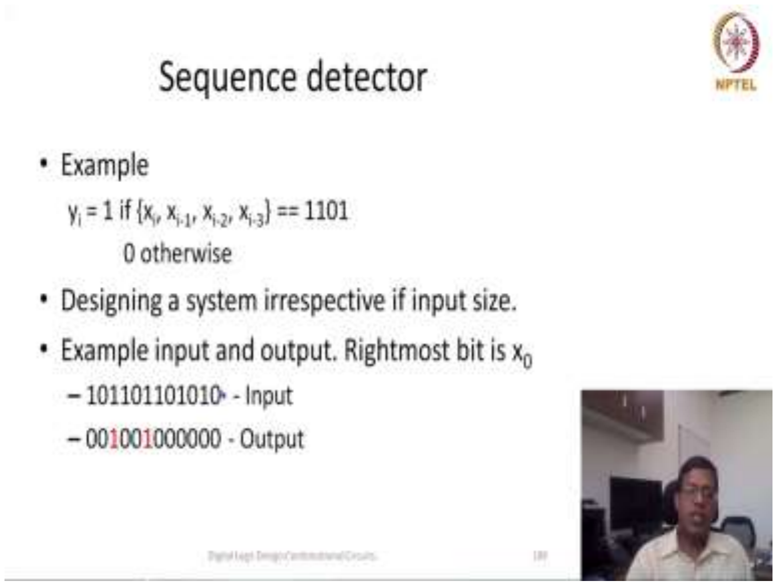
Digital System Design
Professor Neeraj Goel
Department of Computer Science Engineering
Indian Institute of Technology, Ropar
Integrative Circuits-2

(Refer Slide Time: 00:16)



Sequence detector

- Example
  $y_i = 1$ if $\{x_i, x_{i-1}, x_{i-2}, x_{i-3}\} == 1101$
  0 otherwise
- Designing a system irrespective if input size.
- Example input and output. Rightmost bit is x_0
  - 101101101010 - Input
  - 001001000000 - Output

So, the third problem is more intricate and it is more moving involving to solve, so now we are trying to design some sort of a sequence detectors, so what is a sequence detector? So, we are given some sort of a sequence, we are given some sort of a pattern and in the pattern we would like to design given an input we would like to see whether our input is matching with the pattern or not.

So, in your C C plus plus programming course, you would have done this string matching algorithms or pattern-matching algorithm, so this is very similar to this we are now matching the binary sequence of bits. So, in a whole string in a whole number of 0's and 1's we are trying to find out if 4 consecutive numbers 4 consecutive bits are equal to certain pattern or not. So, to consider this example, let us say my output yi would be 1 if xi, xi minus 1, xi minus 2, xi minus 3 is equal to 1101.

So, this is my pattern which I want to match and if it is not matching with this then output will be 0. So, if I want to design this system irrespective of the input size I can again use iterative design

I can think of a design in such a way that whatever is the input size this particular method will always work.

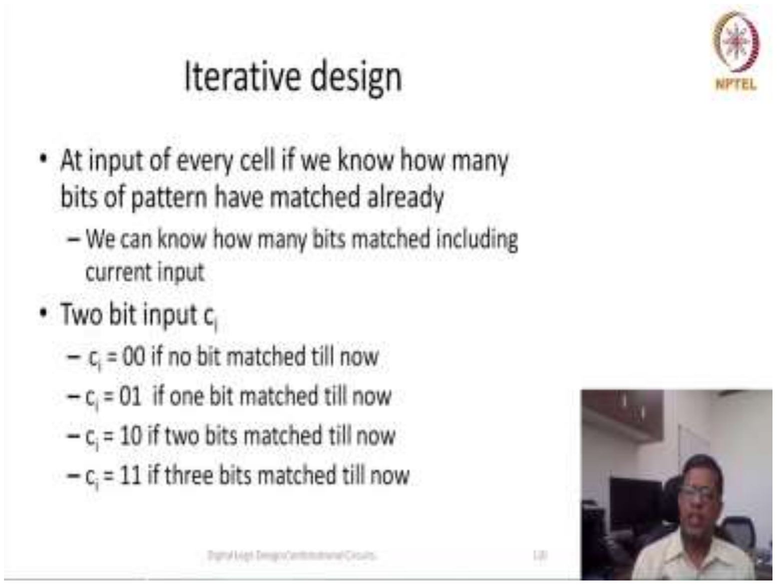
So, now why we are now we are trying to design the sequence detector using this example, but again I would say that it does not matter it does not depend on whether this pattern is this this this method of designing sequence detector is generic enough and it can work for any kind of patterns any kind of bits pattern I would say.

So, if this problem is given to us that I would like to match a particular pattern in a sequence of bits, how should I go about it? So, to understand how we should go about it the first thing we have to do is we have to first maybe take an example and see how does it actually work, so now these are the 12 bits of input and now if I want to see the output see this particular bit is the LSB, so that means this is x0 this is x1 this is x2 x3, so for x3 2 1 0, it does not match.

So, that means my output would be 0 for all of these cases. Similarly, output would be 0 for this y4 also because this pattern does not match., y5 also 1010 it does not match, so it would be 0, so what work for y6? It is 1101 so, y6 is 1, y7 is again 0 because this pattern 0110 does not match and y8 is also 1011, so it is also not matching, so output is 0. But for y9 1101 it is the pattern is matching, so the output is 1 and for y10 and y11 also it is not matching so the output is 0.

So, essentially we are taking some example to see that how it work out and how the output would be 1. Now, I want to design this, how should I go about it?

(Refer Slide Time: 04:16)



Iterative design

- At input of every cell if we know how many bits of pattern have matched already
  - We can know how many bits matched including current input
- Two bit input c_i
  - $c_i = 00$ if no bit matched till now
  - $c_i = 01$ if one bit matched till now
  - $c_i = 10$ if two bits matched till now
  - $c_i = 11$ if three bits matched till now

So, basically I have to find and to create a cell I have to create a cell and if I somehow even to identify that how many bits has matched the pattern already, so if the input to this cell is how many bits of pattern has matched already then considering the current input then we can say that including this particular current input how many bids have matched, that would be the output.

So, if we have such kind of a generic input and output then we can have this iterative cell starting from bit 0 bit 1 bit 2 bit 3 and this can go onward and number of bits. So, that means somehow we have to create this information and you have to process this information that what is the how many bits has already matched?

So, now there could be possibility that number of bits has not matched, none of the bits has matched or 1 bit has matched or 2 bit has match or 3 bit has matched, now with the results of those possibility that 4 all the 4 bits has matched, but all the 4 bits has match this particular possibility we can remove because we can always see that for 4th bit to match so, we have this y output as 1.

But this particular information that how many bits has already matched we can reuse we can use only this first no bit has matched 1st bit has matched 2 bit and 3 bits has matched, so you can keep information only till that level. So, if I have this 4 pieces of information which need to be stored or which need to be remembered that if either none of the bit has matched, 1 bit has matched, 2 bit has matched or 3 bit has matched, so how many bits I would require to represent this information?

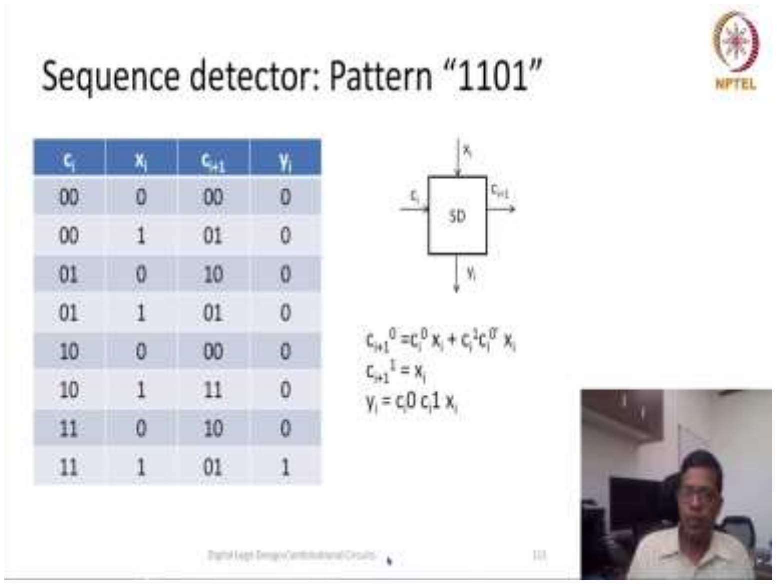
Since, total number of information is 4 the total number of bits I need to keep to remember this information would be 2 bits, so let me call these 2 bits as ci and I can now say so this is called encoding so this encoding could be independent of whatever we say, but let us keep it simple, so we say that ci is 2 bit input as well as 2 bit output also, so if ci is 0 0 we say that no bit has matched till now and if ci is 0 1 we say only 1 bit has matched and if ci is 1 0 then we say 2 bits has matched till now and ci equal to 1 1 means 3 bits has matched till now.

So, I will repeat this particular question again that why we are not keeping another information that all the 4 bits has match till now, since this is an iterative design and the input to the iterative cell would be given as a output to this in iterative cell would be given as input to the other one. So, this particular information that only 4 bits has matched is of no value to the input circuit. So, input circuit I just want to understand whether one of the previous bits or whether 2 bits of the pattern has match or 3 bits of the pattern has matched.

So, here we can also say generically that if my pattern will have N bits I have a N bit pattern to be matched then this value of ci would be or the number of bits in ci would be equal to log N or the base 2. So, and which particular pattern which particular information should I call 00 which particular information should I call 01 10 and 11 this is actually independent it does not matter.

For design, it does not matter for optimization, yes there could be certain sometime advantage over one or another but as a design exercise it is more scalable that we keep in a order 00011110 so that it is simple to write and after that for analysis also. So, what is the next step? So, here till now we have understood that this is the information to be copied.

(Refer Slide Time: 09:14)



Sequence detector: Pattern "1101"

c_i	x_i	c_{i+1}	y_i
00	0	00	0
00	1	01	0
01	0	10	0
01	1	01	0
10	0	00	0
10	1	11	0
11	0	10	0
11	1	01	1

$c_{i+1}{}^0 = c_i{}^0\, x_i + c_i{}^1 c_i{}^{0'}\, x_i$

$c_{i+1}{}^1 = x_i$

$y_i = c_i0\; c_i1\; x_i$

The overall design or the pattern would be something like this that so this is the single slice or single bit design of my sequence detector SD, so here ci is the input ci is a 2-bit wide input and every time we are checking for xi and the output is ci plus 1, which is also 2-bit. The output is yi, so now if I want to cascade so it will start from 0, so basically it will start from right towards the left, that was the pattern anyway in the sequence detector that yi equal to 1 when xi xi minus 1 xi minus 2 xi minus 3 is equal to 1011 1100 1101.

So, what is the next step now? We have found out the slice we have found out what is the input what is the output how many bits in the input how many bits in the output and so what is the next step? Now, it would be better that we write truth table, so after writing the truth table then we can find out how we can determine the output. So, I am writing the truth table at once, but we will try to understand it in a row by row.

So, let us consider that input is no bits has already matched, because no bits has already match and if my xi is 0 then again I have to say that no bit has already matched, but if no bit has already mentioned in current input is 1 so then I have to say that at least 1 bit has matched, because 1 bit has matched the output will become 0 1. Now, in case of 0 1 input, so that means 1 bit has already matched now the current input is 0.

If current input is 0 so that means now 0 1 these 2 bits have already matched, so the output will become 1 0, output sealed remains 0. Similarly, if the current status is 0 1 that means 1 only 1 bit has matched and again xi is 1, so that means it become 11, so 11 is of no use so that means we again say that only single bit has matched. So, the output will also be 0 1, ci plus 1 is equal to 0 1 and the yi would be 0 1.

Now, in the second state where we say that 2 bits has matched to between 0 1 is already there, now the third bit is 1 or third bit is 0, if third bit is 0 then everything has gone away so we will say that no bit has matched the pattern does not match with the existing pattern so it become 0 001, 001 does not have any meaning here so we will say that none of the bit has matched.

But if the current status is that 2 bits has smashed and the next input is 1 so then we have to say 3 bits have already matched, so that means we will write in ci plus 1 as 11 so that means 3 bits in the pattern has already matched output is still 0. Now, if 3 bits has already matched and the current input is 0, so that means it will become 0101, 0101 is we can say it is of no use but we can still say that 2 bits has already matched their we can utilize that and so the state will become the second state which means 2 bits have already matched.

Now, if 3 bits were already matching the current input become 1, that means the output will be 1 and the next state the next carry output would be a only single bit has matched, so 1101, so in a long sequence we see that we can say that only one input as matched in the pattern, so 0 1 would be the output here for ci plus 1.

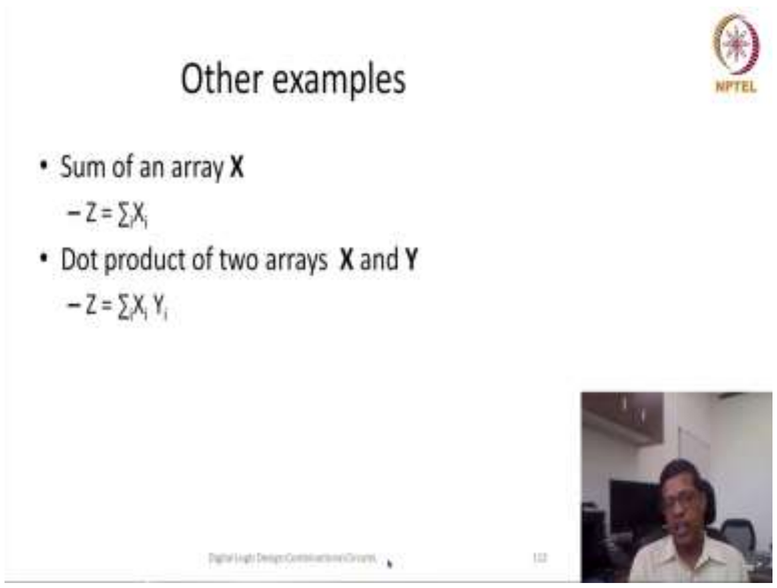
So, in this way we have to see carefully for each and every input pattern and we see where we have formed, what we have created so far, whether that particular pattern has matched how many bits of (())(13:57) current pattern. So, once we have this truth table now we know what to do, we can write 3 k maps 1 for 0th bit of ci plus 1 another for 1st bit of ci plus 1, another for yi.

So, if I write all these Boolean equations I will create a create using my truth table, then I will have some equations something like this which says that 0th bit of ci plus 1 would be determined with this equation, 1st bit of i plus 1 would be determined by xi and yi would be determined by this these three basically all of these condition has to match, basically this has to be 1 this as be 1, this has to be 1, then the output would be 1.

So, this sequence detector is a very interesting application and which can be used but yes you may feel that we are not done enough, so if some more examples would have been taken then it would be even better but I will tell your from is that in one of the next week's we will do a couple of more exercises, but then the sequence detector would be designed using some alternative technique which are applicable here also, but there we will use instead of iterative design we will use sequential design.

So, in the sequential design some other techniques can be applicable we can apply those thinking those thoughts at least in designing it using combinational circuit as well.

(Refer Slide Time: 15:45)



Other examples

- Sum of an array **X**
  - $Z = \sum_i X_i$
- Dot product of two arrays **X** and **Y**
  - $Z = \sum_i X_i Y_i$

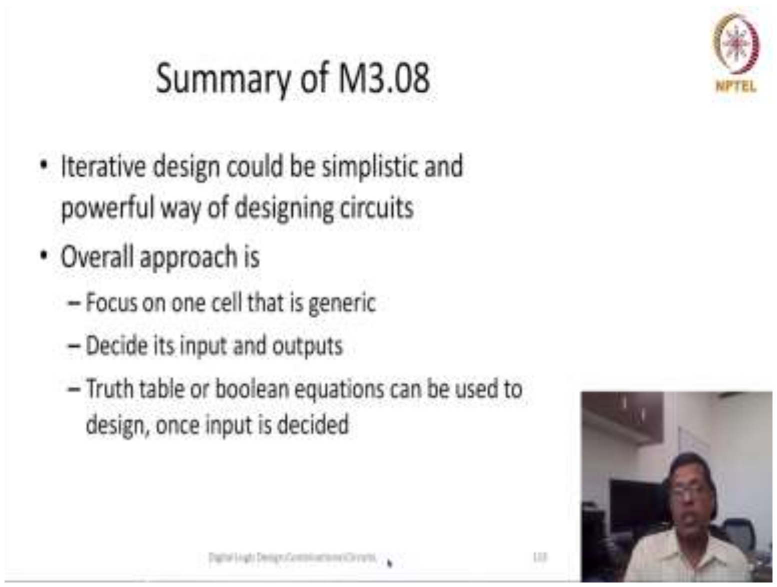
So, with this I would like to close this particular lecture and with couple of more applications, which you can try yourself you can take it as a quick tutorial of this particular lecture. So, let us say if I want to do sum of an array X, X can have N number of element. So, here also if we do an iterative design we create an adder, we create an adder and that adder will give the output of the next header, so this way this also could be an iterative of design and can be used effectively to find out the sum of an array X.

So, let us say similarly if I want to know the dot product of two arrays, X and Y so that I need to multiply each element of X with the corresponding element of Y and as well as add them. So, then also we can use such an iterative design where both of them would be multiplied as well as added and the add output would again be given as input to the next stage.

So, it in each stage I will have three inputs 1 as the addition or accumulated addition from the previous stage as well as Xi and yi and there would be one output that would be again accumulated addition so far. Sometimes we see that all these techniques of iterative design may not be very very effective, so for example, we have seen in in ripple carry adder that aipple instead of ripple carry adder is slow because we are iterating through all the stages and carries rippling through all the stages and then we get the output.

But the other design like carry look ahead adder or carry skip adder they are more efficient, taken, yes, this is this could be the possibility but this iterative design offers still more advantage some advantage in terms of simplicity and a repetitiveness, so design would be simple design could be designed very fast because when you are designing a cell, then you are designing only one cell you can verify it more efficiently and then reusing will cause less of the verification effort as well.

(Refer Slide Time: 18:15)



Summary of M3.08

- Iterative design could be simplistic and powerful way of designing circuits
- Overall approach is
  - Focus on one cell that is generic
  - Decide its input and outputs
  - Truth table or boolean equations can be used to design, once input is decided

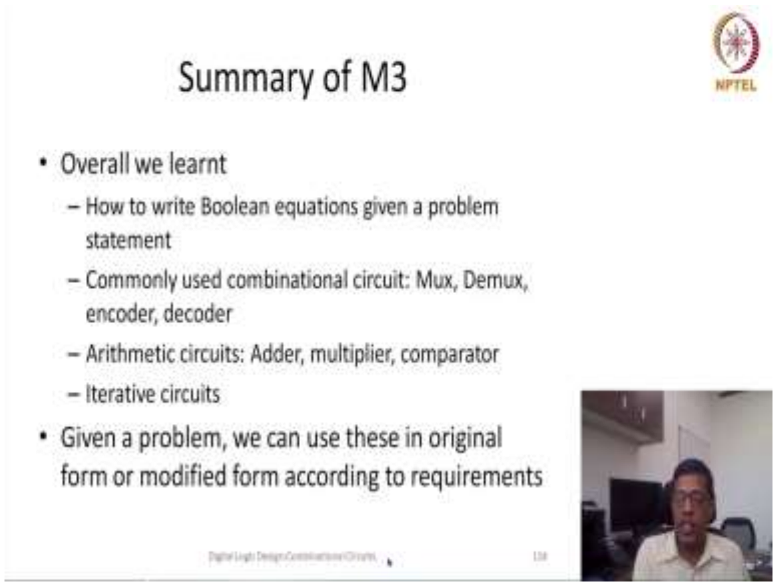
So, we can summarize this particular lecture as that say this a very simplistic but powerful way of designing circuits, but given a problem let us say some problem is given to you whether first you have to identify whether that problem can be solved using iterative design or not. So, that decision can be made quickly by focusing on the overall problem, if problem looks that it could be sold in a similar session for all the bits then we can use iterative design.

And once we have decided that we are going to use iterative design then focus on one cell and for one cell try to identify what would be the inputs, what would be the outputs and what is the meaning of that particular input that is more important. So, once we know the input and output we know their semantic behaviour.

So, for example in our in our today's lecture we our sequence detector, the sequence detector has input ci and ci plus 1 so the semantics of ci and ci plus 1 we have identified ourselves we said that ci will be 2-bit, ci plus 1 will also be 2 bit and this is the meaning of 00 01 10 and 11 in case of sequence detector.

So, that way we had to decide what are the inputs output what is the semantic meaning of input and output, once we know input and output then we can go ahead and create truth table or sometime Boolean equations are passed than you can use Boolean equations also and after that our design is almost ready.

(Refer Slide Time: 20:00)



Summary of M3

- Overall we learnt
  - How to write Boolean equations given a problem statement
  - Commonly used combinational circuit: Mux, Demux, encoder, decoder
  - Arithmetic circuits: Adder, multiplier, comparator
  - Iterative circuits
- Given a problem, we can use these in original form or modified form according to requirements

So, these also the last lecture of this particular module, so I will also summarize this overall module in couple of minutes, so in this particular module the objective was to design a Boolean combinational logic or Boolean system, so our digital system using basic gates. Now, the question which we the question with which we started this module that how to write Boolean equations even a problem statement, so doing this whole module last 8 lectures, we have seen different scenarios different cases and in most of them we have focused on how to write how to give this Boolean equations how to give input specification.

So, the overall observation is that these Boolean equations are essentially synonymous to the behaviour or the functionality of the design. So, if functionality of the design is clear to us behaviour of the design is clear to us in most of the cases Boolean equations were formed automatically by using our rational and by using our logic.

So, using those Boolean equations, then we have also formulated we have also seen couple of combinational circuits which are very very commonly used in different applications, like multiplexer, de-multiplexer, encoder, decoder, so various circuits that has been discussed and along with using X understanding this commonly used combinational circuits we have also seen a design of arithmetic circuits, adder, multiplier, comparator and we have also seen how to design them more efficiently.

So, this adder as well as multiplier both offer some sort of parallelism. A sequential process is usually slow so we have found different way, different techniques of doing things in parallel, so that multiplication could be faster or addition could be faster, we have also seen there is a area and delay trade off, so if we try to optimize the delay then area would usually increase while the simplistic technique or the Naive technique will have lesser area, but the delay is usually more.

And in today's lecture we have seen iterative circuits which is also an interesting approach or a different approach to design any particular circuit. So, as a end goal of this particular module we should be able to attempt a digital design problem with a combinational logic, so if a problem is given to us then and we have two soul we had to design that particular behaviour or that particular of X that particular function, then first we need to break that function into smaller pieces and we need to see that can of each and individual part of this these individual parts or small pieces can they be implemented using anything which you have which we have learnt so far.

So, for example multiplexer de-multiplexer adder multiplier, so it could be quite possible that we may not be able to design directly using them, but during this whole module we have not only learned how multiplier is working, but we have also I learned how to design a multiplexer. We have not only learned how a de-multiplexer can be used but we also seen how to design a de-multiplexer or a decoder or an encoder.

So, given a requirement given that particular module we have identified that lets a multiplexer need to be used then but the same multiplexer cannot be used because let us say number of inputs are not used power N, then we may modify or we may redesign that multiplexer de-multiplexer or we can have a unique kind of an adder, so because the whole module has not only focused on not only focus on application oriented part that we can use only the cells but we can also design so given a problem statement first we can think of that how we can break things into different parts.

Then each part which particular kind of design can be useful and if nothing works out then we may have to go for a custom solution for that particular problem that is also fine, but with all these techniques in your Arsenal, then you can find out how to design a new concept or a new system which has a different functionality. So, with this particular end of module, I believe that you should be able to design different kind of a combinational logic with whatever techniques we have learnt so far. Thank you very much.