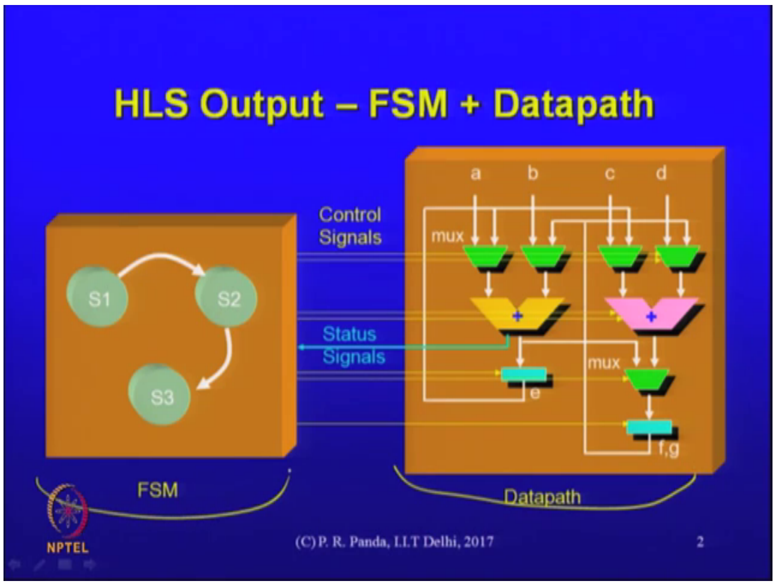
Synthesis of Digital Systems
Dr. Preeti Ranjan Panda
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi

Lecture – 16
Finite State Machine Synthesis: Introduction to FSM Encoding

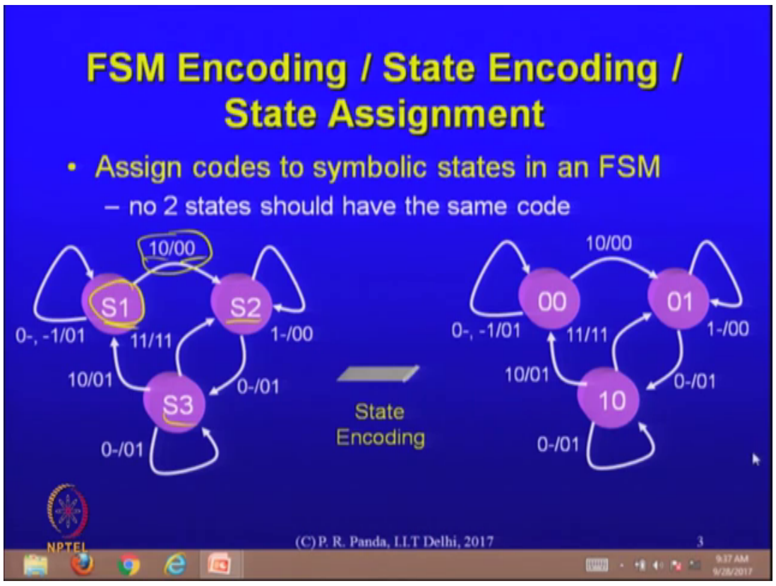
Different topic today; FSM encoding is an important step that takes us closer to hardware detail from the behavioural level of abstraction.

(Refer Slide Time: 00:36)



If we look at the output of high level synthesis, there was this data path which looks very structural, right, this part of it is closer to a net list, whereas, this part is still symbolic as of now nowhere close to a net list. So, we need to do something about it that is what this FSM encoding is about.

(Refer Slide Time: 01:07)



FSM Encoding / State Encoding / State Assignment

- Assign codes to symbolic states in an FSM
  - no 2 states should have the same code

It is also variously called state encoding state assignment and so on, all of them referring to the same problem; what is the problem? We assign codes to symbolic states in an FSM. These are the symbolic states S 1, S 2, S 3, we can name them whatever usually you choose some meaningful name and for the states, this is a mealy machine, we understand this annotation right 1 0 refers to the condition

And the 0 0 here or refers to the actions, we need to assign codes to the symbolic states. So, that no two states should have the same code, that is the objective; what does the encoded FSM look like, if this is the original FSM, the encoded FSM would be different in what way? What do we mean by assign codes to symbolic states?

Student: (Refer Time: 02:24).

Yeah. So, what do we do to the FSM to encode the states?

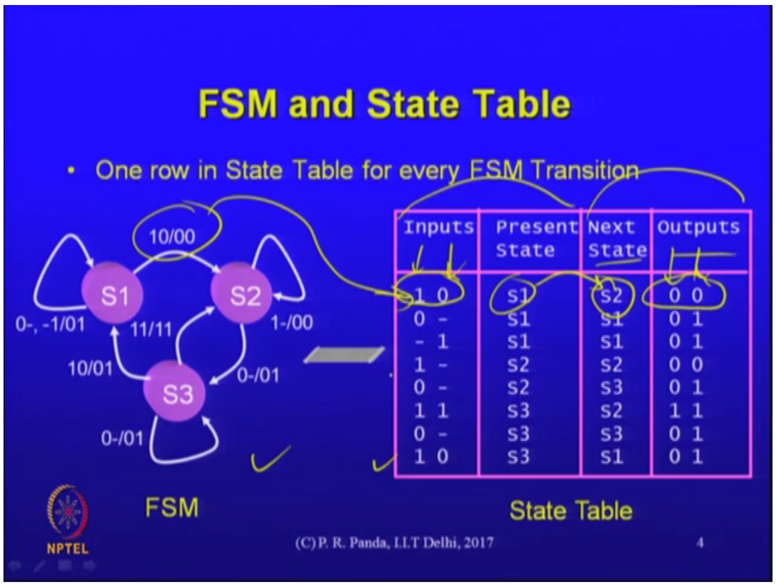
Student: Thank you, try some binary code.

Yeah, now you replace these symbolic names with some codes 0 0 0 1.

Student: (Refer Time: 02:45) in a way at actual test level it will be seen as a sigma log of variable with certain size as the bit vector.

Yeah. So, there are some decisions we need to take as part of the state encoding thus length of that bit vector is one of them, but of course, what is the value that we give to each state is the other one; that is the state encoding problem, this is just a specification of what we do, there is the question of an optimization function, what should be optimized as the result of this action, clearly, there are many different choices for state encoding. So, that is the FSM encoding problem. Let us define the optimization function a little later.

(Refer Slide Time: 03:30)



FSM and State Table

- One row in State Table for every FSM Transition

Inputs	Present State	Next State	Outputs
1 0	s1	s2	0 0
0 -	s1	s1	0 1
- 1	s1	s1	0 1
1 -	s2	s2	0 0
0 -	s2	s3	0 1
1 1	s3	s2	1 1
0 -	s3	s3	0 1
1 0	s3	s1	0 1

State Table

NPTEL

(C) P. R. Panda, I.I.T Delhi, 2017

4

But first let us understand that there is the diagram that is used for illustrative purposes. Equivalently, one could look at the FSM as just a state table these two things on the left and right, these two representations on the left and the right, they are equivalent. So, I have the input part and that is the output part of the state table and we have these columns, the state table that refer to an input section a present state section and next state section and an output section, how many columns do I have?

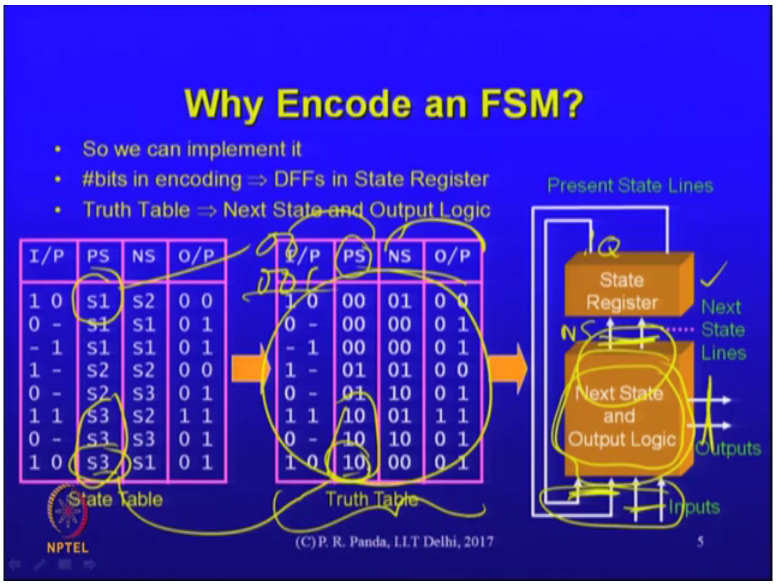
The inputs; this refers to the number of inputs assuming, these are all single bit number of inputs, I have present state that is the state next state is also the state outputs, again there will be as many bits on the output side as there are output bits in the FSM, these two are identical, they contain identical information every row of the state table corresponds to what in the FSM diagram.

Student: A transition.

A transition if I say on 1 0 I transition from S 1 to S 2 and in the process the output is 0 0 I should be able to tie that to a row where I have essentially captured the same information condition is 1 0 S 1 to S 2 that is the transition and the output is 0 0. So, for every transition that I have I can expect a row that is capturing the same information.

So, it is a same thing, FSM is fine, the diagram is ok, from a visualization point of view, but from an automatic processing point of view clearly that state table is something that will be useful. So, why do you encode the FSM?

(Refer Slide Time: 05:36)



Why Encode an FSM?

- So we can implement it
- #bits in encoding ⇒ DFFs in State Register
- Truth Table ⇒ Next State and Output Logic

I/P	PS	NS	O/P
1 0	s1	s2	0 0
0 -	s1	s1	0 1
- 1	s1	s1	0 1
1 -	s2	s2	0 0
0 -	s2	s3	0 1
1 1	s3	s2	1 1
0 -	s3	s3	0 1
1 0	s3	s1	0 1

State Table

I/P	PS	NS	O/P
1 0	00	01	0 0
0 -	00	00	0 1
- 1	00	00	0 1
1 -	01	01	0 0
0 -	01	10	0 1
1 1	10	01	1 1
0 -	10	10	0 1
1 0	10	00	0 1

Truth Table

(C) P. R. Panda, I.I.T Delhi, 2017

Student: Differentiate between the states basically in hardware everything is 0 1.

Student: Sir, you have to assign it to a variable.

Now, what values is process of an of implementing it somewhere that symbolic name needs to be translated into 0s and 1s. So, that I could realize whatever the logic is in terms of some gates; gates understand only 0s and 1s.

So, there is a need for the motivation for encoding it, first question is how many bits should I have in that encoding that bit there is relevant because we know the standard template right, how that logic looks like the hardware looks like that would implement a particular FSM, every bit in the state encoding translates to what or a longer or a shorter encoding length has what implication on the hardware that is inferred.

Student: It is the bit of (Refer Time: 06:53).

I have to answer this question; how many bits will I use for representing a state encoding means I have to assign a bit stream right for each of them 0 0 or 0 0 1 and so on how many bits should I use.

Student: 1; 1 or 2 or number of number of states number of number of.

I take the number of states and the minimum, I need is the number of bits that are needed to at least get so many distinct numbers that is the logarithm to the base two of the number of states so.

You take the ceiling because it is an integer that we need, here why do we use; that could I have used a longer state register that translates to, but why did I you took the minimum, but it was not necessary to take minimum.

Student: More storage will be.

More storage will be required, yes, each bit translates to a flip flop, it is a state register that I found that consists of a number of flip dependent flip flops and if I use 2 bits, then I have 2 flip flops, there if I have 3 bits, then I have 3 flip flops. So, of course, there should be an attempt to minimize the number of that helps in some way, but if I had an area metric for optimization, I will try to minimize the number of flip flops and therefore, I try to use the minimum length encoding here.

Student: What is the other trend of using minimum number?

I will get to we have not defined the optimization function yet. So, maybe we postpone that discussion ok. So, when I do this encoding right S 3 becomes. So, I just replace the occurrences of the symbolic state by the respective encoding all these S 3s are replaced by all the equivalent codes, then I have essentially what is a truth table ok, what you see here is a truth table, what can I do with a truth table? Now that it is in this representation what is the next step?.

Ultimately, I want to generate hardware out of this logic, this somewhat represents the combinational logic that is involved, right, what is the next step after the truth table, you can generate gates out of this; this truth table is nothing, but the equivalent of a Karnaugh map or whatever, right, all the bits are there all the do not cares are also nicely specified. So, I can proceed with my logic optimization and realization of a gate level; netlist from that truth table, of course, the architecture may be illustrated like this, I have a state register that captures the current state of the system what is there in the state register.

Student: The flip flops.

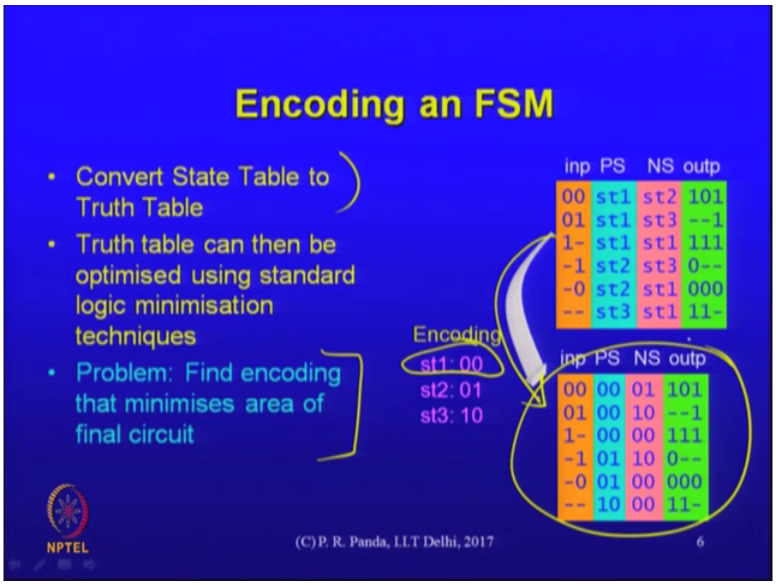
The flip flops; it consists of only the flip flops; nothing else and if I have a 2 bit encoding, then that state register will have 2 flip flops, the standard representation is you just assume it as a D flip flop. So, that whatever the next value is going to be that is what is computed in the logic here and that goes as input at the D input of the flip flop and what you see here, these are the Q outputs of the flip flop, those are fed back into that same combinational logic and they become the present state right that computation here is the next state.

Computation that goes into the into the D flip flop that is essentially preparing the data for the next state that is the architecture this combinational logic consists of 2 parts, there is a next state logic and an output logic, of course, all of these are the output parts, right. So, these are computed the inputs are these.

So, these are input the inputs and the present states are these inputs to the next logic and these two consisting of the present state because of the Q inputs and these are the external inputs into the FSM what are the outputs there are 2 output sets, one is the external output from a HLS output point of view, you can see that these are the control signals that go from the FSM into the data path, the other set of outputs are these are the next state outputs right.

So, that combinational logic is all of this that is essentially a translation of this logic that is specified here in the truth table that is the overall flow and the hardware architecture, we are assuming any questions on just the overall picture, we will get to then the question is how do you take that decision that S 3 translates to 1 0, what is the basis and how do you take that decision that is the FSM encoding problem, but it is embedded in the bigger problem that is this ok.

(Refer Slide Time: 12:44)



Encoding an FSM

- Convert State Table to Truth Table
- Truth table can then be optimised using standard logic minimisation techniques
- Problem: Find encoding that minimises area of final circuit

inp	PS	NS	outp
00	st1	st2	101
01	st1	st3	--1
1-	st1	st1	111
-1	st2	st3	0--
-0	st2	st1	000
--	st3	st1	11-

Encoding
st1: 00
st2: 01
st3: 10

inp	PS	NS	outp
00	00	01	101
01	00	10	--1
1-	00	00	111
-1	01	10	0--
-0	01	00	000
--	10	00	11-

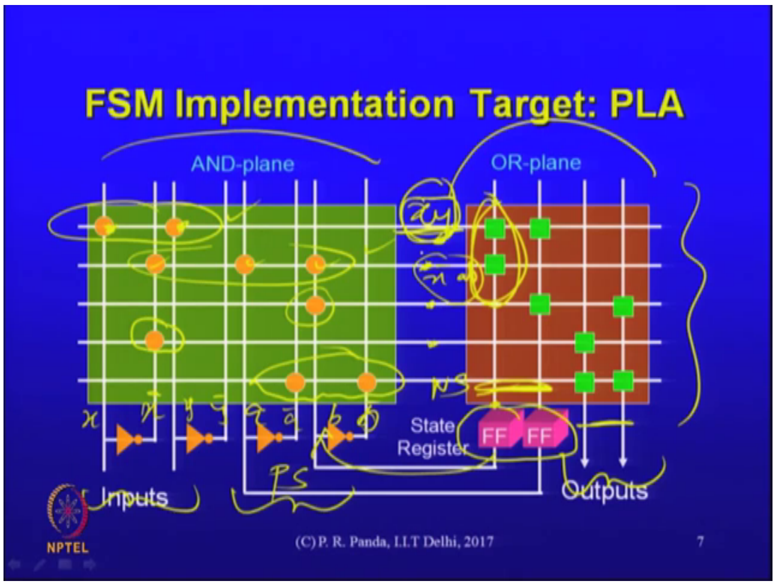
(C) P. R. Panda, I.I.T Delhi, 2017 6

I have to actually specify an objective by itself, it is hard to argue that one encoding is better than another encoding, you have to say in terms of what is it better. So, our objective is the translation is from a state table into a truth table; that truth table can then be optimized using standard logic minimization techniques. So, we should hopefully be familiar with the basic ideas, but we will talk about this in the next step.

But I have to specify an objective and for now, let us say that the objective is to find an encoding that minimizes the area of the final circuit; what is important to note here is that as part of this encoding step, we are not generating the circuit, we are only anticipating; what will happen later encoding. In fact, is just this you take the symbolic FSM and perform just these encodings?

So, that you get this truth table; that is all, but it is hard to see at this level, why one would be better than the other. So, somehow you have to argue in terms of what happens, later if I take this decision, then the later logic optimization step can benefit in this way or that way that kind of an indirect argument has to be made as we make the choices for the encoding.

(Refer Slide Time: 14:12)


FSM Implementation Target: PLA

This is not so trivial and there is a lot of dependence on what that target architecture looks like how are you going to implement that FSM, we did have a truth table representation, truth table is also a high level representation, it is only a specification of the functionality of the logic, you can say that you will use an array logic kind of a a target in which case, there is some such construction, are you familiar with what a programmable logic array looks like.

So, it could be this kind of a target as you know; there would be an and plain and there would be an OR plane in that AND plane you would be generating, what are these? These are the inputs to my system, these are the outputs if combinational logic that that PLA actually ultimately implements, I have a couple of flip flops here outside that is my state register and these actually go back over here forming the inputs that is essentially the present state right and these outputs here are the next state computations and these other outputs are the external outputs of the FSM.

So, that is the implementation. So, what would these be that this. So, each of these; what is the meaning in this implementation what we are saying is if you had the input variable if this was x, then this is x prime right, x bar if this was y then this was y bar.

So, such a connection here in that AND plane means what means you select these are what are these rows mean what and columns mean what in the real life that is a initially just in the AND plane, there we have introduced a connection that that is the programmable part of this logic I have to decide; how to program it and programming it essentially means that I decide on those connections which connection be there and which one should not be there I introduce these two connections to mean that this line has some logic what is that logic.

Student: x bar.

Since; so, I selected x and I selected y the AND of those is what that line there represents, similarly I selected all these 3 here for the next row. So, that would refer to x prime y is not there and if these present state lines, you call them whatever a and b, then a is there and b is there.

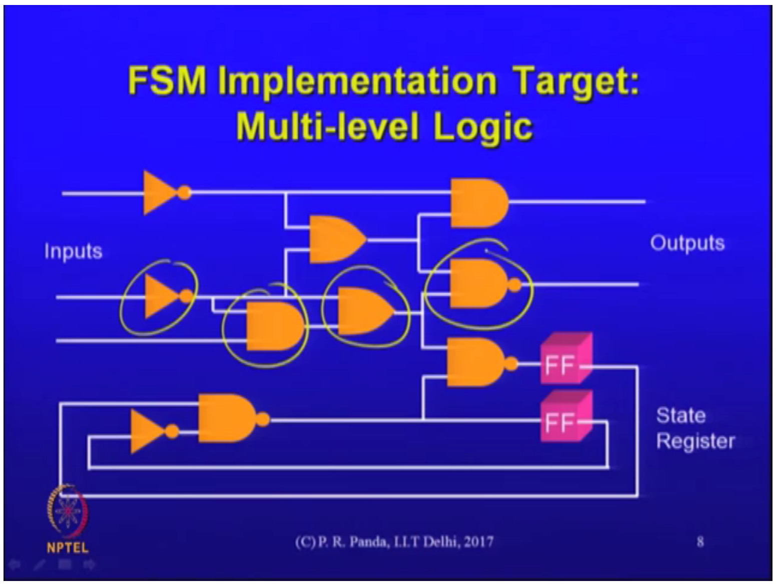
So, that is the meaning and so, terms are for product terms are formed and you select from the AND plane of the PLA which lines you want and the complemented versions are also available. So, that we can form any expression so, but these are all individual terms and terms there may be as many literals as many variables here as the number of connections you make there. So, that is what these are what is there in the or plane.

Student: Sir columns.

There, they are the columns they represent what that these. So, if I call them a and b, sorry, this is a prime in this is prime, then if I have chosen these two, it means that x and y is what is selected. So, x y is one enter and this is the other and term corresponding to x prime a b.

So, that is x prime a b and through selecting both of them, we mean that that output line here, it is actually a next state line is this term, this product term plus this product. So, in the AND plane, the horizontal lines are used to compose the product terms in the or planes the vertical lines are used to collect the appropriate product terms to form whatever expression you want that is if the target was a PLA.

(Refer Slide Time: 19:10)


FSM Implementation Target: Multi-level Logic

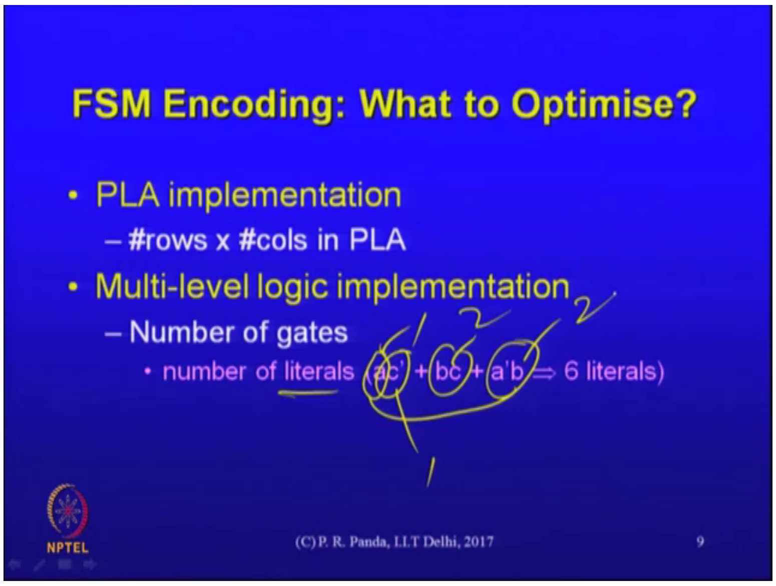
(C) P. R. Panda, I.I.T Delhi, 2017 8

It could also be that I have multi level logic the PLA is an example of a two level logic structure, you have and terms and or of appropriate and terms nothing more than that the only two levels, but you do not have to do it that way, you can have any number of levels that is one level this is another level there is a different level and so on in general, of course, nobody is stopping you from having any number of levels.

But that target architecture is different, you could have a standard cell based architecture where you pretty much have random logic and you can implement any number of levels, if you use one of those predefined architectures like a PLA, you are constrained with respect to what kind of expressions you can implement there this target architecture has to be specified to the FSM encoding strategy because the way you would encode.

Even if it is for minimum area the way you would encode if the target architecture were PLA or an FPGA for example, may be different from the way you would optimize it if it is an the multi level logic and ASIC library is used to implement a standard cell library is used to implement it.

(Refer Slide Time: 20:31)



FSM Encoding: What to Optimise?

- PLA implementation
  - #rows x #cols in PLA
- Multi-level logic implementation
  - Number of gates
    - number of literals (ac' + bc + a'b ⇒ 6 literals)

(C) P. R. Panda, I.I.T Delhi, 2017 9

That needs to be specified. So, let us identify; what do you need to optimize, if I say area is our target to begin with, then what should you optimize in the PLA of implementation what do we optimize.

Student: Number of connections.

Number of connections what difference does that make in the.

Student: I can put more logic in the same area.

The number of connections would it have an implacation on the area of the PLA.

Student: No.

No, so.

Student: Area of the logic.

So, that is the area, right, this is my area that I would like to minimize, what is that actually available in my hands.

Student: You can put a bigger logic FSM; I mean efficient utilization of the resources.

Let us assume that we are going to construct a PLA that implements our logic FSM is there I need to realize it in terms of a PLA. So, I am looking for the smallest PLA structure that would yeah; that would be good enough for our logic. So, what does that translate to what do I actually need to optimize here in this picture in order to minimize the area of the PLA.

Student: The number of employees.

First, what is a very high level model of the PLA area? This is a rectangular structure right its area is determined by.

Student: Length; length.

Length and breadth; that is a great beginning, length here translates to what and breadth here translates to what?

Student: Number of rows and number of columns number of rows and columns.

Number of rows and number of columns that is a great improvement of refinement of all the abstract idea of area of PLA; what determines the number of rows what determines columns?

Student: (Refer Time: 22:39).

Number of inputs just number of inputs number of columns is what here you determine by what things?

Student: Number of inputs.

Number of inputs and number of outputs also because those are also occupying space, right, there is a column dedicated to every output. So, these are determining the number of columns; number of rows is determined by.

Student: Number of expressions we are (Refer Time: 23:07) min term.

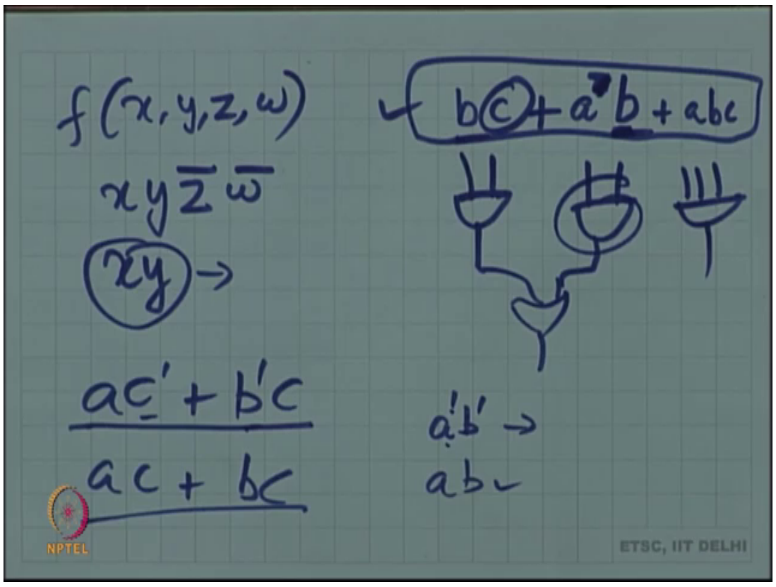
Can you be more precise; what is a min term; if you have a logic of 4 variables, what is an example min term.

Student: You cannot break it down further beyond what it is.

Give me an example min term.

Student: Like I said will be a min term like all four terms will appear in that.

(Refer Slide Time: 23:32)



All the variables will appear in the free min term either complimented or un-complimented, but they all appear that is the min term definition, this is a min term this is not a min term. So, now, can we get back what determines the number of rows in the PLA.

Student: The number of min terms in the expression total number of.

Not number of in this expression has how many min terms.

Student: One; this is the product, no, no, one, sorry, sorry.

This is a product term, this is in general, this is a product term that actually corresponds to 4.

Student: 4.

Min terms the sum of 4.

Student: Sum of 4.

Min terms right; what we want to minimize here? If I want to minimize the number of rows is the number of distinct product terms, I am using in composing these expressions distinct because I may actually reuse some product term, I use it here, there is a this is a product term that is available for me for use in any of these output lines right. So, this product term has been used once for that output also once for this output.

So, I do not count that twice because that line is available to me, it does not constitute additional area in this kind of technology, the number of connections are irrelevant, if you look at the layout this may be done, I mean, this is actually part of a pre designed structure establishing a connection or not establishing a connection does not add or subtract area the way, these are designed you would turn some metal on or off either you connect some line to voltage one or like an input gate to a transistor either you say you connect it to 0 or a 1, but the transistor is already there. So, you are not saving that those are the properties of structures regular structures like these ok.

So, if I were to now minimize the area of this PLA, one thing is I should try to reduce the number of distinct product terms, I use that reduces the number of rows.

Student: Rows.

What can I do about the number of columns, clearly, I have some flexibility in choosing this because depending on how my logic looks like I may have more product terms here in terms of the number of columns what flexibility do I have.

Student: As we discussed earlier, the for the state register if we use the minimum logic and solve that.

Yes I have some flexibility here, if I use more bits, then I have more columns for the state register. So, if I just want to minimize area perhaps I can just use the minimum number of bits.

Student: It is because (Refer Time: 26:47) kind of problem because if I use exploratory state of this term then my number of product terms can reduce if I use less then my distinct product terms.

Can increase so that is where the optimization angle comes in whether you optimise number of columns more or you optimize number of rows more that is very balanced.

Yes, we should understand the tradeoffs, here some of the choices interlinked, for example, the number of state bits by itself, it does not say much, but it depends on how you are actually doing the encoding, it is possible that for some choices of length of the state register the logic is simpler.

You actually might need fewer product terms, then, since we want to optimize the product of the columns with rows, there is some such uncertainty, but other than that; what are the column choices that we have these output columns, I do not really have a choice because they are going outside those need to be there what about the inputs.

Student: Inputs are also fixed.

Inputs are fixed. Now, there is this question of does the inverted input also need to be there or not. So, some choice you can say, it is there. So, that depends on the methodology a little bit you could optimize. So, that you could give preferences to those kind of expressions where either the variable or its complement is there, but both are not there.

So, there is some such flexibility; that is available, but in general when we talk about a PLA structure we put both the x and x bar line. So, there is really not much flexibility overall not much flexibility is there in the choice of the number of columns other than the state register length, but this is the bigger dimension on which there is some flexibility and we can make an encoding choice that optimizes the number of rows in the PLA what if it was a multi level logic implementation and I want to reduce area.

Student: Sir, how are we gonna implement multi level or it simply gates structures.

Yeah, our implementation is just this let us assume that we can just pick up discrete gates and compose a circuit that looks like this. So, area optimization here translates to somehow minimizing the total area that is occupied by all the gates, the some of the areas at this stage, how do you do that that should also translate to something at the high level like we argued in the other case; that I want to reduce the number of product terms that is taking us a little higher level.

So, that we can argue algebraically, as we take decisions on the encoding here, what kind of argument it would be if I say I just want to reduce total area that is total area of the gates can we translate that into something of an algebraic nature?

Student: Sir, standard logic minimization techniques can be applied to.

Standard logic optimisation would indeed be applied after we perform the encoding. So, at this stage, somehow we need to anticipate what happens there, remember, we are not generating the circuit as part of encoding, we are not generating the circuit, we are only taking those decisions of the translation of the symbolic correspondence of the symbolic states to our distinct encoding that is all that we are doing, but in the process of doing.

So, we should anticipate what happens later number of gates is an interesting low level metric that one can try and anticipate often the proxy that is used at the at a high level for estimating the number of gates is the number of literals I have in an expression ultimately they say there are some bunch of Boolean expressions that are generated, right and our choice here for encoding has to be based on which Boolean of expression is better so; that means, that I have some expression like this and I have some other expression may be with the same result maybe they are all equivalent logically to each other, but they are not all equivalent with respect to implementation if you were to later translate them.

So, given an expression, we can do something to estimate its complexity that would translate to an area number ultimately, the measure that is often used is called literals literal refers to the presence or absence of a variable in an expression ok. So, if a is present, then I count once if c prime is present, then too I count once why it is worth thinking about if b and c. So, this too is two literals a prime b is also two literals.

Student: Now, b has (Refer Time: 32:16).

No, every time it occurs in the expression a particular variable occurs in an expression, we still in any product term in an expression, we say that it is an additional literal, it is worth it or it is not worth it.

Student: If we call literal count.

Yeah. So, here literally literal count is 6 in this expression, why do we do that or first of all, is it good to do that or we should be counting only the number of literals number of literals, anyway, tells us what it says something about the inputs that I have to the expression, we want to capture the complexity of the circuit that will ultimately be generated. So, a literal count is what we want to do here, but is it to count b twice, the way it appears in the expression or should b appear, once as a measure as an approximate measure of the of ultimately what should translates to gate count right, so, if I have this expression here b c plus a prime b.

So, point is b occurs twice should I count it twice; that is what this counting is doing lets be convinced that it is to do. So, maybe for now, let us just ignore the prime here that is a different argument, why should we consider the variable and it is complemented form as one literal each is a separate argument that we will come to, but for now, let us just be convinced that counting be twice is in that kind of an expression.

Student: Because we need two different gates.

It translates to an additional gate, it does not matter that it has occurred earlier, this is one AND gate the other one is implemented using a different AND gate, maybe there is an OR structure there. So, we may occur any number of times, but the argument here essentially is that every time it occurs, it has an impact on the gate count of the circuit that is the intuition; that is the reason for a separate counting of every occurrence of a literal if you had another one if you had plus.

Student: a, b, c.

a, b, c here that you can separately optimize, it is a different matter, we have are not going this, let us say, this is my ultimate expression and I want to translate that in a straightforward way, later on into gates, then that a, b, c indeed translates to a separate gate structure. So, we are saying that every time such a literal occurs then that translates in some ways.

To an additional two input gate or whatever our minimum element is therefore, we count all of them all those occurrences.

Student: Sir, whether it is if we just consider b c plus a b that is b into c plus a so.

Not we are considering the optimisation though we are considering that this is the final expression that we have and I want to take that and build a netlist out of it that there is a better representation of that expression that there is a more compact representation of that expression, if it is there then you replace it and that is what we want to count. So, somehow this is the last stage of all the optimizations and we want to evaluate how good it is without generating a circuit out of it.

Student: Sir, can we put at this way for example, if I take two input 9, then for each literal in that expression, I will be consuming one stack of p and n, if it is a complementary consequence.

Yeah, yeah.

Student: That is what it leads to.

That is what this translates to, it translates to more area and the equivalence here is it is like a two input gate or something like that.

Student: Basically, in that way, you will be able to cover the logic implementation details also.

Yeah.

Student: In one short.

That is what we are trying to estimate, every time something like this occurs here it translates to 2 transistors in the logic implementation, indeed yeah; that is one, the other thing is our approximation that I have a c prime plus b prime c or some such thing this is still counted as 4 literals are we justified in doing. So,

Student: Yes sir.

Why they have got these two inversions.

Student: An inverter has to be realised to inverter signals

Right, I should count that or I should not count.

Student: We should count.

Right, so, I have 4.

Student: Count.

So, I am taking this as equivalent to a c plus b c, if literal count is the only thing that we use as an approximation of the area, then both of these are equivalent, but you know that they are not equivalent because the two additional inverters are there, but still remember this is a high level approximation would we be justified would there be an argument for actually ignoring.

Student: Sir, (Refer Time: 37:34) we can assume that complimentary (Refer Time: 37:36).

Yeah, it becomes a little tricky to capture these effects, exactly, this is an approximation after all the point is; you may have something like this a prime b prime. Now compare this with a b which is the more expensive?

Student: (Refer Time: 37:53).

This because this translates to an or gate right whereas, this is actually a more expensive circuit the and the realization consists of a NAND gate followed by an inverter, right that is why we get rid of this distinction on an average, we ignore the compliment we just say existence.

Student: Overall global picture things will normalise.

Right, the right; so, these are some approximations which are necessary at the high level you need to make that at high level it is hard to and without realizing it right it is hard to come up with a more exact picture. So, we just ignore those I think; so, although if it is very simple, then you should be able to do something about it being more accurate with respect to that estimate.

So, there are some limitations, but still overall the literal count is considered as a very nice proxy for the complexity of a an expression when it comes to multi level logic implementation in the two level logic for that PLA structure would literally count be good.

Student: Sir in a PLA structure indoor can be implemented very efficiently and we have all the inputs available for us.

Right.

Student: So, literal problem cannot be.

The equivalent argument here would be let us say you have some logic here that is represented by this picture and consider the addition of an extra connection that translates to one more literal.

Student: Yes.

Right in the Boolean expression does that translate to more area.

Student: No, no.

It does not, we just argued that it is the area is independent of the number of connections in the multi level structure; the area is dependent on the number on the size of that expression. So, here we are dependent the area is dependent on the number of product terms that are appearing. So, it is not as though there is no dependence at all on the complexity of the expression, but it depends in a different way not merely because the size of the literal the literal count is more or less. So, in this we do not use literal count, we would use product term.

Student: For a given stack of inputs, we can have all the product terms and mid terms.

Right, right.

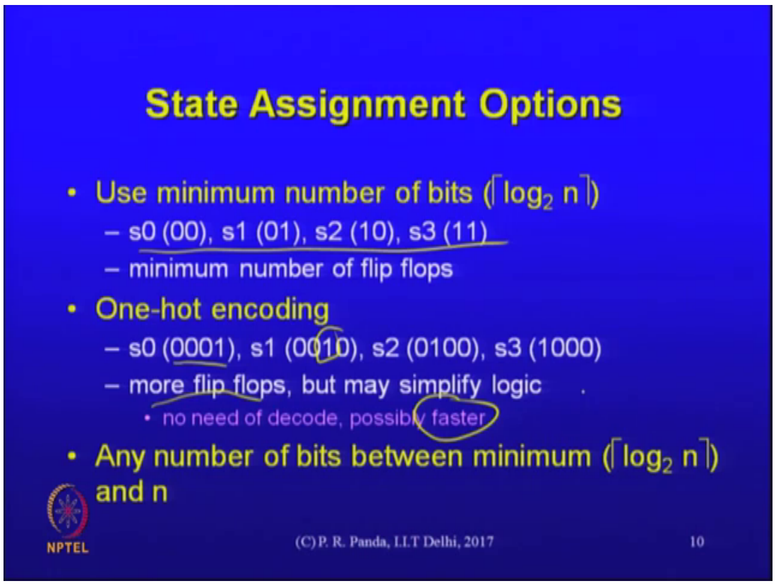
Student: (Refer Time: 40:14) also that.

Yeah.

Student: Will mention the difference if we increase an input then it will (Refer Time: 40:22).

If you have an additional input, then it makes a difference because you have a more column yeah ok. So, this is what we would try to optimize if the target is a PLA implementation, then I try to optimize essentially the number of rows which is number of product terms that is a different optimization if it is a multi level logic implementation, then I try to optimize the gate count and that gate count at the Boolean expression level; what we try to optimize is the number of literals in the expression that will be later built up, remember, we are not going to build up these expressions that is actually part of the logic minimization step which comes later, but we try to anticipate what happens in that step as we argue about the state encoding let us see what options do I have for state assignment.

(Refer Slide Time: 41:10)



State Assignment Options

- Use minimum number of bits ($\lceil \log_2 n \rceil$)
  - s0 (00), s1 (01), s2 (10), s3 (11)
  - minimum number of flip flops
- One-hot encoding
  - s0 (0001), s1 (0010), s2 (0100), s3 (1000)
  - more flip flops, but may simplify logic
    - no need of decode, possibly faster
- Any number of bits between minimum ($\lceil \log_2 n \rceil$) and n

(C) P. R. Panda, I.I.T Delhi, 2017 10

I can use the minimum number of bits if n is the number of states, then I take log n to the base 2 ceiling of that tells us the number of bits that are there in the state register. So, this kind of a choice can be made for the encoding, but of course, I have a large number of choices there even if I use the minimum number of bits. So, that minimizes the number of flip flops. So, that is some step in the direction of optimizing area, but that is only one the other extreme you might be familiar with is what is called a one hot encoding I use as many bits as a number of states.

Student: States.

Right; So, that would mean that if I have 4 states, then I use four bits and the encoding could be such that the corresponding bit is one and all the others are 0 or could also be the other way around which is all the other bits are one and for that state it is 0. So, this leads to more flip flops, it for this example, it is 4 flip flops instead of 2, but it actually might simplify the logic we just talked about it in the p la context buzz, but it may simplify logic why would it simplify logic.

Student: Sir, the next state derivations will simplify your output derivations may also simplify depending on the.

Student: Expressions in the.

Conceptually; so, that logic has two components the logic for determining the next state or the logic for determining the output has two components; one is which state are you in and the other is what are the inputs because those are my inputs there which state you are in that part of the logic is the simpler implementation, if you have one hot encoding why because you just pick up the appropriate.

Student: Bit.

Bit and that bit is one means that you are in that state, if it is 0, it means that you are not in that state. So, if you are looking for a particular transition from s 2 to s 3 and you want to know are you in s 2 that logic is very simple in one hot because you just pick up the appropriate Q output of the flip flop and that being 1 or 0 tells you whether you are in that state, if it were the minimum bit encoding, then you would need a little bit of logic to tell you are you in this state or not because it would be encoded that as 1 0 0 or something which means that you need two inverters and at three input and gate to tell you that you are in that state.

Student: If you encode you have to decode.

Yes, there is a decode that is implied, there the logic may not be as explicit as this because of course, you are sharing logic all the time, but conceptually this is the argument that there is an explicit decoding that has already happened as part of the one hot encoding step. So, logic could be simplified which means that the area could be lower the for the logic part the area of the flip flop has increased, but the logic could possibly be simplified, but more important is this could possibly be a faster design because the critical path is countered from flip flop to flip flop write out Q output of the flip flop to the d input of the next flip flop stage.

If the logic is simpler, then it is likely to be faster. So, there is some merit to one hot encoding also and in general it is not clear which one you should use there, it may depend on w hat, your objective for optimization is although the other thing to note here is that if you take it to the extreme even one hot leads to complicated circuits what if you had 500.

It is not an uncommon design where you have a large number of states.

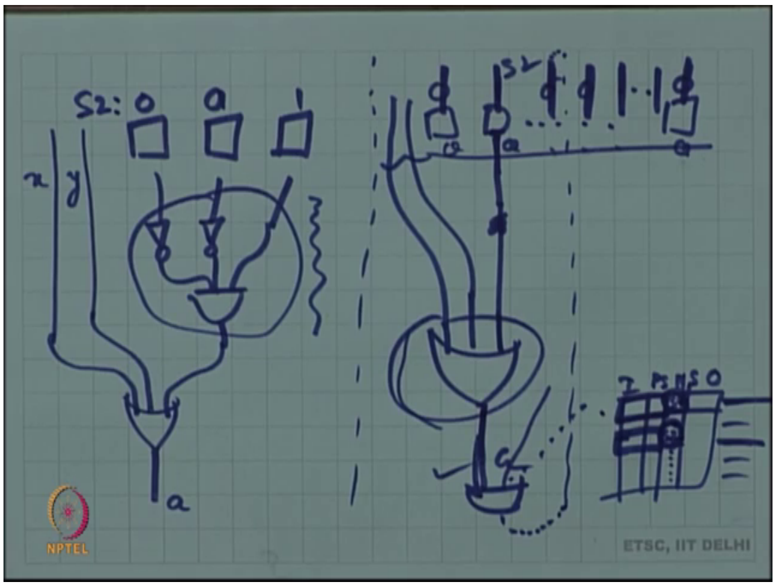
Student: 500 is a very large number.

Right, so, so, then we are talking about nine flip flops versus 500 flip flops somewhere this argument ought to break down in terms of the implications later on with respect to layout output.

Student: Can you explain why this will be faster.

Why is this likely to be faster the reason is the same because the logic may be simpler.

(Refer Slide Time: 45:51)



So, in one case, we have three flip flop design. So, if whether a particular whether you are in a particular state or not, you need to determine by let us say 0 say that is the encoding that we are looking for. So, the circuit is this, this tells you are you in state S 2 whose encoding is 0 0 1, this is not the only thing. In fact, there are maybe a couple of other inputs these are the primary inputs x and y and that transition could be that maybe you have an OR gate, if you are in that state and either x is 1 or y is 1, then the output is something; let us say that is my logic if I use a minimum with encoding of three bits maybe my design had seven states.

So, instead of 3 flip flops, maybe I could have used 7 flip flops and S 2 corresponds to this the seventh flip flops if this is the Q output if this Q output is 1, then I minus 2 in the 1 hot encoded version that line tells me that I am in state S 2 x and y are still there and my logic could be this just getting this output tells me that I am in state S 2 that is nice this whole of this logic was avoided that is the decoding logic of the state that was avoided I directly got this and except the rest of it still remains the same.

So, in the critical path of the design, we omitted a part of the logic that is the simplification we are talking about it could lead to lesser area, but that is a little trickier argument because there is a counteracting increase in the area for the flip flops, but the speed argument is usually there because that logic is now missing here and this has a smaller critical the path.

Student: Sir, next state encoding had complicated with one hot scheme.

The next state; so, what is the next state logic anyway next state logic is also if you look at the. So, that is my input present state next state and output right. So, just like an output logic look like this the next state logic is. So, this truth table is just a general truth table 0s and 1s are there everywhere, this could well have been one of those next state lines same logic in order to find that the next state should be S 2, what you need to do you need to check whether current state is S 1 and the inputs are such and such and that decoding is omitted in that logic also because;

Student: But actually we need to send 0s to the other.

Use ok. So, each of those computations is smaller, but now you have more computations because you have seven lines going to the next state instead of 1.

Student: The connections will increase, but not.

Connections will increase the each of them is simpler logic, but now you have more logic perhaps right the speed argument is still there.

Student: Speed can be.

Right; because each of the logic elements is simpler and therefore, the critical path could be smaller, but the area is a trickier that I did not propose this as an area optimization, I only said this is possibly faster area is more complicated argument it is hard to argue yeah.

Student: Sir, why do we need all 7 line (Refer Time: 50:05) because if we know that we are in state 2 end (Refer Time: 50:08) status will be the state two we need only two lines the state two should be set to 0 and 3 should be set to one all others are already 0 when state 2.

This is the implementation of one transition, right.

Student: Fine.

Yeah.

Student: What I am talking about is like we are not saying that when we are calculating the next state we need to send all seven signals.

Seven signals need to be d seven d inputs need to be given to the

Seven flip flops right they would correspond to all those transitions where that particular state is the next state.

Student: Ok sir, but the rest of the flip flops they are just sitting there only previous.

Rest of the flip flops do not change it is true.

Student: So, given in the logic realisation we need only two lines problem re state for calculating the next state we need to set one to 0 and the other state to the next state.

See the this seven these are the Q outputs of the flip flops the input d values do need to be provided right how do I provide it I pick up from these rows all the rows where let us say S 3 is the next state and build some logic like this let us say there is a nor gate or and gate and. So, some combinational logic any one of them is one then my S 3 should be one.

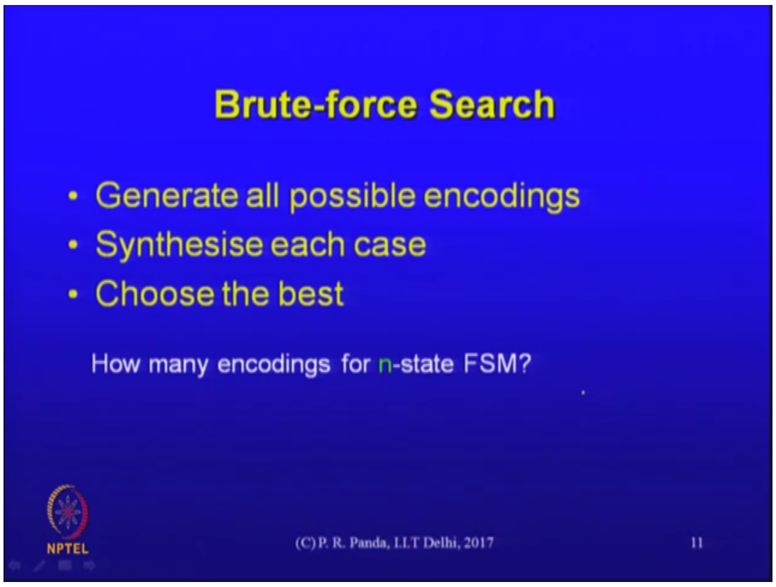
So, that comes from perhaps an OR gate that is picked from many different parts. So, every line that I have in this state table corresponds to some logic that is built and all of these would be. So, if I have S 3 is there, then I would pick up all of that. So, that logic has picked up this logic is picked up and so, correspondingly I would build something and I would send that to the respective input of the flip flop.

Now, inputs have to be sent to 7 different flip flops as opposed to earlier I had to send inputs to two three different flip flops, but it is true that I need to compose the logic for the same number of transitions in both cases the number of transitions is the same.

Remember that that by itself does not mean anything because you can share logic these are not distinct pieces of logic that you have the point of optimization logic minimization of course, is that you want to generate the minimum area circuit by sharing logic, but these options need to be known to us we do have at least the minimum number.

So, we have a range of choices one is the minimum number of bits other is the one hot encoding that hopefully is the maximum number of bits you do not need more than that many number of bits because some of those may be redundant perhaps for a particular encoding you could choose any number of bits between log n and n; there may be an argument in its favour, but the simpler treatment of course, is these two extremes the log n.

(Refer Slide Time: 53:46)



Brute-force Search

- Generate all possible encodings
- Synthesise each case
- Choose the best

How many encodings for n-state FSM?

NPTEL

(C) P. R. Panda, I.I.T Delhi, 2017

11

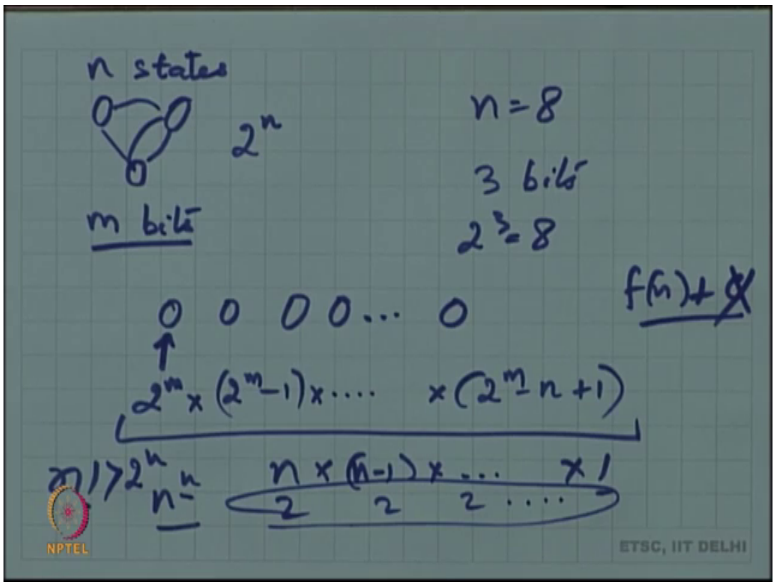
And the and the one hot encoded lets first understand even before solving this what are the possibilities I have a brute force search could be that you generate al the encodings synthesized circuits from each of them and just choose the minimum area circuit, it is not a good one, how many encodings are there for an n state FSM can be precisely captured this number this binary encodings cannot be that hard.

Student: 2 raised to the power n.

2 raised to the power n; can we be more precise than that why two raised to power n.

Student: Yeah should be 2 raised to power n.

(Refer Slide Time: 54:33)



If I have some such logic S 1, S 2, S 3, how many encodings would be there, it is it is 2 raised to power n, if is there a dependence on the number of flip flops, we choose number of bits I choose.

Student: Yeah 2 to the power n, if we choose number of bits as n; then it is like if it is.

If you choose number of bits as n, but n is my number of states.

Student: Yes, one hot encoding is tow raised to power n, but like that it is fixed.

If you choose one hot encoding.

Student: Yeah then.

Then.

Student: Then n n states n states no any state can have one I can choose.

One hot encoding; does just one encoding.

Student: no, but I can have S 1 as 1 0 0 or 0 1 0 or 0 0 1. So, I still have an option. So, there is still an encoding option there.

Yeah, you can or you can have that although when it translates to logic they are all same.

Student: Probably equivalent to yeah.

Should be in that sense they are, but I have n states how many choices are there it is not independent of the number of bits you are choosing there ought to be a dependence on the number of bits that you have chosen suppose I choose m bits where I know what that m is m is in what range.

Student: n and log n.

Between n and log n, those are my choices. So, m bits I choose some number in that range now how many possibilities.

Student: 2 to the power m m.

2 to the power m are we sure. So, if n was eight if I had eight states then I would use three bits suppose 2 to the power 3 is 8 do I have only 8 choices for encoding.

Student: No sir.

No; that would make it easier actually, if the number of choices were only the number of states then it is a linear function and I could actually use the brute force method just a very small number of possibilities. So, it is a instinct should tell you that it should be a large there should be an exponential somewhere, but where would that be.

Student: Sir, this make it as 3 bits, then the eight states can be captured using those three bits that is every state can be assigned any number from those.

Student: Sir, combination or permutation states.

Yeah.

Student: So,

So, if I have n states how many possibilities do I have for the first one.

Student: 2 power n.

2 power n.

Student: no, no, no, m.

2 power m

Student: m, m, m.

Yeah.

Student: yes, sorry, 2 power m, but everyone.

They are independent things, remember, they are we have to make the choice of what m is. So, m is not derivable directly from n that many choices for the first state.

Student: Second also.

Once you have made this, how many choices for the second.

Student: 2 m minus 2 m minus 1 minus 1 that c combination, I think not permutation.

That should be.

Student: Then minus 2 and 1 obtain 1.

Then; what until.

Student: n minus n or minus 1.

It is not exactly, but it is something like that you started with minus 0. So, minus n minus 1, some such expression captures the number of possibilities of course, you can for the minimum value of m, you can substitute it and you can see that it is an exponential function which is 8 times, 7 times, yeah are we convinced that this is an exponential.

Student: That only, yes, but it is an exponential permutation.

Where what is the proof that it is exponential.

Student: Sir, factorial value thing.

It is a factorial function.

Student: 8 factorial.

Right. So,

Student: Combination; not permutation, I was;

So, factorial is factorial exponential in n is n factorial exponential in n.

Student: It is right, no.

Right, I am asking.

Student: (Refer Time: 59:37) factorial (Refer Time: 59:39).

The graph looks like that. So, somehow it is like a cup, informally, these are cup shaped graphs and therefore, one ought to be equivalent to the other is that a precise mathematical argument you are giving it is not so hard to prove that n factorial actually there is a precise, I am not sure there is an approximation for n factorial called Stirling's approximation; where the function looks something like n to the power n there is. So, n does appear in the there are some other things it is not exactly of course, n to the power n, but it is not so hard to prove that n factorial ought to be.

Student: n exponential.

At least exponential you can show that.

Student: Sir, (Refer Time: 60:26) we can break it down n factorial n into n minus one till when there is a if we just ignore the constants, then it is basically there is an mostly bad at times. So, it comes n power n n minus.

What? If you ignore the constant everything is a constant.

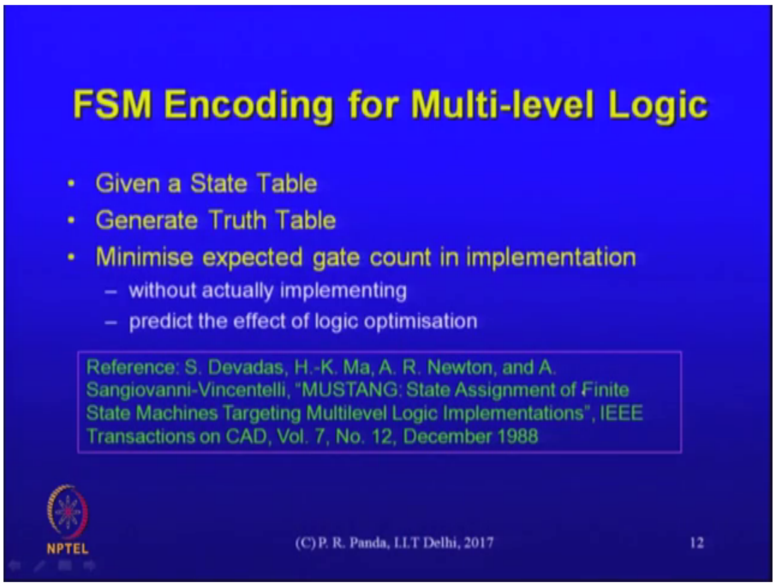
Student: No sir, like we are n times n minus 1 n times n minus 2 n minus (Refer Time: 60:48). So, could we ignore this minus n minus 2 these are.

If you ignore all those then you have totally different, it is a who says that every constant can be ignored in this way if you have some function of n plus a constant, then you can ignore the constant a constant is not always to be ignored in that way, although I appreciate the insights that may come up from it should lead to a clue that it is probably exponential, you can easily show that this ought to be worse, even if you cannot derive that closed form exponential you should be able to prove for example, that this is greater than 2 to the power n, is it obvious from this expansion? Each of these guys has greater than if you really work this. So, there are n terms here.

Student: (Refer Time: 61:48).

Yeah, not hard to say that this should be (Refer Time: 61:52). So, there is no question of generating n factorial encodings and generating circuits out of each of them and then checking which one, it was although if it was linear like if it was n then that might have been a possibility.

(Refer Slide Time: 62:19)



FSM Encoding for Multi-level Logic

- Given a State Table
- Generate Truth Table
- Minimise expected gate count in implementation
  - without actually implementing
  - predict the effect of logic optimisation

Reference: S. Devadas, H.-K. Ma, A. R. Newton, and A. Sangiovanni-Vincentelli, "MUSTANG: State Assignment of Finite State Machines Targeting Multilevel Logic Implementations", IEEE Transactions on CAD, Vol. 7, No. 12, December 1988

NPTEL (C) P. R. Panda, I.I.T Delhi, 2017 12

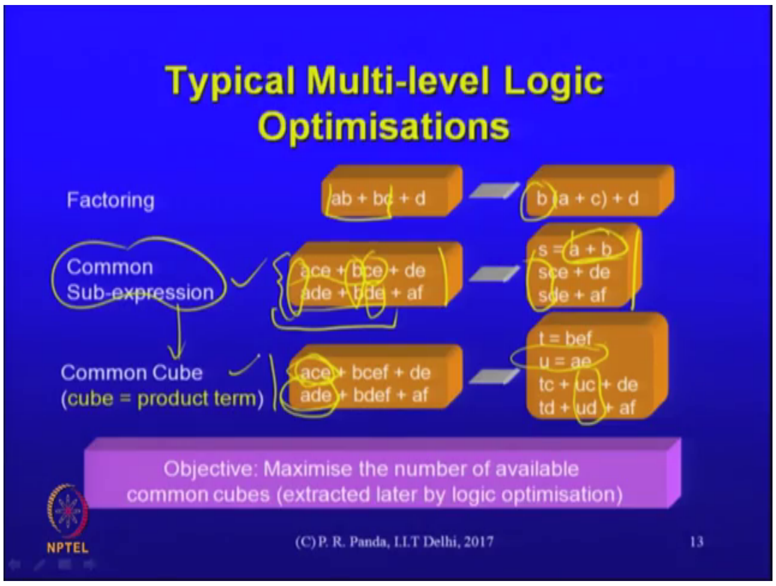
But with that background, let us look at one FSM encoding strategy that targets multi level logic, this discussion is taken straight from that paper it is an old paper, but its selected for analysis just because there are some interesting concepts there that anticipate the effect of later optimizations ok, later means whatever decisions we are going to be taken. Now, are going to be subject to logic optimization that is happening, later, we are not going to do that optimization as part of this process and yet we can prepare the inputs for that optimization in a way that that optimization is able to find common terms and so on that is the objective here.

It is a nice way of estimating these are estimates that we are making, but it is a what we will go through in this argument is a is a concrete example of estimates of essentially literal count. So, what we want to optimize is a literal count in that Boolean expression, but problem is that the Boolean expression is not n there at this time the final Boolean expression is not there.

In spite of that; we would like to anticipate what happens in the later stages which is a little tricky and therefore, a little bit of a more concrete example might help us in quantifying those estimates in some way that is what we will go through in this algorithm.

So, objective of course, is to minimize the expected gate count in the implementation when the implementation is not there you do not want to actually implement it, but you want to argue at a logic level if I did this then this kind of logic is likely to be output and those kind of opportunities the optimization tool is likely to extract and therefore, this is better than that that kind of an argument is made here.

(Refer Slide Time: 64:28)



Typical Multi-level Logic Optimisations

Factoring	ab + bc + d	b(a + c) + d
Common Sub-expression	ace + bce + de ade + bde + af	s = a + b sce + de sde + af

Common Cube
(cube = product term)
	ace + bcef + de
	ade + bdef + af
		t = bef
		u = ae
		tc + uc + de
		td + ud + af

Objective: Maximise the number of available common cubes (extracted later by logic optimisation)

NPTEL

(C) P. R. Panda, I.I.T Delhi, 2017 13

For that; let us first introduce some typical optimizations that are actually happening in the multi level logic optimizer, this will formally explore later on in the last stage of this course how the logic optimization tool works logic synthesis mechanisms, but the principles are of course, known principles there is an idea of factoring if you see something like this might be occurring the same variable occurring in complemented way or in an un complemented way in multiple terms like that then I can extract out a common factor.

Why is this a good idea; what does this optimize?

Student: The number of gates number.

Optimizes number of gates our approximation of the gate count is literal.

Student: Literal number of.

So, it reduces literal, I should be able to argue that it reduces literal and therefore, I would do this does reduce literal where I take a common b out instead of two occurrences of b I have one occurrence of b and therefore, there is a reduction in the little count the rest of it does not change, but b bs occurrences reduce and therefore, the total literal count reduces there of course, is a common sub expression argument here also of what nature if I look at these two expressions jointly, I see that a plus b is a common expression or it is a common sub expression of each of them is expression because here that logic can be represented as a plus b times something.

Student: c e.

Times c e and this logic can be expressed as a plus b times d e and therefore, if I do that a plus b turns out to be a common sub expression doing this would increase or decrease the literal count actually I increase the number of expressions I had two expressions earlier now I have three expressions.

Student: Because number of gates will give you the;

Right, the gates is likely to be reduced just because you see that there is some common computation that was taken out that translates the number of gates and that would reduce you see that literal count instead of a b occurring to a was occurring twice b was occurring twice, at least as far as this part was concerned, we reduced 2 occurrences of a to 1 occurrence of a reduced 2 occurrences of b to 1 occurrence of b S was added here.

So, anyway overall you can see that such an exploitation of common sub expressions would lead to reduced literal count in whatever it is that I am implementing a third one is a common cube occurrence of a common cube is identified as follows cube is sort of a special case of the common sub expression where there is just one product term in that expression; that is what we are calling a cube in some ways dealing with cubes is easier logic optimization logic synthesis algorithms tend to work with cube because it is an easier analysis we look at that analysis, later, but finding a common cube is an easier evaluation than finding a general common sub expression that might be the sum of four different product terms right.

So, common cube what common cubes are there in this expression here I have an a e, here I also have an a e here I can see that there is a common cube between these two that is extracted out and so, u is extracted out and I have the product of u with the c and d this is a little easier because depends of course, on how you represent these things, but to take one expression like that one product term like that and take another product term and find that there is a common cube between these two, you can see is an easier activity than take a general expression and some other expression and find that there is a common sub expression like this one is a harder analysis than this because we are limited to just single product terms not sum of product terms at least the or part of it is all gone and we need to worry only about the ands.

Student: Sir, one that is tangential question.

Yeah.

Student: The data structure point of you doing the common cube approach probably simpler with the matrix implementation versus a graph right because when it is;

Graph we did not talk about the representation but.

Student: But.

So, in an in a graph what is the matrix and what does the matrix have and what is the graph have.

Student: These are;

This is a general Boolean expression.

Student: Right.

Right, so, matrix means.

Student: In matrix, I can keep these equations as a product of rows and columns I am may be I am not able to express.

If actually the representation is totally non trivial what is a good representation is not obvious from here, but what we are just arguing in principle is that limiting ourselves to one product term the possibility of finding a common what is common between those two product terms is of course, a simpler version of the general expression where you have sum of product terms and two sum of product terms and you find what is common between the two, this seems to be a more restricted version of the everything that you want to do here you also want to do in the other one, but you also have sum of products.

So, it is just a conceptual argument. So, far, but we will get to because everything in the logic synthesis has to do with how well you represent these things.

Student: I would only collect from the constant multiplications in that here it is not obvious how you can represent, but in case of filter designs and stuff where multiple constant multiplications are there in parallel.

Yeah.

Student: And then graph based representations are very hard to code in terms of reducing the logic and if you put it into a matrix form like a y equals a x.

In certain specific kinds of computation certainly more interesting data structures could be generated graph does suggest itself as an obvious way to represent just because if you look at the netlist it is like a graph.

Student: Yeah.

Right, yeah, but still you have to do more and there are I will we should spend some time on the interesting representations when we get to the logic level as of now this is only a principle argument, alright.

Student: Sir, do we do both of these (Refer Time: 71:58) sometimes common sub expression and common cube?

If you have a good algorithm for doing common sub expression you are also doing common cube as part of it.

Student: Part of it.

Right

Student: if you were doing.

Yeah.

Student: Recursive in nature.

Yeah, it is just that sometimes in the interest of simplification, but particularly in this algorithm why we introduced these at this stage is that this algorithm tries to maximize the availability of common cubes assuming that the logic optimizer is handling at least common cubes if common sub expressions its fine, but even if not common sub expression it is if it is doing common cube then it will try to expose the availability of common cubes to the logic synthesizer that is these reason why we are talking about it here fine let me stop here will you get to the actual algorithm in the next class.

Student: Sir, why we do this common cube actually if we have common sub expression.

No, if you have common sub expression then you do not do common cube, but you could do common cube and not do common sub expression because that is more difficult, it is still a reasonable logic synthesis tool that would do common cube not necessarily common sub expression the other thing to note is that this can be done well an algorithm that identifies and exploits common cube can perform very well quality of that algorithm can be good common sub expression is a difficult problem common cube is not hard to as long as you have all the expressions every pair you can take and you can easily argue whether a common cube is there or not.

What is the size of the common cube, right in an optimal way that is not very expensive, but the common sub expression is hard to do remember a plus b is same as b plus a that introduces significant complexity into the process of identifying the common sub expression, you will find that as part of your I assure you, you will come across that difficulty and be convinced totally by the end of this semester that common sub expression is hard, but common cube is much easier to do optimally even.