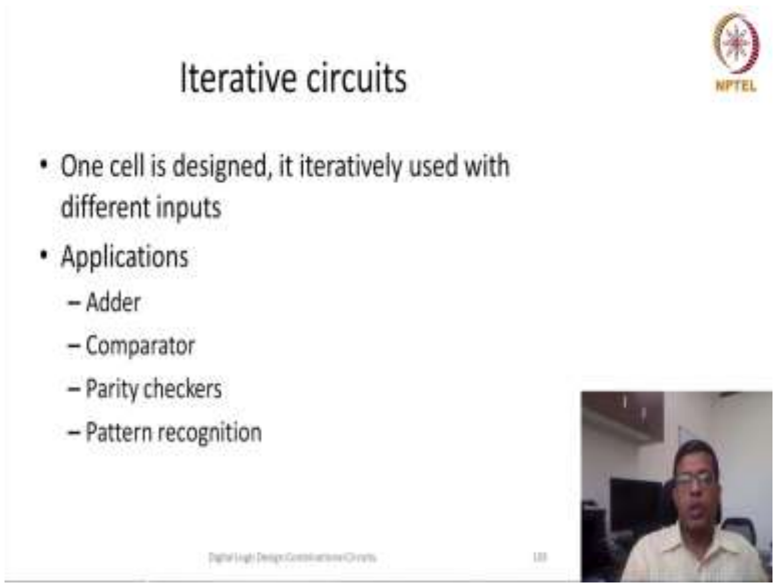
Digital System Design

Professor Neeraj Goel
Department of Computer Science Engineering
Indian Institute of Technology, Ropar
Lecture 38
Iterative Circuits-1

Hello everybody, this is the last lecture of our this current module where we are designing combination circuits. So, in this particular lecture, we will learn about a new technique or different technique of designing circuit.

This technique is called as iterative circuits. So, the beautiful nature of this particular technique is that we do not focus on entire design, but we focus on a one particular cell or one particular part of the design which would be replicated for replicate for N number of times to which would be replicated for N number of times for a complete design.

(Refer Slide Time: 01:09)



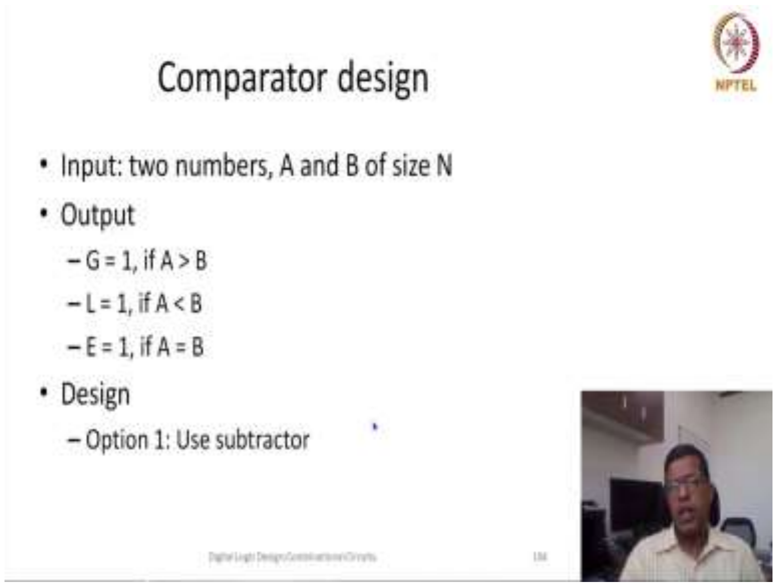
Iterative circuits

- One cell is designed, it iteratively used with different inputs
- Applications
  - Adder
  - Comparator
  - Parity checkers
  - Pattern recognition

So, in our last couple of lectures also we have seen some utilization of this particular iterative design. So, for example in whenever we are doing a ripple carry adder, so this adder one full adder was actually an iterative circuit which was repeated again and again. Similarly, in our previous lecture also in array multiplier, the adder design is also an iterative circuit which was repeated in two-dimensions, because it was repeated in two-dimension, so overall we were able to create a full N cross N multiplier.

Now, we will see this particular technique in a more methodological way in this particular lecture, so we will take a couple of more examples and we will see it from the perspective of iterative design that if we need to design an iterative circuit, what would be the characteristics or how we will design that a iterative circuit. The first example I am going to take is a comparator.

(Refer Slide Time: 02:22)



Comparator design

- Input: two numbers, A and B of size N
- Output
  - G = 1, if A > B
  - L = 1, if A < B
  - E = 1, if A = B
- Design
  - Option 1: Use subtractor

So, in a comparator, we are taking two inputs, let us say A and B and irrespective of the size, so but let us say for the matter of this lecture let us say we say the size is N, N could be anything. Now, when we compare two numbers A and B, there would be three outputs G means greater than if A is more than B then G would be 1, if A is less than B then L would be 1, if A and B both are equal then E would be 1.

So, essentially there are three outputs G L and E, if it looks like that in this particular case at least one of them has to be 1, either it would be greater or less than or equal to. So, but still three different variables are used or three different outputs are used so that we can be clear that whether it is greater less or equal. So, how to design a comparator this has been also an interesting topic.

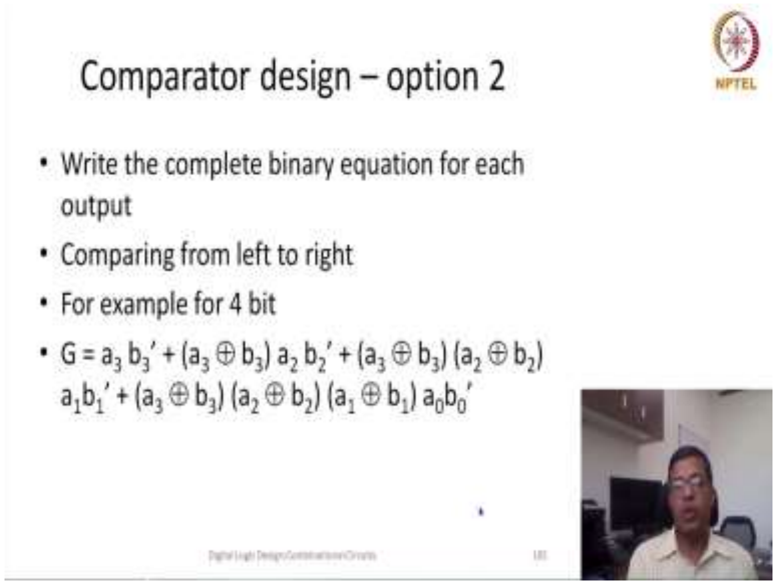
So, one of the option which has been usual multiple people is if we use subtractor, so let us say we use our adder circuit and B we take two's complement of B so that measure we are subtracting B from A. Now, if you are subtracting B from A and we see what is the output of my adder if output is do we care about output actually? Actually no, we only want to know what is the sign of the output if the output is positive that means A was bigger, so we can set G equal to 1.

If the output turns out to be negative that means B was bigger so we can say L equal to 1, but if output is equal to 0 that means A and B both were equal, so we have to set E equal to 0. So, in addition to doing subtraction, we also have to have another circuit which will see whether the output is 0 or not. So, that means there will be other than having a subtractor other than having using adder plus two’s compliment we also required to or we also require to and all the outputs.

Sorry, we have to OR all the outputs to make sure that the output is whether all the bits were 0 or not. So, this is one particular way if somehow in the circuit where you we are using comparator if in that circuit there is already an adder and that adder could be reused then people try to use this particular type of comparative.

So, now because we are able to share that adder then or basically because we utilize that particular adder for other purposes then we use comparator. And for example in our computer systems or a processor only one of either addition would take place or subtraction would take place or comparison would take place in that case it is very common that we use this particular type of comparator. So, what could be the other alternative way?

(Refer Slide Time: 05:55)



Comparator design – option 2

- Write the complete binary equation for each output
- Comparing from left to right
- For example for 4 bit
- $G = a_3 b_3' + (a_3 \oplus b_3)\, a_2 b_2' + (a_3 \oplus b_3)(a_2 \oplus b_2)\, a_1 b_1' + (a_3 \oplus b_3)(a_2 \oplus b_2)(a_1 \oplus b_1)\, a_0 b_0'$

So, for other alternative way we have to see that how do we use to compare two numbers, so when we were in elementary classes, so when we were in elementary classes the way we used to see we used to see from we use to compare from left to right and if the any particular bit in the left is 1 and corresponding bit in the second number is 0, so that means we used to declare that the first number is bigger or whatever number has the first bit as 1 is the bigger one.

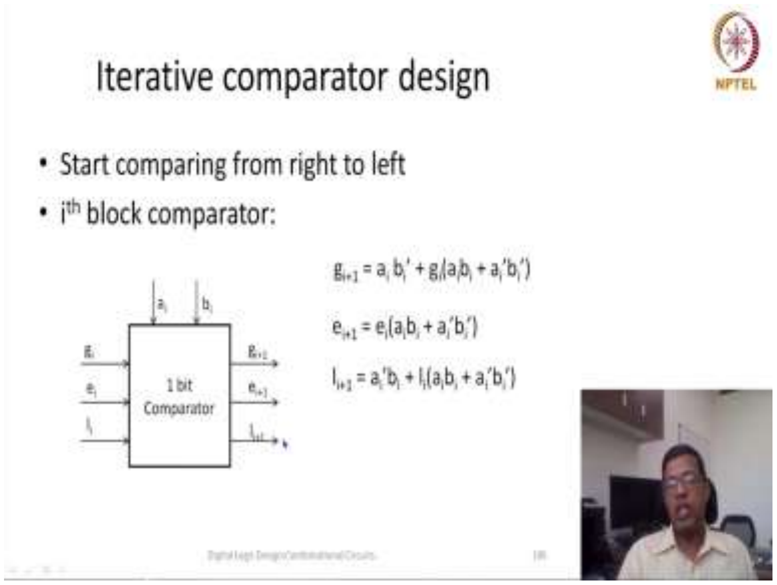
Now, if we try to translate this particular approach for a digital circuit, then we have to write a complete binary equation. So, we have to we cannot leave it in between so for example, if it is an N-bit adder and N let us say is 16, so even though we came to know that it is bigger at only at the first bit itself, but we still have to go through all the bits till 0 because in the worst case it could be anything, because we do not know the what the number would be.

And our latency as we have discussed in our previous classes or our latency or delay of total circuit depends on the worst case path, so the is that we had to finally write all the equations till like an to a0, so for example I have written one equation for 4 bit we will say that let us say we let us remember that G would be 1 if A is bigger B is smaller.

So, that means either a3 bit the MSB a3 is 1 and b3 0 then G would be 1 or a3 and b3 both are same that means XOR would be there. But a2 is 1, b2 is 0 or a3 b3 are same a2 b2 is same but a1 is 1 and b1 is 0 or all the 3 bits are 1 same sorry and a0 is 1, b0 is 0, so this essentially takes care of all the conditions and this would be the binary equation expression or maybe function for G. similarly, way to find out the binary function for less than as well as equal to.

So, this approach although is good complete but has one challenge that as soon as my number of bits are changing I have to redesign I have to find out these equations again. So, what would be better if we can find out find out a iterative you design which we can repeat every time, so we say that for 1 bit we design this circuit and that keep on happening.

(Refer Slide Time: 08:57)



Iterative comparator design

- Start comparing from right to left
- i^{th} block comparator:

$$g_{i+1} = a_i b_i' + g_i(a_i b_i + a_i' b_i')$$

$$e_{i+1} = e_i(a_i b_i + a_i' b_i')$$

$$l_{i+1} = a_i' b_i + l_i(a_i b_i + a_i' b_i')$$

So, in iterative comparator design what we will do is we will let us say right now, actually we can do it from either way right to left or left to right but for an illustration, what I am doing right now is from a right to left, so basically from LSB to MSB side. So, in LSB to MSB side the ith that block or basically ith comparator design would be something like this that there would be some input basically it says that from 0th bit to a ith bit it has been found that 0th bit to ith bit if I compare then a is bigger than b, if gi is 1.

If Ei is 1 that means from 0th bit to ith bit it has been found that a equal to b and li equal to 1 means that from 0th bit to ith bit it has been found that b is bigger than a or in other words a is less than b. So, this also take into ith bit of a and ith bit of b as a input and it generates an output which also include ai and bi other than the previous comparison. So, this is an if we are able to create this kind of a circuit that means then we can traverse from 0th bit to 1st bit then second bit and we can keep on comparing.

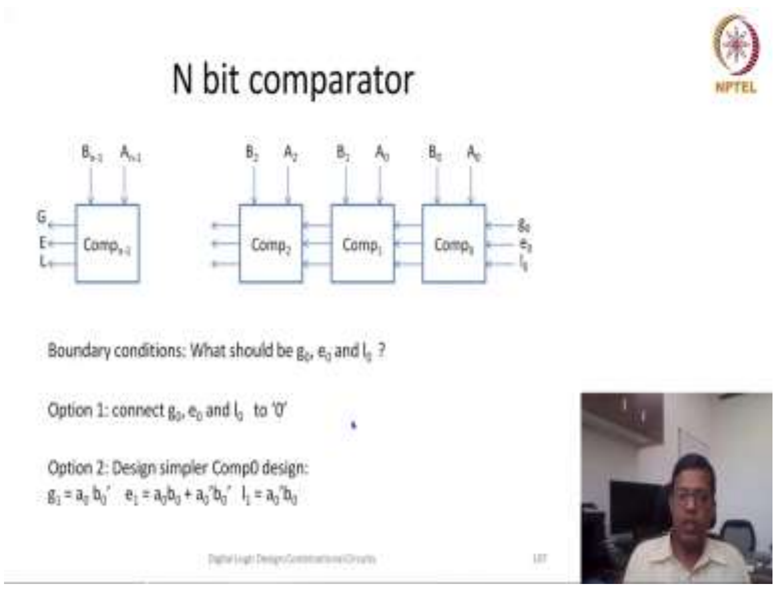
So, at the time of 0th bit we know that weather which one is and then the result of 1st bit will say that whether 0 combination of 0 and 1 bit which one is bigger, which one is smaller, so this comparison keep on happening till the end. Now, if I want to find out how to write out the find the output so there could be multiple methods again, so one method could be I can write a complete truth table and then find out in what cases gi plus 1 would be bigger or would be 1 otherwise it will in what cases it will be 0.

There could be alternative method also, by logical reasoning we can say we can find out what would be the value gi plus 1. So, from the logical reasoning I am trying to find out for you. So, now my a would be bigger than b if it was already bigger and a and b both are same or it does not matter what was the previous gi value, but if ai is more than bi then also gi plus 1 would be 1. So, in other words we can say gi plus 1 would be 1 if ai is 1 bi is 0 or if gi is 1 that means it was greater than my a was greater than b for i minus 1th bits.

And my current bits ai and bi both are same, so that means either both are 1 or both are 0. Then we can say that this a is greater than b till ith bits. Similarly, we can also find out for the equal bit, for equal bit it is more reasonable and easy to understand that if previously also it is equal and now also it is equal, so previously it has to be equal ei and currently also ai and bi both should be same either both should be 1 or both should be 0.

And similarly for li plus 1 ai has to be 0, bi has to be 1 and previously li should be 1 and the current bit should be both ai and bi should be same. So, this way if we try to by logical reasoning we can find out what are the equations for the outputs gi plus 1, ei plus 1 and li plus 1.

(Refer Slide Time: 13:14)



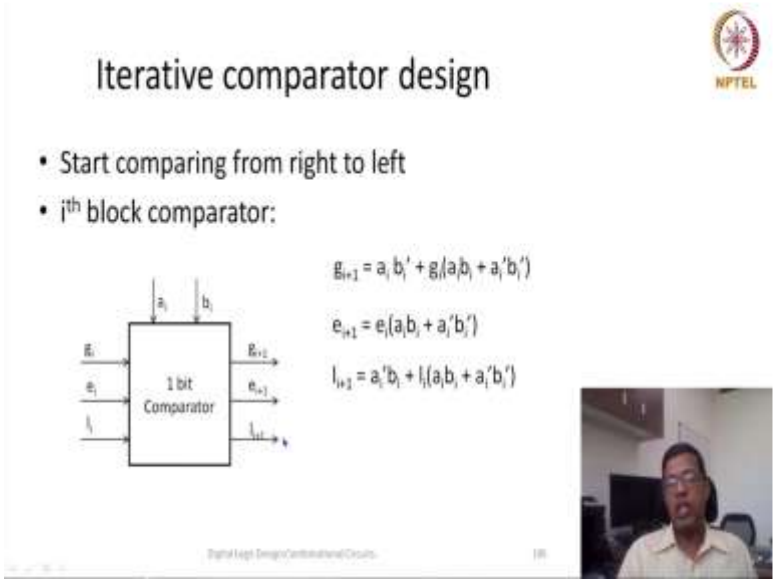
N bit comparator

Boundary conditions: What should be g_0, e_0 and l_0 ?

Option 1: connect g_0, e_0 and l_0 to '0'

Option 2: Design simpler Comp0 design:
$g_1 = a_0 b_0'$ $e_1 = a_0 b_0 + a_0' b_0'$ $l_1 = a_0' b_0$

Digital Logic Design Combinational Circuits 137



Iterative comparator design

- Start comparing from right to left
- i^{th} block comparator:

$$g_{i+1} = a_i b_i' + g_i(a_i b_i + a_i' b_i')$$

$$e_{i+1} = e_i(a_i b_i + a_i' b_i')$$

$$l_{i+1} = a_i' b_i + l_i(a_i b_i + a_i' b_i')$$

Now, using this particular block we can see that first we will have a comparator 0, so in comparator 0 will have g0, e0 and l0 as a input and a0 and b0 would be the input from the input side and similarly there would be the first comparator zeroth comparator first comparator second competitor. So, remember the output of my second comparator this comp2 depends on all the bits from 0 to 2, 0 to let us say a0 to a2 and b0 to b2.

And then this particular output tells me that whether a is greater than b whether a is equal to b or a is less than b till this particular bit, till ith bit. So, it is comp then that till ith bit whether which one was bigger or which only smaller. So, similarly we can have the final output also if for a n bit comparator, now n minus 1 A n minus 1 and B n minus 1 would be input and also the result of the previous greater than and equal to would be the input.

Now, see because this design is iterative I need to design only the one particular cell or ith cell and then I can keep on repeating itself, it does not matter what is the value of n, n could be 4, n could be 20, n we 100, so the only thing is I have to re-iterative or basically iterate that particular cell, in very low (())(14:51) interesting it is easier because we have this generate statement where we can simply generate from 0 to n and then all these blocks would be instantiated.

So, and similarly if we design it in a for a layout then also iterative designs are easier for placement because you can wire them easily and they could be interconnection is very less. So, yes, then one, so one thing one question which comes to our mind that okay so usually the input to comparator is A and B but it is it looks like that this particular design has three more inputs g0 e0 and l0 from where and what should be the value of g0 e0 and l0?

So, this we can call boundary condition, so other this iterative cells would be similar for all the points but for boundary points some time for nth point or some time for 0th point it may be slightly different. So, here we clearly know that the input is A and B, so find out what should be the value of g, e and l, so that my comparator 0 is correct, so in this particular case we can say the one option is that we say that it is neither greater than or equal to no less than so essentially all of them are 0.

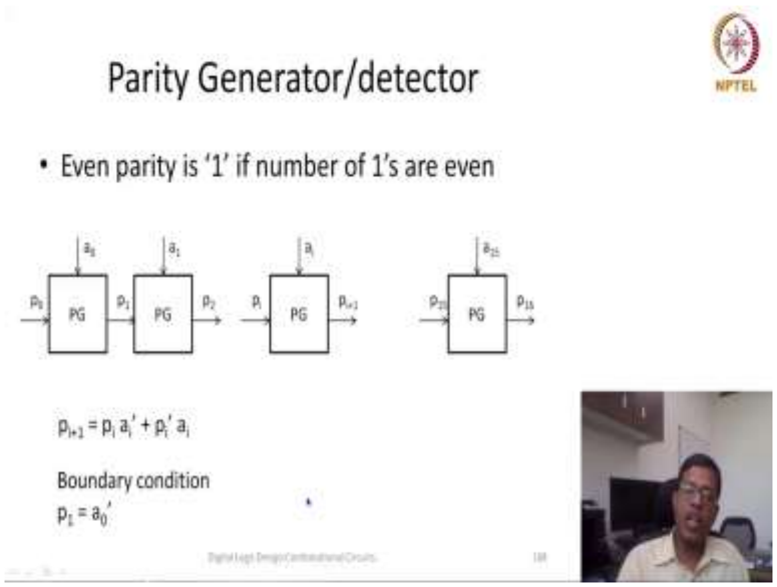
Initially, we said that one of them has to be 0, one of them either g, e or l, one of them has to be 1, but now we are saying all of them are 0, so because its initial condition, it is a reset condition, that in initial when we have not started comparing it cannot be greater it cannot be equal it cannot be less.

So, if that is so then we can also simplify our design of comparator 0 either we can simply assign all of them as 0 or we can simplify our design of comp 0 comparator 0 or 0th instance of my comparator by saying that g1 the output of this comparator g1 is equal to a0 b dash, b0 dash and e1 equal to a0 b0 plus a0 dash ab0 dash and less then l1 is equal to a0 dash and b0, which means that it depends just on the 0th bit it 0th bit of A and B does not depend on g, e and l all of them are 0.

So, in other words you see the previous conditions because this is 0 we can eliminate this part and because this is 0, now ideally we could have eliminated everything but we have seen that if this is 0 then we just depends on this has to be a separate condition where both of them are equal. So, basically e1 we are assuming as 1 correct and l for this we are simply keeping it 0, so that means this this whole part get eliminated.

So, my condition was wrong here, so I actually g and l both were 0, but e0 was actually 1, so the overall design is something like this. Now, this was from right to left can we design it from left to right also? Actually in case of comparator we can do either way, so even if it is designed from left to right then also it is possible to write these equations and create a comparator design which is iterative and which can which is comparing bits from left from left to right. But you can take it as an exercise and do it in your as your tutorial or a homework.

(Refer Slide Time: 19:11)



Parity Generator/detector

- Even parity is '1' if number of 1's are even

$p_{i+1} = p_i a_i' + p_i' a_i$

Boundary condition

$p_1 = a_0'$

Digital Logic Design Combinational Circuits

So, the next example we are going to take is parity generator and detector. So, before explaining this particular problem first let me explain let me tell you what is parity? Parity is usually 1 bit which helped us in checking whether the sequence is correct or not. So, how does it do? Basically that is why it is there at both the end generator and detector.

So, in generator and your parity would be one bit would be generated and at detector and again it is detected from the sequence if the bit comes out to be same as whatever is the parity bit then we say that yes there is no error. And a single bit parity can detect 1 bit of error or 1, actually it can detect 1 bit of error. So, there are two kind of par is one is called even parity another is odd parity. Even parity is 1 if the number of bits are 1 and old parity is 1 if number of 1's are actually odd. So, now how these parity bits are useful?

So, if you remember in our very first lecture we were talking about the advantages of digital design or analog design and other things, so at that time we said that a digital design has one beautiful feature that if by chance one of the bit get flipped from 0 to 1, then there are techniques which can help them recover help them identify that there is an error.

So, parity is one of the technique which can be used to detect if there is and flip of one particular bit. So, this single bit parity can check can detect if there is a flip of 1 bit, more sophisticated error correction code, error detection codes are there which can also correct if some of the bits are wrong and they can also find out if more than 1 bits are have flipped or are wrong.

So, let us focus on this even parity, in this even parity what we are doing is is the number of bits are 1 then the output is 1. Now, if I want to design a parity generator or a parity detector, what I can do is I can think of a design where it is also an iterative design, it does not depend on how many number of inputs how many bits are there in input whether input vector is 8-bits long or 16-bit or 32-bit I just need to replicate this single bit design, parity generator at all the places.

Now, since we are saying that even parity is 1, if number of 1's are even, so that means this pi would be 1, if the number of 1's previous to this, 0 to i minus 1, so a0 to ai minus 1 if all of them were even then my output would be even if ai is 0. Similarly, if all of them were odd and this ai is 1 then I can say my final output will be 1.

So, in other words I can say that in the previous parity was 0 was 1 and my current input is 0 then also my output is 1 or if previous parity was 0 and current input is 1 then also I will have a I will have this even parity. So, now if I have this single bit design now I can iterate I can say that for 0 bit also I can say p0 and a0 is the input and then for the 1st bit and for let us say it is a 16 bit vector for 16 bit also a15 and p15 would be the input and p16 is the final parity output.

So, all these p1 p2 p3 up to p15 there all intermediate inputs and outputs, but my p16 is the final parity output. Now, what about the boundary conditions? So, in case of boundary condition we have to say that the previous parity was we have to consider whether the number of bits were even or odd because 0 is considered even in case of binary, so we have to say that the previous was previous bit was 1.

So, basically previous parity p0 is 1, so that means my p1 is to be a0 dash. So, I will repeat, so it happened because we consider 0 as a even number, so 0 would be so that means p0 would be considered as a even number or even parity and that is why p0 would become 1 because p0 is 1 so we can simplify this equation as 8 a0 dash and this will become 0 so we can remove this part.

So in boundary condition the first cell will become p1 equal to a0 dash or alternatively what we can do is we can say the input parity is 1 and then we can use simple iterative design without any simplification to the boundary cell. So, this is how a parity generator detector can also be designed using these iterative design.