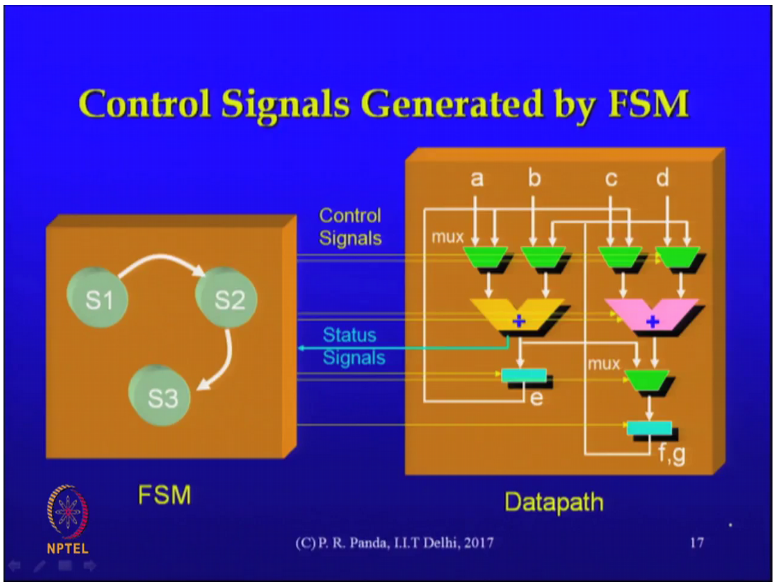
Synthesis of Digital Systems

Dr. Preethi Ranjan Panda
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi

Lecture – 08
Language front-end & Design Representation

So, we introduced some of the major tasks of high level synthesis in the previous lecture.

(Refer Slide Time: 00:27)



One important thing to note is that for the same specification in the form of an HDL there may be a very large number of implementations. Where were all the different variations come from? So, they are not all equivalent. So, somehow the job for high level synthesis should be that we pick up at the end the best out of all the choices that exist,. But where are the choices, in this example that we started off with we constructed an FSM and a data path this is the output of the behavioral synthesis, but where did we have a choice or was this the only possible output from the synthesis process.

Student: Sir, because constraints we had that we have to.

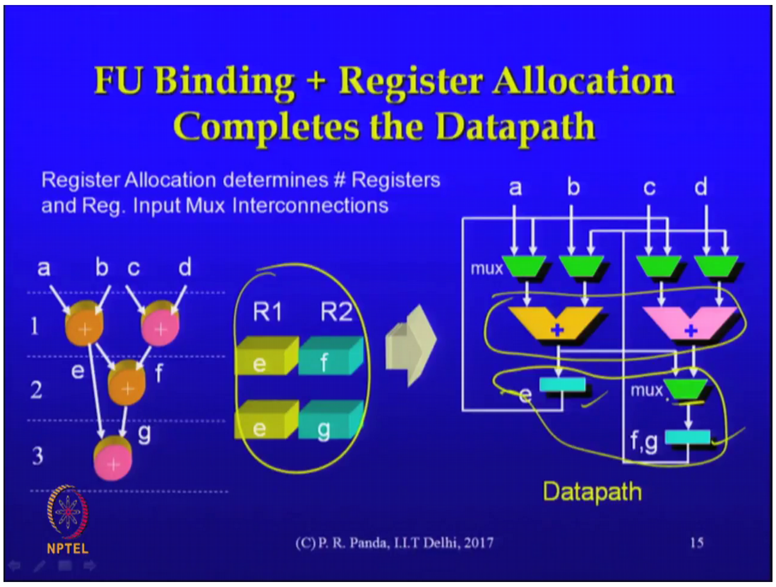
Constraints were externally imposed to the synthesis what are the inputs we said that the primary input of course, is the behavioral description in the HDL, but there are other inputs. The constraints in the form of resources or delay those are inputs we do not have a choice it is assumed that the designer through other means may be through whatever external means there are these constraints that are imposed. So, we have to operate within the constraints, constraints have to be respected, but within those constraints we may still have a large number of choices did we have any choice in this example, small example that we built up where was the choice that was excessive in the FSM did we have a choice.

Student: Yes.

What was the choice?

Student: Started dummy states.

(Refer Slide Time: 02:21)



FU Binding + Register Allocation Completes the Datapath

Register Allocation determines # Registers and Reg. Input Mux Interconnections

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

15

Could you have inserted dummy states what in the behavioral synthesis process influences the creation of the FSM they should make sense they should arise naturally in some way. What choice did we have?

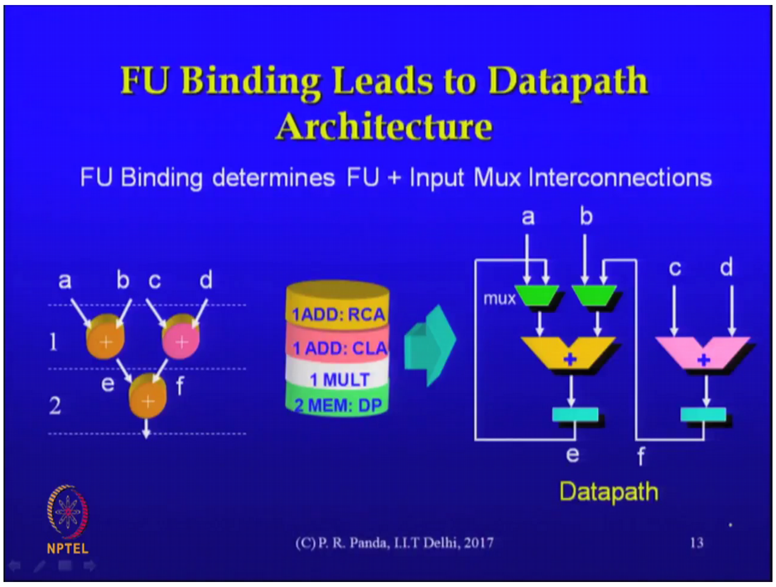
Student: To be optimized everything so that we can finish things in 3 cycles.

Right.

Student: We can designer has given the maximum usage of the adders, but we can choose.

Yes, latency which is 3 cycles in this case was not a constraint there are some constraints that it is not clear what they were perhaps.

(Refer Slide Time: 02:57)



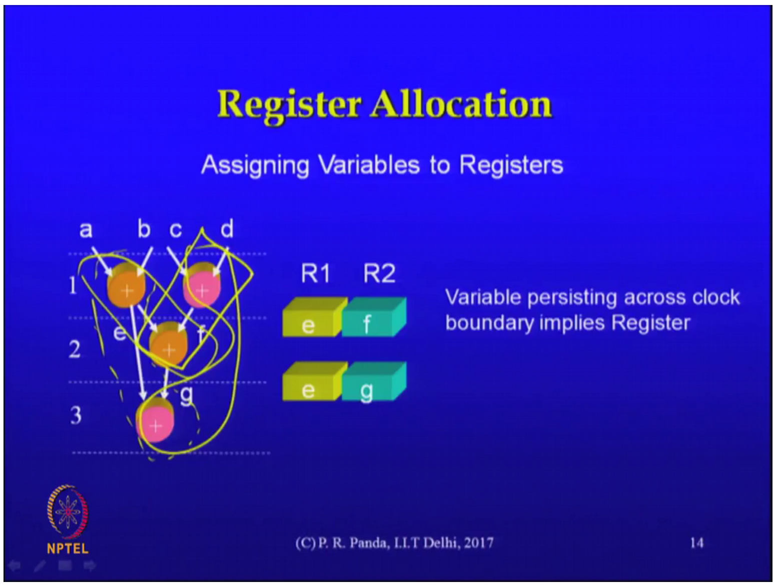
FU Binding Leads to Datapath Architecture

FU Binding determines FU + Input Mux Interconnections

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

But I did say that in that library when we started off right these were constraints that you have one adder of each type and two memories one multiplier and so, those were the only constraints, but on delay there was no constraints.

(Refer Slide Time: 03:10)



Register Allocation

Assigning Variables to Registers

Variable persisting across clock boundary implies Register

(C) P. R. Panda, I.I.T Delhi, 2017 14

So, the schedule that we were working with which is this one that finished in 3 clock cycles that was our decision this is not the only valid schedule. Why did we choose this schedule?

Student: Earliest.

We assumed that constraint that you have one pink adder and one orange adder in this. So, with that as the resource constraint it seems as though you perhaps do not have too many choices here. So, two add additions have been mapped to one adder and that is those two and the other two additions have been mapped to. This adder this is the function unit binding step right that is the function hundred binding that is been done for this graph did we have a choice. What other binding could have been done.

Student: Swap orange and pink.

You could have done this kind of sharing right, maybe this kind of a sharing. That would also have respected the resource constrained and also this the resulting schedule that we already decided make sense in the first step. What difference would that have made to the data path?

Student: (Refer Time: 04:41).

Different hardware what different hardware would have been generated we are still operating under the same resource constraint which means that these function units of course, would have still been there, but the muxes and the connectivity might have been slightly different. In this example it is not clear that it makes a qualitative difference to the result, but in general you actually have a very large number of choices you have a very large number of operations and some number of resources, but the moment the number of resources is more than 1 you have the resource sharing decisions to be made. We also made register sharing decisions here we said that we will take two registers it seems as though this makes sense for that particular design, but when you have n variables and r registers actually here register is not a constraint how many registers you use is inferred by our algorithms. So, we did not take that as a constraint although you could formulate the synthesis problem also in a way that the registers is a constraint.

So, here we just wanted to minimize the number of registers that we are using and a certain sharing of variables into registers was used to generate this part of the circuit let the number of registers and also the select circuits at the input of the registers. So, it seems as though in this small example there were not too many choices some choices were there, but a large number of choices were not there. But the moment the number of operations becomes large and number of associated variables becomes large then we have to make major choices with respect to the register allocation with respect to the function in it binding, but to begin with the shuttling you have resource constraints that need to be respected. But otherwise you have a large number of operations and they have to be mapped onto clock cycles it is not as though you have only one choice we have a very large number of choices to exercise with respect to scheduling.

Actually all 3 problems would fall into what are called n p hard problems when you analyze their computational complexity. Scheduling of course, is a classical problem that is valid in many different contexts. Instances of the scheduling problem arise in a large number of different applications this is just one such application. But the others function unit binding register allocation and so on they too are difficult problems to solve in general.

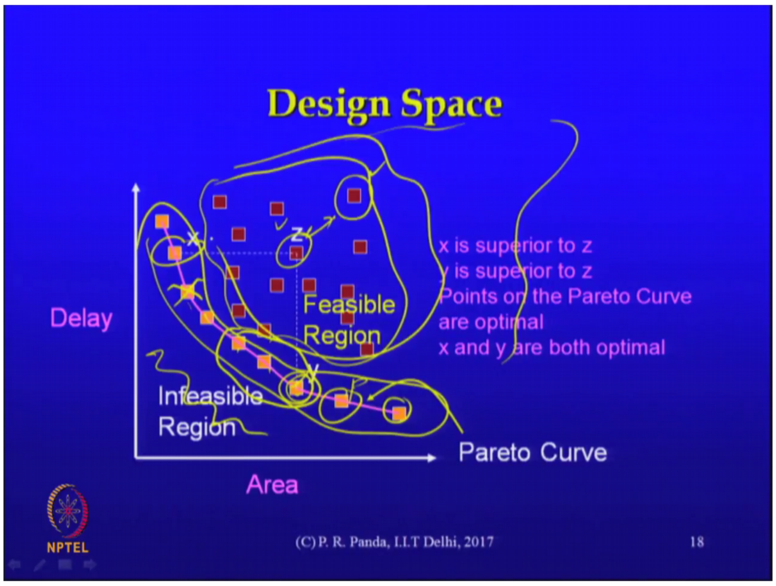
What do we mean by difficult problem? It is not as though you cannot come up with an algorithm to solve the problem, but you cannot guarantee optimality in the algorithms that you come up with on. One hand these are practical problems and therefore, we need algorithms that are practical which means that they should run reasonably fast on large size designs, but what you are compromising is that you are giving up on the claim of optimality that you cannot say you have the best possible solution.

Student: Are these polynomial class or exponential class?

Are these polynomial class or exponential class of algorithms. We did not get to the algorithms of course, in the subsequent discussion we will talk about specific algorithms for scheduling and register allocation and binding and so on. So that is that the time to discuss the complexity of the respective algorithms. But we are warning ahead of time that you cannot expect optimality from practical scheduling problems. So, you want them to be polynomial in their complexity because that is more manageable in a practical tool, but you do give up on the claims of an exact solution.

We were here where we generated an FSM it is not as though there is only one possible FSM. Here it seems as though the problem was simple enough and we scheduled everything into 3 clock cycles and therefore, this is an FSM with 3 states, but in general the number of states of the FSM and the structure of the FSM also these are all variable these are the result of the scheduling decisions that have been made, but there are a very large number of decisions to be made in the scheduling as also in the other steps of the high level synthesis.

(Refer Slide Time: 09:25)



Design Space

x is superior to z

y is superior to z

Points on the Pareto Curve are optimal

x and y are both optimal

That is just to motivate the overall picture of a design space that exists for problems like this what we have indicated here is a two dimensional picture in which you have area on one axis and delay on the other axis considering that these are the parameters of importance when it comes to evaluating our design.

What do each of these points represent? They represent a sample solution, solution consists of all our decisions of scheduling resource, allocation binding and so on. It would result in some delay some area delay here just refers to the number of clock cycles area is the total area of the resulting circuit. So, a large number of possibilities might be there or each of these is a possibility and I can plot all of them in an area delay curve to understand where all the solutions stand particularly with respect to each other. Looking at a distribution like this what can we say about the desirable solutions? Each is a solution I have a large number of solutions can I throw out some of those solutions right away just by looking at this distribution, I can throw it out if I can establish that there is at least one other solution that is always better than this one with respect to every parameter. What are examples?

Student: (Refer Time: 11:02).

The one that I have circled I can throw that out why.

Student: Because there were solution where like which is labeled as z that is more efficient in delay as area.

Yeah. So, I have at least one other point if I compare these two points it is clear that this is always better than that one. So, I can throw out so in fact, I can throw out all of these points that are colored brown. Why can I throw them out because there is at least at some point in this space that seems to be better than or equal to all those other solutions in the interior of the feasible region. So, this region here we can call infeasible because there are not any points in there you do need a minimum area and a minimum latency without which a realistic solution might not be there. But there may be a number of solutions in the other space in the feasible space all of which are not necessarily interesting. The ones that are interesting we can cover in what is called a pareto curve, consists of design points. That belong to this class where you do not have another solution that is clearly better than that solution with respect to every axis.

But the relationship between these points in the set is that you take any pair of points compare these two points y and x, y is better than x with respect to delay, but x is better than y with respect to area. So, we can capture all of these in what is called a pareto curve and all of those solutions would be called pareto optimal solutions. This is an example of a problem domain where there is not a single best solution. Sometimes your problems might be of the form that there would be a single the solution and the objective would be to find it, but that might not be the case always.

Student: Should all of these points have been generated after running (Refer Time: 13:09).

How did this point get generated of course, this is an illustration with respect to comparing a large number of solutions that does not mean that you must generate all those solutions first and then pick up all the solutions that lie along the pereto curve. This is just warning that is the design space is large and also an indication that some of those solutions are interesting for us automatically. It is hard to at least given that your parameters are just delay and area it is hard to say that one is better than the other. So, we connect all of them and we could submit these as candidate solutions to a designer who may make a manual decision with respect to which one is best. It could be that the final decision is based on other parameters not necessarily just delay and area maybe there are other parameters that come into the picture which might help in pruning the design space down further maybe all of these are not interesting with respect to a third parameter called power or security or whatever there may be a large number of other such parameters. So, maybe we can boil the solution space down to just 3 of them and then one can take a decision based on the relative importance perhaps of area and delay between the selected designs maybe.

Student: Sir, if I have multiple parameters to look into let us say the area and power that would mean that my data is an into multiple dimension then probably doing this kind of analysis manually would be cumbersome.

This is not a manual analysis this like I said it is not as though our algorithms are going to generate all of those design points and we are going to select among them. The algorithms are of the kind that will usually generate one of these solutions the more practical algorithms that we will be using will generate one of them, but perhaps they will generate this one right. But of course, we want to understand that that there are other solutions that are also pareto optimal that our algorithm might not be generating at all that depends on how we have written algorithm we do want it to converge fast we do want it to generate us a valid result first. In the rest of the space the points that I just distributed they are implicitly being thrown out by whatever our algorithms are for these things it is not as though they are being explicitly enumerated and the choice is being made yeah.

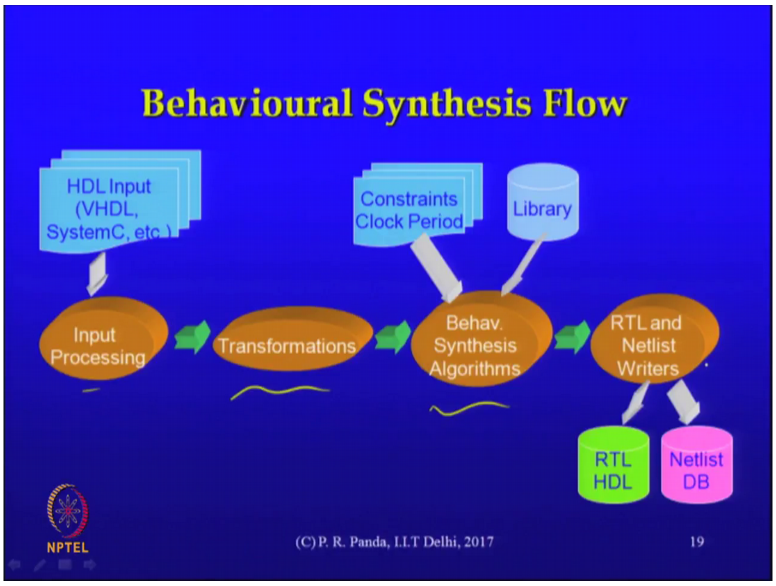
Student: Report a member of area of the delay or any other matrix also (Refer Time: 15:51) point there are the way so far. Understand is as well as practical create the design then compute the delay. So, or is there are an early estimation kind of (Refer Time: 16:05) where it does not do all these kind emphasis. In practical (Refer Time: 16:10) of cases that some of these data points in the design space will have basically we come up with the pareto come in too much faster.

HLS tool will always proven the design space and will select a subset out of the large number of design points that are then there is no hope of enumerated all the possible solutions. The space is just too vast even though I have shown a small number of points in this graph here there is no question of enumerating everything it just blows up. So, a large number of those points are getting thrown out by HLS. So, if there are some solutions that it is generating for us to investigate further maybe take it down to physical design or something like that it has already implicitly done a lot of pruning. That we can easily see as we understand scheduling when you pick up one of those algorithms like scheduling there are a very large number of choices that a scheduler needs to make and in the interest of simplicity of the algorithm it has to throw out a number of those choices who knows and among them it could be that the optimal choice actually got thrown out yeah its certainly possible right.

Student: And eve in the whole design throughout typically a higher structure levels much of let us say part of (Refer Time: 17:36) a kind of optimization would have been done virtual (Refer Time: 17:38).

This happens at every stage of the space, space that is the way to keep it manageable and the way to ensure that you do not submit every possible solution all the way to place in route and detected that time that this design is better than the other. As you know we are arguing here at a little higher level of abstraction when we say this area is greater than that area we are just adding up all the component areas and are saying that this is the area of the design, but that is not all like we have seen earlier when you actually do the placement and routing there is a routing related area that we are not taking into account when we just add up the component areas. But such simplifications have to be made in the interest of a practical design scenario right. You cannot afford to go through go down the entire path and generate masks for all the possibilities that are there. So, every tool is implicitly making all of those choices and possibly at the end of its step it is submitting for further investigation a small subset of the possibilities.

(Refer Slide Time: 18:51)

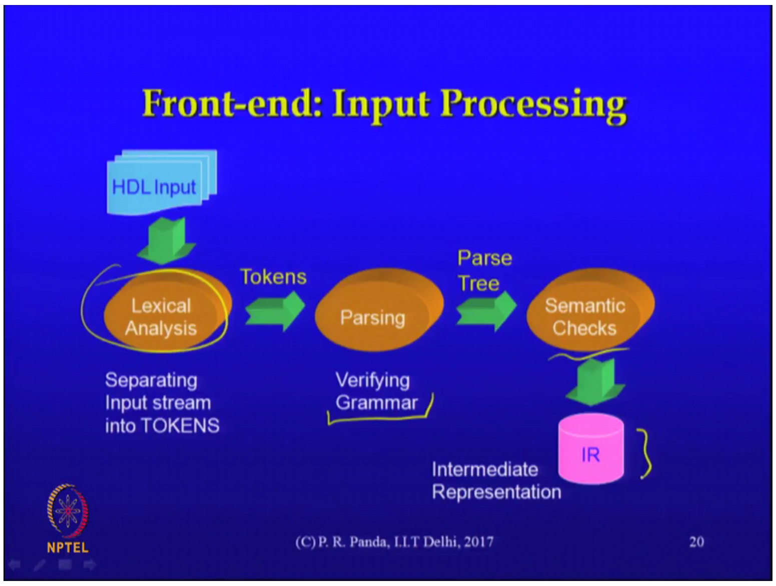


Behavioural Synthesis Flow
HDL Input (VHDL, SystemC, etc.)
Constraints Clock Period
Library
Input Processing
Transformations
Behav. Synthesis Algorithms
RTL and Netlist Writers
RTL HDL
Netlist DB
NPTEL
(C) P. R. Panda, I.I.T Delhi, 2017
19

Let us look at a high level behavioral synthesis flow and put in perspective where the algorithms that we will be discussing fits there. So, you have an input processing stage where however, you have specified the input in the form of some kind of HDL whether it be a VHDL or system see and so on. The file input is converted into some kind of an internal representation early on there are some high level transformations that you would be performing will study them immediately afterwards. These are somewhat independent of the target architectures the details of the algorithms that you would be running in the behavioral synthesis. So, they are sort of abstract and there is a lot in common with what compilers do for instance because the direction is similar.

Then these are the core behavioral synthesis algorithms they consist of those tasks that we already saw the scheduling and resource binding and register allocation. After all these decisions are taken there would be maybe a conversion into an RTL format or some kind of a netlist format depending on whether it is reasonably symbolic output like it would be if it is an FSM or whether it is a more structural output of course, either way you can write an HDL out, but you could also write an internal representation in the form of some kind of database which as the final output.

(Refer Slide Time: 20:33)


Front-end: Input Processing
HDL Input
Tokens
Parse Tree
Lexical Analysis
Parsing
Semantic Checks
Separating Input stream into TOKENS
Verifying Grammar
IR
Intermediate Representation
NPTEL
(C) P. R. Panda, I.I.T Delhi, 2017
20

Expanding on that first step which is the input processing we have an HDL input the opening step is what is called a lexical analysis is in the lexical analysis you separate out your VHDL model into tokens some of which as are recognized as keywords we have entities a token and architecture the word is a token. And others would be classified as identifiers that are not necessarily keywords and you would also have separators you would have constant integers and so on. So, it is just a classification of the input into a bunch of lexical objects called tokens.

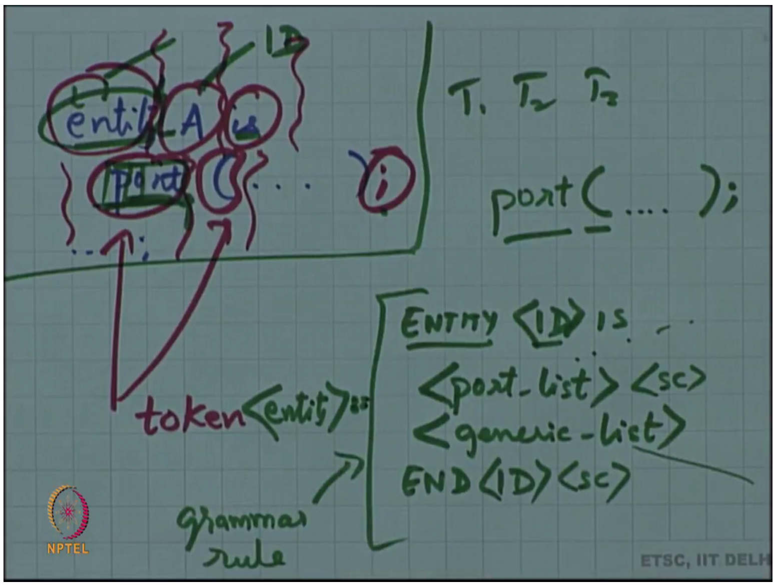
These tokens are submitted to a parsing stage where what you have submitted what you have written in the HDL is verified against a grammar of the language. So, every language has an implicit grammar defined and the parser first establishes that what you have written is grammatically correct, if not then you get what is called a syntax error that is flagged at this stage you have violated the grammar of the language. If everything is fine then if there are a few other simple things that we can do at this stage the semantic checks include things like type mismatches this may not directly violate the rules of the grammar, but a separate step usually would be performed after grammar is verified to check whether you have any other errors of the kind of type mismatch.

So, if things are fine then we store the entire input description in some kind of intermediate representation so that we can continue with the rest of the processing. So, it is important to note that you do not get back to the input file. So, in a later stage you do not go back to the VHDL file to see what was the port what was the architecture what were the processes and so on. All of those are somehow captured into an intermediate representation which of course, should be powerful enough to store within itself anything that you could possibly write that is valid in the language.

Student: Sir, what is this r stream and token what is the token.

The token is what we are separating the input into. So, if you have entity A is port and so on each of these is a token entity is a s token these are words, but that is the early stage that is the first stage of a parcel right. So, this is a token, that is a token, that is a token, this is a token if you have semicolon that is a token we are just separating out an input stream into different tokens, each is a token.

(Refer Slide Time: 23:19)



That is language defined. So, the language says what is the separator that is permitted here typical separators maybe space or a tab or a new line and so on that the language defines what that is the lexical analyzers job it is to start with this to construct these tokens and throw out all of those. So, for example, whether you have two spaces here or 3 spaces here or some tabs and, all of that is filtered out because as part of the specification to the lexical analyzer you have said that the space is a separator. So, at that stage all of those artifacts are thrown out.

Student: You makes that what is this tree this is.

The parse tree is what is the output of a parser where. So, it is at this stage in fact, your entire design is stored in that parse tree format where the tree structure comes from it is just that is an output that you construct as you do the parsing.

Student: Dependency is (Refer Time: 25:16).

No, no there is no debit there is no analysis at this stage this is purely a syntactical captured right, the program has been written in some way this is by the way this is identical to what happens in a compiler; when you write a programming language. So, of course, as of now there is nothing that is hardware specific or dependent here these keywords. So, there is of course, a distinction here between a keyword entity this is expected so that is provided as an input to the lexical analyzer. Entity is, port these are the keywords that are already provided so that it reads these things a character at a time and it builds that word called entity.

How did it build that word? It took these things a character at a time it looked at that separator and built that word entity and it is matching it against a set of keywords that we have already submitted to the lexical analyzer and the output from that part of the lexical analysis is this token number 5 that might correspond to entity. So that is all that is happening here.

So, that is what is output here is it is essentially a stream of tokens. This entity is a known token is not a known token. So, it may say that this is an identifier it is just some identifier not one of the keywords and it also will pass what that string is so that somebody later on can do something about it. That is all that is there when it sees a semicolon it says that this is a semicolon. I am already informed that a semicolon is one of the recognized characters of the language so that is also is a token. So that is all that is here I have a stream of tokens as the output of the lexical analysis.

Now, that parsing stage consists of a verification of the grammar. So, just like we gave some rules to the lexical analyzer those rules were these are the key words these are some of the characters that I am expecting in this language they mean something to me. So, so far it is not checking whether that character should be there or not at that particular place that is the job of the grammar verification strategy. Here it is just saying that I found that character right so that character means something that is part of this stream of tokens that I am sending. But it is intelligent enough of course, to realize that you have a p here that p does not match any keyword, but it will extend the matches will try to extend that word until a match could be found. So, it will read the next one.

So, you see that the word terminates when that separator that we have already specified we specified what the separators are when that is encountered that is when that word is constructed and is first matched against the set of keywords that you have specified. There is a little finite state machine that is built as part of that lexical analyzer we are not going into that, but you can see conceptually it is just a stream of output tokens from here. The meaning is not necessarily analyzed at this stages.

The next stage is a grammar verification. So, to this we somehow have to indicate the entire grammar that is expected for the language what. So, for our entity specification what would that grammar be. It would be let us say I have an entity that is a token. So, the stream of tokens that are acceptable for a particular statement this is an entity statement there are many kinds of statements here. So, let us say it is an entity declaration and what would the entity declaration look like in the most general case, you already know the syntax you you know examples of how to write entities how would you describe it formally you start with the keyword entity then.

Student: (Refer Time: 29:24).

Then there is an identifier that identifier is something that the user is choosing right it does not matter any as long as it does not conflict with one of our keywords we will accept it or some rules might be there on what you will permit and what you will not permit in an identifier so that could be this why I say entity and then identify then.

Student: Is.

Is should be there then.

Student: port (Refer Time: 29:56).

Port, remember I can have port and this, this is a token this is also a token parentheses is also a token after that what. But I have some things that are that is a little more complex right, I have to somehow carefully define what is there because that is not fixed you may have any number of ports and my grammar rules should be such that it is able to anticipate and correctly match and capture a list of ports that is of variable length. So, we will get to that, but after that list is captured I would have a closing parenthesis and I may have a semicolon let me just call it a port list. So, I will call this as an identifier I will say that this should be followed by a port list. Let me just call this that list of ports that could be a variable length.

Remember even though I have formatted it like this your own format and indentation all of that is different so that is handled in a way that we inform the lexical analyzer that the separator character is space or tab or new line and so on. So, this is not sacrosanct that I wrote it in the next line that is immaterial here because we are just taking tokens here. That you put 3 more lines in between is something that is lost at the interface of the lexical analyzer and the parser the parser does not even see that you have put tabs or new lines or 10 lines or 20 lines 20 blank lines right there is a port lists and.

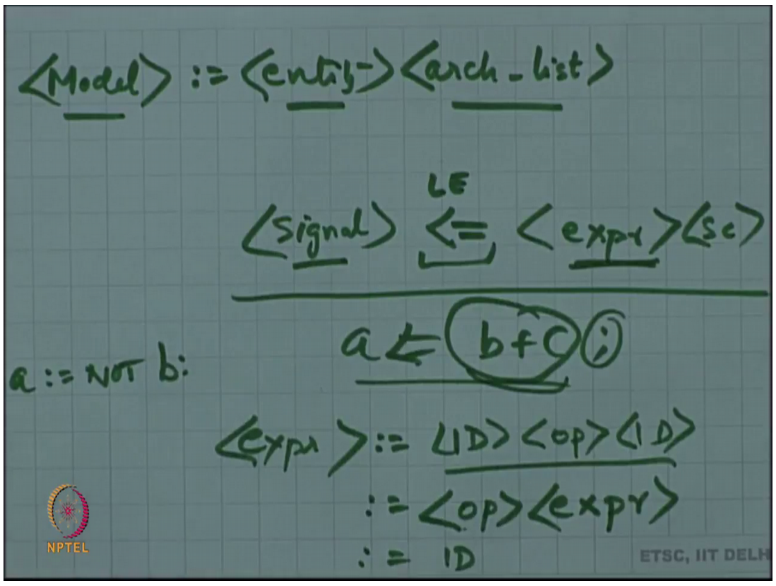
Student: (Refer Time: 31:40) generic.

Then the generic parameters might have to be given. So, there would be an associated generate list. Then, then I would have that end ID and if there was a semicolon you can say that if there was a semicolon, semicolon is also one of our tokens. So, you can say there is a semicolon here you can say it is a semicolon. Do you see that this is an effective high level capture of what to expect in entity declaration. Any entity that the language permits you to write I should be able to anticipate in what is called a grammar rule. This is one grammar rule of the language. It is worth going into a little more detail this is of course, not the complete description what is port list, what is the generic list these two we have not defined so, but this is the rule for an entity. This is one rule of the language let us say we start off with an overall loop rule what does a complete VHDL model consists of what we just said is just the entity part of it, but your entire design consists of what.

Student: (Refer Time: 33:20).

In addition to the entity there is an architecture so I say that there is a model a complete model may consist of an entity followed by architecture.

(Refer Slide Time: 33:25)



```
<Model> := <entity> <arch_list>

                LE
         <signal> <= <expr> <sc>
         ____________________________
a := NOT b;      a <= (b+c) ;

         <expr> := <ID> <op> <ID>
                := <op> <expr>
                := ID
```

Student: (Refer Time: 33:36).

But actually I may have any number of architectures remember that you may have multiple architectures. So, you have you can maybe have a list of architecture. That is my single line grammar rule for the highest level structure of the language. Of course, we are developing this a little informally the language will have a reference manual where a precise grammatical structure is outlined its worth taking a look at it if sometimes we are not clear about whether the syntax should be this way or that way those rules are what are sacrosanct for a language every language comes with such a bunch of grammar rules that have to be clearly satisfied.

So, I had model consisting of entity and arch list entity is what we just developed so that is my entity. So, similarly architecture I would have another rule within that architecture I would allow a number of concurrent processes or signal assignments right. Each now getting to a signal assignment you can see how this is recursively developed right, I can say that a signal assignment should consist of in the very simple case I have a signal or an identifier where as of now it is just an identifier you have less than or equal to this is also a token I should have alerted the lexical analyzer to expect such a thing that looks like less than or equal to, but depending on where it is used it should be able to correctly disambiguate the meaning of that symbol.

Then I will have an expression let me just call it an expression right this is the equivalent of a is equal to b plus c that is an architecture statement concurrent signal assignment at the architecture level and that is anticipated by this kind of a rule that is specified to the parser that is part of our grammar. So, I have signal this assignment operator and expression, that expression I should be able to capture any kind of expression you may have 100s of operations in there and that is also a valid expression right there is no limit to how complex that expression is there is one more thing this also needs to be specified this grammar has to have everything so that semicolon token also goes there. That is simplified version of a concurrent signal assignment for us.

And so that is one in the architecture as part of the architectures grammar I should be able to say that I can have any number of such constructs, each of which is either a concurrent signal assignment or a process right because we said that they are all allowed at the architecture level and they are all operating independently with respect to one another.

Then what so, this is simpler, but the expression in turn I should have another rule that says expression means what, yet what might that translate to each needs a rule. So, for expression I need a grammar rule what might that be.

Student: (Refer Time: 37:02).

Ending in semicolon actually that is not true the statement ends in a semicolon expression is this part of it only right. So, expression does not end in semicolon in this particular language other languages it might be different what might a rule be for an expression.

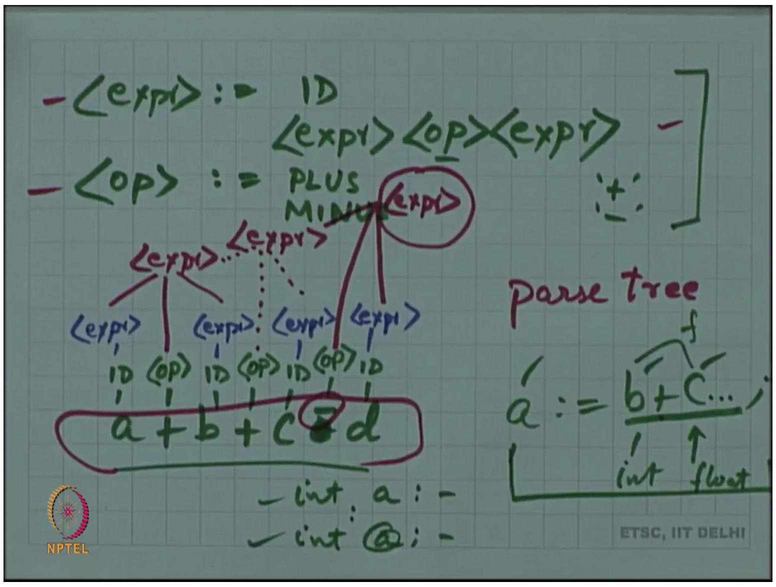
Student: Operands and operators.

Yes, it would be some sequence of operands and operators operands are what here there they are those identifiers right. So, an example would be I have an identifier, I have an operator and another identifier perhaps that is an example.

Student: Unary operator in beginning.

Yeah, you actually have to allow for many different kinds of operator this is a binary operator that we just said, but that is I should be able to expand that expression rule to all the kinds of operators that you may have. So, if you have something like a NOT a equals NOT b that does not fit this format. So, you need a different so that maybe has this operator actually in the general case it is an expression right an operator followed by another expression. So, it is recursively specified in that way and then among the rules of the expression would be that you just have an ID that is also an expression that could be used to hierarchically compose complex expressions because let us say I specified two things I have expression could be expanded as just a simple ID or I say it could be expression operator expression.

(Refer Slide Time: 38:41)



Using that just these two rules let us see if we can reconstruct an expression like this a plus b plus c plus d or may make it minus right. So, this part of it is not defined yet I need a separate rule for me to tell me what is allowed as part of that operator. So, I add another rule that says operator could be plus it could also be minus both are binary operators where did that plus and minus come from we have to instruct the lexical analyzer to return that token it is just a token that if you see plus that character then you should return that plus token if you see the minus character you should return the minus token these are just tokens fine. Identifiers of course, is a another token.

So, can we construct this complex expression from multiple applications of the expression rule you see that at an elementary level a b c d these are what these are recognized as identifiers by the lexical analyzer itself. So that is an identifier, that is an identifier, that is an identifier.

The plus is an operator these so that rule that I have here for the operator that would be matched for me to get an operator from that plus character right. So, I have an operator that is what it looks like. Then let us proceed left to right and finally, we want to recognize that this whole thing is an expression right so that is the job of a parser you see that this part means. So, expression being the simple identifier means that all of these guys right I all of these identifiers can be. So, let us say recognized as expressions by themselves right. So, this could be recognized as an expression this and this is recognized as an expression.

Now, that rule for the binary operator could be matched because you say that an expression could be composed of an expression an op and an expression I have fine this is recognition of an expression. So, far we are proceeding just left to right. After that I have this I have that operator and I have this expression these could be matched to generate a new expression finally, this and this could be matched to final expression and in the process what I have done is there was a complex specification of an expression you can see how they generalizes to any number of operators and any level of complexity of a single expression.

So, essentially we have recognized that whole thing as an expression the tree structure comes about from this recognition process. So, this whole thing is what would be called a parse tree. The grammar itself has a tree structure in the way it is specified and the way it is recognized a tree is explicitly built out of this recognition process. So, we did there that in formally here, but this is a formally what happens in a parser that you have a large number of these kinds of rules that are specified that constitutes the grammar of a language and you have specified the model in a file with some syntax this kind of a process can be used first of all to check whether your syntax is correct or not. If it is incorrect then some rule will fail in its matching it started the matching process, but let us say you input a recognized operator you put some garbage symbol out there.

This recognition process that we just informally went through now that will notice the violation and will report what is called a syntax error, that you provided something and I was not expecting that token at that time those are the syntax checks that are first done. Yeah.

Student: Sir, in this representation.

Yeah.

Student: In the flow chart, it seems r lexical analyzer will not build the final IR data structure to started here or do not do anything about it.

Right.

Student: But will just create the parse the entire thing to be parse (Refer Time: 44:12).

Yeah, but just correct the lexical analyzer is not building it lexical analyzer is only sending a stream of tokens to the parser, it is the parser they are not concurrent they are sequential the output of you could implement it concurrently with that the output is just a bunch of numbers and strings from the lexical analyzer. It is at the parser level that we are first going through this recognition process, but you can see that as part of that recognition process I can build that data structure.

Student: Yes, that is what I was telling (Refer Time: 44:50).

Right, that parse tree is my data structure that is being built of course, if there is an error then everything is abandoned it does not make sense to build at the parse tree if your program is wrong right. But at the end of it if everything is recognized and a correct model is recognized this you can see this is nothing to do with VHDL. It is just that this same process would be followed irrespective of what your language is the rules would be different for different languages type. But at the end of it if the parsing is successful then you have constructed along the way elements of that parse tree and then, there is no separate parts that is needed for actually creating the representation you do have a valid representation that is what we are calling this intermediate representation at this stage.

Student: Sir, in hierarchical representation this was flant (Refer Time: 45:42) in the sense getting hard entity inside entity called inside entity. So, let us say we have a (Refer Time: 45:48).

We did not have an instantiation.

Student: Yes.

Of an object S.

Student: In case we have hierarchical sense in it in the input data.

Yeah.

Student: Then would it make sense to create IR as soon as we are done with one of the hierarchy that the.

Ok.

Student: (Refer Time: 46:04).

There is an interesting question of should you put more and more intelligence into that parsing process as of now we are saying that the job of the parsing process is just this recognition part at the end it creates some data structure for us and in a later stage we will analyze that data structure. Semantics checks is that to find a type mismatch what is the right time; according to this diagram it is you are completely doing the recognition process building the parse tree and then moving to the type mismatch which might have been there in the first line itself right. So, your question is why do not I just do that immediately and that would save me some useless work because I could have come back and said this is your that.

Student: (Refer Time: 46:55) it is also.

Yeah, yeah.

Student: (Refer Time: 46:56).

It is an interesting question, it is not as simple as this. So, some elementary checks are performed along the way this is just a conceptual picture, but simple checks that are essentially of a local level those are ok to perform as you are building it remember there is a trade off involved this is one side of the argument. The other side is as you put more and more intelligence into the parsing process you are delaying the parsing there is a large file to be processed and an error is there in the late stages then you are necessarily doing a lot of useless work which leads to a an engineering solution of the kind that you do simple very simple checks during the parsing, but you do not do complex checks. So, and it is possible that at the end of the parsing there would still be some kind of errors that are not locally detectable you have to do a lot of traversal to find that there is a mistake of somewhere, but simple type errors you could also detect (Refer Time: 48:00).

Student: (Refer Time: 48:02) one is working the next.

These things could be arranged as independent processes that are working as soon as, but yeah there is a level of there is some argument for simplicity of the design also you do want to reserve most of the time for later analysis and optimization this step you might not want to spend too much time on right yeah.

Student: Is there any kind of optimization in the front end for example, let us say there is some dead code (Refer Time: 48:38).

Optimizations are almost never performed at this stage not because they do not make sense, but it is just that you may have to throw out everything just because there is a syntax error much later yeah. So, right it might be worth it, but then if everything is then maybe you perhaps you might save some time also in the process. But just in the interest of simplicity they do not do too much of occupation.

There are some things that are done to optimize this step the lexical analysis and the parsing step, what happens is when you have a syntax error, you could abort the whole process immediately but you can try to patch up that process or somehow your entity specification or declaration did not match. So, there was a syntax error in the entity declaration itself. You could abandon everything and just say that this is an error, but it is often considered useful from a users point of view to give you all the errors that are there in your program not one at a time you may if you have 200 hours you want to fix all of them together its irritating to run the compiler again and again to be informed that there is a semicolon that is missing and then you go back and said comma is missing so on.

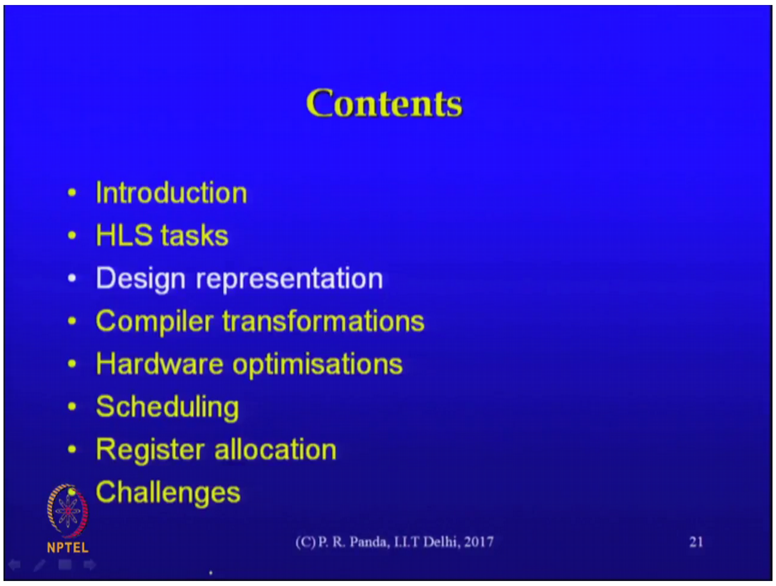
So, from a users point of view it is a nice thing to say I want all my errors to be reported at once, but you see that actually imposes some, it creates some very serious technical challenges. How do you optimizations, we talked about a while this was not a formal exposition on how parsing is done, you got the sense of how it is done right the rules are there if we try to match the rules and depending on what you have specified we pick the appropriate rule. This is a simplification, but a proper how do you decide the appropriate rule is an interesting question, but either way you see that you run into technical difficulties when the rule is not matching then what happens how do you progress the user wants all the errors to be determined when in fact, we have got completely off track because that the entity rule itself did not match right.

So, they do lot of smart things to recover from errors that are specified by a imperfect users like us. Often one can recover in a reasonable way like a semicolon is missing, you can patch it up assume that the semicolon was there and just go ahead even though this the statement rule did not match you assumed that it matched and then anyway proceed to the next statement and so on. That explains sometimes when you compile a program you get all kinds of errors that you have no clue about your reaction is I do not have all of these things in my program why is it reporting, but what happened is that it got confused because inappropriate patching up could not be performed and it went into some other rule where it was expecting something else and you are saying where are all those things in my program there are they are not there.

So, it is useful most of the time, but sometimes might throw you out. In C++ compiler sometimes you might have seen examples of all kinds of errors that you do not know where it came from, but that happens because of two things, one is this other is because you are using libraries from somewhere and it says that this operator is not overloaded for that object alright yeah.

Student: Sir, input (Refer Time: 52:17) destroys any standard HDL language is there any standard for this intermediate representation as well or.

(Refer Slide Time: 52:31)



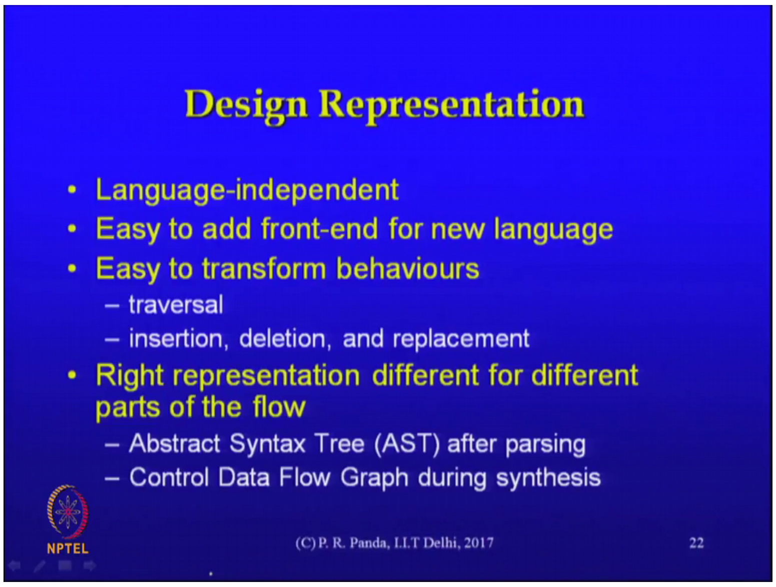
Contents

- Introduction
- HLS tasks
- Design representation
- Compiler transformations
- Hardware optimisations
- Scheduling
- Register allocation
- Challenges

NPTEL (C) P. R. Panda, I.I.T Delhi, 2017 21

Yes, yes, but that is the topic for the discussion. Yes, we do want that intermediate representation to be language independent.

(Refer Slide Time: 52:33)



Design Representation

- Language-independent
- Easy to add front-end for new language
- Easy to transform behaviours
  - traversal
  - insertion, deletion, and replacement
- Right representation different for different parts of the flow
  - Abstract Syntax Tree (AST) after parsing
  - Control Data Flow Graph during synthesis

NPTEL (C) P. R. Panda, I.I.T Delhi, 2017 22

Because languages may change, but those changes may be of a very superficial nature because it is just laying sort of entity you have module or something like that, but your synthesis mechanisms are not going to be changed just because the language changes.

So, usually you would design that intermediate representation in a way that you are shielded from changes in the language. You cannot be shielded hundred percent because of course, you know that if you add a new keyword to the language then of course, the parser has to change that keyword has to be accommodated hopefully with a very minor change the representation also has to change, but hopefully the changes will be minor. The design is such that it is intended to be generic and would stay through iterations of language or variations of language newer and newer versions of the language and so on. We try to make it independent as far as possible just so that our analysis and later steps are not influenced by change of language or changes of that nature.

Student: Syntax error and (Refer Time: 53:46) what kind of check is sematic checks.

Type mismatch is a semantic check you say a equal to b right, if I have a statement that says a equal to b plus c and so on right this is of some type and all of these guys are of different types. You know the type of that an expression may not be obvious to start with, this you know the type of you know the type of b c and so on at the end of this. So, some analysis would be needed to find what is the type of that entire expression because you may promote some types one is a an integer and the other is a floating point number the rules of the grammar have to explicitly permit this which they do, but it would also be that at the end of this you add an integer to a float what kind of expression do you get, what is the type of the resulting expression that this results in a float is a language specification we do not make that assumption its specified in a way that is reasonable, but you have to actually do all of this to find out what the type is for an expression and the a equals something. So, this might actually from the point of view of matching the grammar rule this statement might be ok right, but from a type point of view there might still be a mismatch that is an example of a semantic check. There is more to the language than just the grammar rule. So, those other things need to be verified also.

So, a number of other things would also fall in this category for example, re-declaration of a variable right. So, you have an int a, some other stage you also have an int a right both of these are valid statements by themselves, but the second declaration is in error because you had already declared that variable first. That is not obvious from the rules of the grammar because the grammar will correctly recognize both the statement and these statements there is no harm in the statements itself. But they are linked in a way that there is an error that is created because of this, so those kind of errors would fall in the semantic chain. They are do not necessarily performed sequentially like we are saying this kind of an error is easily detectable at the time that you get to the second statement get to this declaration you should be able to detect that in the current scope there is an a that is already visible it has already been declared. So, you could if you want to this kind of semantic check during the parsing.

So it is important that the design representation be almost language independent to the extents possible. There will be some features that are there in one language that are not there in the other language and so for those who do need the intermediate representation to be able to capture also. But it is to have that representation that you are using being somewhat of a superset of a particular language right. If in a particular language a feature is not there then it is alright you do not create elements of that nature when you create your representation.

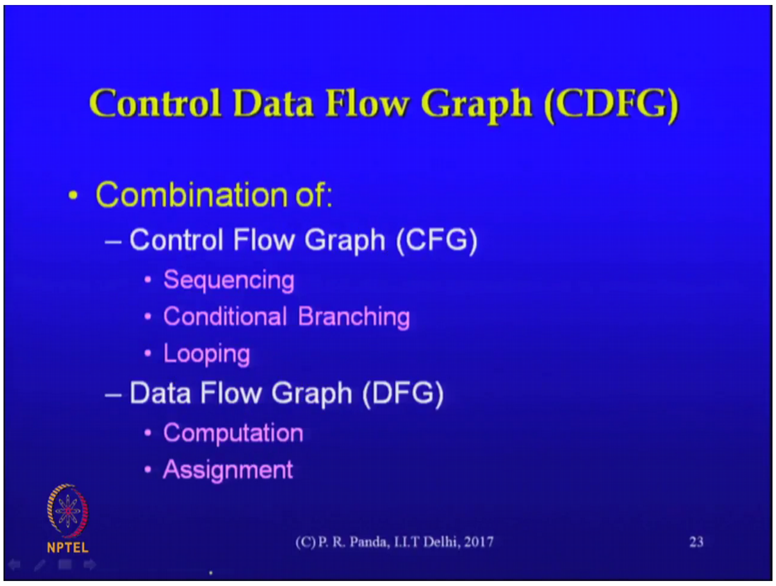
What else. So, it should be easy to add a new front end for a new language normally this analysis techniques synthesis techniques and so on, they might last longer than a particular language, new languages may involve they do not redesign everything from scratch tools from scratch when languages evolve because most of things that we are doing might still be the same. So, it should be of the kind that ideally I should have only a new front end when the language changes which means that the parser and all of that an those things would change, but the synthesis steps hopefully would not do not need to change those are elementary requirements.

Then what is expected within that IR itself, it should be relatively easy to do transformations. Why transformations are important? We will get to as soon as we talk about optimizations we perform as part of the synthesis process all kinds of transformations on the code transformations means you read it in the form of a tree structure at the end of the parsing, but we do not keep it in that way depending on what is it that we are doing we modify that representation in various ways.

Either way I should have easy ways to traverse the design I should have insertion, deletion, replacement those kind of features these are tree or graph kind of structures anyway. So, I should be able to traverse tree or a graph I should be able to replace some section of a graph with a new section that I have created that would represent an optimization of a transformation.

The right representation would be different in different parts of that overall flow. So, early on it would be what is called an abstract syntax tree that is the tree the parser creates, but that structure may or may not be best in later stages of the design of course, we will be talking about these representations in more detail. But the idea of a control data flow graph or some variation of this is something that is often used through large parts of the synthesis process.

(Refer Slide Time: 59:30)



Control Data Flow Graph (CDFG)

- Combination of:
  - Control Flow Graph (CFG)
    - Sequencing
    - Conditional Branching
    - Looping
  - Data Flow Graph (DFG)
    - Computation
    - Assignment

NPTEL

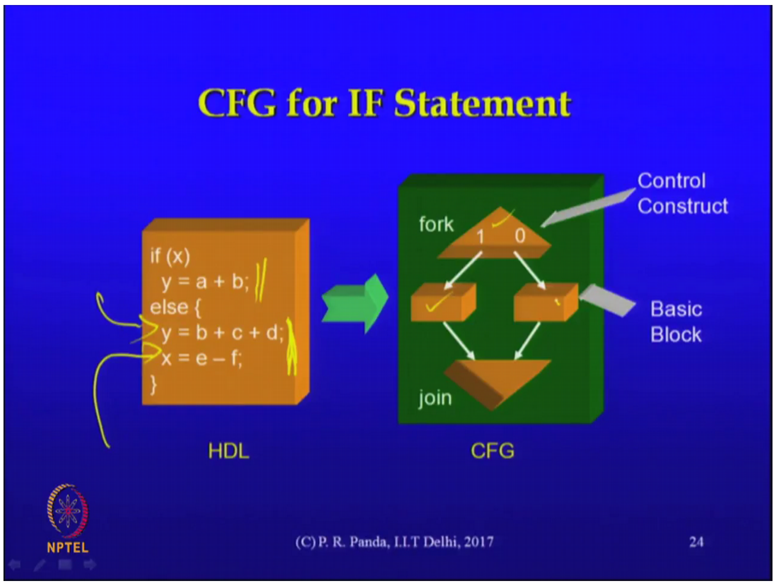
(C) P. R. Panda, I.I.T Delhi, 2017

23

Let us just try to understand what that control data flow graph looks like because in various algorithms later on we may be using implicitly this c d f g that the what we have written in the HDL is captured in the form of a c d f g so that other algorithms operate on the c d f g. They do not operate on the file that you have written they operate on this intermediate representation.

So, this is a combination of a two things, one is a control flow graph that is responsible for capturing sequencing conditional, branching and looping structures wherever control flow exists in our specification. Data flow graph is somewhat of an orthogonal structure it captures computation essentially. These are easier to illustrate with an example.

(Refer Slide Time: 60:24)



CFG for IF Statement

```
if (x)
  y = a + b;
else {
  y = b + c + d;
  x = e – f;
}
```

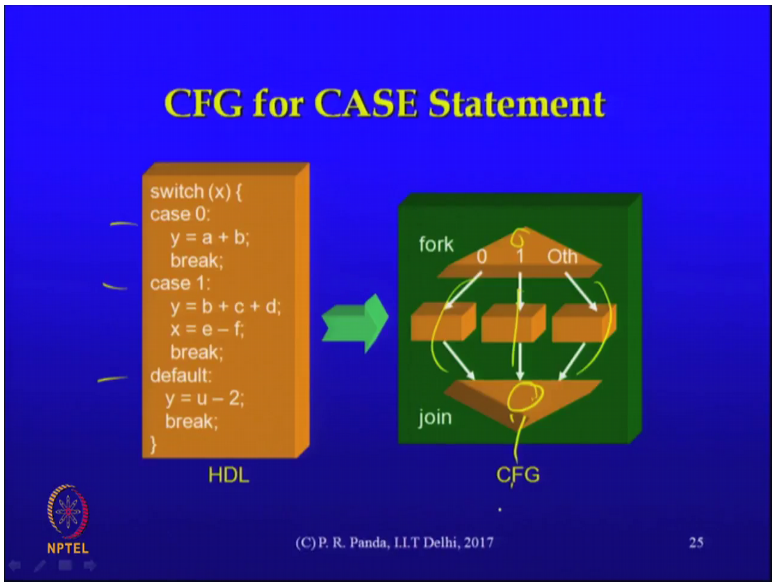
Let us say this is what my HDL was looking like and I did not respect a particular HDL here this looks more like c, but it does not matter. Remember this representation is supposed to be independent of the language. So, we can use various HDLs at the input.

Let us say I have a conditional structure here and that is getting captured in the form of a control construct that looks like this, this refers to a forking structures where I have two different branches this corresponds to the then part and this corresponds to the else part that is just a dummy node as of now it does not carry that information of what is happening in that branch it is just carrying the information that there is a branch there right. And this idea is called a basic block its terminology from a compiler you can think of it just as a sequence of statements with no branching within it its more formally defined separately, but it is like within the branch you have a sequence of statements there is only one flow of control within a basic block.

Other elements that go into the definition of a basic block is that there should be only one entry into that basic block and one exit from the basic block. How is that different from just a sequence of statements? A language may permit you to go to a particular statement from somewhere else when that happens you see that this definition is violated, if this is my sequence of statements constituting a basic block then I have multiple entries this is one entry that is the default entry the first statement by if you allow entry from somewhere else to the middle of that sequence of statements then that is no longer a basic block.

Why that is defined that way is that we want to be able to move a a basic block maybe from one place to the other or something like that and some properties are not retained if you allow jump from some part of the code to some other part. So, anyway that is what the basic block is for now it is the longest sequence of statements that has only one entry and one exit. So, this is one such basic block that is one such basic block. And so that is not captured in the c f g, the c f g is only capturing in a dummy node like this that there is a basic block. So, we have not yet completely captured in an intermediate representation that HDL, but at least the control part of it we have captured.

(Refer Slide Time: 63:05)

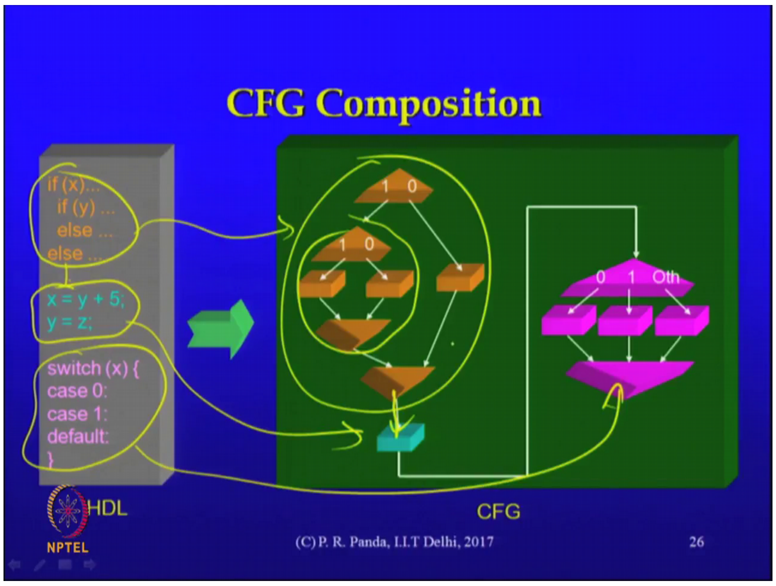


CFG for CASE Statement

```
switch (x) {
case 0:
   y = a + b;
   break;
case 1:
   y = b + c + d;
   x = e – f;
   break;
default:
   y = u – 2;
   break;
}
```

Generalizing into a multi way branch if you have a case statement with multiple entries then that is what that generalization is, there is a fork, but you have multiple branches in that way all of these come together in a corresponding joint node before we proceed with other elements of the control flow graph. This is just a sample you can think of other constructs if you have a loop then what that might translate to and so on.

(Refer Slide Time: 63:43)



CFG Composition

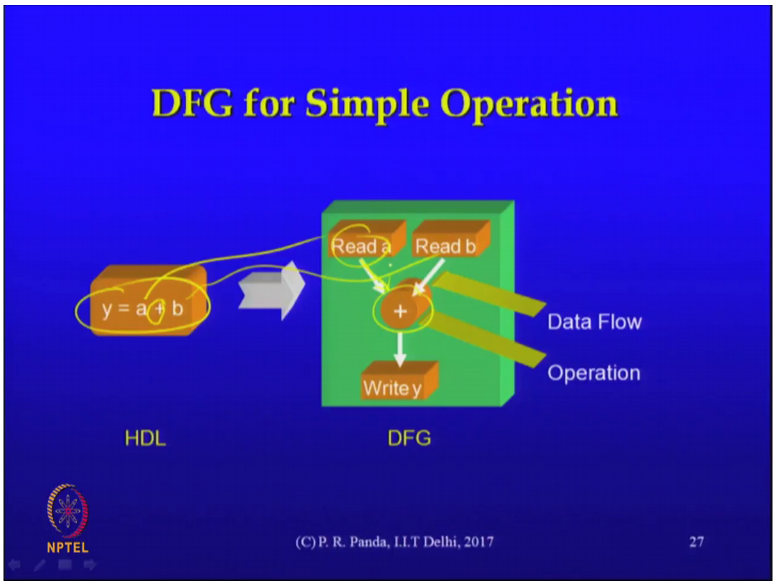
(C) P. R. Panda, I.I.T Delhi, 2017

26

These can be composed with each other in a natural way. So, first of all that can be hierarchy in the conditional structures that we write that could be translated to a hierarchical control flow graph in this space. So, the inner if statements corresponds to that part of the structure these are all graphs even though I have shown these using different colors and different forms and so on, they are all nodes in the graph and they will be distinguished from each other through properties of the node that properties that we annotate on to the so that is the hierarchical structure there can be sequences of these structures.

So, this is a basic block by itself, that is this part this corresponds to that part and that switch that comes next corresponds to this part and these are dependencies the sequentiality is maintained through the presence of edges like that. So, far we have not said anything about what we do with it this is only a representation, as of now this is what the user has written and we just have some intermediate graph based structure for capturing the specification. We are not doing any optimization, but as part of an optimization it is possible we restructure this graph in some way.

(Refer Slide Time: 65:11)



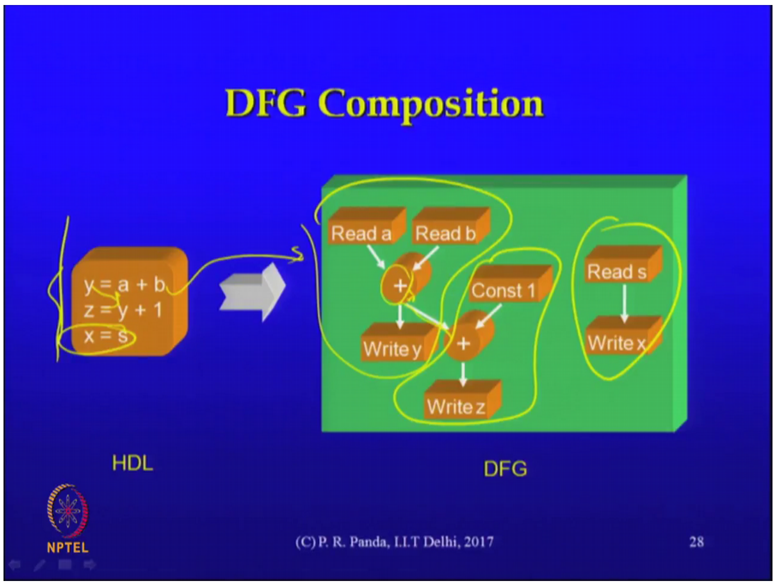
DFG for Simple Operation

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

27

Completing this I can have a data flow graph I had the statement that as of now we just had a dummy node, but the computation part of it we can capture in what is called a data flow graph. So, I have nodes corresponding to these operators. There may be other nodes corresponding to these operands depending on where they came from this is just the most general form you might not need all of these nodes to read. And remember in the general case maybe reading of an a corresponds to a memory operation or something like that so that node might actually need to be created, but I have nodes corresponding to both operands and operators output is going into y and these edges in the data flow graph represent dependencies between nodes right.

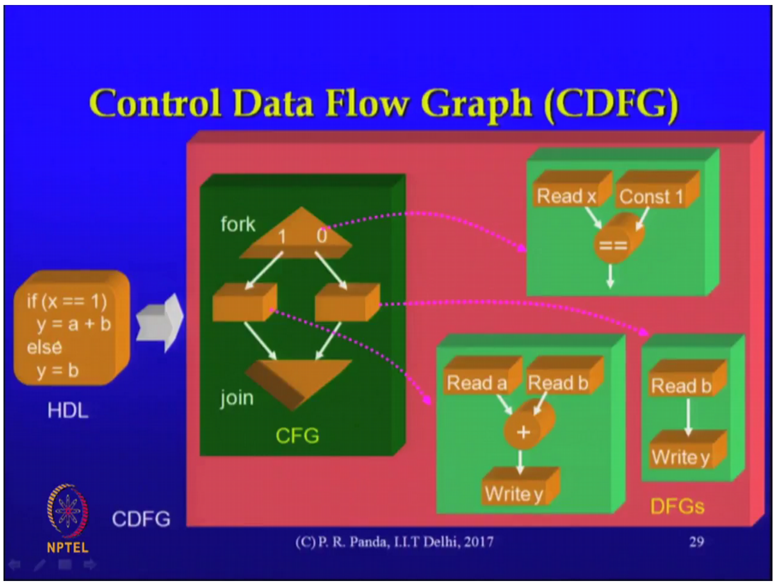
(Refer Slide Time: 66:06)


DFG Composition

If I have a sequence of statements like this, this part of it is the first statement right. I have z equals y plus 1 that is the second statement, but there is an edge from the first output to the second input so that operators output is an input to the second operator that dependency is represented by introducing an edge between the two nodes.

The entire basic block need not be a one connected data flow graph there may be other parts that are unconnected, but the whole thing together is a data flow graph. So, it corresponds to one sequence of statements that you have. So, a basic block in a code would translate to a data flow graph.

(Refer Slide Time: 66:55)



Control Data Flow Graph (CDFG)

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

29

And with that I can compose a control data flow graph out of it where the high level structure the control structure is captured in the form of a control flow graph, but what is missing from here is the computation part of it. What happens in each basic block? That is not captured. So, this basic block translates to a d f g that other basic block here translates to some other d f g and in fact, the way we have written this the fork also has a computation associated with it, so there is a different d f g. This is just by way of illustration. You might actually move that computation out into the prior code so that might actually become part of the previous basic block and that is where that d f g might be absorbed right.

But a control data flow graph is an overall structure where you have a hierarchy of graph structures the higher level is a control flow graph where you have captured control flow and the basic blocks within that the computation that is captured in the basic blocks is expanded out in the form of individual data flow graphs.

Let us stop here for today.

Student: Data is annotated as (Refer Time: 68:11).

Yeah, we just had the pictures here we did not label the edges, but there would be a label. So, I have over here I would put that label here on the edge yeah. Sometimes those labels are like we discussed the other day are explicitly declared variables, sometimes they are implicitly created by us inside the synthesis tool, but this still have a name and we annotate those names wherever we have information about the names.