Chapter One: Introduction

I must create a system, or be enslav’d by another man’s; I will not reason and compare: my business is to create.

“William Blake”

1.1 EDA Tools

Digital design flow regardless of technology is a fully automated process. As described in future chapters, design flow consists of several steps and there is a need for a toolset in each step of the process. Modern FPGA/ASIC projects require a complete set of CAD (Computer Aided Design) design tools. Followings are the most common tools available in the market that are briefly explained here:

1.1.1 Design Capture Tools

Design entry tool encapsulates a circuit description. These tools capture a design and prepare it for simulation. Design requirements dictate type of the design capture tool as well as the options needed. Some of the options would be:

- Manual netlist entry
- Schematic capture
- Hardware Description Language (HDL) capture (VHDL, Verilog, …)
- State diagram entry

1.1.2 Simulation and Verification Tools
Functional verification tool confirms that the functionality of a model of a circuit conforms to the intended or specified behavior, by simulation or by formal verification methods. These tools are must have tools. There are two major tool sets for simulation: Functional (Logic) simulation tools and Timing simulation tools. Functional simulators verify the logical behavior of a design based on design entry. The design primitives used in this stage must be characterized completely. Timing simulators on the other hand perform timing verifications at multiple stages of the design. In this simulation the real behavior of the system is verified when encountering the circuit delays and circuit elements in actual device. In general, the simulation information reflects the actual length of the device interconnects. This information is “back annotated” to the corresponding design entry for final logic simulation. That is why this process of simulation sometimes is called “back annotation simulation”.

1.1.3 Layout Tools

ASIC designers usually use these tools. Designers transform a logic representation of an ASIC into a physical representation that allows the ASIC to be manufactured. The transistor layout tools take a cell level ASIC representation and for a given technology create a set of layers representing transistors for each cell. Physical design tool works in conjunction with floorplanning tools that show where various cells should go on the ASIC die.
1.1.4 Synthesis and Optimization Tools

Synthesis tools translate abstract descriptions of functionality such as HDL into optimal physical realizations, creating netlists that can be passed to a place and route tool. Then, the designer maps the gate level description or netlist to the target design library and optimizes for speed, area or power consumption.

In Appendix- A a list of major EDA tools vendors is given. The objective is to provide a tool set for FPGA / ASIC design such that the number of vendors to design and build the ASIC /r FPGA is minimized. Each vendor brings a new set of learning curves, maintenance and interfaces that eventually end up spending more time and money. Tool maturity is another important factor. New tools almost always come with certain number of bugs and issues.

1.2 Design Hierarchy

Hierarchical systems in general refer to the systems that are organized in the shape of pyramid with multiple rows of system objects. Each object in a row may be linked to the objects beneath it. Hierarchical systems in computer designs are as popular as they are in everyday life. A good example of a hierarchical system in everyday life could be a monarchy system that has the King on top and then Prime Minister in the next level down. People are the base of this pyramid. An obvious example in computer world would be a file system that has the root directory on top in which directories have files and subdirectories underneath. Generally speaking hierarchical systems have strong
connectivity inside rather than outside the modules. Figure 1-1 shows an example of design hierarchy.

![Design Hierarchy System](image)

**Figure 1-1**: Design Hierarchy System

Design hierarchy trees do not have a regular pattern in size and number of nodes and it is really up to designer to decide how the tree should look like. Figure 1-1 outlines partitioning process in a top-down design methodology. This figure shows that the partitioning procedure is recursively called until the design of all sub-components is feasible by the hardware mapping procedure.

### 1.2.1 Design Methodology

Digital circuits are becoming more complex while time-to-market windows are shrinking. Designers cannot keep up with the advancements in engineering unless they adopt a more
methodological process for design and verification of digital circuits i.e. Design methodology. This involves more than simply coming up with block diagram of a design. Rather, it requires developing and following a formal verification plan and an incremental approach for transforming the design from an abstract block diagram to a detailed transistor level implementation.

High-level ASIC or FPGA designs start off with capturing the design idea with a hardware description language (HDL) at the behavioral or register-transfer (RTL) level. The design will be then verified with an HDL simulator and synthesized to the gates. Gate level design method such as schematic capture was a typical design approach up to few years ago but when the average gate count passed the 10, 000 gate threshold, they started to break down. On the other hand the pressure to reduce the design cycle increased and as a result, high-level design became an imperative part of the digital design engineers. Industry experts agree nowadays that most of the FPGA/ASIC designers will turn to high-level design methodologies in the near future. This happens primarily because of the technology improvements that have taken place in the EDA (Electronic Design Automation) tools, hardware and software.

HDL allows the designer to organize and integrate complex functions and verify the individual blocks and eventually the entire design with tools like HDL simulators. Designers making the switch to a high level design methodology leverage some obvious benefits. First, individual designers are able to handle increased complexity by working at higher levels of abstraction and delaying the design details. Second, designers can shorten cycles and improve quality by verifying functionality earlier in the design cycle, when design changes are easier and less expensive to make.
Some designers may think that FPGAs require significantly less functionality and features compared to ASICs. The truth is that the improvements in FPGA technology including hardware and software tools have enabled the FPGA manufacturers to come up with a high level of integration in FPGAs. Today, several features are being implemented in FPGAs such as processors, transceivers and even debugging tools are available inside the FPGA. This makes the design cycle faster and simpler than what it was before. FPGA designers are using as many features and as much functionality as is made available to them.

Economic requirements on the other hand play a role in the design methodology arena. Functionality and flexibility are two key feature of the design tool sets. FPGA design tools are more cost effective compared to the ASIC design tools, however they lack the functionalities and features available in the ASIC design tools. An ASIC design tool seat software may cost over $100,000 while FPGA design tool seat barely runs over $10,000. There is always a compromise between different factors in order to decide which way to go. These could be time to market, NRE (Non Recurring Engineering) cost, ease of use, programmability, flexibility, etc.

A number of FPGA designs are more complex and have higher density compared to some ASIC designs that are being implemented. Furthermore, high end FPGA design methodology nowadays mirrors an ASIC design methodology. To meet this emerging requirement, tool vendors must provide solutions that will enable FPGA designers to deal with complex designs and at the same time be able to integrate variety of functions and features at high level of abstraction.
HDL based designs represent a new paradigm for some designers. Design flow is straightforward, but like anything else, it requires going through a learning curve. Regardless of which tools you use, the bottom line is that advanced HDL design tools are a must for FPGA and ASIC designers. Price is not the only factor in making the decision for the appropriate tools. Rather, designers must consider the features required to meet their design goals and find out which tools are leaders in their class.

1.2.2 Typical HDL Design Flow

Figure 1-2 shows a typical HDL design flow. This figure does not take into account whether an ASIC or FPGA is being designed. In fact it is a high level view to HDL designs. In later chapters the detailed view for each technology will be covered. As it can be seen, once the design is created, it must be verified prior to the RTL to make sure that its functionality is as intended. A testbench should be created to verify the functionality of the design. This testbench will be used throughout the design flow to verify the functionality at the functional, RTL and timing levels. A test bench is a separate VHDL or Verilog piece of code that is connected to the design's inputs and outputs. In fact, design itself is considered as a black box with a set of inputs and outputs. The input test stimulus is applied to the design inputs and the outputs are observed to ensure the correct functionality. Since the test vectors stay constant throughout synthesis and place and route, they can be used at each stage to verify functionality. The main purpose of the testbench is to provide the stimulus and response information, such as clocks, reset, and input data, that the design will encounter when implemented in an ASIC or FPGA and installed into the final system.
1.2.3  **Bottom-up or Top-down Design Flow?**

The complexity of digital systems has been increasing rapidly over the past years. Fortunately fully automated EDA tools support design synthesis. But there is still a great need for efficient techniques to accelerate the design process and this essentially means choosing the right methodology. The first step in designing a product is to choose the design methodology. For the vast majority of applications it is hard to prefer one implementation to the other: bottom-up or top-down. There is always a certain amount of manipulation involved and evaluating between two correct implementations is usually
based on the designer’s preferences. One may prefer one method for developing a certain type of application while another person prefers the other method. The key idea of both methodologies is the hierarchical propagation of the design units based on behavioral modeling and optimization at each level.

A Bottom-up design methodology starts with individual blocks, which are then combined to form the system. Design of each block starts with a set of specifications and ends with a transistor level implementation, but each block is verified individually and eventually all the blocks will be combined and verified together.

On the other hand a top-down design approach defines the architecture of the whole design as a single unit. The whole design is simulated and optimized afterwards. The requirements for lower level blocks are derived based on the results obtained in the previous steps. Each level is completely designed before proceeding to the next step. Circuits can be designed individually to meet the specifications and finally, the entire design is laid out and verified against the original requirements. Top-down design is referred to as partitioning of a system into its sub-components until all sub-components become manageable design parts. Design of a component is available as part of a library, it can be implemented by modifying an already available part or it can be described for a synthesis program or an automatic hardware generator. Partitioning the design into smaller modules provides the facility for the designers to work as a team and be more productive. It also reduces the total time required to complete the design because it reduces the effect of the late changes in the design process. Any change in a module needs updating the rest of the system.
Generally speaking, bottom-up design methodology is effective for small designs. As size and complexity of the digital designs continues to increase, they cause some implications in this approach. It is very likely to have a number of blocks in a design and once they are combined, simulation takes a long time and verification becomes difficult. Also performance, cost and functionality are typically found at the architectural level and this makes the design modification difficult since any change at the higher levels of abstraction should propagate to the lower level modules and it needs redesigning of the lower level blocks. The other challenge is that in bottom-up design methodology several steps have to be done sequentially which elongates the design process especially when it comes to modification of the design. To address all these challenges, many designers prefer a substitute method namely top-down design methodology.

![Diagram of Top-Down and Bottom-Up Approach for Design Implementation](image)

**Figure 1-3**: Top-Down and Bottom-Up Approach for Design Implementation

Figure 1-3 shows the common approach to solve complex problems. The idea is to break down the design into smaller pieces so that each piece can be designed at a time. Of
course, all the pieces have to be put together again and this should provide the solution to original design. Assembly of the smaller blocks is referred to as bottom-up design. This approach has been applied to complex engineering projects and is now finding its way into digital designs. Top-level design hierarchy specifies the partitioning of the system into manageable blocks as well as each block interface. One of the strengths of this method is that once the top-level schematic is specified, the design process for all the blocks can be started concurrently. It is imperative to avoid complex models if not necessary in top-down design methodology since this makes the design verification process complicated. Each block can be specified by its behavior and design can be expanded gradually.

Mapping to hardware depends on target technology, available libraries, and available tools.

Generally, unavailability of good tools and/or libraries can be compensated by further partitioning of a system into simpler components. After the completion of this top-down design process, the bottom-up implementation phase begins. In this phase, hardware components corresponding to the terminals of the tree are recursively wired to form the hierarchical wiring of the complete system. This figure also shows that the original design is initially described at the behavioral level. In the first level of partitioning, one of its sub-components is mapped to hardware. Further partitioning is required for hardware implementation of the other two components. This procedure goes on until the hardware implementations of all components are available.

At each and every step of a top-down design process a multilevel simulation tool plays an
important role in the correct implementation of the design. Initially a behavioral description of the system under design (SUD) must be simulated to verify the designer’s understanding of the problem. After the first level of partitioning, a behavioral description of each of the sub-components must be developed, and these descriptions must be wired to form a structural hardware model of SUD. Simulation of this new model and comparing the results with those of the original SUD description will verify the correctness of the first level of partitioning. After verifying the first level of partitioning, hardware implementation of each sub-components must be verified. For this purpose, another simulation run in which behavioral models of sub-components are replaced by more detailed hardware level models will be performed.

The process of partitioning and verification stated above continues throughout the design process. At the end, a simulation model, consisting of the interconnection specification of hardware-level models of the terminals of the partition tree, will be performed. The simulation of this model, and comparing the results with those of the original behavioral description of SUD verify the correctness of the complete design.

In a large design where simulation of a complete hardware-level model, is too time consuming, subsections of the partition tree will be independently verified. Verified behavioral models of such subsections will be used in forming the simulation model for final design verification.

1.3 Hardware Description Languages (HDLs)

With only gate-level design tools available, it soon became clear that better, more
structured design methods and tools would be needed. On the other hand, there was an increase in the demand for electronic components in the continual growth of personal computers, cellular phones and high-speed data communications. Electronic vendors are providing devices with increasingly greater functionality, higher performance, lower cost, smaller packaging and lower power consumption. The trend for electronic design is continuing to provide complex designs and systems with more capability using fewer devices that take up less area on Printed Circuit Boards (PCBs). Problems were encountered with increasingly complex electronic systems using the existing Electronic Design Automation (EDA) tools coupled with the accelerated time-to-market schedules. ASICs and high-Density Programmable Logic devices (HDPLDs) and Very High Speed Integrated Circuit (VHSIC) Hardware Descriptive Language (VHDL) became the key element for developing methodologies in handling complex electronic design. To meet these challenges, a team of engineers from three companies; IBM, Texas Instruments and Intermetrics – were contracted by the DOD to complete the specification and implementation of a new, language-based design description method. The first publicly available version of VHDL, version 7.2, was released in 1985. In 1986, the IEEE (Institute of Electrical and Electronics Engineers) was presented with a proposal to standardize the language. The resulting standard, IEEE 1076-1987, is the basis for virtually every simulation and synthesis product sold today. An enhanced and updated version of the language, IEEE 1076-1993, was released in 1994, and VHDL tool vendors have been responding by adding these new language features to their products.

1.3.1 What is VHDL?

VHDL, which stands for VHSIC (Very High Speed Integrated Circuit) Hardware
Description Language, was developed in the early 1980s as a spin-off of a high-speed integrated circuit research project funded by the U.S. Department of Defense (DOD). During the VHSIC program, researchers were confronted with the daunting task of describing circuits of enormous scale and of managing very large circuit design problems that involved multiple teams of engineers from industry as well as academia. VHDL is becoming popular as a way to express complex digital design concepts for both simulation and synthesis. The power of VHDL makes it easy to describe very large digital designs and systems, and the wide variety of design tools now available make the translation of these descriptions into actual working hardware much faster and less error-prone than in the past.

The power and depth of VHDL does have its price, however. To make effective use of VHDL as a design entry tool, it is imperative to learn the language and, perhaps more importantly, to learn to use the higher level design methods made possible by this powerful language.

VHDL is a programming language that has been designed and optimized to describe the behavior of digital systems. In its present form, VHDL satisfies all aspects of the DOD requirements document. The experience of researchers, software developers, and other users with VHDL since it became the IEEE standard in 1987 indicates this language is sufficiently rich for designing and describing today’s digital systems. As originally required, VHDL is an HDL with strong emphasis on concurrency. Concurrency will be covered in details in later chapters but briefly it is a key feature and the most important advantage of a VHDL. Concurrent languages mimic the hardware behavior. Even though the design can be written in a language similar to the software languages, but the execution flow is in parallel for concurrent statements in the
design. The language supports hierarchical description of hardware from system
to gate level. As such, VHDL combines features of the following:

1.3.1.1 A technology independent and portable language

One of the major strengths of VHDL over other hardware languages ABEL is that it does
not provide specialized language features for accessing common device features (such as
flip flops and I/O buffers). In other words the designer can delay the decision of which
technology to target the design to till much later.

VHDL is an excellent tool for complex electronic designs using PLDs. With very large
complex electronic system designs and large capacity CPLDs and FPGAs, it is very
difficult and almost impossible to design using Boolean equations or schematic capture
containing gate level descriptions. VHDL provides high-level language constructs for
design engineers to describe large circuits or systems. VHDL allows electronic designs
or modules of the design to be stored as libraries allowing for use and portability to other
designs. It is a standard language, IEEE Standard 1076, that is portable between
simulation and synthesis tools. VHDL allows the design to be device-independent,
targeting the technology and architecture towards the end of the design process and
development. It allows for smoother transition of the design from PLD to ASIC.

Schematic capture compared to VHDL does have some advantages. For instance, it
provides a graphical view of the design and divide-and-conquer techniques with the use
of schematic hierarchy. Schematic capture has the following drawbacks:

- Control logic must be designed with traditional techniques.
Schematics can be difficult to maintain because the intent of the design gets lost in the complexity and detail.

Schematics must be accompanied with documentation to scribe the functionality of the design with respect to the application.

Portability is sometimes a problem with schematic capture. It is difficult going from one project using one design tool and importing that to another tool on another project. VHDL is tool independent.

Simulation environments for schematic capture design may not be the same for system level electronics making the verification of performance difficult.

Designing using VHDL increases the efficiency of the designer. VHDL designs facilitate capturing, understanding, maintaining and providing well-defined designs.

1.3.1.2 A standard language

One of the most compelling reasons to become experienced with and knowledgeable in VHDL is its adoption as a standard in the electronic design community. Using a standard language such as VHDL virtually guarantees that there is no need to throw away and recapture design concepts simply because the design entry method which is chosen is not supported in a newer generation of design tools. Using a standard language also means that the designers are more likely to be able to take advantage of the most up-to-date design tools and that they have access to a knowledge base of thousands of other engineers, many of whom are solving problems similar to their own.

1.3.1.3 A low cost language with fast turn-around time
VHDL, programmable logic devices and the EDA tools available these days allow for an
efficient fast design cycle. VHDL permits very large complex designs to be completed
quickly. On the other hand PLDs reduce Non Recurring Engineering (NRE) costs and
allow for quick turn-around with multiple design iterations. Design modifications can be
made easily and be verified with verification tools. Reprogrammability of the FPGAs and
CPLDs also give the designer the ability to try different design revisions in order to
choose the most optimized design.

1.3.1.4 A design entry language
Just as high-level programming languages allow complex design concepts to be
expressed as computer programs, VHDL allows the behavior of complex electronic
circuits to be captured into a design system for automatic circuit synthesis or for
system simulation.
Like PASCAL, C and C++, VHDL includes features useful for structured design
techniques, and offers a rich set of control and data representation features. Unlike
these other programming languages, VHDL provides features allowing concurrent
events to be described. This is important because the hardware described using VHDL
is inherently concurrent in its operation. Users of PLD programming languages, such
as ABEL, PALASM, CUPL and others, will find the concurrent features of VHDL quite
familiar. Those who have only programmed using software programming languages
will have some new concepts to grasp.

1.3.1.5 A simulation modeling language
In 1980s the designers used schematic capture and gate level simulation tools to design and validate the functionality of their designs. With the increasing complexity of designs and as the performance demands accelerate, the tried and true methods of schematic capture and gate-level simulation were not enough. Using VHDL, the design of a system can begin at a very high-level of abstraction. Such a high-level model can be simulated and its functionality checked against specifications.

VHDL has many features appropriate for describing (to an excruciating level of detail) the behavior of electronic components ranging from simple logic gates to complete microprocessors and custom chips. Features of VHDL allow electrical aspects of circuit behavior (such as rise and fall times of signals, delays through gates, and functional operations) to be precisely described. The resulting VHDL simulation models can then be used as building blocks in larger circuits, (using schematics, block diagrams or system-level VHDL descriptions) for the purpose of simulation.

1.3.1.6 A verification language

One of the most important applications of VHDL is to capture the performance specification for a circuit, in the form of what is commonly referred to as a test bench. Test benches are VHDL descriptions of circuit stimuli and corresponding expected outputs that verify the behavior of a circuit over time. Test benches should be an integral part of any VHDL project and should be created in tandem with other descriptions of the circuit.
1.3.1.7 A netlist language

While VHDL is a powerful language with which to enter new designs at a high level, it is also useful as a low-level form of communication between different tools in a computer-based design environment. VHDL’s structural language features allow it to be effectively used as a netlist language, replacing other netlist languages such as EDIF. Using VHDL the interconnections among the digital system components can be described.

1.3.2 VHDL Shortcomings

Designers have three concerns about implementing a design using VHDL and they are as follows:

- The designer does not have control of the gate level implementation of the design due to high level abstract constructs.
- The logic implementation of the design by synthesis tools is sometimes inefficient.
- The quality of the synthesis tools varies from one tool to another.

The very first concern is difficult to get around, because the intent of VHDL was to relieve designers form having to specify gate level implementation of a circuit. Compilers that synthesize logic are designed to optimize most constructs eliminating the need for the designer to specify. Many synthesis tools allow the designer to use synthesis directives providing limited control over the implementation and optimization of the logic. The synthesis directives force the compiler to optimize the design in terms of area
or speed constraints. Many synthesis tools allow the designer to specify gate level implementation that is technology specific based upon the internal architecture of logic cells and routing resources.

The second problem is that logic synthesis is inefficient. Synthesis tools do not always offer the optimal solution for a logic function because the solution depends on the design objectives. Compilers use algorithms to implement logic in a standard way not always looking at the design problem in a specific way. One solution to the problem is to code the design in a way that the designer has control over the implementation process. Inefficient VHDL code results in redundant logic resulting in a decrease in performance and increase in the area.

The last shortcoming for VHDL addresses the quality of the synthesis tools. Fortunately, VHDL synthesis technology for CPLD and FPGA designs is becoming standard and well beyond the beginning stages and this is becoming less of an issue.

These shortcomings do not take away form the advantages of implementing a design using VHDL. Design engineers need to keep in mind the design objectives and not to be concerned with the details of the design implementation. The design objectives are to fulfill the design requirements, within cost and schedule.

1.3.3 Other hardware languages

There are currently two main Hardware Descriptive Languages, VHDL and Verilog. Verilog syntax is not as complicated as VHDL and is less verbose. It lacks features and capabilities that VHDL can provide. Verilog is easier to grasp and understand and its
constructs are based on 50% C programming and 50% ADA. EDA environments support them both with documentation, simulation and synthesis capabilities.

1.4 FPGAs

FPGA which stands for Field Programmable Gate Array is a logic chip that can be programmed multiple times. FPGA architecture will be discussed in later chapters in details but for now we can define them as a matrix of logic cells. The routing resources and interconnects are located between the logic cells. FPGAs have changed dramatically over the past few years compared to 20 years ago when they were first introduced. In the past, FPGA were primarily used for glue logic, prototyping and lower volume applications while custom ASICs were used for intense logic applications, high volume and cost sensitive designs. FPGAs had also been too expensive and too slow for many applications. Integration plus development tools were often difficult to learn and lacked the features found in ASIC designs.

Now, this has all changed. Silicon technology has progressed to allow chips with tens of millions of transistors. This not only promises new levels of integration onto a single chip, but also allows more features and capabilities in reprogrammable technology. With today’s deep sub-micron technology, it is possible to deliver almost 10 million system gates in an FPGA. In addition, the average ASIC design operating at 50-100 MHz can be implemented in a FPGA using the same RTL synthesis design methodology as ASICs. By the year 2004, the state-of-the-art FPGAs will be operating at speeds surpassing 300 MHz. Many designs, which previously could only achieve speed and cost-of-density
goals in ASICs, are converting to much more flexible and productive reprogrammable solutions.

The availability of FPGAs in the 1-million system gate range has started a shift of SoC designs towards using Reprogrammable FPGAs, thereby starting a new era of System-on-a-Reprogrammable-Chip (SoRC). SoRC is no different from SoC in that it requires leveraging existing intellectual property (IP) to improve designer productivity. Reusable IP is essential to constructing bug-free multimillion gate designs in a reasonable amount of time. Without reuse, the electronics industry will simply not be able to keep pace with the challenge of delivering the “better, faster, cheaper” devices that consumers expect. With the availability of new FPGA architectures designed for system level integration and FPGA design tools that are compatible with ASIC design methodologies, it is now possible to employ a similar if not identical design reuse methodology for ASICs and FPGAs. For design teams that have intended to eliminate NRE costs and improve time-to-market without going through the learning curve to become FPGA experts, this design reuse methodology adds a new level of freedom. However, this methodology is not without its limitations. There are still some unresolved issues in high speed designs that make the FPGA’s inefficient.

1.4.1 System-on-a-Reprogrammable-Chip (SoRC)

Most of the industry agrees that SoC is the incorporation of an entire system onto one chip. The definition of SoRC is just beginning to evolve and in this book is defined as system level integration implemented on a Reprogrammable FPGA device. The SoRC definition is similar to SoC since it generally includes a compute engine, user logic gates
and on-chip memory. The definition of SoRC includes partial system integration into
multiple chips as well as entire system level integration on a single chip. The challenges
facing SoC and SoRC designers are similar even if the entire system is not integrated into
one single chip. However SoRC is not defined as the gathering of glue logic, since SoRC
designs contain system level integration issues that separate them from a general glue
logic design.

1.4.2 Why FPGA?
System-Level-Integration (SLI) using reprogrammable FPGA technology is made
possible by advances in IC wafer technology especially in the area of deep submicron
lithography. Today, state-of-the-art wafer fabs find FPGAs as an excellent mechanism for
testing new wafer technology because of their re-programmable nature. Incidentally, this
trend in the wafer fabs means that FPGA companies have early access to the newest deep
sub-micron technologies, dramatically increasing the number of gates available to
designers as well as reducing the average gate cost sooner in the technology life-cycle
than before. This trend, together with innovative system level architecture features, is
leading FPGAs to become the preferred architecture for SLI. Listed below are some of
the factors an FPGA designer will have to be aware of:

1.4.2.1 Flexibility
System-Level-Integration (SLI) increases the complexity of implementation into the
target device. Reprogrammable logic eliminates the risk and expense of semi-custom and
custom IC development by providing a flexible design methodology. Systems designed in
FPGAs can be prototyped in stages, allowing in-system testing of individual sub-modules. The design engineer can make design changes in minutes avoiding multiple cycles through an ASIC manufacturing house.

1.4.2.2 Faster Testing and Manufacturing

ASICs require rigorous verification strategies to avoid multiple cycles through manufacturing. The manufacturing and test strategies must be well defined during the specification phase. Often different strategies must be used depending on the type of block. Memory blocks often use BIST (Built In Self Test) or some form of direct memory access to detect and troubleshoot data retention problems. Other logic such as microprocessors requires custom test structures for full or partial scan or logic BIST.

FPGA designs, on the other hand, are implemented on reprogrammable devices that are 100% tested by the manufacture, before reaching the designer. In fact, FPGAs can be used to create test programs downloadable into other non-reprogrammable devices on the board. There is no nonrecurring engineering (NRE) cost, no sign-off, and no delay while waiting for prototypes to be manufactured. The designer controls the entire design cycle, thereby shrinking the design cycle as well as the time to prototype. This allows essential steps such as firmware designing to occur at a stage late in the design flow but actually earlier in the actual design time.

1.4.2.3 Verification
ASIC technology requires strict verification before manufacturing. In large complex system level designs many more unexpected situations can occur. SoRC designers have a much more flexible in-system verification strategy. The designers can mix testbench verification strategies with in-circuit testing, thereby offering faster verification without the cost of accelerated simulators. Surveys of design engineers have found that the area of test vector generation and vector-based verification is the least favorite part of system design process.

**1.4.2.4 Hardware and Software Co-Designing**

FPGAs are inherently flexible in a sense that design modifications can be done at any point of the design cycle. This feature gives the designer the advantage of having concurrent hardware and software design. The board layout can begin while the internal FPGA design is being completed. This allows to try different approaches in a design iteratively and improve design performance as well as decrease in time to market. At anytime hardware changes can be made and design improvements can be incorporated. Software and hardware integration can also be risky in some cases and designer needs to take into account some important factors. For the most part, design performance, routability and pinout are the factors to consider.

**1.4.2.5 Reconfigureability**

Reconfigureable computing term was brought up after introduction of SRAM based FPGAs. Most often initial design is not exactly what it is supposed to be. There are
always last minute changes or even when the product is deployed in the field. It is very advantageous to have the capability of modifying the design at any time.

FPGA devices provide an array of logic cells and routing resources that could be altered to create any logic. Device gets configured on power-up of the system. Different design configurations can be downloaded to the device on each power-up. There are two categories of FPGAs today. There are traditional FPGAs that span from low to high densities and there are second generation FPGAs (such as Xilinx Virtex-II FPGAs). The former is mainly replacing glue logic to high-density gate arrays and ASICs. The latter can be split into two parts: an embedded processor and a reconfigurable logic.

### 1.4.2.6 ASIC Verification

ASIC designers are under pressure to produce designs that provide maximum performance and integration. They also have to respond the time to market demand, and quickly produce a fully functioning device. In that sense verification is the biggest issue that they confront in this challenge. Verification consumes 50 percent of today’s ASIC design cycle and this rate will grow over the following years since as design size grows, the number of test vectors needed to adequately verify the design grows even faster.

Continually advancing process technology, and the corresponding explosion in design size and complexity, have led to verification problems that are impossible to solve using traditional software simulation tools. In addition, an increasing number of applications that process large quantities of data in real time such as video require verification techniques that run at near real-time speeds. As a result, a growing number of ASIC designers are building prototype boards using FPGAs to verify their ASIC designs.
Another advantage of building prototype boards using FPGAs is that it allows software
development and debugging to happen in parallel with the design and debug of the final
ASIC, reducing time to market.

One point that a designer should be aware of is that even though this approach offers
numerous advantages as a verification tools but it also has its own drawback since it
introduces delay to the design cycle.

1.5  ASICs

A chip designed for a specific application for a specific product is called an ASIC or
Application Specific Integrated Circuit. An ASIC can be pre-manufactured for a special
application or it can be custom manufactured (typically using components from a
"building block" library of components) for a particular customer application. The details
of ASICs will be covered in future chapters.

ASICs normally increases reliability and performance while reduces power consumption,
printed circuit space and eventually cost per unit volume. There is always Non Recurring
Engineering (NRE) cost associated with the ASICs. This is due to the different costs
involved in testing, manufacturing and designing the internal logic of the chip as well as
making first samples of the silicon. The value of NRE is normally $25000-$100,000 for
low to mid range density devices. This high upfront cost can only be justified if the sales
volume of the product is high enough to offset it. Rapid performance growth and cost
reduction trend of FPGAs has made this justification even more difficult nowadays. The
broad definition of ASIC ranges from semi-custom ICs to fully customized design. Semi-
custom ASICs offer preprocessed chips to which the designer only needs to add the final metallization connection steps while in full custom ASIC, designer needs to build the entire chip layer by layer.

1.5.1 System on a Chip (SoC)

A system-on-a-chip (SOC) design is an integrated circuit that contains a processing engine, memory and logic on a single chip. Driven by the rapid growth of the Internet, communication technologies, pervasive computing, and wireless and portable consumer electronics, Systems-On-Chip (SOC) has become a dominant issue in today’s ASIC industry. SOCs have created new challenges in Design Methods, Design Tools, Design Automation, Manufacturing, Technology, and Testing. Suppliers of embedded solutions are increasingly looking to system-on-a-chip (SOC) solutions to satisfy stringent time-to-market and price requirements in a competitive market. Growing demand for more intelligence on lower-cost, specialized chips, SOC designs has caused dramatic changes in terms of complexity, number of cores, and diversity of functions over the past few years. Today, SOCs are more sophisticated devices incorporating many applications to satisfy the market demand.

1.5.2 Why ASICs?

Growth in the demand for bandwidth and advanced functions is outpacing the performance of the standard off the shelf ICs software based solutions. ASIC technology is now so advanced that many functions traditionally implemented in software can be
migrated to ASICs. They can also consolidate the work of many chips into a single, smaller, faster package, reducing manufacturing and support costs while boosting the speed of the device built with them. More specifically, ASICs let designers use the power of constantly improving silicon technology to build devices targeted at specific functions. The process of designing ASICs requires an incredible amount of effort for customer requirements and target market. Re-spinning silicon after rollout means disaster and consequently losing market share. As chip densities have increased, ASIC development is now even more cumbersome, risky and pricy. Only the largest companies can afford the time, development costs and risk for developing an ASIC. Listed below are some factors an ASIC designer will have to be aware of:

1.5.2.1 Performance

Clock speed of digital circuits increase with each migration to lower size geometry. Today full custom ASIC designs run at clock speeds up to 2.5 Ghz. Most of the semi-custom ASIC designs do not run at that speed. This is due to the limitations in current EDA tools and design methodologies. Clock distribution networks and design partitioning should be addressed at early stages of the design cycle in order to achieve the performance needed for a specific application.

1.5.2.2 Power Consumption

In order to reduce the power consumption of chips, the supply voltage needs to be reduced. As supply voltage continue to decrease, noise immunity can turn into a major issue. Also small geometries can affect the static leakage current, which increases the
power consumption of the device. The designer has to perform power calculations according to the data that an ASIC vendor provides. These calculations then will be included in the design specifications.

1.5.2.3 ASIC Design Tools and Vendor

Selecting a good design tool set as well as a good vendor can determine the success of and ASIC development. An unwise selection can prolong a schedule and put sever strain on budget. There are lots of questions that need to be answered in order to make the right decision. First of all of course is the overall cost of the ASIC which includes both ASIC design tool set and development cost. Then factors such as the development time, inputs from previous users of the vendor, existing development tools (if any), vendor data on technology and tools are other things to consider. To meet demands for faster, better and cheaper systems, ASIC designers face tremendous challenges and they need to overcome the obstacles wisely. One other important aspect is the manufacturing stage that has a significant impact on expenses. Manufacturing process includes three phases: wafer fabrication, wafer probe (test) and assembly. The vendor also takes the responsibility of packaging the parts.

1.5.2.4 Partitioning

With the SOC design size approaching multi-million gates, partitioning the main design into sub-modules and constituting a hierarchical design can help address the capacity issues. The designers may partition the system into one or more modules. System architects have a good idea how to accomplish this task. Design specifications are not
complete at this point but a good architect always avoids repartitioning the system in the middle of the design phase. Factors such as testability and performance requirements should be taken into account in functional partitioning of the design.