

# Chapter 2

## Synthesis Basics

Synthesis is the first step in the design process, where timing constraints are used.

### 2.1 Synthesis Explained

Let us consider a 3-bit counter, which counts in the sequence  $0 \rightarrow 5 \rightarrow 2 \rightarrow 7 \rightarrow 6 \rightarrow 3 \rightarrow 5 \rightarrow 1 \rightarrow 0$ . If we have to realize the gate-level circuit for this counter, it would take a lot of time to draw the *Karnaugh map* and then realize the logic feeding into the *D* pin of each of the 3 flops which form the counter.

However, it is much faster to write an HDL code, which describes the above functionality. This HDL code can then be taken through a tool, which will create the corresponding netlist.

Synthesis in the context of electronic design means realization of a gate-level netlist to achieve a specific functionality. Besides the specific functionality, the process of synthesis might also meet certain other requirements, namely, power, frequency of operation, etc.

Sometimes, there are specialized synthesis tools for specific kinds or portions of circuit, e.g.:

- Clock tree synthesis – which creates the clock tree
- Data path synthesis – which creates a repetitive structure in the data path
- Logical synthesis – used for realizing all kinds of logical circuits

Usually, the word “synthesis” just by itself means logical synthesis only.

## 2.2 Role of Timing Constraints in Synthesis

The design process involves a lot of steps. These steps are of various kinds, e.g.:

- Capturing intent
- Verifying that the design is in line with what we desire
- Estimating certain characteristics
- Actually realizing the design

The last series of steps are also called implementation steps. Synthesis is the first among the implementation steps. The following subsections give a few examples of the choices that a synthesis tool might need to make and the basis of the decision. These are all examples of additional information (beyond functionality) that the synthesis tool needs to be provided through constraints.

### 2.2.1 Optimization

For a synthesis tool to realize a netlist, it needs several pieces of information. The first information is the functionality that the realized netlist needs to perform. This information comes from the HDL description.

For a device, obviously functionality is the most important consideration. However, designers have to be also very sensitive to:

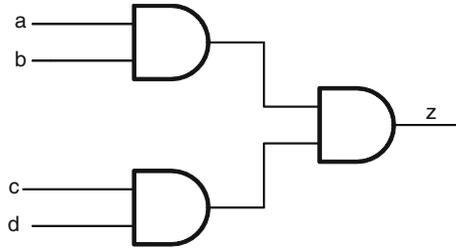
- Area: We want to fit as much functionality into the same unit area as possible.
- Power: We want to conserve battery power and also reduce junction heating.
- Performance: We want to get highest possible speed from the device.

However, each of the above goals may impact the others and sometimes negatively. For example, if we want to get best speed, we will need to have higher drive devices, which will mean higher power and greater area on silicon. So, instead of a designer trying to squeeze out the maximum performance, the designer might want to get just about enough performance that would achieve the purpose and, in the process, save on area and power.

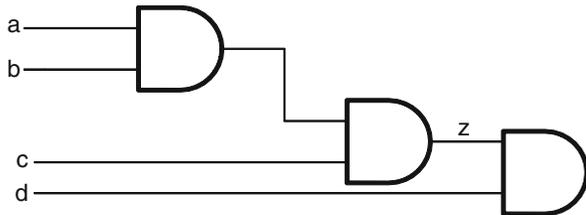
A designer communicates his requirements around area, power, and performance to the synthesis tool through *constraints*. Once the synthesis tool is able to achieve a circuit that meets these goals, the tool need not make any further effort to realize a “better” circuit. Any further attempt to improve in any one dimension could worsen the other dimensions.

So constraints are used to tell the synthesis tool – among the many possible implementations possible to realize the same functionality, which should be chosen so that all the three requirements on area, power, and performance are met.

**Fig. 2.1** ANDing of 4 inputs



**Fig. 2.2** Alternative realization of Fig. 2.1



### 2.2.2 Input Reordering

Let us consider a function involving *ANDing* of four inputs, *a*, *b*, *c*, and *d*. One of the simplest realizations of this circuit is as shown in Fig. 2.1.

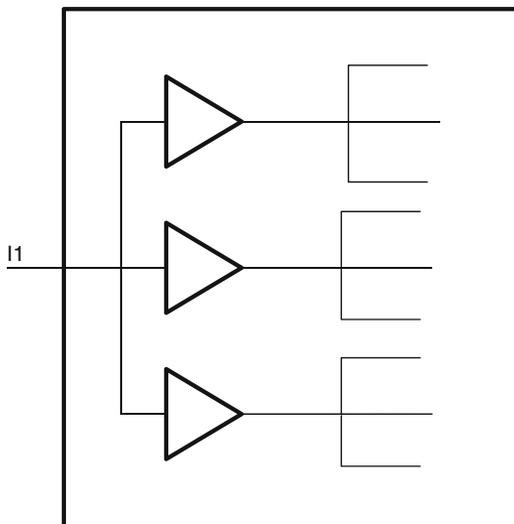
However, now imagine that the input *d* arrives much later than other inputs. So the final evaluation of the circuit has to wait till *d* arrives and passes through 2 *AND* gates. On the other hand, there can be an alternative realization of the same functionality as shown in Fig. 2.2.

In this circuit, by the time *d* arrives, the other three signals have already been evaluated, and *d* has to travel through only one *AND* gate.

Though both circuits perform the same functionality and have similar area (3 *AND* gates), a designer might prefer Fig. 2.1 or 2.2, depending upon whether *d* comes along with all other signals or whether *d* comes much later than all other signals. If instead of *d*, it was some other signal which was coming much later, then *d* might be swapped with that late arriving signal.

Thus, depending upon the relative arrival time for various inputs feeding into the same combinational logic, the synthesis tool might need to decide which design should be chosen among the available choices – so that the last arriving signals have to cross the minimum number of logic.

Designers use constraints to convey to the synthesis tool about the arrival time of various input signals.

**Fig. 2.3** Input being buffered

### 2.2.3 *Input Buffering*

*Drive* can be thought of as current-carrying capability. Thus, higher drive means output would switch faster and a higher amount of load can be connected. Let us say, a specific input has to drive a huge fanout cone. But, whether the specific input can drive such a huge cone or not depends upon the driving capability of the signal which is driving the input. If the signal driving the input cannot drive the load for the whole fanout cone, then the signal would need to be buffered before it can be fed into the huge cone.

Figure 2.3 shows an input which has to drive a fanout load of 9. However, it does not have the drive strength for that kind of load. Hence, buffering is done on the input, before feeding into the load. With this buffering, the load that the input has to drive is only 3.

Designers need to tell the synthesis tool the driving capability of the external signal which is driving the input so that synthesis tool can decide whether or not to put additional buffers. And constraints are used to convey information about the drive strength of the external inputs.

### 2.2.4 *Output Buffering*

Similar to input buffering, a design might need to have additional drive capability at the output side, if the output port is expected to drive a huge load externally. So designers need to convey to the synthesis tool – the external load that a port

might have to drive. Synthesis tool will then choose appropriate cells or buffers with the right drive strengths that can drive the load. And constraints are used to convey information about the external load that needs to be driven by the output port.

## 2.3 Commonly Faced Issues During Synthesis

Synthesis step can have different class of issues. In fact, one could write a whole book around issues faced during synthesis. This section gives a glimpse of some issues around synthesis related to constraints. These same topics are discussed in much more details in subsequent chapters of the book.

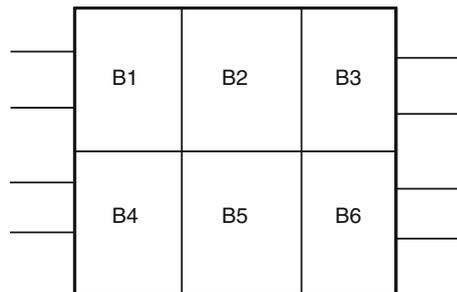
### 2.3.1 Design Partitioning

Though synthesis techniques have provided a major leap in terms of designer's productivity, the biggest bottleneck of a synthesis tool is the size of a design that it can synthesize. The design sizes today are humongous, compared to the sizes of design that synthesis tool can synthesize.

Thus, a full design has to be broken into smaller units, called blocks. During synthesis stage, the blocks are created based on logical view of the design, namely, related functionality being put into one block. This kind of partitioning is called *logical partition*. A synthesis tool would synthesize one block as a unit. Thus, a synthesis tool can view only a block at any given time, and it does not see how the block interacts with the rest of the design. Figure 2.4 shows how a design is composed of logical blocks.

The outermost rectangular boundary denotes the complete design. Usually, the design would have requirements listed for the whole design. Because the synthesis tool cannot synthesize the whole design, so the design is partitioned into smaller blocks (*B1* through *B6*), represented by inner smaller rectangles.

At any time, synthesis tool views a block. But, the requirements are known for the complete design. So the top-level constraints for the complete design have to be



**Fig. 2.4** A design partitioned into blocks

broken into constraints for individual blocks. These constraints for individual blocks have to be created – based on interaction of the block with all other blocks. For example, for the block *B1*, the constraints have to be specified to define its interaction with primary inputs for the design as well as its interaction with other blocks *B2* and *B4*.

So what was supposed to be just the constraints at the top level now gets translated into many more constraints defined at each interface. And as the number of constraints grows, there are higher chances of errors. In the figure, the partitions are shown as regular rectangular blocks. In reality, all the blocks interact with many blocks, and that increases the complexity of the total set of constraints.

Let us consider the interaction between blocks *B1* and *B2*. Based on this interaction, there would be some constraints for blocks *B1* and corresponding constraints for block *B2*. Many times, the people or the team working on these different blocks are different. There have been many instances where the constraints written for interfacing blocks are not consistent. For example, *B1*'s designer might assume that he will get 50 % of the total path time for his block and the remaining 50 % would be for rest of the path. Similarly, *B2*'s designers might also assume 50 % of the path time available for his block. So between the two blocks, they might consume the entire path time, leaving nothing for the top-level routing for connecting the two blocks.

### 2.3.2 *Updating Constraints*

It seems slightly strange that such inconsistency might happen among blocks of the same design. However, such inconsistencies usually creep in gradually as various blocks keep getting impacted due to some other block not meeting their initial requirements.

Let us assume block *B1* failed to meet some of its timing, which causes an impact on *B2*. Block *B2*'s designer might now have to update his constraints, and its impact might be on the *B2/B3* and *B2/B5* interface. However, at this stage, either *B3* or *B5* constraints might get out of sync with the updated constraints of *B2*, and in many cases, these changed constraints might disturb delicate balance of area, performance, and power. Thus, the block-level constraints may have to be updated depending on how the block is integrated in the subsystem or chip.

### 2.3.3 *Multi-clock Designs*

Most designs today have multiple processing cores, running on different clock frequencies. There could be different peripherals for these cores. In the process of integrating these cores which are being developed simultaneously by design groups, an inadvertent mistake of constraining a high-frequency core with a low-frequency

constraint may be missed during initial implementation. These may be eventually caught during full-chip STA, post integration. That could be pretty late as the block constraints would now have to be redone to the original specification adding an unnecessary iteration to the chip integration.

## 2.4 Conclusion

This chapter gave a glimpse of the need for constraints and nature of some of the problems related to synthesis. Synthesis has been used just as an example of an implementation tool. All implementation tools are driven by constraints. Most of the discussions mentioned in this chapter would apply to all implementation tools, not just synthesis. So incorrect constraints impact the ability of these tools to implement a circuit which will meet its performance, area, and power goals.

As design complexities are growing, the constraints themselves are also becoming complex – in order to be able to correctly represent the complex requirements as well as relationships. The nuances of the design process involving partitioning, integration, and multiple cores operating at different frequencies all add to further problems around creating constraints.

Several implementation tools also allow constraints to provide physical information, such as physical shape of a block, or specific location of ports, etc. These physical constraints are not covered in this book.



<http://www.springer.com/978-1-4614-3268-5>

Constraining Designs for Synthesis and Timing Analysis  
A Practical Guide to Synopsys Design Constraints (SDC)

Gangadharan, S.; Churiwala, S.

2013, XXVII, 226 p., Hardcover

ISBN: 978-1-4614-3268-5