) remainder (syndrome). The four possibilities are: for $c1(x)= x^5 + x$ (Error in MSB) remainder is 101; for $c2(x)= x^6 + x$ (error in second MSB) remainder is 111; $c3(x)= x^6 + x^5 + x^4 + x$ (error in third MSB) remainder 110 and $c4(x)= x^6 + x^5 + x^3 + x$ (error in third MSB) remainder is 011. Clearly it is seen that the third case with third MSB bit being erroneous is the correct answer.
An alternative (more elegant approach) is to identify the GF(2^3) element that corresponds to 110 (using irreducible polynomial x3+x+1) and match it to the bit location in this case ($\alpha^4 = \alpha^2 + \alpha$)means that the erroneous bit corresponds to the 5th bit or 3nd MSB.
The above approach is possible as long as $k < 2^n$ and for single error correction only with unique syndromes for individual error patterns.

b [8 marks]

i) [2 marks]
Degree of $\sigma(x)$ lead us to assume that no more than two errors have occurred; a Chien search would confirm this. Of course a correctable pattern is assumed here..

ii) [6 marks] (students will have to build their *GF(2^4)* using the primitive polynomial $p(x) = x^4 + x + 1)$

Need to evaluate $\sigma(\alpha^i)$ *for i=1,2,3,4 and check which of these values are root*

$\sigma(1) = 1 + \boldsymbol{\alpha}^4 + \boldsymbol{\alpha} = 1 + \boldsymbol{\alpha} + \mathbf{1} + \boldsymbol{\alpha} = \mathbf{0}$ **LSB bit is erroneous**

$\sigma(\alpha) = \alpha^2 + \boldsymbol{\alpha}^4\alpha + \boldsymbol{\alpha} = \alpha^2 + \boldsymbol{\alpha}^5 + \boldsymbol{\alpha} = \alpha^2 + \alpha + \boldsymbol{\alpha}^2 + \boldsymbol{\alpha} = \mathbf{0}$ therefore 2nd LSB bit is the second error location

No need to continue as we have 2 errors only, the limit of our code

Correct codeword is $r(x) = x^{10}+x^9+x^8+x^6+x^2$

3.
a) (3)

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

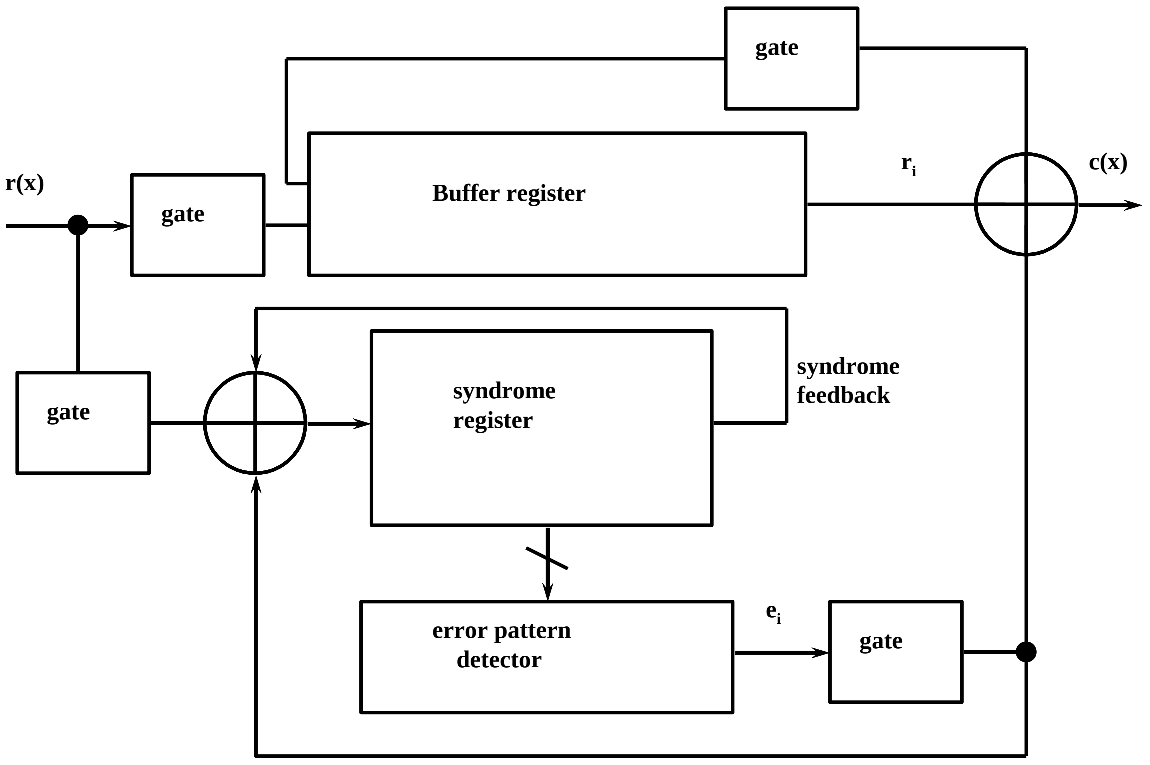
$$\sigma(x) = \prod_{k=1}^{t}(x + X_k) = x^t + \sigma_1 x^{t-1} + \sigma_2 x^{t-2} + \ldots\ldots + \sigma_t$$

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

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

using $\sigma(x)$ allows (*) to be translated into a set of linear equations

b) (5 marks) Generic Circuit for Meggit Decoder



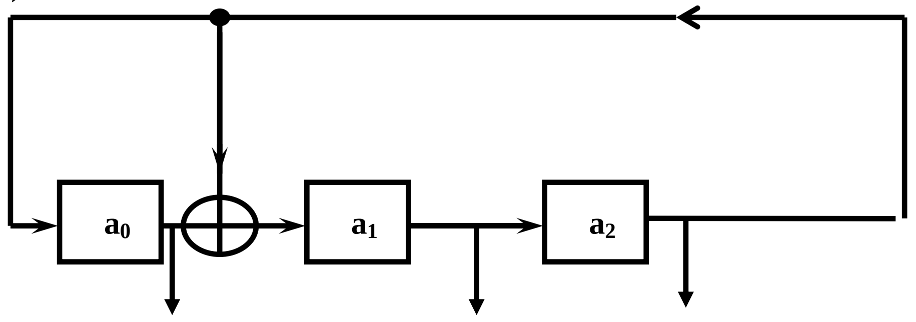
Meggit decoding

- 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

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.(9 marks)
Since it is specified that the 2 errors are in the fisrt 4 MSBs, the error pattern detector in our meggitt decoding will need to detect the 2-bit error patterns corresponding to the syndromes for the 2-bit erros patterns involving MSB+ 2ndMSB, MSB+ 3rdMSB and MSB+4thMSB only. To simplify the problem the syndromes corresponding to the patterns have been provided to the students in the question.
$r(x)= x^{14}+x^{13}+x^{12}+x^{11}+x^9+x^7+x^2+x$. Using Meggitt decoding, we first divide r(x) by g(x) = ***$g(x)=(721)_{Octal}$*** = 111010001 = ***$x^8+x^7+x^6+ x^4+ 1$*** resulting in an non-zero syndrome that is different from the syndrome provided, this means that the first MSB is not wrong and the 2 errors must be within the next 3 MSB bits; ***$r(x)$*** is then cyclically shifted resulting in ***$r'(x)= x^{14}+x^{13}+x^{12}+x^{10}+x^8+x^3+x^2+1$*** this is then divided by g(x) and the remainder compared to the provided syndrome, this is found to be $S' = S =$ ***$x^7+x^4+x^3+x^2$*** the same syndrome provided which points to the fact that the 2 error patterns correspond to the current MSB (2nd MSB) and either (one of the following 2 bits). A further cyclic shift will give: ***$r''(x)= x^{14}+x^{13}+x^{11}+x^9+x^4+x^3+x^2+x+1$*** divided again by g(x) results in a reminder/syndrome $S''=$ ***$x^7+ x^6+x^5+ x^4+x^2+1$*** that is different from S which means that the 2nd error must be the 3rd LSB as we both errors are located in the first 4 bits. Therefore the errors are x^{13} and x^{12} and the correct codeord is: c(x)= ***$x^{14}+ x^{11}+x^9+x^7+x^2+x$***
For validation divide corrected codeword by g(x) and check that syndrome is zero.
From above corrected r(x) = c(x)= ***$x^{14}+ x^{11}+x^9+x^7+x^2+x$*** divided by g(x) gives ***$x^6+ x^5+x^2+ x$*** and remainder ***0*** and therefore a valid codeword.

4.
a) (4)



Pre-load the register with α *(010)* and obtain an element every clock cycle

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	$1 + \alpha$	011
α^4	$\alpha + \alpha^2$	110
α^5	$1 + \alpha + \alpha^2$	111
α^6	$1 + \alpha^2$	101

b.

i) (2marks)

A t-error correcting (n,k) RS code has : $d_{min}=2t+1$, $2t = n-k$, $n= 2^m -1$
In thiscase: k=3, n-k=2t=4 = number of check symbols giving RS(7,3). Each symbol is 3-bits and therefore any 2-symbol error with errors up to 3-bits/symbol can be corrected, also bursts up to 2-symbols (6-bits can also be corrected).

ii) 7 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 : $g(x)=(x-\alpha)(x-\alpha^2)(x-\alpha^3)\ldots\ldots(x-\alpha^{2t})$
For a (7,3) RS code over $GF(2^3)$, using
$p(x) = x^3 + x + 1$, $g(x)=(x+\alpha)(x+\alpha^2)(x+\alpha^3)(x+\alpha^4) = x^4 + g_3x^3 + g_2x^2 + g_1x + g_0 = \mathbf{x^4 + \alpha^3x^3 + x^2 + \alpha x + \alpha^3}$ (3marks)

To find the codeword:
Pre-multiply the message *u(x)* by x^{n-k} giving $\mathbf{x^6+ \alpha^3x^5+ x^4}$
Divide $I(x)x^{n-k}$ by g(x) (modulo-2) and obtain the remainder $\mathbf{\alpha x^3 + \alpha^3x^2}$
Append the remainder to $u(x)x^{n-k}$ to form the RS codeword $\mathbf{C(x)= x^6+ \alpha^3x^5+ x^4+ \alpha x^3 + \alpha^3x^2}$
(3marks)
For validation, enough to check that $c(\alpha) = \alpha^6+\alpha^3\alpha^5+\alpha^4+\alpha\alpha^3 +\alpha^3\alpha^2 =\alpha^2+1+\alpha+\alpha^4+\alpha^4 + \alpha^2 +\alpha +1= \mathbf{0}$
(1 mark)

c) **(7 marks)**
Algebraic decoding uses the the syndromes indirectly to locate the errors by defining an error locator polynomial, *σ(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 $X_k = \alpha^j$, $\alpha^j \chi$ $GF(2^m)$, $k= 1,2,\ldots,t,$denotes an error in j^{th}

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

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:

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

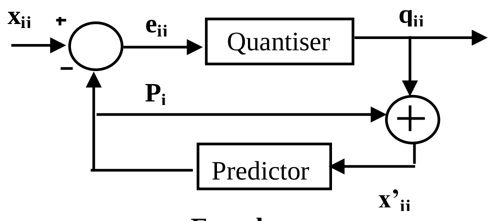
Need to work out first 2 syndromes:
In our case $S_1= r(\alpha) =\alpha^4$ and $S_2= r(\alpha^2) = \alpha^5$ giving $X_1=\alpha$ (need to correct r_1; the second LSB symbol)

$$Y_1 = \frac{{S_1}^2}{S_2} = \alpha^3$$

To correct r(x) add α^3 to r_1 to give: $\alpha^3 +1 =\alpha$ resulting in a corrected
$\boldsymbol{r(x)= \alpha^5x^6+x^5+\alpha^2x^4+\alpha^5x^3+\alpha x^2+ \alpha x+1}$

5.
a
4 marks

DPCM: spatial domain technique that is sample based adopted for highly correlated/predictable data such as speech and in DC values in blocks in transform domain based compression. Im image compression for example:

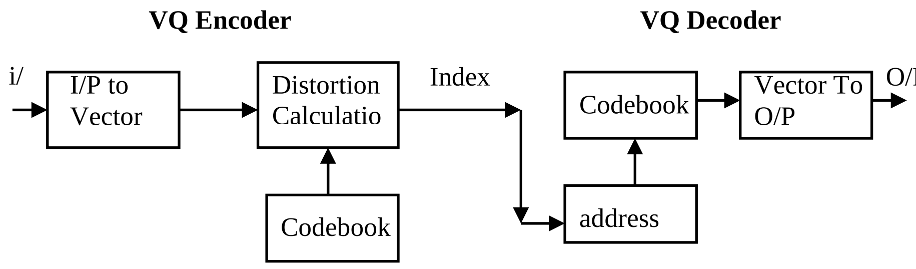


Encoder

- For each input pixel xij an error signal eij is generated using the prediction signal Pij based on weighted(by wi's) previously decoded pixels in proximity of x'ij
- Typically, Pij = w1x'ij-1 + w2x'i-1j-1 + w3x'i-1j
  and x'ij = Pij + qij
- The encoder and decoder use the same predictor
- For a highly correlated image Pij tracks xij and the error eij will be small and therefore offers higher rates of compression here.
- Less efficient than block based compression techniques in general
- Lower compression rates and more complex prediction when dealing with less predicatble data.

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



b.

i. (2marks)

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

ii. (1mark)

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

iii. (1mark)

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

c. 7 marks

Using row-column decomposition:
For first and fourth rows:
DCT0 = 1/√2 (4/√2 cos 0 + 1/√2 cos 0 + 1/√2 cos 0) = 3
DCT1 = 1/√2 (4 cos Π/8 + 1 cos 3Π/8 + 2 cos 7Π/8) = 1/√2 (0.924x4 + 0.383–0.924) =2.23
DCT2 = 1/√2 (4 cos Π/4 + 1 cos 3Π/4 + 1 cos 7Π/4) = 1/√2 (4x0.707 –0.707 + 0.707) =2
DCT3 = 1/√2 (4 cos 3Π/8 +1 cos 9Π/8 +1cos 21Π/8) = 1/√2 (4x0.383-0.924 -0.383)= 0.159

resulting in 1-D DCT matrix:

```
3   2.230   2   0.159
0     0     0     0
0     0     0     0
3   2.230   2   0.159
```

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/√2 (3/√2 cos 0 + 3/√2 cos 0) = 3
DCT1 = 1/√2 (3 cos Π/8 + 3cos 7Π/8) = 3/√2 (0.924 –0.924) =0
DCT2 = 1/√2 (3 cos Π/4 + 3 cos 7Π/4) = 3/√2 (0.707 +0.707) =3
DCT3 = 1/√2 (3 cos 3Π/8 + 3 cos 21Π/8) = 3/√2 (0.383 - 0.383) = 0
Therefore the 2-D DCT matrix in this case is :

```
3       0     3       0
2.23    0     2.23    0
2       0     2       0
0.159   0     0.159   0
```

d.

i. **3 marks**

After quantisation by above matrix and rounding the DCT transformed data becomes:

$$\begin{bmatrix} 3 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

giving a compression ration of 16:3

ii. 2marks

After DCT transformation still significant energy at the higher-frequencies, with coefficients having relative high-values, quantisation as above will result in significant loss of detail when recovering the data; expect important loss of quality after decoding.

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.

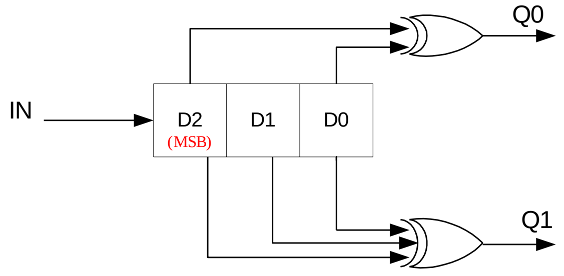
b) **(5)**
with $\boldsymbol{m(x)} = \alpha \boldsymbol{x}^2 = (\alpha, 0, 0\,)$: Append $2t=4$ consecutive zeros before the $k=3$ information symbols to get: $Cj=(0,0,0,0,\ \alpha,\ 0,\ ,0),\quad j=0,1,\ldots 6;$ then perform an inverse $GF(2^3)$ Fourier Transform on *Cj* to obtain the time-domain codeword *ci*, $i=0,1,..6$

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

$c0= C0\,\alpha^0 + C1\,\alpha^0 + C2\,\alpha^0 + C3\,\alpha^0 + C4\,\alpha^0 + C5\,\alpha^0 + C6\,\alpha^0 = C4\,\alpha 0 = \alpha$
$c1= C0\,\alpha^0 + C1\,\alpha^{-1} + C2\,\alpha^{-2} + C3\,\alpha^{-3} + C4\,\alpha^{-4} + C5\,\alpha^{-5} + C6\,\alpha^{-6} = C4\,\alpha^{-4} = \alpha^{-3} = \alpha^4$
$c2= C4\alpha^{-8} = 1$
$c3= C4\,\alpha^{-12} = \alpha^3$
$c4= C4\,\alpha^{-16} = \alpha^6$
$c5= C4\alpha^{-20} = \alpha^2$
$c6= C4\alpha^{-24} = \alpha^5$
Giving $c = (\alpha,\alpha^4, 1, \alpha^3, \alpha^6, \alpha^2, \alpha^5\,)$

c.

i) 3 marks



ii).(2 marks)
Encoded data sequence: ***00 11 10 00 10 01 00 10 01***

iii) 7 marks
Viterbi decoding performed; paths with more than 3 errors discarded

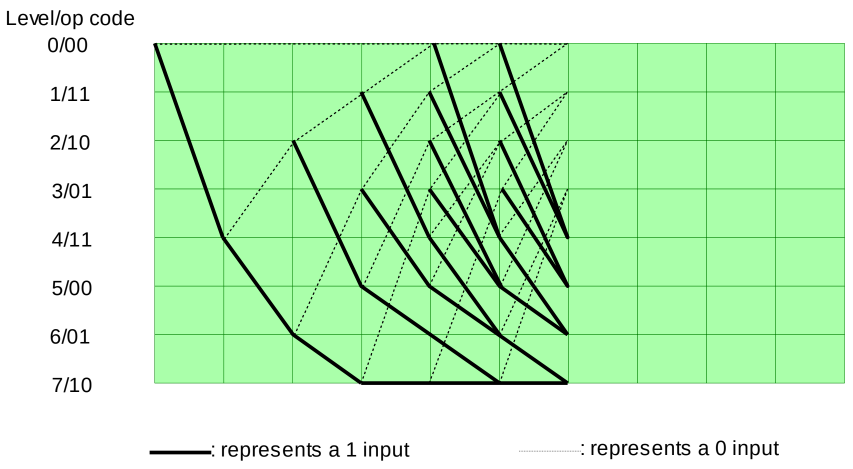


Figure Q.3

Depth	00	10	01	11	00	11	10	01	00
0=00	0	1	2	4*	3	5*	4*		
1=11			3	3	4*	3	4*	4*	5*
2=10		2	3	3	4*	3	3		
3=01				2	3	4*	5*	3	
4=11	2	1	2	3	5*	3	4*		
5=00			3	5*	2	5*	4*	4*	3
6=01		4*	1	4*	4*	3	5*		
7=10				2	3	4*	3	5*	

* : stop more than 3 errors

Corrected data is therefore:**011011101**