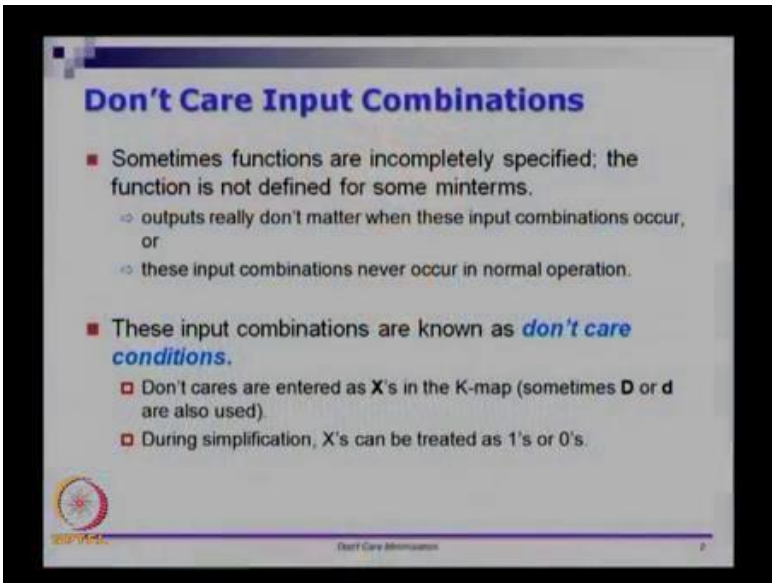
Digital Circuits and Systems
Prof. Shankar Balachandran
Department of Electrical Engineering
Indian Institute of Technology, Bombay
And
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module - 10
Minimizing Don't Cares

Hi, welcome to module ten of this course. In this module, we are going to learn minimization of Karnaugh maps in the presence of, what is called, do not care conditions. So, at times what happens is, you have minterms for which the output is actually do not care. So, what we mean by that is these input combinations. So, we talked about if you have n inputs, you will have 2 power n input combinations, but sometimes some of these input combinations may never occur in the real world. If such a thing happens what you want to do is to be able to treat the output if such a condition has to be treated in the k map as a do not care.

(Refer Slide Time: 01:08)

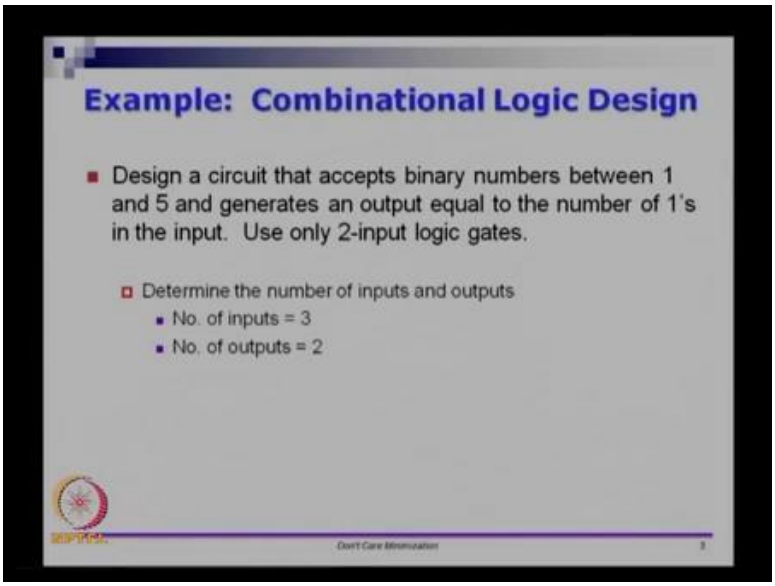


So, the reason why this is useful is that when you minimize in the presence of what are called do not cares, you can actually cut down the complexity of the circuit. You may actually start using lesser number of case than what would you have otherwise. So, the setup is as follows.

So, you have several input combinations, 2 power input, 2 power n input combinations of

which some input combinations never occur and you would want to treat those minterms in a different way. So, the do not cares are the output for these things are not 1s or not 0s. So, they could be treated as 1s or 0s because the output does not matter at all and such things are marked as X's. And when we simplify in that K-map, we can either treat them as 0s or we can treat them as 1s.

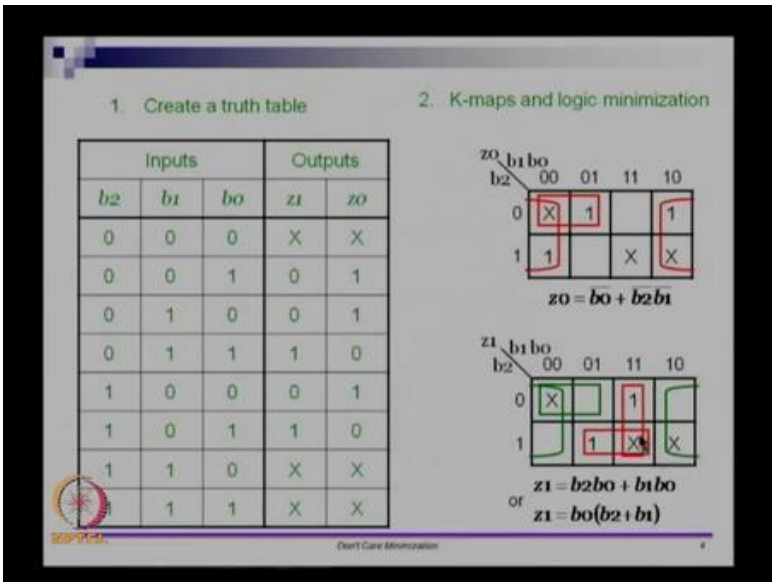
(Refer Slide Time: 01:54)



So, let us take a small example. So, I am going to give you a number, 3-bit binary number, as a 3-bit binary number I am telling you, that it can take values from 1 to 5. So, if I tell you, that there is a 3-bit number, all combinations from 0 to 7 are possible. But in this circuit the time designing, let us say, I am interested only in the numbers 1 to 5, then a number 0, number 6, number 7 do not appear in the inputs and I can treat them as do not cares. What I want here is, I want to count the number of 1s that are there in the input. So, it is a 3-bit input that I am going to give and of this you have to count the number of 1s that are present in the input and the additional constraint is that we want to use only two input logic gates.

So, given any design problem like this, the first thing you want to determine is the number of inputs and the number of outputs. So, the numbers are between 1 and 5. We need only 3-bits for that. So, that is three inputs and if there are three input bits I can have utmost three 1s in it. To represent three I need 2-bits. So, I, I have number of inputs equals 3 and number of outputs being 2. So, we are now done with the basic black box. As a block box, it is going to take three inputs and it is going to give two outputs.

(Refer Slide Time: 03:16)



Now, let us go and look at the truth table. So, the truth table is as before except for some few changes. So, let us look at the truth table. I am going to number the input bits as b2 b1 and b naught and the output bits are going to be named z 1 and z naught. So, we have three input bits and we have three output bits and for these three input bits we have 8 combinations and of the 8 combinations I already mentioned, that this minterm 0 minterm 6 and 7, these inputs are not valid inputs to this circuit. So, what we can do is we can mark their outputs as do not cares. So, z1 is a do not care if the input combination is 0; z naught is also a do not care if the input combination is 0. The same thing happens in these two.

Now, let us go and look at what happens from rows minterm 1 to minterm 5. So, for minterm 1 if you go and count the number of 1s, there is exactly one 1. So, that is marked as 01. So, so it is for number 2 and number 4. So, number 1, number 2, number 4 has only one 1 in them. So, you have those rows with only one 1, number 3 and number 5, they have two 1s in them. So, we have them as 10. So, the rest is do not care.

Now, you want to go and minimize this. So, let us see how to do minimization of this. So, the first thing is, there is nothing magical about these Xs. You can do the same thing that you did for taking a truth table and entering them to a K-map. So, as before we have b2, which goes along the rows and combinations of b1, b naught that goes along the columns and notice, that with 00, 01, 11, 10 and you take the entries from here and plug them here. So, you have X 1 0 1, that is how you should enter it, X 1 and then 1 and then come back and fill in a 0, then 10. So, 10 X come back and fill a X. So, we have that

now. Now, let us think about minimization of this.

So, if I go and minimize this you notice the 1s, that we, that we have here, are not adjacent to any other 1s. If you want to write this directly, if I treat the Xs as 0s, then I would not map this with anything, not group this with anything, not group this with anything. We have, will have three product terms in this. However, the moment we have Xs, something interesting happens. So, first thing is, if you go and look at these two 1s, they both have adjacent Xs next to them. So, I can group this four together and if you look at this one, it is adjacent to this X, I can group this together.

So, what I am going to do is, I am going to treat this X and this X as 1. Remember, it is a do not care, I do not really care about the output if some other minterm like 0, 6 or 7 comes through. So, I can as well go and treat them as 1s. So, it is a, it is a, if it is jump input, you, you get, so garbage in and garbage out. In this circuit I am supposed to be taken in only 1 to 5. If you feed 0 to this, then the output is not something that you can expect to be correct. So, now I can treat this Xs 1s. Let me group them.

So, when we group them, these four come in this quad as we call it this X 11 and X is called quad. So, in the quad we have four terms and the quad minimizes to this term b naught complement. So, you can notice that. So, b2 is travelling across two rows. So, we cannot have b2 in the term. If we will look at this column and this column, only b naught, in b naught is remaining steady at its complemented value. So, we have b naught complement and if you go and look at this grouping again, we have used X equals 1, X has 1 here. So, this belongs to the row b2 complement. So, that is 0 here and then you have b1 complement coming from these two columns. So, the overall result is b naught complement plus b2 complement and b1 complement.

The interesting thing to notice in this one is this X is not grouped at all. So, it is a do not care. And if we want to use it, we can use it as a one, but if it is not going to help us, we do not have to use it at all. So, in just taking this term and putting another product term it is not necessary, because it is supposed to correspond to minterm 7 anyway and minterm 7 is not in, is not a valid input. So, I can, I can treat this X is 0 and leave it as it is. So, what we have done is, we had three Xs, two of them we used it as 1s and one of them you can, we are using it as 0. This is perfectly acceptable because the value of X is really a do not care. We could, we should be able to use any combinations of 0s and 1s for the Xs that are there.

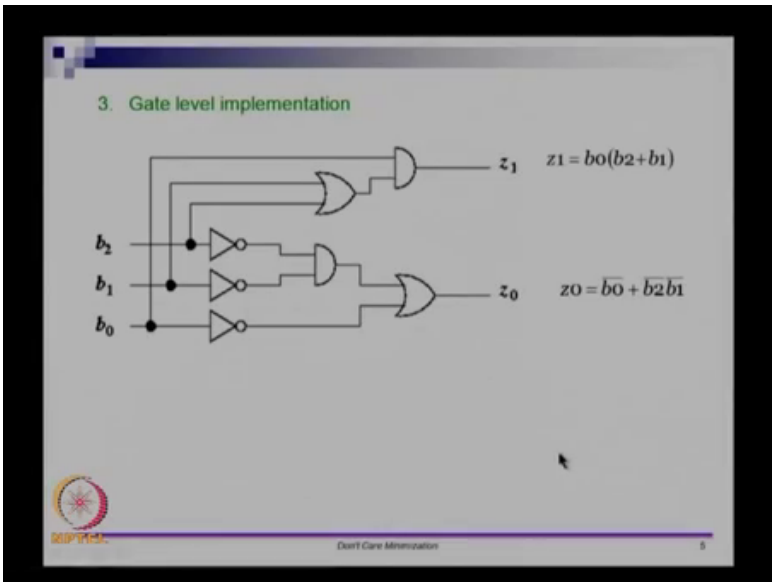
Now, let us go and look at z1. For z1 the table is as follows. So, you have two 1s. So, there are only two places where there are 1s and there are three places where that are 0s. So, we have them, the rest are Xs. Now, again if we had not done any minimization with do not cares, this will be any individual term and this will be an individual term and you would have two product terms and the two product terms would each have three literals in them. So, this will have a literal, which is b2 b1 complement been b naught and this one would have been b2 complement b1 b naught. So, each one would have add more literals in them.

Now, let us see what happens to the grouping. So, this, this setup, this red one, what you see is, there are two kinds of grouping that are shown here. One grouping that is shown in red is the sum of products and the one that is shown in green is the product of sums. So, let us start with the sum of products. So, this X if I treat it as 1, I group it together, that will cut down one term and again, I can do the same thing here. So, this term is b 2 and b1. So, this is, sorry, this term, the horizontal one is b2 and b naught and this one is b1 and b naught. So, that is what we have here. So, this is b1, b naught, this is b2 b naught. So, the sum is given here. So, it is sum of products.

In the same picture we can also notice, that if you have 0s we can group them together. So, I have these, these three blank positions, which is 0s and we have two Xs in the two corners and this makes a quad of 0s. So, if I treat this X as 0 and this X is 0, we get a quad of four terms that is a 0 and then this 0 can be grouped with the 0, which is neighboring to it and this gives us this expression. So, this quad is just b naught because if you notice, this is startling across these two, these two rows. So, b2 cannot appear, it is travelling across these. The first and last column, there b naught is steady and it is marked as 0. So, it comes in the uncomplemented version, and this one is in the row corresponding to b2 and you get b1 from the, these two columns here. So, it is b2 or b1. So, we can, so it is actually you can also verify, that these two are the same.

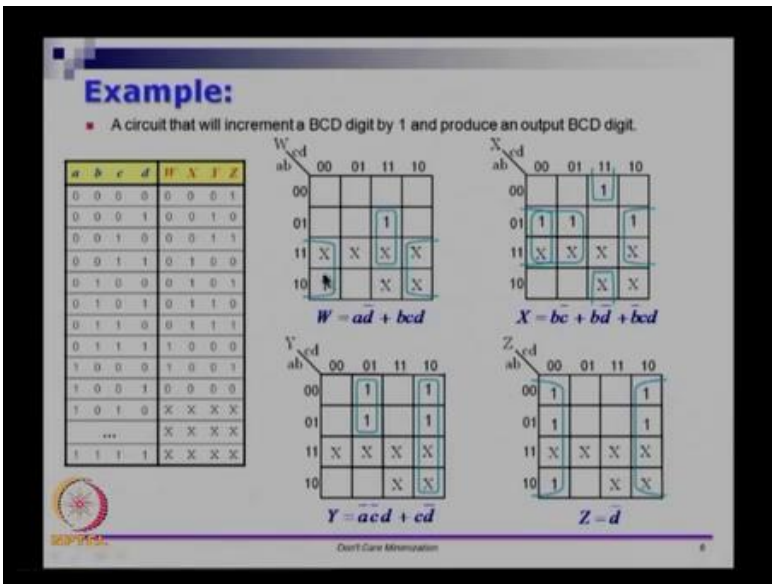
So, what this picture is showing is that the do not cares can be used as 1s or 0s. You usually treat Xs as 0s where if you want to combine them for, if you want to get product of sums, you will treat, you will start reading Xs as 1s and combined them if you want to do sum of products. And in either one you can actually leave out Xs if this is not going to help in minimizing anything. For example, in this one you notice, that this X is not adjacent to anyone. There is no point in reading this X as 1. We, we instead treat it as a 0.

(Refer Slide Time: 11:28)



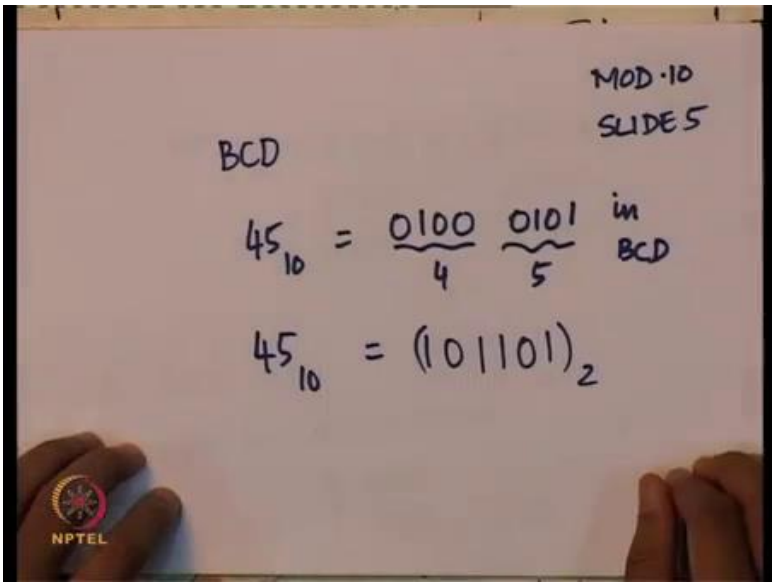
So, this is the gate level implementation for this circuit. We have b 2, b 1, b naught going into this part of this circuit as well as this part of the circuit and it is a straight forward two input implementation of the circuit.

(Refer Slide Time: 11:43)



So, let us look at another example here. In this example, we are going to convert, we are going to design a circuit that will increment a BCD digit by 1. So, BCD stands for binary coded decimal and I will show you what it is in the piece of paper soon. So, binary coded decimal, we are given an input, which is binary coded decimal and we going to increment it by 1 and produce another output binary coded decimal digit.

(Refer Slide Time: 12:12)



So, let us look at what a BCD is. BCD stands for binary coded decimal. So, let us look at this number called 45 here. So, 45 in BCD is as follows. You take that number 4 and you write the binary equivalent of it, which is 0100 and you take the number 5, you write the binary equivalent of that, which is 0101 and if you put them together or concatenate them, you get the sequence 01000101 and that is the binary coded decimal representation of 45. So, what you do, you take the decimal digits and for each digit you put the 4 digit representation for it and you put it from left to right, that is called BCD.

So, remember, BCD is not the same as binary representation. If you look at 45, the binary representation of 45 is not this. So, 45 represented in binary is given here. So, this is 32 plus 8, 40, then 44, 45. So, this is 45 in binary. So, if you notice, this is this 45 base 10, this has positional information. So, this starts with 2 power 0, 2 power 1, 2 power 2 and so on, whereas this one is not positional. This you do not assign weights to each of the bits and add them, of them. So, binary coded decimal is slightly different. So, let us switch back to the problem statement. ((Refer Time: 11:43)) So, in the problem statement what we have is, we say, that a circuit, we want a circuit that will increment a BCD digit by 1 and produce an output BCD digit. So, let us see how that can be done. So, first of all, if, if you notice, the decimal can only be from 0 to 9 and when, when we increment, 0 will increment to 1, 1 will increment to 2 and so on and 9 will increment to 0. So, this is what we want. So, we have rows from 0 to 15 here.

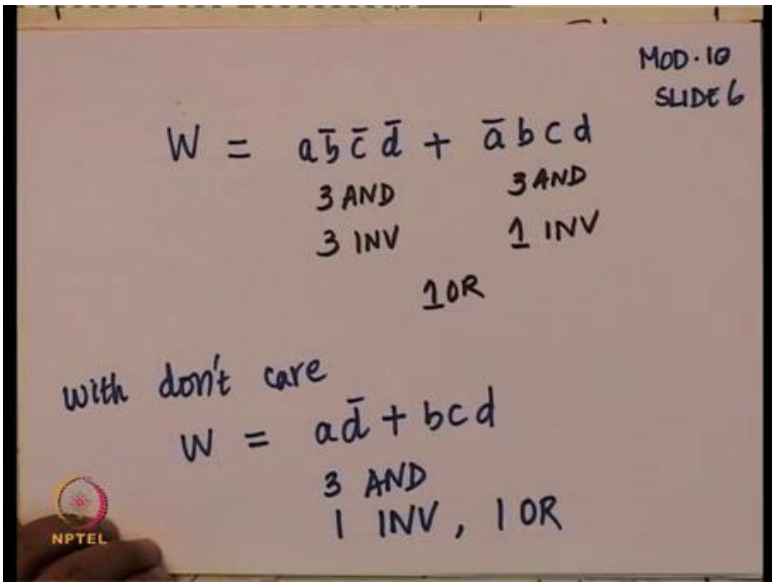
So, 0000 stands for decimal 0 here and when we increment, we get decimal 1. So, when we increment decimal 7, we get decimal 8; when we increment decimal 8 we get 9.

However, when we increment decimal 9, we get decimal 0. So, imagine, this is like a speedometer rolling from 9 to 0, right. That is what we are having there. So, in the binary coded decimal, each decimal gets one 4-bit value, right. So, number 10, 11, 12 and so on. So, this combinations minterm 10 to minterm 15 cannot happen because a single digit cannot be 10. So, ten is actually two digits, 1 followed by a 0. So, you cannot have a single digit with 10. So, the circuit is asking to take a BCD digit and increment by 1. So, in this case, it is natural to see, that the cases from minterm 10 to minterm 15 are all do not cares. So, that is what we have now.

So, now, let us go and put them in tables. I have already done that for you here. So, there are four bits, W, X, Y and Z and they are put in the respective tables. In all of these have used ab along the rows and cd along the columns and the tables you can see them here. So, they are also grouped in place. So, let us look at this one for example, this one can be grouped only with X. You cannot go to the third one because remember, we have to group in terms of two only. So, this 1 and X forms one thing, this X, 1 and X, X actually forms a quad. This is a very interesting example.

There is 1, 1, if you had left it alone, it, it would have resulted in something, but then it is in the adjacency of three other Xs, which makes it a quad. So, similarly if you go and look at these two 1s here, this would have been a group of two 1s if not for the two Xs that we have used here. And in this example, we have two 1s here and two 1s here, you would have actually made a quad combining these two. However, because of the Xs here and this one in the corner, we can group all of this into a group of 8. So, this is the power that we get from do not cares. So, notice, again we have left out several Xs ungrouped. It is just because there is an X, you do not have to cover it. So, I want to reiterate that point. So, you put an X and if you want to cover it as a 1, you cover it. If you do not want it, do not. You do not need to cover it as a, as a term.

(Refer Slide Time: 16:34)

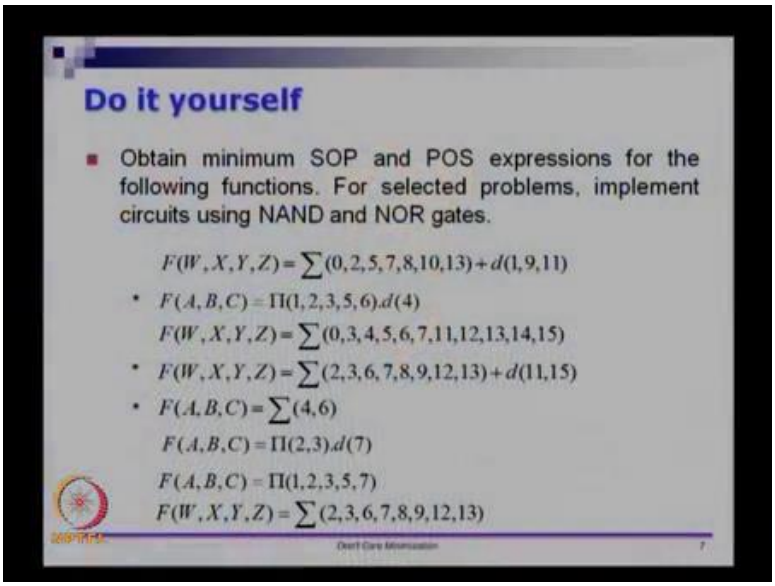


So, let us see the power of what we got from BCD. So, I have written this down in a piece of paper. So, let us look at W. If we had to minimize it without the usage of do not cares, we would had this term a b bar c bar d bar, this would have been one of the terms and a bar bcd would have been the another term. These are the two terms that would have been, that both, both these terms would have been loners in the K-map.

So, to implement this we would have needed three AND gates and three inverters. To implement this term you would have needed 1 inverter and three AND gates and you need one OR gate here. So, overall you have needed 6 AND gates and 4 inverter and 1 OR gate. But with do not care condition the expression that we have is ad complement plus BCD. So, in both these terms, some variables, some literals are knocked off. So, this requires one AND gate here and two AND gates here. So, that is 3 AND gates, 1 inverter plus 1 OR gate. So, the usage of this do not care condition has helped. This helped in minimizing the circuit for W quite a bit.

So, we should keep this in mind, if you do not need the Xs, do not use them. But if it can reduce the complexity of the expression that you are putting there, then use it as 1s. If you going to do POS minimizations, then do it the other way round. You treat the Xs as 0s and combine them with 0s, but again remember, not to combine all the Xs if you do not need them at all.

(Refer Slide Time: 18:12)



So, this brings me to end of the lecture. So, I have, I am giving you a bunch of exercises here for, as a do it yourself exercises. So, try and get them the minimum SOP form and the POS form for all of them. And for the ones, that are marked with X, see if you can implement them using NAND gates only and NOR gates only. So, go and do the exercises for all of them and implement them as SOP and POS, but also implement the ones that are shown with star using only the, using just the NAND gates or using just the NOR gates. So, go and do this. So, this will be an interesting exercise to do and it will also help you and practicing all the terms that we have learnt so far. So, this brings me to the end of module 10. I will see you, I will see you in a little while in module 11.