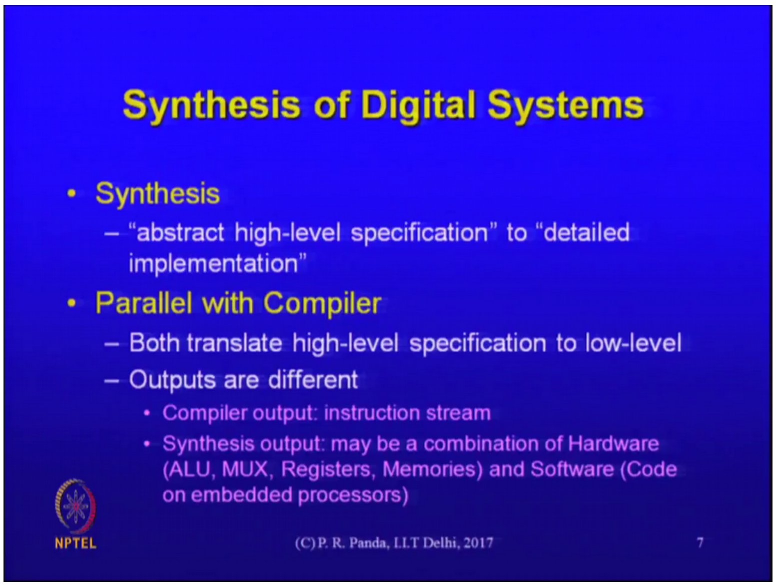
Synthesis of Digital Systems

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

Lecture – 01
Outline: What is Synthesis?

Let us start welcome to this course on synthesis of digital systems. Today s class we will be short we will go through an outline of the course. Just define the problem what is meant by synthesis of a digital system. And maybe we will just have some discussion about the prerequisites for the course.

(Refer Slide Time: 00:46)



Synthesis of Digital Systems

- Synthesis
  - "abstract high-level specification" to "detailed implementation"
- Parallel with Compiler
  - Both translate high-level specification to low-level
  - Outputs are different
    - Compiler output: instruction stream
    - Synthesis output: may be a combination of Hardware (ALU, MUX, Registers, Memories) and Software (Code on embedded processors)

NPTEL (C) P. R. Panda, I.I.T Delhi, 2017 7

Simple definition would be synthesis is the process of moving from an abstract high-level specification to a detailed implementation.

The context here is that of an electronic design, but also within electronics we are restricting ourselves to a digital subsystem. So, which means that on a chip that has any number of components, including handcrafted components, including analog components and so on, the synthesis of the kind that we will be studying here is applicable to those components that are digital in nature.

So, that is the subset that we are talking about, if it is to be captured in one sentence, it would be this that abstract high-level specification is what is captured in the in an hardware description language; that is what we will get to the reason we will spend some time on hardware description languages in the beginning is to make sure that we understand how to specify a design.

The specification is not the same as how to design the electronic system, but we are just capturing the behavior in some way without necessarily going into a lot of detail. The output of this synthesis process is supposed to be a detailed implementation. So, what is detailed again we will have to define here as we will see later on.

Student: (Refer Time: 02:20).

There are various levels of detail that are associated with the designing of an electronic system. Starting would be some very high-level description, but the detailed implementation for this context stop somewhere near gate level. Of course, there are other levels of abstraction more detailed than the gate level, but that is not the subject of this course.

But if I were to define it in one sentence it would be this, the process is called a synthesis there is a heavy automation that is involved. So, idea of course, is that you start with this high-level description. But then go through a series of processes and ultimately arrive at a gate level description. That somehow implements whatever was specified. Now that process is what we are calling synthesis, there is an obvious parallel with what happens in a compiler in the software world. What is the parallel? Both of them actually translate high level specification to low level, our c or c plus plus programs are the equivalent of high level specifications right.

If you look it is syntactically they look not very different from the hardware description languages. There are ways to specify interfaces, there are ways to specify data types there are ways to specify flow of control you have your if statements and loops and procedures and processes and so on. And so, at some high level of abstraction that these 2 are very similar. Therefore, there is quite a bit of analogy between what happens in the steps within a compiler and the steps that is synthesis tool goes through it is worth keeping in mind we will come back to this context, again and again and point out as we talk about synthesis.

What are some of the parallels with the compilation strategy, but also what are some of the differences. So, the front end is very similar both are sort of programming language inputs that are taken from the user. What happens early on in a compiler is also very similar to what happens early on in a synthesis tool, if you can think of that implementation of a synthesis strategy as a sequence of steps, where early on we are dealing with the hardware description language or some manipulations at high level, but later on we are talking about gates and smaller components.

Student: Yeah.

The outputs are very different. Even though the inputs are sort of similar. Abstraction levels of the inputs are similar, but the outputs are different. What are the outputs? In the compiler context, the output is what? Well, it is a sequence of machine instructions. You start off with something that is independent of the machine, right the programming language you is supposed to be independent of the machine.

But the compiler goes through a series of steps, in each step it somehow takes you closer to processor specific detail or machine specific detail. Even if you haven t taken a course in compilers, you should hopefully be familiar with the usage of a.

Student: (Refer Time: 05:39).

Of course, you would have invoked a compiler. So, you are familiar with what the output of the compiler is. It is an instruction string a sequence of instructions, that somehow implements the same thing that you have specified functionally the same thing that you have specified in your c or c plus plus or whatever the language.

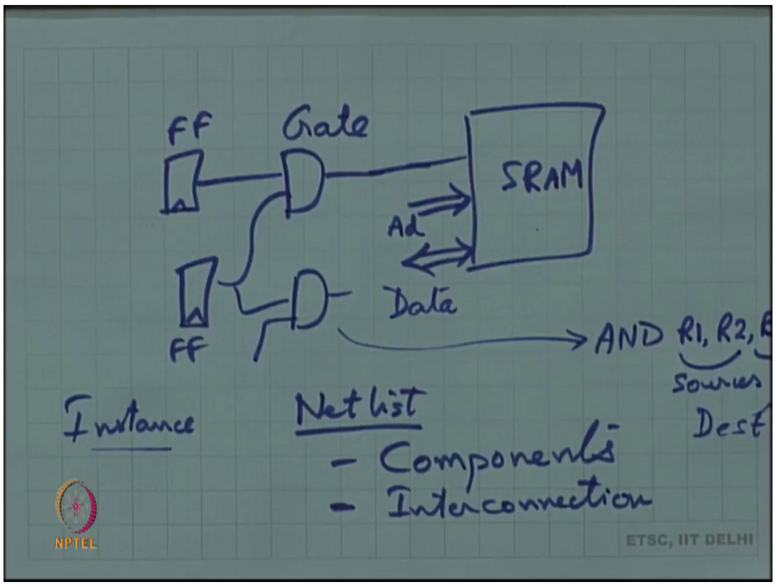
Synthesis tool is similar, but the output here is very different. Our objective here is to get an electronic system, get a digital system as the output. So, our output would consist of a net list of hardware components. What would they be these components would be? Like ALU s and MUX es and registers, memories, flip flops gates and so on. Nothing below gates, but the kind of synthesis that we will be talking about. Stop somewhere near gate level, but these are the components that are the output of the system.

Now, as you can see this is very different from the compilers output, right. This is a gate level.

Student: Netlist.

Netlist you instantiate some modules, whatever they are maybe. They are gates, maybe they are flip flops, maybe they are register files or ALU s or memories and so on. The instantiation of these components along with an interconnection of all of these components constitute a net list for us.

(Refer Slide Time: 07:09)



Student: (Refer Time: 07:08).

So, what those components are it is not very important maybe they are gates. Maybe they are flip flops. Maybe they are memories much bigger systems SRAM or some such memory sort of systems. As long as we understand the properties of the system well, we should be able to handle it as part of the synthesis process. So, if there are memories then you can think of other components to a memory what are the interfaces to a memory what do they look like to a; what is the functionality of a memory?

Student: Store data.

Store data and access data right. So, what would the external interface of a memory be?

Student: (Refer Time: 07:51).

Yes, at the very least, you need and an address bus. You also need some data bus that could be bi directional. Is it understood the main difference.

Student: Yeah.

Between a unidirectional and a bi directional, why did I draw these pictures this way? The address bus was an input, but the data bus is both an input and an output. Is it clear why it should be that way? Because on the data bus we want to both read and write, but on the address bus you only supply the address. And in addition to that there may be a few other control signals that is the interface of the memory.

So, actually even though it is a very complex as opposed to gate to which uses which is small. The memory is large, but in spite of it being large there are obvious ways to export it is functionality in a simple and meaningful way so that an automatic tool like a synthesis tool what we are talking about would be able to capture it is behavior, would be able to instantiate that as one of the components in the design.

So, it could instantiate an and gate, but it could also instantiate more complex components. As long as the interface is such that it is easily understood and we are able to make it part of our algorithms that are going to understand or the users input, and realize that it makes sense to have a memory of that configuration there. And include that memory in the output design.

So, the even though I said gate. It does not necessarily mean that output circuit is necessarily simple in any way. It can be the abstraction there again can be high or low. As long as we understand it we should be able to handle such a system. That is what I would call a netlist, yes. Let us so, it has 2 things one is a components instantiation of components means that it includes those components as part of the netlist.

But other than the components the what is the other element it is the interconnection. These are the 2 basic elements. So, you need some way of specifying that these components are present. And some way of specifying that this output of this component is connected to that input of that other component and so on. So, there is some usually it could be a language specification, these hardware description languages allow us to specify this kind of behavior; which is in addition to saying what the components are doing. Remember, as I drew this netlist here, I did not say anything about what is happening in this component, what is happening we did talk about the memory. But separately though that was not obvious in the diagram.

What it is each block is doing that was not obvious at least not yet. So, netlist is sort of a description of structure only. And that structure is actually very simply specified, it is specified by some inclusion of components. Remember, you can have and gate, but you could have any number of 2 input and gates right.

The same component type, you could have multiple instances off in the netlist. You may have 100 and gates both. So, both of these are flip flops. And so, it is essentially you are like picking up one element from a library, and you are including that element several times. Each is called a separate instance. And the process is essentially called instantiation. You may have one component. But so, if you have worked on a schematic capture kind of tool where you build a an electronic circuit by picking up components from a library and including them as and forming some simple circuit. This idea of instantiation is something that is that is fundamentally is useful is the difference between a type the equivalence in a programming languages a type is like a gate. And there is instantiations of gates are what equivalent of.

Student: Variables.

Variables, which you could have any number of variables of a particular type right. So, that is the sort of equivalence here. So, that is the output of a synthesis system. Ideally it is that, it could have the simple components, it could have complex components, but this is the space that we are talking about. So, we know the input and the output what they look like. But of course, synthesis means, I take the input, and I somehow go through some automated reasoning process, why automated it is just that that is essential for us to handle large designs.

You may have already done digital designs by hand. Specification is given a paragraph specification is given in English text. And you are able to understand that, and from there go through as mentally as series of steps. And the output may be something like this what we have drawn here right. So, synthesis refers to the process where we attempt to automate that line of reasoning as much as possible. That is what the topic is. Would there be something similar between an instruction stream and that netlist that we drew here? Differences are clear. Instruction stream looks nowhere like our Netlist. But still there are some similarities.

Student: (Refer Time: 13:45).

Yeah, actually flow of program is an interesting difference. Not necessarily similar similarity. When you write an instruction sequence, what is the assumption about the execution of that sequence.

Student: (Refer Time: 14:05).

If there is a sequential execution. So, the processor is going to pick up one instruction at a time and execute is the same the case in the hardware in this netlist that we draw.

Student: No.

What is the, what is the case here? Do we assume that so, I had 5 components here, what is the assumption about the order in which these 5 components would be evaluated or executed?

Student: They could run in parallel.

They could run in parallel? In fact, the fact that we have drawn at this way means that, it is by default parallel. All these components are executing in parallel. They can take different amounts of time. They do not all take the same amounts of time, but of course, they all can be all our millions of gates in the system can be active simultaneously; that is the assumption here. So, that is one important difference though.

But what are the similarities, this is a similarity between the gate here, and an instruction in a processor.

Student: Both are not physical. Neither of them is physical. Both are just descriptions.

This gate here is just a diagram, as of now it is just a diagram.

Student: (Refer Time: 15:14).

By not physical you are saying that it is just some format as a sequence of bits it is to be interpreted in whatever way, that is what you are saying. That is true. Both are some in some ways abstract description. This one does translate to something physical later on. Even though as of now it is still a picture that we are just seeing on the board. Later on, this does translate into actually picking up an element from the from the library placing that element in a chip. So, of course, at some point it does become something concrete. What can we say about that instruction that is being executed, yeah.

Student: Like both have direct specific functions input outputs.

Both have functions is actually both are simple one instruction by itself is a relatively simple is doing something very simple. It is somewhat the equivalent of one gate here.

Student: (Refer Time: 16:07).

If we are saying this is an nand gate. It does something very simple, it is external interface is clear. There are 2 input bits, and in response to anything changing on those input bits, there is an output that is changing. That is the functional description of an and gate. But a functional description of an instruction might not be very different. Let us say you take an and instruction that way it would be the equivalent of what is happening here. To an and instead of what is there in an and instruction. Are you familiar with the format of an instruction? Yeah.

So, and is of course, by it is just doing it is taking maybe 2 bits, but maybe in the instruction case it may be too big vectors or something like that it is taking. Idea is not very different, but what is the external interface of that, what is the input what is the output of that one instruction?

Student: it can change (Refer Time: 16:58).

Yes, as part of that instruction format you specify where the inputs are coming from, and where the output is going right. So, if they are coming from registers, if they are if the output is going to a register, then you would essentially other than saying that this is an and instruction which tells the processor what the functionality is; you would say that you get the fetch you fetch your data from these locations, and perform the and operation and the output output is available to us, but it needs to be stored somewhere.

So, all of these we would specify as part of that instruction.

Student: So, what is neutral different in the instruction (Refer Time: 17:35)?

Yes, this is is not a perfect equivalent. So, the functional equivalence is there. Now as regards how we actually implement that functionality there is a lot of leeway, that is that is available to us, right. There is a memory hierarchy instructions are stored in the memory somewhere before you can execute the instruction you have to fetch it from an instruction memory. You have to go through a decoding process and so on. When you instantiate an and gate in your design you are performing some of those steps explicitly. There is no fetching of an instruction.

So, some of those the implementation details of the processor of course, is not there here. So, the important differences are there, but functional functionality wise they are actually similar. They are not very, very different from each other. You have instructions that could also be complex instructions. That would be the equivalence of some of our modules instantiated in the netlist being complex.

Student: performance wise they will be better things.

Performance wise

Student: Yes.

Would an explicit hardware circuit be better than writing up an equivalent program, and executing it on a processor? It is a very interesting question. Not necessarily answerable right now.

Student: Right.

But answer is not obvious. It seems.

Student: (Refer Time: 19:04).

That is some overhead that is involved. What is that overhead? The real thing you want to do is just an and, but what kind of overhead are we going through. Even before we can get to that functionality. There are lots of other things that come in from a point of view of course, optimizing the execution of a larger program.

But if you look at one instruction at a time, then it seems as though you are going through a lot of other additional work that a hardware implementation might not have to go through. What is that additional work you are fetching that instruction from the memory, right. That seems to be a terrible large terribly large amount of additional work to be done. Whereas, you actually want to just perform an and the.

So, if you take one instruction at a time it seems as though what we what is happening in the processor is an overkill in many ways. But the equivalence here is if you need an and gate you just instantiate an and gate, and that is it seems as though for simple functionality there is.

Student: There is.

Too much extra that you are doing if it is a software implementation on a processor. But when it becomes large though, if the design becomes large, then the answer might not be obvious. It is a very interesting question finally, what you care about is the standard metrics like performance like area power and so on. Does not matter whether it is a hardware implementation or a software implementation, but it is not completely obvious. If your system is complex enough, then should you go for a hardware implementation or for a software implementation. This particular question and some of the implications of that these are covered in other courses, we will not be able to go through it all of it in this course.

But the point at this motivational stage is to point out just that you have some specification. And you have alternatives with respect to implementation. You need not build any hardware, you can just take a processor existing processor, and you just write a high-level program you compile into that processor and that is it. So, essentially you build no hardware, that to there might be a case for doing it, but the other extreme is you have an explicit hardware that is doing exactly just that much and nothing else it is likely to be more efficient in some ways.

It is likely to be less efficient in some other ways we have to define first what is efficiency what is it that we want to optimize here. But much of that we will talk about during the course, because those are the care abouts of a synthesis procedure like, forget about the existence of the software alternative a compiler alternative. But even as you start from one specification and sdf specification, there would be many different implementations all of which are functionally correct translations of that high-level specification.

But they may not be equivalent with respect to our parameters that we want to we want to optimize like delay like area power and so on. So, there is a need to choose between the large number of candidate solutions. So, we will be talking about this trade off that exists all the time, but it is important to note that even at the very beginning there is an a tradeoff about whether I should build custom hardware for my specification or not. Answer is not obvious it does depend on several other things it depends on what other modules are doing, and whether you already have parts of that specification implemented and you are just looking for a small addition.

How critical it is from a performance point of view, it may be that some of those blocks are. So, critical that the processor that is available to you is just not good enough just not fast enough. And so, some of those care abouts affect the decision, not only later of how to do synthesis, but also in the beginning of whether to do synthesis at all. Whether to exercise this option of a hardware path, because you know that for a large number of possible specifications, you can deal with that you can come up with an implementation without designing any additional hardware at all. You can take up off the shelf processor components and implement your design if the cost equations are maybe that is the better solution.

Remember, designing hardware takes time it is expensive. Why is it expensive? Of course, designers need to be paid. So, it is expensive the, you will go through some of the synthesis steps and appreciate perhaps that a lot of tradeoffs that are involved the whole thing requires a lot of time there is an issue of convergence. The synthesis tool gives you an output, but that is not the end of it the gate level netlist is not the end of the chip design process.

Student: Yes.

We are still somewhere in the middle, then I have to take that and generate layout which is essentially a geometry. Because finally, I need to fabricate the chip. So, for fabricating the chip I need the geometry which is that is what is called a layout. So, that is another step that we will not get to talk to too much about in in this course, but this is not the end our netlist is not the end of the process.

It is the end of what we will be covering here. But that is taken as an input to what may be called a physical synthesis tool or a layout synthesis tool. That takes that netlist as an input and comes up with the geometric component is important you have to decide there, where you know considering it is in like a 2-dimensional structure the chip is a 2-

dimensional structure. Where should I place this flip flop where should I place this and gate and memories and so on.

Again, the answer is not obvious, right. You would like to place everything as close as possible, why? The wire delays would be minimal if you place them close, but then the kind of interconnection is such that everything cannot be next to each other. So, there are important tradeoffs that a physical synthesis tool also has to go through to decide. So, that itself should actually alert you, that there are some uncertainties in this process.

There is timing that may be involved, we will get to the details study later on, but the synthesis tool assumes a certain timing. Or the timing is input you provide the timing as a constraint to the synthesis tool. The synthesis tool gives you a design that is supposed to meet those constraints otherwise it is not a valid design. But it may happen that the physical synthesis tool later on, did not actually successfully managed to find a geometry that meets those timings.

So, there is some uncertainty there because of which this process may involve some iteration. You start off with the synthesis you go through, all the other steps, but at a later stage when you do timing analysis you might actually realize that we made the wrong assumptions is of course, important from a product point of view you would like to.

So, a specification comes you would like to implement that product in some way. You would like to implement it as fast as possible you need to get to the market as fast as possible. So, you do need to evaluate this trade off if an existing solution is already there, like if a processor-based solution do you still need a synthesized hardware to do the same thing maybe you do maybe you do not. So, they are usually there would be some reason from a cost point of view maybe, the software solution is, but then you have other parameters like there is there is a performance constraint that is imposed on the system.

And the software solution might not be good enough. Even though it is from a functionality point of view, maybe it does not meet the timing requirements. So, when such or power requirements, or whatever the various constraints might be. When that happens you have to explore other alternatives. And of course, building explicit hardware through a synthesis process becomes a requirement. So, over all exploration process is very interesting all of it is not necessarily part of the discourse contents.

But one does need to go through to this all of that to decide which components should be synthesized. If you are building a new system on a chip. Many of those components we might just pick from here. And there maybe if we take some processors from here, and there we may take a memory component from somewhere that somebody elses design. So, all of it will not be designed from scratch. So, but there will be some components that we should be able to argue that these components merit a closer look and the merit hardware implementation.

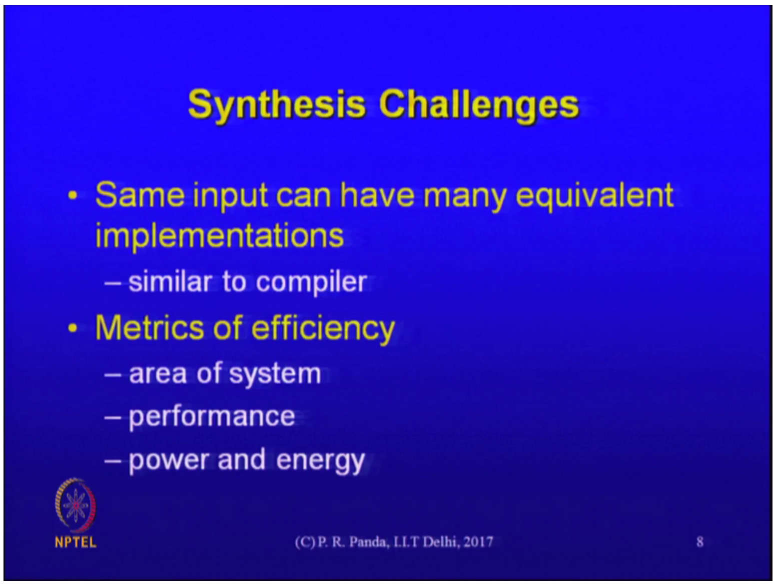
So, the synthesis itself is a term that is not although the context was hardware to begin with. You can apply it at a little higher levels of abstraction also. You can call the overall soc design and a system synthesis problem, in which the in which the specification is given to us. And it is up to us to realize an efficient implementation, but that efficient implementation does not consist only of gates or flip flops or something like that. We could insensate a processor, then we could instantiate a memory there all of these that is still technically part of a synthesis process.

You could expand the definition to include some to more complex components like that s ram component that we included here. This brings us into the system space, we are here no longer restricted to the gate level of abstraction, even though from an understanding point of view it is useful to think of it that way. But through the course we will try to touch various levels of abstraction starting from gate level, but also higher levels of abstraction registers, MUX es ALU s. These are a little more complex than gates, as these are the sort of the next level of abstraction.

But memories and queues and processors and so on, could technically also fit within the overall context of a specification that is given to us and we need to synthesize a design for it. So, in fact, as might happen a lot of the complex chips that are there today have a significant amount of software component in there. There are embedded processors large number of embedded processors might be there. So, that the system design context already has the software components in it.

But there may be some other components for which we need to build custom hardware. Custom hardware means, explicit hardware that is capable of performing only that function, and nothing else.

(Refer Slide Time: 30:16)



Synthesis Challenges

- Same input can have many equivalent implementations
  - similar to compiler
- Metrics of efficiency
  - area of system
  - performance
  - power and energy

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

So, this just to specify the range of possible implementation choices that a synthesis tool would have or the similarity between the instruction and the netlist we took a gate and said that it is actually a very simple function. The equivalent instruction to that gate may be something like and register 1, register 2, register that would be functionally an equivalent instruction.

These 2 may be the sources, and this may be a destination register. So, addresses a register addresses or something like that. They are they are part of that instruction. That is the part that is similar with respect to anything every single instruction would have some at least the simpler instructions would have some equivalent gate that is similar with respect to the level of abstraction with respect to the functionality. Even a complex instruction you can think of as a more complex digital system that actually implements that complex instruction. So, the that is the context in which this is similar. That similarity is important to note.

Because remember, we start off from something that is similar. We start off with a text description hardware whether it is HDL or programming language. Both of these are similar in the way they look. So, I should be able to look at operations that you have used in your programming language or in your HDL, and realize that I could implement that in terms of these hardware components. This these compiler things in terms of taking those high-level operations that you have, and implementing it in terms of instructions.

Synthesis tool would ask the same question. Is there something in my library using which I can implement that simple specific notions? That a manner in which these 2 are similar.

Because of this similarity, the strategies would be similar. In the way the compiler works and takes it is instruction related decisions, and the way the synthesis tool works in selecting for example, elements from libraries. So, of course, we will talk about more of that later. And we will point out some of the very interesting parallels, between the way a compiler works and the way the synthesis tool works. The compiler part of it is meant for just reference. You should be able to understand it, even if you haven t formally taken a course in what are the challenges.

Student: Yeah.

Of course, the main challenge is that the given specification if I have to realize it in terms of a netlist. It is not as if there is only one solution, if that was the case then it would just be a simple translation, there would be no need for any particular sophistication in the strategies. But the point is that you can have many different implementations that are functionally still equivalent, but they are not equivalent with respect to many other parameters including speed, including area and current power and these things.

Often what happens is there is a trade- off to be exercised. The tradeoff is of the kind that you are you are comparing 2 designs. One is a smaller design. So, area wise one design is more efficient, but the other one that is more complex might be a faster design. So, it is not obvious which one is better I have to look at more look a little bit more with respect to what the constraints are and so on, which is higher priority.

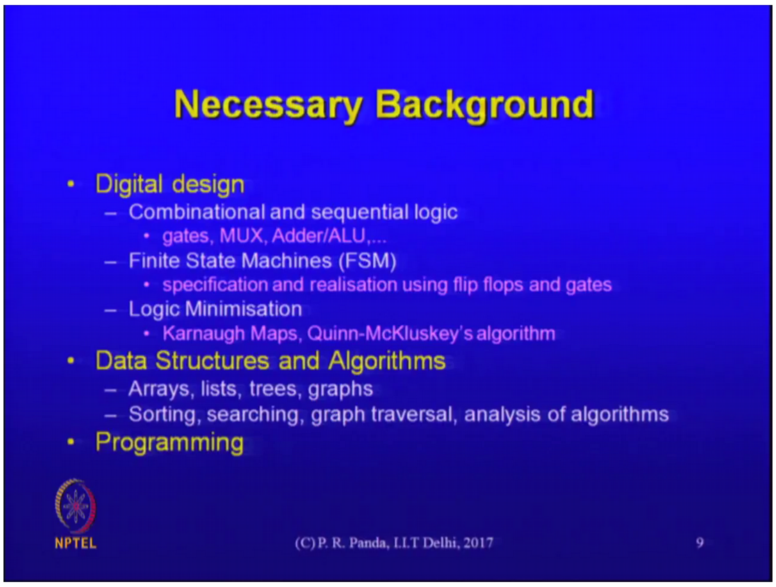
So, at some external information is needed for us to resolve between the many candidate solutions that are possible and choosing what is the best or maybe in the in the worst case we can actually present a set of solutions to a designer who can manually make the decision finally.

Even if that final selection is manual, it makes sense to go through a detailed synthesis process like this. Because through that at least we can analyze, and out of the 100s or thousands of possibilities, we can narrow the solution space down to maybe 2 3 positively candidates.

Student: Yes.

Which may be finally, manually chosen. That could also be a realistic design scenario. So, hopefully the motivation is cleared regarding what is involved in the synthesis process. Our metrics of efficiency we already talked about. So, that means, there are a lot of other metrics, but the standard ones are area performance and energy and so on.

(Refer Slide Time: 34:59)



Necessary Background

- Digital design
  - Combinational and sequential logic
    - gates, MUX, Adder/ALU,...
  - Finite State Machines (FSM)
    - specification and realisation using flip flops and gates
  - Logic Minimisation
    - Karnaugh Maps, Quinn-McKluskey's algorithm
- Data Structures and Algorithms
  - Arrays, lists, trees, graphs
  - Sorting, searching, graph traversal, analysis of algorithms
- Programming

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

Just a quick word about the necessary background for a course like this, it is expected that you have taken some digital logic or digital design kind of course. Earlier so, from there, you should understand the difference between combinational and sequential logic right gates and MUX es adders flip flops and so on.

The idea of a finite state machine, you have seen those fsm diagrams that you have a bunch of states, and a bunch of transitions between states together constituting a finite state machine, is that familiar? The it is very essential that this should be familiar. We will not be spending more time on that, but they show up in the context of a synthesis tool in ways that we will get to, but that is expected. But the elementary ideas on logic minimization using k maps, and the tabular algorithm of quin McChesney of all these also familiar.

From a computer science point of view, there are other fundamentals that it is good if you have picked up a little bit of at least. These are data structures and algorithms. So, what kind of data. So, we will talk about arrays lists which hopefully should be familiar to everybody, but even things like trees and graphs particularly. Even though the context seems to be a hardware context our output here is hardware.

But in the process of getting there, we actually use a lot of these computer science concepts of data structures, what is the right data structure to use that I should be able to analyze. I should be able to write an algorithm for this is an algorithmic process right finally, we start off from a HDL specification.

Student: And.

A couple of other things make sense for you to review. Analysis of algorithms at least a basic analysis of algorithm it is a good idea if you can go through yourself. In fact, that to prescribe book by Jamie Callie. It has one chapter in which a lot of this data structure related. Particularly data structures that are relevant to synthesis and algorithms that are relevant to synthesis that is covered there in a preliminary chapter. You could read that, but and if there are enough other sources.

That it is good as we study the synthesis process that we have an idea of analyzing and an algorithm for it is efficiency given 2 algorithms. That do the same thing, I should be able to say which one is better. You should be able to analyze that algorithm for it is complexity. That is why when we say which is more efficient, there is a formal notion of a complexity of an algorithm.

Yeah, go through example analysis that you can find in a book. The kind of algorithms that we will be studying here are not terribly deep. Nevertheless, it is good to have some reasonable background about the analysis of algorithms. So, that you can appreciate particularly if 2 choices are there 2 different ways are there of doing this of solving the same problem. You yourself should be able to analyze both of those choices without necessarily writing the program this is only an analysis of the algorithm.

See if there are 10 choices, then well you may not have the option of implementing all those 10 choices, all those 10 algorithms. And determining which is best therefore, it is essential that you have you pick up some background about analysis of algorithms, just the basics. So, that we can make some of those arguments about efficiency as we discuss synthesis strategies.

So, the level of expertise we are assuming here is not particularly high. But at least a familiarity should be there of all these 4 data structures arrays lists, but trees and graphs what kind of structures, these are they are characterized data structures are characterized by 2 things. One is how the data is stored. If it is an array then you know how it is stored, right. And if it is a list then you know how it is stored it is assumed that at least that much is already known, and what the differences are between an array storage and a list-based storage, right. What are the differences in the way we store the data?

Student: (Refer Time: 39:15).

Right.

Student: (Refer Time: 39:16).

Did I stored contiguously in an array list? You are not necessarily limited to storing they can be all over the memory. Why should we exercise? It seems the second one is more complex than the first one. So, by itself there is no answer. See this is one way of storing it and the other one is a different way of storing it. So, data structure is characterized by 2 things. One is the storage mechanism, but the other is a set of operations that you are performing on that data section. It is meaningless to talk about is a list good or bad, without defining what is it that you really want to do with it, right.

So, the choice of one or the other would be of course, related to what operation you want to perform ultimate. It is not obvious if one was better than the other always, then we would not be talking about the second data structure. So, that is the reason we are introducing trees or graphs or other data structures is that somehow there are contexts in which the array is not good enough linked list is also not good enough therefore, some of the other structures are needed.

We will get to them, and use them heavily in the course, but they are not complex just like a list is very easily defined. And array is very easily defined. These are also very easily defined. But the definition is only one part of it, other is what perform what operation do you want to perform on that data structure. So, that much background it is good if you can pick up on your own. There is enough time for that because we will get to these are part of synthesis algorithms and strategies, but we get to that after the initial discussion of the HDL.

Student: Ok.

So, of course, it is expected that you have a programming background does not matter what language. And these are the necessary elements that are needed for this course prerequisite. If you can say that as you can see, it is not terribly complex the requirement we will build whatever is involved from scratch. It is just that little bit of well the digital design part should only be a refresh if at all you have forgotten, go and take a look. But that concludes what discussion I had planned for today. We will follow up in the next class with first and overall specification of the context, which we in an informal way we already talked about they are being many different steps that are involved in a chip design process.

But before zooming in on synthesis, we will first try to define the overall context a little bit. So, in the that is the subject for the next class.

Thank you.