Abstract
The increasing complexity of VLSI design and the demand for quick-turnaround ASICs has forced the designer to choose the best CAD tools available from different vendors and integrate them into a customized and comprehensive CAD system. Vendors have developed comprehensive but open systems, called CAD frameworks, to ease the process of CAD tool integration.

This paper describes the ASIC development process and issues relating to CAD tool integration. In addition, state-of-the-art CAD frameworks and their impact on ASIC designers are described.

I. Introduction
Application specific integrated circuits (ASIC) technology has brought a great revolution in VLSI system design. Before the advent of ASICs, systems were designed using a set of standard IC chips, whereas whole systems can now be designed into a single chip. ASICs are usually low volume products. Shorter design cycles are required to make the product cost competitive. Short design cycles imply increased dependence on CAD tools. To meet the demand of quick-turnaround ASICs, CAD tools have been developed to automate almost all the phases of ASIC design.

However, with increasing design complexities and design data, the flow of the design data through the CAD tools at various phases of the design cycle has become a cumbersome task for the ASIC designer. To relieve the designer of this burden, integrated CAD systems were developed. Initially, these systems were only a collection of tightly integrated tools. Adding and deleting tools was difficult, and usually these CAD systems were incompatible with each other. Such a closed environment is slow to evolve and the designer is restricted to a limited set of tools. Thus the concept of open design environments called CAD frameworks evolved. This concept is receiving wide attention in the area of computer aided design. A group of tool users, vendors, and system integrators, the CAD framework initiative (CFI), has set out to establish guidelines for a standard CAD framework. Thus we can look forward to a framework that encompasses all areas related to VLSI design such as specification capture, logic design, verification and testing.

II. Application Specific ICs
Definition
ASICs were ICs designed for a specific application. In contrast to standard ICs that are used in different applications. ASICs are a class of ICs that are in between the domain of software programmable generic components, for example microprocessors, and hardware programmable components, for

example PLDs [KEUT,’89]. They can also be called custom-built ICs.

Characteristics
State-of-the-art electronics products such as digital signal processors, voice synthesizers, automatic focusing systems in cameras, etc. use ASICs. For prototypes, the circuits are designed by using a set of standard IC products but later for commercial products they are redesigned as an ASIC chip. System development cost is reduced by incorporating different components of the system, for example system controllers, RAMs, ROMs, PLDs, etc., into an ASIC. This technology also reduces the size of printed circuit boards thus reducing production cost. However, once systems are integrated into a chip, correcting even a small error, requires re-designing and re-fabricating the entire chip. Also, reduced node accessibility creates testing problems. Testing currently occupies 30-50 percent of the production cost of ASIC chips [LEUN,’88].

The ASIC designer uses a lot of CAD tools to reduce ASIC design time but has to sacrifice flexibility with regards to minimizing area and maximizing speed. A survey of ASICs designed for use by AT&T summarizes some common characteristics of ASICs [KEUT,’89].

• Control dominated - Design of the control circuitry in the ASIC takes up a majority of circuit area and design time.

• Arithmetic Structures - Few large arithmetic circuits are present in an ASIC.

• Speediliarea - Even though high speed and component density are achievable, most ASICs use under 10,000 logic (non-memory) transistors and operate at no more than 10 MHz.

• Regularity of structure - ASIC designers use as many regular structures for eg. library cells, gate arrays, etc. as possible.

• Analog interfaces - Many asynchronous and analog interfaces are used.

III. ASIC Design Styles and Methodology

1.Gate Arrays - In this style gates are pre-arranged, with space for channel routing. These gates are then interconnected by customizing metal layers. All levels of masks, except the metal interconnections, are predefined so that the wafer can largely be pre-fabricated. This prefabrication of wafers, called masterslides, is the main reason for fast turn-around time of prototype gate arrays. Due to its limited design freedom, the chip size of the gate array is typically two or three times that of a handcrafted design [LEUN,’88].

2.Sea of gates - This style is similar to gate arrays but has no
*Predefined routing channels. So, sea-of-gates is also called "channel less gate array" and is more effective in area reduction than gate arrays.*

**Standard cell** - Standard cells, for example, basic cells like inverters, NAND gates, Flip-Flops, etc., are pre-designed and are a part of a cell library. Placement and routing can both be customized. Usually all cells are of fixed height and a channel router is used to accomplish the interconnections. Pre-fabricated wafers are not used in this style.

**PLDs** - Programmable logic devices include PLAs, which are generated by automatic PLA generators and are usually used to implement system controllers.

**Custom cells** - Custom cells are designed to implement special functions that are not available in the library. If these custom cells are designed according to standard cell height specifications, then they can be added to the cell library and used with standard cells.

**Silicon compilers** - Once silicon compilers come of age the designer need only provide behavioral descriptions of the circuit and the corresponding hardware will be synthesized on silicon automatically. Presently, silicon compilers are used for generating megacells [LEUN98].

Usually, an ASIC is a mixture of modules designed using different design styles. For example, IBM has developed a design system that allows complete mixing of standard cell and gate array functions [LEUN98]. Thus we are moving towards an era where changing from one design style to another will be easy. This implies that designers will no longer be required to make tradeoffs between the various design styles. An integrated design environment will provide the designer with a complete solution to intermixing various styles.

**Design methodology**

Various aspects of VLSI design can be partitioned hierarchically into levels of abstraction. Design tools are used to implement the design at these levels. Table 1 describes an example of hierarchical levels, levels of abstractions, and CAD tools required to implement the design at these levels. A methodology is a sequence of design steps that links the design process from specification capture to mask layout. There are basically two types of hierarchical design methodologies: top-down (see Figure 1) and bottom-up. In top-down design methodology, a design is first described in general terms at some high level of design abstraction. Designers then recursively decompose and elaborate the design. In the bottom-up method the most detailed parts are designed first and then global layout is determined by combining these parts. Today, most designers employ the top-down design approach.

**IV. ASIC Development Process**

The ASIC development process is divided into four major areas. The design process being an important part of the overall ASIC development process, is presented here in greater detail.

**Specification Capture** - During specification capture, the objective or goal of designing the chip is decided. The performance characteristics and functionality of the chip are set forth. All constraints regarding the environment (i.e. all external factors affecting the chip in any possible way) are precisely defined [CAVI90].

**Design Synthesis and Verification** - This stage of ASIC development encompasses all the design phases from system design to the final mask layout. It can be viewed as a process of successive transformations from one hierarchical level to another.

Different phases of design synthesis and verification for a top-down design approach are listed as follows:

- System is designed according to the specifications.
- Decisions on fabrication process, for example the technology to be used, are made.
- Functional partitioning of the system into modules is done.
- Specifications for each module are set at some level of abstraction as shown in Table 1.
- Through a series of mapping and translation steps the design is taken down through the hierarchical levels, as shown in Table 1, to the logic design level.
- Logic is verified.
- The design is mapped down to the layout design level.
- Inverse mapping (mapping from a lower level to a higher level) is done and the layout is verified for consistency with the logic design.
- Logic and timing verification is performed at the layout level.
- Mask layout for the chip is obtained in a standard format, for example CIF, and sent to the fabrication house.

**Table 1. Hierarchical and abstraction levels.**

<table>
<thead>
<tr>
<th>Hierarchy Levels</th>
<th>Abstraction Levels</th>
<th>CAD Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>Timing behavior, pin assignments</td>
<td>Flowcharts, Block diagrams, high level languages</td>
</tr>
<tr>
<td>Architecture</td>
<td>Organization of functional blocks</td>
<td>HDLs, Floorplanning, Block diagrams</td>
</tr>
<tr>
<td>Register transfer</td>
<td>Developing specifications for functional modules</td>
<td>Synthesis, Simulation, Verification, Test analysis</td>
</tr>
<tr>
<td>Logic</td>
<td>Boolean functions, Gate level circuits</td>
<td>Schematic entry, Simulation, Verification,</td>
</tr>
<tr>
<td>Transistor</td>
<td>Electrical properties of transistor circuits</td>
<td>SPICE, Timing verification</td>
</tr>
<tr>
<td>Layout</td>
<td>Geometric constraints</td>
<td>Layout editor, DRC and EIRIC programs, Netlist extractor, Photomask and Routing tools</td>
</tr>
</tbody>
</table>
Figure 2 depicts a typical ASIC design process. It can be seen from the figure that backtracking and iteration are required throughout the design process.

**Fabrication** - VLSI chips are fabricated by a complex series of about 100 or more steps. These steps create transistor parts, circuit elements, insulating layers, and metallized paths on silicon. Some major steps are thermal oxidation, lithography, etching, ion implantation, thermal redistribution, insulation, and metallization [STRO,'88].

**Testing and Verification** - Design verification techniques are used to qualify the ASIC to be marketed/supplied. At this stage the fabricated chip is tested for the specified external environment under accelerated stress conditions. Other tests to detect manufacturing defects are also carried out. Undetected errors in an ASIC can be very costly due to additional manufacturing costs and delay to market.

**V. CAD Tools for ASIC Design**

It is apparent from Table 1 and Figure 2 that a number of CAD tools are required to accomplish ASIC design tasks. These tools are classified as follows. (Examples for each class are taken from the UC Berkeley VLSI tool suite [SCOT,'85].)

**Mapping Tools** - These are used for schematic capture, interactive layout, graphical entry, text entry, placement and routing. For example, MAGIC layout editor is used for interactive layout, PEG is a text entry tool used for obtaining Boolean equations corresponding to a text that describes state diagrams.

**Translation Tools** - These are used for transforming files from one format to another to ensure compatibility between tools. For example, EXT2SIM is a tool for transforming the output of MAGIC layout editor to a format that is readable by simulator tools like ESM and CRYSTAL.

**Validation Tools** - These tools check for violations of design rules. For example, the built-in Design Rule Checker (DRC) in the MAGIC layout editor concurrently checks for design rule violations while the designer is laying out the design. There are CAD tools that perform Electrical Rule Checking (ERC) and check for connectivity of all the elements of the design.

**Verification Tools** - Logic and timing simulation on designs are performed by these tools. For example, ESIM is used for logic verification, CRYSTAL and SPICE are used for Timing verification.

**Optimization Tools** - Optimization tools are used to optimize area and speed. Area can be optimized by either layout compaction or reduction in the logic required to implement the design. For example, PLEASURE is used for PLA folding, and ESPRESSO is used for minimizing Boolean equations. Speed optimization depends largely on the designers expertise.

Along with the CAD tools the ASIC designer requires cell libraries. Cell libraries are a collection of primitive components, which can be individual transistors, logic gates or entire subsystems. These libraries may be traditional standard cells, gate array building blocks, parameterized cells such as those synthesized by silicon compilers, or sophisticated cells generated by module generators [NEWT,'86].

**Invoking and Executing CAD tools**

Some basic terms required for this discussion are defined [FIDU,'90] as:
- **Process** is a specific combination of tools and/or other processes that perform a design function.
- **Task** is an abstraction of a design function, e.g., simulation. Tasks are performed by invoking specific processes.

Whatever design methodology/style is employed, an ASIC designer needs to perform a sequence of tasks by using a set of CAD tools and cell libraries. A designer needs to invoke and execute each CAD tool required for ASIC design. This implies that he needs to select an appropriate tool for the given task and then execute that tool so that it accomplishes exactly what needs to be done. To select a tool, the designer needs to know the various available tools for the given task and then chose the best. This can only be done if the designer knows the exact functionality of all the CAD tools. The designer also needs information on the versions of CAD tools available. Thus many decisions are required even before invoking a tool. To execute the selected tool the designer needs to know the exact syntax for tool invocation, and semantics of the tool.

The designer also has to manage all design data files related to the tools. As most tools are incompatible with regards to input and/or output file formats the designer needs to translate these files [BUSH,'89]. For example, to run a simulation on a layout, geometrical information needs to be extracted from the layout and then converted to a file format that can be input to a simulator. The whole design, from behavioral description to mask layout, needs to be steered by the designer using the CAD tools. The designer can be relieved of these painstaking tasks if the following information about all the tools is embedded in the system:
- **Data requirements**; type, access modes, format, etc.
- **Argument definitions**; format, required/optional, purpose, etc.
- **Tool commands**; syntax, arguments, purpose, effect, etc.
- **Resource requirements**; regarding CPU time, memory, etc.
- **Description of tool functions**.

In [DANI,'89] an object-oriented approach to build models of CAD tools has been proposed. By binding a CAD tool to its representative model, they create a CAD tool knowledge object
(CTKO). The CTKO represents the abilities of the tool to the designer, manages the low level programming details associated with the original tool, and provides a control mechanism between the tool and the designer.

**CAD tool integration**

CAD tools in the past decade have automated almost all the phases of the design process. The need of this decade is to integrate CAD tools into a design automation system. ASIC designers require an environment for integrating heterogeneous CAD tools while providing an open and distributed control mechanism. The need for CAD tool integration can be discussed with respect to four sub-topics as follows:

**Design data management** - With increasing design complexity, managing the input and output design files from the CAD tools is becoming extremely difficult. This is true especially when the data generated by CAD tools is enormous.

**Number of CAD tools** - CAD tool vendors are continuously supplying the designer with better CAD tools. Since many tools are incompatible in many ways, it takes a lot of time and effort to integrate them into a comprehensive design environment. This process takes a lot of time and effort especially when the CAD tools have different user-interfaces and file formats.

**Complexity of CAD tools** - Learning new CAD tools is not a trivial task; it involves a lot of time and effort. And if the designer does not continuously update his information about new and better tools he is left with a small set of CAD tools. To remain competitive the designer has to learn, master, and integrate new CAD tools into the existing CAD environment.

**Tool invocation and execution** - An integrated CAD environment will allow the designer to automatically invoke and execute CAD tools to perform a given task.

Three vital aspects of tool integration are visual, data, and control [SUNT, '90].

**Visual integration** - Visual integration implies that all the tools integrated should have the same look and feel. The interface between the designer and the tools should be homogeneous. This interface must hide the specific of tool invocation and execution. The user must be able to choose from a list (usually icons) of tools or the execution environment must be able to choose one automatically based on the functionality of the tool. The interface should track and display the state of any on-going task including status of all related processes.

**Data integration** - Data integration can be classified as data linkage, data interchange, and data sharing as shown in Figure 3.

- **Data linkage** means establishing semantic relationships between pieces of information maintained by different tools. For example, Sun Microsystem's NSE Link Service 1.0 provides limited capability of this type [SUNT, '90].
- **Data interchange** means transferring information between tools in a mutually agreed upon representation. Examples of data interchange mechanism include window cut and paste, data import/export. Case Data Interchange Format (CDIF), is an example of a standard representation [SUNT, '90].
- **Data sharing** means that tools directly access data stored in a mutually accessible place. The Mentor Graphics' data repository approach is an example of data sharing that involves a common database schema and storage of data under a common data management system [WINK, '90].

**Control integration** - Control integration can be classified as interprocess control and meta control as shown in Figure 4.

- **Interprocess control** means that a tool can cause another tool to perform some action.
- **Meta control** means causing a sequence of tool invocations and requisite data inter-changes to accomplish some specific task automatically. Meta control can be implemented in terms of interprocess control, in which an 'agent' tool co-ordinates actions performed by other tools.

**VI. The CAD Framework**

We have already seen the need for CAD tool integration. This need has forced tool developers to create suites of tools that shield the designer from as much lower level detail as possible. Traditionally these suites were tightly integrated into a design environment [SIEW, '83] [BROW, '83]. CAD tools were bound into a monolithic entity. A problem with this approach was that tools were difficult to add or delete. This led to the creation of CAD Frameworks that are intended to be open architectures that support a large population of heterogeneous tools. CAD frameworks may be used to configure a set of VLSI tools and to develop appropriate interfaces to support schematic capture, simulation, timing verification, and test generation for ASIC design [HARR, '90]. It should act like a conduit between the
designer and the tools, matching the abilities of the tools to the needs of the designer. A framework should allow various tools to co-operate and work interactively with each other and with the designer. The designer need not learn all the subtle details of any tool; instead, the tools should present their general abilities to the designer as accurately and concisely as possible.

A generic CAD framework with its major components is shown in Figure 5. At a very high level the CAD framework can be viewed as having five major components described below [CAVI'90]:

- A common user-interface which provides a consistent, graphical, and natural front end to the CAD tools.
- A design database containing design and library information which could be a centralized or distributed database.
- A design management database containing information on revisions, relationships, access authorizations, methodologies, and tools.
- A design data and process manager which utilizes the design management database to control design information and design process.
- CAD tools for all the phases of VLSI design.

![Diagram of a generic CAD framework](image)

**Characteristics**

Some important characteristics of CAD frameworks that directly affect the designer are summarized in this section.

A framework supports multiple users, thus the design can be divided among various designers. A distributed heterogeneous computing environment can be created thus allowing the designers to use tools that work under different computing environments.

The designer does not have to worry about the syntax of tool invocations and the semantics of tool commands since CAD frameworks will allow for automatic execution of tools. The designer is relieved of the burden of making available the proper data at the proper time to all the tools required in the design process. Once fully integrated into a framework all the tools will have the same look and feel making it easier for the designer to get accustomed to many different tools.

The designer can intermix design entry levels, for example hardware description languages (HDLs) and schematics, for specifying a design. CAD frameworks will allow for simulations of such intermixed design entries.

A framework supports different design styles and methodologies. It also allows an expert designer to record and save successful design methodologies, allowing less experienced designers to create good designs using the knowledge of an expert designer. CAD frameworks combine top-down design with bottom-up schematic entry oriented design approach; this lets the designer tackle extremely complex IC designs. The designer can make arbitrary changes to the design at any stage and the framework will take care of design consistency by automatically propagating the effects of the changes to all related files.

Design complexity makes it impossible for the designer to foresee all situations where incompatibilities occur between the physical structure and the functional specifications. CAD frameworks will keep track of various versions of complex designs through all the phases of the development cycle while ensuring data integrity. Therefore, the designer can easily backtrack through all his previous design steps. He can also track the evolution of the design.

The framework eliminates long error-prone translations of data files between CAD tools since all the databases of the individual tools are either linked together by a common database or there is just a single database for all the tools.

CAD frameworks can be used for concurrent hardware and software design. Thus it will be possible to test software code on simulated hardware, permitting changes in either or both, to better meet the needs of the system [ADAM'90]. It decreases the total turn-around time of the design process.

Until now, most of the decisions at every phase of the design were a matter of experience. With increasing complexities of the design process, just one wrong decision can prove fatal. The designer needs some decision support at the earlier stages of the design process. For example, if the designer decides on a particular floorplan he can get information about the affect of that floorplan on the final layout area of chip. Such decision support is provided by the framework.

CAD frameworks will allow the designers to work at a higher level of abstraction by allowing for simulations at the behavioral level. Designers need not become expert users of a baffling array of complex CAD tools, but instead need only be concerned with higher level design tasks. Designers can rectify potential problems before they happen at lower levels of design.

Knowledge required for good designs, such as the fundamentals of material properties, fabrication processes, device characteristics, microelectronics and circuit techniques, can also be built into the frameworks.

**CAD Framework Initiative**

Some vendors have come up with commercially available frameworks, for example Mentor Graphics' Falcon Framework, DEC's PowerFrame and Cadence's Design Framework II. Though there is general agreement over the basic definition of a framework, there are many different ways to implement the idea.

The CAD Framework Initiative (CFI), a body of CAD tool users, vendors, and system integrators was formed to supervise these ongoing framework efforts and lay down industry acceptable guidelines for design frameworks [CADF'90]. CFI's purpose is to provide industry standard interfaces to CAD framework components, thus reducing the cost of combining multi-sourced CAD tools into an effective CAD system.

The CFI group defines a framework [CADF'90] as:

"A software infra-structure that provides an environment where CAD tools are developed, integrated, and operated."

It further specifies that through a CAD framework, a user should be able to launch and manage tools, create, organize, and manage data; graphically view the entire design process; and perform design management tasks such as configuration management, version management, etc..
CFI and CAD tool integration

CFI has come up with a seven layer conceptual model [FRAM,'90] of a framework (Figure 6). Out of these seven layers two layers (Levels 4 and 5) refer specifically to CAD tools. Level 4 allows for tool management which means that the tools have to be managed within the context of the overall design process. It specifies that the frameworks should supply models and services that describe inputs, outputs, options, invocation sequences, control files, etc. to manage tools. Usually tools can be integrated into any CAD system but the integration software has to deal with data translation between the tools. Level 5 allows for easy integration by separating the data descriptions from the tool integration programs.

![Diagram of CFI's conceptual model of frameworks.](image)

VII. Conclusion

In this paper we have identified the ASIC development process and CAD tools required for an ASIC design process. Then we have shown the evolution of computer aided VLSI design from the point where individual CAD tools were available to perform some of the design tasks, to a unified design environment called the CAD Framework. CAD frameworks not only integrate CAD tools into an open environment but also automate the design process. We have identified the impact of CAD frameworks on the designer and thus hope that this paper helps the ASIC designers to introduce the benefits of CAD frameworks into their existing design environment.

We look forward to CAD frameworks that will support a fully automated design process. The designer need only be involved in the process of specification capture and feeding behavioral/functional specifications to the CAD system. By the end of this decade we hope that there will be accepted standards for integrating not only CAD tools but tools that will encompass all other IC related areas such as manufacturing, control systems, process and device modelling, etc.

References


