CHAPTER
High-Level Design Flow

Checked 7/3 7h:57

This chapter describes the design flow used to create complex FPGA and ASIC devices. The designer starts with a design specification, creates an RTL description, verifies that description, synthesizes the description to gates, uses place and route tools to implement the design in the chip, and then verifies that the final result is correct in terms of function and timing. The high-level design flow is shown in Figure 11-1.

The first step in a high-level design flow is the design specification process. This process involves specifying the behavior expected of the final design. The designer puts enough detail into the specification so that the design can be built. The specification is usually written in the designer's native language and specifies the expected function and behavior of the design using textual description and graphic elements.

Figure 11-1
High-Level Design Flow.

HDL Capture

After the specification has been completed, the designer or designers can begin the process of implementation. Some design teams create a high- level behavioral or algorithmic description of the design to verify design intent, then convert that description to RTL (Register Transfer Level) later. However, most design teams skip the behavioral description and implement the RTL directly. The RTL is created during the HDL capture step. The designer

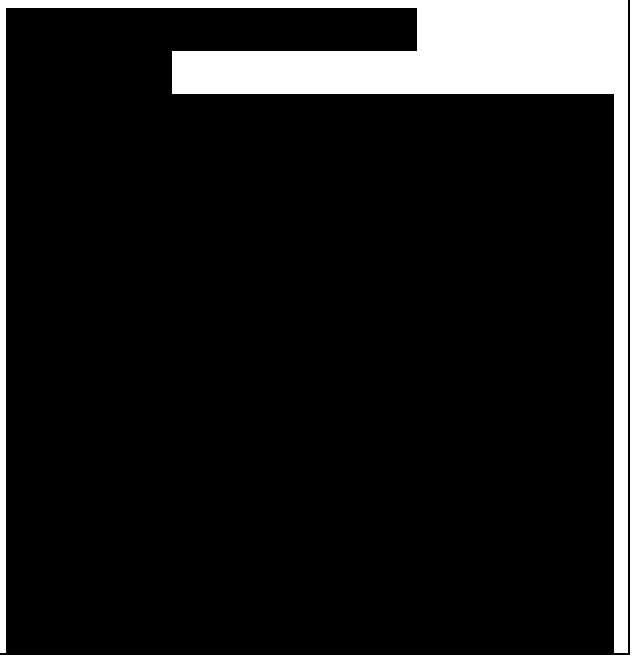
CHƯƠNG 11
Quy trình thiết kế mức cao

Design Flow: có thể dịch là "lưu lượng thiết kế"

Chương này trình bày quy trình thiết kế để tạo ra các thiết bị FPGA và ASIC phức tạp. Nhà thiết kế bắt đầu với đặc tả kỹ thuật về thiết kế, tạo ra mô tả RTL, kiểm tra mô tả đó, tổng hợp các mô tả thành cổng, sử dụng các công cụ đặt và định tuyến để thực thi thiết kế trong chip, và sau đó kiểm tra xem thiết kế cuối cùng có thỏa mãn chính xác về chức năng và thời gian hay không. Hình 11-1 biểu diễn quy trình thiết kế mức cao.

Bước đầu tiên trong quy trình thiết kế mức cao là quy trình mô tả đặc điểm kỹ thuật của thiết kế. Trong quá trình này, chúng ta xác định những đặc tính mong đợi của thiết kế cuối cùng. Nhà thiết kế phải tạo ra bảng đặc tả kỹ thuật đủ chi tiết để phục vụ cho quá trình thiết kế của mình. Bảng đặc tả thường được viết bằng ngôn ngữ bản xứ của nhà thiết kế và mô tả chức năng và đặc tính mong đợi của thiết kế sử dụng mô tả văn bản và các yếu tố đồ họa.

Hình 11-1



creates the VHDL RTL description that describes the clock-by-clock behavior of the design. The designer most likely uses a common text editor such as Emacs, or vi, whatever is available on the designer's computer. Some designers also use high-level entry tools that contain block editors and state machine editors that automatically create the VHDL code.

The designer enters the VHDL code for entities of the design and checks them for correct syntax. After the syntax errors have been removed, the designer can begin the process of verifying the correctness of the VHDL using RTL simulation.

RTL Simulation

The RTL simulation step is used to verify the correctness of the RTL VHDL description. The designer has described the clock-by-clock behavior of the design. Now, the designer uses stimulus that represents the design environment to drive the design and check to make sure that the results are correct. A standard VHDL simulator can be used to read the RTL VHDL description and verify the correctness of the design.

The VHDL simulator reads the VHDL description, compiles it into an internal format, and then executes the compiled format using test vectors. The designer can look at the output of the simulation and determine whether or not the design is working properly.

The usual RTL simulation step looks like Figure 11-2.

The designer creates the VHDL as described earlier and compiles the VHDL RTL description to remove any syntax errors. After the syntax

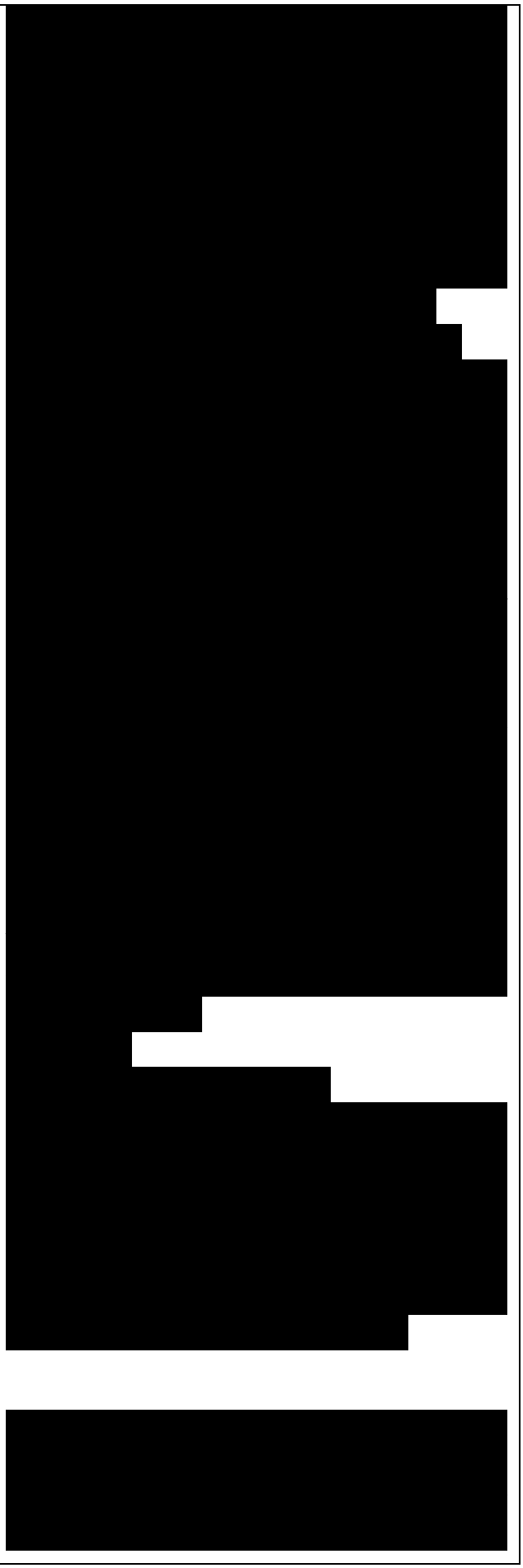
errors have been removed, the design is simulated to verify the correctness of the design. After the simulation has completed, the designer analyzes the results of the simulation to determine if the design is correct or not. If not, the designer must fix the VHDL code and compile and simulate the design again. This process continues until all errors are removed.

The designer loads the compiled VHDL description into the simulator and applies stimulus to the design. This may be a file of input stimulus, a set of commands the designer enters, or an automatic testbench that applies the stimulus and checks the results. (These are discussed in Chapter 14, "RTL Simulation.") After the stimulus has been entered, the designer runs the simulation for as long as needed to generate enough output data to determine if the design is correct. At the beginning of the design process, this may be only a few vectors to make sure that the design resets properly. But later, more and more of the vectors are run as the design starts to function properly.

Figure 11-2
RTL Simulation Flow.

After the simulation has been run, the simulator will have generated output data that can be analyzed. The designer usually has a number of ways to analyze the data. Most common are waveform output and text tabular output. A sample waveform output is shown in Figure 11-3.

A waveform display shows the values of the signals of the design over time. The designer can see the relationships between signal transitions very easily.



Using the waveform display, the designer can determine when system clock edges occur and if the proper signal transitions are present.

The text tabular output is the same data as the waveform display, but in a different format. A sample output is shown in Figure 11-4.

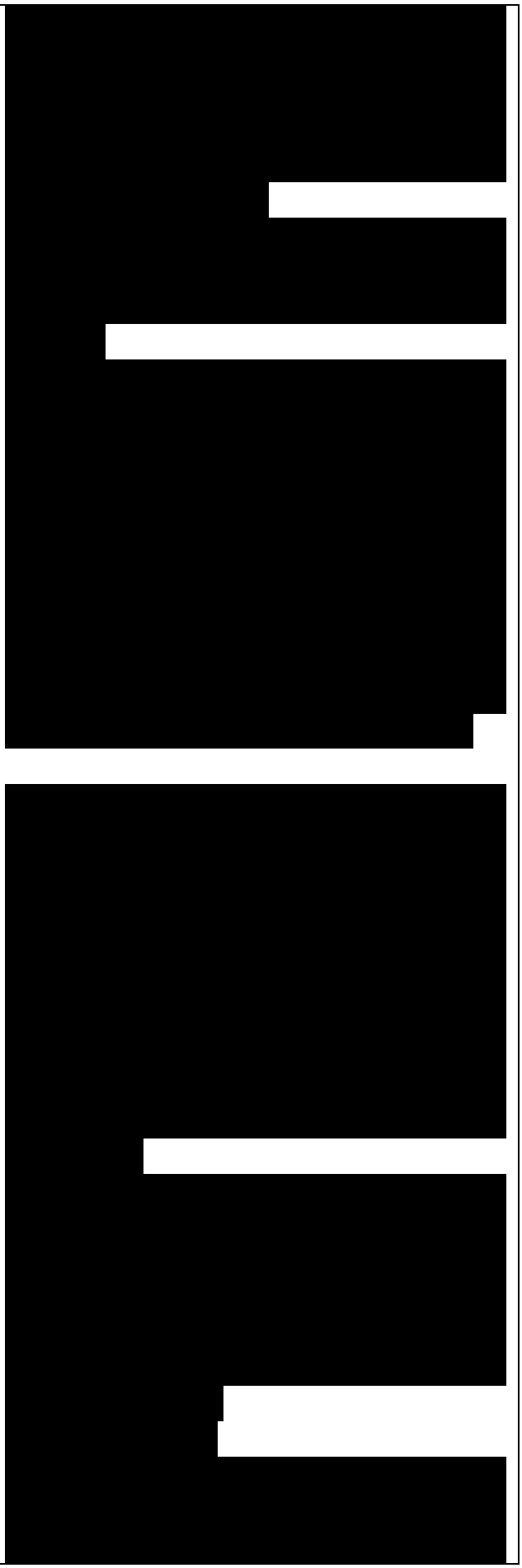
All of the signal transitions are shown from top to bottom instead of left to right. It is also easier to read some of the signal values when the signal has a lot of changes in a short amount of time and the signal values are represented by a number of text characters. Most text table outputs can also filter the output data using a number of different mechanisms such as only on Print on Change or Print on Strobe.

While the output data is being analyzed, the user finds errors in the design description. The user uses the waveform and tabular displays to trace down the source of the errors in the VHDL code, make a change to the VHDL to fix the problem, recompile the design again, and rerun the test. If the problem is fixed, the designer tries to find the next problem, until all problems have been found.

When the designer is happy with the behavior of the design, the designer can start the process of building the real hardware device. To implement the design, the designer uses VHDL synthesis tools. The next step in the process is the VHDL synthesis step.

VHDL Synthesis

The goal of the VHDL synthesis step is to create a design that implements the required functionality and



matches the designer's constraints in speed, area, or power.

The VHDL synthesis tools convert the VHDL description into a netlist in the target FPGA or ASIC technology. For the VHDL synthesis tool to perform this step properly, the VHDL code must be written in a particular style, as discussed in Chapter 10, "VHDL Synthesis."

To synthesize a VHDL description, the designer reads the verified VHDL description into the VHDL synthesis tool in the same way that the designer read the design into the VHDL simulator. The VHDL synthesis tool reports syntax errors and synthesis errors. Synthesis errors usually result from the designer using constructs that are not synthesizable. For instance, access types in VHDL are not synthesizable, because they could specify hardware that is dynamic in nature. Of course, syntax errors result from improper VHDL syntax being read by the VHDL synthesis tool. Presumably, most all of these errors will already have been taken care of because the VHDL code has already been verified with the VHDL simulator. The VHDL synthesis tool also reports warnings of constructs that have the possibility of generating mismatches between the RTL simulation results and the output netlist simulation results.

The designer reads the VHDL design into the VHDL synthesis tool. If there are no syntax errors, the designer can synthesize the design and map the design to the target technology. If the designer had to make changes to the VHDL description, then the VHDL

description needs to be simulated again and the output validated for correctness. First, the designer needs to make sure that the synthesizer is producing an output in the target technology that looks reasonable. The designer looks at the synthesizer output to determine whether or not the synthesizer produced a good result.

The synthesizer produces an output netlist in the target technology and a number of report files. By looking at the netlist, the designer can determine whether or not the design looks reasonable. For most reasonable size designs, however, it can be very difficult to determine how well the synthesizer implemented the function. The designer looks at the report files to determine the quality of the synthesis output. The most common output files are the timing report and the area report. Most synthesis tools produce a number of other reports such as hierarchy reports, instance reports, net reports, power reports, and others. The most useful reports initially are the timing and area reports, because these are usually the most critical factors.

Following is a sample area report:

****
****

Cell: adder View: test Library: work

****
****

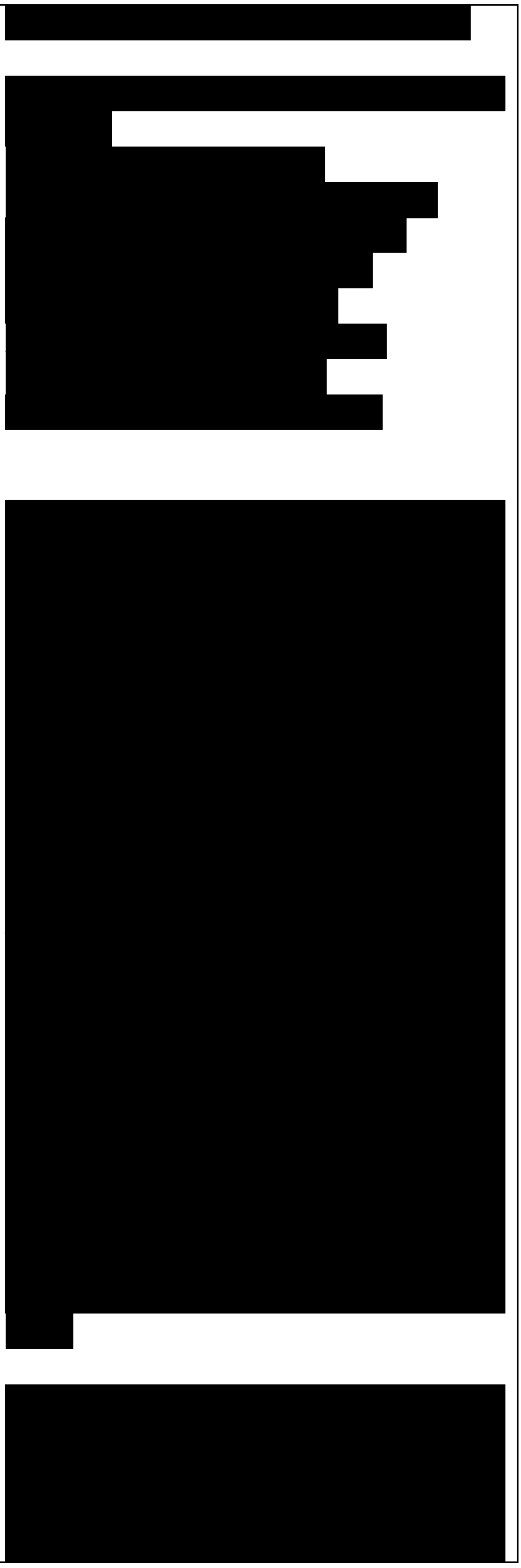
Total accumulated area :
Number of LCs : 8
Number of CARRYs : 7
Number of ports : 24
Number of nets : 107
Number of instances : 91

Number of references to this view : 0

Cell	Library	References	Total Area
GND	flex101	x 1	1 GND
OUTBUF	flex108	x 1	8 OUTBUF
INBUF	flex1016	x 1	16 INBUF
CARRY	flex107	x 1	7 CARRYs
OR2	flex1014	x 1	14 OR2
AND2	flex1021	x 1	21 AND2
LCELL	flex108	x 1	8 LCs
XOR2	flex1016	x 1	16 XOR2

The area report tells the designer the size of the implemented design. The units of measure are determined by the units used when the synthesis library was implemented. For instance, the typical unit for ASIC designs is equivalent 2-input NAND gates, or gate equivalents. Using this measurement, a 2-input NAND gate would consume one gate equivalent, a 2-input AND gate would also consume one gate equivalent. A 4-input NAND gate would consume two gate equivalents. For FPGA designs, equivalent gate measurements vary from manufacturer to manufacturer because each has a different-sized basic cell. In the preceding sample area report, the design produced 8 LC (Logic Cells) and 7 Carry devices. A typical LC is 10 to 12 logic gates; the Carry device is 2 to 3 gates. So, this example would represent about 90 to 120 gates.

The area report shows the designer how much of the resources of the chip the design has consumed. The designer can tell if the design is too big for a particular chip and the



designer needs to target a larger chip, if the design should go into a smaller chip, or if the current chip will work fine. The designer can also get a relative size of the design to use in later stages of the design process.

The timing report shows the timing of critical paths or specified paths of the design. The designer examines the timing of the critical paths closely because these paths ultimately determine how fast the design can run. If the longest path is a timing critical part of the design and is not meeting the speed requirements of the designer, then the designer may have to modify the VHDL code or try new timing constraints to make the path meet timing.

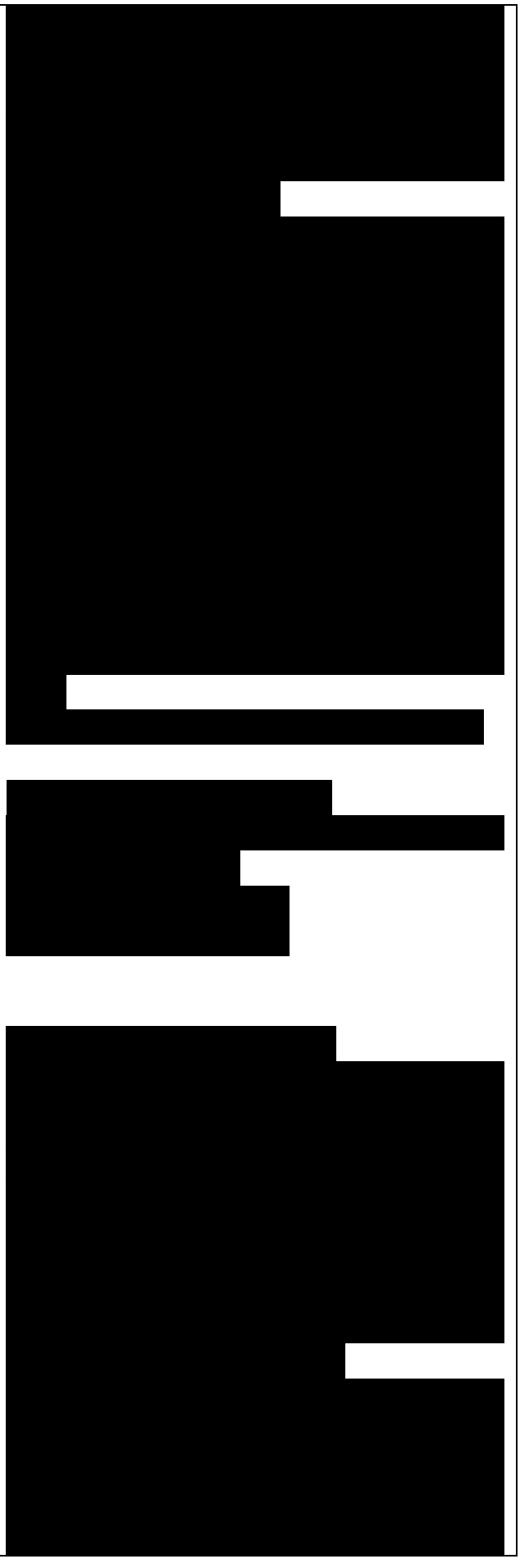
The following is a sample timing report:

Critical Path Report
Critical path #1, (unconstrained path)

NAME GATE ARRIVAL LOAD
NAME GATE ARRIVAL LOAD
data arrival time 13.80

In this report, the worst-case path is listed shown with estimated time values for each node traversed in the design. The timing analyzer calculates the time for a path from an input pin to a flip-flop or output, or from a flip-flop output to a flip-flop input, or output pin.

The designer has the ability to ask for the timing for particular paths of interest, or of the paths that have the longest timing value, and how many to display. As mentioned previously,

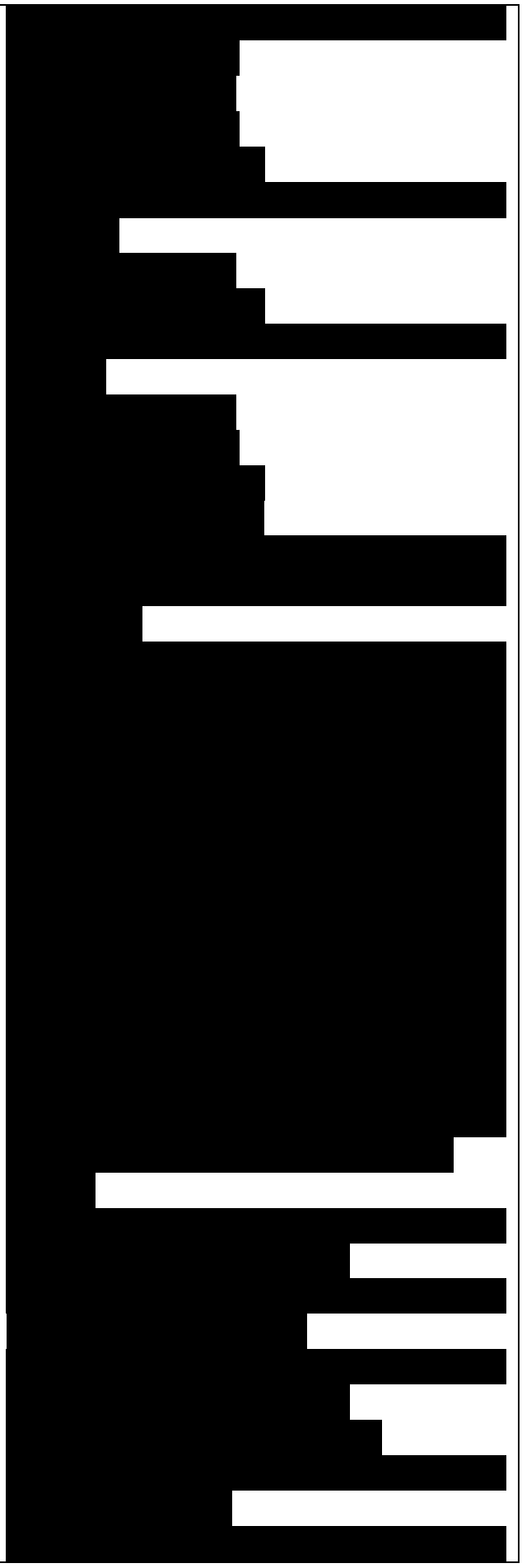


the worst-case paths ultimately determine the speed of the design. For instance, in this case, the worst- case path is 13.8 nanoseconds; therefore, the fastest this design would be able to run is about 72 MHz.

The last type of output data that the designer can examine is the netlist for the design in the target technology. This output is a gate or macro-level output in a format compatible with the place and route tools that are used to implement the design in the target chip. For instance, most place and route tools for FPGA technologies take in an EDIF netlist as an input format. The primitives used in the netlist are those used in the synthesis library to describe the technology. The place and route tools understand what to do with these primitives in terms of how to place a primitive and how to route wires to them. The following example uses a VHDL netlist for ease of understanding. To save space (and boredom), this is not a complete netlist, but gives the reader an idea of how a netlist is structured. The complete netlist can be found on the included CD:

-- Definition of adder
library IEEE, EXEMPLAR; use IEEE.STD_LOGIC_1164.all; use EXEMPLAR.EXEMPLAR_116 4.all;
-- Library use clause for technology cells
library altera ;
use altera.all ;
entity adder is port (
a : IN std_logic_vector (7 DOWNTO 0) ; b : IN std_logic_vector (7 DOWNTO 0) ; c : OUT std_logic_vector (7 DOWNTO 0)) ;
end adder ;

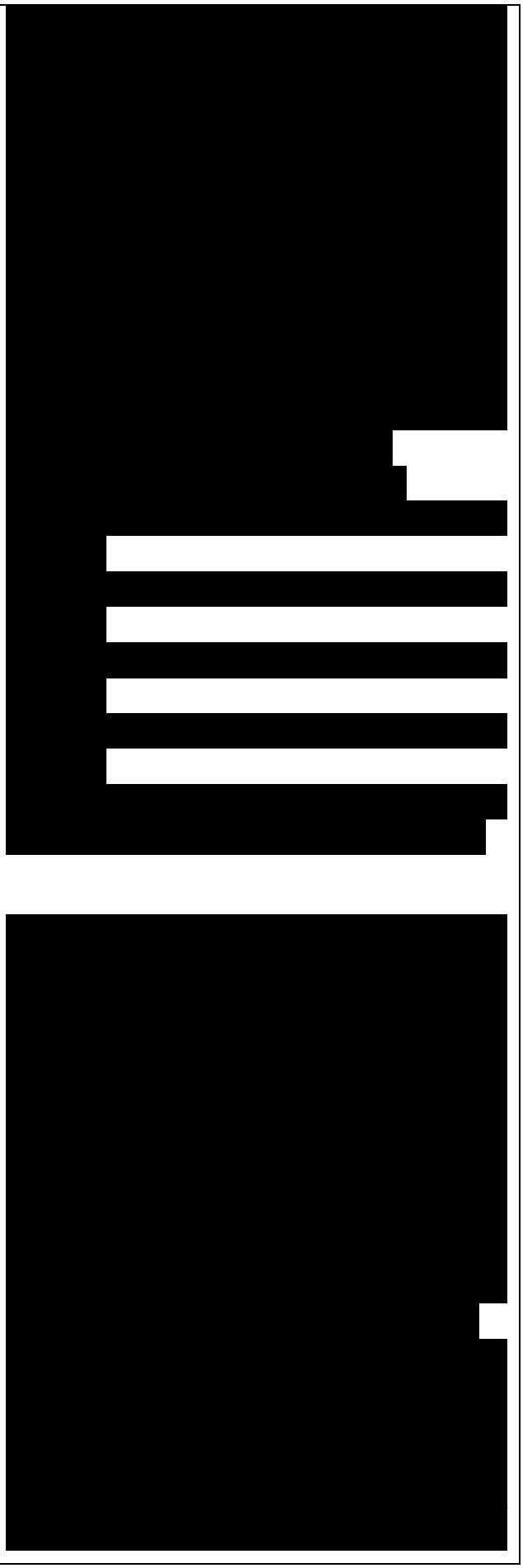
```
architecture test of adder is
component XOR2 port (
Y       : OUT std_logic ;
IN1 : IN std_logic ;
IN2 : IN std_logic) ;
end component ; component LCELL
port (
Y       : OUT std_logic ;
IN1 : IN std_logic) ;
end component ; component AND2
port (
Y       : OUT std_logic ;
IN1 : IN std_logic ;
IN2 : IN std_logic) ;
end component;
signal c_dup0_7, c_dup0_6,
c_dup0_5, c_dup0_4, c_dup0_3,
c_dup0_2,
c_dup0_1, c_dup0_0,
modgen_0_l1_l0_c_int_7,
modgen_0_l1_l0_c_int_6,
modgen_0_l1_l0_c_int_5,
modgen_0_l1_l0_c_int_4,
modgen_0_l1_l0_c_int_3,
modgen_0_l1_l0_c_int_2,
modgen_0_l1_l0_c_int_1,
modgen_0_l1_l0_l0_0_l0_s1,
modgen_0_l1_l0_l0_0_l0_s2,
modgen_0_l1_l0_l0_0_l0_w1,
modgen_0_l1_l0_l0_0_l0_w2,
modgen_0_l1_l0_l0_0_l0_w3,
modgen_0_l1_l0_l0_0_l0_w4,
b_2_int, b_1_int, b_0_int, U_0:
std_logic ;
begin
modgen_0_l1_l0_l0_0_l0_sum0 :
        XOR2 port map  ( Y=>
modgen_0_l1_l0_l0_0_l0_s1,
        IN1=>a_0_int,  IN2=>U_0);
modgen_0_l1_l0_l0_0_l0_sum1 :
        XOR2 port map  ( Y=>
modgen_0_l1_l0_l0_0_l0_s2,
IN1=>modgen_0_l1_l0_l0_0_l0_s1,
IN2=> b_0_int);
```



modgen_0_l1_l0_l0_0_l0_sum2 : LCELL port map (Y=>c_dup0_0, IN1=> modgen_0_l1_l0_l0_0_l0_s2);
modgen_0_l1_l0_l0_0_l0_c0 : AND2 port map (Y=>modgen_0_l1_l0_l0_0_l0_w1, IN1=>a_0_int, IN2=>b_0_int);
modgen_0_l1_l0_l0_0_l0_c1 : AND2 port map (Y=>modgen_0_l1_l0_l0_0_l0_w2, IN1=>a_0_int, IN2=>U_0);
modgen_0_l1_l0_l0_0_l0_c2 : AND2 port map (Y=>modgen_0_l1_l0_l0_0_l0_w3, IN1=>U_0, IN2=>b_0_int);
ix4 3 : OUTBUF port map (\OUT\=>c(3)
ix4 4 : OUTBUF port map (\OUT\=>c(2)
ix4 5 : OUTBUF port map (\OUT\=>c(1)
ix4 6 : OUTBUF port map (\OUT\=>c(0)
U_0_XMPLR : GND port map (Y=>U_0); end test ;

Notice that all of the other interconnect signal names have names such as modgen_0_11_xx or ix123. There is no corresponding signal name in the source file to specify the signal name; therefore, the synthesis tool generates names for these signals. The netlist can be used to figure out how well the synthesizer implemented a part of the design, or to track down a problem net. It can be very useful to find out why a critical path was implemented too slowly.

When the netlist meets the designer's timing, area, power, and other constraints, the next step is to pass the netlist to the gate level simulator. This simulator checks the functionality of the synthesized design.

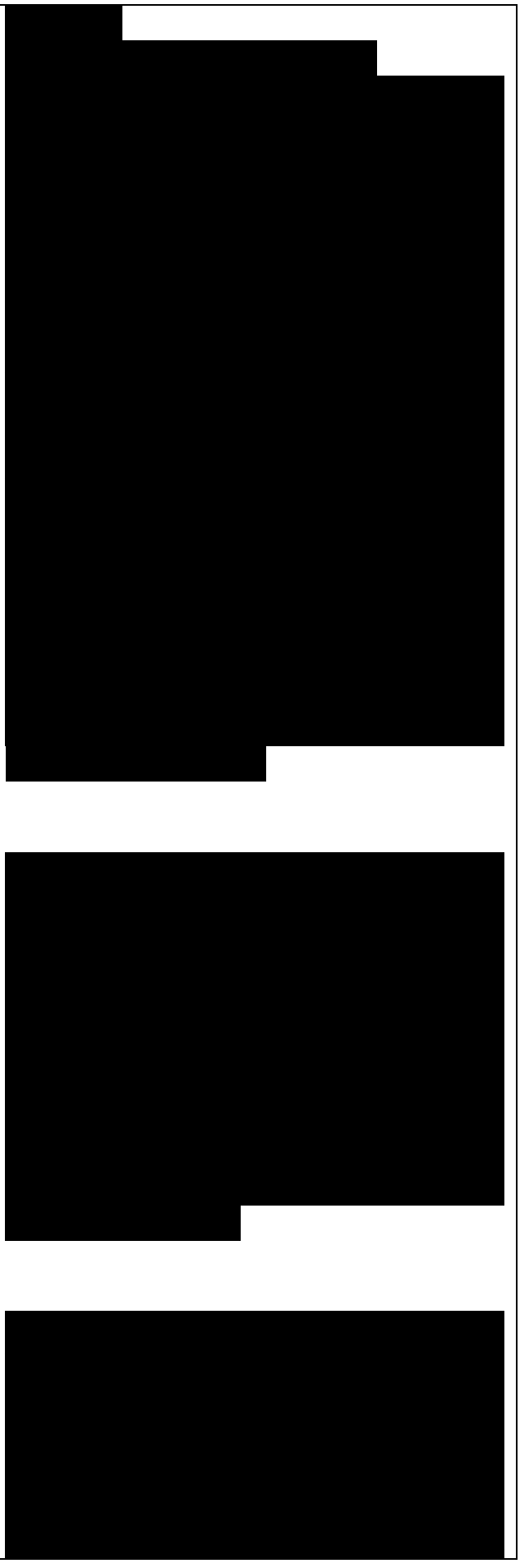


Functional Gate-Level Verification

Some designers might want to do a quick check on the output of the synthesis tool to make sure that the synthesis tool produced a design that is functionally correct. If proper design rules are followed for the input VHDL description, the synthesis tool should never generate an output that is functionally different from the RTL VHDL input, unless the tool has a bug. However, if some of the warnings or errors are ignored or some part of the design is written using a strange VHDL style, the synthesizer can produce an output netlist that does not exactly match the RTL input in terms of functionality. Most designers like to run a quick check on the results of the synthesis tool to make sure the synthesis tool produced a functionally correct output.

To do this, the designer runs a functional gate-level verification. The designer reads the output VHDL netlist from the synthesis tool plus a library of the synthesis primitives into the VHDL simulator and runs the simulation using the RTL verification vectors. If the design matches, then the synthesis tool did not produce logic mismatches; if it does not match, the designer needs to debug the VHDL RTL description to see what is wrong.

The most common method for performing this step is to run a VITAL simulation of the netlist from the synthesis tool. For a completely functional simulation, no timing is back-annotated. If the synthesis tool supports estimated timing and SDF



file generation, the synthesis tool could write the VHDL netlist and an SDF timing file for the design. The designer could use these two files to run a VITAL simulation with estimated timing. After the design has been functionally verified, it is passed to the place and route tools to implement the design.

Place and Route

Place and route tools are used to take the design netlist and implement the design in the target technology device. The place and route tools place each primitive from the netlist into an appropriate location on the target device and then route signals between the primitives to connect the devices according to the netlist. Place and route tools are typically very architecture and device dependent. These tools are tuned to take advantage of each architectural and routing advantage the device contains. FPGA vendors provide these tools because the differences in architectures are large enough that writing a common tool for all architectures would be very difficult. Place and route tools for ASIC devices can be obtained from the ASIC vendor or EDA (Electronic Design Automation) vendors. ASIC architectures do not have as wide a variation between architectures as FPGA architectures and, therefore, place and route tools exist that can handle lots of different ASIC architectures.

Figure 11-5

Place and Route Data Flow.

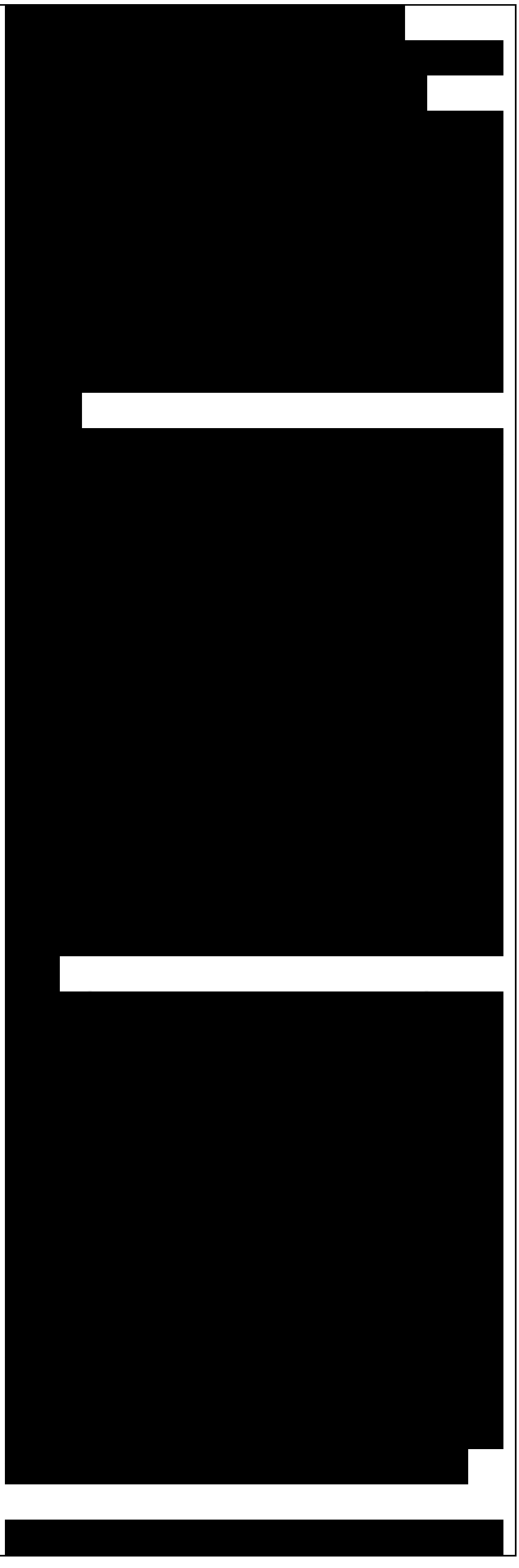
Figure 11-5 shows a dataflow diagram of the place and route tools.

Inputs to the place and route tools are the netlist in EDIF or another netlist format, and possibly timing constraints. The format of the netlist input file varies from manufacturer to manufacturer. Some tools use EDIF; others use proprietary formats such as XNF.

Another input to some place and route tools is the timing constraints, which give the place and route tools an indication about which signals have critical timing associated with them and to route these nets in the most timing efficient manner. These nets are typically identified during the static timing analysis process during synthesis. These constraints tell the place and route tool to place the primitives in close proximity to one another and to use the fastest routing. The closer the cells are, the shorter the routed signals will be and the shorter the time delay.

Some place and route tools allow the designer to specify the placement of large parts of the design. This process is also known as floorplanning. Floorplanning allows the user to pick locations on the chip for large blocks of the design so that routing wires are as short as possible. The designer lays out blocks on the chip as general areas. The floorplanner feeds this information to the place and route tools so that these blocks are placed properly. After the cells are placed, the router makes the appropriate connections.

After all the cells are placed and



routed, the output of the place and route tools consists of data files that can be used to implement the chip. In the case of FPGAs, these files describe all of the connections needed to make the FPGA macrocells implement the functionality required. Antifuse FPGAs use this information to burn the appropriate fuses, while reprogrammable devices download this information to the device to turn on the appropriate transistor connections.

The other output from the place and route software is a file used to generate the timing file. This file describes the actual timing of the programmed FPGA device or the final ASIC device. This timing file, as much as possible, describes the timing extracted from the device when it is plugged into the system for testing. The most common format of this file for most simulators is SDF (Standard Delay Format). Sometimes, proprietary formats are generated and later translated to SDF. SDF is used to back-annotate the post route timing information from place and route tools into the post layout timing simulation.

Post Layout Timing Simulation

After the place and route process has completed, the designer will want to verify the results of the place and route process. There are a number of methods to accomplish this task but the most common is to use post route gate-level simulation. This simulation combines the netlist used for place and route with the timing file from the place and route process into a simulation that checks both functionality and timing of the design.

The designer can run the simulation and generate accurate output waveforms that show whether or not the device is operating properly and if the timing is being met.

If the design has been properly structured, the same test vectors used for the RTL simulation can be used for the post route gate-level simulation. In this way, the designer is saved the process of generating a new set of vectors to check the gate-level design and verifying the new vector output values.

Post route gate-level simulation, if done properly, also uses the same simulator as the RTL simulation. For VHDL simulations, this requires a VITAL-compliant (standard way of describing designs with designs that allow SDF timing back-annotation) VHDL simulator. VHDL simulators that are not VITAL-compliant do not accelerate the execution of the gate-level primitives and cannot accept SDF to back annotate the timing.

Static Timing

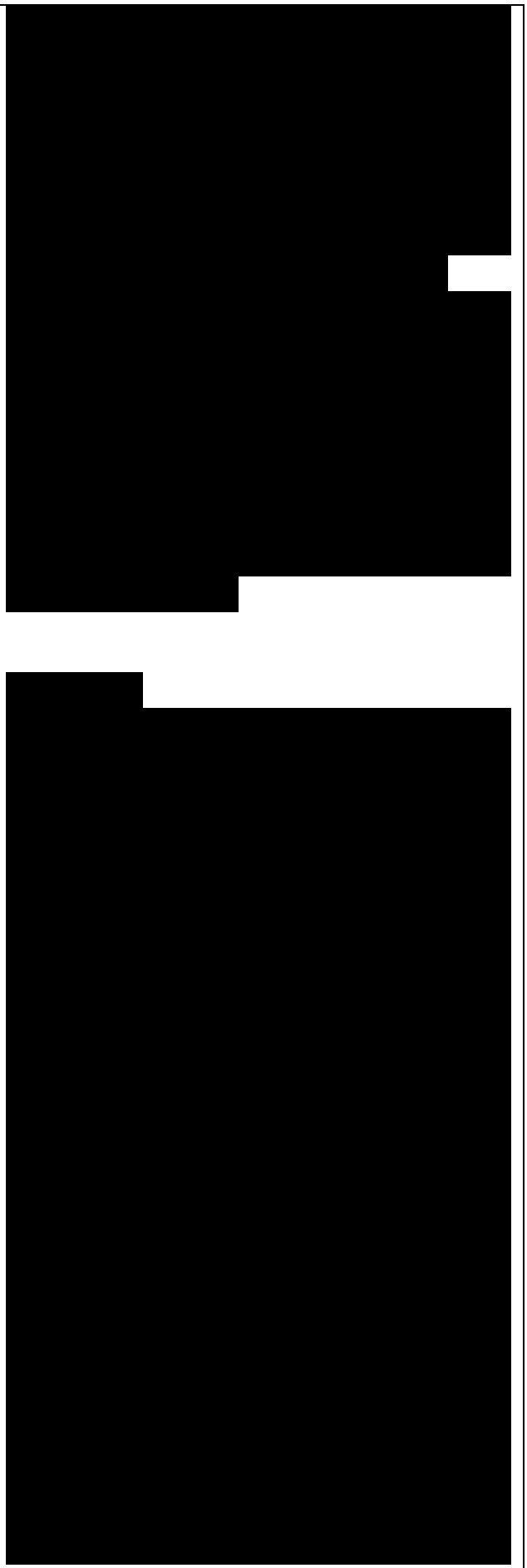
For designs of 10,000 gates to 100,000 gates, post route timing simulation can be a good method of verifying design functionality and timing. However, as designs get larger, or if the designer does not have test vectors, the designer can use static timing analysis to make sure the design meets the timing requirements. A static timing analyzer traces each path in the design and keeps track of the timing from a clock edge or an input. A timing report is then generated in a number of formats. For

instance, the designer can ask for all paths and get an enormous listing of every path in the design. A more intelligent method, however, is to ask for the most timing critical paths in the design and make sure the timing constraints have been met.

Typical static timing analyzers have a number of report types that can be generated so that the designer can make sure the critical paths of the design can be found and verified to be within the required specifications. If paths are not within the specifications, the static timing analyzer shows the entire path so that the designer can try to fix the problem.

SUMMARY

In this chapter, the complete VHDL design process using synthesis was described. This process is very similar no matter which VHDL synthesis or simulation tool is used. The designer must follow a number of steps that add more detail to the design. At each step, the designer has checks to make sure that the correct behavior is being implemented. At the beginning of the process, RTL simulation is used to verify correctness. After synthesis, the netlist, timing report, and area report are all examined to make sure the design fits the designer's constraints. Functional simulation is then run to verify that the synthesis tool produced a functionally correct design. The design is put through the place and route process to implement the design in the target technology. The final check is then to verify using post route gate level simulation that the design is functionally correct and



meets timing.	[redacted]