Performance and reliability usually play the most important role in designing the digital system. Optimal performance and high reliability can be achieved, if optimization issues are considered almost in each step of the design process. Optimization of important design objectives such as size of the circuit, critical delays, power dissipation are included in most of the today's design tools from a high level synthesis tools to the logic and layout synthesis tools.

Design Synthesis

The classical approach to design an ***A**pplication **S**pecific **I**ntegrated **C**ircuit* (ASIC) either in a fixed monolithic semiconductor technology such as *gate arrays* (GA) or *standard cells* (SC) or in programmable technology like Field Programmable Gate Arrays (FPGAs) is divided into several phases. The design cycle model, as shown in Figure 1.1, forms a framework within which subsequent steps in a design flow may be meaningfully integrated to produce a functional, reliable product. Here we identify four major phases in a design process, namely *high level synthesis, logic synthesis, layout synthesis* and *manufacturing*.

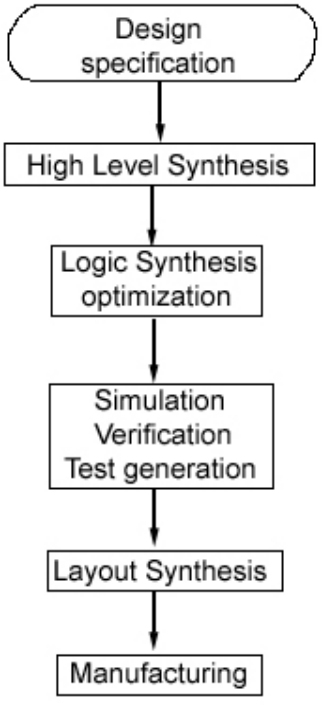

Figure: Phases in VLSI design synthesis

The process usually begins with a product requirement, which results from interaction between marketing and prospective costumers. An engineering response to the proposed product takes the form of a product definition document, which addresses issues of feasibility, general methods or techniques, cost, technical risk, trade offs between different approaches, and similar items necessary to get product developments approvals. A general design specification evolves into a precise technical description, where focus is on the external behavior of the circuits and communications between the circuit and its external environment rather that the internal realization.

The main objective of high level synthesis is to begin with the circuit specification details, transforming them into a high-level description of how the circuit will be structured, what major functions need to be accomplished within the circuit and how these functions can be realized as an interconnection of smaller circuit constituent to meet the circuit specification details. The high level synthesis phase takes an abstract behavioral representation of a digital system and produces a *register-transfer level* (RTL) structure that realizes a given behavior. At this point in the design,

we will see each functional block being defined in terms of interconnected registers, multiplexers, control elements, and the like. This is usually a critical phase in the evolution of the circuit design, for the designer must ensure that the behavioral description being produced matches both performance and functional specifications. Describing a complex digital system with hundreds of thousands of gates at the gate level may be an extremelly challenging task, and therefore, a high level synthesis tools were introduced. The VHSIC (very high speed integrated circuit) Hardware Description Language, known as VHDL was adopted as an industry standard in 1987 and used to describe hardware from the abstract behavioral to the concrete level. VHDL was defined because a need existed for an integrated design and documentation language to communicate design data between various level of abstractions.

The logic synthesis begins with the RTL description and a collection of logic primitives. The primitives are usually determined by a selected implementation style, and may be a collection of available gates, flip-flops, control functions, etc. Each functional block described in the behavioural design phase is transfered into a description which consists of logic primitives. They are interconnected in a manner that satisfies both functional and performance portions of the circuit specification. Logic design usually begins with a straightforward transformation of the RTL description into equivalent structure, expressed in terms of logic primitives. The main objective of logic synthesis tools is to transform RTL description in such equivalent structure of logic primitives, that either size or performance in terms of critical delay or combination of both are minimized.

The layout synthesis traditionally consists of two major steps, namely *placement* and *routing*. In a placement phase, logic primitives are assigned to physical location in the carrier environment selected for realization of the circuit, with the objective being to ease interconnection wiring design. Typically, placement algorithms attempt to minimize the total expected length of interconnect required for the resulting placement. In some design styles, such as for example gate arrays, other important issues must also be considered during the placement phase. For example, in FPGAs limited routing resources, routing channel congestion and routing delays must also be considered during the placement phase. In a routing phase, placed logic primitives are interconnected to form a desired logic design. Routing algorithms need not only to ensure 100% routable design, but are target to minimize routing congestions and required routing space, as well as routing delays, that are imposed with the parasitic effects on routing resources. After the layout phase, fabrication depends on selected technology and design style. In case of semi-custom design style such as standard cells and gate array, masks are created for the fabrication in a technology center, while for implementation in a programmable device, devices only need to be programmed.

The division into multiple design steps is artificial and has been used to reduce the complexity of the design process. However, such design process has some drawbacks. High level synthesis tools often lack of information about the efficiency of logic synthesis tools and layout considerations and limitations since they based on abstract models. Decisions, made in earlier phase of the design process may inhibit the ability to meet specified requirements such as area, power consumption, critical delay at the

physical design level that comes later. On a contrary, synthesis and optimization tools at the lower level are based on the representation of the circuit consisting of logic primitives, and thus may provide an exact information about the feasibility of the implementation and performance issues. They derive its efficiency through the local transformations of the circuits, while they often lack global view of the whole circuit. Since design tools work more or less independently from each other, target objectives of each individual tool may produce oppositional effect.

Due to the large complexity of digital circuits, discontinuous nature of the objective functions and to the discrete nature of the design space, the optimization problems in a circuits design cannot be solved with the analytical methods in most cases, but are usually solved using algorithmic approach. An *exact* algorithm always provides the exact solution, often called *global optimum* of the optimization problem. Unfortunately, some exact algorithms may have such a high computational cost that their use can be prohibitive on problems of typical size. Hence, there is a need for *approximation* algorithms, which are not guaranteed to find the exact solution of the optimization problem, but can provide good approximation of the exact solution, which may be valid for practical applications. Approximation algorithms are often called *heuristic algorithms*, because they use problem-solving techniques based on experience. *A main practical result of this work* is a set of well defined and efficient heuristic algorithms implemented as computer programs which solve a specific problem in layout synthesis of partitioning large circuit into multiple partitions combined with concepts from logic synthesis, considering various objectives and constraints imposed with the technology limitations.

Application in VLSI CAD

Partitioning is a technique widely used to solve diverse problems occurring in VLSI CAD. Applications of partitioning can be found in logic synthesis, logic optimization, testing, and layout synthesis.

High-quality partitioning is critical in high-level synthesis. To be useful, high-level synthesis algorithms should be able to handle very large systems. Typically, designers partition high-level design specifications manually into procedures, each of which is then synthesized individually. However, logic decomposition of the design into procedures may not be appropriate for high-level and logic-level synthesis . Different partitionings of the high-level specifications may produce substantial differences in the resulting IC chip areas and overall system performance.

Some technology mapping programs use partitioning techniques to map a circuit specified as a network of modules performing simple Boolean operations onto a network composed of specific modules available in an FPGA.

Since the test generation problem for large circuits may be extremely intensive computationally, circuit partitioning may provide the means to speed it up. Generally, the problem of test pattern generation is NP-complete. To date, all test generation algorithms that guarantee finding a test for a given fault exhibit the worst-case behavior requiring CPU times exponentially increasing with the circuit size. For most practical cases, it has been observed that the time taken to generate tests is $O(n^3)$, where n is the number of gates in the circuit. If the circuit can be partitioned into k parts (k not fixed), each of bounded size c, then the worst-case test generation time would be reduced from $O(n^3)$ to $O((n/c)(c^3))=O(n-c^2)$, and thus become linearly related to the circuit size.

Partitioning is often utilized in layout synthesis to produce and/or improve the placement of the circuit modules. Partitioning is used to find strongly connected subcircuits in the design, and the resulting information is utilized by some placement algorithms to place in mutual proximity components belonging to such subcircuits, thus minimizing delays and routing lengths.

Another important class of partitioning problems occurs at the system design level. Since IC packages can hold only a limited number of logic components and external terminals, the components must be partitioned into subcircuits small enough to be implemented in the available packages.

Partitioning has been used as well to estimate some properties of physical IC designs, such as the expected IC area.

Need for Partitioning and its Applications

Systems with several million transistors are now common, presenting instance complexities that are unmanageable for existing logic-level and physical-level design tools. Partitioning divides a system into smaller, more manageable components; the number of signals that pass between the components corresponds to the interactions between the design subproblems. In a top-down hierarchical design methodology, decisions made early in the system synthesis process (e.g., at the system and chip levels) will constrain succeeding decisions. Thus, the feasibility - not to mention the quality - of automatic placement, global routing and detailed routing depends on the quality of the partitioning solution.
A bottom-up clustering may also be applied to decrease the size of the design, typically in cell or gate-level layout. The current emphasis on a quick-turnaround ASIC design cycle reinforces the need for reliable and effective algorithms.

Partitioning heuristics also have a greater impact on system performance as designs become interconnect-dominated. In current submicron designs, wire delays tend to dominate gate delays; the differences between on-chip and of-chip signal delays and the increasingly pin-limited nature of large chips make it

desirable to minimize the number of signals traveling of a given chip. Larger die sizes imply that long on-chip global routes between function blocks will more noticeably affect system performance. Other considerations (e.g., design for testability, low-power design, etc.) also require partitioning algorithms to identify interconnect structure, albeit at more of a functional or communication-based level.

Finally, partitioning heuristics affect the layout area: wires between clusters at high levels of the hierarchy will tend to be longer than wires between clusters at lower levels, and total wirelength is directly proportional to layout area due to minimum wire spacing design rules. The traditional minimum-cut objective is natural for this application: if the layout area is divided into a dense uniform grid, total wirelength can be expressed in “grid" units or equivalently as the sum over all gridlines of the number of wires crossing each gridline. This view can also improve auto-routability since it suggests reducing the wire congestion in any given layout region.

Today, leading applications of partitioning include:

1.**General design packaging**. Logic must often be partitioned into clusters, subject to constraints on cluster area as well as possible I/O bounds. This problem is known as **design packaging** and is still the canonical partitioning application; it arises not only in chip floorplanning and placement, but also at all other levels of the system design. The design packaging problem also arises whenever technology improves and existing designs must be repackaged onto higher-capacity modules (“technology migration"). Note that the problem is usually associated with large cluster sizes, with few constraints on the internal structure of the clusters. When a finite library of available module types is specified, the optimization is more along the lines of “covering" or “technology mapping".

2. **Netlist-level partitioning in HDL-based synthesis**. Synthesis tools have emerged which reduce the design cycle by automatically mapping a high-level functional description to a gate or cell-level netlist. However, even with increased maturity of such high-level synthesis tools, netlist partitioning remains central to the success of the design procedure. This is essentially because writing HDL code - as opposed to performing layout - can abstract away the physical layout effects of design choices (for example, a few lines of HDL code that specify a register file or crossbar connection can correspond to a large portion of the final layout area). As a result, the block decomposition of the functional (software) description does not necessarily map well into a decomposition of the physical layout. Hence, in contrast to previous building block methodologies, which yielded a small number of function blocks that could be optimally hand partitioned, HDL-based synthesis virtually requires the physical design methodology to shift from working with a small number of building blocks to working with large, flattened design representations. Partitioning of flattened

inputs is also necessary for such applications as the design of “precursor systems" (i.e. finding the packaging tradeoffs that correspond to optimum cost-performance points at early stages in the product life cycle).

3. **Estimation for design optimization.** Accurate estimation of layout area and wireability has always been a critical element of high-level synthesis and floorplanning. Now, such estimates are becoming critical to higher-level searches over the system design space. Predictive models often combine analysis of the netlist partitioning structure with analysis of the output characteristics of placement and routing algorithms, in order to yield estimates of wiring requirements and system performance. This use of system partitioning hierarchies is increasingly prominent as “design optimization" and “electronic system design automation" captures the attention of CAD users and vendors.

4. **System emulation and rapid prototyping (FPGA partitioning).** Many logic emulation systems and rapid system prototyping methodologies (e.g., those from Quickturn or Zycad/Inca) use partitioning tools to map complex circuit designs onto hundreds or even thousands of interconnected FPGAs. Typically, such partitioning instances are challenging because the timing, area, and I/O resource utilizations must satisfy hard device-specific constraints. Furthermore, the partitioning optimization is affected by the discrete nature of system resources - e.g., interconnect delay in routing segments, layout area in configurable logic blocks (CLBs), or individual FPGA chips in a multiple-FPGA system - all of which have large “quanta".

5. **Hardware simulation and test**. A good partitioning will minimize the number of inter-block signals that must be multiplexed onto the bus architecture of a hardware simulator or mapped to the global interconnect architecture of a hardware emulator. Reducing the number of inputs to a block often reduces the number of test vectors needed to exercise the logic.