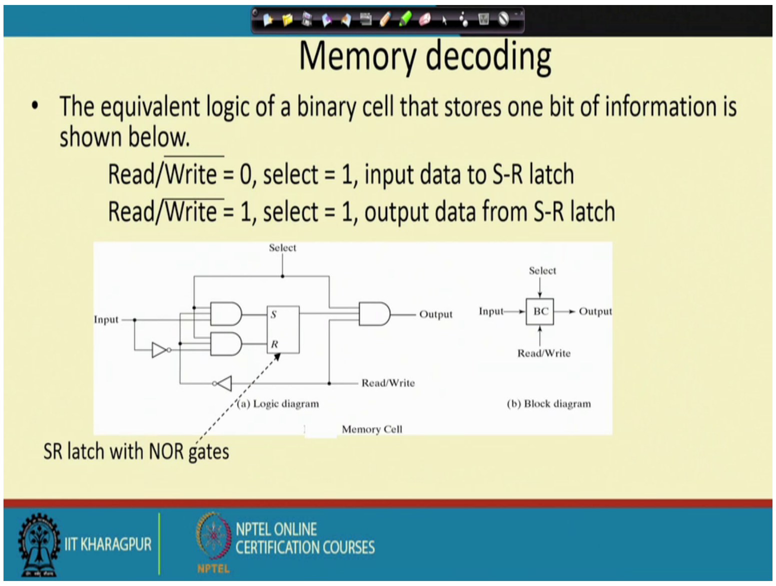
Digital Circuits

Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 42
Memory (Contd.)

The equivalent circuitry for a logic circuitry for a binary cell that stores 1 bit of information can be like this.

(Refer Slide Time: 00:19)



Memory decoding

- The equivalent logic of a binary cell that stores one bit of information is shown below.

  Read/$\overline{\text{Write}}$ = 0, select = 1, input data to S-R latch

  Read/$\overline{\text{Write}}$ = 1, select = 1, output data from S-R latch

So, this is as I said that this is just an equivalent circuitry, so actual ram cells are not like that. So in that case I have I told you that individual s ram cells they require only 6 transistors, but here you see that I have put an SR flip flop. So SR flip flop means you have got some 4 NAND gates so a 4 NAND gate each NAND gate takes a 4 transistor so that will take 16 transistors.

So, this SR flip flop itself will take 16 transistors, then we have got all this AND gates and etcetera inverter etcetera they will take more number of transistor. So this is just a logical diagram do not take it as the actual memory diagram, but from the digital logic point of view so this is an equivalency of equivalent of what is happening in the memory cell.

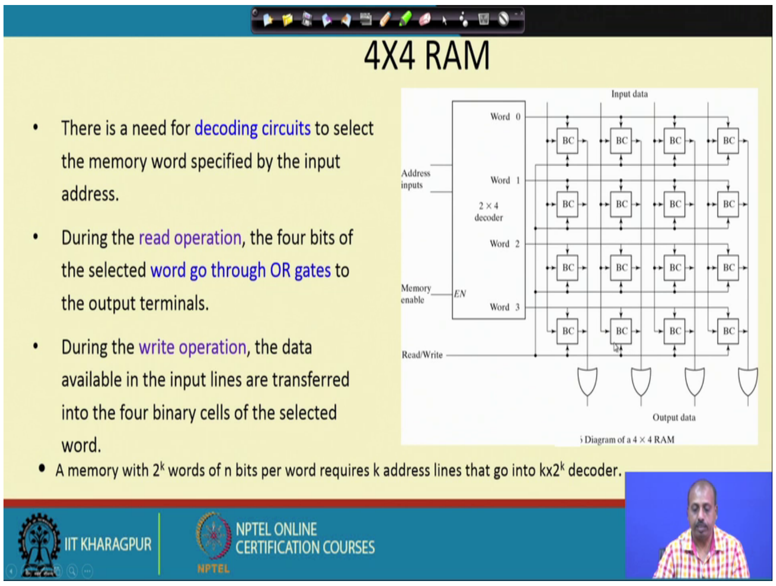
So, the equivalent logic of a binary cell that stores 1 bit of information is like this, that if read write bar is equal to 0 then it will select and select equal to 1, then the input data is put into the SR latch. So input is coming so if read write bar is 0; that means, we are trying to do a write operation. So a select line is actually the enable line of the of the memory chip you can say so at this location I am trying to write the value, so whatever is coming at the input they should be stored here. So if you look into this diagram so when this when this read write is equal to 0 read write bar is equal to 0.

So, this line so this and gate so it has got this select equal to 1, this read write bar equal to 1 and a input is whatever be the input value that will be coming here. And you see since this read write bar line is equal to 0 so since this is equal to 0, so this will be coming as this will be coming here. So this will be coming as a 1 here and the input bar will be coming to this point this reset point.

So, in the S I have got input and in the R I have got input bar. So as a result whatever be the value of the input that will get stored here so if input value is equal to 1 in that case I have got s equal to 1 and R equal to 0 as a result this flip flop will be set to 1. On the other hand if input value is equal to 0, then I will get here S equal to 0 and R equal to 1 as a result the value of this q output will be equal to 0.

So, this way this value input is getting stored in this SR latch, and then for the output part so we have got this if the select line is 1, then the whatever be the output that will go to a whatever be the out flip flop value here the latch value here. So that will come to the output provided this read line is made equal to one so that is a read operation. So, conceptually we can say that if this is the basic cell so this whole thing is the basic cell so it has got the inputs like select input, read write and output the output line and the output so that way this cell works. Of course, it should as I have already said that this is just a logical diagram not the physical one.

(Refer Slide Time: 03:36)



4X4 RAM

- There is a need for decoding circuits to select the memory word specified by the input address.
- During the read operation, the four bits of the selected word go through OR gates to the output terminals.
- During the write operation, the data available in the input lines are transferred into the four binary cells of the selected word.
- A memory with 2^k words of n bits per word requires k address lines that go into $k \times 2^k$ decoder.

Now, how can we construct a memory array? So I have talked about in the previous slide the design of this Basic Cell or BC.

Now, if you are suppose we are trying to construct a 4 by 4 ram that is there will be 4 locations or 4 address locations, and each location is consisting of 4 bits. So I have take 4 by 4 that is 16 basic cells arranged in a 4 by 4 fashion, and then there is a so this address lines, so this is there are 4 such memory locations so I will need 2 address lines so 2 address lines are coming here so, and I use a 2 to 4 decoder.

So this memory enable line is connected to the enable line of the decoder, so if this memory enable is equal to 0; if this memory enable is equal to 0 as a then all this decoders all this outputs will be 0 so none of them will be none of the location values will be available at the output. On the other hand if enable equal to 1 so depending upon this address lines depending upon this if enable is equal to 1 so, depending upon this address lines so one of this word lines will be selected.

Suppose for example, if the address line is say 0 1 then what will happen? So this will be 0, this will be 1. This will be 0 and this will be 0. So the content of this basic 4 basic cells so they will be available here; so the other otherwise since the select line is 0 so will be so this BC lines will be this basic cells outputs will be 0.

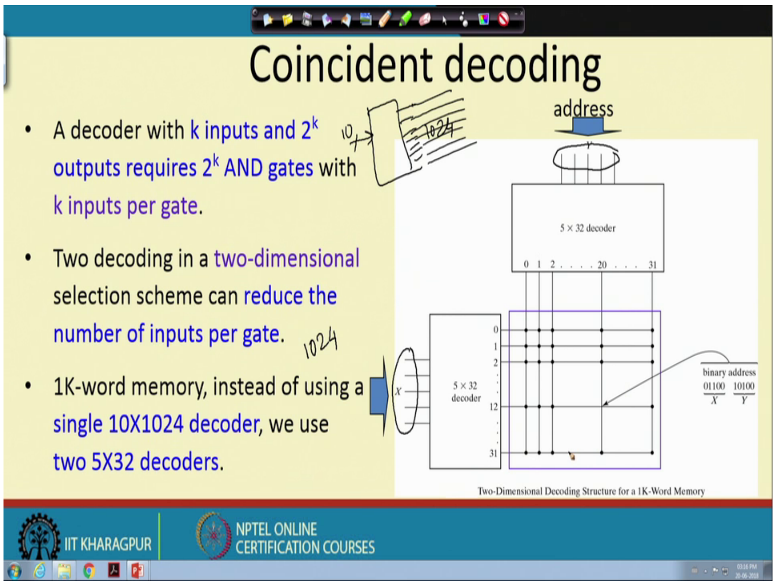
So, they are this basic cell outputs in column wise so they are ORed at this point so then you get the 4 bit output data. So we need some sort of decoding circuitry from the address lines so to select the appropriate row of this array. And then, this individual columns of this array so they are ORed to produce the individual the output part of from the chip and for the input part so they will be connected here so they will come to the basic cell. So there is a need for decoding the circuit there is a need for decoding circuits to select the memory words specified by the input address; during the read operation the 4 bits of the selected word go through the or gates to the output terminals as I explained.

And during the write operation, the data available on the input lines that is this points so they will be who available on the input lines and they transfer to the 4 binary cells that we have selected. And then so that way the whatever input data will be coming and whatever address line are given so based on that a particular row will get selected and this input data value will be stored in those binary cells.

So, a memory with 2 to the power k words of n bits per word will require k address lines, and that will go to a go into k 2 k by 2 to the power k decoder. So this decoding circuitry becomes a major concern because if you are having a large sized memory. So you have got a large number of rows in that memory and so this decoder has to have so many outputs. So this is a simple 2 to 4 decoder when you have got say 4 memory words.

So, if there are 1 kilobyte of 1 kilo memory word. Then this decoder size should be 10 by 2 to the power 10 so that way it will go on increasing significantly, exponentially the requirement of the decoder will go on so that is a major concern.

(Refer Slide Time: 07:06)



Coincident decoding

- A decoder with k inputs and 2^k outputs requires 2^k AND gates with k inputs per gate.
- Two decoding in a two-dimensional selection scheme can reduce the number of inputs per gate.
- 1K-word memory, instead of using a single 10X1024 decoder, we use two 5X32 decoders.

Two-Dimensional Decoding Structure for a 1K-Word Memory

So, to reduce the complexity of this decoder to some extent, so what is done? We have got some structure which is known as coincident decoding so, this coincident decoding is like this.

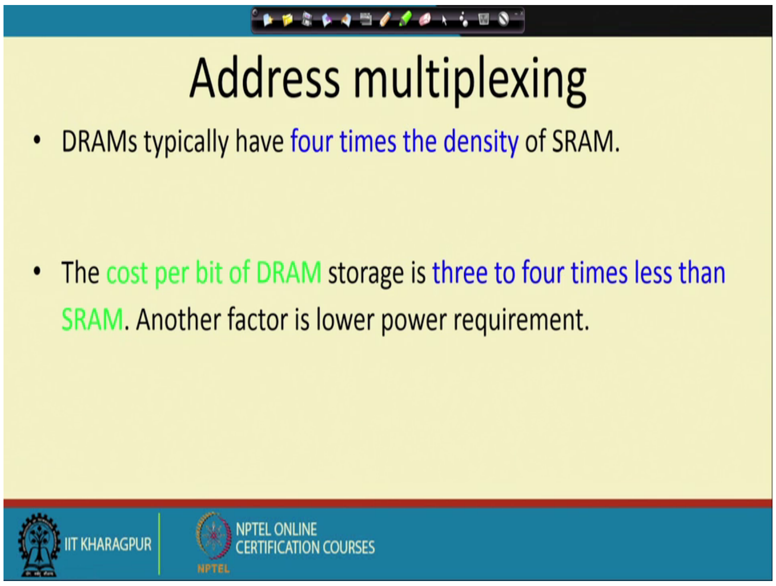
So a decoder with k inputs and 2 to the power k outputs require 2 to the power k AND gates and k inputs per gate, so this is a standard design of the decoder that we have seen previously. The 2 decoding in a 2 dimensional selection scheme can reduce the number of inputs per gate. So what you have to so in this particular case suppose I have got a memory array that has got 1 kilo word of memory.

So, 1 kilo word of memory so that is there are 1024 locations so 1 kilo word means there are 1024 locations. So if you are thinking about a simple design, then I should have a decoder like this that has got 10 number of inputs and this 1024 number of outputs; so individual outputs going to individual arrays individual rows of that cell array.

However, you can do it like this, so you can divide this 10 lines into a row address and column address part. So first you put this put this so this x and y are divided 5 bit 5 bit they are divided and then so to take 5 by 32 decoder, so it selects this one of this 32 locations, one of this 32 rows and then we have got a 5 by 32 decoder on the column side that will select again 5 by 32 5 by that that will again select one of the 32 columns.

So, this is the 2 dimensional organization of 1 kilo word of memory, that way reducing the decoder complexity.

(Refer Slide Time: 09:03)



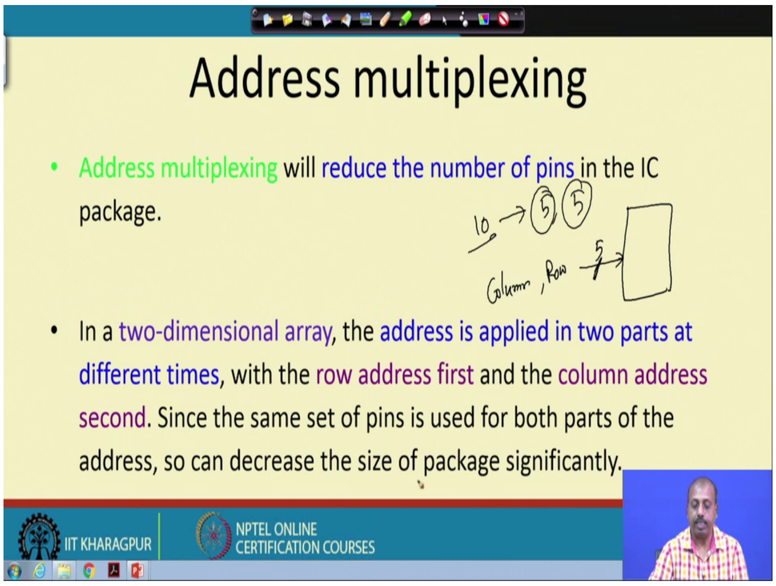
Address multiplexing

- DRAMs typically have four times the density of SRAM.
- The cost per bit of DRAM storage is three to four times less than SRAM. Another factor is lower power requirement.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, normally these DRAMs they have typically 4 times the density of SRAM; so this DRAM density is pretty high and the cost per bit or DRAM storage is 3 to 4 times less than SRAM. So this is one of the basic advantage of DRAM over SRAM so the cost per bit is less and other factor is the power requirements; so DRAM power consumption is also low compared to this SRAM.

(Refer Slide Time: 09:31)



Address multiplexing

- Address multiplexing will reduce the number of pins in the IC package.
- In a two-dimensional array, the address is applied in two parts at different times, with the row address first and the column address second. Since the same set of pins is used for both parts of the address, so can decrease the size of package significantly.

$10 \rightarrow \textcircled{5}\ \textcircled{5}$

Column, Row $\xrightarrow{5}$ $\boxed{}$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

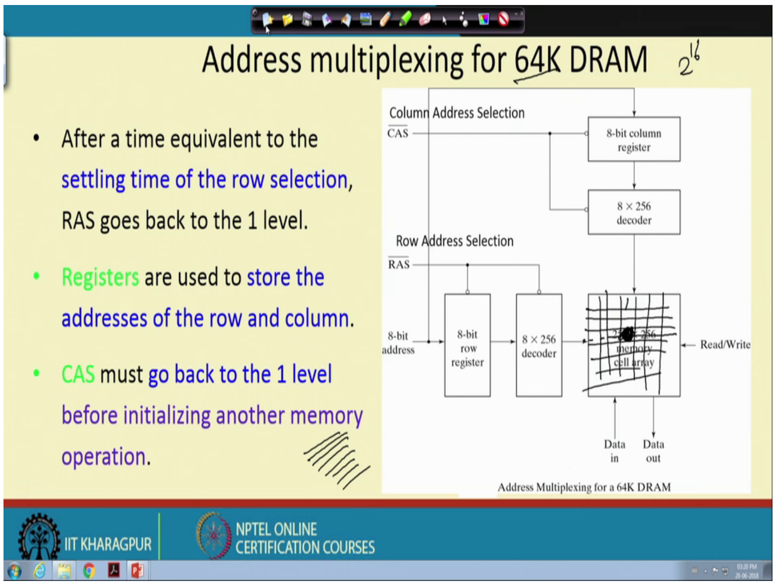
Address multiplexing technique so that that can reduce the number of pins in the IC package. Because this what you can do so as I said that the total 10 bit address is divided into 2 parts 5 and 5 bit and 5 bit so 5 bit row address and 5 bit column address.

The address what we can do instead of by applying this 10 address bits simultaneously, we can apply 5 bit address first and then again 5 bit address. So if I have got this memory chip and then I can have a 5 bit address lines instead of 10 bit, first I apply the row part of the address and then after the row has been selected, we can apply the column part of the address.

So, we can we can divided into we can divide the address part into row part and column part and apply them one after the other. So the advantage that we get is if you have to apply 10 bit address, then this memory should have 10 address lines. So instead of that I should have only 5 address lines and some extra control by which I can tell whether it is the row address or a column address. So in a 2 dimensional array the address is applied in 2 parts at a different times with the row address first and column address second.

Since the same set of pins is used for both the parts of the address, we can decrease the size of the package significantly.

(Refer Slide Time: 10:57)



Address multiplexing for 64K DRAM 2^{16}

- After a time equivalent to the settling time of the row selection, RAS goes back to the 1 level.
- Registers are used to store the addresses of the row and column.
- CAS must go back to the 1 level before initializing another memory operation.

Address Multiplexing for a 64K DRAM

So, how is it done we will see in the next slide. So what is done so I have got a 26 by 256 memory cell array, so initially the so the 8 bit address is given. So 8 bit address is stored in this 8 bit row register and then from this row register it goes to an 8 by 256 decoder.

So, that way this particular row is selected and then we apply the column part of the address. So a same 8 bit address lines is used to feed the column part of the address to the memory chip. So this is giving me this is giving me this is actually selecting the corresponding column; so this is give this is stored in the 8 bit column register, then it goes to 8 by 256 decoder and then it selects one of the 256 alternative columns.

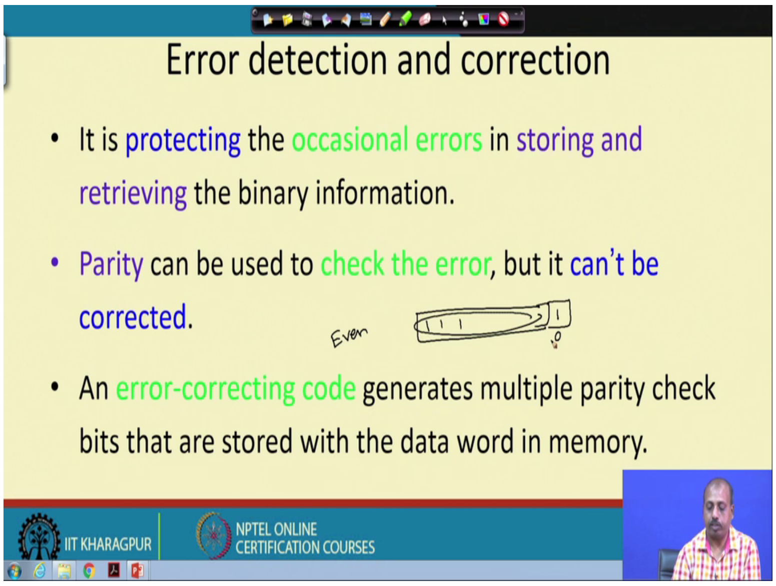
So, this memory that we have here is 64 k sorry so memory that we have here is 64 k so that is 2 to the power 16 locations should be there. So what we have done, this 2 to the power 16 locations instead of organizing them in a serial fashion so what we have done? We have organize them in 2 dimensional fashion so such that in each so we have got 256 such rows and 256 such columns. So once the row selection has been done, so this row has been selected then we select a particular column say we select say this particular column then ultimately so this location gets selected. And this location itself may be your an 8 bit location or 64 bit or 16 bit location depending on the word size of the memory, but um this particular memory cell will get selected. So that way I do not need to apply a 16 bit address simultaneously, I apply 8 bit at a time and it will be reducing the complexity in term in the number of pins.

However, I need to tell like when I am applying the row address and when I am applying the column address. So for that purpose so there are 2 special pins RAS bar and CAS bar; so load address or row address selection and column address selection. When I am giving the row address this RAS bar line should be made equal to 0, so that the value the row address is stored latched on to this row address register. And this decoder can enable the corresponding row and then this column address after that the column address is put and then at that time this CAS bar line is made low.

And then the value that I have that is available on this 8 bit register is a latch 8 bit address is latched on to this 8 bit column register, and then it goes to this decoder and by it selects a one of the 256 columns. So that way finally, one particular memory location or memory word gets selected. So this CAS must go back to the 1 level before initializing another memory operations, so that has to be that is the requirement that first this column selection and the row selection will be done.

And it must go back to 1 and then only it can become 0 or any, to ensure that the operation is done properly.

(Refer Slide Time: 14:18)



Error detection and correction

- It is protecting the occasional errors in storing and retrieving the binary information.
- Parity can be used to check the error, but it can't be corrected.
- An error-correcting code generates multiple parity check bits that are stored with the data word in memory.

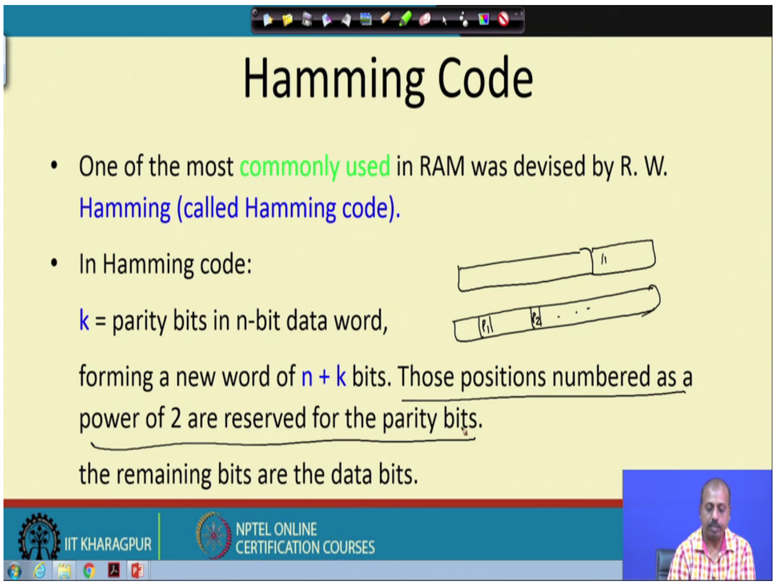
Next we will be discussing about another very important topic, which is known as error detection and correction, like when you are transferring data or storing data in the memory locations. So it since the data is coming from outside the memory, so it may so happen that in the in the transfer process some bits got corrupted ok.

So, how to detect that situation and what to do like if we see that some bits have got corrupted? So this error detection and correction mechanism so they are for protecting the occasional errors in storing and retrieving the binary information. So one possibility is the parity; so parity can be used to check the error so we have seen parity separately previously where if I have got say an 8 bit data, then I can just add another bit parity.

So, I count the number of ones here and if the if I say that finally, I should have an even parity, then if the number of ones in this part is ORed then this bit will be turned 1 on the other hand if this is even then this bit will be turned 0. So that we have seen in the x and you know that this can be done very easily using the some XOR circuit. Now this is so parity can check the errors so it can detect the error, but correct it.

So, this error correcting codes so they can generate multiple parity check bits that are stored with the data word in memory and that can lead to a better their possible correction of the this corruption. So we will see how this is done.

(Refer Slide Time: 15:56)



Hamming Code

- One of the most commonly used in RAM was devised by R. W. Hamming (called Hamming code).
- In Hamming code:

  k = parity bits in n-bit data word,

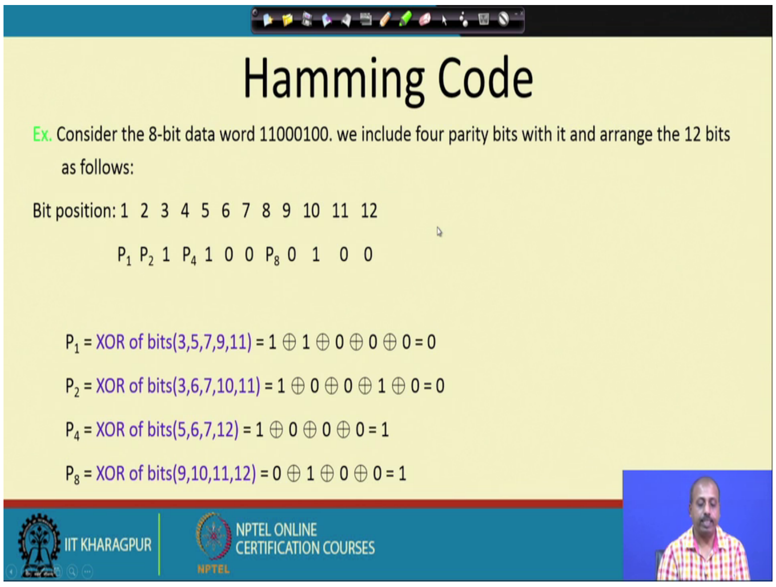
  forming a new word of n + k bits. Those positions numbered as a power of 2 are reserved for the parity bits.

  the remaining bits are the data bits.

So, one commonly used code for this error detection and correction is the hamming code. So this is this was in way divided by R W Hamming and that is why it is called Hamming code so it is say in a hamming code so for n bit of a data word we have got k parity bits ok.

So, for n bit data there are k parity bits and these k parity bits are attached with this n data bits. So total number of a total word size becomes n plus k bits and these positions are numbered as a power of 2 are the positions which are powered of power of 2 they are reserved for the parity bits. So it is not that I have got an n bit data and this k bits are computed. And they are just attaching a so it is not done like that but this bits are actually inserted into the total n plus k bit may be I have got 1 bit here, another so the 1 parity bit P 1 here another parity bit P 2 there so like that. So it says that the positions are numbered as a power of 2 are reserved for the parity bits while the remaining bits are the data bits. So that is how this organ organization is done.

(Refer Slide Time: 17:11)



Hamming Code

Ex. Consider the 8-bit data word 11000100. we include four parity bits with it and arrange the 12 bits as follows:

Bit position: 1 2 3 4 5 6 7 8 9 10 11 12

P_1 P_2 1 P_4 1 0 0 P_8 0 1 0 0

P_1 = XOR of bits(3,5,7,9,11) = $1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0$

P_2 = XOR of bits(3,6,7,10,11) = $1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0$

P_4 = XOR of bits(5,6,7,12) = $1 \oplus 0 \oplus 0 \oplus 0 = 1$

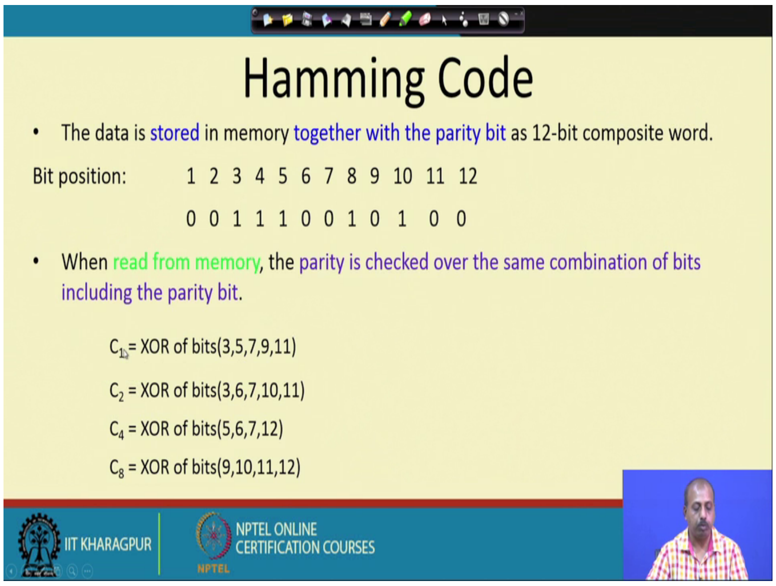
P_8 = XOR of bits(9,10,11,12) = $0 \oplus 1 \oplus 0 \oplus 0 = 1$

So let us take an example; suppose we have got an 8 bit data word 1 1 0 0 0 1 0 0 and we want to include 4 parity bits and then we get a 12 bit pattern. So this is the 12 bit pattern now this bit positions are the since this is a power of the so this is a bit position 1 that is 2 to the power 0, this is bit position 2 so 2 to the power 1, so this is bit position 4 which is 2 to the power 2 so, at this powers of 2 so we put the parity bits.

So P 1 is put as bit 1, P 2 is put as bit 2, P 4 is put as bit 4, and the remaining bit 1 1 0 0 so they are coming here this is 1 this is 1 0 0 then 0 at bit number 9 so that way it goes and how to compute this P 1? So this is fixed so P 1 is given by XOR of bit 3 5 7 9 and 11, so if you do this the bit is turning out to be 0, P 2 is given by XOR of 3 6 7 10 and 11 so this is 0.

So, this XORing pattern is fixed, so P 4 is the XOR of 5 6 5 6 7 12 so that becomes equal to 1. So this way this P 1 P 2 P 4 and P 8 are calculated and once this is done, the data is stored in the memory together with the parity bit as 12 bit composite word. So this is the thing that is stored in the stored in the memory so 12 bit word that is node, now when you are reading the data from memory so we get this 12 bits. And then we check whether the parity bits are correct or not. If there is an error then the parity bit this parity bits will differ. We compute the check bits based on the similar formula like P 1 was done computed using XOR of 3 5 7 9 11.

(Refer Slide Time: 19:03)



Hamming Code

- The data is stored in memory together with the parity bit as 12-bit composite word.

Bit position: 1 2 3 4 5 6 7 8 9 10 11 12

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

- When read from memory, the parity is checked over the same combination of bits including the parity bit.

C_1 = XOR of bits(3,5,7,9,11)

C_2 = XOR of bits(3,6,7,10,11)

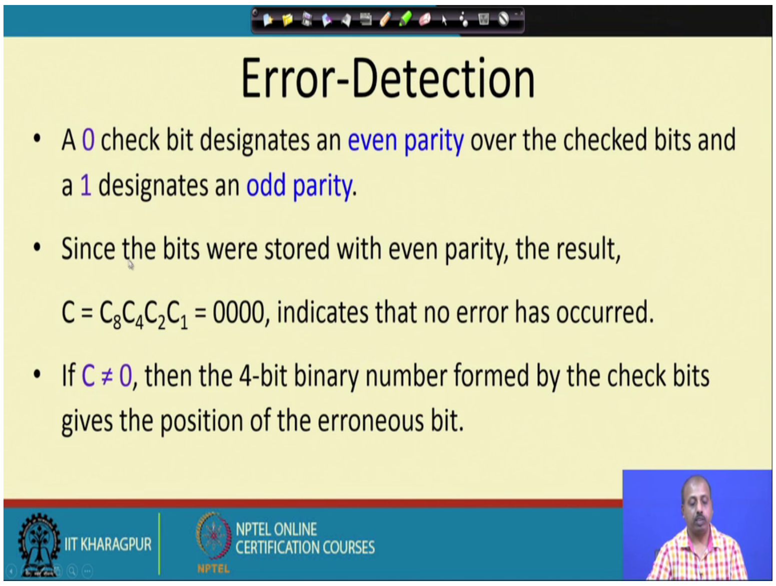
C_4 = XOR of bits(5,6,7,12)

C_8 = XOR of bits(9,10,11,12)

So, here we compute C 1 which is the XOR of the red values of a 3 5 7 9 11. Now if P 1 differs from C one; that means, there is some error in any of these bits.

Similarly, if C 2 differs from P 2 there is some error in this bit. So if your P 1 and P 2 both are creating a P if C 1 and C 2 both are not conforming with P 1, then you can say they are not conforming with P 1 and P 2 then you know that possibly there is an error in bit number 3, 7 or 11. So this way you can figure out a few locations so which are which may be wrong.

(Refer Slide Time: 19:43)



Error-Detection

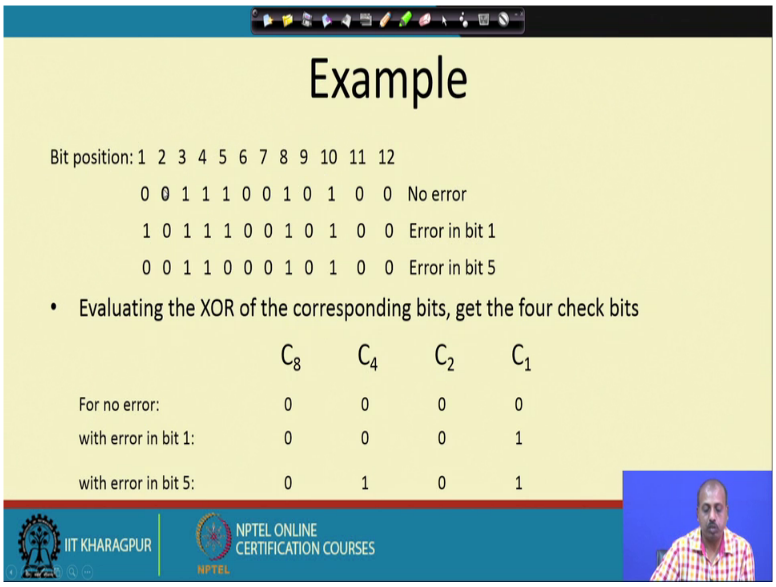
- A 0 check bit designates an even parity over the checked bits and a 1 designates an odd parity.
- Since the bits were stored with even parity, the result,

  $C = C_8C_4C_2C_1 = 0000$, indicates that no error has occurred.
- If $C \neq 0$, then the 4-bit binary number formed by the check bits gives the position of the erroneous bit.

So the how do you do a check? So a 0 check a 0 check bit designates an even parity over the check bits, and 1 designates an odd parity. And since these bits was stored with even parity, the results C indicates that there is no error. So since we have followed even parity so the if it is if this C 1 C 2 C4 C 8 is 0 0 0 0 there is no error, but if C is not equal to 0; that means, the 4 bit number formed by the check bits gives the position of the erroneous bit.

So, this so that this patterns have been selected in such a way this positions have been selected in such a way that this property holds that, the 4 bit number that you get here gives the position of the erroneous bit.

(Refer Slide Time: 20:31)



Example

Bit position:	1	2	3	4	5	6	7	8	9	10	11	12	
	0	0	1	1	1	0	0	1	0	1	0	0	No error
	1	0	1	1	1	0	0	1	0	1	0	0	Error in bit 1
	0	0	1	1	0	0	0	1	0	1	0	0	Error in bit 5

- Evaluating the XOR of the corresponding bits, get the four check bits

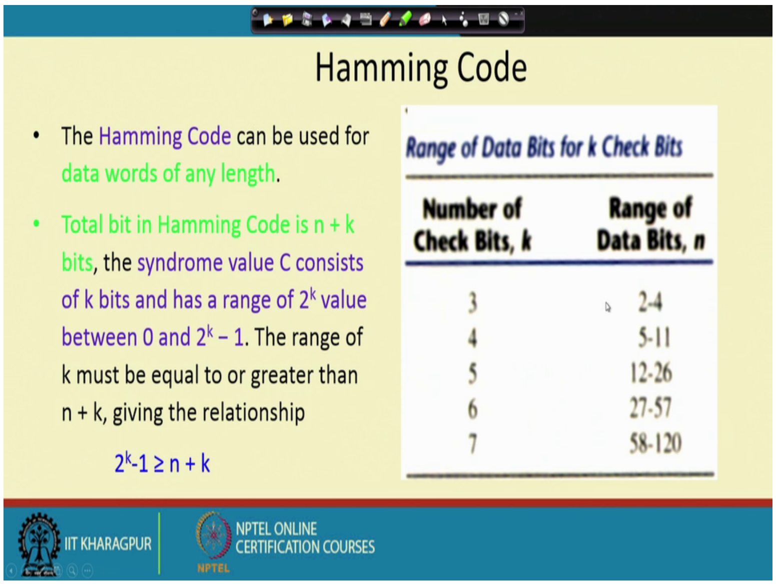
	C_8	C_4	C_2	C_1
For no error:	0	0	0	0
with error in bit 1:	0	0	0	1
with error in bit 5:	0	1	0	1

We will take an example, suppose this is the bit that we have this is the so this is the no error so this is the actual one. Now if there is a error in bit number 1, if there is an error in bit number 1 that is this bit has got transform from 0 to 1, but rest of the bits are not.

Then you will get the this when you compute this C 1 C 2 C 4 C 8 you will get this particular pattern, so that tells the bit number 1 there is some problem. On the other hand if there is an error in bit number 5; so bit number 5 was 1 here it has become 0 there, and rest of the bits are then after you have computed C 1 C 2 C 4 C 8 you will find that the pattern coming is 0 1 0 1. So this identifies the bit at which the error has occurred so you can just flip that particular bit on the received pattern, and then you can get the corrected version of the bit.

So, this way that that is the beauty of this hamming code, so it is XORing in such a fashion that this when you check it, so you can figure out like which particular bit has created problem.

(Refer Slide Time: 21:42)



Hamming Code

- The Hamming Code can be used for data words of any length.
- Total bit in Hamming Code is n + k bits, the syndrome value C consists of k bits and has a range of 2^k value between 0 and $2^k - 1$. The range of k must be equal to or greater than n + k, giving the relationship

  $2^k-1 \geq n + k$

Range of Data Bits for k Check Bits

Number of Check Bits, *k*	Range of Data Bits, *n*
3	2-4
4	5-11
5	12-26
6	27-57
7	58-120

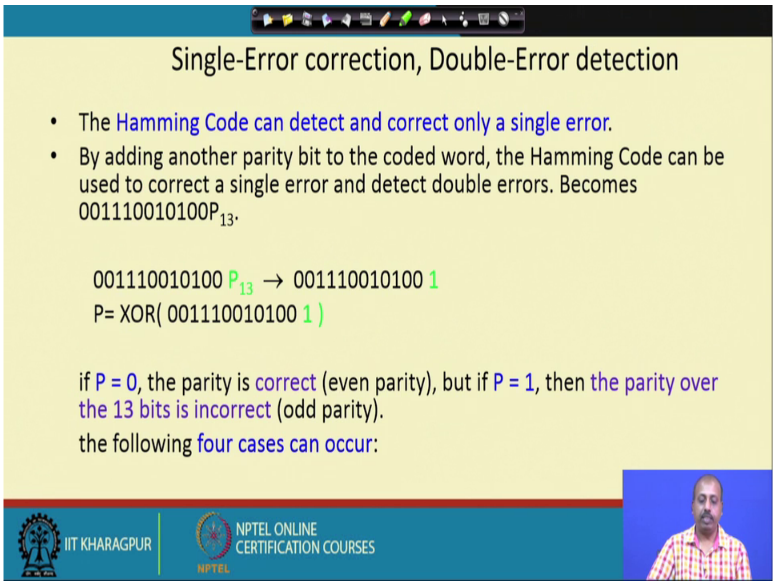
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, the hamming code can be used for data words of any length, and total bit in a hamming code should be n plus k bits, the an syndrome value C consists of k bits and has a range of 2 to the power k values between 0 to 2 to the power k minus 1.

And range of k much be equal to a larger than n plus k so this is the requirement. So if you are using an n bit code and a k bit k bit k extra bits or check bits, then the requirement is that 2 to the power k minus 1 should be greater or equal n plus k, then only this hamming code will work. For example, if number of check bits is 3, then the data bits that you can that that you can be in the range of 2 to 4.

So, if you have got data bit of length more than 4, then 3 check bits are not sufficient. If you have got 5 to 11 data bits, then you should have check bit k equal to 4 4 check bits will be necessary. So this is coming from the theory of hamming code so we do not have a spoke to discuss on that, maybe you can look into some coding theory course which will be talking about this hamming code in more detail.

(Refer Slide Time: 22:55)



Single-Error correction, Double-Error detection

- The Hamming Code can detect and correct only a single error.
- By adding another parity bit to the coded word, the Hamming Code can be used to correct a single error and detect double errors. Becomes $001110010100P_{13}$.

$001110010100\ P_{13} \rightarrow 001110010100\ 1$

P= XOR(001110010100 1)

if P = 0, the parity is correct (even parity), but if P = 1, then the parity over the 13 bits is incorrect (odd parity).

the following four cases can occur:

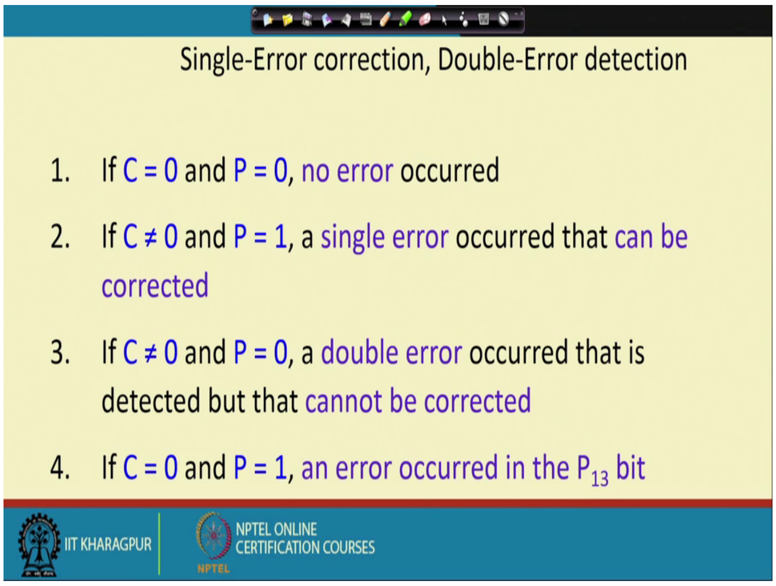
So, if you are trying to detect single error so single error and correct it, so this hamming code can detect and correct only a single error. So if you if you want to have this situation single error correction and double error detection like as I have said that in the this pattern that the that check pattern that we generated was such that it identifies one particular location as erroneous.

Now, if there are multiple bits which are erroneous, then this check will not able to pin point to the correct errors. So it is just an assumption that in a transmission or a at most one error has occurred so hamming code can correct it. So but if you want to detect the situation in which if there are multiple if there are multiple bits which are wrong, so you should be able to detect it, but you may be happy if correcting only a single bit.

So, single bit error correction and double bit error detection if you are looking into that, so for that purpose we have to add another parity bit and to the coded word, the so the we add this word P 13 here, so P 13 is connected here. And then P 13 is so this P 13 is computed here, so P 13 is becoming a is 1 at this point, so that is by means of taking XOR of all these bits.

So, this P equal to XOR of this with 1, so that naturally so if you take XOR so if P equal to 0. That means, the parity is correct even parity, but if P equal to 1 then the parity over this 13 bits is incorrect the odd parity.

(Refer Slide Time: 24:36)


Single-Error correction, Double-Error detection

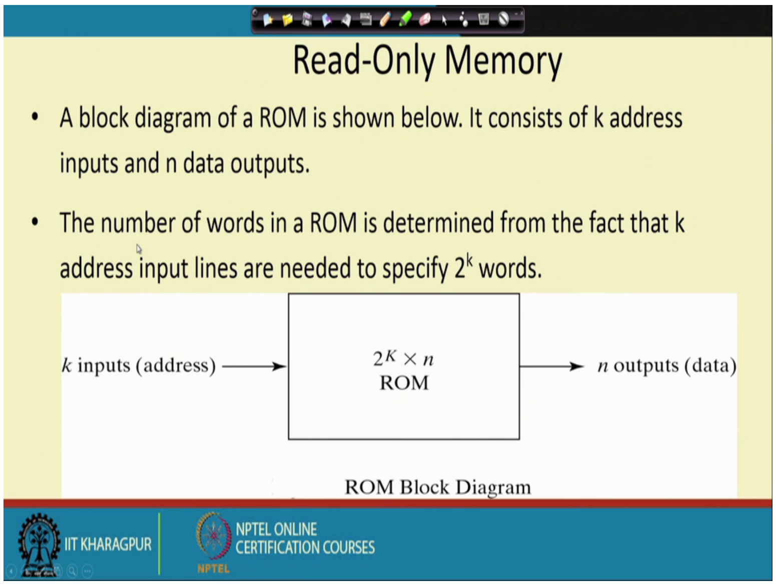
1. If $C = 0$ and $P = 0$, no error occurred
2. If $C \neq 0$ and $P = 1$, a single error occurred that can be corrected
3. If $C \neq 0$ and $P = 0$, a double error occurred that is detected but that cannot be corrected
4. If $C = 0$ and $P = 1$, an error occurred in the P_{13} bit

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

So, in that case we can have different situations C equal to 0, P equal to 0 no error has occurred; C naught equal to 0 P equal to 1 a single error has occurred and that can be corrected.

If C C naught equal to 0 and P naught P equal to 0, then double error has occurred it is that can be detected, but cannot be corrected if C equal to 0 and P equal to 1 and error occurred in the P 13 bit. So these are the inferences from this C and P bits. Again it is not possible to proof these results in our class so you I will suggest that you look into some discussion on this coding theory, for getting more details about this type of coding.

(Refer Slide Time: 25:17)

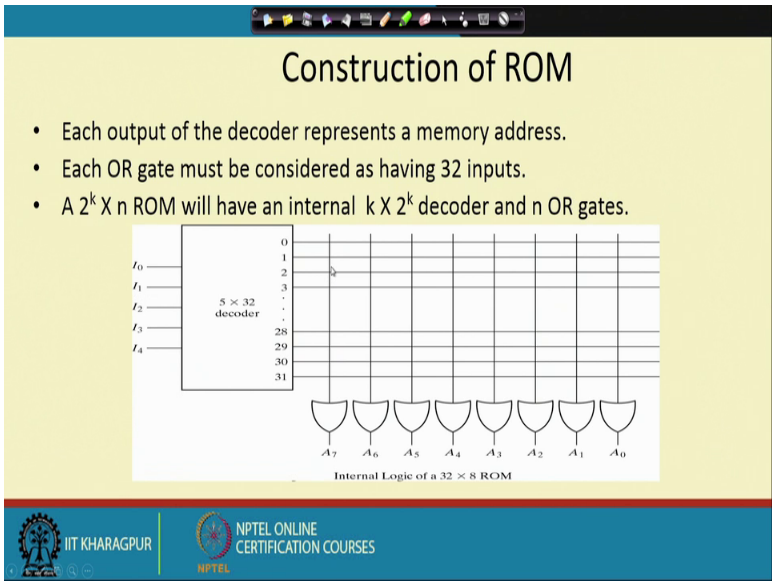


Read-Only Memory

- A block diagram of a ROM is shown below. It consists of k address inputs and n data outputs.
- The number of words in a ROM is determined from the fact that k address input lines are needed to specify 2^k words.

Next we look into read only memory. So read only memory so if the block diagram is like this so you have got k input lines, and we have got n output lines. So this the number of words in a ROM is determined from the fact that the k input address lines are needed to specify 2 to the power k address 2, to the power k words. So 2 to the power k words so are there so and a k bit input address is necessary then there are n output lines from there.

(Refer Slide Time: 25:47)



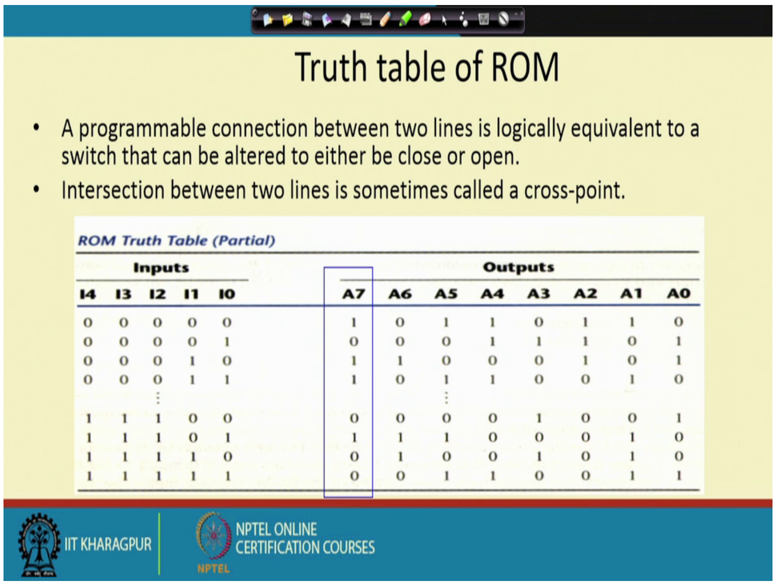
Construction of ROM

- Each output of the decoder represents a memory address.
- Each OR gate must be considered as having 32 inputs.
- A 2^k X n ROM will have an internal k X 2^k decoder and n OR gates.

Now, how do you construct a ROM? So logically a ROM is constructed like this that, each output of the so we have we take a 5 to 32 decoder, so if I have got say their 32 location ROM that we want to make. So we take this a 5 to 32 decoder address lines they come to this I 0 I 1 I 2 I 3 I 4 and then this 32 lines are coming out of the decoder.

Now, we have got this columns and this columns are summed in the OR gates so each OR gate is considered is having 32 inputs, so 2 to the power k by n ROM will have an internal 2 by k by 2 to the power k decoder and n such OR gates. But essentially so whatever design that we have shown here is just a logical diagram physically it is never done like this, but logically you can think of it that these are the components in the ROM.

(Refer Slide Time: 26:51)



Truth table of ROM

- A programmable connection between two lines is logically equivalent to a switch that can be altered to either be close or open.
- Intersection between two lines is sometimes called a cross-point.

ROM Truth Table (Partial)

Inputs					Outputs							
I4	I3	I2	I1	I0	A7	A6	A5	A4	A3	A2	A1	A0
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
		⋮					⋮					
1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, you can program the ROM like say a programmable connection between 2 lines is logically equivalent to a switch that can be altered to be either closed or opened. So for example, we can say that say this for this I 0 so if this address is 0 0 0 0 0 at that location this is the output pattern that I expect similarly, at say 1 1 1 0 0 so this is the input pattern that I expect.

So if we have got separate if we if in this in this point so you have got some connecting switches at these junctions, we have got connecting switches then those switches can be programmed in this fashion, so that it can talk about it can store the corresponding pattern that we need for the ROM location. So this can be done by means of putting some transistors and that will act as switch. And, there are switch the transistors can be programmed to either 1 or 0 so that the location content becomes 1 or 0.