Module 12

Introducing the Process of Synthesis

cadence

This module explores the basics of synthesis – how it works, where it fits in your flow, its strengths and weaknesses, and some things to look out for.

Module Objective

In this module, you:

- Briefly examine the synthesis process and its results

Topics

- What is Logic Synthesis?
- How do you use synthesis?
- What does synthesis do?
- What synthesis sometimes can't do well
- Technology-specific issues
- Synthesis in your design flow
- Synthesizable Verilog

cadence

Your objective is to be able to briefly describe the synthesis process and its results to team members that are unable to take this course.

To do that you should know at least somewhat about:

- How you use synthesis.
- What the synthesis tool does.
- Synthesis tool strengths and weaknesses.
- Technology-specific issues.
- The Verilog language synthesis "subset".

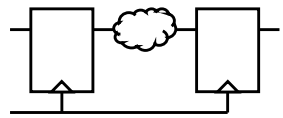
What Is Logic Synthesis?

Logic synthesis is a tool that:

- Infers storage elements (latches, registers).
- Infers combinational logic feeding storage elements.
- Optimizes generated structure to meet cost factors:
  - Area / speed / power / routability

A person new to synthesis might ask:

- How much of the design?
- How accurately?
- How optimally?



cadence

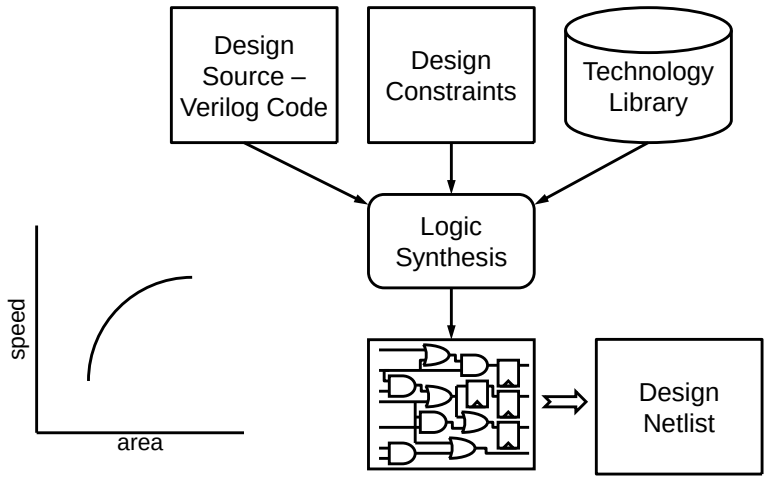
Synthesis automatically transforms your RTL Verilog design into an optimized netlist of predefined cells. Some obvious first questions might be the following:

- What parts of the design can synthesis handle?
- Is the result truly functionally equivalent to the RTL?
- Just how optimal is that result really?

How Do You Use Synthesis?

You provide design source, design constraints, and technology library.

The tool infers logic from the HDL source, maps the inferred logic to technology library macros, and optimizes the circuit to meet constraints.



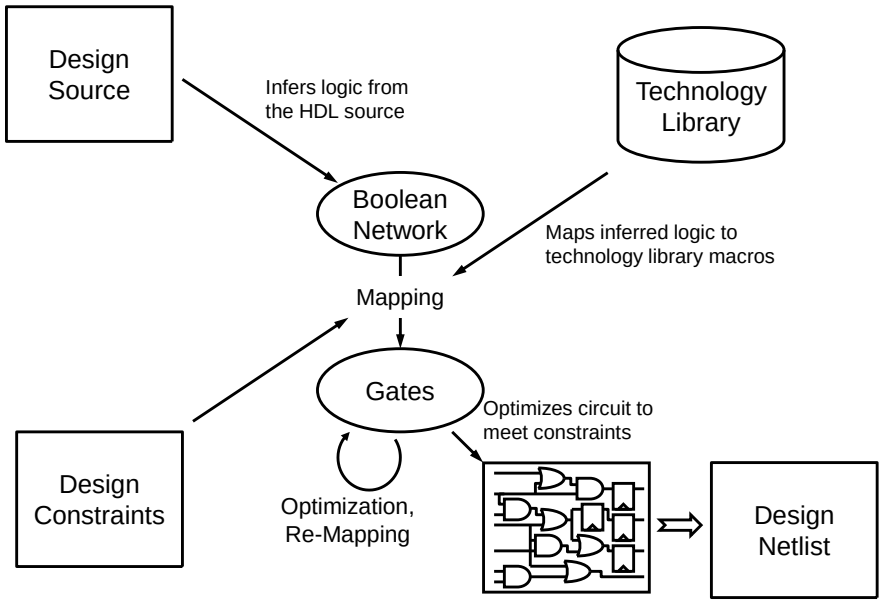
A synthesis tool can read your RTL Verilog code and convert it into a gate-level netlist, using gates and cells from a specified target technology library.

The synthesis tool will try to meet your design constraints for the area and performance, and will produce reports to tell you how successful it was.

Design constraints include the required clock speed, input drive strength and arrival time with respect to the clock, and output load and required arrival time with respect to the clock. Design constraints can also include power consumption and can sometimes include noise immunity, testability, and factors affecting ease of place and route.

The synthesis tool trades circuit speed and circuit area. As a general statement, it can produce a small design or a fast design or a compromise between small and fast. Several factors shift the area/speed curve. Among those factors are the target technology and the circuit architecture.

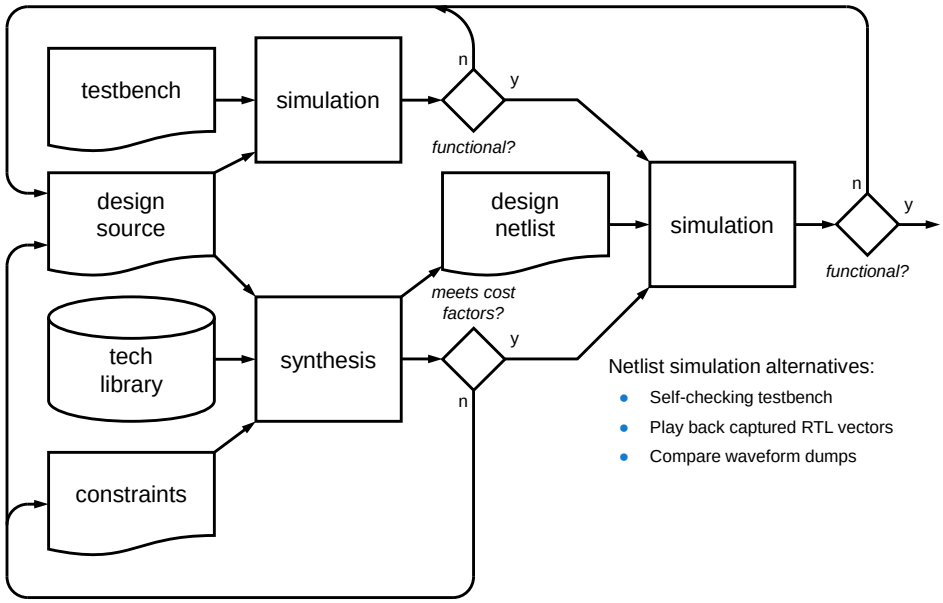
What Does Synthesis Do?



This diagram is of course very simplified. Synthesis technology continues to evolve and this course does not intend to describe its particulars anyway. In a very general sense:

- The tool first transforms HDL input into an internal Boolean network of sequential storage elements and the logic “cone” expressions that feed them.
- The tool then does some logic restructuring, such as to prune logic that does not drive anything and to share common subexpressions. It also at this point makes a guess about operator implementation, basing its choice on expression length. For example, for a add operation it might choose a carry look-ahead adder instead of a ripple-carry adder based on operand width.
- The tool next maps the design to the gates actually available in the target library and performs optimizations based on your constraints and the actually parameters of those gates.

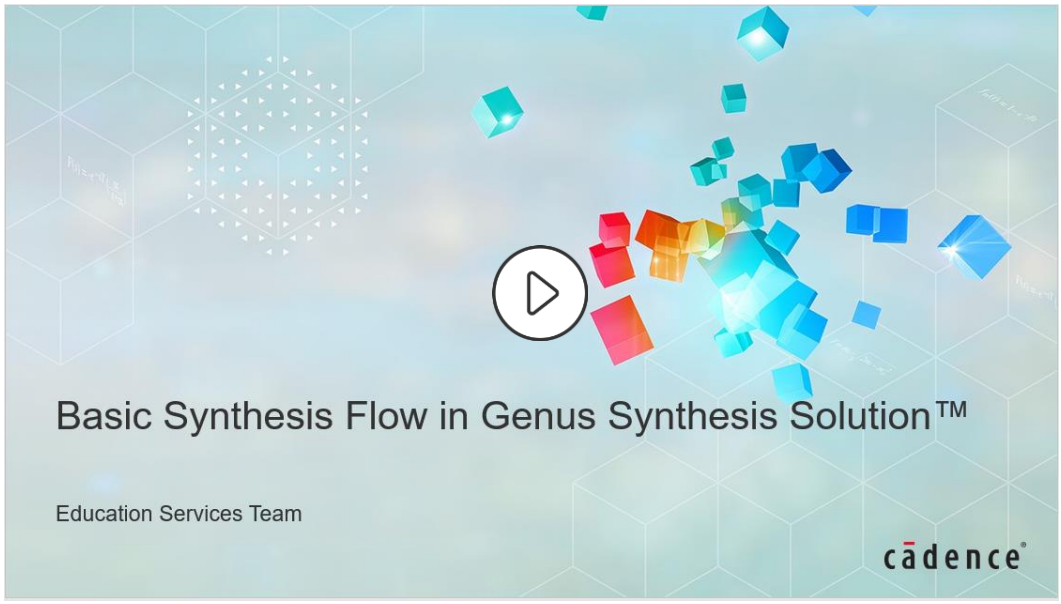
Synthesis in Your Design Flow



This diagram is of course very simplified. Synthesis technology continues to evolve and this course does not intend to describe its particulars anyway. In a very general sense:

- The tool first transforms HDL input into an internal Boolean network of sequential storage elements and the logic “cone” expressions that feed them.
- The tool then does some logic restructuring, such as to prune logic that does not drive anything and to share common subexpressions. It also at this point makes a guess about operator implementation, basing its choice on expression length. For example, for a add operation it might choose a carry look-ahead adder instead of a ripple-carry adder based on operand width.
- The tool next maps the design to the gates actually available in the target library and performs optimizations based on your constraints and the actually parameters of those gates.

Video: Basic Synthesis Flow



221

cadence

Click below for a direct video link in Cadence Online Support or search for the video name in Support:

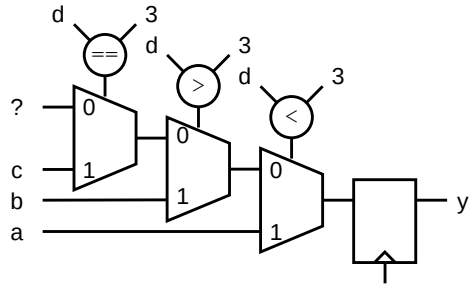
https://support.cadence.com/apex/ArticleAttachmentPortal?id=a1O3w000009frEfEAI&pageName=ArticleContent

Logic Inference Is Literal

Synthesis tools infer logic from HDL structure and statements.

- They then optimize the logic as needed to meet constraints.
- Here the == operator is redundant but could still appear in the netlist.

```
always @(posedge clk)
  if (d < 3)
    y <= a;
  else if (d > 3)
    y <= b;
  else if (d == 3)
    y <= c;
```



What is the "?" input?

cadence

Synthesis tools do not interpret or analyze your code to understand your intention. They initially transform your RTL design to Boolean logic quite literally, and then optimize it. They stop optimization when the design meets your requirements or they run out of time. For any non-trivial design, synthesis produces an optimization that is hopefully “good enough” but not likely to be perfect.

You can help the optimization process by explicitly organizing and coding your design as closely as possible to what you want the optimized result to look like. Write RTL but “think” hardware. A later module discusses such RTL coding best practices for synthesis.

The design as first parsed and elaborated will reflect the RTL code. It will contain a priority structure of multiplexors and operators. If all inputs arrive at the same time, then the optimized design will likely consist of a sum of three products, the priority structure having been optimized.

Coding Style Affects Results

Synthesis tools infer logic from HDL structure and statements.

- Logic synthesis tools do not insert storage elements.
  - Pipeline considerations (number of registers) are totally your responsibility.
- Some logic synthesis tools can move storage elements upstream or downstream in combinational logic to balance delays.
- The quality of results depend mainly on code.

cadence

A synthesis tool does not optimize the registers or latches in your design. Be careful how you structure your registered processes so that you only infer the registers you need. Check synthesis reports carefully to make sure.

You also need to be careful how much combinational logic is placed between adjacent register banks. If you place a 32-bit multiplier between adjacent register banks with a 10ns clock period, don't be surprised if the synthesis tool struggles to produce a result.

Generally, you need to be careful how you partition and structure your design into combinational and registered logic.

What Synthesis Sometimes Can't Do Well

- Clock trees
  - Usually require detailed, accurate net delay information.
- Complex clocking schemes
  - Synthesis tools prefer simple, single clock synchronous designs.
- Memory, IO pads, technology-specific cells
  - You will probably need to instantiate these by hand.
- Specific macro-cells
- Synthesis tools can analyze hundreds of potential implementations in much less time than you need to analyze just one.

Logic synthesis primarily targets for optimization of the logic cones between registers. Some things that require special handling are, for example:

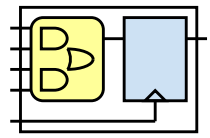
- Clock tree design is a global, chip-wide problem, which particularly in deep submicron designs requires detailed and realistic net delay information. No single clock tree generation method can apply to all designs. You will likely need to design the clock tree later as part of silicon layout.
- Synthesis tools typically have difficulty with complex clocking schemes. Synthesis works most reliably with a single clock per RTL module, and some tools restrict you to at most two clocks or require that the multiple clocks be harmonically related.
- You may have to manually instantiate I/O cells and large regularly structured cells that synthesis tools do not readily select. You may want to utilize the technology vendor's silicon compilers to generate such large regularly-structured datapath elements and large memory blocks.

For any nontrivial design, synthesis produces an optimization that is hopefully good enough but not likely to be perfect. For a critical path that does not meet timing requirements, it is quite possible that you can manually examine the path and find some way to improve it.

Technology-Specific Issues

- Each technology suggests its own optimal architecture, with architecture-specific “tricks” for good utilization and speed.
- Code becomes technology specific.
- Refer to vendor literature.

ASIC	FPGA
Technologies are similar	Technologies are not similar
Basic units are small and flexible	Basic units are large, not flexible
Optimization algorithms are similar	Optimization algorithms are not similar

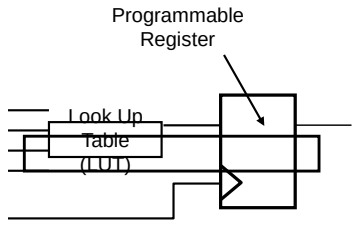


Each technology suggests its own optimal architecture, with architecture-specific “tricks” for good utilization and speed. To understand these differences, let’s compare synthesis of an ASIC to synthesis of an FPGA:

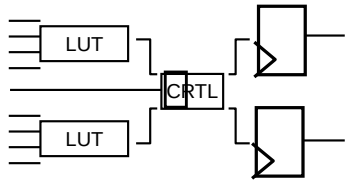
- Synthesis of an ASIC is similar for all technologies – the library cells are relatively small and similar between technologies, routing delays are relatively more predictable, and the tools use basically the same optimization algorithms.
- Synthesis of an FPGA can vary widely between technologies – the FPGA can be EEPROM, SRAM, or anti-fuse, each vendor has different building blocks that tend to be relatively large, and an FPGA architecture is relatively inflexible, leading to relatively widely-varying routing delays. FPGA design employs technology-specific and architecture-specific tricks, implemented as user-defined attributes, comments or direct instantiation, which are difficult for a synthesis tool to implement, thus making your Verilog code technology dependent and reducing its portability.

Programmable Logic Device Specific Issues

- Different architectures for different technologies
- Fixed architecture within a specific technology
- Architecture specific “tricks” for best utilization/speed
- Technology specific/generic code trade-offs
- How your synthesis tool handles your technology



Logic "Cell" – technology 1



Logic "Cell" – technology 2

Many of the comments so far have assumed synthesis to an ASIC. With an FPGA there are some different problems. ASIC synthesis tools all work in the same way, using the same optimization algorithms, and the target technology libraries all have similar cells. With FPGAs there are different technologies, EEPROM, SRAM and anti-fuse, and each vendor has different building blocks, which tend to be much larger than the gate-level primitives used in an ASIC.

For a particular FPGA, the architecture is fixed, limiting the usefulness of static timing analysis since routing delays can vary widely – with an ASIC they are much more predictable. What’s more, to get a good “fit” the FPGA designer traditionally employs architecture specific tricks which are difficult to implement in a synthesis tool.

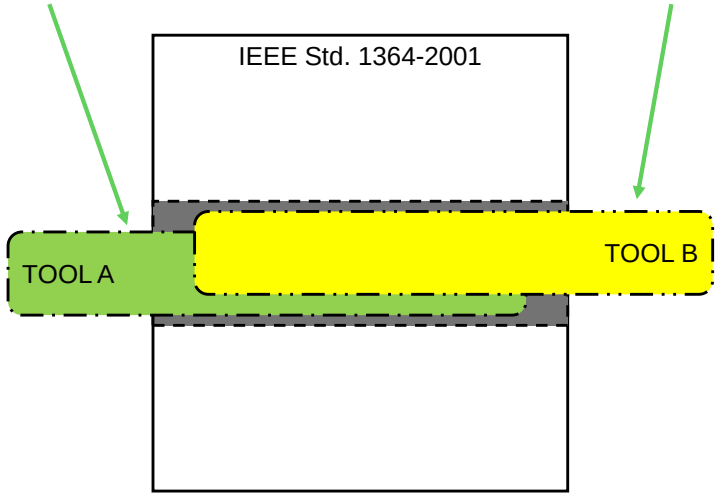
VHDL code may need to make use of technology- and architecture-specific techniques and tricks. These may be expressed with user-defined attributes, comments or direct instantiation of specific cells. Such techniques restrict the technology independence and portability of your code, but allow the most efficient implementation with a specific technology.

A key issue is how well your synthesis tool handles your chosen technology. To achieve a good result, it will need architecture specific algorithms.

Synthesizable Verilog

IEEE Std. 1364.1 defines the synthesizable Verilog constructs.

- Not all synthesis tools yet accept all the standard constructs
- Most synthesis tools still accept also their own legacy constructs



The IEEE Std. 1364.1-2002 for Verilog RTL synthesis describes use of the Verilog "synthesizable subset" in a manner from which logic synthesis tools can infer logic.

- You will rarely but perhaps occasionally use a logic synthesis tool that does not accept some standard synthesizable construct.
- As logic synthesis existed long before the standard, the predominant tools still support constructs that did not become standard.
- This course describes what is standard. If you write your RTL code according to this course, then your code will be portable between the maximum number of synthesis tools.

Module Summary

You should now be able to briefly describe what is synthesis.

This module described:

- What you provide to the synthesis tool:
  - Design source, design constraints, technology library
- What the synthesis tool does:
  - infers logic from the HDL source, maps the inferred logic to technology library macros, and optimizes the circuit to meet constraints
- What the synthesis tool might need help with:
  - Complex clocking schemes, large memories, I/O pads
- What part of Verilog is synthesizable:
  - Defined by IEEE Std. 1364.1

cadence

This module explored the basics of synthesis – how it works, where it fits in your flow, its strengths and weaknesses, and some things to watch for.

Module Review

1. What are the user-provided inputs to logic synthesis?
2. ***True or False?*** Given multiple different functionally accurate descriptions of a design function, a synthesis tool will for each produce the exact same one “best” implementation.
3. What design blocks might you need to instantiate rather than infer?
4. What entity determines how you must code Verilog designs for inference by a logic synthesis tool?

cadence

This page does not contain notes.

Module Review Solutions

1. What are the user-provided inputs to logic synthesis?
   - The logic synthesis user must provide the design source, design constraints, and target technology library.
2. True or False: Given multiple different functionally accurate descriptions of a design function, a synthesis tool will for each produce the exact same one “best” implementation.
   - False. A synthesis tool infers logic exactly as described, and then optimizes it to meet constraints. It stops when it meets the constraints or exceeds the computational limits you set.
3. What design blocks might you need to instantiate rather than infer?
   - You are likely to instantiate rather than infer large memories and I/O pads, and potentially at least partially instantiate clock trees.
4. What entity determines how you must code Verilog designs for inference by a logic synthesis tool?
   - The “IEEE Standard for Verilog Register Transfer Level Synthesis” determines how you must code Verilog designs for inference by a logic synthesis tool.

cadence

This page does not contain notes.

Lab

Lab 12-1 Exploring the Synthesis Process

- For this lab, you synthesize a small multiplexor model and examine the results.

cadence

Your objective for this lab is to briefly observe the synthesis process and examine the results.

For this lab, you synthesize a simple design and examine the resulting netlist to associate the netlist components to the RTL operators and statements.