Chapter 6
EDA Tools

The Electronic Design Automation (EDA), is a group of software tools for designing electronic systems such as integrated circuit (ASICs), printed circuit boards (PCBs), or reprogrammable hardware as FPGA, etc. The general ideas of EDA tools and the particular for FPGA designs will be discussed in this section. Typically these tools work in a design flow that hardware and system designers use to design and analyze entire system behavior. This chapter explains the main concepts related to the EDA tools and presents an example using Xilinx ISE and Altera Quartus tools.

6.1 Design Flow in FPGA EDA Tools

The design flows are the combination of EDA tools to carry out a circuit or system design. Current digital flows are very modular and are the consequence of the evolution of the standalone tools for synthesis, placement, and routing.

Naturally, the tools are evolving, driven by Moore's law from standalone tools to integrated construction and analysis flows for design closure.

The FPGA design flow is a kind of simplification of the ASIC design flow and we will concentrate on it. The FPGA vendors groups the EDA tools in "design suites" such as Libero from Actel [5, 7] or Quartus II from Altera [2]. Figure 6.1 shows a typical design flow; in what follows we will describe the different main stages.

The design flow can be used as command-line executables or scripting, or by using the GUI (*graphical user interface*). Figures 6.5 and 6.14 shows the GUI aspects for Xilinx ISE and Altera Quartus II, respectively. The graphical interface is the preferred design entry for newcomer designers and for small projects.

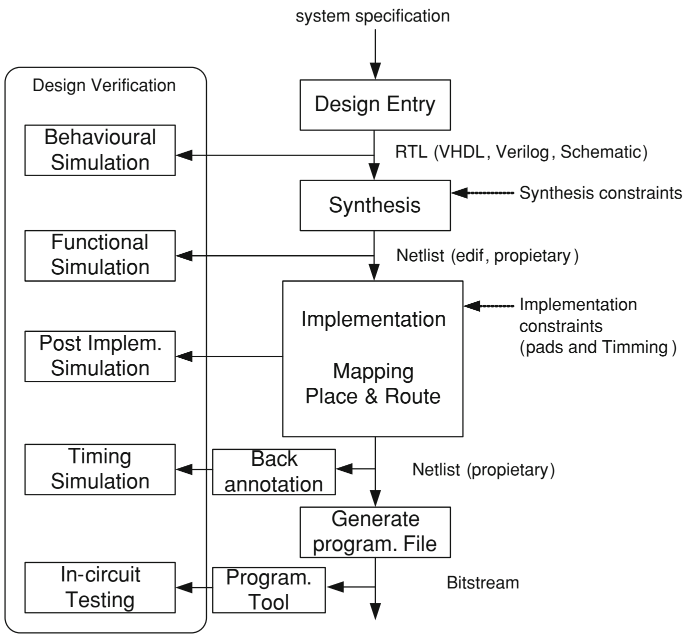


Fig. 6.1 A typical design flow for FPGA design

6.1.1 Design Entry

The design entry is the way you describe the hardware that you want to implement into an FPGA (or in an electronic device) based on a system specification. There are different methods:

- Using a Hardware Description Language (HDL).
- Using schematics.
- Using Intellectual Property (IP) blocks.
- Using Electronic System Level (ESL) languages.

Today's EDA tools allow the mixing of different design entries in a hierarchical structure. It is common to see a schematic top level, with several predesigned IPs, some specific components developed in VHDL and or Verilog and subsystems designed in an ESL language.

6.1.1.1 HDL Design Entry

There are now two industry standard hardware description languages (HDL), Verilog and VHDL. Some vendors used to have their own proprietary HDL languages but these are displaced by the use of the standard languages.

HDL is the de facto design entry in most digital designs. In ASIC designs Verilog is much more used but, FPGA designer's use either VHDL and/or Verilog. All FPGA tools support both languages and even projects mixing the use of booth languages.

The Verilog hardware description language has been used far longer than VHDL and has been used extensively since it was launched by Gateway Design Automation in 1983. Cadence bought Gateway and opened Verilog to the public domain in 1990. It became IEEE standard 1364 in 1995. It was updated in 2001 (IEEE 1364-2001) and had a minor update in 2005 (IEEE 1364-2001).

On the other hand, VHDL was originally developed at the request of the U.S Department of Defense in order to document the behavior of existing and future ASICs. VHDL stand for VHSIC-HDL (Very high speed integrated circuit Hardware Description Language) became IEEE standard 1076 in 1987. It was updated in 1993 (IEEE standard 1076-1993), and the last update in 2008 (IEEE 1076-2008, published in January 2009).

Xilinx ISE and Altera Quartus have text editors with syntax highlighting and language templates for VHDL and Verilog to help in the edition.

6.1.1.2 Schematic Design Entry

With schematic capture or schematic entry you draw on your computer using a schematic capture tool. The main advantage in the use of schematics is that it documents the design in an easily readable format. However big designs rapidly become difficult to maintain and the file formats are incompatible between vendors. A HDL code is easier to be parameterized and regular structures are easily replicated. ASIC and FPGA users rarely use schematics.

6.1.1.3 Intellectual Property (IP) Blocks

In order to make simpler the design of complex systems, there exist libraries of predefined complex functions and circuits that have been tested and optimized to speed up the design process. These predefined circuits are commonly called IP cores (or IP blocks) and are available from FPGA vendors and third-party IP suppliers. The IP Cores can be distributed as a compiled netlist or as an HDL source code. Moreover, the FPGA vendors have tools to generate most typical IP cores (Xilinx CORE generator in Xilinx, Altera megafunctions). The simplest IP cores are typically free, but the complex one rarely is without charge, and typically released under proprietary licenses.

In the FPGA arena a related concept is the idea of "hard IP core" and "soft IP core". In today's FPGA several heterogeneous block are built in the FGPA such as multipliers, blocks of memories, clock managers, transceivers, memory controllers, etc. These components are called "hard IP cores" to differentiate from the ones implemented using general purpose logic (soft IP blocks).

6.1.1.4 Electronic System Level (ESL) Languages

As FPGA applications have grown in complexity and FPGA chips have become more complex, there is an increasing need for more efficient less time-consuming development methodologies. In order to address this requirement, many chip and tool providers have developed “high-level” development tools. In this environment, “high-level” means that the input to the tool is a target independent language as C, C++or Java instead an HDL language.

To refer to these tools and methodologies, several names are used such as High-level synthesis (HLS), High Level Languages (HLL), C synthesis, C to hardware, electronic system level (ESL) synthesis, algorithmic synthesis, or behavioral synthesis and others.

The high-level synthesis seems to be the future design entry for numerous fields of applications. There are several successful products from the big EDA vendors and from small companies fighting in this field. Most of those products generate, as a result, an HDL (Verilog or VHDL) description of the circuit in order to use it in a traditional design flow.

Related to this “high-level” concept and in the DSP (Digital Signal Processing) field, the design productivity of DSP system is increased using MatLab/Simulink design entry. The big EDA companies have tools for DSP design based on MatLab/Simulink, even booth leaders in the FPGA field: Altera (DSP builder) and Xilinx (System Generator).

6.1.2 Synthesis

The synthesis (or logic synthesis) is the process by which an abstract form (an HDL description) of the circuit behavior is transformed into a design implementation in terms of logic gates and interconnections. The output is typically a netlist and various reports. In this context, a “netlist” describes the connectivity of the electronic design, using instances, nets and, perhaps, some attributes.

There are several proprietary netlist formats but, most synthesizers can generate EDIF (Electronic Design Interchange Format) that is a vendor-neutral format to store electronic netlist.

FPGA vendors have their own synthesizers (Xilinx XST, Altera Quartus II Integrated Synthesis) but, main EDA vendors have syntheses for FPGA (Precision by Mentor Graphics and Synplify by Synplicity) that can be integrated in the FPGA EDA tools.

6.1.2.1 Synthesis Optimizations

The synthesis process performs several target independent optimizations (logic simplification, state assignment, etc.) but, also the synthesis tools take into account the target technologies and make target dependent optimizations.

The optimizations available depend on the synthesizer but, most typical optimizations are present in all of them. Typical optimizations are for the area reduction, the speed optimization, the low power consumption, and the target frequency of the whole design.

Nevertheless, much more details can be controlled, such as:

- Hierarchy Preservation: control if the synthesizer flattens the design to get better results by optimizing entities and module boundaries or maintaining the hierarchy during Synthesis.
- Add I/O buffers: enables or disables the automatic input/output buffer insertion: this option is useful to synthesize a part of a design to be instantiated, later on.
- FSM Encoding: selects the Finite State Machine (FSM) coding technique. Automatic selection, one-hot, gray, Johnson, user defined, etc.
- Use of embedded components: use embedded memory or multipliers blocks or use general purpose LUTs to implement these functionalities.
- Maximum fan-out: limits the fan-out (maximum number of connections) of nets or signals.
- Register duplication: allows or limit the register duplication to reduce fan-out and to improve timing.
- Retiming or Register Balancing: automatically moves registers across combinatorial gates or LUTs to improve timing while maintaining the original behavior.

The complete description of synthesis optimization can be found at the synthesis tool documentation (for example for FPGA [3, 4, 6, 8]).

The synthesis behavior and optimization can be controlled using synthesis constraints. The constraints could be introduced using the integrated environment, a constraint file or embedding in the HDL code.

6.1.2.2 Synthesis Constraints

The synthesis optimizations are controlled globally (for the complete design) or locally for each part of the HDL code. The global optimization can be introduced in the integrated environment (then translated as switches in command line or to an additional file), or in a specific constrain file. The local optimization can be specified in a constraint file or embedded in the HDL code.

The synthesis constraint file is, typically, a plain text file having different syntax, depending on the tool. Xilinx use the xcf (Xilinx Constraint File) [8], Altera for timing use SDC files (TimeQuest Timing Constrains) and QSF (Quartus Settings File) for I/O and others [2, 3], and simplify uses SDC (Synopsys Design Compiler) constraint files [6].

The main advantage of using constraint files (generated by a graphical interface or entered in a text editor) is that it makes your source code more portable and separates the functionality described in the HDL of the synthesis detail.

The HDLs (VHDL and Verilog) allow you to embed constraints in the design code. This method is recommended for constraints that define the desired result of the HDL (encoding of FSM, type of memories or multipliers) or specific optimization (duplicate registers, registers balancing, maximum fan-out, etc.). In VHDL this is done using “attributes” defined in the declaration. The following lines of code define the maximum fan-out of signal “a_signal” to 20 connections in XST [8].

```
attribute max_fanout: string;
attribute max_fanout of a_signal: signal is "20";
```

6.1.2.3 Synthesis Reports

The synthesis reports show the results of the netlist generation synthesis process. The synthesizer gives, typically, a text report, where you can see a summary of your synthesis options, and a summary and analysis of the netlist generation. Some tools generate an HTML, XML or other proprietary formats which are easier to navigate into the information.

This report is important in order to see if the hardware generated by the synthesizer agrees with the described HDL. The main parts in a synthesis report are:

- Synthesis Options: a summary of selected optimizations used. Important to check if differences in results appear.
- HDL Compilation and Analysis: syntax errors and hierarchy are analyzed and information reported.
- HDL Synthesis: is the main part, where the inferred hardware is reported. The tool informs what is generated in each part of the code.
- Optimization reports: the advanced optimization and low level optimization are also informed.
- Final resource and timing: a summary of the device resource utilization and a preliminary timing analysis with the worst path is informed. Take into account that this information is previous to the implementation (placement and routing) and the interconnection delay is only estimated.

Additionally, most synthesis tools can generate graphical information (a schematic) of the resulted synthesis. Typically, you can see graphically the RTL view or a technological view (using the target low level components).

6.1.3 Implementation (Mapping, Placement and Routing)

The implementation step is a vendor specific tool that mainly places and routes the design in the target device. In Altera Quartus II this tool is known as a “fitter” [2, 3], meanwhile, in Xilinx ISE [7, 8] is composed of three processes (translate, map, and place & route).

The inputs to the implementation are the netlist(s) generated in synthesis and the design implementation constraints. The output is a proprietary placed and routed netlist (ncd file in Xilinx, an internal database representation in Altera) and several reports summarizing the results. Typically the design implementation uses timing and area constraints; a general description is in Sect. 6.2.

6.1.3.1 Implementation Reports

The implementation tools generate different information about that process. Typically, a text report includes detailed information about the used resources, the clock distribution (including skew), the final timing obtained with respect to the constraints and other information.

Nevertheless, more information can be obtained of the implementation, such as internal delay of the interconnections, a description of the used pads, a graphical view of the placed and routed design in the target device, a simulatable HDL file including timing information, and a power consumption estimation.

6.1.4 Programming File Generation and Programming

The programming file generation creates a bitstream file (.bit in Xilinx,.rbf in Altera) that can be downloaded to the device or can be recorded in an external EPROM. This tool in Altera Quartus II is call "assembler", meanwhile, in Xilinx ISE "Bitgen".

The generated bitstream can be converted in a standard format for EPROM programming or directly downloaded to an FPGA device using a JTAG connection (iMPACT in Xilinx, Programmer in Altera).

6.2 Implementation Constraints

The complexity of today's FPGA designs and the demand for higher performance makes necessary the use of complex timing and placement constraints to meet the performance requirements. Implementation constraints are instructions given to the FPGA implementation tools to direct the mapping, placement, routing, timing or other guidelines for the implementation tools.

Implementation constraints are placed in constraint file(s) but, may exist in the HDL code, or in a synthesis constraints file and propagated for implementation. Xilinx uses the User Constraint File (.ucf), meanwhile Altera uses the Quartus II Settings File (.qsf) or, in the case of timing constraints, the Synopsys Design Constraints file (.sdc). The constraint files are plain text but, there are several graphical tools that help in the edition of such constraints avoiding the necessity to know the exact syntax.

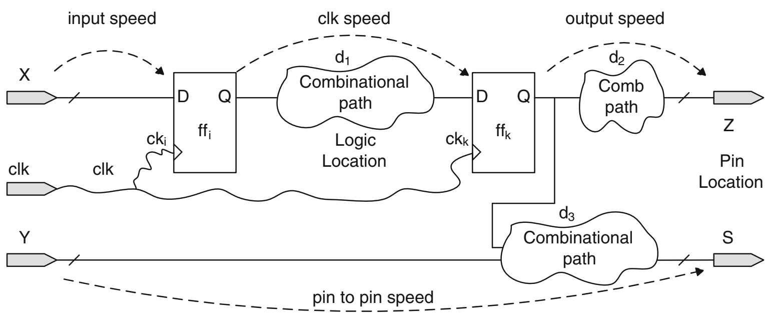


Fig. 6.2 What is possible to be constrained

Figure 6.2 shows the basic constraints of a digital synchronous design:

- Internal clock speed for one or several clocks.
- I/O speed.
- Pin to Pin timing.
- Pin Locations and Logic Locations (floor-planning).

The first three are timing constraints, meanwhile, the last are area constraints.

6.2.1 Timing Constrains

As suggested previously (Fig. 6.2) the timing constrain includes the clock definition, input and output timing requirements and the combinatorial path requirements. Creating global constraints for a design is the easiest way to provide coverage of the constrainable connections in a design, and to guide the tools to meet timing requirements for all paths. For example given a constraint of frequency of 100 MHz, we are constraining each combinational path in the design.

Nevertheless sometimes the designer needs to relax some global constraint and inform the implementation tools that some path can take more time. The typical cases are:

- False Paths: if there is a paths that is static in nature, or is not of much significance for timing.
- Multi-cycle paths: paths between registers (combinational path) that intentionally take more than one clock cycle to become stable.
- Fast or Slow path: combinational path that can work at slower speed than the global constrain.

In these cases you can use directives to eliminate the paths from timing consideration in implementation.

6.2.2 Placement and Other Constrains

The placement constraints instruct the implementation tools, where to locate the logic element into the FPGA (I/O pads, Flip-flops, ROMs, RAMs, LUTs, etc.). Since every component in the design carries a unique name, you can use these name to assign to a region in the FPGA where put this components. A representative placement constraint, always used in a design, is the position of the input/output pins. Nevertheless, there are others commonly used constraints:

- Relative placement constraints: allow to place logic blocks relative to each other to increase speed and use die resources efficiently.
- Routing constraints: mechanism of locking the routing in order to maintain timing.
- Input/output characteristics: specifies the i/o standard, the use of internal pull-ups or pull-downs, slew rate, asynchronous delay, etc.
- Dedicated block configuration: how the multipliers are configured, digital clock managers, memory blocks, SERDES, etc.
- Mapping Directives: allow eliminating the simplification of internal nets for observability, or how the logic is grouped.
- Maximun Skew: allow to control the maximum skew in a line (Sect. 5.3.1).
- Derating Factors: specifies different supply voltage and temperature. Used to derate the timing factors (Sect. 5.1.2.5).

As previously mentioned, implementation constraints (timing and placement) are placed in separated constraint file(s) but, may be directly written in the HDL code.

6.3 System Verification

Figure 6.3 shows the typical system verification flow for FPGA. The logic simulation is the most used technique in early stage of development. Nevertheless, the reprogrammable nature of FPGA gives other alternatives as in-circuit simulation, testing and debugging.

6.3.1 Simulation

Logic simulation is the primary tool used for verifying the logical correctness of a hardware design. In many cases, logic simulation is the first activity performed in the process of design. There are different simulation points were you can simulate your design, the three more relevant are:

- RTL-level (behavioral) simulation. No timing information is used.
- Post-Synthesis simulation. In order to verify synthesis result.

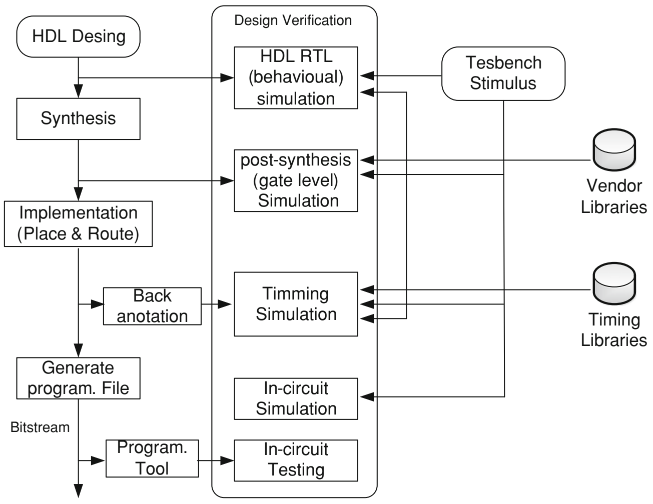


Fig. 6.3 The design verification flow in FPGAs

- Post implementation (post place & route) simulation. Also known as timing simulation because it includes blocks and nets delays.

The behavioral (RTL-level) simulation enables you to verify or simulate a description at the system level. This first pass simulation is typically performed to verify code syntax, and to confirm that the code is functioning as intended. At this step, no timing information is available, and simulation is performed in unit-delay mode to avoid the possibility of a race condition. The RTL simulation is not architecture-specific, but can contain instantiations of architecture-specific components, but in this case additional libraries are necessary to simulate.

Post-synthesis simulation, allows you to verify that your design has been synthesized correctly, and you can be aware of any differences due to the lower level of abstraction. Most synthesis tools can write out a post-synthesis HDL netlist in order to use a simulator. If the synthesizer uses architecture-specific components, additional libraries should be provided.

The timing simulation (post implementation or post Place & Route full timing) is performed after the circuit is implemented. The general functionality of the design was defined at the beginning but, timing information cannot be accurately calculated until the design has been placed and routed.

After the implementation tools, a timing simulation netlist can be created and this process is known as back annotation. The result of a back annotation is an HDL file describing the implemented design in terms of low level components and additional SDF (Standard Delay Format) file with the internal delays that allows you to see how your design behaves in the actual circuit.

Xilinx ISE has his own simulator (ISE simulator, ISim) but, can operate with Mentor Modelsim or Questa and operate with external third-party simulation tools

(Synopsys VCS-MX, Cadence NCSim, Aldec Active-HDL). Altera Quartus II uses Mentor Modelsim and can use other third party simulators.

6.3.2 Formal Verification

The formal verification is the act of proving the correctness of a system with respect to a certain formal specification, using formal mathematical methods. Since hardware complexity growth continues to follow Moore's Law, the verification complexity is even more challenging and is impossible to simulate all possible states in a design. In order to implement the formal verification, Hardware Verification Language (HVL) can be used. A HVL is a programming language used to verify the designs of electronic circuits written in a hardware description language (HDL). System-Verilog, OpenVera, and SystemC are the most commonly used HVLs. The formal verification is widely used by the big companies in the ASIC world but, is relatively new in the FPGA arena. The adoption of formal verification in FPGA flow is still poor but, increasingly important.

6.3.3 In-Circuit Co-Simulation

The idea is to simulate the system at hardware speed but, maintaining a part of the flexibility of a traditional simulation. The simulation executes the RTL code serially while a hardware implementation executes it fully in parallel. This leads to differences not only in execution time but, also in debugging. In simulation, the user can stop simulation to inspect the design state (signals and memory contents), interact with the design, and resume simulation. Downloading the design to an FPGA, the visibility and observability is greatly reduced.

EDA vendors offer products to simulate the complete or a part of the design in circuits but, controlling externally the execution. In this line, Xilinx added their simulator (Isim), the so called "Hardware co-simulation", as a complementary flow to the software-based HDL simulation. This feature allows the simulation of a design or a sub-module of the design to be offloaded to hardware (a Xilinx FPGA regular board). It can accelerate the simulation of a complex design and verify that the design actually works in hardware.

6.3.4 In-Circuit Testing and Debugging

In order to test the functionality, a simple configuration of the FPGA allows testing of the circuit. In the typical in-circuit debugging, the user employs an external logic analyzer for visibility but, can see only a limited number of signals which they determined ahead of time.

The reprogramability of FPGA opens new ways to debug the designs. It is possible to add an "internal logic analyzer" within the programmed logic. Xilinx ChipScope Pro Analyzer and Altera SignalTap II Logic Analyzer are tools that allow performing in-circuit verification, also known as on-chip debugging. They use the internal RAM to store values of internal signals and communicate to the external world using the JTAG connection.

Another intermediate solution includes inserting "probes" of internal nets anywhere in the design and connecting of the selected signals to unused pins (using Xilinx FPGA Editor, Altera Signal Probe, Actel Designer Probe Insertion).

6.3.5 Design for Test

A recurring topic in digital design is the design for test (DFT). The DFT (or also Design for Testability) is the general name for design techniques that add certain testability features to a hardware design.

In the ASIC world, tests are applied at several stages in the manufacturing flow. The tests usually are accomplished by test programs that execute in Automatic Test Equipment (ATE). For test program generation, several automatic algorithms are used as the "Stuck-at" fault model and other algorithmic methods.

One of the main issues in a test is to gain control (controllability) and observe (observability) internal nodes in order to check functionality and this leads to the concept of scan chain.

In scan chain, registers (flip-flops or latches) in the design are connected in a shift register chain in order to set the vales or read the values when a test mode is selected. Figure 6.4 shows the addition of a multiplexer to a simple register to support the scan chain mode. Observe that only using four signals (clk, Sin, Sout, test) and an option reset it is possible to gain access to any register inside a device.

The FPGA are pretested ASIC circuits that have their own circuitry to test the manufacturing procedure. Nevertheless, some techniques of DFT are useful for FPGA design debugging and testing. The main useful characteristic is the access to the internal state of the FPGA using the internal scan chain through the JTAG port. JTAG (Joint Test Action Group) is the name for standard test access port and boundary-scan architecture. It was initially devised for testing printed circuit boards using boundary but, today it is also widely used for integrated circuit debug ports.

The FPGA devices allow the accessing of the internal state (read-back) of any internal register, even the configuration ones using the JTAG port. The possibility to read and write the internal state of the device allows several standard and ad-hoc techniques for testing purposes. The access to the entire registers content allows, for example, the reconfiguration of the device, the partial reconfiguration, read internal states for in-circuit debugging (Sect. 6.3.4), communicate internal values to implement an internal logic analyzer, and several other techniques.

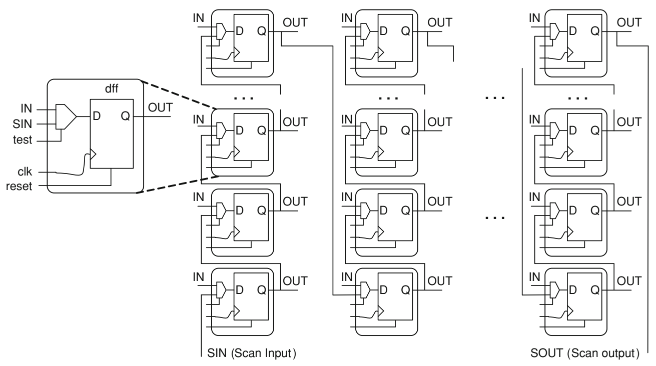


Fig. 6.4 Scan chain register and organization of a scan chain

6.4 Timing Analysis

Static timing analysis (STA or simply timing analysis) is the method of computing the expected timing of a digital circuit without requiring simulation. The word *static* refers to the fact that this timing analysis is carried out in an input independent manner. The objective is to find the worst case delay of the circuit over all possible input combinations.

The computational task is to review the interconnection graph that represents the final netlist and determine the worst case delay. The methods used are specific optimization of typical graph algorithm such as a depth-first search. Modern static timing analyzers bear similarities with project management models such as PERT (Program Evaluation and Review Technique) or CPM (Critical Path Method).

There are some common terms used in timing analysis:

- **Critical path**: is the path with the maximum delay between an input and an output (or two synchronous points).
- **Arrival time**: is the time elapsed for a signal to arrive at a certain point.
- **Required time**: is the latest time at which a signal can arrive without making the clock cycle longer than desired (or a malfunction).
- **Slack**: is the difference between the required time and the arrival time. A negative slack implies that a path is too slow. A positive slack at a node implies that the arrival time at that node may be increased without affecting the overall delay of the circuit.

In a synchronous digital system, data is supposed to move on each tick of the clock signal (Sect. 5.2). In this context, only two kinds of timing errors are possible (see Sect. 5.2.2):

- **Hold time violation**: when an input signal changes too fast, after the clock's active transition (race conditions).
- **Setup time violation**: when a signal arrives too late to a synchronous element and misses the corresponding clock's edge (long path fault).

Since the timing analysis is capable of verifying every path, it can detect the problems in consequence of the clock skew (Sect. 5.3) or due to glitches (Sect. 5.1.3).

The timing analysis can be performed interactively, asking for different paths but, typically is used to report the slack upon the specified timing requirements expressed in the timing constraint (Sect. 6.2.1).

The FPGA tools, after implementing the design, make a default timing analysis to determine the design system performance. The analysis is based on the basic types of timing paths: Clock period; input pads to first level of registers; last level of registers to output pads, and pad to pad (in asynchronous paths).

Each of these paths goes through a sequence of routing and logic. In Xilinx ISE the tool calls "Timing Analyzer" and in Altera Quartus II Altera "TimeQuest". More advanced options can be analyzed upon using the specific tool.

6.5 Power Consumption Estimation

Power dissipated in a design can be divided into static and dynamic power (Sect. 5.3). In FPGA designs, due to the reprogramability nature, the total power is also divided into three components for each power supply:

$$\text{Total Power} = \text{Device Static} + \text{Design Static} + \text{Dynamic Power}$$

Where the components are:

- Device Static: depends on manufacturing, process properties, applied voltage, and temperature.
- Design Static: blocks in an FPGA (I/O termination, transceivers, block RAM, etc.) are disabled by default and enabled depending on the design requirements. When these blocks are enabled they consume power, regardless of user design activity.
- Dynamic Power: depends on the capacitance and activity of the resources used, and also scales with the applied voltage level.

FPGA vendors offer early power estimators spreadsheet (Altera PowerPlay Early Power Estimator, Xilinx XPower Estimator) typically used the pre-design and pre-implementation phases of a project. These spreadsheets are used for architecture evaluation; device and power supply selection and helps to choose the thermal management components which may be required for the application.

For a more accurate estimation, the power analyzer tools (Xilinx XPower, Altera PowerPlay) perform power estimation, post implementation. They are more precise tools since they can read from the implemented design netlist the exact logic and routing resources used. In order to obtain good and reliable results the activity of each node should be provided. If you do not supply the activity, the software can predict the activity of each node but, the accuracy is degraded.

The power analyzer takes the design netlist, the activity of the circuit, the supply voltage and the ambient temperature and reports the consumed current (power) and the junction temperature. Nevertheless, the junction temperature itself depends on the ambient temperature, voltage level, and total current supplied. But the total current supplied includes the static current as a component that depends on temperature and voltage, so a clear circular dependency exists. The tools use a series of iterations to converge on an approximation of the static power for given operating conditions.

A significant test bench that models the real operation of the circuit provides the necessary activity of each node. The simulation result of the activity is saved in a file (SAIF or VCD file) that is later used in conjunction with the capacitance information by the power analyzer.

The Value Change Dump (VCD) file is an ASCII file containing the value change details for each step of the simulation and can be generated by most simulators. The computation time of this file can be very long, and the resulting file size is typically huge. On the other hand, the SAIF (Switching Activity Interchange format) file contains toggle counts (number of changes) on the signals of the design and is supported also for most simulators. The SAIF file is smaller than the VCD file, and recommended for power analysis.

6.5.1 Reducing the Power Consumption

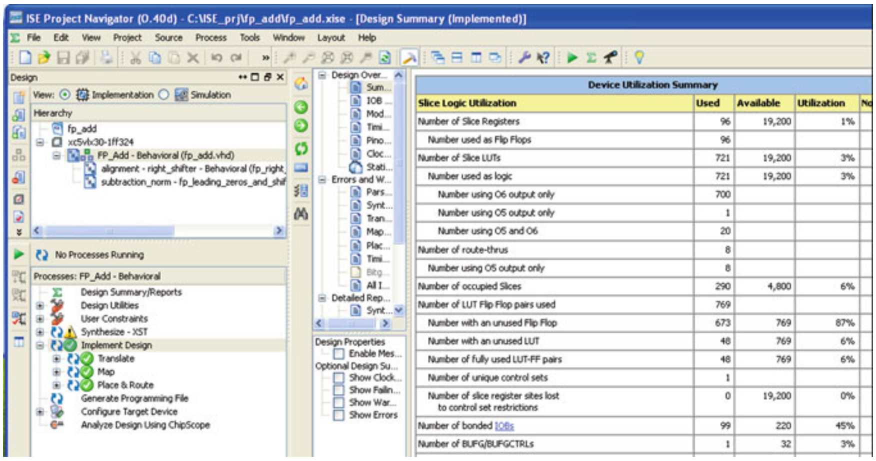
Using the results of a power analyzer, having a complete system-level understanding and the accurate power model will permit the designer to make the decisions necessary to reduce the power budget (Sect. 5.3), including:

- Selecting the best device.
- Reducing the device operating voltage.
- Optimizing the clock frequencies.
- Reducing long routes in the design.
- Optimizing encodings.

With regards to the EDA automatic improvements for low power, as mentioned previously, in synthesis it is possible to have, as a target, the power reduction. In implementation, it is possible to specify optimal routing to reduce power consumption. In this case, it allows to specify an activity file (VCD or SAIF) to guide place & route when it optimizes for power reduction.

6.6 Example of EDA Tool Usage

In order to review the concepts of EDA tools we will deploy a simple example using Xilinx ISE 13.1 and Altera Quartus 11.0. The example is the combinational


ISE Project Navigator (O.40d) - C:\ISE_prj\fp_add\fp_add.xise - [Design Summary (Implemented)]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
fp_add
xc5vlx30-1ff324
FP_Add - Behavioral (fp_add.vhd)
alignment - right_shifter - Behavioral (fp_right_
subtraction_norm - fp_leading_zeros_and_shif
No Processes Running
Processes: FP_Add - Behavioral
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Translate
Map
Place & Route
Generate Programming File
Configure Target Device
Analyze Design Using ChipScope
Design Over...
Errors and W...
Detailed Rep...
Design Properties
Enable Mes...
Optional Design Su...
Show Clock...
Show Failin...
Show War...
Show Errors

Device Utilization Summary

Slice Logic Utilization	Used	Available	Utilization	No
Number of Slice Registers	96	19,200	1%	
Number used as Flip Flops	96			
Number of Slice LUTs	721	19,200	3%	
Number used as logic	721	19,200	3%	
Number using O6 output only	700			
Number using O5 output only	1			
Number using O5 and O6	20			
Number of route-thrus	8			
Number using O5 output only	8			
Number of occupied Slices	290	4,800	6%	
Number of LUT Flip Flop pairs used	769			
Number with an unused Flip Flop	673	769	87%	
Number with an unused LUT	48	769	6%	
Number of fully used LUT-FF pairs	48	769	6%	
Number of unique control sets	1			
Number of slice register sites lost to control set restrictions	0	19,200	0%	
Number of bonded IOBs	99	220	45%	
Number of BUFG/BUFGCTRLs	1	32	3%	

Fig. 6.5 A simple project in Xilinx ISE

floating point adder of Chap. 12 that we will implement in both vendor tools. The simple example uses three VHDL files: *FP_add.vhd*, *FP_right_shifter*, and *FP_leading_zeros_and_shift.vhd*.

If you want to begin with no previous experience, it is highly recommendable to start with the tutorial included in the tools. In the case of Quartus, follow the "Quartus II Introduction for VHDL/verilog Users" accessible form help menu. If you will start using ISE we recommend using the "ISE In-Depth Tutorial", accessible using help → Xilinx on the web → Tutorials.

6.6.1 Simple Example Using Xilinx ISE

We create a new project (file → new project…) with name fp_add. We select a Virtex 5, XC5VLX30 ff324 -1. For synthesis XST, simulation ISIM and preferred language VHDL. Then we add to the project the three VHDL source files (project → add source) (Fig. 6.5).

6.6.1.1 Design Entry and Behavioral Simulation

The design entry in this simple project is not used, since the VHDL code is provided. Nevertheless to create a new source file you can use "project → new source", and then choose the design entry type of source code. In the case of using VHDL or Verilog module you have a graphical wizard to describe the input and output of the circuit.

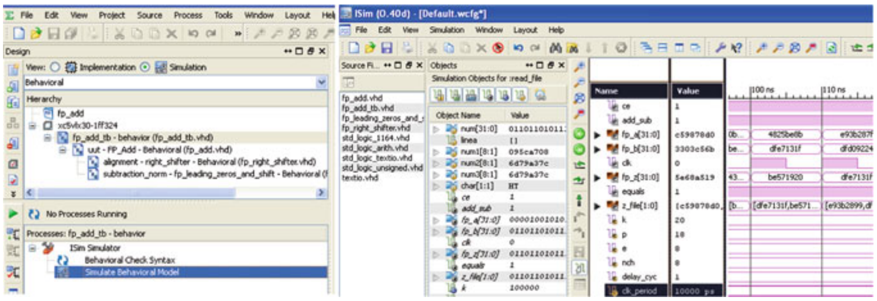
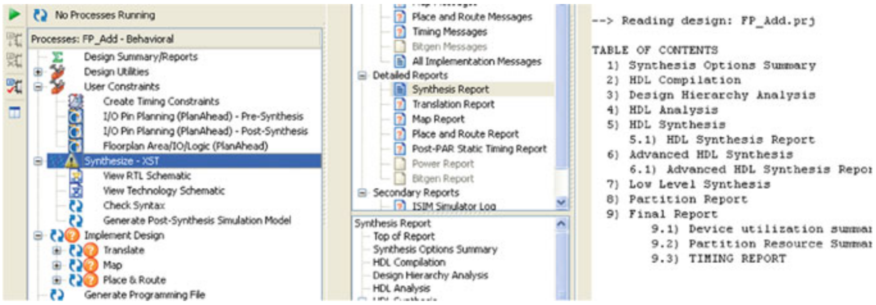

File Edit View Project Source Process Tools Window Layout
ISim (O.40d) - [Default.wcfg*]
Design
View: Implementation Simulation
Behavioral
Hierarchy
fp_add
xc5vlx30-1ff324
fp_add_tb - behavior (fp_add_tb.vhd)
uut - FP_Add - Behavioral (fp_add.vhd)
alignment - right_shifter - Behavioral (fp_right_shifter.vhd)
subtraction_norm - fp_leading_zeros_and_shift - Behavioral
No Processes Running
Processes: fp_add_tb - behavior
ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model
fp_add.vhd
fp_add_tb.vhd
fp_right_shifter.vhd
std_logic_1164.vhd
std_logic_arith.vhd
std_logic_textio.vhd
std_logic_unsigned.vhd
textio.vhd
Objects
Simulation Objects for read_file
Object Name
Value
Name
Value
100 ns
110 ns

Fig. 6.6 Behavioral simulation in Xilinx ISE using ISIM simulator



```
--> Reading design: FP_Add.prj

TABLE OF CONTENTS
  1) Synthesis Options Summary
  2) HDL Compilation
  3) Design Hierarchy Analysis
  4) HDL Analysis
  5) HDL Synthesis
     5.1) HDL Synthesis Report
  6) Advanced HDL Synthesis
     6.1) Advanced HDL Synthesis Repo
  7) Low Level Synthesis
  8) Partition Report
  9) Final Report
        9.1) Device utilization summa
        9.2) Partition Resource Summa
        9.3) TIMING REPORT
```

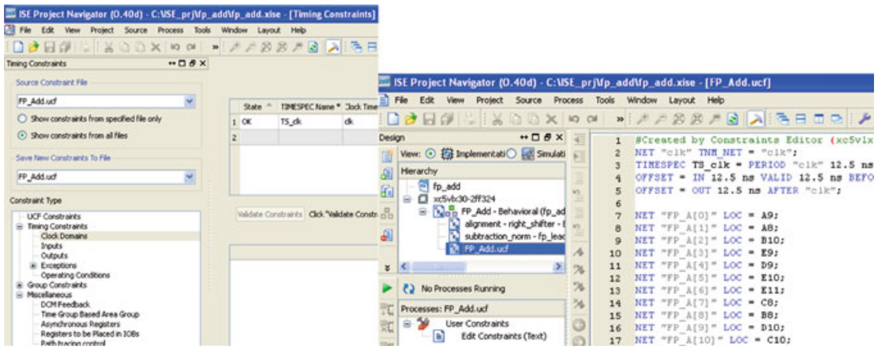
Fig. 6.7 Synthesis and synthesis report

In order to make a behavioral simulation we add to the project a test bench (fp_add_tb.vhd) selecting "project → add source". The simple test bench reads the stimuli text file (contains two encoded operands to add and the corresponding result) and verifies the results (Fig. 6.6).

In order to launch the simulation, select the test bench (fp_add_tb.vhd), and double click "Simulate Behavioral Model" in the processes window. Before that, ensure that you have selected the "simulation" view radio button, in order to be able to see the simulation process.

6.6.1.2 Synthesis and Synthesis Report

To run the synthesis, double click the "synthesis—XST" process in processes view (Fig. 6.7). The global options of the synthesis are available using the right button at the option "process option". The more specific constrain can be either embedded in the HDL code or in the Xilinx constraint file (.xcf). The xcf is a plain text file that needs to be linked in the process option. The details are available at [8].


ISE Project Navigator (0.40d) - C:\ISE_prj\fp_add\fp_add.xise - [Timing Constraints]
File Edit View Project Source Process Tools Window Layout Help
Timing Constraints
Source Constraint File
FP_Add.ucf
Show constraints from specified file only
Show constraints from all files
Save New Constraints To File
FP_Add.ucf
Constraint Type
UCF Constraints
Timing Constraints
Clock Domains
Inputs
Outputs
Exceptions
Operating Conditions
Group Constraints
Miscellaneous
DCM Feedback
Time Group Based Area Group
Asynchronous Registers
Registers to be Placed in IOBs
Path tracing control
State
TIMESPEC Name
Clock Time
OK
TS_clk
clk
Validate Constraints
Click "Validate Constr
ISE Project Navigator (0.40d) - C:\ISE_prj\fp_add\fp_add.xise - [FP_Add.ucf]
Design
View: Implementation Simulation
Hierarchy
fp_add
xc5vlx30-2ff324
FP_Add - Behavioral (fp_ad
alignment - right_shifter - (
subtraction_norm - fp_lead
FP_Add.ucf
No Processes Running
Processes: FP_Add.ucf
User Constraints
Edit Constraints (Text)

```
#Created by Constraints Editor (xc5vlx
NET "clk" TNM_NET = "clk";
TIMESPEC TS_clk = PERIOD "clk" 12.5 ns
OFFSET = IN 12.5 ns VALID 12.5 ns BEFO
OFFSET = OUT 12.5 ns AFTER "clk";

NET "FP_A[0]" LOC = A9;
NET "FP_A[1]" LOC = A8;
NET "FP_A[2]" LOC = B10;
NET "FP_A[3]" LOC = E9;
NET "FP_A[4]" LOC = D9;
NET "FP_A[5]" LOC = E10;
NET "FP_A[6]" LOC = E11;
NET "FP_A[7]" LOC = C8;
NET "FP_A[8]" LOC = B8;
NET "FP_A[9]" LOC = D10;
NET "FP_A[10]" LOC = C10;
```

Fig. 6.8 Assigning timing constraints, and the generated ucf text file

After the synthesis you can review the graphical representation of the synthesized circuit (View RTL schematic) or review the synthesis report (.syr file). This report gives relevant information. The "HDL synthesis" part of report describes the inferred component as a function of the HDL and the "final report" summarizes the resource utilization and performs a preliminary timing report.

6.6.1.3 Implementation: Constraints and Reports

In order to implement a design, we need to generate the implementation constraints. We will firstly assign the position of the pads. We can edit manually the ucf (user constraint file) text file or double click the "I/O pin planning (plan ahead)" and use the graphical interface.

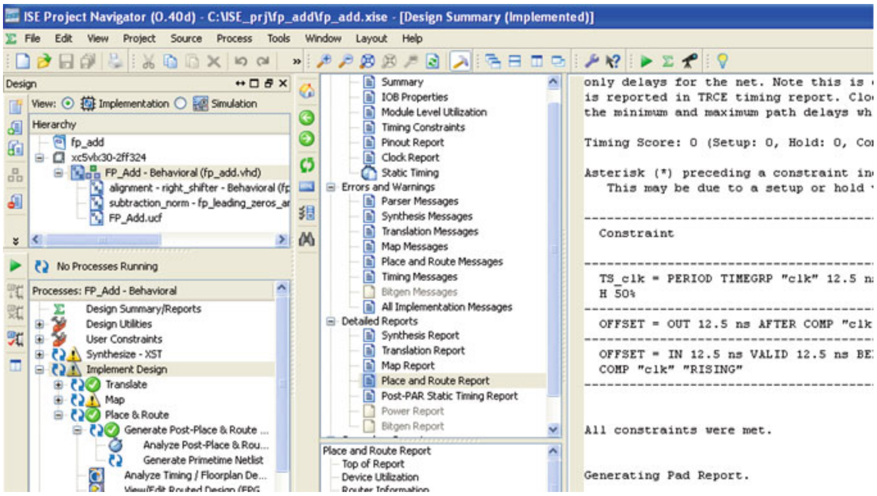
Then we can assign timing constraints using the "create timing constraint" option. We will assign a period constraints ("clock domain" in constraint type view) of 80 MHz (12.5 ns) and the same restriction for inputs and outputs ("inputs" and "outputs" in constraint type view, respectively). You can check the generated restriction editing the ucf file (Fig. 6.8).

We can assign global options to the implementation using right click over the implementation icon in processes window and selecting "process properties…"

We implement the design double clicking in "implementation". The three step in the Xilinx implementation flow will be executed (translate, map, place & route).

In ISE the design implementation, comprises the following steps:

- Translate: merges the incoming netlists from synthesis and the imple-mentation constraints into a proprietary Xilinx native generic database (NGD) file.
- Map: fits the design into the available resources on the target device. The output is a native circuit description (NCD) file.
- Place and Route: takes a mapped NCD file, places and routes the design, and produces another NCD file.



```
only delays for the net. Note this is
is reported in TRCE timing report. Clo
the minimum and maximum path delays wh

Timing Score: 0 (Setup: 0, Hold: 0, Co

Asterisk (*) preceding a constraint in
   This may be due to a setup or hold

----------------------------------------
  Constraint

----------------------------------------
  TS_clk = PERIOD TIMEGRP "clk" 12.5 n
  H 50%
----------------------------------------
  OFFSET = OUT 12.5 ns AFTER COMP "clk
----------------------------------------
  OFFSET = IN 12.5 ns VALID 12.5 ns BE
  COMP "clk" "RISING"
----------------------------------------


All constraints were met.


Generating Pad Report.
```

Fig. 6.9 Implemented design. Access to the reports

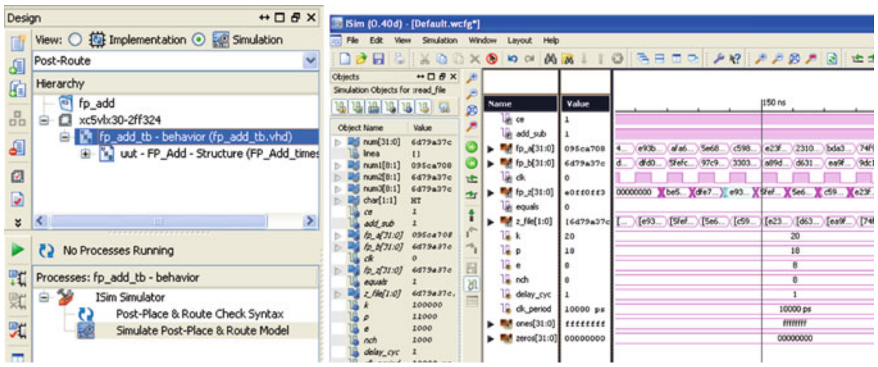
The result of the implementation is a proprietary placed and routed netlist (ncd file) and several reports. There are three main reports generated: the map report (.mrp), the place and route report (.par) and the "Post-PAR static timing report" (.twr) (Fig. 6.9).

The map report shows detailed information about the used resources (LUTs, slices, IOBs, etc.), the place and route report gives clock network report including the skew and informs which constraints were met and which were not. Finally, Post-PAR static timing report gives the worst case delay with respect to the specified constraints.

You can obtain additional graphical information about the place and route results using FPGA editor and PlanAhead. Use the FPGA Editor to view the actual design layout of the FPGA (in the Processes pane, expand "Place & Route", and double-click "View/Edit Routed Design (FPGA Editor)"). The PlanAhead software can be used to perform post place & route design analysis. You can observe, graphically, the timing path onto the layout, and also perform floorplaning of the design. In order to open PlanAhead in the Processes pane, expand Place & Route, and double-click "Analyze Timing/Floorplan Design (PlanAhead)".

6.6.1.4 Post Place and Route Simulation

The post place and route (or timing) simulation is accessed by selecting "simulation" in the view panel and selecting "post-route" from the drop-down list. Then select the test bench and double click "Simulate Post-Place & Route Model". This will execute the back annotation process (netgen). Then the simulation is performed. Observe that the resulting simulation gives some errors. Why? The answer is simple, the test bench generates a clock of 100 MHz (10 ns period)


Design
View: Implementation Simulation
Hierarchy
No Processes Running
ISim (0.40d) - [Default.wcfg*]
File Edit View Simulation Window Layout Help
Objects
Simulation Objects for :read_file
150 ns

Post-Route

- fp_add
- xc5vlx30-2ff324
  - fp_add_tb - behavior (fp_add_tb.vhd)
    - uut - FP_Add - Structure (FP_Add_times

Processes: fp_add_tb - behavior

- ISim Simulator
  - Post-Place & Route Check Syntax
  - Simulate Post-Place & Route Model

Object Name	Value
num[31:0]	6d79a37c
linea	[]
num1[0:1]	095ca708
num2[0:1]	6d79a37c
num3[0:1]	6d79a37c
char[1:1]	HT
ce	1
add_sub	1
fp_a[31:0]	095ca708
fp_b[31:0]	6d79a37c
clk	0
fp_z[31:0]	6d79a37c
equals	1
z_file[1:0]	6d79a37c,
k	100000
p	11000
e	1000
nch	1000
delay_cyc	1

Name	Value
ce	1
add_sub	1
fp_a[31:0]	095ca708
fp_b[31:0]	6d79a37c
clk	0
fp_z[31:0]	e0ff0ff3
equals	0
z_file[1:0]	16d79a37c
k	20
p	18
e	8
nch	8
delay_cyc	1
clk_period	10000 ps
ones[31:0]	ffffffff
zeros[31:0]	00000000

Fig. 6.10 Post place and route simulation

and the implementation that we perform is slower. Then, some input pattern combination violates setup time and gives, consequently, errors. You can modify the test bench in order to operate at a lower frequency (Fig. 6.10).

6.6.1.5 Running a Static Timing Analysis

We can review, more carefully, the timing aspects, by opening the "analyze post-Place & Route Static Timing" (expand Implement Design → Place & Route → Generate Post-Place & Route Static Timing to access this process). By default, this runs an analysis of the worst case delays with respect to the specified constraints giving the three worst paths (Fig. 6.11).

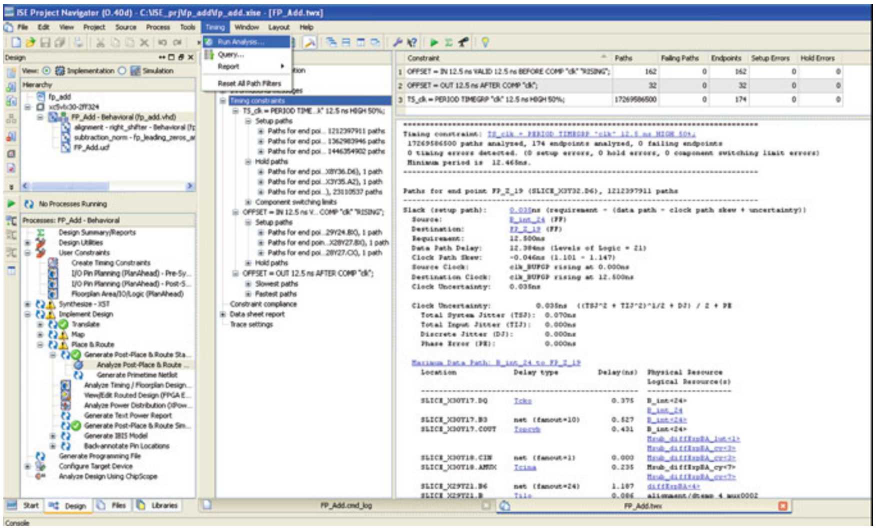
In order to control more aspects of the timing analysis you can run an analysis (Timing → Run Analysis), you can control the type of analysis (Analyze against), control the amount of path reported, control the timing derating factors (Sect. 5.1.2.5), filter nets and paths in the analysis, etc.

6.6.1.6 Generating Programming File and Programming the FPGA

After implementing the design and performed the corresponding timing analysis, you need to create configuration data. A configuration bitstream (.bit) is created for downloading to a target device or for formatting into a PROM programming file (Fig. 6.12).

6.6.1.7 Using Command Line Implementation

All the previous step could be executed in command line and group it in order to create scripts. The ISE software allows you extract the command line arguments



	Constraint	Paths	Failing Paths	Endpoints	Setup Errors	Hold Errors
1	OFFSET = IN 12.5 ns VALID 12.5 ns BEFORE COMP "clk" "RISING";	162	0	162	0	0
2	OFFSET = OUT 12.5 ns AFTER COMP "clk";	32	0	32	0	0
3	TS_clk = PERIOD TIMEGRP "clk" 12.5 ns HIGH 50%;	17269586500	0	174	0	0

```
================================================================================
Timing constraint: TS_clk = PERIOD TIMEGRP "clk" 12.5 ns HIGH 50%;
 17269586500 paths analyzed, 174 endpoints analyzed, 0 failing endpoints
 0 timing errors detected. (0 setup errors, 0 hold errors, 0 component switching limit errors)
 Minimum period is  12.465ns.
--------------------------------------------------------------------------------

Paths for end point FP_Z_19 (SLICE_X3Y32.D6), 1212397911 paths
--------------------------------------------------------------------------------
Slack (setup path):     0.035ns (requirement - (data path - clock path skew + uncertainty))
  Source:               B_int_24 (FF)
  Destination:          FP_Z_19 (FF)
  Requirement:          12.500ns
  Data Path Delay:      12.384ns (Levels of Logic = 21)
  Clock Path Skew:      -0.046ns (1.101 - 1.147)
  Source Clock:         clk_BUFGP rising at 0.000ns
  Destination Clock:    clk_BUFGP rising at 12.500ns
  Clock Uncertainty:    0.035ns

  Clock Uncertainty:          0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter (TSJ):  0.070ns
    Total Input Jitter (TIJ):   0.000ns
    Discrete Jitter (DJ):       0.000ns
    Phase Error (PE):           0.000ns

  Maximum Data Path: B_int_24 to FP_Z_19
    Location             Delay type         Delay(ns)  Physical Resource
                                                       Logical Resource(s)
    -------------------------------------------------  -------------------
    SLICE_X30Y17.DQ      Tcko                  0.375   B_int<24>
                                                       B_int_24
    SLICE_X30Y17.B3      net (fanout=10)       0.527   B_int<24>
    SLICE_X30Y17.COUT    Topcyb                0.431   B_int<24>
                                                       Msub_diffExpBA_lut<1>
                                                       Msub_diffExpBA_cy<3>
    SLICE_X30Y18.CIN     net (fanout=1)        0.000   Msub_diffExpBA_cy<3>
    SLICE_X30Y18.AMUX    Tcina                 0.235   Msub_diffExpBA_cy<7>
                                                       Msub_diffExpBA_cy<7>
    SLICE_X29Y21.B6      net (fanout=24)       1.107   diffExpBA<4>
    SLICE_X29Y21.B       Tilo                  0.086   alignment/dtemp_4_mux0002
```

Fig. 6.11 Post place and route static timing analysis

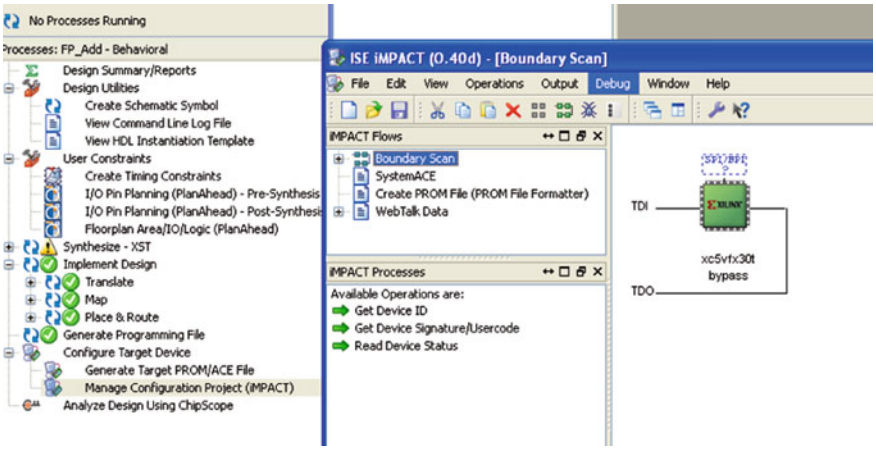
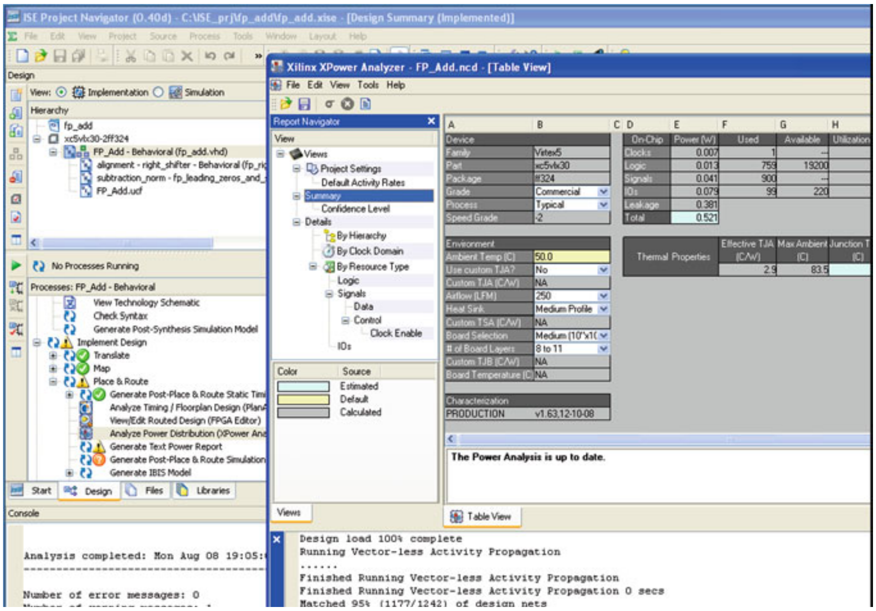

No Processes Running
Processes: FP_Add - Behavioral
Design Summary/Reports
Design Utilities
Create Schematic Symbol
View Command Line Log File
View HDL Instantiation Template
User Constraints
Create Timing Constraints
I/O Pin Planning (PlanAhead) - Pre-Synthesis
I/O Pin Planning (PlanAhead) - Post-Synthesis
Floorplan Area/IO/Logic (PlanAhead)
Synthesize - XST
Implement Design
Translate
Map
Place & Route
Generate Programming File
Configure Target Device
Generate Target PROM/ACE File
Manage Configuration Project (iMPACT)
Analyze Design Using ChipScope
ISE iMPACT (0.40d) - [Boundary Scan]
File Edit View Operations Output Debug Window Help
iMPACT Flows
Boundary Scan
SystemACE
Create PROM File (PROM File Formatter)
WebTalk Data
iMPACT Processes
Available Operations are:
Get Device ID
Get Device Signature/Usercode
Read Device Status
SPI/BPI
TDI
TDO
xc5vfx30t
bypass

Fig. 6.12 Generating programming file and programming the FPGA

for the various steps of the implementation process (Design Utilities → View Command Line Log File). This allows you to verify the options being used or to create a command batch file to replicate the design flow.


ISE Project Navigator (O.40d) - C:\ISE_prj\fp_add\fp_add.xise - [Design Summary (Implemented)]
Xilinx XPower Analyzer - FP_Add.ncd - [Table View]
Report Navigator
Views
Project Settings
Default Activity Rates
Summary
Confidence Level
Details
By Hierarchy
By Clock Domain
By Resource Type
Logic
Signals
Data
Control
Clock Enable
IOs
Color
Source
Estimated
Default
Calculated
Processes: FP_Add - Behavioral
View Technology Schematic
Check Syntax
Generate Post-Synthesis Simulation Model
Implement Design
Translate
Map
Place & Route
Generate Post-Place & Route Static Timi
Analyze Timing / Floorplan Design (PlanA
View/Edit Routed Design (FPGA Editor)
Analyze Power Distribution (XPower Ana
Generate Text Power Report
Generate Post-Place & Route Simulation
Generate IBIS Model
Start
Design
Files
Libraries
Console
Views
Table View

Device	
Family	Virtex5
Part	xc5vlx30
Package	ff324
Grade	Commercial
Process	Typical
Speed Grade	-2

On-Chip	Power (W)	Used	Available	Utilization
Clocks	0.007	1	---	
Logic	0.013	759	19200	
Signals	0.041	900	---	
IOs	0.079	99	220	
Leakage	0.381			
Total	0.521			

Environment	
Ambient Temp (C)	50.0
Use custom TJA?	No
Custom TJA (C/W)	NA
Airflow (LFM)	250
Heat Sink	Medium Profile
Custom TSA (C/W)	NA
Board Selection	Medium (10"x1(
# of Board Layers	8 to 11
Custom TJB (C/W)	NA
Board Temperature (C)	NA

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction T (C)
	2.9	83.5	

Characterization	
PRODUCTION	v1.63,12-10-08

The Power Analysis is up to date.

```
Analysis completed: Mon Aug 08 19:05:
--------------------------------------

Number of error messages: 0
```

```
Design load 100% complete
Running Vector-less Activity Propagation
......
Finished Running Vector-less Activity Propagation
Finished Running Vector-less Activity Propagation 0 secs
Matched 95% (1177/1242) of design nets
```

Fig. 6.13 Xpower analyzer result

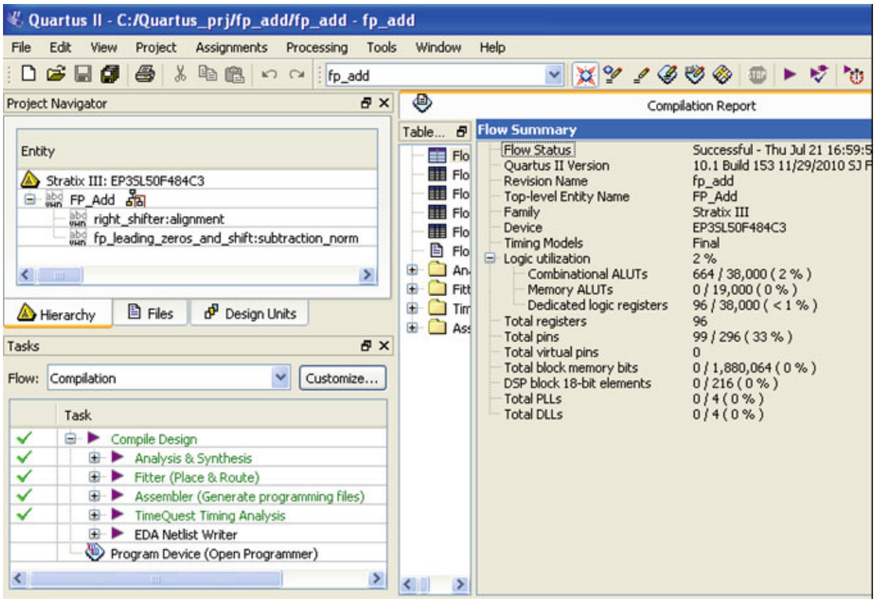
6.6.1.8 Estimating the Power Consumption

After place and route of the design it is possible to obtain power consumption estimation. The accuracy of the estimation relies upon the activity given to the xpower tool. The activity file is generated with a post place & route simulation (Sect. 6.6.1.4) but instructing the ISim simulator to generate the SAIF (Switching Activity Interchange Format) file. In order to do that, right click "simulate post place and route model", click on "process properties…" then select "Generate SAIF File for Power Optimization/Estimation", it is possible to assign different file names.

In order to check the power estimation results, based on the activity computed, you can even generate a simple text report by double clicking on "Generate Text Power Report", or by opening the interactive tool Xpower double clicking "Analyze Power Distribution (Xpower Analyzer)" (Fig. 6.13). Observe that by browsing into the details you have access to the activity of each node, the fan-out of nets and the corresponding power.

6.6.2 Simple Example Using Altera Quartus II

Designing for Altera devices is very similar, in concept and practice, to designing for Xilinx devices. In most cases, the same RTL code can be compiled by Altera's



Stratix III: EP3SL50F484C3
FP_Add
right_shifter:alignment
fp_leading_zeros_and_shift:subtraction_norm

Flow: Compilation

Task
✓ Compile Design
✓ Analysis & Synthesis
✓ Fitter (Place & Route)
✓ Assembler (Generate programming files)
✓ TimeQuest Timing Analysis
EDA Netlist Writer
Program Device (Open Programmer)

Flow Summary

Flow Status	Successful - Thu Jul 21 16:59:5
Quartus II Version	10.1 Build 153 11/29/2010 SJ F
Revision Name	fp_add
Top-level Entity Name	FP_Add
Family	Stratix III
Device	EP3SL50F484C3
Timing Models	Final
Logic utilization	2 %
Combinational ALUTs	664 / 38,000 (2 %)
Memory ALUTs	0 / 19,000 (0 %)
Dedicated logic registers	96 / 38,000 (< 1 %)
Total registers	96
Total pins	99 / 296 (33 %)
Total virtual pins	0
Total block memory bits	0 / 1,880,064 (0 %)
DSP block 18-bit elements	0 / 216 (0 %)
Total PLLs	0 / 4 (0 %)
Total DLLs	0 / 4 (0 %)

Fig. 6.14 Quartus II project

Quartus II as was explained for Xilinx ISE (Sect. 6.6.1). Altera has an application note “Altera Design Flow for Xilinx Users” [1], where this process is carefully explained.

We can start creating a new project (file → New Project Wizard) with name fp_add. We add to the project the three VHDL source files (step 2 of 5).We select a Stratix III device, EP3SL50F484C3. Leave the default synthesis and simulation options (Fig. 6.14).

For the rest of the steps in the implementation and simulation flow, Table 6.1 summarizes the GUI (graphical user interface) names for similar task in Xilinx ISE and Altera Quartus II.

6.7 Exercises

1. Implement the floating adder of Chap. 12 in Altera Quartus in a Stratix III device. What are the area results? Add implementation constraints in order to add FP numbers at 100 MHz. Constraints are met?
2. Implement the floating point multiplier of Chap. 12 in Xilinx ISE in a Virtex 5 device. What are the area results? It is possible to multiply at 100 MHz? Remember to add implementation constraints. Multiplying at 20 MHZ what is the expected power consumption? Use the XPower tool and the provided test bench.

Table 6.1 GUI names for similar tasks in Xilinx ISE and Altera Quartus II

GUI feature	Xilinx ISE	Altera quartus II
HDL design entry	HDL editor	HDL editor
Schematic entry	Schematic editor	Schematic editor
IP entry	CoreGen and architecture wizard	Megawizard plug-in manager
Synthesis	Xilinx synthesis technology (XST)	Quartus II integrated synthesis (QIS)
	Third-party EDA synthesis	Third-party EDA synthesis
Synthesis constraints	XCF (xilinx contraint file)	Same as implementation
Implementation constraint	UCF (user constraint file)	QSF (quartus II settings file) and SDC (synopsys design constraints file)
Timing constraint wizard	Create timing constraints	Quartus II timequest timing analyzer SDC editor
Pin constrain wizard	PlanAhead (PinPlanning)	Pin planner
Implementation	Translate, map, place and route	Quartus II integrated synthesis (QIS), fitter
Static timing analysis	Xilinx timing analyzer	Timequest timing analyzer
Generate programming file	BitGen	Assembler
Power estimator	XPower estimator	Powerplay early power estimator
Power analysis	XPower analyzer	Powerplay power analyzer
Simulation	ISE simulator (ISim)	Modelsim–Altera starter edition
	Third-party simulation tools	Third-party simulation tools
Co-simulation	ISim co-simulation	–
In-chip verification	Chipscope pro	SignalTap II logic analyzer
View and editing placement	PlanAhead, FPGA editor	Chip planner
Configure device	iMPACT	Programmer

3. Implement the pipelined adder of Chap. 3. Analyze the area-time-power trade off for different logic depths in the circuit. Use, for the experiment, a Virtex 5 device and compare the result with respect to the new generation Virtex 7. What happened in Altera, in comparing stratix III devices with respect to Stratix V?
4. Implement the adders of Chap. 7 (use the provided VHDL models). Add to the model input and output registers in order to estimate the maximum frequency of operation.
5. Compare the results of implementation of radix 2 ^{k} adders of Sect. 7.3 in Xilinx devices using *muxcy* and a behavioural description of the multiplexer (use the provided VHDL models of Chap. 7).
6. Implement the restoring, non-resorting, radix-2 SRT and radix 2 ^{k} SRT divider of Chap. 9 in Xilinx and Altera devices (use the provided VHDL models of Chap. 9).
7. Compare the results of square root methods of Chap. 10 in Altera and Xilinx design flow.

References

1. Altera corp (2009) AN 307: Altera design flow for Xilinx users. http://www.altera.com/literature/an/an307.pdf
2. Altera corp (2011a) Altera quartus II software environment. http://www.altera.com/
3. Altera corp (2011b) Design and synthesis. In: Quartus II integrated synthesis, quartus II handbook version 11.0, vol 1. http://www.altera.com/
4. Mentor Graphics (2011) Mentor precision synthesis reference manual 2011a. http://www.mentor.com/
5. Microsemi SoC Products Group (2010) Libero integrated design environment (IDE) v9.1. http://www.actel.com/
6. Synopsys (2011) Synopsys FPGA synthesis user guide (Synplify, Synplify Pro, or Synplify Premier). http://www.synopsys.com/
7. Xilinx inc (2011a) Xilinx ISE (Integrated Integrated software environment) design suite. http://www.xilinx.com/
8. Xilinx inc (2011b) XST (Xilinx Synthesis Technology) user guide, UG687 (v 13.1). http://www.xilinx.com/