June 1986

A design methodology for high-performance, general-purpose VLSI

Dan Hammerstrom

Roy Kravitz

A Design Methodology for High-Performance, General-Purpose VLSI

Dan Hammerstrom
Roy Kravitz[1]
Oregon Graduate Center
19600 N.W. von Neumann Dr.
Beaverton, Oregon 97006

Technical Report No. CS/E-86-008
June 1986

Abstract

As the size and complexity of high-performance VLSI grows, it is necessary to enforce a structured methodology to allow the proper control and abstraction of the various levels required when designing these devices. In this article we describe a Large Chip Methodology (LCM) that guides the design of complex, high-performance VLSI and provides a foundation and framework for CAD tool development.

[1]Intel Corporation, 2111 NE 25th Ave., Hillsboro, Oregon 97123.

INTRODUCTION

The last decade has seen an explosive increase in the complexity of large scale integrated circuits. This functionality has led to a set of components that offer a large range of performance and function, and that are revolutionizing electronics and computation. Unfortunately, our ability to utilize silicon of this density is sorely tested by its complexity. A typical 32-bit microprocessor may have 400,000 devices, logic that is described by several hundred pages of schematics, and a mask design with over 2 million rectangles. Furthermore, fully automatic synthesis cannot effectively handle these designs.

The question addressed by this article then is, "How does one reliably design highly optimized VLSI?" Since VLSI design is fundamentally an exercise in complexity management, a design methodology is necessary to control this complexity and to provide a structure for the organization of the design team, the CAD tools that the team uses to do its job, and a framework for future design evolution.

This article describes a Large Chip Methodology similar to that developed by Intel as part of the iAPX-432 component family development. Interested readers are referred to a previous paper [2] describing Intel's Large Chip Methodology.

Our Large Chip Methodology borrows heavily from Software Engineering techniques that have been developed to reliably build large, complex software systems and is based on the concepts of abstraction, decomposition, and step-wise refinement:

(1) *Decomposition* is the process of splitting the design into independent modules whose interfaces can be accurately and succinctly described. These modules can then be hierarchically combined.

(2) *Abstraction* is the hiding of unnecessary detail. For example, during the decomposition process, module interfaces are simplified to where only a small amount of function (implementation) is visible externally (specification).

(3) *Step-wise refinement* of the abstracted modules allows the design details to be added in a controlled and incremental manner. In other words, the design is begun as a set of abstracted interfaces and iteratively refined until the final, complete design has been obtained. This process continues hierarchically, until at the last level of refinement, the final design is complete.

The Large Chip Methodology, therefore, is a multi-level design hierarchy that enforces the above techniques in the design process. It is also a mental discipline that becomes a fundamental part of each engineer's thinking.

Another characteristic of the methodology is its continual evolution in response to changes in design techniques, and CAD and fabrication technologies. By providing a framework and focus for this change, the methodology helps integrate these improvements smoothly into the design process.

OVERVIEW

The design effort for a complex VLSI component may be broken into several distinct levels, each level being a step-wise refinement of the previous level. Design generally proceeds in a top-down fashion with more detail being exposed at each level. Figure 1 shows the design flow of the Large Chip Methodology.

Implementation begins with a Product Specification that clearly states the performance goals, cost goals, and essential features of the component. An *architecture* is then defined that meets or exceeds those requirements. For a processor, this definition could include the instruction set specification, register and memory models, and external pin definitions. The architecture specification is then transformed into a block or register transfer level implementation (or microarchitecture) of the state variables (e.g. registers) and interconnecting data paths needed to implement the architecture. For a microprogrammed processor, part of this task would include the microinstruction set definition. *Logic* and *circuit* design follow, resulting in a set of circuit schematics and a list of layout requirements. Finally, mask generation involves the translation of those schematics into the rectangles that actually comprise the various mask layers. These rectangles are placed according to the design rules for the process and result in a data base that is used to create the mask plates used in chip fabrication.

Each of these implementation levels can be broken into four specific activities:

(1) *Synthesis* is the design process itself. During synthesis, a designer begins with a description of the design at the next higher level and creates a description for the current level. Synthesis is generally manual, although automated synthesis tools are appearing.

(2) The *description* is a computer data base that is a result of the synthesis activity. An important element of the Large Chip Methodology is that descriptions at all levels (except the mask level which is only indirectly behavioral) are executable. For example, the description at the logic design level is a schematic data base that can be transformed into a logic simulation.

(3) *Evaluation* consists of checking the design against the goals set out by the Product Specification. The major design constraints evaluated are die size, power consumption, and performance. The evaluation is based on experience and on results obtained using the executable description of that particular level. For example, a logic simulator may be used to obtain cycle by cycle performance estimates.

(4) *Validation,* the last activity for a design level, is the key to producing functional die on the first iteration of silicon. Validation checks that the synthesized design for the level is functionally correct, or, more precisely, that it has retained the function of the previous level. Although implementation proceeds in a generally top-down manner, validation is bottom-up. The layout description is compared to the schematics, the schematics are compared to the microarchitecture description, and the microarchitecture description is compared to the architecture specification. Also, there are situations where several levels can be skipped during the validation process; for example, to close the loop on the entire design process, the architecture description is compared to the chip itself.

The remainder of the paper examines each activity for each level of the methodology. We then examine its evolution, as well as the results of the methodology actually being used at Intel that served as a basis for that discussed here.

ARCHITECTURE

The first step of any chip design is the specification and description of the chip architecture. Here we define architecture as the functional interface as seen by the user of the chip; this includes the instruction set and the bus interfaces.

Architecture Synthesis

The architecture level design begins with a Product Specification that includes a description of the general attributes of the architecture such as intended market segments, and performance and compatibility requirements. The result of architecture synthesis is the creation of a detailed Architectural Specification and a *macrosimulator* that is an executable description of the architecture.

Architecture synthesis is a difficult process where the architects develop the functional description of the part based on 1) the implementation technology that will be manufacturable when the design process is complete, and 2) the intended applications for the part being designed. Given a fixed set of resources (silicon area, device density, and time to market), the architects must select function that, in the long term, will maximize the applicability and market of the chip. Adding to the difficulty is the fact that no matter how experienced the design team, not all implementation constraints can be foreseen this early in the design process. As a result, the architecture level is by no means fixed when work begins at the next level down; thus requiring iterative improvement on the initially specified description.

Architecture Description

The primary result of the synthesis process is the macrosimulator, a computer program that simulates the instruction set of the target architecture. For processor[2] chips, the macrosimulator takes as input/output a physical memory image containing programs for the target architecture and their data. The simulator then "executes" the programs on the data, updating the memory image as necessary. Generally, only the architecture externally visible state (such as general purpose registers) is simulated. Architecture invisible (internal) state is not simulated, thus making the macrosimulator easier to write and debug, and execute, while keeping it independent of the actual implementation of the component. Occasionally, it is desirable to simulate certain internal architecture state (e.g. caching) to allow more accurate performance estimation.

The macrosimulator's most important characteristics (not necessarily in order of importance) are:

(1) it displays the externally visible architecture state;

(2) it is simple and easy to change - this is important because of the initial instability of the architecture;

(3) it has a simple user interface that allows single step, breakpoints, instruction tracing (of the simulated processor), and observation of all architecturally specified state variables - these functions are essential, since the initial system software (e.g. compilers and operating system kernel) will be written and debugged using the macrosimulator;

(4) it is fast, so that software development on the architecture can begin early in the design cycle;

[2]Special purpose chips will sometimes require different types of input/output models, however, for the purposes of this paper, we will assume that the chip being designed is some type of processor that fetches instructions

(5) it generates approximate performance data;

(6) it is portable, since execution by different computer systems is desirable; and

(7) it executes from a standard "physical" memory load image that allows memory images to be interchangeable with those of the microsimulator and of the chip itself.

Of the above characteristics, the two most important are the performance estimation features and the execution of the standard physical memory load image for the target architecture. By being able to estimate performance execution from the macrosimulator, the macrosimulator becomes not only an important design tool for the microarchitecture level synthesis, but also an invaluable tool for the evaluation activity at the architecture level.

The macrosimulator generates performance numbers via two techniques. First, by using ***cycle tables*** that contain estimated operation times, totals can be obtained for program execution. And second, certain on-chip state (e.g. caching) can be simulated and those times incorporated into the total. It should be noted that for internal state simulation to be done easily, the appropriate mechanisms need to be added to the macrosimulator from the beginning. Furthermore, this state should be parameter driven as much as possible to allow efficient exploration of the design space. The numbers generated by the macrosimulator are only approximate. However, even though not all state is simulated, our experience has shown that macrosimulator estimated execution times can be within 10% of actual execution times.

Another major use of the macrosimulator is as a tool for validating the functionality of this and lower level simulators. Here the ability to execute a standard load image is imperative. It allows software development to begin early in the design process. Combined with a set of representative benchmarks, the macrosimulator also becomes a valuable design tool in assessing trade-offs during the microarchitecture synthesis process at the next level.

Architecture Validation

By executing programs on the macrosimulator, the architecture is validated early in the software development cycle, thus providing a check of whether the architecture functions as expected by the software development group from their understanding of the Architecture Specification. Architecture validation also occurs during the microarchitecture validation process when the two levels are validated against each other.

The Architecture Specification contains algorithmic descriptions of the target architecture's instructions. It is possible to automatically use those descriptions to generate test programs and results that can be used to validate the macrosimulator, though it is only practical for simple instructions.

Architecture Evaluation

Chip size and power dissipation are estimated by the chip designers. These estimates are, for the most part, based on the designers' experience. Sometimes it is known what large structures (e.g. register files, buffers, ROM, and caching) will exist on the chip and the sizes of their constituent cells, thus providing additional information. Performance estimation is also done with the macrosimulator.

and operates on data according to the fetched instructions.

When architecture synthesis is complete, the designers have a precise, executable description of the function to be implemented, as well as a knowledge of what operations are critical to the performance of the chip. Furthermore, the macrosimulator can be used to assess trade-offs in the microarchitecture design space as microarchitecture synthesis proceeds.

MICROARCHITECTURE

Microarchitecture synthesis can best be thought of as the process of converting the high level description of the architecture, into the hardware necessary to implement it, but at a Register Transfer Level (RTL) of description. As in architecture synthesis, it too is a difficult task requiring the expertise of engineers who have an in depth understanding of both computer architecture and hardware system design.

Microarchitecture Synthesis

Microarchitecture design begins with the Architecture Specification and results in a block level description of the hardware needed to implement the architecture and an executable model of that hardware called a microsimulator.

To translate an architecture into hardware, the engineers must first understand the actions that must be performed for every operation supported by the architecture, and then assign the hardware resources necessary to carry out those actions. For example, a Register_Add instruction requires hardware registers to contain the data, an ALU capable of executing the add, and a means of moving the data between the registers and the ALU. Once this resource list has been compiled, the engineers can turn their attention to defining the data flow through the chip and the control structures to support that flow. These may then be combined into an RTL implementation of the architecture that describes the hardware structures (e.g. registers, ALU's, busses, and state machines), the interconnect between the structures, and the timing of the data transfers through the chip.

The RTL description can take many forms during the synthesis process. Initially, the RTL is a block diagram and a *Microarchitecture Specification.* The RTL design is then iteratively refined until an executable algorithmic description, the microsimulator, is obtained. A microsimulator is a computer program that accurately simulates the internal operation of the chip. It is more detailed than the macrosimulator but shares many of its features. Ease of modification, a user interface with a broad range of commands, and the ability to execute load images are important features. Fast execution of the microsimulator, however, is the most important and is paramount to the success of the entire methodology.

Microarchitecture Description

Although the primary output of microarchitectural level design is a working RTL description (the microsimulator), there are several intermediate steps along the way. The first of these is the creation of the top level block diagram of the chip on an engineering workstation. (See Figure 2 for an example of a simple block diagram.) This block diagram is also the top level of a hierarchical description of the detailed logic of the chip and constitutes the first step in the decomposition and abstraction process. This block diagram partitions the functionality into several major blocks and defines the interface signals between the blocks. For the sake of clarity, block-to-block signals are grouped together instead of drawing each wire individually. Since the block diagram is resident on a workstation, it can be automatically converted into the basic format (e.g. global variable declarations, and procedure parameters for the basic block function calls) of the microsimulator.

In addition to the block diagram, each functional block and interconnect signal is also documented with a short description of its function and its expected timing. This information is then reviewed by the project team and agreed on before the design of the functional blocks begins. This entire process is analogous to the separation of external specification from internal implementation of software engineering.

To create a microsimulator for the chip, the designers write models for each of the functional blocks (FUB's) in a general-purpose programming language. Since no hardware design language is involved, any of the programming constructs available in the language may be used to create the model. The code for each FUB is not a detailed logic design (e.g. each register is implemented as a variable, rather than as multiple instances of a register bit cell), but control signals, timing, internal signalling protocols, PLA, RAM, and ROM logic are all modelled to the bit level. Timing resolution is to the sub-cycle or *phase* level; a two phase design (PH1 and PH2) is executed at least twice[3] per cycle - once during PH1 and once during PH2. An example of the code needed to implement a register is shown below:

```
loadit := IF (ph1=1) AND ((loadreg=1) OR (reset=1)) THEN '1 ELSE '0;
reg1   := IF loadit THEN databus ELSE reg1;
```

In this example, the register named "reg1" is loaded with the contents of the data bus "databus" during phase 1 whenever either of the control signals "loadreg" or "reset" are asserted.

The code for the various FUBs is then combined to create a simulation environment. The microsimulator environments currently in use at Intel, for example, consist of an event scheduler and a user interface that allows single stepping, breakpoints, and the ability to display, trace, or modify any signal or variable in the model. The scheduler executes the model at fixed time increments, or can optionally execute a model until it "relaxes." In "relaxation" mode, the scheduler executes the model until all of the interface signals have stabilized, thus minimizing any order dependencies that result from simulating the parallel operation of hardware on a serial machine.

One technique that is being used to simplify the logic synthesis process at the next lower level is to restrict the model writer to a limited set of well-defined functions or "structures" at the microsimulator level. Instead of complex conditional expressions to simulate control signals, a set of standardized structures (e.g. logic gates and register cells) are used. Logic design then is simply a matter of examining the model code and drawing the equivalent structure on a schematic. An example of this approach is given below. The model is equivalent to the previous example. Although the structured example presented here appears harder to follow than the conditional case, it is more straightforward to translate since the AND2, OR2 and LATCH structures have direct MOS equivalents.

[3] It is often necessary to execute two passes of each phase by using "start," (e.g. PH1ST) and "end" (e.g. PH1END) signals. The use of two passes can solve most code ordering problems. The need for more than two passes per phase can generally be traced to serious critical paths where the designer is attempting to accomplish too much during a single phase.

```
loadit:=AND2(PH1, OR2(loadreg,reset));
LATCH(reg1,loadit,databus);
```

Most general purpose, structured programming languages support some type of macro processing and, as a result, can directly support this structured approach. A macro is written to simulate each of the structures, and then instances of the macros in the microsimulator code are expanded by the compiler at compile time to form the executable model.

The two styles presented above show two fundamentally different approaches to writing a microsimulator. The first provides less detail of the logic design at the microarchitecture level, and for that reason can be produced and debugged more easily. Also, it has higher performance than the second approach. Simulator validation technology requires the execution of large numbers of test suites. The faster the simulator the more effective the validation. The second approach, on the other hand, greatly reduces the "distance" between microsimulator description and logic description, thus reducing logic design time and decreasing the risk on the logic design. However, the price is a slower simulator — a more detailed simulator is roughly an order of magnitude slower. Both approaches are a response to two conflicting goals of the microsimulator: simulation speed and design abstraction. The resolution of this problem may come from the availability of high-speed logic simulation engines, since these provide the necessary performance for simulation at the logic level, thus making the performance goals of the microsimulator less important. Also, we expect that improved microsimulator source language and compilation techniques will eliminate some of the performance disadvantages of the macro-based approach.

It is at the microsimulator level that the hierarchy of the design becomes evident. The models of the major functional blocks are connected by the framework produced from the top level block diagram. Each of these major functional blocks, in turn, has several sub-blocks, and these are connected by the frameworks produced from the second level block diagrams, and so on. The result is a hierarchical simulator that closely matches the microarchitecture as well as the physical implementation.

Microarchitecture Validation

The microsimulator is a key tool in the Large Chip Methodology. It not only serves as the first implementation of the hardware, but also reflects the current state of the design at any time. This is done by making and testing all changes to the design in the microsimulator before updating the schematics and the layout. Because of its closeness to the microarchitecture, and its relatively fast execution speed (compared to full logic simulation), the microsimulator is an ideal tool for validating the hardware implementation against the architecture specification.

Microarchitecture validation consists of creating a rigorous test suite that checks that the implementation properly executes all the operations specified by the architecture. Fortunately, many of the test suites generated to validate the macrosimulator may also be executed here. On the other hand, validation of the microsimulator is more difficult than validation of the macrosimulator because the microsimulator executes about 1000 times slower. As a result, there are significantly fewer simulated cycles available.

There are no suitable formal techniques or algorithms that guarantee complete functional equivalence between the macro and microsimulators. However, by being able to arbitrarily execute identical load images, the results of executing particular programs on each simulator can be accurately and automatically checked by comparing memory images. By the time the macrosimulator is complete, a large set of small and some large programs are usually available for execution by the architecture. These programs are then executed on the microsimulator and their results compared to identical macrosimulator runs. By having a full time group of engineers devoted to "architecture validation," this process can be quite effective. In any event, it is impossible to over-validate the simulator, and validation should continue through the entire hardware implementation phase.

Microarchitecture Evaluation

Since the microsimulator is the first detailed description of the chip, it is also used to obtain information about the physical realization of the design. As mentioned earlier, the microsimulator accurately describes both the block-to-block interconnect and the major data and control structures on the chip. This information can be used to complete a preliminary chip plan and derive first pass power and area estimates. In addition, the timing resolution provided by the microsimulator produces accurate information that verifies the assumptions made in the Architecture Specification.

The microsimulator serves as both a working implementation of the chip that is available months before the actual silicon, and as a functional reference to which the various design descriptions (including the actual silicon) are compared. Because it simulates the individual control signals and data paths and is accurate to the phase level, it can produce the stimulus for the logic simulator and component test bed, and can even be used to produce the test vectors for a VLSI tester. For chips that have microcoded macroinstructions, the microsimulator also becomes a microcode debug and validation vehicle.

LOGIC

Logic design involves, for all the functional blocks in the microarchitecture, the conversion of the microsimulator models into gate level schematics.

Logic Design Synthesis

Ideally, the design process should proceed serially from RTL simulator to logic schematics, but this is generally not the case. The level of detail required to create an accurate microsimulator forces the engineer to consider logic implementation details as the RTL model is being written. Thought must be given, for example, to the logical implementation of a register so that the proper control signals can be produced in the microsimulator. Likewise, logic that can lead to potential critical circuit paths must be avoided. As a result, an engineer writing a microsimulator model usually works with both preliminary logic designs and the code used to simulate it in the microsimulator.

Even with a highly structured microsimulator, detailed logic synthesis is still required, though the design space for this synthesis is more constrained. Several examples of this synthesis are:

(1) Registers, busses, etc. are represented by a single variable in the microsimulator. These must be expanded into multiple instances of the appropriate cells.

(2) Control signals, as a rule, are implemented as positive-asserted signals using AND-OR structures in the microsimulator. These must be converted to their equivalent NAND-NOR implementations.

(3) Phase traps are not always explicitly represented in the model. They must be inserted into the schematics where necessary.

(4) Large structures such as PLA's and ROM's are modelled at a functional level. The appropriate buffers and drivers must be added to the schematics.

Once the models have been expanded, the design team reviews the results and looks for common structures such as decoders, register cells, counters, etc. These common functions are combined where possible and a set of "standard" gates and cells are selected and the logic design completed for them. Sometimes minor changes to the design can eliminate duplicate effort, resulting in a significant reduction in the total design effort. The logic schematics are then entered on an engineering workstation. The hierarchy established by the block diagrams and microsimulator is preserved and refined in the schematics, resulting in a single logic schematic tree for the chip.

Logic Design Description

The description for the logic design level is a set of hierarchical, gate level schematics. In MOS the relationship between logic and circuit design requires that the logic data base be represented as a circuit data base. This is possible with a set of design tools that can hierarchically abstract many commonly occurring circuits such as logic gates. This means that the circuit design is described by the same data base that is used by the logic design. Although default device sizes may appear on the schematics, the final device sizes are added during the circuit design phase of the process, so the logic designers do not concern themselves with accurate sizing at this stage of the design. (Hence, these schematics are often referred to as "unsized"). The schematic data base is actually a network description of the chip that can be formatted and transferred from the workstation to other computers for logic and circuit simulation.

Logic Design Validation

An important validation task is the checking of the logic schematics to the microsimulator. By automatically comparing the layout to the schematics and the schematics to the validated microsimulator, the probability of producing highly functional chips on the first stepping is significantly increased.

Logic validation is done by simulating the logic design with a switch-level MOS logic simulator [7]. This type of simulator does a good job of modelling the bi-directional operation of MOS transistors and yields more accurate results than more traditional logic simulators.

The schematics are first grouped into *Logic Simulation Units* (LSU's). Each LSU consists of a group of related schematics that operate more or less independently, have a well defined interface (usually an LSU is a sub-block in the hierarchy) and are easily accessible from the external pins. A hand written test suite[4] is then generated for each LSU and debugged on the microsimulator. The test suites are written so that as many devices in the LSU as possible are toggled during the test run, and that as many faults as possible in the LSU are sensitized and made observable at the external pins. This requirement means that the tests developed for logic validation can form the basis of the component characterization and sort tests. Minimal acceptable coverage for any test is that every node in the

[4] In a processor the tests are either macrocode or microcode that can be brought on chip during execution. For other more special purpose chips, the test suite can be a variety of stimulus response vectors.

circuit be toggled, but the tests are invariably expanded until the degree of coverage is higher. Intel is developing a set of tools that help assess fault coverage for a variety of MOS faults.

Once the test suite is debugged, the microsimulator is used to generate both the stimulus to the LSU and the expected results. This is possible because the microsimulator explicitly models all signals that cross the boundaries separating the schematics. These stimuli and expected result data are then reformatted and the test is run on the logic simulator. The logic simulator outputs and microsimulator outputs for the LSU being tested are then compared to check that they match exactly. The network to be simulated is generated from the schematic data base stored in the workstation.

Logic Design Evaluation

The set of logic schematics for a chip are the first detailed representation of the individual devices of the design. The devices and their interconnections are final, however, some adjustment of device sizes is still required. While the microsimulator can be used for rough chip planning activities, the logic schematics are necessary for detailed chip planning to proceed and for the first accurate drawn/** device estimates to be obtained.

Once the schematics for a major functional block are completed, a detailed block plan is derived. This block plan shows all the interconnect and preliminary positioning of the devices. If the technology permits and area constraints are severe, a coarse-grain block plan can be produced with an auto-place/route tool using the schematic data base as input, otherwise manual techniques must be used. When block plans have been completed for all the major functional blocks they are assembled into a detailed chip plan that is also used to derive rough speed estimates on manually identified critical paths.

CIRCUITS

Circuit design is the process of altering device sizes of the unsized logic schematics to insure the chip operates according to its electrical specifications. Although most of the major circuit synthesis is done during the logic design phase when creating the pre-designed cells, there is still much work to be done in checking critical paths and adjusting device sizes. The result is a set of schematics that are correctly "sized" for the frequency and electrical requirements of the chip.

Circuit Synthesis

Circuit synthesis begins with the development of a standard set of cell designs for use during logic synthesis. This set consists of a large variety of standard logic gates (e.g. NAND, NOR, XOR) of various inputs and sizes, as well as special cells such as RAM, PLA (one and two phase), and various types of bus drivers and receivers. The design of these cells occurs early in the design cycle of the chip and helps establish coordination between the chip designers and the technology group responsible for the process with which the chip will be fabricated. By doing preliminary design, the circuit designers experience the various strengths and limitations of the process itself and can often influence the development of

/** These are cells actually hand drawn by the mask designers.

process design rules.

The interaction with the process development group continues throughout the design cycle. This is important since a state-of-the-art chip will generally be using a new process for which there is limited manufacturing experience and the continual risk of small changes in design rules. The effect of these rule changes must be appraised and incorporated into the chip design.

Some of the design of the chip is done with standard cells. However, in any high-performance device there is always special logic (e.g. arithmetic function units) that requires custom designed circuitry. This constitutes the next phase of circuit design, and occurs concurrently with the logic design effort.

The last phase of the circuit synthesis process involves the fine-tuning of the entire design to guarantee that the slowest path on the chip can be executed in the target clock period of the chip under adverse temperature and voltage conditions. This phase is discussed in the circuit validation/evaluation section.

Circuit Description

The logic and circuit descriptions are described by a single circuit level data base. This means that it is not necessary to do a separate functional level validation of the circuit design, since this is done as a part of the logic validation process. Exceptions to this are analog circuits (e.g. a sense amplifier) which cannot be adequately checked during logic validation.

Circuit Validation/Evaluation

Evaluation of the circuit design level has three major goals: first, to guarantee that the delays in all circuit paths on the chip are within the specified clock time plus engineering tolerance to attain sufficient yield; second, to guarantee that there are no fatal circuit design problems; and third, to guarantee that the power dissipation of the chip is within the specified norms.

When checking circuit speed, the primary circuit evaluation tool is the circuit simulator. SPICE-like programs can be used to do circuit simulations of small portions of the chip. Special purpose circuitry (e.g. a high-performance carry-save adder) is always circuit-simulated to guarantee correct function and device sizing. This is easy to do for highly visible, special purpose logic, but is impractical for the entire chip. Some technique must be available for identifying non-obvious critical paths as candidates for further simulation. Consequently, tools are being developed that evaluate all paths in a design unit and then estimate the path delays based on device sizing, metal path lengths, and capacitive loading [3].

Normally, circuit path delay tools need only be applied to those paths that cross one or more major boundaries, since the shorter paths (those within a single block) are generally known by the unit designer. Once the critical paths have been identified these paths can often be fixed via logic changes, but eventually the designer must resort to circuit simulation for the most stubborn paths.

The second activity involves the identification of illegal circuit combinations, for example, a dynamic gate which is fed by a signal that is also dynamic in the same phase. Many of these problems will be discovered during logic validation and circuit design walk-throughs; however, an automated means (e.g. a circuit design rule checker) is needed to identify illegal combinations. Circuit debug tools such as this are now being developed at Intel.

The last task is that of power dissipation estimation. When most of the circuits for the large arrays (ROM, RAM, and PLAs), and bus and pin drivers have been designed, the designers can make a rough estimate of the power dissipation for the chip.

MASK DESIGN

Mask design involves the translation of the sized schematic set into the graphics data base used to create the masks that are used during fabrication of the chip. For most general purpose, high volume VLSI, this job is done by a group of expert mask designers.

Mask Design Synthesis

Mask design activity does not begin with the sized schematic set, but is an ongoing process starting early in the design cycle. Once a set of design rules is released for the process being used to build the chip, preliminary work can be done on general purpose structures such as RAM and register cells. These initial layouts are used both to gain experience with the process and to serve as a library of structures used to build the chip. The layouts are also used to derive data on parasitic capacitances so that the operation of critical circuit paths can be accurately characterized by the circuit simulator.

As the sized schematic set becomes available, the emphasis shifts from cell layout to detailed chip planning and device entry. First, detailed layout plans for all the major functional blocks on the chip are combined into a composite plan of the entire chip. This plan serves as a reference during the mask design process. The sized schematics are then translated into their device equivalents in keeping with the design rules for the process. Techniques such as auto-place/route and standard cells can be used occasionally to speed up this effort, though their applicability is highly dependent on the required optimization (of circuit speed and/or silicon area) of the target area. Once all the devices are entered, the data base for the entire chip is assembled and automatically checked for both design rule violations and connectivity errors across major block boundaries, and the graphical data for the various layers is converted to the format used by the mask vendor to produce the masks.

Mask Design Description

The mask design description is a geometrical graphics data base for each of the layers of the chip. This description is entered and manipulated on a graphics workstation with computation-intensive activities such as design rule checking and connectivity verification taking place in batch mode on mainframe computers networked to these workstations. Data entry is via a digitizing pad and keyboard with visual output being directed to a high resolution color monitor. Large plotters are used to create a permanent record of the layout for checking purposes.

Mask Design Validation

Mask design validation consists of design rule checking and connectivity verification. Both of these are handled by sophisticated computer programs that can extract behavioral descriptions from the physical descriptions of the layout. Design rule checking (DRC) is done by breaking the mask design into the polygons that comprise the devices and comparing these polygons (and their interactions) with the design rules for the process. For example, a design rule may specify the minimum spacing between two metal lines. The DRC program first breaks the layout into polygons and then checks all instances of adjacent metal for compliance to this rule. The polygon data base can also be used for the next validation step, connectivity verification.

Since it is not cost-effective to evaluate all aspects of the design at every level, the particular validation/verification effort at each level is oriented towards discovering the most common and most serious errors that occur at that level. At the mask design level, the most common error is connectivity mismatch; consequently, at Intel a complete set of automatic connectivity verification tools have been developed [8]. In connectivity verification, the polygons are assembled into MOS devices and a connectivity graph of the entire area being verified is generated. This new data base is checked for shorts and opens, and is eventually compared for equality to a similar connectivity graph generated from the sized schematic data base (logic/circuits description). This is the next link in the validation process, checking whether the layout matches the sized schematics.

Mask Design Evaluation

Once the layout has been completed, the precise characteristics of the chip (such as die size and parasitic capacitances) are known. This information can be used to verify the assumptions made during logic and circuit design. Several new tools are recently available that automatically extract resistance and capacitance information from the mask design data base. A tool such as this closes one of the major gaps in the Large Chip Methodology, which is the identification of candidate circuit simulation paths. As these tools become operational, it should be possible on the first stepping to produce functional die that also meet the electrical specifications and performance goals.

RESULTS

A Large Chip Methodology, similar to that described here, has been in use at Intel since 1980. Since then, most of the VLSI processor and peripheral chips following the methodology have shown high functionality in early steppings. For example, the 43204 (Bus Interface Unit for the iAPX-432 system with roughly 62,000 devices) and the 43205 (Memory Controller Unit with roughly 82,000 devices) were logically functional on their first iteration and 43205 samples were actually shipped from this first silicon.

Although both of these chips were functional, they contained circuit design errors that limited their usefulness. In the 43204, one of the internal phase clocks was driven to a portion of the chip by a single diffusion. The chip passed the connectivity verification because diffusion provided the connection even though the intended metal line was missing. The problem was corrected with a single mask change and the second iteration was sampled.

The 43205 had an unintentional feedback path that prevented one of its queues from operating properly at conditions other than 5 volts and 5.0 MHz. This problem was not detected by the logic simulator as it was caused by the bi-directional nature of MOS transistors, and the logic simulator in use at the time modelled MOS transistors as unidirectional switches. Intel has since adopted a switch-level logic simulator to avoid similar problems. That and several other circuit problems were fixed and the second iteration was fully functional over the entire operating range.

EVOLUTION

The methodology described here is adequate for today's technology, however, continued evolution is necessary to track technological change. There are many factors that influence design methodology evolution, these include:

(1) Technology push. It is the consensus of most researchers in silicon technology that commercial CMOS can be reliably manufactured down to between 0.25 and 0.1 micron. This level should be reached by the end of the century and will lead to chips with roughly one billion devices [4].

(2) Powerful, high-performance logic and circuit simulation hardware. These simulators will be constructed from special purpose hardware or from general-purpose, highly parallel machines such as Intel's iPSC (hypercube multiprocessor).

(3) More intelligent CAD technology. CAD software will continue to evolve as work continues in the areas of standard cell and procedural layout systems as well as silicon compilation.

(4) Powerful, single user workstations based on next generation 32-bit microprocessor technology. Individual design stations are now available that are more powerful than the midrange computers previously shared by entire groups of designers.

Though it is difficult to assess the exact effect that these and other factors will have on methodology evolution, it is our feeling that the following changes are possible.

(1) A major result will be highly integrated CAD environments, with unified data structures and user interfaces.

(2) Design environments will be two-tiered, with the individual workstations providing the most support for the designers, and large, powerful, batch-oriented central systems providing circuit and logic simulation support as well as global connectivity and design rule verification. Central computational and file services will be shared by many workstations [5,6].

(3) Though complete silicon compilation is not yet an option for the general-purpose, high-volume chips discussed here, intelligent CAD tools will automate ever larger portions of the design process. Likewise, as we pass the million devices per chip barrier, designs will, out of architectural necessity become even more regular, which in turn will lend itself to automated lay-out.

Logic validation will remain a problem in the near term, since it is our belief that formal verification techniques will not exist in the next five to ten years that can efficiently verify a chip with one million devices. However, we believe that the designers can keep abreast of technological expansion by:

(1) Improved test generation methodology. The application of software testing methodologies and interactive automatic test generation programs will improve our ability to create effective test suites.

(2) High-performance logic simulation. Better test suites coupled with high performance hardware based logic simulation of a million or more devices [1] will improve confidence in the logic validation of very large chips.

(3) Logic oriented microsimulation. Though it is unclear what the performance and capacity trends of logic simulation engines will be over the next ten years, it is likely that most of the validation activities will fall to these systems, thus freeing the microsimulator to become a design aid. Currently, our microsimulators represent a high level of abstraction because of the performance requirements of RTL validation. However, if most of the validation load can be assumed by high performance RTL/logic simulation engines, then microsimulator performance is less important and the design abstraction capabilities of the simulator may become more desirable.

(4) More modular design. Continued emphasis on decomposition and modular design will allow designers to control design complexity in the next generation of devices. Modular design will also be supported by a variety of CAD tools created to ease this modularization process.

One advantage to a formal methodology is that it provides a framework for evolution so that it can adapt to handle the next generation of chip development. The Large Chip Methodology provides just such a framework for CAD tool evolution. Figure 3 shows a two dimensional matrix, where each row corresponds to a design level and each column to an activity, with a CAD tool or set of tools for each entry. Some entries have tools that totally automate that activity for a given level (e.g. mask level connectivity verification), and some entries have no real tools (e.g. architecture synthesis). This matrix is invaluable in identifying weaknesses in a methodology and the needs for next generation CAD tools.

SUMMARY

The Large Chip Methodology is based on the notion that VLSI design is an exercise in complexity management, and hence, such techniques as hierarchical decomposition, abstraction, and step-wise refinement must be used. In addition, it is a general unifying force for the entire design process and serves as a framework within which CAD tools are developed. The existing methodology at Intel, on which our proposed methodology is based has achieved impressive results and is an invaluable part of most chip design and planning processes. As VLSI complexity increases, the methodology will continue to adapt and conform to the available technologies in work stations, tools, fabrication processes, and automated synthesis techniques.

REFERENCES

[1] T. Blank, "A Survey of Hardware Accelerators used in Computer-Aided Design," *VLSI Design*, August 1984, pp. 21-39.

[2] W. W. Lattin, J. A. Bayliss, D. L. Budde, J. R. Rattner and W. S. Richardson, "A Methodology for VLSI Chip Design," *Lambda*, Second Quarter 1981, pp. 34-44.

[3] J. Mar and Y. P. Wei, "Performance Verification of Circuits," *Proc. 21st Design Automation Conference,* June 1984, pp. 479-483.

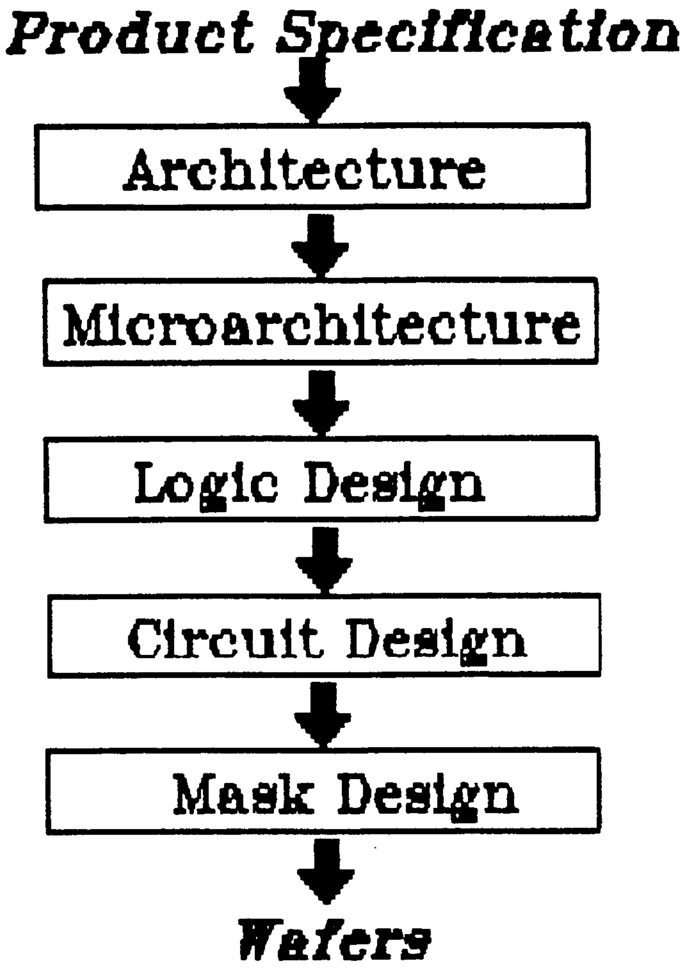
[4] J. D. Meindl, "Limits on ULSI," *Proc. VLSI 85,* Tokyo, JAPAN, August 1985.

[5] S. Nachtsheim, "The Intel Design Automation System," *Proc. 21st Design Automation Conference,* June 1984, pp. 459-465.

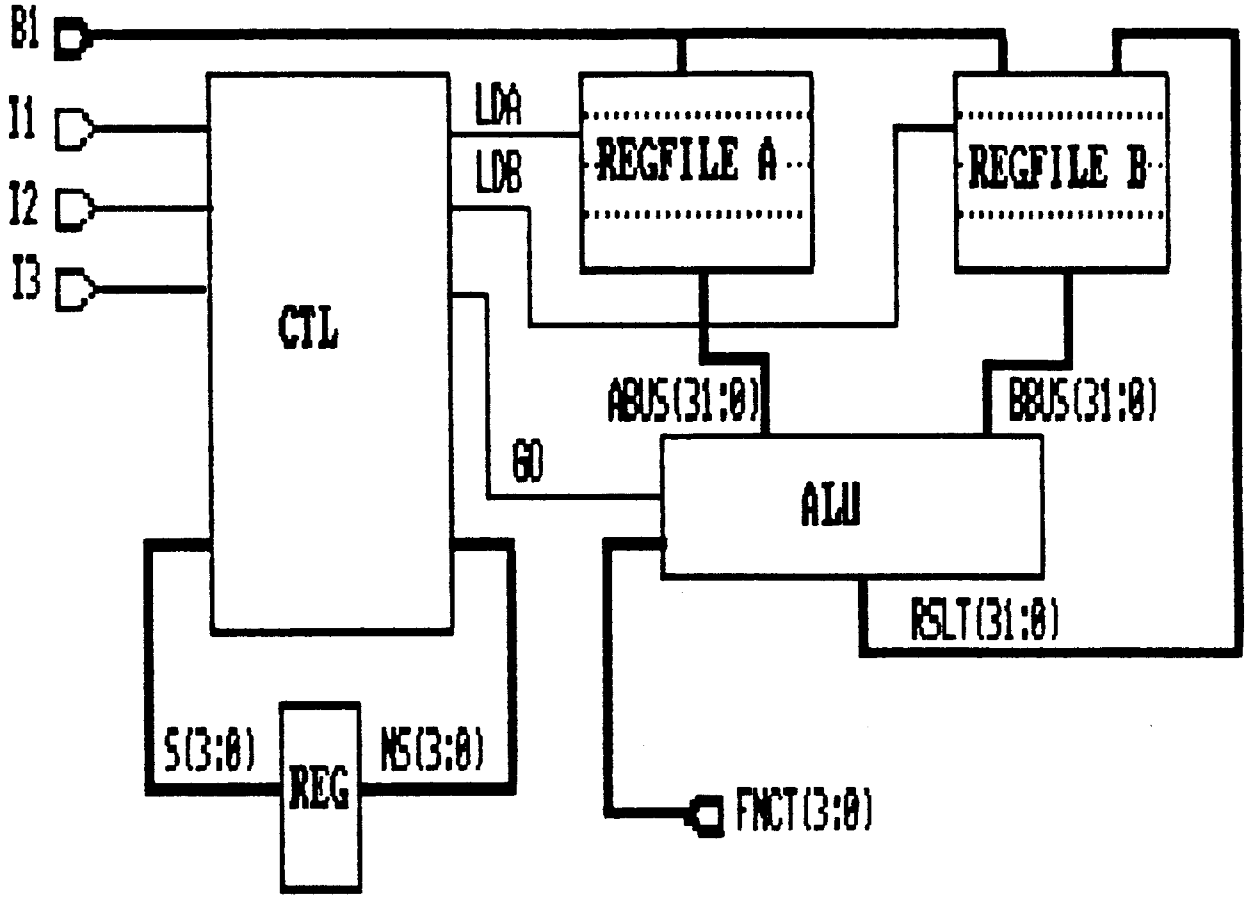
[6] K. Sherhart, M. Vershel and J. Owen, "The Engineering Design Environment," *Proc. 21st Design Automation Conference,* June 1984, pp. 466-472.

[7] K. Tham, R. Willoner and D. Wimp, "Functional Design Verification by Multi-Level Simulation," *Proc. 21st Design Automation Conference,* June 1984, pp. 473-478.

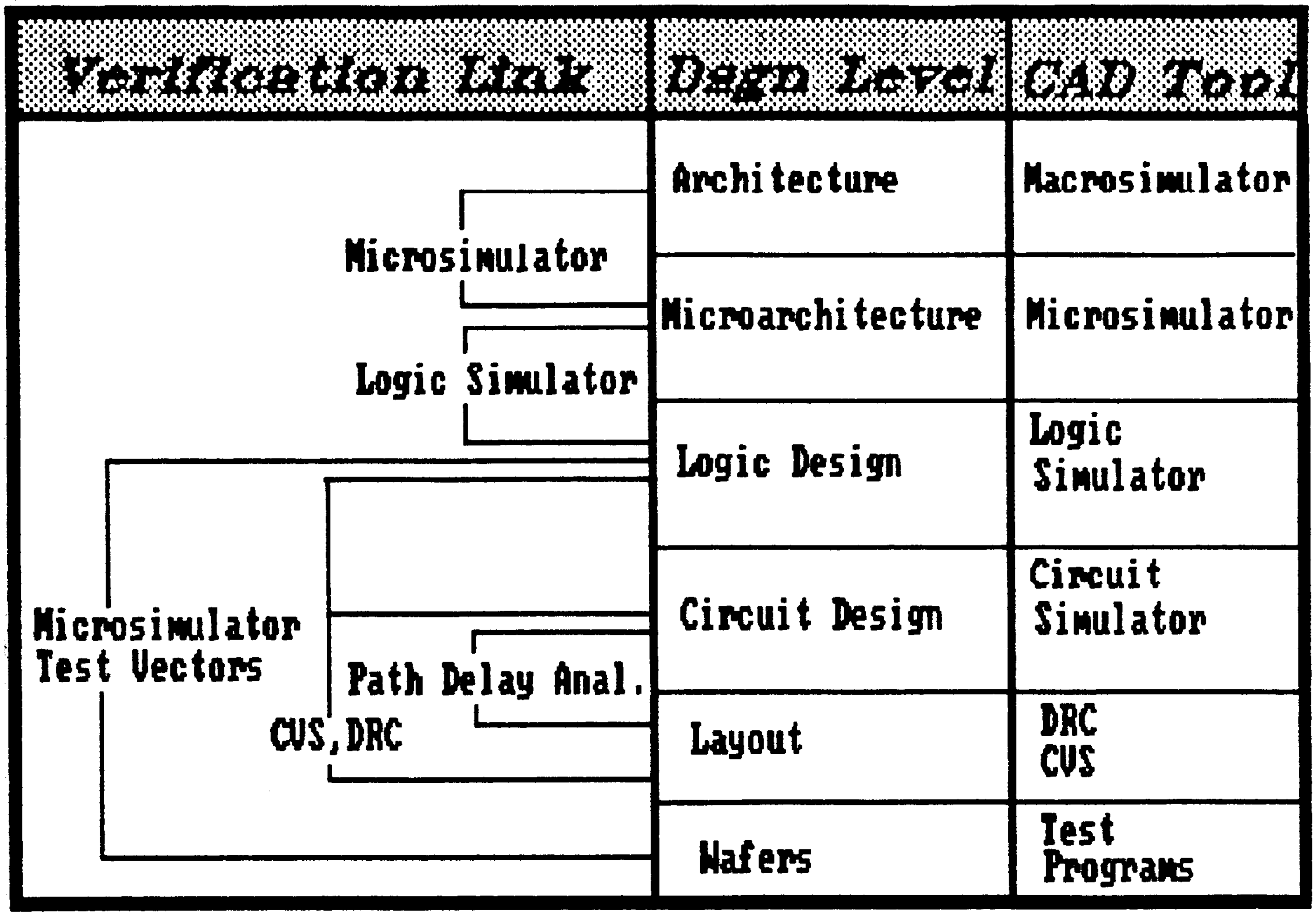
[8] T. Wagner, "Hierarchical Layout Verification," *Proc. 21st Design Automation Conference,* June 1984, pp. 484-489.


Product Specification
Architecture
Microarchitecture
Logic Design
Circuit Design
Mask Design
Wafers

(Fig. 1)


B1
I1
I2
I3
CTL
LDA
LDB
REGFILE A
REGFILE B
ABUS(31:0)
BBUS(31:0)
GO
ALU
RSLT(31:0)
S(3:0)
REG
NS(3:0)
FNCT(3:0)

(Fig. 2)


Verification Link
Dsgn Level
CAD Tool
Architecture
Macrosimulator
Microsimulator
Microarchitecture
Microsimulator
Logic Simulator
Logic Design
Logic Simulator
Microsimulator Test Vectors
Circuit Design
Circuit Simulator
Path Delay Anal.
CVS,DRC
Layout
DRC CVS
Wafers
Test Programs

(Fig. 3)