

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

Lecture – 23
Multi-level Logic Optimisation

(Refer Slide Time: 00:18)

Multi-level Logic Optimisation

- **More degrees of freedom**
- **Exact optimisation**
 - computationally complex and impractical
- **Heuristics employed**
 - Phase 1 - optimise using generic gates
 - Phase 2 - technology specific mapping

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

A new topic today; Multi-level Logic Optimization is in some ways, a generalization, more degrees of freedom here than in 2 level because you could do everything that you are doing in 2 level minimization, but there is also the opportunity to have as many levels as you want in the logic.

So, this problem is in general more difficult than the 2 level logic optimization problem, of course, their objectives are also different here for the optimization here two as you would expect an exact optimization is difficult, it is computationally complex. So, some heuristics are implied and overall, one can group this heuristics into 2 different phases, the first phase is an optimization using generic gates and generic gates means that we do not necessarily take into account the detailed physical properties of the gates in terms of area or timing and so on, but just treat these as generic 2 input NAND gates and NOR gates and so on and the optimization is done in terms of those generic gates in the first phase.

In the second phase, there is a technology specific mapping that you take the optimized netlist from the first phase and then make decisions about; how that generic netlist would be realized in terms of cells that are available in a particular technology library.

So, this is where the target architecture; the target library and so on; all of those come into the picture, but the first phase; we just try to perform a number of optimizations just keeping high level objectives like literal count; for example, in mind right. So, we will look at both of these, in that order, the first phase would be generic optimisation in which if it is multi level logic, you may be looking at just literals for exactly.

(Refer Slide Time: 02:50)

Representation: Logic Networks

- Graph
- Nodes
 - Local function
 - multiple inputs, single output ($y = a + bc$)
 - Primary inputs ✓
 - Primary outputs ✓
- Edges
 - Nets
 - single source, single destination
 - if multiple dest, net represented by multiple edges

The diagram shows a network of nodes and edges. A central node is circled in yellow and has three arrows pointing to it from above. Below it, there are several other nodes connected by lines representing edges. Some edges are also circled in yellow.

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

Student: Now the generic gates change, when we give a different library input, when does the tool prepare depending; for example, in my library that I am going to give for technology mapping does not contain certain kind of e o is or some other complex scale.

Right.

Student: Then would the generic gate list change or it remains same.

It is you may pre process it, essentially a replacement process would have to be done for some set of gates in the generic to one or more gates in the actual technology library. So, if even if a particular gate is not available a combination of the gates from; the generic netlist would be available in terms of implementing by a combination of gates in the technology library the generic netlist might still be the same, but we are we are.

Student: Generic is independent or;

Yeah, it is independent of what is coming later on, you here too you could argue that ideally, you should do them together, but it is just the complexity of the engineering that leads us to divide it into 2 phases.

Student: Actually what I meant was read the library find the functions that are supported in the library use them as generic.

Yeah, but it is very unlikely that there are functions in the generic library that are not implementable in the real library; of course, the whole point of designing a library is that you want it be suitable for all kinds of purposes. So, what are the chances that you will not NAND gate right once you have a NAND gate everything is realizable in terms of that NAND gate? So, the issue of course, is that there may be many different ways of realizing the same logical function and therefore, we need to know all the details in terms of 5 different drive strengths that may be available.

So, maybe 5 different gates do the same thing, but the properties are different in terms of area and delay tradeoffs right, but those numbers what is the area what is the delay of a particular gate we do not take into account in the initial phase when we are just dealing with generic gates and a generate netlist, you start with the representation these are logic networks that are composed here. So, graph returns as a natural choice we had actually dropped the graph structure for the 2 level just because it was a more specific structure some tabular representations were used there, but now that you have any number of levels the graph again is suitable.

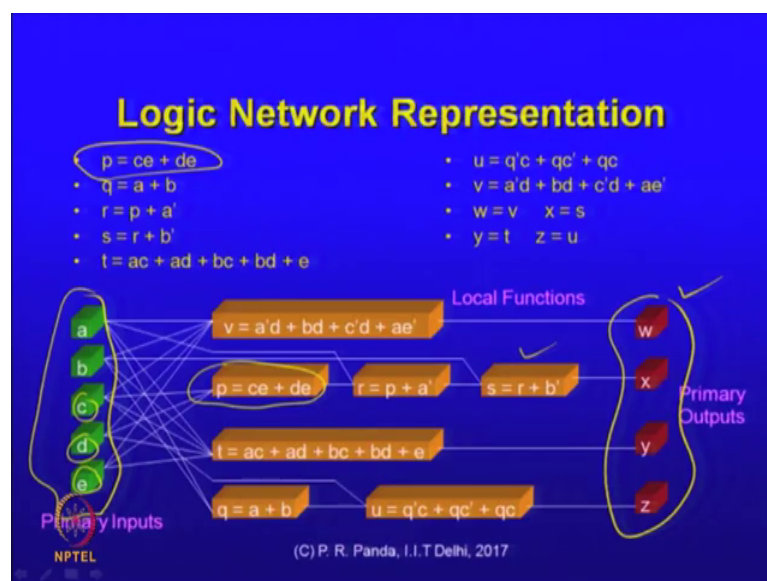
What do the nodes represent they could represent some local functions where there could be multiple inputs and a single output usually. So, it could be a function like this. So, that is a node into which there are 3 incoming edges each of a b and c may be a primary input or it could also be the output of some other node some other function and the output here would be a single edge if that y is connected to many different nodes then you would have different edges from that node to all the other nodes all of them having the value of y. So, that is what in general; this is it is a local function it could also represent a primary input or a primary output we will look at an example.

So, what are the edges adjusts represent nets in the design wires in the design with the assumption that this is a single source single destination which is a sort of a breaking of the abstraction of the meaning of a wire in general you could have multiple sources if you design it properly, but nevertheless, a single wire is a single structure which is broken up in the representation into multiple edges all with the same source, but multiple destinations like that right all of this is the same physical wire, but the way to use a normal graph representation and through that perform all the traversals and optimizations and. So, on is to make this assumption break up that wire into multiple edges.

Of course, you know that this is not the only way to compose a graph you could have hyper edges in a graph that could actually connect multiple nodes that could be the more correct way of representing wires in general for most of these optimizations, it may not make too much difference, but actually when it comes to a later phase of physical synthesis that is where that representation is more important, this particular approximation usually does not hurt the optimizations at this level of abstraction.

But it does make a difference when it comes to routing for example, there too this assumption is made, but that assumption may interfere in some ways with the quality of the result, but here it is an easier assumption, we just say that if the same wire is connecting one source and multiple destinations then there are different edges from that node to all the other nodes.

(Refer Slide Time: 09:13)

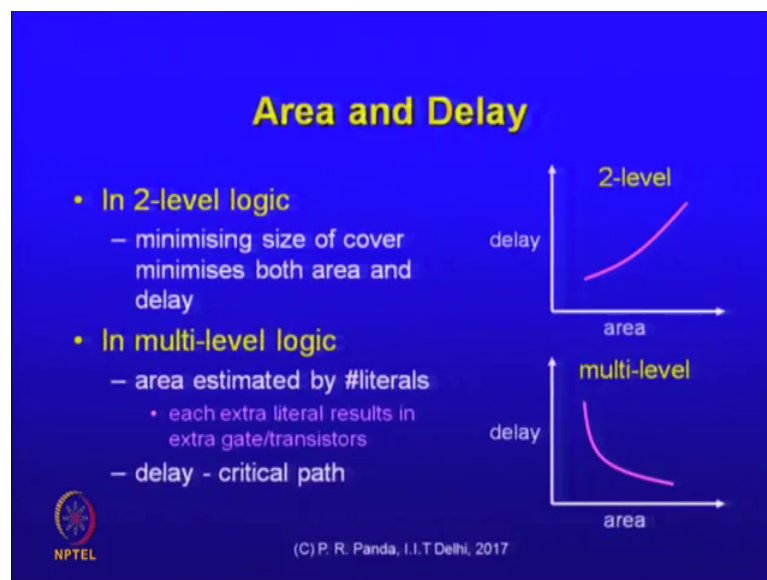


Let us look at an example here. This is a graph that represents a logic network, what you have here are the primary inputs, these are nodes in this graph, these are the primary outputs, those are also nodes the whole thing could be a just a combinational function and I have some nodes that are primary inputs and primary outputs, but in general a node represents a local function like this.

So, I have $p = c \oplus d \oplus e$ in one of the stages and that is one node, here it has inputs c, d and e . So, these are the edges that are connecting c, d and e to that node and there is one output. So, each node is performing one Boolean function. So, that out that is the output that other node, there performs the function $r = p \oplus p'$ comes from the output of that node there and a prime comes from that node original node itself that is the general structure.

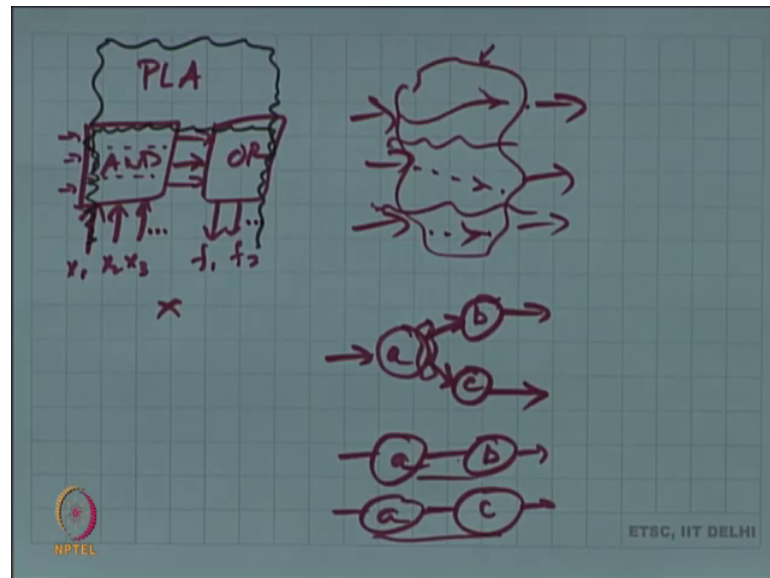
So, there is no limit to how complex that local function could be, it could be a large function, but the representation of that could be a 2 level function in the general, if it were multi level, then you would have the different levels being abstracted out into different nodes in the structure ok.

(Refer Slide Time: 11:06)



For such kind of representation, let us see; what is the relationship between area and delay in 2 level logic, minimizing the size of the cover which is what our objective was remember it minimizes area of course, there is a straightforward relationship between the number of Implicants, we are including in the cover and the size of the circuit.

(Refer Slide Time: 11:46)



What about delay? Remember, this was a PLA structure with an AND plane and an OR plane, I had my inputs and outputs, this was the general structure the optimization objective was to reduce the number of rows, I had in the PLA with the implication that the each row counts as some additional area over it, remember, the number of connections do not count towards area, all right; that is just the way we have designed this design the fabric itself what about delay.

Student: Number of connections number of connections thus;

Number of connections has an impact, but it is a sort of a secondary impact does this number of Implicants which determines area directly also determined delay.

Student: In PLA no.

No, not at all; or;

Student: Sir, it will realise the more number of Implicants, there will be more number of area each single unit which will have its own delay area; it is of;

What you means each single unit, we have a bunch of functions f_1, f_2 and these are my inputs x_1, x_2, x_3 and. So, on there is a bunch of inputs and these are the bunch of function. So, the let us assume that we are realizing one PLA; what about the number of functions we still have one PLA structure. So, I want to know; what is the critical path

through this structure, all of this is still a combinational logic, right, there is no sequential element there, if you have a sequential element, it is outside like we had in the case of the state machine, where is the critical path through the structure? It is hard to answer that without showing where the connections are because the connections are making a difference, but assume that the connections are uniformly distributed or they are there everywhere.

So, for now, if you remove the effect of the connections the question is does the number of Implicants have an effect or it has no effect on the critical path through the PLA.

Student: (Refer Time: 14:07) because the all the all of them are computed in parallel and all the (Refer Time: 14:13) is also computed in.

It can we go beyond the gates and look in terms of wires they also contributes to delay, right.

Student: Yes.

Yeah.

Student: Sir, more than number of rows we have delay will delay.

Why.

Student: The vertical delay distance.

The vertical distance increases the part is likely to be something like this and this that might be the longest path through the design, what happens in the gates is an additional effect that does need to be taken into account, but you can see that if you arbitrarily increase the number of Implicants, then this part gets longer and longer n right at some point, it would begin to dominate our delay through the structure.

Student: Just the wire delay.

Just the wire delay; yes.

Student: There are no like capacitance involved in which take the most part of the delay calculations.

Wire; how is the wire delay computed?

The load on the gates; so, there must be a gate in the AND plane, its delay will depend on the wire.

Right, right.

Load. So, that is what?

Yeah, you could refine each of these this critical path, dependence is not an exact the formulation as you can see, it is hard to determine the critical path, if I do not tell you anything about the connections, but if I assume that the connections are what they are then and they follow some kind of uniform distribution, then this certainly is one orthogonal effect, but it simplifies one of the things which is this is not a bad optimization objective even for delay, we argued in terms of area primarily as we develop the theory, but in fact, you could also use it as an objective for delay minimization.

So, in general, you may expect this kind of a curve that represents an area delay trade-off in 2 level logic because a smaller area could actually mean smaller delay and the larger area could mean larger delay multi level logic that is certainly not the case, area could still be estimated by the number of literals using the argument that we have seen earlier essentially each extra literal results in extra gate or transistors like it is like if you had instead of 2 elements, if you had 3 elements in a cube, then the NAND gate corresponding to the realization of that product would have 2 more transistors one n transistor and one p transistor.

So, in terms of multi level logic if I am going to count the number of transistors, then the literal count is actually a reasonable estimate delay though is a completely different argument how do you find delay in a multi level logic network delay corresponds to what it corresponds to a critical path of course, I should be computing the critical path what dependence does that have on the number of literals in the circuit.

Student: Sir, one question on the previous.

Yeah.

Student: so, when we go for break you have to when we selected the particular library do we also consider things like a four input and gate versus 2 to 2 input AND gates or;

Yeah, but that is the topic of the next phase. So, why do not we wait until that idea is I should be able to evaluate all the possibilities I have and choose the best one.

Student: Sir, just quick question; just;

Yeah.

Student: The generic or NAND is n input NAND in the synthesis, it is not only a 2 input NAND or a 3.

It could, no, you could have assumed it to be an 2 input NAND, it is it is all right, but again.

Student: What is the norm? I just I was;

Norm is the 2 input.

Student: Norm is 2 input.

Right, but then I know that if I have a 3 input that is available, then I can realize it in terms of something the critical path computation if you have some logic here some bunch of inputs and some bunch of outputs this is a multi level structure not this structure, but it is a multi level structure is there a dependence between the number of literals which in some ways is a measure of the area complexity of this design versus the critical path in general, actually, it is hard to say anything about it you could have structures like these, right.

These are disconnected structures, there are paths from here to there, there are paths from here to there, there are paths from next to previous, you are essentially computing the longest path from any input to any output that in a graph structure, in general has nothing to do with the number of nodes.

Student: Yeah.

Of the graph right. So, you may expect an area delay trade-off that looks like this, there might be a trade off that you spend some more area and generate a circuit in which the delay is actually smaller.

Student: Yes.

We have seen examples of this; what are examples of circuits? You have already seen that go too far. In fact, adder designs that you have seen, they are a perfect example of this the simplest adder in terms of area is which one?

Student: Sir, this a is an ripple carrier.

The ripple carry structure is a very small and compact design, but that is not the best in terms of timing, there are other structures that are actually more efficient in terms of timing, but the trade off is that the area is larger, right, if you compare a carry look ahead adder, for example, it is larger. So, there is this kind of a trade off that usually multi level logic would exhibit, you can see that in general. Suppose, I have this kind of a function where this node, the node a is reused which is a good idea in general from an area point of view because that is output from here is leading to some function here and some other function here and certainly, we do not want to re-compute a twice; it is the same function.

So, from an area point of view, this the sharing makes perfect sense, what about alternatively, if I did this there is some redundancy in this structure because you have duplicated the logic, but could this be good from any point of view.

Student: This is faster.

It could be faster, why?

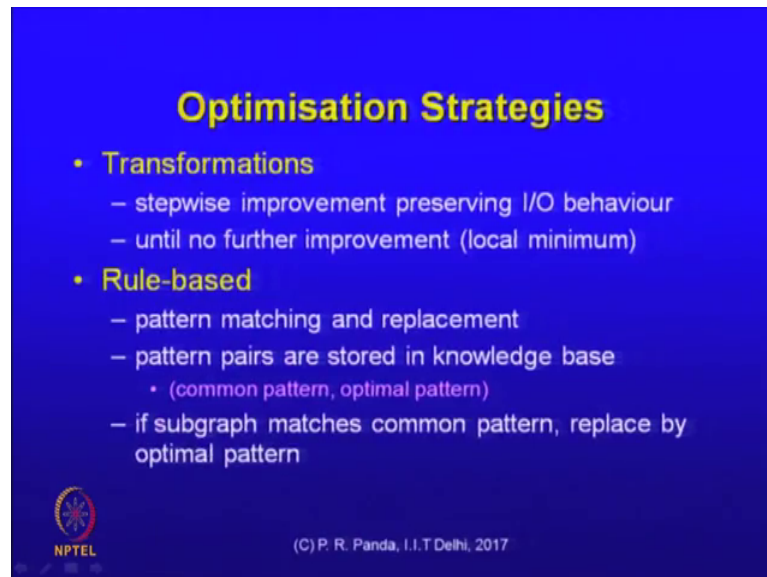
Student: The fan out.

Yes, that fan out here causes larger load to be driven by whatever is the gate that is driving the output of a.

Student: We will get more opportunity to restructure a and b and a and c together.

Yeah, each of them independently could be optimized separately, maybe, you will find some opportunity for further, but even this simple argument shows that some redundancy actually might help us arrive at a faster circuit.

(Refer Slide Time: 22:21)



Optimisation Strategies

- **Transformations**
 - stepwise improvement preserving I/O behaviour
 - until no further improvement (local minimum)
- **Rule-based**
 - pattern matching and replacement
 - pattern pairs are stored in knowledge base
 - (common pattern, optimal pattern)
 - if subgraph matches common pattern, replace by optimal pattern

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

Because optimization strategies for multi level logic effectively are a bunch of transformations just like a set of operations; that we had defined for 2 level logic each of them could be a stepwise improvement that essentially preserves the I O behaviour of the circuit, but internally, we are trying out different possibilities for the node structuring of the nodes and the edges until there is no further improvement in terms of perhaps, the number of literals if it is an area optimization, it were trying, these could be rule based which means that essentially, it is a pattern matching and a replacement.

You have a pattern or you have a bunch of patterns that you already know could be realized in terms of more efficient patterns. You are searching for those patterns and when you find them you are replacing it by the more efficient patterns that is all effectively that is happening there you have a question? Yeah.

Student: Rules are predefined by us or they are;

Yeah, it is; no, it is defined by us, yeah by us, but this can be a lot more complex to start with, we seed those library of patterns a library of pair of patterns with the most elementary rules of Boolean logic, but it could be more and more complex based on our

experience based on ultimately maybe in a later stage on the knowledge of the target library even, but this could be a large number of patterns, but we look at some of them.

This in general could be the search for a graph a sub graph in our larger graph and replacing it by some other graph, but even though that graph isomorphism is what would be the algorithm that would help us identify that and we already said that it is a difficult problem, but these patterns may not be very complicated, it may be that the number of nodes here is actually very small. So, you could afford to have a little more exhaustive kind of pattern matching strategies because the sub graphs are not large.

(Refer Slide Time: 24:51)

Transformations: ELIMINATION

- Remove a node
- Replace its occurrences in the network by its expression
 - maybe simple local expression... optimisation opportunities if combined

$p = ce + de$ $r = p + a'$ $s = r + b'$

$p = ce + de$ $s = p + a' + b'$

NPTEL

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

Let us take a look at a bunch of these transformations again each of them independently is actually very simple the one kind of elimination in which we just remove a node removing a node means replace its occurrences in the network by the expression that corresponds to that function. So, the idea is to remove that node from. So, if I remove that node then that output is being used here. So, I need to replace the r by whatever was the function that corresponds to r as you can see this is a low level analogy of what optimization what transformation that we have already seen.

Student: this is function in lining.

Yes, this is just an in lining equivalent of an in line. This is also a function, these are Boolean functions and the appearance of that literal r here on the right hand side

wherever its going would lead to replacement of that literal by the function that it corresponds to. So, structurally it would lead to dropping of a node, but also adjusting of the input edges into wherever that node is going. So, this node now has additional edges why we do this does it affect the number of cubes.

Student: We are not assuming the nodes of optimist.

Yeah, this is again a local transformation this is a routine that could be called many number of times by itself what does it achieve question of course, is that does it take us towards our optimization goal which is minimizing the number of literals that is what I am would do here, what literals are there in this circuit in initial circuit and the final circuit.

Student: That or is being eliminated.

Or is being eliminated and that leads to a reduction in the number of literals is there an increase because of some other reason because of the elimination.

Student: (Refer Time: 27:19).

Yeah.

Student: But ordinary (Refer Time: 27:25) only a b c d are there.

A b c d are the literals, r is also a literal, this is multi level logic and what does this r? Anyway, this is like a NOR gate and it does perform some Boolean function, the output of that is going into that other node which will also be realized in terms of some gates there r is a because it is an input to that gate.

Student: Can we think of this as well. So, with in function in lining here what we have assumed is now the last node has 3 literals rather than 2 literals.

Yes.

Student: But if I look at the previous picture there, I had 3 gates to realize this no number of literals have reduced in any case. So, in a it is;

This leads to increase or decrease in the number of literals in the general case.

Student: Decrease increase.

Decrease.

Student: Like one node, they are increase and one node I have eliminated.

Right. So, in general case what might happen.

Student: It may be we can say what you are saying but.

No, in the general.

Student: Sir, which must be needed probably is reduced like we had, we stored r 's value somewhere.

This is purely combinational logic. If r has to be stored somewhere that happens in a separate, this is just an intermediate output of a gate of a combinational gate and its going into another combinational gate right.

Student: There are 8.

That is why the question was not what happens here, but what happens in the general.

Student: General sense.

Notice, this is plural means this may have many outgoing edges.

Student: Or it will increase you are saying.

It will increase the general in general, it must increase; what is happening is one occurrence of that function is dropped, but many occurrences are added. So, why is it useful if it is increasing, then it is taking us away from the desired optimal target, but the reason it is worth doing it is that in the process. You have a bigger expression the small, it is likely that the original expression there was too small to afford any further optimization, but if you actually observed it earlier, we actually looked at this in the context of that. Now, just what we discussed if in a bigger expression, there are more opportunities, hopefully for finding optimizations, but in a much smaller one it may be difficult yeah.

Student: Also from multi level logic perspective, nor the let us say this graph is was on critical path right.

Right.

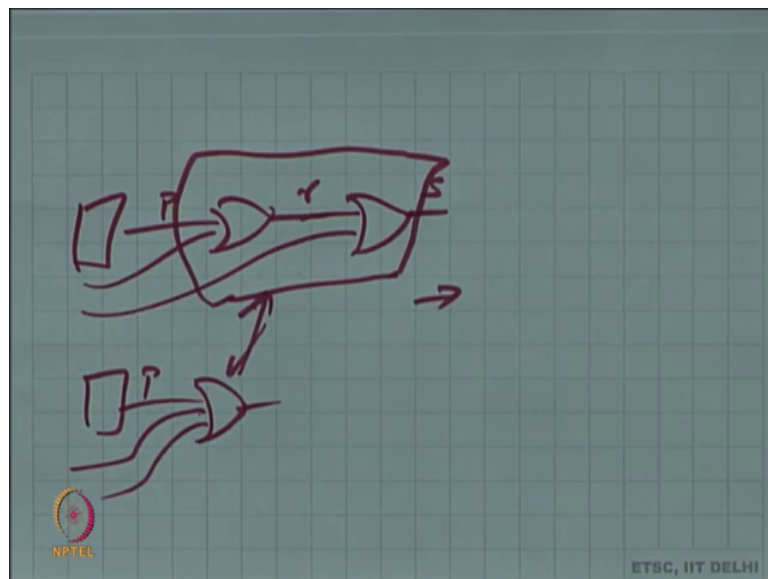
Student: So, if that is the case, then we have reduced one node in our graph. So, if I consider each one has one node, the node is here time being here is the critical path are we looking at timing here.

This reduction of the literals is timing optimization or a area optimization yeah we just argued that actually literal count is not a faithful objective if it is a timing optimization.

Student: moving node I O optimised node.

Here, in this particular case, there is no difference in timing it depends on what the target gates are, but what you have there is there is an output, there is an OR gate.

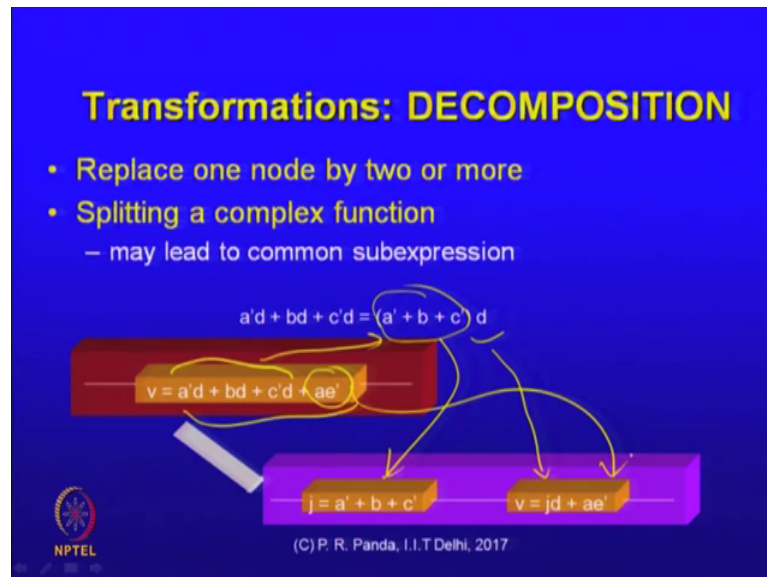
(Refer Slide Time: 31:05)



If you ignore the literal and there is another or gate that corresponds to the third one the first one is $c \oplus e$ plus $d \oplus e$ whatever that is that is not changing and I had p here, I had r here, I had s here, yeah, this could be the ultimate realization in terms of logic, well, in the transformed one, what we are saying is this part is the p part is unchanged and I now have a 3 input or gate perhaps not very different in terms of the timing behaviour either, remember, we are still talking about generic gates here, we are not talking about picking

up elements from a technology library, but it is not hard to anticipate that you submit, this to a technology mapping phase, if there is a 3 input or gate in the library that is better than 2 cascaded 2 input or gates we would definitely discover it and replace it by essentially this.

(Refer Slide Time: 32:19)



This particular example; there is really no difference in timing, yeah, yeah.

Student: (Refer Time: 32:20) which you made like for the literals we do not focus on timing that is not.

Right, literal count is not a good measure of timing critical path.

Student: A critical when we look at a real problem, then we will look at all the states, right.

Yeah.

Student: Delay power area.

Yeah.

Student: So, then these are where does this literal transportation a transportation comes in a whole flow. So, first we start with.

The optimization objectives are different as you know, it would take if it is area and you want to focus only on literals that takes you in one direction, if it is delay, then it may take you in a very different direction, but you have to inform the optimization tool in some way about the mix of area versus both of these are weighted in some ways in terms of priority and similarly, the direction that it takes for optimization would be influenced by those priorities yeah.

Student: Sir, compute the (Refer Time: 33:28) moving.

The tool may take a very minimal input from the user it may be if it is an area versus delay optimisation it may just take; for example, just one fractions is how much priority what percent area is 50; 50 is how I should be doing my exploration or forty sixty or whatever it wants to simplify the interface to the user and therefore, it may do that alternatively it might also actually expose some of the functionalities as individual optimizations that could be turned on or off if you want to go into more detailed of that.

Student: How much unit good are industry tools in this ah?

They are very good.

Student: People; do people even make optimisations after tools makes that or its good enough in certain theoretical?

Do designers interfere manually with the results of optimization they do all the time it is done, but I think after decades of commercial use they are mostly trusted you might still make changes, but it is a reasonably mature technology, we are not able to discuss all of those in course like this in which there are only a couple of hours on a complex optimization like this what we are reading about; obviously, here is the very basic set of steps these would certainly be done whatever we are studying here they would certainly be part of more complex routines.

But the routines have evolved in a direction of essentially knowledge of the patterns from previous designs and populating those tools with more and more intelligence about what are the patterns to look for and what makes sense the generic arguments are not very different, but they are tuned in various ways by specific observations that are kind of messy from a theoretical point of view, but conceptually they are fine we are looking

at pairs of patterns and you could grow that set of pairs to anticipate all kinds of effects and learn from earlier experiences of optimization.

Student: Do not give the opportunity to the end user to add more patterns.

Tool may not give the end user the opportunity; however, the tool itself incorporates a number of such learnings from previous the EDA vendor, of course, is exposed to a large example designs that they get all the time and they try to optimize and designer says that this is my expectation and the tool is not meeting it.

You can see that you can hack it by introducing some pattern and from it is all right. If it is done from the tool vendors side, but it still it is a reasonably sophisticated list of patterns, perhaps, we cannot interfere with it as users, but they are there anyway and those that set is likely growing as and when new things are discovered that the tool is not doing well at.

Student: Is there an opportunity of a learning phase scale min here, I mean instead of going to a cycle?

They evolved by way of heuristics and simple lookup and replacements perhaps by now it would be more formal. So, that learning has always been happening is that we are not calling it learning, but now we can call it learning yeah that is what; now we can call it machine learning, but they refer to intelligence otherwise that designers have had and tools have always had yeah.

Student: human (Refer Time: 37:22) the tool.

Yeah we can call it machine learning, today, it would all be called machine learning.

Student: (Refer Time: 37:30) not saying (Refer Time: 37:31) machine learning phase.

You could make that learning process customized to its particular.

Student: Yeah; that is all I learnt.

Ah use.

Student: Not that otherwise, we have to wait for nine months, they do not give enhancements so easily.

The well the thing is that particular tailoring that you are doing in one context may not work in a different context.

Student: That is ok

That is why it makes sense to expose.

Student: Yes.

It in some ways to a particular design environment for my library it makes sense. So, I will use that. So, there is an argument in favour of that level of customization indeed ok, let us move on to the next transformation. We will try to put these together in one small algorithm later on, but we are just looking at these transformations independently decomposition refers to replacing of a node by 2 or more nodes.

This is if I have a complex expression, I can try to realize it in terms of simple expressions, this is the other way around this also may make sense in certain contexts. So, in splitting a complex function why would you split it, it may lead to the identification later on of common sub expressions between this node and some other node.

I have an expression like this part of this we can see that there is a factorization possibility factorization is good in multi level logic because it reduces the number of literals, of course, so, I could realize this bigger function in terms of this ex[pression]- which is the first node, you have the product of that with D which is this and that part remains unchanged this too could be thought of as an intermediate step, but in general factorization is always good.

(Refer Slide Time: 39:52)

Transformations: EXTRACTION

- Common sub-expression
- Leads to simplification

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

It helps us reduce the number of literals extraction is the name given to this common sub expression elimination transformation, it leads to simplification in terms of the number of literals, if you had something like that this could be realized in terms of those factors a bigger expression like that could be realized in terms of some factors like that then you can identify that these 2 factors are actually the same.

Student: Same.

That is the same sub expression and therefore, I could restructure my graph to have c plus d computation, first introduce a new c plus d computation, but that output could be used to generate p by a product with e and a different function t that also uses scale.

(Refer Slide Time: 40:48)

Transformations: SIMPLIFICATION

- Normal 2-level logic minimisation within an expression

$u = q'c + qc' + qc$

$u = q + c$

NPTEL

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

Of course as part of all of this, I can also do our normal 2 level logic minimization within an expression within one node we can always invoke a 2 level logic minimization tool that would take whatever that expression is and try to come up with a simpler one.

(Refer Slide Time: 41:10)

Transformations: SUBSTITUTION

- Replacing by expression in terms of other nodes
 - creation of new dependency
 - maybe dropping of others

$q = a + b$

$p = ke$

$k = c + d$

$t = ka + kb + e$

$q = a + b$

$p = ke$

$k = c + d$

$t = kq + e$

NPTEL

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

Substitution would be replacing an expression in terms of other nodes what we have here is I have a plus. So, this part of it is the same, but in doing a factorization we realized that the $a + b$ is actually common I could replace that by k times $a + b$.

Student: Substitute.

For example, then I can instead of recreating that function k a, k b or I have not shown all the edges here this one has the k, but it also has a and b as inputs there, but since, q is already computing a plus b I can then realize this in terms of q. So, that would result in the creation of a new dependency because you are expressing one node in terms of the output of a new node, but it could also lead to dropping of some dependencies because you now do not have the a and b feeding into that node you have only k q and e feeding into that node.

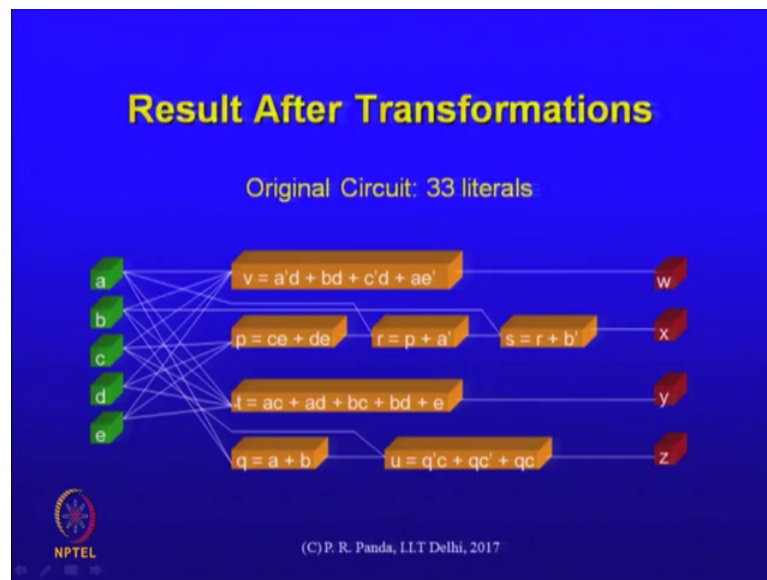
Student: How is this different from CSE.

It is in general not different, but this is a simpler operation than.

Student: Ok, cost wise.

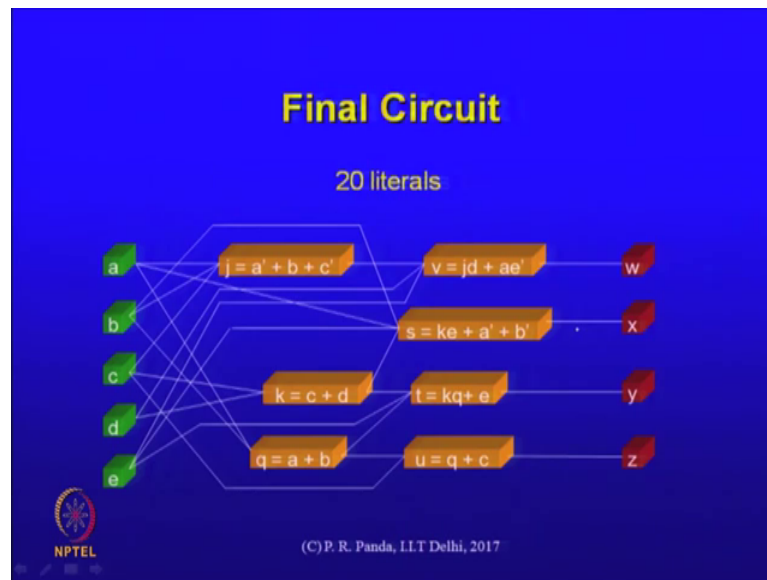
CSE, remember we are finding any sub expression in one node and noticing that that sub expression is there buried in a different node, but here we are just trying to express one node in terms of the output of different node. So, this is an easier optimization to perform.

(Refer Slide Time: 42:57)



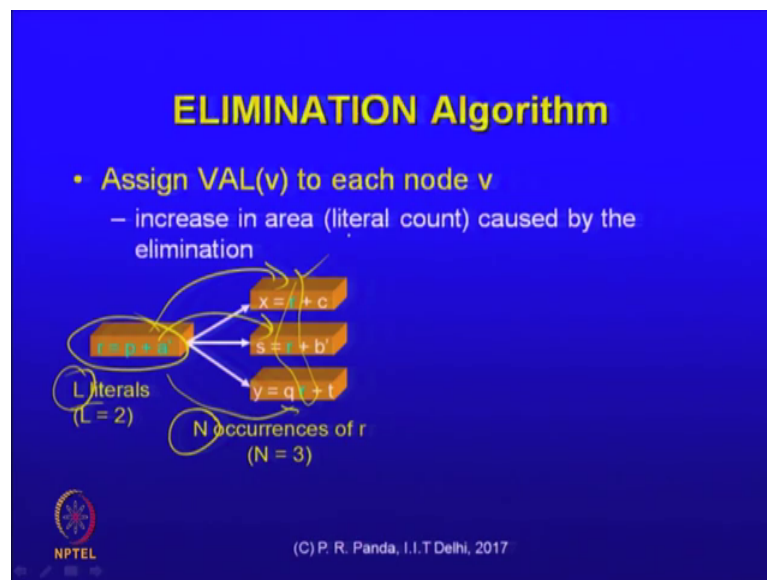
Anyway, if I did all of those in some right order, then if that was my original circuit, I can count the number of literals that are there the number of literals would be the count of all of these literals on the right hand side of the expressions and my final circuit would look like this.

(Refer Slide Time: 43:18)



It all looks very clean and much smaller looks more efficient, but that is because it is a small circuit and we were able to do that.

(Refer Slide Time: 43:30)



But in general our optimization strategy would consist of a bunch of routines that we would call that hopefully take us towards the optimum of course, we can run it until there is no further improvement and that would be the local minimum we are looking for.

So, we will not be able to go into too much detail here, but let us just quickly look at some of the simpler algorithms like an elimination algorithm that was the first one that

we looked at and remember the elimination in the general case leads to an increase in the literal count it is like in lining increasing the number of operations that we are doing to the elimination algorithm. We somehow, have to guide it by saying how much you are allowed to increase the literal count by because this may be very expensive if it turns out that there is one function that is used in a large number of other nodes then we should not be performing that elimination this leads to an increase. So, we would like to control it in some way that algorithm can actually take a number like 10 percent or 5 percent or something like that and say that you can increase, but only by this much not much more than that.

So, if I have one node, let us compute the cost of the elimination in terms of number of literals I have one node with l literals and there are n occurrences of r in different nodes, those are the sides where we could perform this inline, if we eliminate the r from all of these then that would lead to the c plus a prime getting duplicated into all of these, right. So, in general if this expression has l literals and there are n occurrences of r . So, those are the places where we can eliminate the r from can we quickly quantify what is the difference in terms of number of literals for the whole circuit how many more literals are there because of the elimination.

Student: n including minus n .

First of all, I am getting rid of n occurrences of r which means that each of them is of size l right that is what is getting in right I have p plus a prime p plus a prime occurring in a number of places that is the size and the number of this is n times its occurrence.

(Refer Slide Time: 46:47)

$L \times N - N - L$

$A = (X) + (R)C$

$B = (Q)$

$(C + d)$

C

$(A) / (B)$

$Q \times B + R$

NPTEL

ETSC, IIT DELHI

So, that counts as an increase, but there are some decreases that are also happening the r was of course, there earlier and that R is dropped now. So, how many Rs are dropped?

Student: 3 n.

n of them are dropped.

Student: n.

So, that is one reduction anything else is dropped.

Student: that initial block.


Yeah, this block is dropped right remember this block is dropped. So, I, I can drop also from this; that could be my expression for computing the cost of the elimination of one node right I just have to see how many outgoing edges are there and what is the literal count of this not. So, extra literals are these that would have to be compared with how much we are allowed to expand it by which could be an input to the elimination algorithm.

(Refer Slide Time: 47:57)

ELIMINATION Algorithm

```
ELIMINATE (G, k)
repeat {
  vx = select node with VAL (vx) ≤ k
  if (vx = ∅) return;
  Replace x by fx in logic network
}
```

Increase literal count by no more than k per node eliminated

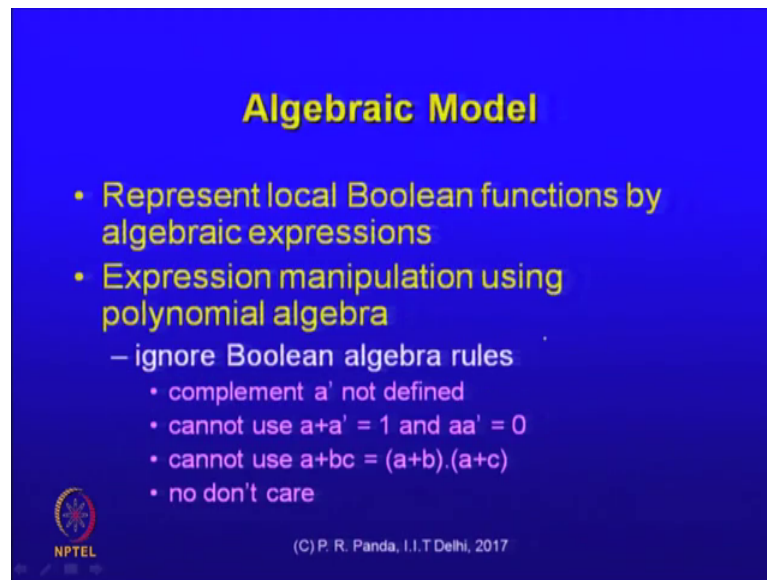


(C) P. R. Panda, I.I.T Delhi, 2017

This is my graph G and k is the increase in literal count that we can tolerate per node that is eliminated, but it could be in general a percentage or something like that. So, you just repeat the following select the node with the VAL which is that cost computation less than or equal to k if that cost is exceeding k then do not eliminate otherwise you do the elimination here this just refers to if it is a node v_3 or something, then if there is nothing further to eliminate then you return, but otherwise you replace that node x by the logic function that corresponds to that node everywhere in the logic network where x and the algorithm is simple after that you just repeat the process until there is nothing further to do.

This is a relatively simplistic computation here that just saying it is less than or equal to k it could be that you define this naught on a per node basis, but for the entire network, then you are about to exceed the total k that you are allowed to expand by then you can stop the algorithm.

(Refer Slide Time: 49:33)



Algebraic Model

- Represent local Boolean functions by algebraic expressions
- Expression manipulation using polynomial algebra
 - ignore Boolean algebra rules
 - complement a' not defined
 - cannot use $a+a' = 1$ and $aa' = 0$
 - cannot use $a+bc = (a+b).(a+c)$
 - no don't care

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

That is a simple strategy, let us look at one representation. So, in the process of this transformations we would have to worry about the representations, we pointed out graphically what those operations were whether its elimination or extraction substitution and so on; how do you perform that in general; there has to be some simplification some kind of a representation that you would have to use it turns out that the general algebraic model is often used as an approximation of the Boolean algebra model which is what we want to perform here this is like a division operation that you want to perform, right.

For example; that substitution where you are trying to see whether one node could be expressed in terms of another you are trying to perform a division operation division is not. So, difficult in general algebra there are of course, tools for us that can perform that efficiently doing it in Boolean algebra is a little trickier and actually what could be used is just a variation of the same algebraic model, we do understand, how to perform division automatically in a generic algebraic model because just reuse it reusing, it means that some of the Boolean algebra rules that additional would have to be ignored for example, all of these things about complement do not make sense in real algebra right.


This is a Boolean algebra specific, but if I use those algorithms for performing my manipulation then it means that I cannot exploit complementation and its associated simplifications we would just have to treat a prime as a different literal from a ok.

If you have to do that you cannot use rules like this, you cannot use rules like this, but you also cannot use simplifications like these because a times a is not a in other algebras in Boolean algebra it turns out to be the case and the idea of do not care of course, is not there in those algebras and nevertheless it is just that the correctness is not interfered with you may have something that is not necessarily the most efficient, but the rules of algebra can certainly be used because they are applicable here it is just that Boolean algebra has a few more rules.

(Refer Slide Time: 52:20)

Algebraic Division

- **Support Set - set of variables in a cube**
 - $\text{sup}(abc) = \{a, b, c\}$
- **F is algebraic divisor of D when**
 - $D = F \times Q + R$
 - $F \neq \emptyset, Q \neq \emptyset$
 - $\text{sup}(F)$ and $\text{sup}(Q)$ are disjoint
- **E.g.**
 - $(ac + bc + ad + bd) / (a+b) = (c + d)$
 - but $(a + ab + ad + bd) / (a + d) \neq (a + b)$


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

Let us quickly go through one representation and the implementation of a division operation, it seems very simple conceptually, but you do have to go through some steps to make sure, it can happen properly, you do need an efficient implementation here and let us quickly define some terms and then define division and then quickly go through how you would perform division this is not difficult, but you do have to go through a certain set of steps.

Let define a support set of a cube to be the set of variables in that cube ok. So, a b c is the cube means the set of a b c is the support set defined for a cube. So, we say F is a divisor of d, if I can express d in terms of this you say F times some quotient plus some remainder where F is not null and Q is also not null, we also have the support set of F and the support set of Q being disjoint, this is the division that I am trying to perform a b plus a c plus b c plus a d that expression is being divided by a plus b, I should get c plus

d that is true in Boolean algebra, it is also true in other algebras, but if I have a rule like this that you divide this by a plus d, you should actually get a plus b, it is it is in Boolean algebra, but this will not be discovered by this standard division algorithm because in other algebras that is not necessarily true.

(Refer Slide Time: 54:16)

Cube Notation

- Expression regarded as set of cubes ✓

$$A = ac + bc + ad + bd + e$$

$$= \{ac, bc, ad, bd, e\} \checkmark$$

$$= \{C^A_i, i = 1, 2, \dots, n_A\}$$

$$C^A_1 = \underline{ac} \quad C^A_2 = \underline{bc} \quad C^A_3 = \underline{ad} \quad C^A_4 = \underline{bd} \quad C^A_5 = \underline{e}$$

$$B = a + b$$

$$= \{a, b\} \checkmark$$

$$= \{C^B_k, k = 1, 2, \dots, n_B\}$$

$$C^B_1 = a \quad C^B_2 = b$$

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

This is merely a matter of notation, but let us keep this in mind and. So, that we can actually come up with an algorithm for division 2 expressions in are there and you are trying to divide one expression by the other expression you can regard an expression as just a set of cubes this is a 2 level logic, we are talking about each of the individual nodes has essentially a 2 level logic function, right.

So, let us say I have an expression like this it consists of that set of cubes. So, you can regard that expression as the set of cubes c a one means that the set of cubes for expression A, the first one first cube is a c the second one is b c and. So, on and if I have a different expression a plus b then the set of cubes here is a and b and the c b 1 would be a and c b 2 would be. So, this is just a notation which will help me later on when I actually perform the division.

(Refer Slide Time: 55:33)

Division Algorithm

```
DIVIDE (A, B) {
  for (k = 1 to nB) {
    D = { CA | sup(CA) ⊇ sup(CBk) }
    if (D = ∅) return (∅, A);
    Qk = D / sup(CBk);
    if (k = 1) Q = Qk
    else Q = Q ∩ Qk
  }
  R = A - Q × B;
  return (Q, R);
}
```

For all cubes in B
Set of all cubes in A divisible by i-th cube of B
Quotient for i-th cube
Common quotient
Remainder

NPTEL (C) P. R. Panda, IIT Delhi, 2017

So, my divide algorithm which essentially divides A by B can be in general the following for k equals 1 to n B where n B is the number of cubes in B A is divided by B here, right. So, what are we doing we are going through all the cubes, I have in the set B, I collect first those cubes of A that satisfy this relationship support set of that cube of A is a superset of this support set of the kth cube of B, what it is doing?

Suppose this was A and B was in that; then it is picking one of these cubes of A, it is already picked the kth cube of B and it is checking whether this support set is a superset of this support set in other words, what is it doing whether one is divisible by the other one cube is divisible by the other that is all that it is doing. It is written in a way that looks very mathematically impressive, but there is not much happening there if D equals null, then return null A, what does it mean, if you cannot find a cube, then null a means its returning the quotient and the remainder.

Student: Remainder.

The quotient is null and the remainder is A.

Student: A.

So, this is more like some humour in this algorithm that we can try and recognize otherwise if it is not null then we are doing this by now, you can get the joke perhaps Q k equals D by support set of C B k, what would this be doing D is the set of all the cubes

that are divisible by the k th cube of P , all of those we picked up and divided by essentially saying that if it is A/B and A you remove this and you keep me.

If there were multiple here, then you remove a from all of them, we are picking up only those terms that are actually divisible right. So, I would be left with B and C . So, that would be the quotient for the i th k . So, that is alright. So, I picked up the quotient Q_k , if this is the first time, then I will say Q equals Q_k this is an initialization, but in if it is the second iteration or the third iteration.

Then I am doing an intersection of what is already there with what is there from previous iterations with what I just computed in the current iteration what is the point of that it is like saying I had C plus D earlier, but now it turns out that for the k th cube, I have just C , then I should take an intersection of these 2 which means that C should remain. So, it is not C plus D the plus D part of it is gone, but only C should remain.

We will quickly go through a different example, but this is just the algorithm the point of discussing it in this way is just to show that not much is there in this algorithm at the end of this when do you stop the loop is over all the cubes of b . So, that is what the k is iterating over at the end whatever has been maintained in Q that is returned as the quotient and the remainder that is returned is essentially you multiply Q by B .

(Refer Slide Time: 60:08)

Division Example - 1

$A = \{ac, bc, ad, bd, e\}$ $B = \{a, b\}$

	Iteration 1	Iteration 2
k	$k = 1$	$k = 2$
C_k^B	$C_k^B = a$	$C_k^B = b$
D	$D = \{ac, ad\}$	$D = \{bc, bd\}$
Q_k	$Q_k = \{e, d\}$	$Q_k = \{c, d\}$
Q	$Q = \{c, d\}$	$Q = \{c, d\}$
R	$R = \{e\}$	

```

DIVIDE (A, B) {
  for (k = 1 to  $n_B$ ) {
     $D = \{C^A \mid \text{sup}(C^A) \supseteq \text{sup}(C_k^B)\}$ 
    if ( $D = \emptyset$ ) return ( $\emptyset, A$ );
     $Q_k = D / \text{sup}(C_k^B)$ ;
    if ( $k = 1$ )  $Q = Q_k$ 
    else  $Q = Q \cap Q_k$ 
  }
   $R = A - Q \times B$ ;
  return (Q, R);
}
  
```

NPTEL (C) P. R. Panda, IIT Delhi, 2017

Those terms of A that are not there in this, they are read out as a (Refer Time: 60:08) ok, we should be able to quickly look at this with an example, here if that is my A and that is my B, the algorithm I promise is the same as they are the one although you have to do a graph isomorphism or something to check. So, k equals one that is the first cube which one would we pick up.

Student: A.

This;

Student: A will;

B. So, we are iterating through the cubes of B, A is being divided by B. So, we are taking the cubes one at a time and dividing A essentially. So, if k equals 1, then what is D? What would I pick up for D? These are those cubes of a that are divisible by the first cube of D which is.

Student: a; a c, b c, a d and d a, d a and d first 3.

First 3, why?

Student: Only a c and a d.

Only a c and a d these are the ones that have an a in them that cube that you have picked up from b should be a subset of. So, it should be a c and a d if d equals null, then you return null is naught what do I get for q k here..

Student: c d.

Right, you will get c d, it is a division right, I am removing the first cube here that first cube is a ok. So, that is my Q k and since k equals 1 that is my current Q and c and d of course, algebraically you can see that as of now c plus d could be a possible quotient because if you multiply that by a you are actually getting at least part of that expression. So, that is the first iteration second iteration k equals 2 I get what is the corresponding c b k that is b.

Student: c and b d.

Right. So, I would get b c and b d, all of those, terms all of those cubes of a that are divisible by b those I would pick up now when I divide that by this support set of this I would essentially be dropping.

Student: c b, d c and;

The bs and I would get c d this c d is then intersected with whatever I had maintained earlier and that is still c d. So, that is the end there are only 2 cubes in b. So, out of this with the q being c d and R being e does that make sense is that an expected quotient and remainder for this division.

Student: Yes.

Yeah, basically we are saying that a can be expressed as a plus b times c plus d e is the odd one out. So, that is the remainder what if it was slightly different let us say like this then. So, these were my terms b is though the first cube is not just one literal, but it is a bigger cube it is a x, then I do the same thing what do I pick up in my first iteration.

(Refer Slide Time: 63:18)

Division Example - 2

$A = \{axc, (bc), axd, (bxd), e\}$ $B = (ax)(b)$

	Iteration 1	Iteration 2
$k = 1$	$k = 2$	
$C_k^B = ax$	$C_k^B = b$	
$D = \{axc, axd\}$	$D = \{bc, bxd\}$	
$Q_k = \{c, d\}$	$Q_k = \{c, xd\}$	
$Q = \{c, d\}$		
		$Q = \{c, d\} \cap \{c, xd\} = \{c\}$
		$R = \{axd, bxd, e\}$

```

DIVIDE (A, B) {
  for (k = 1 to n_B) {
    D = { C^A | sup(C^A) ⊇ sup(C_k^B) };
    if (D = ∅) return (∅, A);
    Q_k = D / sup(C_k^B);
    if (k = 1) Q = Q_k;
    else Q = Q ∩ Q_k;
  }
  R = A - Q × B;
  return (Q, R);
}
  
```

NPTEL (C) P. R. Panda, IIT Delhi, 2017

All of these terms that have an a x in them, those would be picked up these are the 2 terms that would be picked up and the q one would be.

Student: C and;

C and d because I would drop a x. Now from them that is good that is my first iteration second one for k equals 2 the cube is b. So, the d would now be whichever ones have a b in them right that one this one. So, that is this.

Student: c x.

C and x d are my second Q k now I want to perform an intersection that leaves only c.

Student: c.

Does that make sense?

Student: Yes sir.

It is an unexpected thing. So, that is the end of the loop and I have as remainder, it is a bigger remainder. Now essentially you are saying this is a x plus b times c plus remainder, there are some terms that are not divisible and that is ok, this is just the general algorithm for performing division there is actually nothing Boolean algebra specific in this division this is just a regular division algorithm this process could be made a little more efficient in the following way we have some more information about this process.

(Refer Slide Time: 64:45)

Null Quotient

- Conditions for Null Quotient on Division (A/B)
 - B has a variable not in A
 - $A = ab + c$ $B = a + d$
 - B has a cube C where $\text{sup}(C) \not\subseteq \text{sup}(d) \forall d \in A$
 - $A = ab + bc$ $B = ac$
 - $|B| > |A|$
 - $A = a + b$ $B = a + b + c$
 - Count of any variable in B > in A
 - $A = a + bc + d$
 - $B = ab + ad$
- For functions f_i and f_k for nodes i and k, (f_i / f_k) is Null if path $i \rightarrow \dots \rightarrow k$ exists

$n \in \text{sup}(f_k)$
 $n \notin \text{sup}(f_i)$

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

Now, where is the opportunity for optimizing, we want to divide one node by another potentially there are n square such division there are a large number of such divisions to

check in that substitution algorithm to find whether we can replace one or whether we can realize one node in terms of another node. So, because that that would be a large number of trials, you also expect that most of these are many of these in general would be null many of the division. So, in the general case there might be none why what are conditions which are actually very frequently occurring in a general logic network where you expect the division to not really go through.

Student: In the paths; so, already planned the inputs.

Yes.

Student: (Refer Time: 65:41) because if they are completely disjoint not able to define.

Ah yeah we can realize some in terms of path, but before that one expression is being divided by another expression right and. So, its some expression is being divided by some other this is a being divided by b and we want to see whether I have a Q times b plus some remainder as it turns out, there are a number of elementary checks that can help us prove this research base you do not have to do it a quick look at these 2 expressions can actually tell us that the division cannot be successful what would it be informally can you think of some such.

Student: Results would be there which are common.

Yes, just looked like a joke, but it is actually something we can start off with if b has even one variable that is not there in a , then that division cannot go through because when you come to that here when you come to the d here in this example you will get null.

Student: Null.

If even one of those quotients results in null then the intersection is null and the division does not go through you could actually enumerate many such rules not just it has does not have a variable, but if it has a cube just generalizing this whose support set is not a subset of any of the cubes of a then also that division cannot possibly go through.

Student: Right.

Right, if I have an $a \subset c$ and I do not have an $a \subset c$ as a subset of any of those cubes, then also, it cannot go through, I can actually come up with quite a few such rules, if the number of terms here is 3 and that is more than the number of terms of a then also it cannot.

Student: Will not go.

This is just a small set of these rules. They are useful, they help us not go through that cumbersome division process unnecessarily and finding out in a later iteration that the question is actually null. So, count of a variable in b ; if that is greater than in a ; what is the meaning here you take.

Student: A is;

There are 2 ways here, but there is only one a in this. So, yeah there are many such rules could be there, but there is one from a part the point of view, since you mentioned it, we can say one thing there is a function associated with each of these nodes. So, if that is my f_i , like a function associated with k is f_k . So, if a path exists from I node I to node k then what can I say about divisibility between f_i and f_k .

Student: Can you repeat the question?

This is a path just through that logic network there is a path between every node to another node this is very common, right.

Student: Yeah.

What divisibility relationships, we can expect if there is a path.

Student: You will have n .

Or let us say, I try to divide k by i it looking at the first and the last one or I try to divide i by k .

Student: i by k will be null.

Ah why?

Student: Because k will have n .

Yes right if there is a path from i to k , then function corresponding to f_i divided by function corresponding to k that would be null. You can easily see that look at whatever is the previous node, the literal corresponding to that node is there in k , but it is not there in I it is true even if this is just the immediate criticism then to that relationships of course, of course, because this is the case that function f_k by f_i might be ok.

Student: Yeah.

Yeah, but f_i by f_k would not be there because you do not expect cycles in this graph anyway ok.

(Refer Slide Time: 70:03)

```
SUBSTITUTE Algorithm

SUBSTITUTE (G(V, E))
for all pairs (i, k) {
  A = set of cubes in fi   B = set of cubes in fk
  FILTER (A, B) for Null Quotient
  if Pass {
    (Q, R) = DIVIDE (A, B)
    if (Q ≠ ∅ and Substitution is favourable)
      Substitute fi = k x Q + R
  }
}
```

NPTEL

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

So, with these, I can have a substitute algorithm that looks like that that is my graph of vertices and edges for all pairs and we could still have all pairs, but what we are doing is we are quickly getting rid of many of those pairs in this. So, if a is the set of cubes in f_i B is a set of cubes in f_k you perform a filtering for the null quotient of the kind that we saw all those rules you could actually apply and check whether you can just filter out that division if it is passing then you can go ahead and obtain the quotient and the remainder by actually applying the division algorithm, but these rules are very simple.

So, you do not have to perform the division which might actually be a more expensive operation in general ok. So, if you have a non null quotient and that substitution is actually favourable that favourable is left ambiguous there in the general case, it could be

that you quantify it in some way in the sense that some other parameters could actually influence whether you would go ahead with it or not then you can substitute f_i by whatever is the function for k and. So, you have a quotient and the remainder.

Student: Sir, if we have filtered from null quotient why are we judging that q naught equal to ϕ I mean.

That filter is not an exhaustive filter if we established that path is there then it is not, but it does not mean that if the path is not there, then it is not null.

Student: Ok.

Right, this is just one simple check that definitely tells us that there is no need to divide, but you can still divide and find that it is not yeah that is all there is there; obviously, we did not cover all of the Multi-level Logic Optimization, this was just that substitute algorithm, but some of these are related right this division could be used as part as a sub routine in some of the other algorithms also, but that is the extents to which I wanted to cover the first phase of the Multi-level Logic Optimization which is independent of their technology library the second stage would be the technology mapping where now we have an optimized net list, but that net list is a generic net list, we did not take care of any particular physical properties of the library elements that we can do in the second phase right.