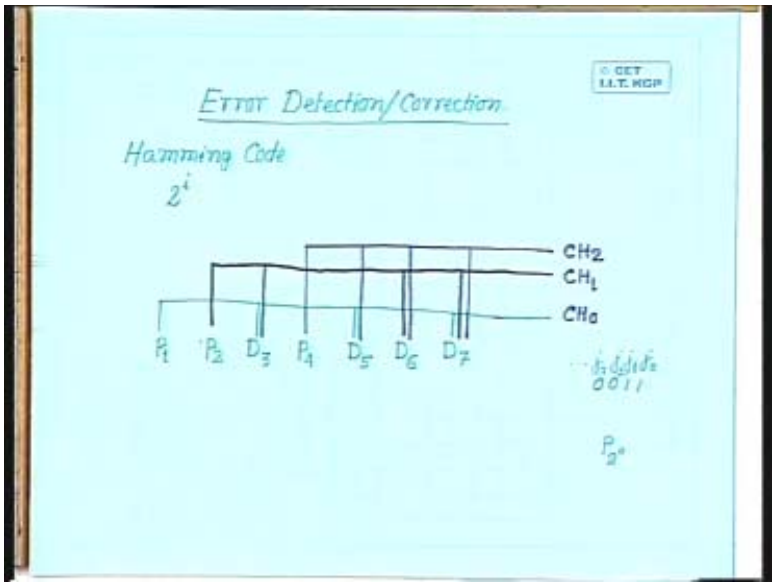
Digital Computer Organization
Prof. P. K. Biswas
Department of Electronic and Electrical Communication Engineering
Indian Institute of Technology Kharagpur
Lecture No. # 28
Error Detection and Correction

Another important consideration for I/O interfacing is error detection and correction because most of the I/O devices are connected over the serial device and if the data is transmitted at a high rate in that case it is quite likely that the data which will be read by the CPU will be erroneous. So instead of giving the erroneous data it will be better that if you can detect, if there is any error in the data that has been read from the device and if possible you correct the data.

(Refer Slide Time: 00:01:35 min)



So there are different error detection and correction mechanisms and particularly error detection mechanism and one of the mechanisms that you already know is the parity bit. So by making use of the parity bit, you can determine whether the data has been read is correct or there is some error in the data but the difficulty with using parity bit you know that you can simply detect the error but you cannot correct it. So another kind of coding technique which can be used for detection as well as correcting the error bit is what is called a hamming code. Do you know what is the Hamming code? Then let me roughly say what is the hamming code.

The Hamming code is something like this. Whenever you have a data stream, what you do is into the data stream you introduce additional bits which are nothing but parity bits but in this case the parity bits are generated in slightly different way than how it is generated in case of parity code. So in case of hamming code it is something like this that every bit position in the bitstream, where the position is of the form 2^i, for some integer I, will contain a parity bit. So I can take an example like this. First bit position, bit position 1 is of the 2^i because 2^0 gives you 1.

That means the first bit location will be a parity bit, it will not contain a data bit. So first bit location I will put as a parity bit P_1. Come to the second bit location 2, which is again 2 to the power 1 so that will also contain a parity bit that I will call as P_2. Third location it is not of the form 2^i, so that will contain a data bit and let me put it as D_3 where the subscript indicates the bit position. Fourth location which is 2^2 again will contain a parity bit, I will put this as P_4. Fifth location which is not 2^i, will contain a data bit that is D_5. Sixth location again it is not of the form 2^i will contain a data bit, say it is D_6.

Similarly, 7^{th} bit location will contain D_7 because it is not 2^i, 8^{th} bit location again 2^3 that will again contain a parity bit. So this way it will continue. Now D_3, D_5, D_6, D_7 they are coming from the bit streams or data bit streams. I have to generate $P_1\ P_2\ P_4$, so let me take this small example. I have to generate P_1, P_2 and P_4. Now how do I generate P_1? While generating P_1, I consider the binary representation of the bit locations. So in this case the binary representation of D_3 is 0, 0, 1, 1 something like this. So when I consider P_1 you consider that P_1 is 2^0. So for generating this P_2^o, what is the zeroth bit in the binary representation of the location of the data? If that is equal to 1 then all those data bits will take part along with the parity bit in generation of the parity bit. Say for example here this binary representation of D_3 is if I put it in the form of 4-bit binary number 0, 0, 1, 1. This is the zeroth bit in the binary representation of D_3 and zeroth bit is equal to 1.

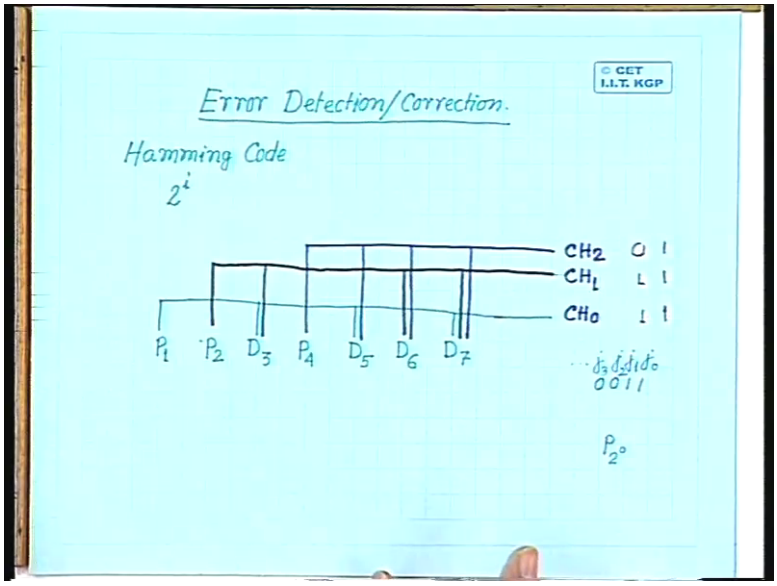
So for the generation of P_2^0, I will consider all those data bits where the binary representation of the position of the data bit, in the binary representation of the position of the data bit, the zeroth bit is equal to 1. See when I represent the position of any data bit in the form of a binary number that will contain a number of positional bits. I put this as let me call this as j_0, j_1, j_2, j_3 and so on. When I am coding this P_1, I am generating P_1. P_1 is nothing but P of 2 to the power 0. So I will consider all those data bits for generation of P_1 where for which j_0 equal to 1. So what are the bits for which j_0 equal to 1? For D_3, j_0 equal to 1, for D_5 j_0 equal to 1, for D_7 j_0 equal to 1. So these bits D_3, D_5 and D_7 along with P_1 will take part in generation of P_1. So let me put it this way, D_3 D_5 and D_7 all these bits take part in generation of P_1. When I generate P_2, P_2 is P_2^1. P_1 will also take part say P_1 will take part, even for generation. See what I need is suppose I decide that my parity will be odd parity then odd parity will include P_1 D_3 D_5 D_7. Whichever way we generate but when I talk about parity, parity includes parity bit. Including the parity bit what parity you are maintaining?

Now if on the receiver side I will receive this entire bit stream P_1 P_2 D_3 D P_4 D_5 D_6 D_7. Any of the bits can be an error, even P_1 can be an error. So if I don't consider P_1 in generation of the parity bit, on the receiver side I cannot detect whether the P_1 is in error. If you consider only D_3 D_5 D_7 and without considering P_1 because at the receiver end I don't know that whether P_1 is in error or D_3 is in error or D_5 is in error or D_7 is in error, even P_1 can also be in error. This one you are talking about? What is bit position of D_3 that is the third position, it is the position which I am representing ... If it is odd parity P_1 XOR D_3 XOR D_5 XOR D_7 should generate 1. You mean that P_1 has to be 1. Let us see.

Similarly, when I try to generate P_2, I will take all the bits where the bit position when represented in the form of a binary number there j_1 has to be equal to 1 because P_2 is, P_2 to the power 1. So you find that in all these bits, what are the bits for which $j_1 = 1$? For D_3, D_3 j_1 equal to 1, D_5 no. D_6 again j_1 equal to 1, D_7 j_1 equal to 1. So I will consider these bits, D_3 I will

consider D_6 and I will consider D_7 then when I generate P_4 that is P_2^2. So I will consider all those data bits for which $j_2 = 1$. So what are those bits for which j_2 is equal to 1? It is 4 that you are generating so that has to come into picture then D_5, D_6 and D_7. So I will call this as say check 0, I will call this as check 1, I will call this as check 2. So if I want odd parity in that case when I generate P_1 my requirement should be P_1 XOR D_3 XOR D_5 XOR D_7 that should be equal to 1. If I want even parity, then P_1 XOR D_3 XOR D_5 XOR $D_7 = 0$.

(Refer Slide Time: 16:45)



Similarly, when I generate P_2 and if I want odd parity, of course if I want odd parity for P_1 I should have odd parity for P_2 also. So if I want P_2 and I decide for odd parity then P_2 XOR with D_3 XOR with D_6 XOR with D_7 that should be equal to 1. Similarly, for P_4, it is P_4 XOR D_5 XOR D_6 XOR D_7 that should be equal to 1. So this equation will tell me that what is P_1, what is P_2 and what is P_4. I have equation P_1 XOR D_3 XOR D_5 XOR D_7 that should be equal to 1. Now you find that all this bits D_3 D_5 and D_7 they are known, only unknown is P_1. I don't have any other unknown term. So this way I can generate P_1, P_2 and P_4.

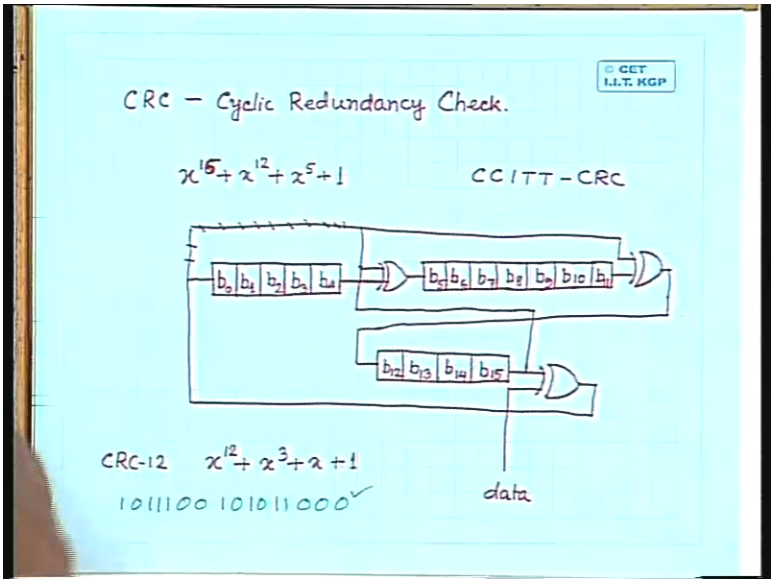
Now when I transmit it on the receiver side, you perform similar operation and see whether the parity is maintained or not. So on the receiver side, if I have a situation suppose D_3 is in error, I don't have any other bits in error, only D_3 is in error. So on the receiver side if I perform similar operation then you will find that these check bits will indicate error where D_3 appears. Now here D_3 appears in CH_0, D_3 appears in check 1. So only these two checks will indicate error, CH_2 will not indicate any error because D_3 does not participate in CH_2. So there if I put a condition that wherever I make an error, at whichever check level I get an error I set the corresponding bit equal to 1 and wherever I don't get any error, the corresponding bit is set equal to zero. So here since D_3 is in error so I will get CH_0 equal to one because this keeps an error, CH_1 will also be equal to1 because that is in error. CH_0 will give me a 0.

Now if I simply decode this it is 0 1 1, that is equal to 3, so that tells me that bit D_3 is in error and because it is a binary number, a binary bit whenever there is an error I simply, what I have to do

is simply compliment it and I get the corrected. So similarly if suppose D_7 is in error; if D_7 is in error then all these check bits will give me error that means all of them will become equal to 1. so it is 1, 1, 1, which is equal to 7 so immediately tells me that it is D_7 which is in error, I can correct it only for single bit error. So this can detect as well as correct to limited extent. So it can detect single bit error, correct single bit error but even that gives a lot of improvement but the advantage over simple parity check is that in case of parity check, we can just find out whether the data is in error or not. Here I can detect a single bit error, I can also correct a single bit error.

However, both that is the parity check or the Hamming code this assumes that I have one, if I have one-bit error then I can correct it, more than that again I am helpless, I cannot do anything. But generally in communication system I mean whether you want to transmit a data from one place another place or you transmit a data from one device to another device, the error comes because of disturbance. And whenever you have a disturbance, it is not a single bit but a number of successive bits will be there in which case these schemes does not give you any advantage. There are different coding schemes, there are different coding schemes in which that can be reduced but in computer system what has become very popular is what is called a CRC code, CRC or cyclic redundancy check.

(Refer Slide Time: 00:18:12 min)



So whenever we talk about cyclic redundancy check then for every such coding scheme there is one polynomial which is called a generating polynomial and depending upon the different forms of generating polynomial, I can have different types of codes. And in this case what will be the code length for this cyclic CRC code length that depends upon what is the generating polynomial that you will take. Now I will not go in to any theory of this cyclic redundancy check.

Now let me say that given a generating polynomial of a cyclic redundancy check how you can generate the CRC bits, just a very simple scheme. Suppose the generating polynomial of a cyclic redundancy check is given something like this say $x^{16} + x^{12} + x^5 + 1$. Now this is a generating polynomial which is used for a particular kind of CRC code which is called CCITT - CRC. Now

there are different standards, the different standard uses different polynomials, now these powers of x in this generating polynomial that tells you that where you have to put an XOR gate in a code of length 16 bits. So given this polynomial, this polynomial can be implemented with the help of a set of shift registers and a number of XOR gates. Is it x^{15} or x^{16}?

So here it will be something like this, suppose this is the CRC code which is implemented by a number of shift registers 4, 5. So this is bit 0, this is bit 1, bit 2, bit 3 and bit 4 and the output of this, after fifth bit I put an XOR gate. The output of the XOR gate goes to the next shift register and here the bits will be from 5 to 11, so b_5, b_6, b_7, b_8, b_9, b_{10} and b_{11}, from b_{11} again it goes to an XOR gate. This XOR gate output goes to another shift registers and here I will have the remaining bits b_{12}, b_{13}, b_{14} and b_{15}, I have total 16 bits, 0 to 15. Total code length is 16 bits so this is b_0 to b_{15}. Now again I doubt whether it is x^{15} or x to the power 16, I think it is x^{16}, not x^{15} because you find that for x^5, I am taking the output from b_4 that goes to an input of the XOR gate.

Similarly, for x^{16} there is output of b_{15} which will go to an input of an XOR gate. And in this case the XOR gate will be placed here, output of this goes to the input of XOR gate. Here I will feed the data and this output will be connected to b_0 and also to the other inputs of two other XOR gate. No Sorry, this inputs will not come from here, this inputs will come from this place. The final output is the content of all these shift registers that is your CRC code. So you find that the powers of x that indicate that at which locations in this set of shift register I should place an XOR gate. Similarly, I can have other polynomials for example a CRC-12 that uses a polynomial of $x^{12} + x^3 + x + 1$, it is the polynomial for CRC - 12. There are some standard polynomials.

Now again in this CRC code you find that this can simply detect an error, it cannot correct the error. Error correction is not possible in the CRC code. So in this case, whenever an error is detected simply the transmitter has to be informed that this chunk of data is erroneous, you retransmit that chunk of data that's all. This does not have any power of correcting the error. Data input is data stream that you want to transmit, it's a bit sequence, a bit sequence that you want to transmit that will come to this data input one after another in the sequence in which they will be transmitted. Output is the content of all this shift registers. All these bits gives you the CRC code, so it should be like this that after you send all the data bits, the data bits have been sent simultaneously, it is also pushed into this set of shift registers following this logic. So after all the data bits are transmitted, you have to transmit or in our case it has to be stored on the device. So after storing these data bits you have to store all the bits which are there in the shift register that gives you the CRC code.
No, data is the actual data that I want to transmit. Suppose I want to transmit a data stream something like this 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0 something like this. So when I want to transmit this first I will transmit 1, I will transmit 1 simultaneously this 1 will be pushed into this set of shift registers following this location then I want to transmit 0. So when I have pushed this 1 in this set of shift registers, this bit combinations are different, different from what it was earlier. Now I am transmitting 0, simultaneously 0 is also being pushed into this. Then I am transmitting 1, simultaneously 1 is also pushed into this.

Now whenever I push a new bit in this that is combined with the previous state of the shift registers and the new state is doing store, so finally when this last 0 has been transmitted this 0

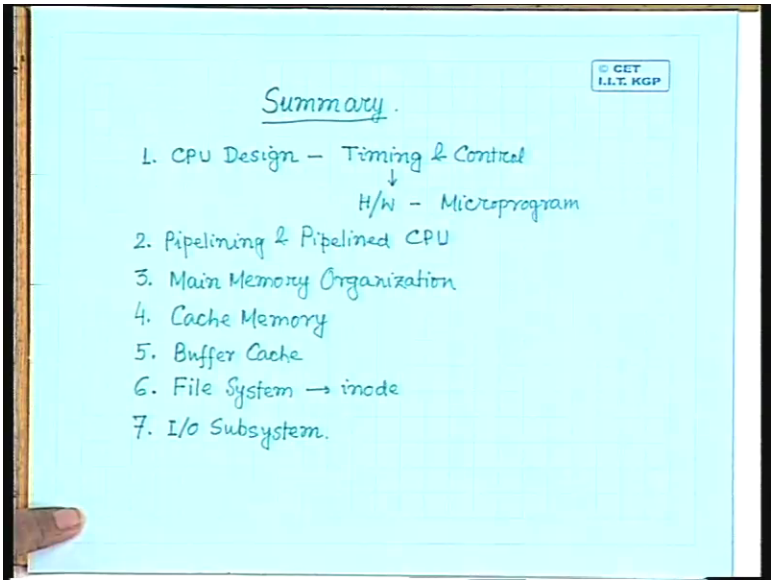
was also pushed into shift registers. So when this last 0 is transmitted at that time this shift registers content will be something. Now what is that something that depends upon how this XOR gets are been placed and whatever is the content of this that gives you the CRC code for this particular bit stream. So at the receiver side, I have to regenerate this CRC code then check whether the CRC code which is being transmitted and the CRC code which is been generated they are same or not. If they are same I assume that the data that have received is correct. If they are not same then I have to assume that the data that have received is not correct.

Now this not correct does not mean that only the data is erroneous, even the CRC code itself can be erroneous because error might have occurred in the CRC code, it might not have occurred in the data. But when I am regenerating CRC code because my data is correct, I will generate the correct CRC code but the CRC code that I am receiving that is erroneous so again there will be a mismatch but I don't have any way to find out whether the error has occurred in the data bits or error has occurred in the CRC bits. So I assume that the whole think is erroneous, so I ask that transmitter to retransmit the entire bit stream. Even accidentally, it may so happen that even if your data bits are wrong the CRC code that you have generated accidentally matches with the CRC code that you have received but in that case you don't have any other way, you have to accept whatever you have got. Sir we have to transmit CRC code? Otherwise how can you check? You cannot check it.

And actually this CRC is used whenever you store a set of data on a disc. See in every sector, I store the data in the form of bit stream so initially the bitstream will be stored followed by the CRC code. So for every sector I have the data bits followed by CRC code. Data transmission is serial, I may decide that after say 50 bits I will transmit the CRC bits, I may decide after 1000 bits I will transmit the CRC bits that is up to the designer. But unique means that depends upon what is the content of the data, data stream. For every bit stream you have an unique CRC code but again this CRC codes can simply detect and accidentally it may not detect also but that you have to accept.

All 0's, yeah. So in summary let us see what all we have done in this course. Yeah, people are trying for it, with CRC it cannot be done, you cannot correct. See the advantage of CRC is it takes care, it generates a code by considering a chunk of data. Hamming code considers a single bit not a chunk. That is why your chunk of data which is to be generated for which a CRC code has to be generated should be much larger than the CRC code length. If I go for a CRC code of 16 bits may be my data length should be 1 KB. Yeah, economy should be the major criteria. So that your data length along with the CRC code that should give you complete packet. Yeah there can be 12 bit codes and usually 12 bit or 16 bit they are used. That is what I said just now, if I go for 16 bit CRC my data length should be much larger than that. So maybe I should go for say 1 kilobits or data ,16 bits CRC for 1 kilo bits of data something like that, otherwise it will be not be economic. Not, 2 kilo bytes, why 2 kilo bytes? 1 kilo byte plus 16 bit, CRC is generated for block of data. So your data stream along with CRC will form a packet so that data packet is to be transmitted.

(Refer Slide Time: 00:34:10 min)



So in summary let us see what all we have done in this course. Initially we have started with the CPU design. We have started with CPU design and the specific aspect of CPU design that we have done in an elaborate way that is the timing and control circuit design and along with this timing and control circuit we have seen that what is hardware control unit and what is a micro program control unit. After this we have seen that what is pipelining and pipelined CPU. then when we discussed about this pipelining, pipelining concept we also have talked about something like collision vector that decides that how the pipelining scheduling has to be done that means what are the time instance when a new job can be placed into a pipeline that will not lead to any collision and we have seen that concept is particularly important when I have multifunction pipeline. So after doing this we had gone to memory hierarchy. Isn't it?

So the next thing that we have done is the main memory organization and in this we have seen the evolution of different memory organization, memory architectures then finally what we had talked about is what is called paged segmented memory management while your basic memory management or basic memory basic memory organization is the segmented memory organization and on top of segmentation, we have imposed the paging technique for better memory management, ease of memory management and after main memory organization I think what we have done is cache memory and there we have seen different models of cache memory starting with associative memory then direct mapped memory and also what is called set associative memory.

So we have said that the associative cache memory is more flexible but at the cost of additional hardware, on the other extreme we had the direct mapped cache memory where the hardware is minimum but at the same time there is a possibility that you will have more number of cache miss. So a compromise between these two that is less hardware, at the same time I want to have more number of cache hits so that is given by what we have said about the set associative cache memory, so that the blocks in the cache memory are partitioned into different sets and within a set it is associative.

So after cache memory the next topic that we have discussed is, I think buffer cache which you have said that it is an interface between the secondary storage device particularly the devices which are blocked storage device, so it is an interface between the blocked storage device and main memory and the buffer cache is maintained in the main memory and for management of the buffer cache or replacement of buffer cache contains the policy that is used is least recently used policy. After that we have discussed about the file system and when we discussed about the file systems, we have talked about what is meant by an inode or an index node of a file. We have said that all the information about the file is contained in the inode including the disc blocks which contained in the file data. Then we have discussed algorithms of inode allocation at the same time we have also discussed about the block allocation and then finally today what we have discussed is about the input outputs of system and the issues related to input output subsystems. So this forms our entire course.