




ASIC Design Flow

By

P.Radhakrishnan,
Senior ASIC-Core Development Engineer,
Toshiba,
1060, Rincon Circle,
San Jose, CA 95132 (USA)

Jan 2000 (Issue-3)



Contents

Introduction.....	3
Application Specific Integrated Circuits (ASIC).....	3
Frontend Design.....	4
Coding	4
Verification.....	5
Synthesis	5
Backend Design.....	6
Design Tools	6
Conclusion & Disclaimer.....	7
Trademarks	7

Introduction

This paper addresses some of the current trends that have been followed in the ASIC design. The amount of information that can be provided about this topic is so vast that one can write a complete book about the ASIC Design. However this paper addresses the important two phases of an ASIC design so that students can get an idea of how a chip is developed from concept to silicon.

Due to the high demand for speed and performance, most of the functions that were implemented in the software are shifting towards the hardware. This means functions like the communication protocols, DSP algorithms, audio and video compression algorithms and many other functions find it most advantageous to be implemented in the hardware for meeting the performance. Another factor that leads to this situation is miniaturization. Due to the fast changes happening in the semiconductor technology the ability to pack more circuits in a small area has increased many fold. While the semiconductor companies are concentrating on the advanced technologies, the designers are increasing the complexity of the design by adding more and more logic in a single chip leading to the System-on-a-Chip (SoC) concept. This paper throws light on some of the key steps involved in realizing an ASIC.

Application Specific Integrated Circuits (ASIC)

What is an ASIC? It is one other chip built to do a specific function. As mentioned above there are high demands in the industry to implement algorithms in hardware. As a result of this, many design groups implement specific functions in silicon in the form of an ASIC.

In the olden days (about 10 years ago) the set of tools available for IC design were very limited, and there were a large amount of manual interventions. These days' ICs are very huge and it is impossible to do them without the help of the EDA (Electronic Design Automation) tools. The EDA tools play a major role in the ASIC design flow and there are hundreds of companies functioning as EDA vendors supplying tools that help in various phases of the chip design.

In the initial days when the EDA tools were evolving, tools were supporting the schematic entry. Schematics are the pictorial representation of the gates and flocs of the design. You would draw the circuit in the same way as you would do in a paper. The schematic editor will have many capabilities to describe hierarchical design and also have capabilities to duplicate, cut, paste, etc. With these features one could build a circuit and edit them with ease. The tools took these schematics as inputs and generated database having connectivity information in them. The next version of tools accepted text entries in the form of Boolean equations and worked on the equation to get the desired logic. Since one can express any logic function in the form of a sum-of-product (SOP) or product-of-sum (POS) terms, these tools took the Boolean equations and run the optimizing algorithms to get the optimal logic which would meet the function and the timing. These optimizations are similar to the optimizations one would do using methods like K-map.

The current days' designers use what are known as HDL (Hardware Description Language) for the design. HDL is a way of describing the logic and state machines

through equations and behavioural descriptions. The tools understand these descriptions and infer the necessary logic to implement the logic function.

Frontend Design

Frontend design is the first phase in the ASIC design where the logic for the chip is built using the HDL. The designer would write codes that would represent the function that he/she wants to implement. This logic realization is done using various steps in which the design is coded, verified and mapped to the actual gates using a process called synthesis. The rest of this section addresses to each one of these steps.

Coding

In the industry there are several HDLs being used. However there are two common hardware description languages that are very popular. They are, Verilog and VHDL. Though VHDL is a much older candidate, Verilog became very popular due to its “C” like syntax and constructs. There are plenty of arguments about which one is easier or better. In the recent years support for Verilog has been the prime focus of most of the EDA vendors. But there is always support for the VHDL codes also. No matter which language one uses for coding, the intention here is to express the design in the form of a behavioural description of code.

For those who are wondering how these codes would look like, there is a very simple example given below. Complex functionalities will have numerous lines of codes distributed in many files.

Example:1 (Verilog Code)

```
// This line is a comment
module test (clk, a, b, c, c_reg);
input clk, a, b;
output c, c_reg;
reg c_reg; wire c;
assign c = a || b; // Logical OR of the signals a and b
// output of the previous step is sent through a flop that
// is clocked by "clk"
always @(posedge clk)
    c_reg <= c;
endmodule
```

The above code describes a design that has two inputs, “a” and “b” and two outputs “c” and “c_reg”. The output “c” is the logical OR of the two inputs. The output “c_reg” is a registered version of “c”. i.e. output “c” is given to the input of a flop that is clocked by “clk” and the “Q” output of the flop is connected to “c_reg”

Verification

Once the coding is completed it has to be checked if the design is doing its expected function. This is called the functional verification. One can develop a test bench using Verilog or VHDL to apply all possible stimuli at the input and check the output that is generated by the code. For the above design it may not be a great deal to check the possible combinations of the input. This is because there are only just four possible combinations to be tested. But for a huge chip, this may be a very involved job. Once the designer is sure that the code functions as expected, he will take the code through the synthesis process to convert it into gates.

Synthesis

This is the next step in the frontend design. Using one of the various synthesis tools available, the designer will target the design into equivalent gates and flops. The output from the synthesis phase is often referred as the **netlist**, which represents the connectivity of the cells used to realise the logic. These netlists can be in the Verilog or VHDL format. There are other interchangeable formats that are used in the industry too. The tool will read the code and map the logic functions into the relevant gates/flops and provide the connectivity also. For this the tool would need a target library from which it can take the cells. The target libraries are provided by the silicon vendor, which depends on the type of technology the designer intended to use for the chip. A synthesized netlist from a synthesis tool will look like the one shown below.

Example:1 (Verilog netlist)

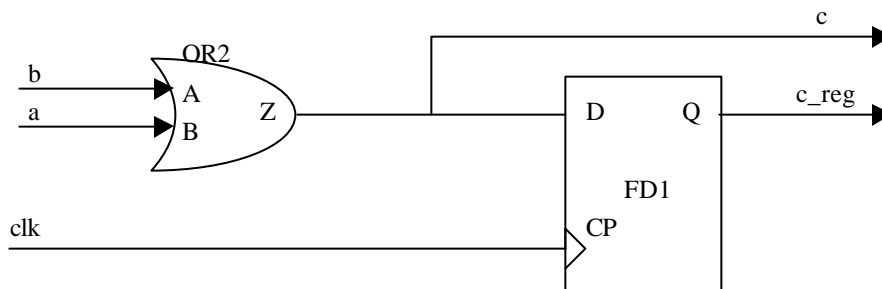
```
module test ( clk, a, b, c, c_reg );
input  clk, a, b;
output c, c_reg;

    FD1 c_reg_reg ( .Q(c_reg), .D(c), .CP(clk) );

    OR2 U9 ( .Z(c), .A(b), .B(a) );

endmodule
```

The above representation is also in Verilog except that the file is having hard celles in them instead of the behavioural descriptions. The cell “OR2” represents a two-input OR gate from the technology library and the cell “FD1” represents a D-flip-flop. This flop’s D-input is same as the output “c” and output-Q of the flop is connected to “c_reg”. The flop is clocked by the signal “clk”. A pictorial representation of the logic is shown below.



Since this netlist also represents the design, one can run the functional verification tests against this netlist also. What is more important is the Static Timing Analysis (STA) on the netlist. STA reveals any potential timing problems like a setup or hold violation in the design. If there are some timing errors, those have to be fixed. A great deal of STA is explained in the article released in July`99 issue-1.

Backend Design

The second major phase in the ASIC design is commonly known as the backend where the cells in the netlist are placed on the die and then routed. There are many tools available for this process also. The netlist, which has no violations is read into the placement tool and placed within the die area according to the guidelines given to the tool. Placement of cells depends on the connectivity and also on the distance between the cells so as not to cause any timing violation. Cells that are placed far apart will have a lengthy wire to connect them, and this will cause additional delay. Another reason for more delay is, the fanout. When the fanout of a cell increases, this increases the capacitance load on the driving cell. This will also cause output signal to propagate slowly from one point to another. The placement tool will be given a set of constraints that will indicate the timing requirements of the design. Hence the tool will try to place the cells in such a way that there are no timing deteriorations. In addition to the cells, the tool will place the IO pads also along the periphery of the die so that the pins can be connected to the pads using metal bonding at a later stage.

Once the cells and IO pads are placed satisfactorily, the design is given to the routing tool for routing. This tool will connect the cells according to the information in the netlist. When the routing is completed the designer can extract the actual delay contributed by the cells and the wires. These timing numbers are to be fed to the STA tool once again to determine the conditions of violations based on the final routing. If there are any violations, they need to be fixed by performing some netlist changes or by improving the routing/placement or by doing both. This will take few iterations to converge on the timing. A considerable amount of time is spent on these iterations and the industry is trying to develop methodologies that will reduce these iterations and are successful to some extent. When the backend phase is completed with no violations, the design is converted into a database that will be used to build the mask for the semiconductor processing. These data will be given to the processing units for manufacturing the chip.

Design Tools

There are numerous companies in the industry, providing EDA tools for chip designing. Some of them are mentioned here just to give an idea to the students. Some of the companies like Synopsys, Cadence, Chrysalis, Avanti, etc. provide various tools that provide support for the frontend and the backend design.

Verilog-XL, VCS, VHDL simulators are some the tools used for design entry (coding) and simulations. These are the properties of companies like Cadence and Synopsys. Synopsys is one of the major vendors of the chip design tools. Its “DesignCompiler” has been used by designers all over the world for synthesis, in the past many years. It has been introducing a series of tools for automating many of the steps in the flow. Some other tools that can be mentioned here are “TestCompiler”

and “TestGen” for scan design, “ChipArchitect”, “FloorPlanManager” and so on. Cadence and Avanti also have a series of tools that help the ASIC design flow both in the frontend and in the backend design. Some of the tools of Cadence that are worth mentioning are, SiliconEnsemble, GateEnsemble, BuildGates, CTGen, etc. (The names of the tools are changed from time to time, but the essence of it is the same) There are other companies like Altera, which has developed their own HDL called AHDL (Altera-HDL) and a set of synthesis tools called MaxPlus. This company’s tools are used for programmable devices. Altera was producing devices called EPLDs (Erasable Programmable Logic Devices). Another company that can be mentioned is Xilinx, which developed tools for programmable devices called FPGAs (Field Programmable Gate Arrays). Though these tools are used for programmable devices, the methodology is almost same as the ASIC development.

Conclusion & Disclaimer

This paper gives only a bird’s eye view of what is happening in the ASIC design process. Though there are hundreds of finer steps involved in the process, this paper does not provide a detailed explanation of those finer steps. Each of those steps can be described in an individual paper by itself. The author wishes to elaborate those topics in the future papers. The sample code given in this paper is very primitive and the students are recommended to refer some books in the topics like Verilog and VHDL. There are many excellent books available in these topics.

Trademarks

DesignCompiler, TestCompiler, TestGen, ChipArchitect and FloorPlanManager are the registered trademarks of Synopsys, Inc., Mountain View, CA, USA.

SiliconEnsemble, GateEnsemble, BuildGates, CTGen are the registered trademarks of Cadence Design Systems, Inc. San Jose, CA, USA

AHDL and Maxplus belong to Altera Corporation, San Jose, California 95134, USA.

