VLSI Design Flow: RTL to GDS

Dr. Sneh Saurabh
Department of Electronics and Communication Engineering
IIIT-Delhi

Lecture 6
Overview of VLSI Design Flow: III

Hello everybody. Welcome to the course VLSI Design Flow: RTL to GDS. This is the fifth lecture. In this lecture, we will be continuing with the overview of VLSI design flow. The topic that we will be covering in this lecture is RTL to GDS implementation. In particular, we will be looking into logic synthesis.

In the earlier lectures, we discussed that we can broadly divide the entire flow from taking the idea up to the final chip into three major parts. The first part is pre-RTL flow or system-level design. The second part is the RTL to GDS flow that we will be looking into today. The third part is related to fabrication, and we call it post-GDS flow.

In the previous lectures, we looked into the pre-RTL flow, and we discussed that at the end of the pre-RTL flow, we get an RTL either through behavioral synthesis or reusing existing RTL, or we may be doing manual coding. At the end of the system-level design, we will get an RTL for our design. Now, we want to take that RTL up to the GDS or up to the final layout, and that part of the flow is known as the RTL to GDS flow. Now, RTL to GDS flow is a very complicated design process, and we can divide it broadly into two categories. The first one is logic synthesis, and the second one is physical design.

So, the logic synthesis deals with taking an RTL and generating a netlist out of it. Netlist is the interconnection of logic gates. And then the second part, the physical design is basically taking that logical netlist and making a layout of it. So, in today's lecture, we will be looking into the logic synthesis part, and in the next lecture, we will be looking into the physical design. So, let us first understand what logic synthesis is.

So, logic synthesis is the process by which an RTL is converted to an equivalent circuit in terms of interconnections of logic gates. And these logic gates are picked from the technology libraries that we give to the tool. So, let us look into the framework of logic synthesis or what are the inputs and outputs of logic synthesis. So, what are the inputs or the major inputs that we give to the logic synthesis tool? The first one is the design or the RTL that we want to implement. This design can either be in the form of Verilog or VHDL, which are hardware description languages, or it can be a combination of Verilog or VHDL format.

The second major input is the library. The library contains the information of the standard cells that we will be using in our design. Typically, this library is in Liberty format, or the file that we use for the library has an extension of .lib. The third input or third major input that we give to the logic synthesis tool is the constraints file or constraints.

Constraints contain the design goals, for example, the expected timing behavior in terms of maximum operable frequency and the environment, meaning that what kinds of signals come into our design, what are the characteristics of those incoming signals in terms of their rise time, fall time and so on and what we expect of the signals that will be generated by a given circuit or the final netlist. So, those are defined in a file or a group of files, which are known as constraints files, and typically, we define the constraints in a format known as synopsis design constraint or SDC file. So, typically, the extension of these files is .sdc. So, taking these three kinds of inputs, the logic synthesis tool generates a netlist.

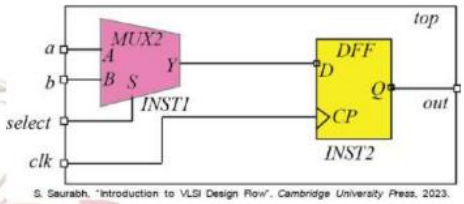
Now, what is a netlist? Netlist is basically an interconnection of logic gates. So, from the last lecture, we can remember that the RTL is basically the description of a design in terms of how signals move from one register to another register inside our circuit. Now, netlist is basically an interconnection of logic gates that implements the same functionality in terms of gates that are given or that are present in the library files. So, functionally, the RTL that we give and the netlist are both equivalent. So, functionally, these two must be equivalent.

One is a description of the design in terms of how the signals move between registers, and the netlist is basically an interconnection of logic gates. The logic gates may be combinational logic gates, flip flops, latches, and complicated gates. So, the netlist that is finally generated by the synthesis tool can be represented in terms of Verilog constructs, or we can also see it in a schematic if the design is small. So, the interconnection of logic gates can be represented in various ways. Typically, the Verilog constructs are used, and finally, the netlist that is generated that will also be in the form of a .v or Verilog file.

Now, let us look into what is an RTL and what it is converted to or what is its equivalent netlist. So, in this slide, I am showing an RTL, an example of an RTL, or a modeling of a circuit in terms of Verilog. So, we have not discussed the constructs of the Verilog language. So, if you are not able to follow all of them, it is okay because we will be covering the constructs of Verilog in later lectures in more detail.

Logic Synthesis: Illustration

```
module top(a, b, clk, select, out);

    input a, b, clk, select;
    output out;
    reg out;
    wire y;

    assign y = (select)? b : a;

    always @(posedge clk)
        begin
            out <= y;
        end

endmodule
```



S. Saurabh, "Introduction to VLSI Design Flow", Cambridge University Press, 2023.

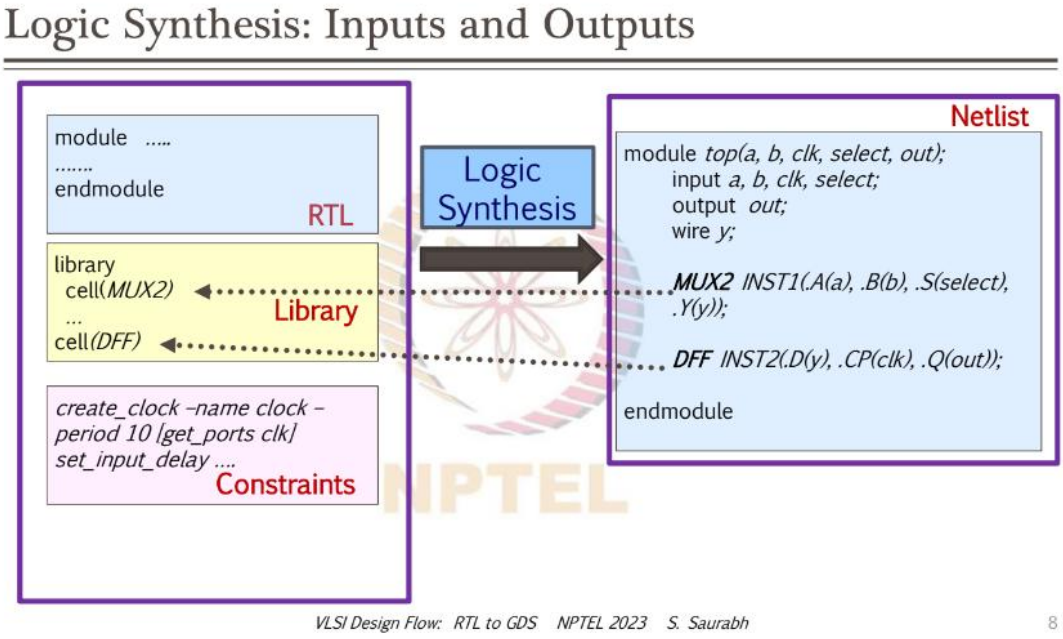
```
module top(a, b, clk, select, out);
    input a, b, clk, select;
    output out;
    wire y;

    MUX2 INST1(.A(a), .B(b), .S(select), .Y(y));
    DFF INST2(.D(y), .CP(clk), .Q(out));

endmodule
```


But a few points can easily be noticed from this Verilog description of a design. The design contains input ports a, b, clk, and select, and there is an output port that is named out. There is a wire y, which gets a value of b if the select is one else, it gets a value of a. So, this construct is describing a multiplexer. And then there is a code which is saying always @(posedge clk), out is assigned a value of y. So, this is a kind of description of a flip-flop.

So, given this description to the synthesis tool, it will translate it into a netlist, and the netlist will look something like the schematic shown in the slide. For the first construct, it will generate a kind of multiplexer, and for the second construct, it will generate a flip-flop and connect them as per the connections given in the Verilog file for the design. Then, the same netlist can be represented in terms of the first one is a schematic representation, and the second is a netlist represented using Verilog constructs. So, the synthesis tool can generate either of them based on what option we give to the tool. So, both are basically representing the same thing. For example, here, we have a multiplexer in the schematic and in the Verilog netlist file. Then we have a flip-flop in the schematic, and we will have a flip-flop in the Verilog file also.

So, both these schematics and the Verilog or netlist in terms of Verilog represent the same thing. Now, let us look into the result of synthesis from the input-output perspective. What is the output generated by the tool or the netlist generated by the tool, and how it was impacted by the inputs that we give to it? So, suppose we have given the input RTL, the RTL we have already seen in the previous slide, and then another input that we need

to give is the library. So, the library must contain some cells which will be used in the netlist. So, for example, the netlist that was generated in this case had this multiplexer and a D flip-flop, and from where do this multiplexer and D flip-flop come? These were basically instantiated, or they were picked from these libraries and put inside our netlist by the synthesis tool.



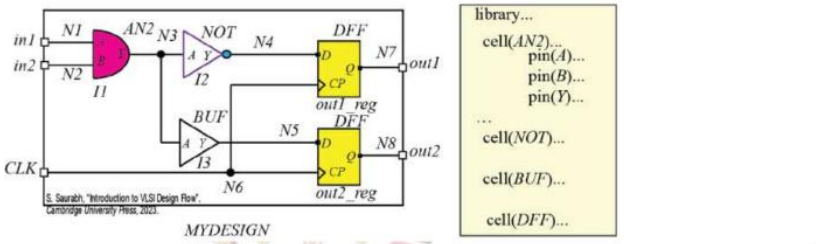
So, the synthesis tool basically picks cells from the given library based on some optimization criteria. And what optimization criteria to use is defined by these constraints. For example, if we give very relaxed timing constraints, then the tool will probably try to minimize the area, and in that case, if there is more than one multiplexer in our library, then probably the logic synthesis tool will pick the multiplexer that has the minimum area because the timing is not that much critical So, the performance can be traded off, and area can be minimized in that case. So, the constraints file basically defines our design goals or conveys our design goals to the logic synthesis tool. Now, the translation of the logic of the RTL to the corresponding netlist is not that difficult if the construct of the language, that is, the Verilog language in this case, is well understood by the tool.

So, translating a Verilog construct or the hardware description language construct to a corresponding gate or logic circuit is not that difficult. The difficult part or the challenging part in the logic synthesis is basically how to do the optimization. What we mean by this optimization is how to pick the cells from the libraries and how to connect them so that the function of this RTL or functionality in terms of logic and of this netlist is equivalent. While ensuring the functional equivalence, how can it minimize the cost in

terms of PPA? So, if we want to minimize the area or the power or maximize the performance, based on it, the synthesis tool needs to pick the cells from the library appropriately, and this is a challenging task. So, we will be looking at this challenge in more detail in the later part of this course. Understanding the challenge in logic synthesis will help us understand many other things or link together other tasks with logic synthesis task.

Now, before going further, it is important to be familiar with some terminologies that are related to a design or netlist or design that is in the abstraction level of the netlist. Why do we want to know these terminologies, or why should we be familiar with these terminologies? The first reason is that in the later part of this course or in the subsequent lectures, when we discuss concepts, we will be using these terminologies very heavily, and therefore, if you are not familiar with these terms, then it will be difficult to follow the subsequent lectures. The second reason is that the tools that we use in the VLSI design flow also follow the same terminologies. So, if we want to give inputs to the tool, we have to give inputs in terms of these terms or these definitions that I will be describing in the subsequent slides. So, the first definition is what is a design. So, suppose we have this netlist or a design that is represented in terms of a netlist, and this is the corresponding library. So, this is the library, and this is the netlist that is generated by the tool, or we have got it from somewhere.

Netlist Terminologies: Design and Ports



Design: Top level entity that represents the circuit. Example: *MYDESIGN*

Ports: The interfaces of the Design through which it communicates with the external world. Example: *in1, in2, CLK, out1, out2*

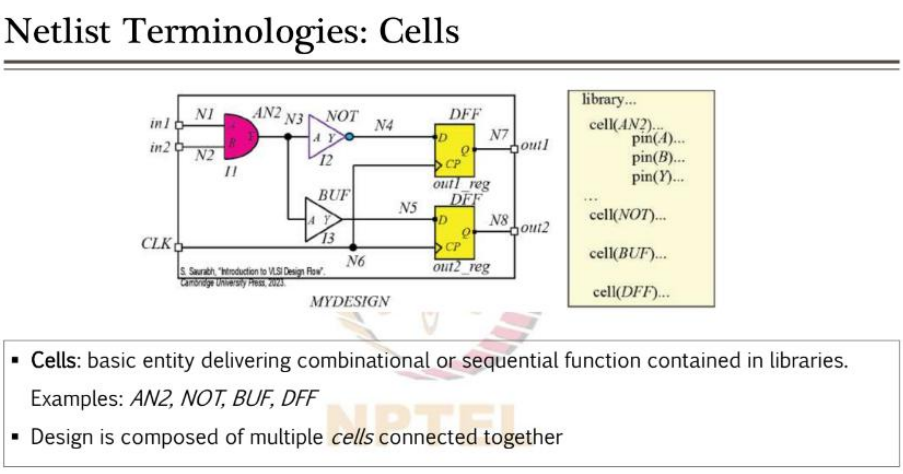
- **Input Ports**: Signals going inside the design. Example: *in1, in2, CLK*
- **Output Ports**: Signals going outside from the design. Example: *out1, out2*

Now, in this, what is a design? Design is basically the top-level entity that represents the circuit. Now, in this case, the top-level entity is this entire circuit, and we give a name to it. So, the name that is given here is MYDESIGN. So, the name of this design or the circuit will be MYDESIGN. So, when we represent a netlist in terms of the Verilog construct, then the design will be basically the top-level module of the netlist.

Now, the design actually takes input from the environment or external source or external world, and it produces some output. So, the interfaces of the design through which the design communicates with the external world those interfaces are known as ports. So, the interfaces of the design are known as ports; for example, in this circuit, in1, in2, CLK, out1, and out2. These are the interfaces through which the design communicates with the external world, and these are known as the ports.

And we can classify the ports in terms of direction. We can call the ports through which the signal goes inside the design as input ports. The ports through which the signals go outside the design are known as the output ports. So, in1, in2, and CLK are the input ports because the signal goes inside the design through these ports. The out1 and out2 are output ports because signals leave the design and go to the external world through these ports.

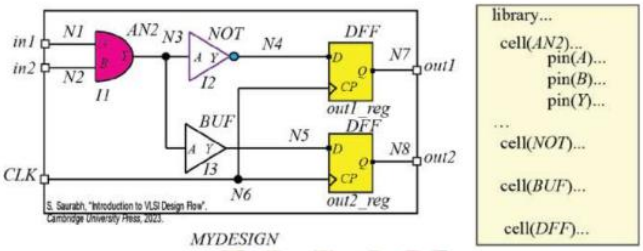
So, these input ports are also known as primary inputs or PIs in short. Similarly, output ports are also known as primary outputs or POs in short. Then, let us focus our attention on the library. Now, inside the libraries, the basic entity that delivers the combinational or sequential function is known as cells. So, inside these libraries, you will have cells. These are examples of cells, and these cells will have some names.



For example, here, the name of the cell is AN2, the name of the cell is NOT, the name of the cell is BUFF, and the name of the cell is DFF. So, these are the cells in the library. So, a design is basically a composition of multiple cells. So, we picked cells from the library and put them inside our design. So, multiple cells will be put inside our design multiple times, and those will be connected together, and we get the required functionality.

Now, let us look into a term which is known as instance. What is an instance? So, instances are cells when used inside a design. So, AN2 was a cell inside our library. Now, this AN2 cell was instantiated or put inside our design by the logic synthesis tool. So, when we put a cell into our design, the process is known as instantiation.

Netlist Terminologies: Instances

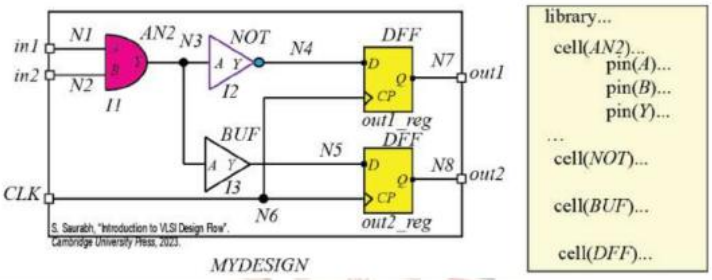


- **Instances**: cells when used inside a design are called *instances*.
  Examples: *I1, I2, I3, out1_reg, out2_reg*.
- Using a cell in a design is called ***instantiation***.
- The same cell can be instantiated multiple times.
  Example: *out1_reg* and *out2_reg* are instances of the same cell *DFF*.

And while instantiating, the tool can give arbitrary names to the instances. For example, here, I1 is the name of this instance, I2 is the name of another instance, and I3, out2_reg, and out1_reg are all the names of the instances. So, a point to note here is that the same cell can be instantiated multiple times. So, in this case, we can notice that the cell DFF is instantiated here also, as well as here.

So, the same cell can be instantiated as many times as the logic synthesis tool wants to instantiate, but the names of those instances will be different. So, in one case, it is out1_reg, and in another case, it is out2_reg. Now, let us look at what pins are. So, a pin is an interface. It is an interface of a library cell, or it is an interface of an instance. Both the interfaces of library cells, as well as instances, are known as pins. So, it is an interface through which a library cell or an instance communicates with the other components in design.

Netlist Terminologies: Pins



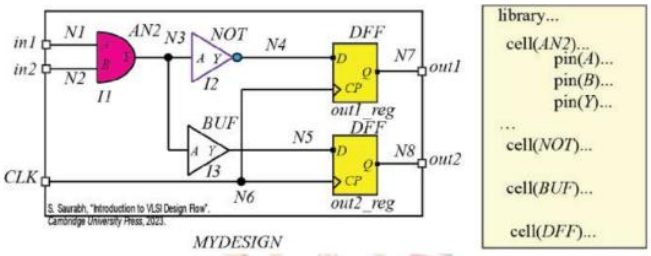
- **Pin**: An interface of a library cell or instance through which it communicates with the other components is called a pin.
- Examples: *A, B, Y* are the pins of the cell *AN2* and the instance *I1*

- **Library Pin** and **Instance Pin** (if we want to be explicit)
  - ➢ Often apparent from the context
- Input Pin or Output Pin based on direction of flow of signal to cell/instance

So, in this design, what are the pins? So, the pins for this cell, AN2, are pins A, B, and Y. For example, if we assume that this cell AN2 is basically two input AND gates, then we can think of it as A and B are the input pins of this cell, and Y is the output pin of this cell. Similarly, if AN2 is instantiated inside our design, then it will have similar pins here: A, B, and Y. If we want to be more explicit, meaning that whether it is a pin of a cell or whether it is a pin of an instance, then we can use the term library pin to signify that it is a pin of a cell and use the term instance pin to specify that it is a pin of an instance? But most of the time, based on the context, it is clear whether we are talking about a library pin or an instance pin. If it is ambiguous, then we can use the term library pin or instance pin.

Another point to note is that the direction of this, the direction of these pins can be input or output, or in some cases, it can be inout also. Though it is not shown in this design, in some cases, it can be inout also. For example, typically for IO cells, the pins can be inout where the same pin can work as an input pin or an output pin based on some condition or based on how the functionality of that pin is defined. Another thing to note about pins is their names. Now, if we look into the instances and the pin names, multiple instances can have the same pin name. For example, if we look into this NOT gate, it has got a pin name Y and A.

Netlist Terminologies: Pin Names



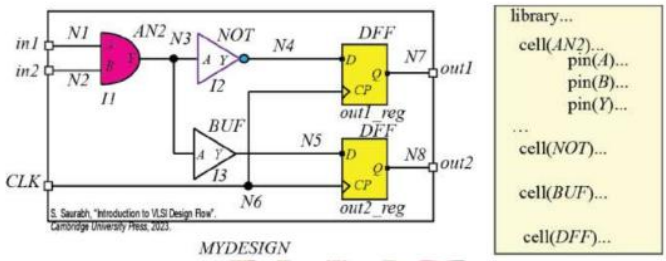
- **Instance pin name**: typically specified as combination of instance name and pin name separated by /
- Examples: *I1/A, I1/B , out1_reg/Q*

Similarly, BUFF, buffer, the pin name is A and Y for another instance also. Now, in a given design, how do we uniquely say which instance pin we are talking about? So, in that case, what we do is that we typically specify the name of a pin by a combination of instance name and a pin name and separated by some separator; we can choose a separator as a slash. So, in this course, for consistency, we will always use a slash to indicate what is the pin name. But the separator between the cell name or instance name and the pin name can be anything that is tool-specific.

Please be careful about it. For consistency in this course, we will always use this separator slash to specify the pin name. For example, if we want to talk about the pin A of this instance I1, then we will specify it as I1, that is, the name of the instance, then slash, and then A (I1/A). So, since the instance name is unique in a netlist. In a netlist, the instance name cannot clash as per the Verilog construct.

This kind of naming convention will always uniquely define a pin. Similarly, if we want to talk about another pin B of I1, we will write it as I1/B. If we want to talk about this Q pin and not about this Q pin, then we can write out1_reg/Q. If we want to talk about this one, then we will write out2_reg/Q. So, this is how you can uniquely define pin names. Now, let us look into what is a net in a design. So, nets are basically the wire that connects different instances and ports in a design.

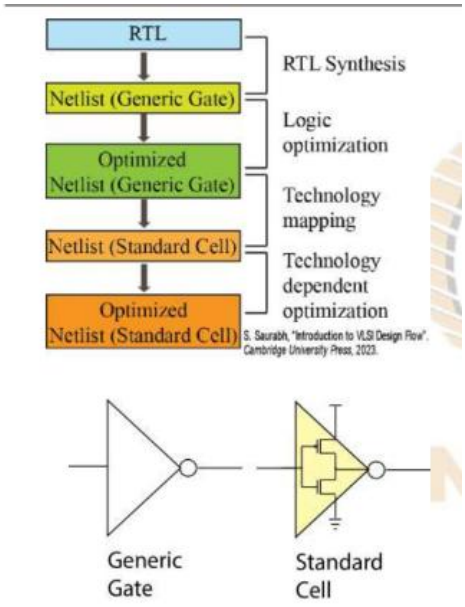
Netlist Terminologies: Nets



- **Net**: The wire that connects different instances and ports is a called net.
  Examples.: *N1, N2, N3, ...N8*

So, in this case, this wire N1, N2 is a net, then this one is a wire. So, N1, N2, N3, till N8, these, are the wires in our design. Now, one point to note is that the name of the nets that are connected directly to the port is typically the same as the port name. So, typically, the tool will choose the name of N1 of this particular net, and it will choose the name and put its name as in1 typically rather than N1. So, for the nets which are connected directly to the port, names are typically chosen as the net name. This is another point that you should be noting.

Now, having looked into the terminologies of netlist, let us look into a little bit about what are the tasks or what are the steps inside the synthesis tool while it is transforming an RTL to its corresponding netlist. So, this figure shows that given an RTL to the logic synthesis tool, what steps will be done by the logic synthesis tool to generate the final optimized netlist in terms of standard cells? So, the first step that is done or the first task that is done by the logic synthesis tool is basically RTL synthesis. So, what is RTL synthesis? RTL synthesis is the initial part of logic synthesis, and it translates an RTL to a netlist of generic logic gates. So, it takes an RTL. It translates the constructs of the RTL; for example, if the RTL is written in Verilog, then it understands the Verilog language, and it translates it in terms of generic logic gates.



Now, what are generic logic gates? So, generic logic gates have a well-defined functionality or Boolean function. For example, there can be a generic logic gate for an inverter, for an AND gate, for a NAND gate, XOR, multiplexer, demultiplexer, and these kinds of simple combinational elements, or it can be latches or flip flops, which are sequential elements. So, a generic logic gate will have a well-defined Boolean function. However, the generic logic gate will not have a fixed transistor-level implementation. So, the generic logic gate is just defining the logic or Boolean function for a given circuit. For example, in this figure, I am showing you a generic inverter. So, this generic inverter is taking a value of 0, producing 1, taking input 1, and producing output 0. This is the function. However, its internal transistor level details are not yet defined. The transistor-level detail of the inverter is defined in the library cell. So, if we consider any standard cell that exists inside the technology library, there we have an inverter whose all the transistor level details are already defined. So, we should be able to differentiate between a generic logic gate and a standard cell.

The standard cell has a well-defined transistor-level description. While a generic logic gate has the function but does not contain the transistor-level description. Therefore, the generic logic gate is a higher abstraction level. And since the transistor-level description of a generic logic gate does not exist, we cannot say anything about the area, delay, or power attributes of a generic logic gate. But we can get those estimates for the standard cells very easily because that information is contained in the technology libraries. So, in the RTL synthesis step, that is, in the first part, we translate an RTL construct in the given RTL design that we give as an input to the logic synthesis tool to generic logic gates: AND, OR, inverter, buffer. So, these generic logic gates will be used while generating

this netlist in terms of generic gates. So, the major task in this RTL synthesis is the translation of the Verilog constructs to the corresponding hardware. Other tasks involved in RTL synthesis are parsing the RTL, elaboration, or checking whether the connectivity is proper or not.

A few optimizations can also be done, for example, related to arithmetic operations and compiler-based optimization like constant folding, etc. So, these things will be discussed in more detail when we discuss RTL synthesis in the later part of the course. Then, once we have got a netlist in terms of generic gates, what the logic synthesis tool does is that it does an optimization of the logic gates. So, what this logic optimization means is that it basically tries to improve the PPA. Since the input that we have given at this stage is just in terms of generic gates, we cannot estimate timing, power, and even area numbers very accurately. But we can estimate the area in terms of a number of generic logic gates, and that can be used as an estimate for the area. So, the logic optimization is done at the level of taking this netlist in terms of generic gates and it finally delivers a netlist that is still in terms of generic gates, but it is optimized typically for area because area can typically be estimated more correctly for generic logic gates than other measures like timing or power.

Now, what is this logic optimization step? So, it is similar to the step that you might have studied in your initial courses of digital circuits where you are given a logic function, Boolean function, and you are asked to minimize it using, say, Karnaugh map and so on. So, when logic optimizations were done in the course of digital circuits, the number of variables that were treated were, say, 4 or maximum, say for the Karnaugh map, you might have gone to 5 variables. But if the number of variables increases a lot, for example, it becomes 100. Now, in that case, how to do the optimization. So, in that case, the techniques that you might have studied in the digital circuit course cannot be applied directly; they are modified or enhanced and made more sophisticated to handle larger numbers or bigger circuits, and at the end of logic optimization, we get a kind of a minimized logic circuit. But note that after logic optimization, the netlist is still in terms of generic gates. Now, once the optimization has been done then a step is done, which is carried out, which is known as technology mapping. Now, in technology mapping, what is done is that the generic gates that we had are translated to appropriate standard cells that are picked from the libraries and put in the netlist.

So, technology mapping maps a netlist consisting of generic logic gates to standard cells in the given technology library. So, at the end of this process, we will get a netlist consisting of standard cells. Now, this is not a very easy task, or it is a kind of challenging task because if we consider, say, a generic gate in terms of inverter. Now, in the technology library, we will have multiple implementations of an inverter, for example, inverters of size 1, 2, 4, 8, 16, etc. So, typically, when we make a technology library for the same logic function, in this case, an inverter, we have multiple cells of

different drive strengths. If we increase the area, typically, the drive strength goes on increasing, but the area cost also goes on increasing. Why do we design multiple types of inverters with multiple drive strengths in a library? The reason is that it gives flexibility to the logic synthesis tool to pick the one that is appropriate. For example, if an inverter is coming in a very timing-critical path, then it will use an inverter of, say, higher drive strength so that the timing improves, delay decreases, and if the inverter is not that critical on the timing path, then it probably can use an inverter with the lowest drive strength and the smallest area. So, there is a lot of flexibility, or there are many options available for a logic synthesis tool to do a mapping of a generic netlist in terms of generic logic gates to a netlist in terms of standard cells. Because the same function can be implemented in many different ways using standard cells, and for the same cell, we can pick many different kinds of standard cells from the technology library.

So, the solution space is huge, and finding an optimum solution from that solution space is a challenge that the technology mapper needs to handle. Then, once the technology mapping is done, we get a netlist, which is in terms of standard cells. Now, we say that the standard cells have the transistor level description very well defined, meaning that using the standard cell information in the technology library, we can derive PPA estimates more accurately. For example, we can estimate what will be the delay of an inverter because, in the technology library, there is information regarding what will be the delay of an inverter under different conditions. From that information, the delay can be derived, and for the full circuit, a kind of delay calculation can be done, and timing estimates can be made. Similarly, power estimates can be made. Now, also, once we have committed to a standard cell, we can estimate the area and so on. So once we have got a mapped design or technology-mapped netlist, then over that, we can do optimization more accurately because PPA estimates can be made much more accurate.

So, after the technology mapping step, the step is done, which is known as technology-dependent optimization for timing, power, and the area. Now, once this optimization is done, we get a netlist implemented in terms of standard cells, and also it is optimized. Now, this netlist can go to the next step, which is basically the physical design design step. There are a few more steps in between related to design for test, but those things we will be discussing in more detail in the later part of the course. But for now, we can consider that once we have done the logic synthesis, we get a netlist, and that netlist can be passed to the physical design for further processing.

So, these are some of the important references that I have used in this lecture, and you may refer to them for more understanding. So, in summary, in this lecture, we have looked into the RTL to GDS implementation flow, and in that, we have looked into the logic synthesis part. and at the end of the logic synthesis, we get a netlist, which is implemented in terms of the standard cells, and then we can take that netlist and carry out

the next part of the RTL to GDS implementation, which is physical design. So, we will be discussing physical design in the next lecture. Thank you very much.