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 - 32
State Machines 3: State Assignment

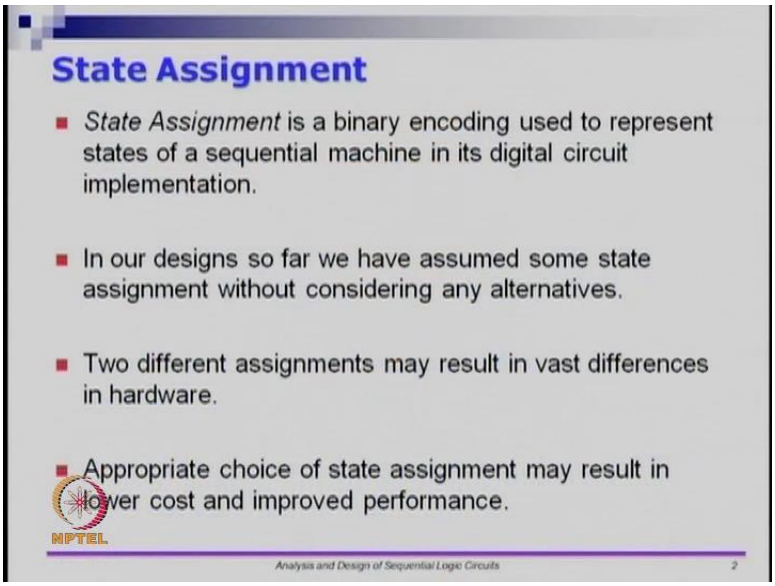
Welcome to module 32. In this module, we are going to learn State Assignment. So, state assignment is a very important part and it is good to know, how to do efficient state assignment at least for small circuits, which we do by hand. So, a state assignment it act, as a process it comes in the, it is an intermediate step. So, you start with the state diagram and the state diagram captures the functionality.

So, when you have a particular problem, the state diagram that you come up with captures, what the input, output relationships are and how this state should transition. The next step that you take is, take the states and we have to assign the bit vectors associated with it. So, so far we have been doing things like for s naught, we will give 0 0, for s 1, we will 0 1 and so on.

But, unless you do that you cannot go to the next stage, where you go and look at the excitation tables of the flip flops. So, if you have only the symbols for the state names, you cannot operate them and you cannot get bit representation. So, state assignment is a binary encoding, you start with your representation, you start with your state diagram and for each of the states, you come up with some kind of encoding. You start giving bit vector names, instead of symbolic names.

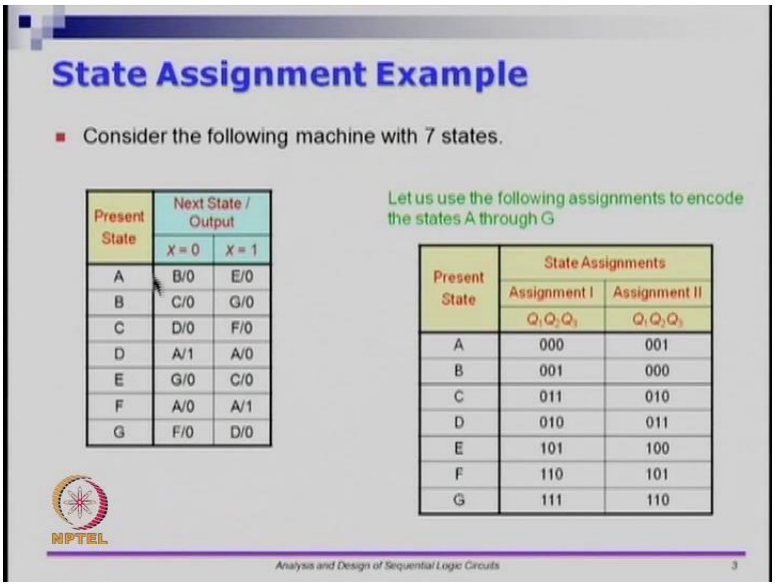
And once you have that, you can then go and look at excitation table of the flip flop arrive at input, output relationship and so on. So, we were so far just doing one state assignment. So, we come up with something and then said, if you do this state assignment, this will be the circuit and this will be the output and so on. But it is quite likely that, your state machines have multiple state assignments, which are all very competing. So, what I mean by competing is, in terms of the number of gates that you have things could be different.

(Refer Slide Time: 02:13)



So, we will see a quick example and see what this state assignment can do to different kinds of sizes.

(Refer Slide Time: 02:16)



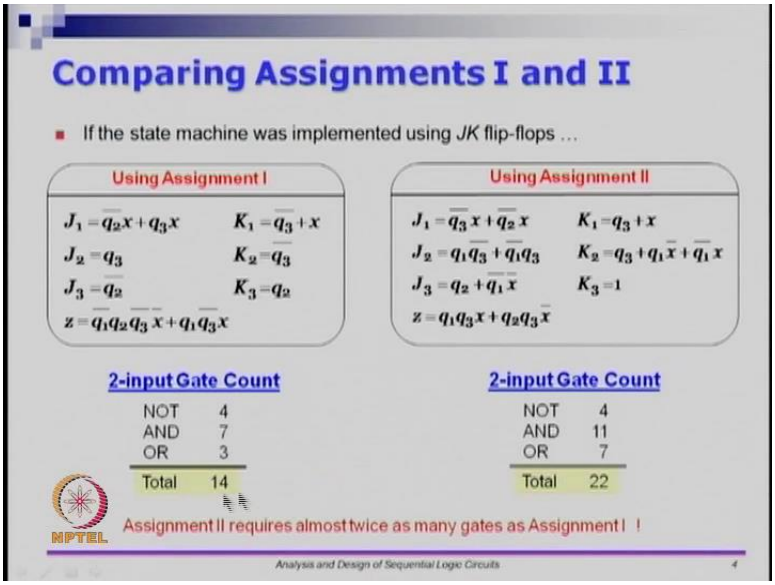
So, let us take a small example, this is a state machine, so let us forget this state diagram, we have a state table and in this state table there are 7 states from A to G and we have the next state as well as output marked in the state table itself. Now, we can do two different kinds of assignments. So, clearly there are only 7 states and for 7 states, we need 3 flip flops first of all.

So, we have 3 flip flops, we can do different kinds of assignments. So, in assignment 1,

we are going to try this, for A, we are going to assign 0 0 0, for B, we are going to assign 0 0 1, for C, 0 1 1 and so on. And assignment 2, we are doing something else. So, let us not worry about, how this was arrived at. So, if you are not given anything, if you do not know, how to do it carefully, then you may come up with in fact any random assignment. Any random assignment as long as each of the states gets a different vector that is actually a valid assignment.

So, we have two valid assignments here, none of the states repeat within the assignment, for different states we have different assignments. So, this is what I call by the state assignment. So, these are two different state assignments and let us see the implication of the two state assignments on the hardware that you will get generated.

(Refer Slide Time: 03:39)



So, we will assume that we are using JK flip flops in this example. Using assignment 1, the circuit seems to take J 1 equals this, J 2 equals this and so on. So, again without questioning how this was arrived at, I believe you know the process for that right now. And let us also believe that, it is done in the most optimal way, then the assignment 1 seems to have these equations and assignment 2 seems to have these equations.

So, you compare assignment 1 and 2, just by looking at the equations, you can probably say that, assignment 2 seems to be a bit complex. Whatever the definition of complex is, it seems to be a bit complex, let us go and validate that by counting the number of 2 inputs AND gates, OR gates and so on. If I look at the 2 inputs AND gates, so there are 4 inverters, there are 7 AND gates and 1 OR gate, which makes a total of 14 gates and

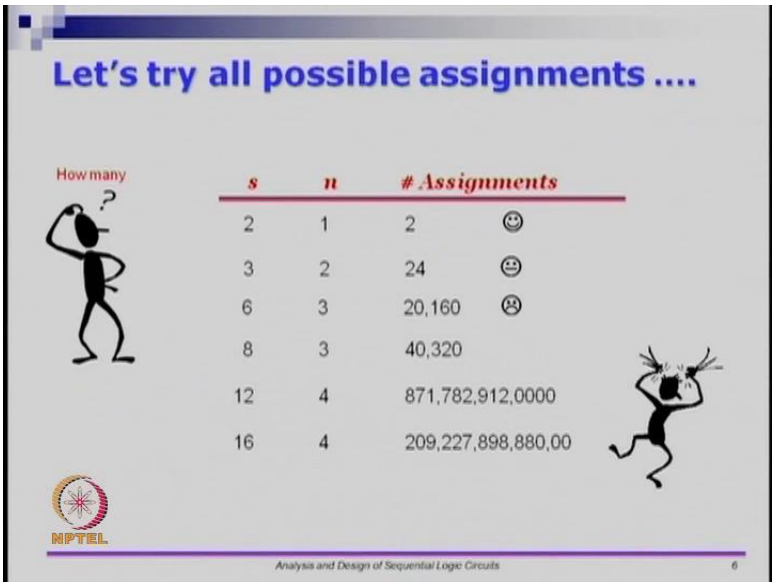
here, there are 4 NOT gates 11 AND gates and 3 OR and 7, 2 inputs OR gates making a total of 22 gates.

So, you have 14 gates here and you have 22 gates here and the choice that we made instead of using assignment 1, we use assignment 2. You suddenly have a gate count, which is greater than this one and almost 50 percent greater than the previous assignment. So, the key thing is coming up with the state machine is one thing, given a problem, you can come up with the state machine and you can also do state minimization and reduce the number of states.

So, let us assume that, that actually happened here, let us assume that the state machine is already minimized, we cannot go to anything lesser than 7 states. So, that is the case for this circuit here, this state diagram here, it cannot go to lesser than 7. The choice of which assignment we are going to make seems to make a difference in terms of number of the gates.

So, in summary there are two things which can change the total number of gates, this could either be the state diagram that you draw itself and the associated minimization that you do with it. If you come up with the bad state diagram or if you do not minimize; that can actually increase a number of gates. Second, even if the state diagram is minimized and what not the state assignment that you do can actually impact the number of gates. So, we already address the first problem in the previous lecture, we are looking at the second problem in this lecture. So, the key thing is, how do you do this assignment carefully.

(Refer Slide Time: 06:24)



If we look at s states in the state machine and let us assume that it requires n flip flops. So, if I have s states typically you need n in such a way that, it is logarithm of s base 2 and the integer that is just greater than that. So, if I take 2 states, I require only 1 flip flop and there are two possible assignments. Let us say, I have states s naught and s 1, then there are two assignments that are possible.

I will label the s naught state as 0 and s 1 state as 1 or the s naught state as 1 and s 1 state as 0, there are only two assignments, so it is not a big deal. In fact, you may actually try both these assignments, see which one gets the least amount of hardware and you can strict to it, let us try and make it is slightly larger. If I have 3 states, for 3 states, you will require 2 flip flops, but the number of assignments goes to 24.

So, the counting that I am doing is, if I have s naught, s 1, s 2, what are the different ways in which I can label them. So, I could assign s naught as 0, s 1 as 1, s 2 as 2 or s naught as 2, s 1 as 3 and s 2 as 0, there are several assignments that are possible, it looks like there are 24 assignments that are possible. And now, if I want to find out the smallest number of gates, then you will still be able to do it. You have 24 assignments, you try all the possible 24 combinations, it is a bit of a laborious work, you do all the 24 combinations, you may still come up with the least number of gates.

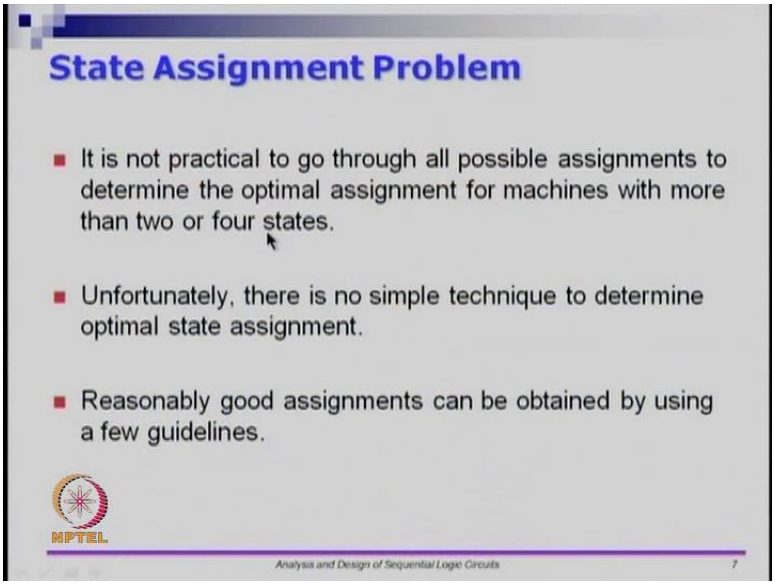
However, it quickly gets very ugly, if I have 6 states, then you need 3 flip flops. So, the 3 flip flops actually can represent 8 states, but you have only 3 states. Only, 6 states that have been used in the 3 flip flops and the number of assignments that is possible, there is

20,000. So, this is already in feasible, you cannot do this exhaustively and this gets very, very ugly when s and n increase.

So, if I have 8 states, then it requires exactly 3 flip flops, all the state should become complete, all the bit combinations will be used and that seems to take 40,000 assignments. So, you have to pick one among these assignments, which is the best and use it. And if you increase the number of flip flops 12, 16 and so on is already going into billions and trillions.

In fact, in a real world machine, the number of states is not even 16; it could be much, much, much larger than this. So, we are talking about 100's and 1000's and so on and you have something like that, the number of flip flops that is possible will also be in the 10's and 100's and number of assignments is you cannot exhaustively search. The key thing is you cannot exhaustively try all different state assignment combinations and pick the best one. In fact, you do not even do it manually, so even if you do it with the computer, it still too many assignments and you cannot finish this in any reasonable time.

(Refer Slide Time: 09:31)

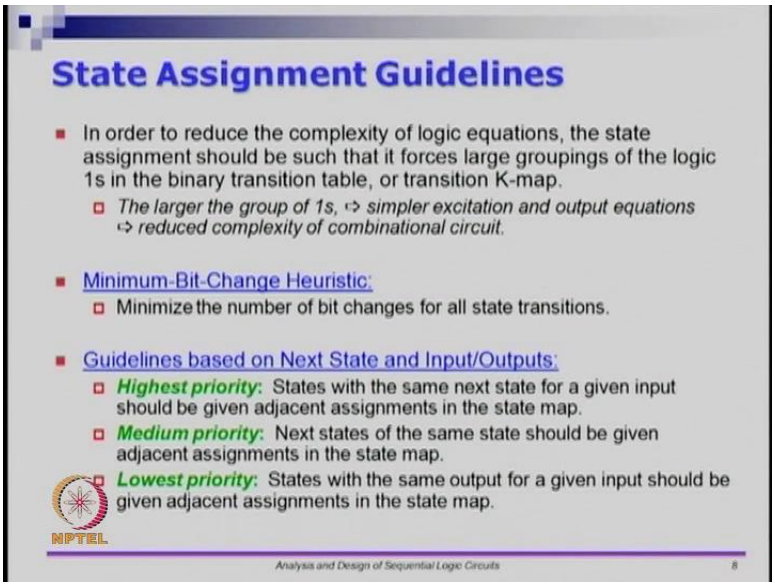


So, the state assignment problem is, it is not practical to go through all the possible combinations, either manually or using a computer and unfortunately, there is no easy way in which you can determine this. So, there is notion of optimal state assignment, the state assignment which brings you the smallest number of gates, unfortunately this is not something that you can do, there is no simple technique to do this.

What we need is, we need what are called reasonably good assignments. As long as we

follow some principles and come up with some way of solving this problem, which is only reasonably good it. It may not be the best way to do it, but it is a reasonably good way to do it. As long as we have that, then such guidelines are useful in minimizing the amount of hardware. So, what we are going to do is, we are going to use, what are called heuristics.

(Refer Slide Time: 10:26)



Heuristic is essentially a more like an intuition or an idea that it might work, but without any guarantee. So, for instance, when you drive from, let us say your home to the railway station, you may have an intuition that, this road is better than that road, this road gets more traffic than that road and so on and based on that, you may actually take a decision.

But, this decision need not be the best decision, because you assumed that some road is not going to be with traffic, you go there and suddenly, there is traffic, because of some accident. So, such things you cannot do anything about. So, in this case, the word heuristic, I am going to use that to indicate that, you come up with some reasonable intuition about, doing a particular kind of assignment is going to actually help and reducing the number of gates.

And the heuristic, as long as there is some bit of Engineering principle in coming up with the heuristic, it might help in reducing the number of states. So, the heuristic cannot be, I will come up with the random assignment that there is no intuition about, how to minimize the number of gates in that. However, if we use what is called a bit change heuristic, what we are going to do is, we are going to look at the states and you go into

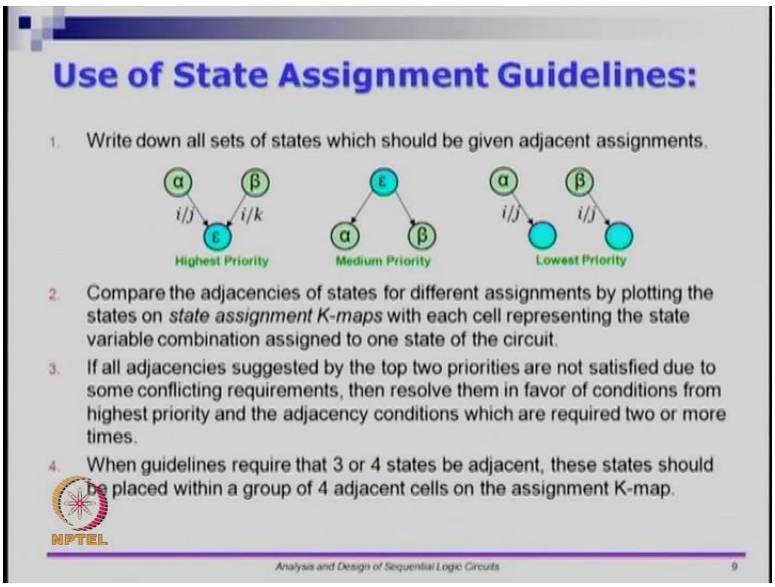
force large groupings of logic 1's.

So, I will show you a quick example in the next slide. So, we know this from the K maps, if I have 1's that are closer to each other, then you can group them and reduce the number of gates. We are going to try and do something very similar with states. So, what we are going to do is, we are going to have a set of guidelines. The guidelines are based on this; we are going to have guidelines based on the next state and the inputs and outputs.

So, the highest priority of grouping would be given to states with the same next state for a given input should be given adjacent assignments in the state map. So, if I have a state with the same next state for a particular input. So, if I have a single bit x equals 0 and x equal to 1 or two possibilities of inputs, on the next state for a given input, if you go to the same state they should be given adjacent locations.

Medium priority is next states of the same state should be given adjacent assignments and the lowest priority is states with the same output for a given input should be given adjacent assignments.

(Refer Slide Time: 12:52)



So, this picture will help you in understanding what I mentioned in the previous slides. Again, we will use this is in an example later; let us look at this picture here. We have two states alpha and beta and alpha and beta both of them on the input i is going to the state called Epsilon. So, there are two states, on the same input we are going to a state called Epsilon.

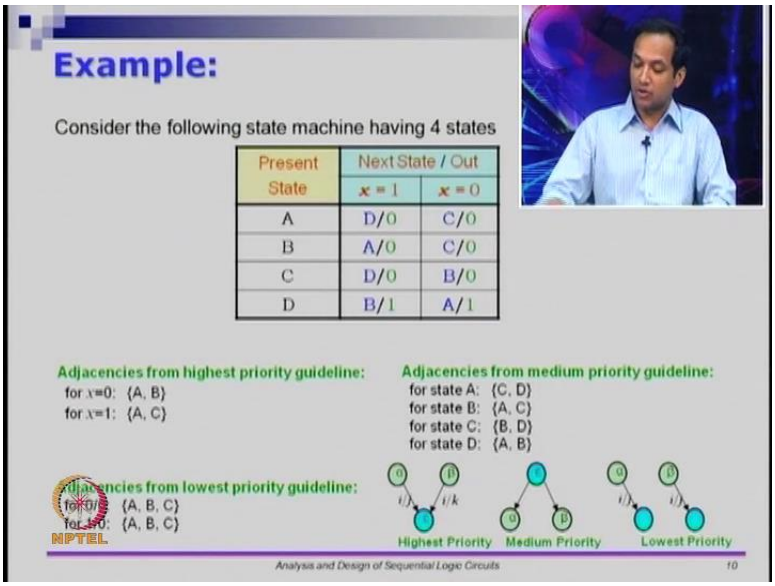
What if we have cases like this, you would want alpha and beta to have state assignments which are closed to each other. Then, if you have a state like this, if you have a single state, let us say on 1 input, you go to alpha and the other combination of input, you go to beta. Then, you would like alpha and beta to be neighbors of each other in the state assignment.

So, by neighbor, they should have bit positions, which are only few bits different and finally, if you have two different states that go to two different states, but they have the same input output combination, then you try and assign alpha and beta to adjacent states. So, this notion of adjacency means, if I take the bit vector assigned to alpha and if I take the bit vector assigned to beta, the number of bit flips that you have to go; that you have to make in the bit vector of alpha to get the bit vector of beta that is called adjacency.

So, if you are given two different states, at least there should be 1 bit difference, because you want two difference states to have two difference state assignments. If the assignments differ by exactly 1 bit, then they are adjacent. If they differ by 2 bit positions, then it is one more distance away and so on. So, the case I was making was, if alpha and beta are actually two different states, which are heading to the same state.

Then, the binary assignment of alpha and the binary assignment of beta, you should have them in such a way that, they are very close to each other. The number of bit flips that you should have for such cases should be minimized. Say, let us see these guideline, these are only guidelines, these are not rules, these are guidelines, let us see, how to use the guidelines.

(Refer Slide Time: 15:06)



So, I have put this picture here, so that, we can use this in grouping and so on. So, let us start with this state diagram, we have 4 states A, B, C, D and it will use 2 flip flops. There are several ways in which you can assign the states to A, B, C and D and we want to see how to assign it the best way. So, what we want is for A, B, C and D; we want the combinations 0 0 0 1, 1 0 and 1 1 assignment.

Let see how to do that, so let us go and look at the highest priority, highest priority says, if there are two states, which are both going to the same state on the same input, try and make them close it to each other. You go and look at this setup here, if you see at x equal to 0, there is C slash 0 and C slash 0. So, from state A and state B on input x equal to 0, we seem to go to the same state C.

So, if this is A and this is B, we are both going to the same state C on the same input x equals 0. In this case, the outputs J and K also happen to be the same. So, what we would like is, we would like A to be a neighbor of B, because if you do that, then the number of bit flips you require to go from here to here and here, to here will be minimize. Then, for x equal to 1, if you notice, there are two states A and C, both of them go to the same state D. So, we would like C also to be close to A in the bit assignment.

Let us look at the medium priority guidelines. So, from the single state, if you have two different states which are neighbors of each other, we would like them to be closer. So, from A, D and C are the neighboring states, you would like D and C to be closer. From B, A and C are the states to which you are heading out, from B, you have x equal to 1

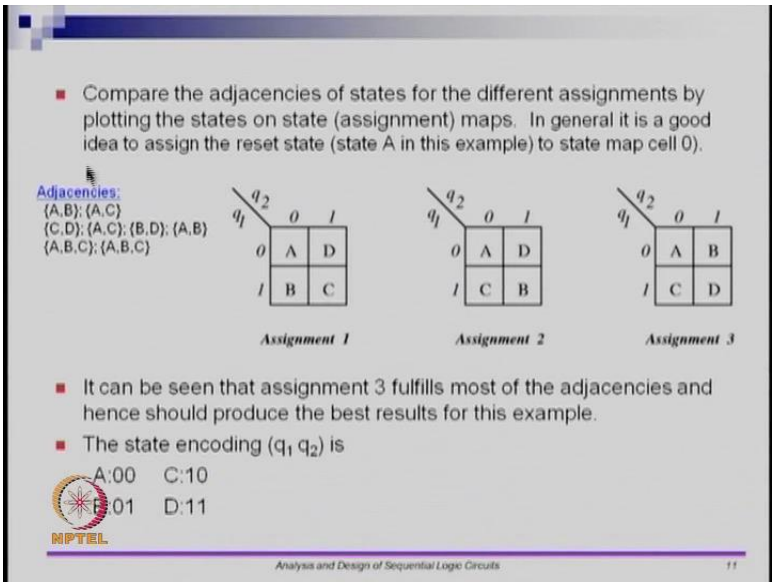
goes to A, x equals 0, goes to C, we want A and C to be neighbors of each other in the state assignment with respective C, it is D and B, with respective D, it is B and A.

So, some of these affinities are already captured here also, but it does naught matter. So, sometimes you also may have the affinities to be the same. So, what we have is, we have from the medium priority guideline, you have these affinities. And finally, from the low priority guideline, the low priority guideline says, if you have two different states and if you get the same input comma output combination. Then, you go to two different states, you do not care about, whether these are adjacent or not you make alpha and beta to be adjacent.

So, you go and look at x equal to 1, then they at x equal to 1, A, B, C all of them produce 0. So, with respective x equal to 1, you want A, B, C to all be neighbors of each other and with respect x equal to 0 again A, B, C states all of them have output 0, you would want them to be neighbors of each other. So, in this case, the affinities that we are getting seem to be the same.

Now, we have these affinities, given these affinities, we have still not made any assignment. We want to make an assignment in such a way that, this state's A, B, C and D follow this affinities as much as possible not that you can always follow it, you want to follow it as much as possible.

(Refer Slide Time: 18:37)



So, let us look at all these adjacencies. So, this is from priority one, this is from priority two and this is from priority three and we are going to do the state assignment in a such a

way that, the highest priority things are taken care of first and so on. So, let us do one assignment first. So, since these are four states, what you do is, if first draw a two variable K map.

So, a two variable K map will have 2 flip flops q 2 and q 1 and in that you mark in 0 1 and 0 1. You start with one of them, let you start with this 1 for instance, you put A in one of these positions, what this guidelines is B and C should be adjacent to A. So, in this assignment you have A and if you go and look at B, B is closer, but C is not. Whereas, in this assignment C is closer, but B is not, in the third assignment, C is closer as well as B.

So, with respect to the first priority this seems to take care of both the priorities, whereas, these 2 do not. Then, let us look at the next set of assignments CD, AC, BD, AB and so on. So, you go and look at each one of them, CD close, yes; is AC close, no; BD close, no; AB close, yes, so of the 4, 2 of them are satisfied. Let us go and look at this, this are C and D close, no; because they are not neighbors of each other, you cannot group them, if these are two ones in a truth table, you will not able to group them together.

So, these are not close to each other, AC yes; BD yes; AB is not. Finally, if you go and look at this CD yes their neighbors of each other, if these are both ones, I can group them, AC; I can group together, BD; I can group together AB; I can group together. So, it is looks like assignment 3 is good even with respect to the second priority. With respect to the third priority AB and C should all be close to each other, these are three different states, clearly you cannot make group all of them together into one thing.

So, as long as A, B, C are neighbors of each other that will work in this case, whatever you do, they will be neighbors of each other. So, in all of them we have A, B, C which is a continuous thing. So, in the lowest priority is actually not useful in this one. Let us look at the assignment; this assignment seems to have the best. So, it complies to these adjacency priorities the best.

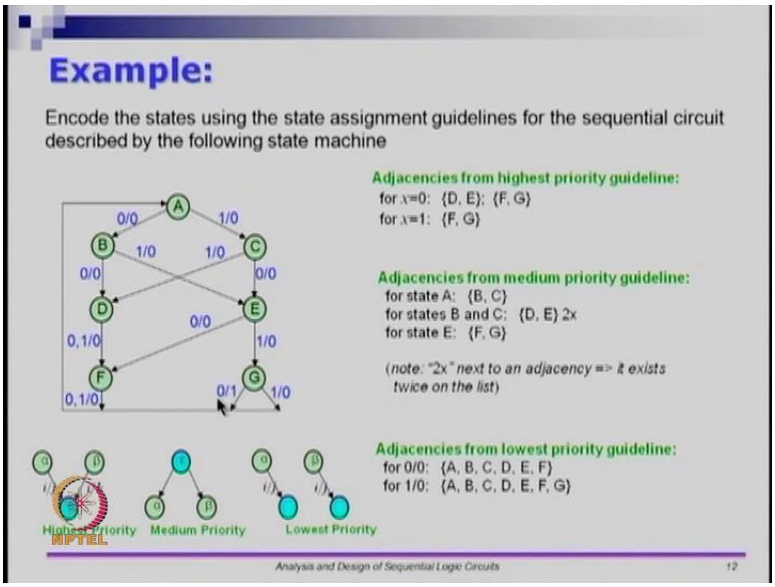
So, this assignment, if we do, if I assign 0 0 to A, 0 1 to B, 1 0 to C and 1 1 to D, then that seems to give the best result for this particular state machine. So, what I would like you to do is, go and try out for assignment 1 and assignment 2. Go and figure out the number of hardware, the hardware units, how many AND gates and OR gates are required for each one of them. Compare that with what happens when you have assignment 3, you would notice that assignment 3, actually has the lesser gates.

So, the reason why we want them to be neighbors of each other is that, if they are

neighbors of each other, later when you go and derive the next state and so on, you will be using these bit positions 1's and 0's and so on. They will become close to each other. So, we go and look at the next state logic or the output logic. So, these are combination circuits.

In these combination circuits, states which are close to each other in the table that is shown here will put 1's in the truth tables, when you do the combinational minimization; that is the reason, why we want to follow this heuristic.

(Refer Slide Time: 22:28)



So, let see a more concrete example, a bigger example. So, let say I have state machines with 8 states and I want to be able to do state assignment for this. So, again the priorities are given below. So, let us start with each one of them and see how to do this. So, let start with the highest priority guideline, highest priority guidelines say, are there two states which are coming in to the same state.

So, you go and look at this with respective x equal to 0, you go and look at are there states, where you are coming in with x equals 0 itself. So, you look at D, D has an input with x equal to 0 and F as an input with where x equals to 0 and if we go and look at G, G has no input with x equals to 0. E is seems to have something with x equals to 0, C does naught have B seems to have something and so on.

So, if I have two difference states that are converging in to the same state. So, let us look at this, D and E, you look at D and E, both D and E on 0 is going to F. So, we would like D and E to be neighbors of each other. Similarly, if you look at F and G on 0, they both

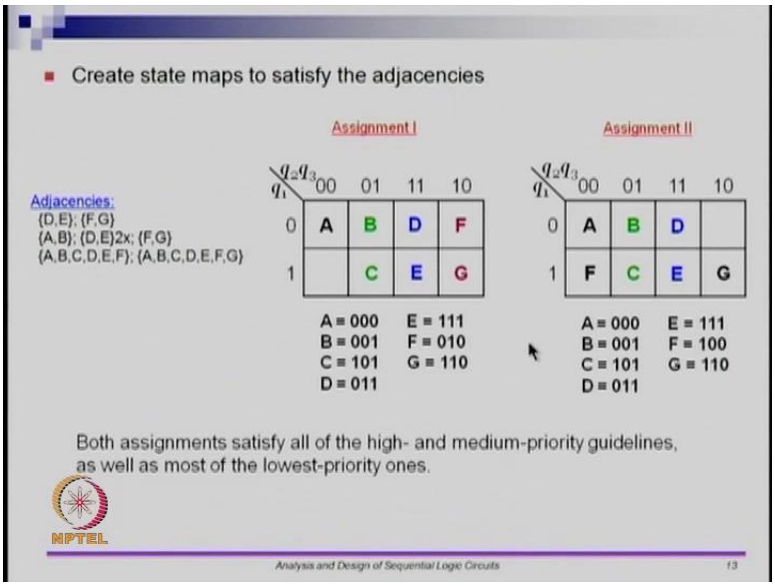
seem to go to A. So, we would like F and G to be neighbors of each other. In fact, on one also, they seem to go to the same state A, so we want F and G to be neighbors of each other, this is the highest priority.

The medium priority is, the case where from a single state you are branching to two difference states, you want them to be close each other. So, for state A, from the state A perspective, you want B and C to be closer to each other, from B and C's perspective, you want B and E to be closer to each other. And from let see from D's perspective, it is only going to F on both 0 and 1, it is going to F.

So, there is no problem from E's perspective, you want F and G to be closer to each other, from F's perspective, you are anyway going to A on both the transitions, on G, you go to A on both the transitions. So, these are the set of preferences. So, this D comma E would apply would come twice once because of B and once because of C. Finally, in terms of grouping similar outputs and input output combinations, if we look at 0 0 combinations then A, B, C, D E, F should all be together, because they all have outgoing transitions, which are 0 0.

So, you look at A, it has a 0 0 transition, B has a 0 0 transition, C has a 0 0 transition and so on. Whereas, F and G, let us, so G does naught have a 0 0 transition, G has a 0 1 transition. You would like all these to be neighbors of each other with respect to 0 0. With respect to 1 0, it look likes all the states have a 1 0 transition going out. So, all of them have to be neighbor of each other. So, you may not be able to satisfy this.

(Refer Slide Time: 25:40)



Let see how to do an assignment based on this, again the adjacencies are listed here, let see how to go about doing this in a step by step manner. So, let start with an assignment, in this case, there are 7 states, we need 3 flip flops, we start with three variable K map. We pick a state arbitrarily, let us pick some state arbitrarily, I am go to put that at 0 0 0. So, I picked A arbitrarily, I put it at 0 0 0.

Now, I am going to look at what are all the affinities for A. So, there is no high priority affinity of A, in the medium priority, A and B has to be closer to each other, we want that. So, I will not satisfy that right now, because that is not the highest property affinity, instead I go and look at the highest priority affinity; I see that B should be closer to E. So, however, D has no affinity to A. So, you go and look at this A should B close to B in the medium priority.

So, I will not placed D in the neighborhood of A and placing it away from A. So, if I take this A, this cell, this cell and this cell are all neighbors of A, I am not placing D in any of those. So, I could placed D here, D here, D here or D here, I am taking D to be here and from the highest priority, I want E to be close to D. So, I put that close to each other and then the other one that is left out is F comma G.

So, if I look at F comma G, F comma G has no particular affinity to A D or E in all these things F and G have to be close to each other, but they do not have affinity to D E. similarly, F and G by itself as there, there is no affinity to A or D comma E, only thing is in this case, we see that out of F and G, F should be closer to E, A, B, C, D and so on. So, one thing you can do is, you can do an assignment for F and G here.

So, F is still too close to this D E thing, for G, there was no affinity that was ask for. So, G is happens to be close to F; that is all. Now, B and C as still left out, we want B to be closer to A. So, there are two slots in which I can put B, I pick one of them arbitrarily and C could either be here or here, again I pick C arbitrarily. So, you can see there are lots of arbitrary things that I have done here, it is not all decisions that are all very careful.

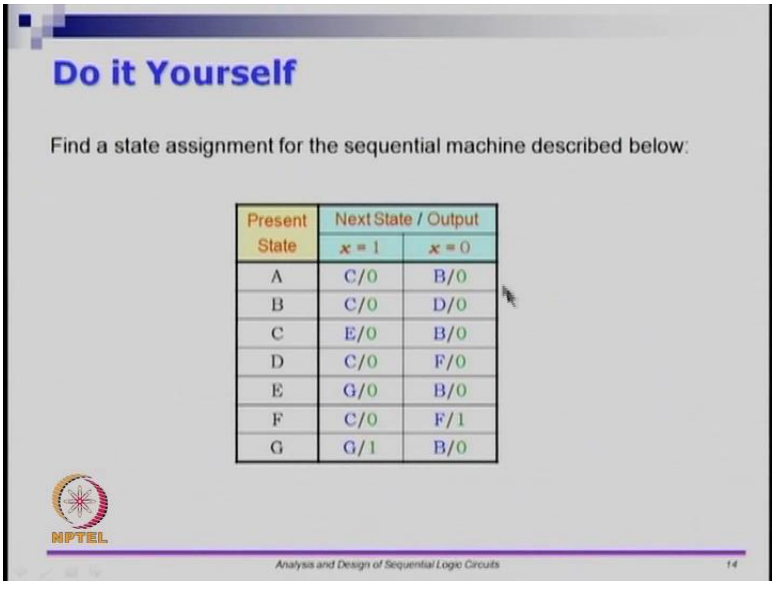
Only, decision that are careful or that, D E must be separate from A, because there is no affinity to A and D and E are neighbors of each other. And similarly, we decided that, F and G could go anywhere, it can go into any of the position, we decided that arbitrarily, the position of B was arbitrary. The choice of A, the very first think that we picked itself

was arbitrary and so on. This is one of the assignments, let us look at the state assignments, I now go and read q 3, q 2, q 1, so I go and read the state assignment.

So, for E, it is 1 1 1, for G, it is q 1, q 2, q 3. So, it is 1 1 0 and so on; that is the way in which you can gets state assignment. There are other assignments possible, so I will show you another assignment now. So, you start with A, which is here, now you go and looked at the next set of assignments, it looks like F and G. And F has some affinity to A in these two cases; I put F and G here. We still have D and E and B and D and E, so I put that here and I put B and C as it was before like this.

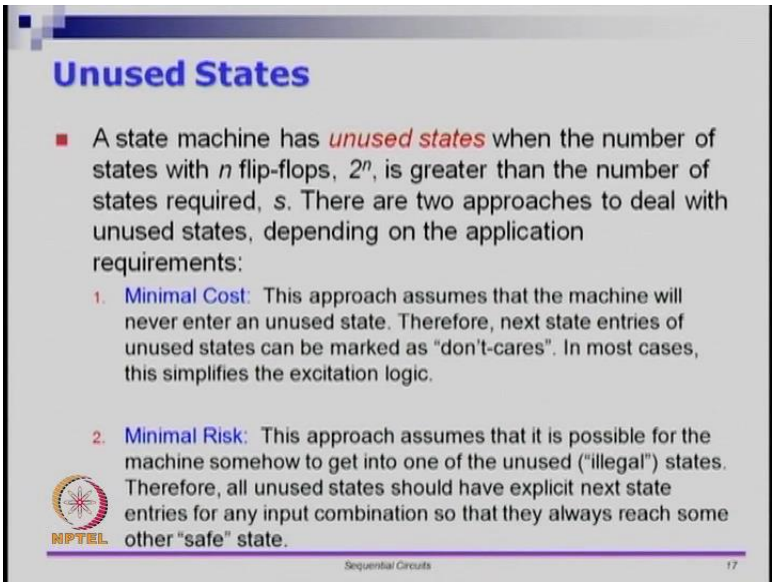
So, these F G choice, remember it was arbitrary, so I put F and G here and if I do this the state assignment seems to be this. So, both these assignments actually satisfy all the high and medium priority guidelines as well as the low priority guidelines. If we look at A, B, C, D, E, F, G they are all neighbors of each other. A, B, C, D, E, F is also some continuous chunk; A, B, C, D, E, F is some continuous chunk; A, B, C, D, E, F is some continuous chunk and G happens to be separate in each one of those. So, this is one assignment. So, for each of these assignments, you can go and try out the number of gates and see whether this one reduces the number of gates or this one reduces the number of gates.

(Refer Slide Time: 30:19)



I would you like you to do this as an assignment yourself. So, this is again a 7 state machine and these are the state transitions. I would like you do use the heuristic and come up with an assignment possible for this.

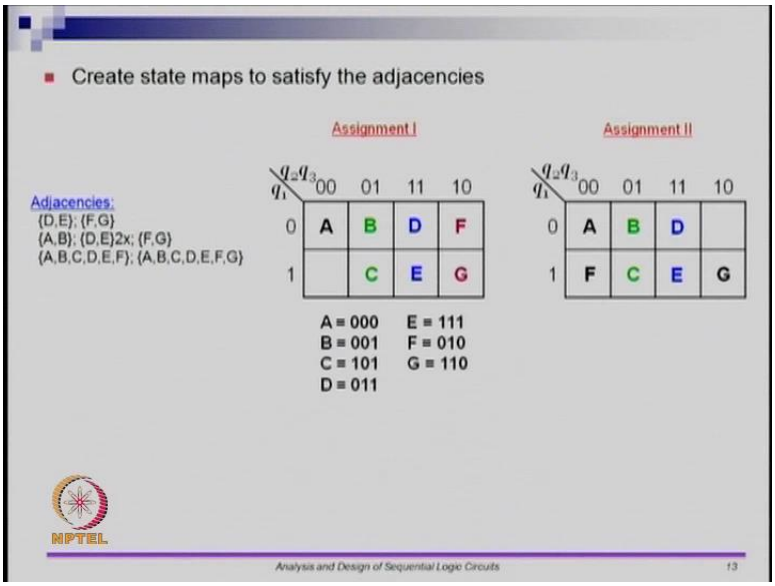
(Refer Slide Time: 30:35)



So, in this whole set up, it is possible that there are unused states. So, if I have n flip flops, it can actually represent 2 power n difference states. But you may not have 2 power n difference states; you may have only s difference states. In which case, what happens is you are not using all the possible combinations of the flip flops to represent a valid state. So, these unused states or the unused state vectors or a problem.

So, in the previous example, here the vector 0 1 0 is not assign to any state in this assignment, the vector 1 0 0 is not assign to any state in this assignment. So, these are unused states, when you look at the unused states, they can cause a problem. So, there are two approaches to deal with unused states. One thing you can do is, you can do what is called the minimal cost approach. The minimal cost approach, you assume that the machine will never enter an unused state by any mistake and you put a do not care, wherever you see an unused state. So, if you want reduce the overall cost, the unused states are all marked as minimal cost. So, this is ok, if you know that under no circumstances, you may actually enter into an illegal or an unused state.

(Refer Slide Time: 32:00)



So, remember the moment you go into a state, which is unused, there are no transitions specified for this. So, at x equals to 0 and x equal to 1, you do not know what to do there, if you know for sure that never enter that state, then we do not have worry about, how to leave the state, so that is the low cost approach the minimal cost approach. Then, there is this other approach call the minimal risk approach.

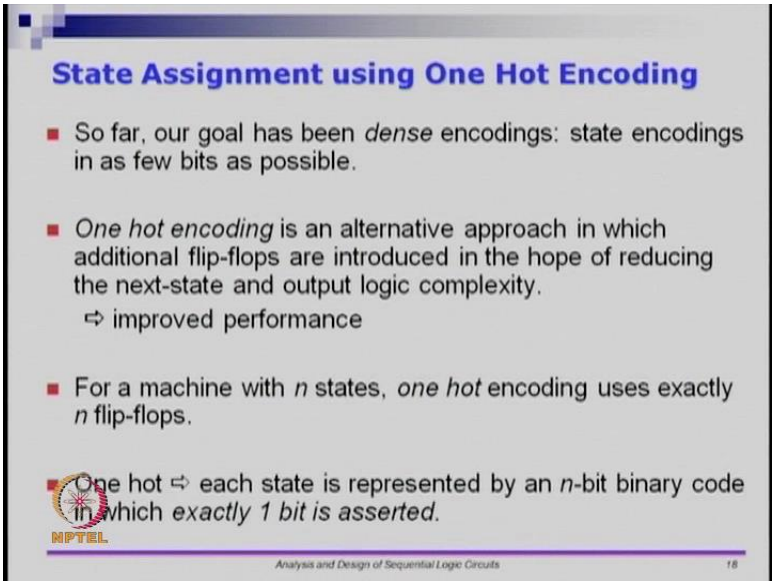
Here, what you do is by accident, if you somehow end up in the unused state. You put some state transitions, there in such a way that you get to some known state. So, both these things have issues. So, minimal cost in minimal risks, both of them have issue, but the primary reason, why something like this can happen is that, when you power up the circuit for instance, you may not what the values of the flip flops are.

You should not assume that, when I turn on a chip, automatically all the flip flop will be 0's or will be 1's and so on, we cannot assume that. So, the flip flop may actually open up or start in any which state, unless you actually put an explicit reason, so that you bring all the state to some known condition, this may be a problem. So, typically what we do is, as a good design philosophy, you go and have an explicit reset state.

So, that you power on the machine, you do a reset, so that, all the flip flops start with some known values. And if you do not do that, it is possible that you can start the circuit in an unused state. The other reason, why you may enter into an unused state is that, there is some problem in this circuit and it could be because of a time violation, more often than not; that a flip flop is suppose to be is…

So, you are suppose to go from a clean state A to clean state B; however, you go through some transition and we get stuck somewhere in the middle. So, this can happen and we will see some of these design issues and philosophy behind, how to design state machines in a later video. But there are two ways in which you can use handle unused states, either you can treat the unused states are do not cares are for each and every unused state, you go and put explicit transitions. So, that you actually end up using it.

(Refer Slide Time: 34:14)



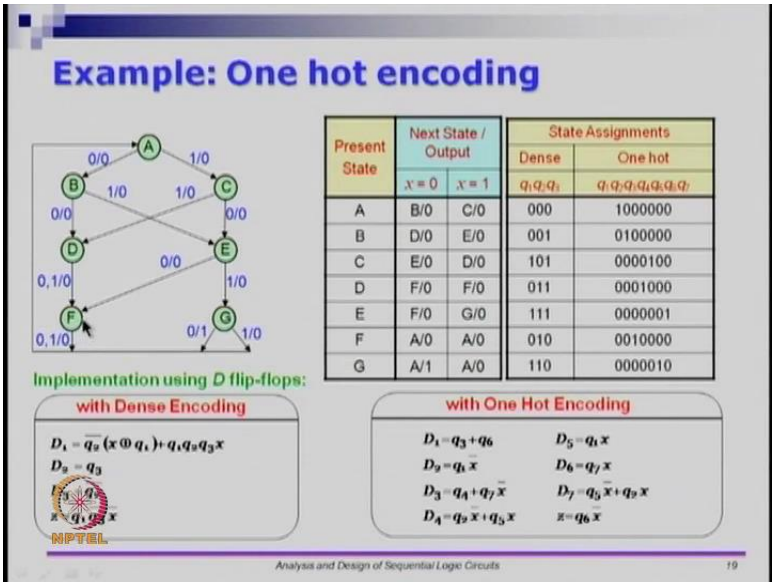
There is one final thing that I want to talk about in state assignment that is the notion of one hot encoding. So, for we have been looking at what is called dense encoding, in a dense encoding what we do is, the number of flip flops at we use, we want to be stringent about that. We want to use as few bits as possible to represent the state's. One hot encoding is an alternate approach, what we do is, we use as many flip flops as there are states.

If I want a n state machine, instead of using log n flip flops, I will use n flip flops and we will do what is called one hot encoding, one hot means at any point of time off… So, we know that the state machine should always be in exactly one state at any point of time. So, whatever inputs you give and so on, a state machine can be an exactly one state at any point of time. This is exactly one means it can be in at most one state and it should be an at least one state; that is what gives you exactly one state.

If you want reflect that, you come up with n flip flops and you after give a guarantee that exactly one of these flip flops will be on and all the other flip flops will be off, this is

called one hot. So, in one hot encoding what you do is, for a machine with n states, you use exactly n flip flops and the one hot condition is that, each state is represented by a n bit binary code in which exactly 1 bit is asserted. So, let see the example here.

(Refer Slide Time: 35:48)



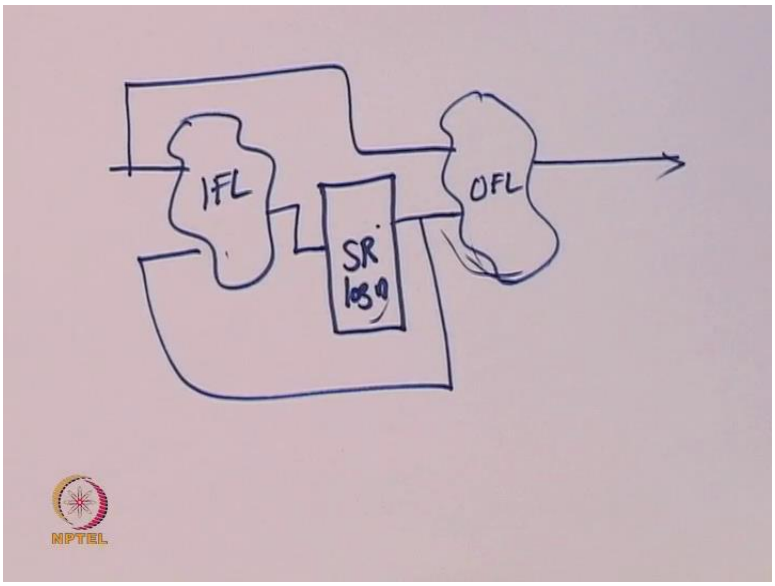
So, for the same picture that I showed earlier, this is the state diagram and for binary encoding, we have to do a lot of heuristic and so on. Instead here, what you do is, you come up with one hot assignment. So, there are 7 states, the first think you do is, you put 7 flip flops, q 1 to q 7 and if you want a represent the case, you are in state q, you are in state 1, which is for A, we assign q 1 equals 1, we assign all the other flip flops to 0.

So, this state assignment we associate with A, the assignment 0 1 followed by all 0's; we associated that B and we give for each one of them some combination. So, in this case, F happens to be 0 0 1 followed by many 0's; G is many 0's, 1 0 and so on. If you notice the state assignment here all of them are one hot. There is exactly 1 1 and all the other combinations are 0 and the state assignment also requires that you cannot have the same state vector for two different states. So, we do not have that here. So, this is called a one hot encoding.

The nice think about one hot encoding is that, if you inspect a particular flip flop, you can say that, you are in that state or not. So, if at any point of time, I give you the state register, you can go and see, what is the one and you can tell me that, you are in that state and not in any other state. So, there are lots of users, because of this one hot encoding.

So, if you go back to binary encoding, you have the canonicals, you have the state register, which in some sense is compressing.

(Refer Slide Time: 37:33)



So, let us look at the picture here, you have a state register, this uses log in flip flop for n states, you have some input forming logic and some output forming logic. So, the current states and inputs form the outputs, the current states and the inputs form the next state. So, this is the input forming logic, this is the output forming logic. So, because you have a encoded it densely, when you have to go and do things like for this state, then the output must be 1 and so on.

Technically, you are actually decoding it here with in this circuit, you decode here and you decode here, because you have encoded this state's here. Whereas, when you do this in one hot encoding, you actually do not do that. So, there is something nice that happens in one hot encoding. In one hot encoding, the decoding is actually simpler, if you want a find out, if you are in a particular state, you only inspect that bit.

So, if you want implement this using D flip flops, I go and look at D 1. So, I want to look at what are all the conditions at which q 1 should be on. For the dense encoding, it is seems to be some complicated equations, so I have to go and derive it, I have to put it in the K map and so on. Whereas, if I want to go and look at each one of these conditions. So, you go and look at this D 1 should be on, when either q 3 or q 6 is on.

So, what that says is, if you go and look at A, what are all the conditions under which you will come to A, you should be an either F state or it should be an G state. So, go and

look at F state for at F state, you have q 3 is on, G state, you have q 6 is on. So, if either q 3 or q 6 is on, the next state you will go to A; that is what this means. Let us go and look at D 3 for instance, so we are looking at state associated with q 3.

So, let us look at the state as F, F is the state associated with q 3, let us look at all the combinations in which you come to F. If you are in D, you go to F, whether it is 0 or 1 you go to F, but if you are in E, if the input is 0, you go to F. So, this is something you can derive by looking at what is the state assignment for D, for state assignment D, we have given q 4, for state for assignment of E, we have given q 7.

So, if q 4 is on, no matter, what x is you will come to F, so that is q 4, if you are in q 7 and if the input is x, input x is 0, then you come to F; that is captured by q 7 x bar. So, by just looking at the picture without doing any minimization, I can write this equation down, you do not have to do anything at all. And if you go and look at the outputs, let us go and look at the circuit, where the output is, so it seems to be that, the output, let see where the one is, here.

If here in state G with x equals to 0, you are producing output of 1, you go and look at the state assignment for G; that is q 6 So, if you are in q 6 and if the input is 0, which is x bar, then the output must be 1; that is all. So, it is as simple as that; there is no decoding at all, the decoding is as simple as looking at a particular bit position and inspecting whether it is 1 or 0.

So, what we have is a very nice encoding mechanism and you do not have any logic at all, even though you gives a lot more flip flops, the amount of logic that you use for decoding is very, very simple. So, some times this one hot encoding is preferred over dense encoding, because the decoding logic. So, every time I have to inspect which state and so on, it gets reduced.

So, this is much nicer, especially when you use what are called FBGS, FBGS have a lot of flip flops, doing one hot encoding is actually preferred and FBGS compare to other techniques. So, this brings me to the end of this lecture and in module 32, we saw state assignment and we saw quite a few heavy duty things in module 32 and I suggest that you go back and work on the example.

So, I have been repeating this in every video, I suggest that you go back and look at the examples, until you do that you may not get all the concepts. So, I am showing you one example, unless you try other examples, you may not get all the concepts and if you do

not understand something, please come back on the forum and do ask question on the forum about these examples are something that you make up.

So, thank you and I will see you in the next module.