A Conceptual Framework for Designing ASIC Hardware*

Steven S. Leung and Michael A. Shanblatt

Department of Electrical Engineering, Michigan State University
East Lansing, MI 48824

Abstract

A conceptual framework consisting of the design process, the design space, and the design repertoire for ASIC hardware is presented. An Inter-Level Design Process Model (ILDP) is proposed as a general model for expressing and implementing hierarchical design methodologies. The proposed conceptual framework is an effective instrument for bridging the increasingly wider gap between application engineers and VLSI designers.

I. Introduction

The term ASIC (Application-Specific Integrated Circuit) conveys in its meaning a mixture of aspects of implementation technology, design approach, market orientation and the subsquent product requirements. Currently, however, the dominant interpretation refers to semi-custom designs implemented in gate arrays, with design freedom only in interconnects, or standard cells, which require generation of all mask layouts. As logic synthesis technologies mature and CAD/CAE systems become more powerful, ASIC technology is destined to become the mainstream of VLSI hardware design. The strategic significance of this emerging technology is recognized as being comparable to the previous microprocessor revolution and failure to take full advantage of it will have long term adverse effects on the health of the U.S. semiconductor industry, and subsequently, the national economic strength.

Despite its importance, the ASIC revolution is creating an increasingly wider gap between application engineers and VLSI designers. This is because integration of circuits is, above all, an integration of knowledge and expertise which requires a prolonged training period. With a real shortage of existing VLSI chip designers, the challenge is for the VLSI design community to adopt a simple yet complete and expandable conceptual model that allows application engineers and system designers to rapidly acquire the necessary knowledge and communicate design information more effectively. In this paper, we describe a conceptual framework which organizes the knowledge of IC system design into three knowledge frames: *design process, design space* and *design repertoire*. Our discussion of these concepts will focus on one particular step — the transformation from task algorithm to architecture.

* This research was supported in part by the State of Michigan Research Excellence and Economic Development Fund.

II. The Design Process

As a technology matures, major breakthroughs occur less frequently and more randomly. At the same time, many low level mapping functions are carried out by computers. As a result, design as a decision-making process, that is, the selection of a solution from a number of alternatives according to a set of cost/performance criteria, will dominate the design process. From this perspective, the two most important questions are: *what* are the design decisions and *how* are they to be made?

Design decisions can be classified into four categories: software/hardware tradeoffs, technology choice, style, and choice of hardware algorithms. Detailed discussions of these decision tradeoffs can be found elsewhere [1-3]. In this paper, we are mainly concerned with how the design decisions should be made.

The fundamental attribute that governs many aspects of the VLSI design process is the unprecedented complexities which have arisen from the combination of the numerous tightly-coupled intermediate steps and the seemingly unlimited freedom of design choices in each step. As a result, the design methodology, as well as its implementation, characterizes the VLSI design process.

1. Design Methodology and Its Implementation

Lying at the core of any VLSI design methodology is a hierarchy of design steps. The hierarchical approach partitions various aspects of VLSI circuits into abstraction levels and defines the order among these levels. Even though there exists no standard with respect to either the abstraction levels or the partition/ordering process, the partitioning of the design hierarchy into layers of task specification, architecture definition, behavioral model, functional structure, logic structure, transistor/switch model, mask layout, and fabrication and testing is gaining popular acceptance [4].

The design process can be described from a hierarchical point of view as successive transformations of specifications from one domain (or abstraction level) to another. CAE tools were first developed individually and later combined to form more powerful automated logic synthesis systems. It is recognized that the side effects of such an *ad hoc* approach are becoming an unbearable burden for the development of integrated design systems. Incompatible data formats and the lack of any industry standard are at least partially responsible

for the slow proliferation rate of these systems. The moral seems to be that the success of a hierarchical design methodology depends on how well its underlying principles reflect the nature of VLSI circuitry and on how well its external expression supports the implementation effort. We informally call these the necessary and sufficient conditions for efficient implementation of design methodologies.

2. *The Inter-Level Design Process Model (ILDP)*

In order to implement the hierarchical design methodology efficiently, a unified conceptual model of the design process is indispensable. We propose a conceptual view of the emerging inter-level design process as illustrated in Fig. 1. Major components of this model are outlined below.

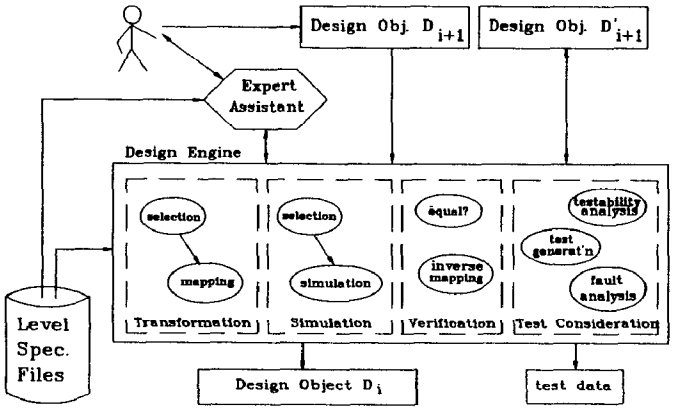

Figure 1. The Inter-Level Design Process Model.

• *The Design Object*

The design object of level i is a description of a target design in terms of the abstractions (primitives) defined at that level. It is viewed as a collection of instantiations of abstract objects or a point in the level i design space. Through the use of languages, the level i properties of the object are expressed in terms of the three basic attributes of a language: the data format (typing), the syntax, and the semantics. Through these three attributes human and machine can communicate.

• *The Design Engine*

The concept of a design engine evolves from four subprocesses: *transformation, verification, simulation,* and *test consideration.* They are uniformly present at all design levels. The transformation process can be further partitioned into two mechanisms: mapping and selection. Mapping represents purely mechanistic and routine procedures (e.g., a transistor is formed by a diffusion region crossed by a polysilicon path) whereas selection represents the most important decision-making function of the designer (e.g., select a particular transistor implementation from a pool of alternatives given an area-time tradeoff).

Verification is achieved by first generating another level i+1 design object through inverse mapping and then by an equivalence test of the two level i+1 object descriptions. This requires the existence of the inverse mapping and a canonical representation of the design object. Currently, this approach is only applicable to some well understood low levels such as logic-gate or gate-transistor levels [5].

The simulation process serves three purposes: verification, performance improvement, and fault coverage indication. Since inverse mapping is not well developed, verification is achieved by extensive simulation. The selection mechanism within this process determines which simulation mechanism is to be used and whether global or local simulation is to be performed.

An important fundamental in VLSI design is the role of design for testability (DFT). Without an upfront objective of DFT, testing of custom or semicustom circuits of high complexity is virtually impossible [6]. The test consideration process includes testability analysis, test vector generation, and fault coverage estimation.

• *The Expert Assistant*

As VLSI design knowledge continues to accumulate, the major function of a system architect shifts towards evaluating existing alternatives rather than creating new ones. An awareness of various alternatives is thus the precondition for making judicious design decisions. Expert assistant systems can be developed to relieve the designer's burden by maintaining a base of static knowledge and keeping the designer aware of various alternatives at each critical point. Moreover, as the boundary between unconstrained and semi-custom designs becomes blurred, a complete path of gate array to megacells, to standard cells, and to unconstrained design is expected to emerge in the next few years. With this outlook, the expert assistant may play an even more important role in assisting the designer.

• *The Level Specifier*

Level-specific information contained in the specifier may be divided into four categories: design object data format, process function, constraint-criteria template, and design schema. Design object data format is used by both the design engine and the expert assistant. Process functions include mapping, inverse mapping, simulation, and other level-specific functions. All are used exclusively by the design engine. On the other hand, both the constraint-criteria templates and the design schema are used by the expert assistant. The constraint-criteria templates define points in the design where alternatives should be considered. The design schema serves as a roadmap or overall procedural guide for conducting the entire design process.

To represent the design process in terms of an inter-level model is simple and has built-in consistency; learning efforts are thus minimized. In addition, the open-architecture style of ILDP allows logic synthesis systems to be built from existing modules or from modules supplied by different vendors. Concurrent design activities at different levels are also possible.

III. The Design Space

While the dynamic aspects of VLSI design are abstracted to form the design process frame and characterized by the ILDP, the static aspects are best captured in the notion of design space. Static aspects refer to those physical properties inherent in the design objects and are largely stable over time. The relationship between the design process and the design space can be represented by Gajski and Kuhn's

Y-chart in which the design process is formulated as a methodology traversing through the design space with more detailed information added toward the center [7]. The significance of the design space concept lies in its classification power allowing the designer to logically organize knowledge and enabling easier recognition of alternatives. However, the Y-chart does not address the details of individual design levels, and because of the multi-faceted nature of VLSI design, the framing of the design space is not unique. Here, we concentrate on the architecture space and the algorithm space.

1. Architecture Space

The architecture superspace is composed of three spaces: *functional modules, control,* and *data flow.* The functional unit space is spanned by three subspaces: arithmetic, logical and storage. Arithmetic units include conventional fixed-point and floating-point adders, multipliers, and the like. Logical units include flip-flops, shifters, counters, multiplexers, decoders, and other modules for logical operations. Storage units include latches, memories, and registers.

The control space has the synchronization, style, and structure subspaces. The synchronization subspace consists of synchronous and asyncronous control with each elaborated by clocking schemes and protocols. The control is also charaterized by the style of instruction execution such as centralized, pipelined, and/or distributed. The structure subspace includes implementation styles of random logic, finite state machine, and microcoding.

The data flow space contains the two subspaces of interconnect topology and data reference. Major interconnect topologies for data exchange networks include bus, ring, star, tree, crossbar, mesh, and shuffle networks. Data reference refers to various memory addressing schemes.

The advantage of this framing scheme is that different architectural primitives with similar high-level appearances tend to naturally cluster close to each other. Implementational alternatives for a given algorithm are thus more visible.

2. The Algorithm Space

The algorithm superspace is composed of three spaces: *data structure, operation,* and *dependency.* Data structure is the physical or visible form of the structural properties intrinsic to the task algorithm. The scope of this space is similar to the communication geometry described by Kung as illustrated in Fig. 2.

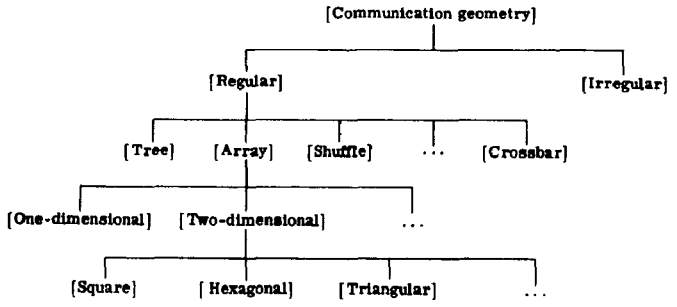

Figure 2. The data structure space [8].

The operation space is spanned by elementary computations seen by the designer. Since an operation is an event occurring in some entities or functional units, the operation space can thus be naturally aligned to the functional space. The actual position of a task algorithm in the operation subspace is hardware dependent. Consequently, the architect's role is then to determine how "elementary" the computation primitives, which are to be mapped directly into hardware functional units, should be.

Kuck has identified four kinds of program dependencies which constitute the dependency space [9]. They are data dependence, anti-dependence, output dependence, and control dependence. Regular control flow and data flow patterns, such as array operation and recurrence, can be expressed in terms of these dependencies.

At the task level, algorithms are developed with specific perspectives; the abstraction mechanisms or techniques at this level are primarily determined by the nature of the problem at hand. However, once the algorithm has been specified and hardware implementation is considered, the designer must adjust his/her perspective to that of the computer engineer. The framing of the algorithm space described above facilitates the transformation of a task algorithm into a hardware algorithm.

IV. The Design Repertoire

In the previous two sections, we have established two basic concepts of design, namely, the *process,* as formalized by methodology and implemented by CAE systems, and the *space,* as the collective physical properties of design objects. Through methodology we limit the search space and hence the number of alternatives. We recognize the existence of alternatives through the framing of the space. Then, the logical question to follow is: how should these alternatives be evaluated? The design repertoire is a collection of design and analysis techniques for evaluation of design alternatives. These techniques can be classified into three categories: *resource configuration, algorithm restructuring,* and *system partitioning.*

1. Resource Configuration

The objective of this approach is to determine the minimum resources needed to satisfy some cost-performance criteria. From the perspective of the architecture space, there are three types of resources as explained before: functional units, control, and data flow facilities. In general, these three types of resources are not completely orthogonal and decisions made in one dimension will inevitably affect the others.

Most techniques for this approach are derived from graph theory [9-11]. The basic idea is to model the elementary computations as nodes and the data dependency as arcs. Then the parallelism or the data dependency can be derived from the upper bound on the size or depth of the equivalent tree. To derive a minimum resource solution, the graph is partitioned into cliques (or clusters) which correspond to various resources. Serial-to-parallel transformation is then applied to generate various alternatives for evaluation.

2. Algorithm Restructuring

Algorithm restructuring techniques were originally developed as a software technique to decompose large problems (especially in matrix calculations) into smaller ones that could be efficiently implemented on a fixed computer architecture. Since regularity is one of the most desirable properties in VLSI design, techniques to restructure an algorithm can be used in ASIC designs to improve the performance.

The data structure and data communication (including processor-to-processor and processor-to-memory) are the most critical ingredients in the performance of an algorithm under a given architecture [12]. Therefore, restructuring techniques are aimed at either improving the regularity of the data structure or reducing the data communication. In many situations, the regularity of the data structure of an algorithm can be improved by simply applying elementary operations under the constraint of data dependency. The proper scheduling of instruction and data streams has great impact on the execution time and intermediate storage requirements. Scheduling techniques have been developed particularly for pipelined architectures and systolic arrays [12-14].

3. System Partitioning

The partitioning of a large system into smaller modules in order to satisfy the physical constraints of the packaging is not a new problem. Semiempirical techniques based on what is known as *Rent's rule* have been developed to relate the average number of pins per module to the average number of blocks per module [15]. There are indications that the partitioning problem has been considerably complicated by the ASIC design environment. For example, different combinations of gate array products and standard ICs can generate a large number of alternatives. Recently, Palesko and Akers presented an algorithm for logic partitioning to minimize the number of gate arrays [16], but the partitioning problem as a whole still depends much on the designer's experience and intuition.

V. Concluding Remarks

A conceptual framework for ASIC design has been described. It is composed of three knowledge frames of *design process, design space,* and *design repertoire.* They address different aspects of how design decisions are made. The global strategy is to limit the search space through methodology implemented by CAE systems. Alternatives are recognized through a proper framing of the design spaces. A set of techniques to search through the design spaces and evaluate the alternatives is classified.

The proposed conceptual framework equips the system-IC designer with a clear and coherent view of VLSI design activities and, at the same time, suggests a systematic way to acquire and accumulate VLSI design knowledge. In this respect, it provides new insights to engineering education for future IC system designers. This conceptual framework also gives CAE developers a unified and farseeing view to implement design methodologies in an integrated environment. Common understandings developed between CAE developers and IC designers can significantly speed up the transfer of ASIC technology to a much wider engineering community.

References

1. A. C. Hartmann, "Software or Silicon? The Designer's Option," *Proc. IEEE,* Vol. 74, No. 6, June 1986, pp. 861-874.
2. P. M. Solomon, "A Comparison of Semiconductor Devices for High-Speed Logic," *Proc. IEEE,* Vol. 70, No. 5, May 1982, pp. 489-509.
3. J. Y. Chen, "CMOS-The Emerging VLSI Technology," *IEEE Cir. & Dev. Mag.,* Mar. 1986, pp. 16-31.
4. S. W. Director et al., "Integrated CAD, CAM and CAT of VLSI Circuits and Systems: The CMU Perspective," *IEEE Des. & Test,* June, 1985, pp. 81-93.
5. H. Jacobs, "Verification of a Second-Generation 32-Bit Microprocessor," *IEEE Computer,* Apr., 1986, pp. 64-70.
6. E. R. Hnatek and B. R. Wilson, "Practical Considerations in Testing Semicustom and Custom ICs", *VLSI Design,* March 1985, pp. 20-42.
7. D. D. Gajski and R. H. Kuhn, "New VLSI Tools," *IEEE Computer,* Vol. 16, No. 12, Dec. 1983, pp. 11-14.
8. H. T. Kung, "The Structure of Parallel Agorithms," *Advances in Computers,* Vol. 19, Academic Press, 1980, pp. 65-111.
9. D. J. Kuck, "A Survey of Parallel Machine Organization and Programming," *Comp. Survey,* Vol. 10, No. 1, Mar. 1977, pp. 29-59.
10. A. M. Despain, "Notes on Computer Architecture for High Performance," in *New Computer Architectures,* ed. J. Tiberghien, Academic Press, 1984, pp. 81-98.
11. C-J. Tseng and D. P. Siewiorek, "Automated Synthesis of Data Paths in Digital Systems," *IEEE Trans. CAD,* Vol. CAD-5, N0. 3, July 1986, pp. 379-395.
12. D. B. Gannon and J. V. Rosendale, "On the Impact of Communication Complexity on the Design of Parallel Numerical Algorithms," *IEEE Trans. Comp.,* Vol. C-33, No. 12, Dec. 1984, pp. 1180-1194.
13. L. M. Ni and K. Kwang, "Vector-Reduction Techniques for Arithmetic Pipelines," *IEEE Trans. Comp.,* Vol. C-34, No. 5, May 1985, pp. 404-411.
14. S. Arya, "An Optimal Instruction-Scheduling Model for a Class of Vector Processors," *IEEE Trans. Comp.,* Vol. C-34, No. 11, Nov. 1985, pp. 981-995.
15. B. S. Landman and R. L. Russo, "On a Pin vs. Block Relationship for Partitions of Logic Graphs," *IEEE Trans. Comp.,* Dec. 1971, pp. 1469-1479.
16. C. A. Palesko and L. A. Akers, "Logic Partitioning for Minimizing Gate Arrays," *IEEE Trans. CAD of IC & Sys.,* Vol. CAD-2, No. 2, Apr. 1983, pp. 117-121.