

```

function fulladd
input [3:0] a, b;
input c_in;
begin
    fulladd = a + b + c_in; //bit 4 of fulladd for carry, bits[3:0] for sum.
end
endfunction

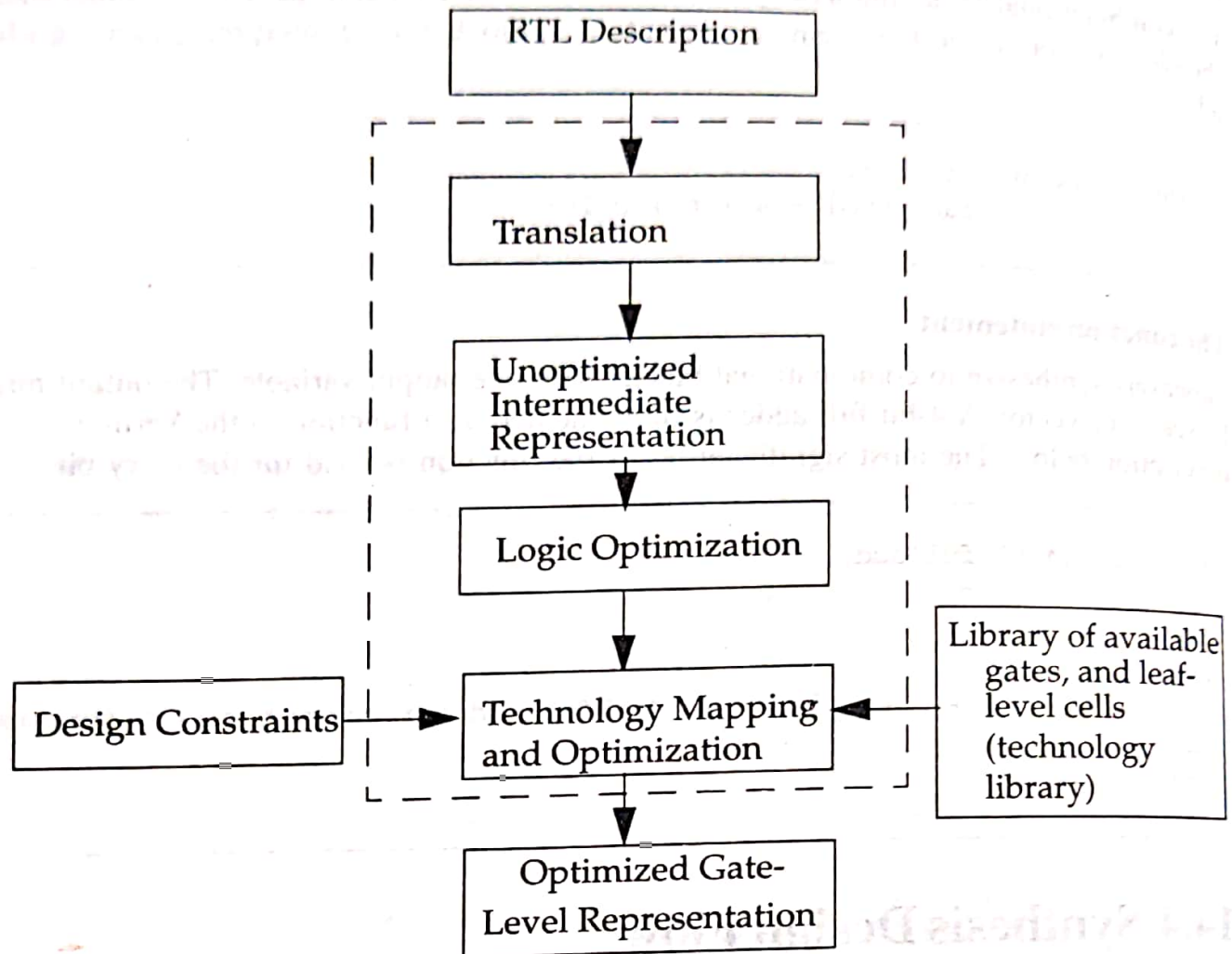
```

## 14.4 Synthesis Design Flow

Having understood how basic Verilog constructs are interpreted by the logic synthesis tool, let us now discuss the synthesis design flow from an RTL description to an optimized gate-level description.

### 14.4.1 RTL to Gates

To fully utilize the benefits of logic synthesis, the designer must first understand the flow from the high-level RTL description to a gate-level netlist. Figure 14-4 explains that flow.



**Figure 14-4** Logic Synthesis Flow from RTL to Gates

Let us discuss each component of the flow in detail.

### RTL description

The designer describes the design at a high level by using RTL constructs. The designer spends time in functional verification to ensure that the RTL description functions correctly. After the functionality is verified, the RTL description is input to the logic synthesis tool.

### Translation

The RTL description is converted by the logic synthesis tool to an unoptimized, intermediate, internal representation. This process is called *translation*. Translation is relatively simple and uses techniques similar to those discussed in Section 14.3.3, *Interpretation of a Few Verilog Constructs*. The translator understands the basic primitives and



14

operators in the Verilog RTL description. Design constraints such as area, timing, and power are not considered in the translation process. At this point, the logic synthesis tool does a simple allocation of internal resources.

### Unoptimized intermediate representation

The translation process yields an *unoptimized intermediate representation* of the design. The design is represented internally by the logic synthesis tool in terms of internal data structures. The unoptimized intermediate representation is incomprehensible to the user.

### Logic optimization

The logic is now optimized to remove redundant logic. Various technology-independent boolean logic optimization techniques are used. This process is called *logic optimization*. It is a very important step in logic synthesis, and it yields an *optimized internal representation* of the design.

### Technology mapping and optimization

Until this step, the design description is independent of a specific *target technology*. In this step, the synthesis tool takes the internal representation and implements the representation in gates, using the cells provided in the technology library. In other words, the design is *mapped* to the desired *target technology*.

Suppose you want to get your IC chip fabricated at ABC Inc. ABC Inc. has 0.65 micron CMOS technology, which it calls *abc\_100* technology. Then, *abc\_100* becomes the target technology. You must therefore implement your internal design representation in gates, using the cells provided in the *abc\_100* technology library. This is called *technology mapping*. Also, the implementation should satisfy such design constraints as timing, area, and power. Some local optimizations are done to achieve the best results for the target technology. This is called *technology optimization* or *technology-dependent optimization*.

### Technology library

The *technology library* contains *library cells* provided by ABC Inc. The term *standard cell library* used earlier in the chapter and the term *technology library* are identical and are used interchangeably.

To build a technology library, ABC Inc. decides on the range of functionality to provide in its library cells. As discussed earlier, library cells can be basic logic gates or macro cells such as adders, ALUs, multiplexers, and special flipflops. The library cells are the basic building blocks that ABC Inc. will use for IC fabrication. Physical layout of library cells is done first. Then, the area of each cell is computed from the cell layout. Next, modeling techniques are used to estimate the timing and power characteristics of each library cell. This process is called *cell characterization*.

Finally, each cell is described in a format that is understood by the synthesis tool. The cell description contains information about the following:



- Functionality of the cell
- Area of the cell layout
- Timing information about the cell
- Power information about the cell

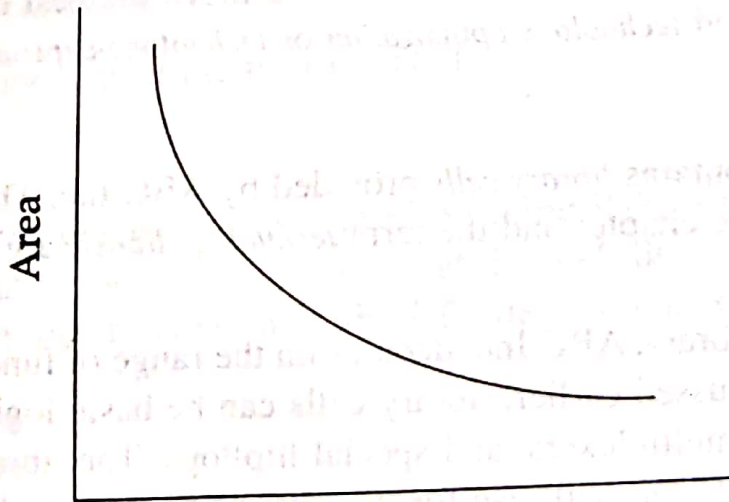
A collection of these cells is called the *technology library*. The synthesis tool uses these cells to implement the design. The quality of results from synthesis tools will typically be dominated by the cells available in the technology library. If the choice of cells in the technology library is limited, the synthesis tool cannot do much in terms of optimization for timing, area, and power.

### Design constraints

Design constraints typically include the following:

- *Timing*—The circuit must meet certain timing requirements. An internal static timing analyzer checks timing.
- *Area*—The area of the final layout must not exceed a limit.
- *Power*—The power dissipation in the circuit must not exceed a threshold.

In general, there is an inverse relationship between area and timing constraints. For a given technology library, to optimize timing (faster circuits), the design has to be parallelized, which typically means that larger circuits have to be built. To build smaller circuits, designers must generally compromise on circuit speed. The inverse relationship is shown in Figure 14-5.



**Figure 14-5** Area vs. Timing Trade-off

On top of design constraints, *operating environment factors*, such as input and output delays, drive strengths, and loads, will affect the optimization for the target technology. Operating environment factors must be input to the logic synthesis tool to ensure that circuits are optimized for the required operating environment.



## Optimized gate-level description

After the technology mapping is complete, an optimized gate-level netlist described in terms of target technology components is produced. If this netlist meets the required constraints, it is handed to ABC Inc. for final layout. Otherwise, the designer modifies the RTL or reconstrains the design to achieve the desired results. This process is iterated until the netlist meets the required constraints. ABC Inc. will do the layout, do timing checks to ensure that the circuit meets required timing after layout, and then fabricate the IC chip for you.

There are three points to note about the synthesis flow.

1. For very high-speed circuits like microprocessors, vendor technology libraries may yield nonoptimal results. Instead, design groups obtain information about the fabrication process used by the vendor, for example, 0.65 micron CMOS process, and build their own technology library components. Cell characterization is done by the designers. Discussion about building technology libraries and cell characterization is beyond the scope of this book.
2. Translation, logic optimization, and technology mapping are done *internally* in the logic synthesis tool and are not visible to the designer. The technology library is given to the designer. Once the technology is chosen, the designer can control only the input RTL description and design constraint specification. Thus, writing efficient RTL descriptions, specifying design constraints accurately, evaluating design trade-offs, and having a good technology library are very important to produce optimal digital circuits when using logic synthesis.
3. For submicron designs, interconnect delays are becoming a dominating factor in the overall delay. Therefore, as geometries shrink, in order to accurately model interconnect delays, synthesis tools will need to have a tighter link to layout, right at the RTL level. Timing analyzers built into synthesis tools will have to account for interconnect delays in the total delay calculation.

### 14.4.2 An Example of RTL-to-Gates

Let us discuss synthesis of a 4-bit magnitude comparator to understand each step in the synthesis flow. Steps of the synthesis flow such as translation, logic optimization, and technology mapping are not visible to us as designers. Therefore, we will concentrate on the components that are visible to the designer, such as the RTL description, technology library, design constraints, and the final, optimized, gate-level description.

#### Design specification

A magnitude comparator checks if one number is greater than, equal to, or less than another number. Design a 4-bit magnitude comparator IC chip that has the following specifications:



- The name of the design is `magnitude_comparator`
- Inputs *A* and *B* are 4-bit inputs. No *x* or *z* values will appear on *A* and *B* inputs
- Output *A\_gt\_B* is true if *A* is greater than *B*
- Output *A\_lt\_B* is true if *A* is less than *B*
- Output *A\_eq\_B* is true if *A* is equal to *B*
- The magnitude comparator circuit must be as fast as possible. Area can be compromised for speed.

## RTL description

The RTL description that describes the magnitude comparator is shown in Example 14-1. This is a technology-independent description. The designer does not have to worry about the target technology at this point.

*Example 14-1*      *RTL for Magnitude Comparator*

```
//Module magnitude comparator
module magnitude_comparator(A_gt_B, A_lt_B, A_eq_B, A, B);

//Comparison output
output A_gt_B, A_lt_B, A_eq_B;

//4-bits numbers input
input [3:0] A, B;

assign A_gt_B = (A > B); //A greater than B
assign A_lt_B = (A < B); //A less than B
assign A_eq_B = (A == B); //A equal to B

endmodule
```

Notice that the RTL description is very concise.

## Technology library

We decide to use the 0.65 micron CMOS process called *abc\_100* used by ABC Inc. to make our IC chip. ABC Inc. supplies a technology library for synthesis. The library contains the following library cells. The library cells are defined in a format understood by the synthesis tool.

```
//Library cells for abc_100 technology

VNAND//2-input nand gate
```

(Continued...)

VAND//2-input and gate

VNOR//2-input nor gate

VOR//2-input or gate

VNOT//not gate

VBUF//buffer

NDFF//Negative edge triggered D flipflop

PDFF//Positive edge triggered D flipflop

Functionality, timing, area, and power dissipation information of each library cell are specified in the technology library.

### Design constraints

According to the specification, the design should be as fast as possible for the target technology, abc\_100. There are no area constraints. Thus, there is only one design constraint.

- Optimize the final circuit for fastest timing

### Logic synthesis

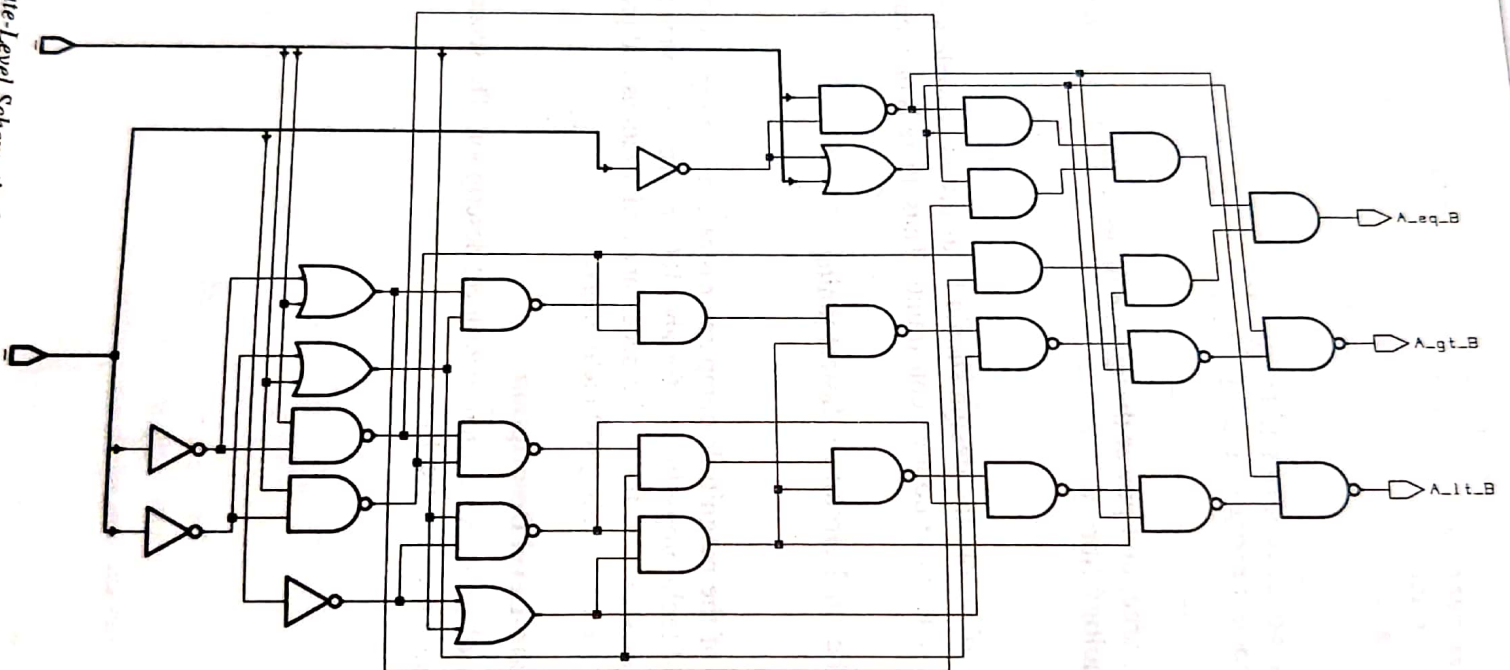
The RTL description of the magnitude comparator is read by the logic synthesis tool. The design constraints and technology library for abc\_100 are provided to the logic synthesis tool. The logic synthesis tool performs the necessary optimizations and produces a gate-level description optimized for abc\_100 technology.

### Final, Optimized, Gate-Level Description

The logic synthesis tool produces a final, gate-level description. The schematic for the gate-level circuit is shown in Figure 14-6.

Figure 14-6

Gate-Level Schematic for the Magnitude Comparator





```

module magnitude_comparator ( A_gt_B, A_lt_B, A_eq_B, A, B );
input  [3:0] A;
input  [3:0] B;
output A_gt_B, A_lt_B, A_eq_B;

wire n60, n61, n62, n50, n63, n51, n64, n52, n65, n40, n53,
     n41, n54, n42, n55, n43, n56, n44, n57, n45, n58, n46,
     n59, n47, n48, n49, n38, n39;

VAND U7 ( .in0(n48), .in1(n49), .out(n38) );
VAND U8 ( .in0(n51), .in1(n52), .out(n50) );
VAND U9 ( .in0(n54), .in1(n55), .out(n53) );
VNOT U30 ( .in(A[2]), .out(n62) );
VNOT U31 ( .in(A[1]), .out(n59) );
VNOT U32 ( .in(A[0]), .out(n60) );
VNAND U20 ( .in0(B[2]), .in1(n62), .out(n45) );
VNAND U21 ( .in0(n61), .in1(n45), .out(n63) );
VNAND U22 ( .in0(n63), .in1(n42), .out(n41) );
VAND U10 ( .in0(n55), .in1(n52), .out(n47) );
VOR U23 ( .in0(n60), .in1(B[0]), .out(n57) );
VAND U11 ( .in0(n56), .in1(n57), .out(n49) );
VNAND U24 ( .in0(n57), .in1(n52), .out(n54) );
VAND U12 ( .in0(n40), .in1(n42), .out(n48) );
VNAND U25 ( .in0(n53), .in1(n44), .out(n64) );
VOR U13 ( .in0(n58), .in1(B[3]), .out(n42) );
VOR U26 ( .in0(n62), .in1(B[2]), .out(n46) );
VNAND U14 ( .in0(B[3]), .in1(n58), .out(n40) );
VNAND U27 ( .in0(n64), .in1(n46), .out(n65) );
VNAND U15 ( .in0(B[1]), .in1(n59), .out(n55) );
VNAND U28 ( .in0(n65), .in1(n40), .out(n43) );
VOR U16 ( .in0(n59), .in1(B[1]), .out(n52) );
VNOT U29 ( .in(A[3]), .out(n58) );
VNAND U17 ( .in0(B[0]), .in1(n60), .out(n56) );
VNAND U18 ( .in0(n56), .in1(n55), .out(n51) );
VNAND U19 ( .in0(n50), .in1(n44), .out(n61) );
VAND U2 ( .in0(n38), .in1(n39), .out(A_eq_B) );
VNAND U3 ( .in0(n40), .in1(n41), .out(A_lt_B) );
VNAND U4 ( .in0(n42), .in1(n43), .out(A_gt_B) );
VAND U5 ( .in0(n45), .in1(n46), .out(n44) );
VAND U6 ( .in0(n47), .in1(n44), .out(n39) );
endmodule

```

If the designer decides to use another technology, say, `xyz_100` from XYZ Inc., because it is a better technology, the RTL description and design constraints do not change. Only the technology library changes. Thus, to map to a new technology, a logic synthesis tool simply reads the unchanged RTL description, unchanged design constraints, and new technology library and creates a new, optimized, gate-level netlist.

Note that if automated logic synthesis were not available, choosing a new technology would require the designer to redesign and reoptimize by hand the gate-level netlist in Example 14-2.

### **IC Fabrication**

The gate-level netlist is verified for functionality and timing and then submitted to ABC Inc. ABC Inc. does the chip layout, checks that the post-layout circuit meets timing requirements, and then fabricates the IC chip, using `abc_100` technology.

### **Gate-Level Netlist**