Design Management In A Workstation Environment

D. Cooke, G. Swan, J. Strout
R. Kane, P. Stevens, J. Yang, D. Chen
Sun Microsystems, Inc.
2550 Garcia Avenue,
Mountain View,
California, CA 94043.

ABSTRACT

This document proposes a design management system which will facilitate the correct and timely creation of VLSI designs. The proposal concentrates on some of the problems uniquely associated with designing IC's using a network of engineering workstations. The requirements for such a system are discussed from both a practical and theoretical viewpoint. The experiences of past VLSI designs and existing design management tools are also discussed. These are blended with the viewpoints to form the basis for the proposed design management system. Finally, the proposed system is described in detail, along with a brief description of its actual implementation.

1. Introduction

The VLSUCMOS group at SUN Microsystems is responsible for the design and implementation of a SPARC (Scalable Processor Archiecture) VLSI circuit for use in a family of SUN general purpose workstations. Currently the group consists of around 30 designers, and 10 CAD/Sys Admin engineers, including engineers from a semiconductor vendor.

Over the past 2 years, the VLSUCMOS group has worked on the design and implementation of a 20 MIP SPARC chip. The lessons learned during the project were fundamental to the ideas we now have on how to control design changes on future projects.

In the IC business, the importance of turning out high quality chips which are functionally correct on first silicon cannot be underestimated. Being on schedule greatly reduces the cost to Sun and to other people, plus ensures Sun's leadership in high-performance, high-quality workstations. However, the market is an aggressive one. Sun has committed to (and the user expects to see) a doubling of performance in Sun workstations around every 18 months. Keeping ahead of this curve forces designer's to use leading edge software and hardware technology. The number of CAD tools now required in a design is massive (the previous SPARC chip used around 100 different tools), and can only increase. Similarly, the size and complexity of the chip designs are increasing exponentially. This leads to an exponential increase in the amount data to be handled. More data, means more people, and more people means more changes.

The major lesson learned on the last SPARC chip was that if the changes made to a design are not carefully controlled, they can destroy the success of the project. To many people, this statement may seem obvious, and adopting some standard revision control system like SCCS will help solve the problem. Sun took this approach, and it did keep the design changes under control. However, we found that revision control is only a small part of the overall design management problem, and in a large group with a large number of networked workstations, the traditional techniques for design management break down if not used with caution. Therefore the rest of this document describes a proposal for a design management system that will attempt to solve most of these problems.

Section 2 describes typical design management approaches that currently exist. The problems with these approaches are outlined, and a brief description of 'areas for improvement' is made. Section 3 describes in detail the proposed design management system. Section 4 then goes on to briefly outline the implementation of this system. Details of the implementation are a subject for a future paper, but it suffices to say that initial performance looks good. Finally Section 5 acknowledges the help of those people who have taken the idea through to reality.

2. Design Management

Design management is a rather vague term which nowadays is used to mean many things. Therefore it is useful to take a moment and describe what the authors of this article believe a design management system to be.

For this paper we define a design management system to be a suite of software tools which control the creation of a design (and all changes manipulation subsequently made to the design) in a way which allows many designers to simultaneously work on the design in an ordered fashion. The design manager manipulates data (i.e files) in the database, but does not concern itself with the contents of this data.

The implementation requirements of an acceptable design manager in a workstation environment are described in Section 3. However, it is useful to first take a look at typical past and present design management systems.

2.1. Existing design management systems

Typically, a design database is created and managed in the following manner:
- The database is usually stored centrally (possibly on one file server).
- Access to the database is through something like Network File System (NFS).
- Serial checkin/checkout. To change data, a designer would check out the data, which in turn gives him write access to the database, and also prevents anyone else from changing the same piece of data. Once the changes are complete, the user would check the data back into the database, unlock the file(s), and create a new revision.

Even though data is revisioned, few database environments can use previous revisions effectively. Instead, they usually use the most current version. If an old version is required, then they generate it from the revisioning system, and this data then becomes the current version.

As the design proceeds, it is useful to make revisions of the whole database. However, if everyone is working off of the most current version of data, then it is very difficult to find a single point in time when the data is stable, and the whole design is consistent.
Changes to the hierarchy are made manually, or possibly with a script to maintain consistency. In general adding/deleting hierarchy simply involves using add/delete directory commands in the native operating system (i.e., design hierarchy is physical, not logical). Similarly, the data in the database is protected using the basic file protection mechanisms provided with the native operating system.

- Designers must run the CAD tools in the correct sequence to generate the data they need. In addition, the designer usually has to decide which machine a particular CAD tool should be run on.
- Validation of the block being worked on is usually performed before the data is checked back into the database. However, it is difficult to validate that the block is valid in relation to the rest of the design.
- In many groups, notification depends on designers communicating. In larger groups, where relying on verbal communication breaks down, some form of electronic mailing system may be put in place to note when data was checked in to the database etc.
- In general, there is no common user interface for all CAD tools used in a design. The designer is required to remember (or figure out) how each tool is run, and in what form the tool expects the input data etc.

2.3. Problems with existing management systems

The above type of systems do work quite well, and do help in the problem of design management. However, for large designs which require a large number of engineers (>10), the system soon starts to break down. Some of the typical problems that arise are:

- A central location for the database helps keep the data consistent, but on large designs, there may not be enough disk space to store all the data together. Even if there is enough disk space, there tends to be disk contention problems when many users access a single fileserver.
- Serial checkout/checkout works fine for small groups where communication is good. In large groups, the process quickly breaks down as people often need previously checked out data to meet their portion of the schedule etc. Rather than wait until the data is checked back in, they circumvent the system by copying all the necessary data to their local machine, and then making the required changes. Merging these changes back into the database then becomes a serious problem (especially if more database changes have been made since the original copy). Typically, some changes get lost in the process.
- Using the most current version causes problems for those people in the project who for one reason or the other, want stability in the design so that they can debug their problem. Copying the 'working' database into a 'golden' database at regular intervals gives such people a 'known-to-be-good' database which is also stable. However, in practice, the transferal of data from 'working' to 'golden' tends to happen infrequently (due to the fluidity of the design), so the 'golden' data quickly becomes out of date, and is therefore useless.
- Relying on the operating system file protections is often troublesome. The problem lies in the fact that it is all too easy to change the protections of a file (accidentally or otherwise), and suddenly data which is supposed to be write protected can be written to by anyone. The fact that data’s write protection has been changed is seldom found until it is too late (i.e., someone managed to write to the library).

- Expecting the designer to remember how to run as many as 100 CAD tools is unrealistic. Good documentation helps, but there are always situations where the documentation is out of date, or incorrect. Often the designer does not know the input parameters required to run a CAD tool or the location of the tool executable(s). This leaves the designer dependent on a CAD group which is frustrating and inefficient for the designer. The designer is left to choose the machine on which to run the CAD tool. In many cases, the only workstation available may be his own one, but in many batch-type tools (i.e., HSPICE) he may be able to run the job on some remote workstation. Without a program (or a network) controller to do "load balancing" in the network, one usually finds that particular workstation(s) on the network get all the work, while other equally competent workstations are lying idle.

2.3. Ideas for the future

Workstation environments are becoming increasingly popular in engineering design groups. Therefore we should rethink any design management methodologies that were developed for mainframe based CAD environments, and develop a design management system which allows the designer to work in a network environment freely, and also utilize the full power of the network whenever possible. Some ideas in this area are:

- Logical view of the database rather than a physical view. By this we mean that the database could be spread over many workstations in a network, or stored centrally on one large fileserver, but as far as the designer is concerned, all the data appears to be resident locally on his own machine. Distributing the database over several workstations in the network has the advantage of significantly reducing the problems of disk contention on the fileservers. Also, if set up correctly, this approach will reduce the network traffic.
- Provide a revisioning system. A good revisioning system is the heart to any good design management system. Being able to revision data as required provides stability to those who need it, provides flexibility for those who want to advance the design, and also solves many of the backup problems associated with large designs.
- Allow parallel checkouts. Since this process is happening anyway, let’s realize that it is happening, and develop some tools which allow parallel checkout in a controlled way. If the design management system can track parallel changes, then the process of merging the two pieces of data becomes much simpler and error-free. Parallel checkout also allows each designer to achieve stability in the design (since he basically has his own copy). His changes are only seen by others once the data is checked back in.
- The design manager should provide a suitable browser which the user can use to view/manipulate the data in the database. If the native operating system is say UNIX! then all UNIX commands (i.e., find, grep etc) should also work in the browser environment, and refer to the name in the logical view of the database, regardless of how the data is actually stored physically.
- The design management software should have a concept of access permissions, and should control access to data, rather than relying on the native operating system to do the work.
- The design management system should also control the CAD tool revisions etc. In this way, derived data (i.e., netlists created from a schematic etc.) can easily be regenerated.
- The design management system should also provide a standard user interface for all CAD tools used to view/modify data in the database. This serves to standardize the use of tools, plus allows the insertion of other tools which will check such things as the data dependencies etc. (i.e. make sure the netlist specified is newer than the schematic BEFORE starting simulation)

*UNIX is a trademark of Bell Laboratories.
3. Proposed design management system

Based on past experiences with design management in large groups, we feel that a suitable design manager should consist of six main pieces:

- **Version control** - As the design progresses, many versions of each portion of the design evolve. The design manager must be able to control the hierarchy of the design. It is imperative that there exists a mechanism which automatically informs designers of changes in the design (i.e., a new revision of a cell in the library, or a new block in the design hierarchy etc.). Early on in the design, notification serves as an information medium. Near the end of the design, notification may serve as the ECN/ECO mechanism which can prevent changes unless all the right people are informed and agree to the change.

- **Notification** - In large groups, it is imperative that there exists a mechanism which automatically informs designers of changes in the design (i.e., a new revision of a cell in the library, or a new block in the design hierarchy etc.). Early on in the design, notification serves as an information medium. Near the end of the design, notification may serve as the ECN/ECO mechanism which can prevent changes unless all the right people are informed and agree to the change.

- **User interface** - With many CAD tools available for use by the designer, it is important that these are utilized in a consistent manner. The design manager should provide an interface to CAD tools. Note however that the design manager has no effect on the user interface once inside a CAD tool. Each of these pieces will now be discussed in more detail.

3.1. Version control

A version controller should provide the following basic features:

- **Incremental and level versioning** - As a design progresses, it passes through many states or versions. A good version controller must be able to manipulate these revisions easily and simultaneously. Versions come in two basic types: *incremental* and *level*. Incremental changes are the step by step, or sequential changes that are made to an object as the design evolves. At some point, these changes are made, a level is reached. A level is a nominal point in the design. It represents the point at which it is desirable to freeze the state of the design. For instance, say a schematic is generated for use by a layout designer in creating the mask geometries of a circuit. The schematic serves as a reference for the layout designer. Therefore, it is desirable to freeze the schematic in this state. However, the logic designer might wish to make further changes to the schematic. And, a circuit designer might wish to make different changes to the schematic.

Figure 3-1 illustrates an example of incremental and level versioning. Each circle represents an incremental version, while each vertical solid line represents different levels. Each circle contains a number, which corresponds to the incremental version of this view. The schematic starts in version 1. A change is made to version 1, and version 2 is created. The designer then decides to maintain this state, but knows that both logic and circuit designers are going to make further changes. Two branches are created, one for logic changes and another for circuit changes (corresponding to the logic and circuit levels respectively). The solid lines between circles in the diagram illustrate this branching. In the logic level, one incremental change is made (from version 5 to version 6). However, it is noticed that there are errors in version 2 of the original schematic; edits are made, which advances the version number to 3. Further changes are made, advancing the version to 4. Notice that the branches have diverged; changes made in the original schematic are not incorporated in the logic schematic. The designer merges the two branches by including changes in the original schematic since the branch to the logic level schematic. This occurs in the figure during the transition from version 6 to version 7. The fine dotted line indicates the merging of the two levels.

Meanwhile, the circuit level is revised from version 8 to version 9. A new version (10) is created to accommodate changes in the original since the branch. The original and the circuit levels are then merged. Finally, the logic changes are included into the circuit level, and all three levels are merged into version 11.

Branching to levels gets complicated. If there are N levels of branching, N(N-1)/2 merges have to be completed before the design is completed. The number of merges may be reduced, in practice, by structuring the process of merging so as to eliminate unnecessary merging.

- **Access control** - Access control is another very important feature in a version controller. The access controller ensures that the design is only changed by authorized people. More specifically, the features required in an access controller are:
  - Definitions of users and groups of users who can access the design.
  - Permissions given to users. For example:
    - Read.
    - Write.
    - Create hierarchy.
    - Create a new version.
    - Create a new level.

Permissions can be defined for each block in the design (i.e., one group has write access to the ALU, while another group has write access to the Register file). If a block has no permissions specified, then by default, it should inherit the permissions from the block above.

- **Utilities accessible by group** - Just as access to the design is restricted, then access to CAD tools should also be restricted (i.e., concept of an executive permission).

- **Links with ECO and ECN** - For most of the design period, the access controller acts in ECO (Engineering Change Notice) mode, and simply notifies managers and dependent
people about changes made to the design. This notification is performed after-the-fact. Near the end of the project, the access controller needs to act in ECO (Engineering Change Order) mode, where no changes are made to the design until all responsible users approve of the change.

- **Maintenance.** The access control definition should be modified through an authorization utility. The default permissions should be maintained by daemon.

- **History.** Version information should be linked to some history mechanism which will record when versions of data in the design changed. This history mechanism should also link with the process controller, so that information about which version of which CAD tool was used to create the data is saved.

  The history mechanism should provide sufficient version information to allow the recreation of design data using a historical trace of the commands originally used to create the data. This would be useful when source data is updated which makes data downstream in the design process out of date. The ability to recreate the design data based on previously entered commands might be limited by the type of commands that have been entered. For instance CAD tools that perform edit like functions will be difficult to re-enter automatically.

Based on these requirements, the history controller would keep track of:

- Creation/Change information.
- CAD tools used to create data.
- Protection changes.
- ECN/ECO information.

Over a period of time the history of a file could become large therefore a means of reducing the amount of historical data associated with a file must be produced.

- **Notification.** The version controller should work with a notification tool (see section 3.5) to inform users about new revisions in the database.

### 3.2. Hierarchy control

A design management system should allow the design database to be hierarchical in nature, and should use this hierarchy to its advantage. In this document, it is assumed that each block in the database may consist of many views of the block (i.e. schematic, layout etc.), or other blocks. (See Figure 3.2)

Note that the physical hierarchy (i.e. directory structure) should not be limited to mirroring the logical hierarchy. In large designs, the database will not reside on one disk, and many CAD tools do not understand symbolic links etc.

It is important that the hierarchy in a design be controlled. If the verification process requires that schematics match the layout in terms of connectivity, then restrictions need to be put in place to make sure that the schematic hierarchy mirrors the layout hierarchy (to some predefined level in the design). In other cases, it is important to control changes to boundaries (i.e. I/O pins) in a block, since all views of the block may have to be updated to reflect this change.

Based, on this, a hierarchy controller should provide the following basic features:

- Add/delete hierarchy in design.
- Copy/Move hierarchy.
- Allow browsing (parent-of and child-of info).
- Allow querying (i.e. Tell me when the hierarchy was last changed, and why).
- Grouping. Allow the user to group many files into one 'virtual' file, which can be then be handled as a single object.

---

**Figure 3-1 Incremental and level versioning**

![Figure 3-1 Incremental and level versioning](image-url)
3.3. Process control

Modern IC design typically uses a large number of CAD tools, all requiring different techniques for invoking them, and different methods of specifying input and output data. A design management system should therefore provide mechanisms to help standardize the use of the tools. This is the job for the process controller. Much of this standardization comes through using a common user interface (described later), but many other areas of standardization remain. For example:

- Checks to see who is allowed to run a particular CAD tool.
- Keep data consistent - Every CAD tool performs (or should perform) the same types of checks during startup and finish. For example:
  - The input/output data must be specified.
  - Once specified, the input output data must be validated (i.e. do the input files exist?).
  - Check to see if the input data is up-to-date. If not, it should ideally automatically regenerate the series of steps required to make the data up-to-date.
- CPU resources - In a network of workstations, there is a large amount of CPU MIPS and disc space available which is usually greatly under-utilized. If all CAD tools are controlled by a process controller, then this controller can be designed to know about job queues, and thus balance the load in the network by distributing batch type jobs to available machines. With such a mechanism, the throughput of CAD jobs can be increased by an order of magnitude or more.
- History tracking - Keep track of which revisions of input data, and which revisions of CAD tools were used to produce output data.

A typical architecture for a simple process controller may be that shown in Figure 3-3:

Since these tasks are common to all CAD tools, it is better (and more efficient) to implement these tasks once in a process controller, and leave the task of operating on the data to the CAD tools. This approach also helps achieve higher quality in those CAD tools that don't do any checking of input data etc.

In addition to this, once designers are familiar with using some sort of shell round their favorite CAD tool, then this gives the CAD engineers an immense amount of freedom to do whatever they like inside of the shell. In this way they can customize the shell to best utilize the available resources. For example, it may make sense in I/O intensive jobs to copy all the data to a local disk, process the data locally, and then copy it back to the global database. Having a shell (such as the process controller) round each CAD tool also isolates the user from any changes required in the way CAD tools are run.

To allow the process controller to handle any CAD tool, there needs to be some form of language which can be used to describe how a particular CAD tool is run (i.e. input files required, optional arguments, machines which can run the CAD tool etc). This language thus serves as the 'master' description for a CAD tool, and can be used for other purposes, such as automatically generating on-line documentation etc.

3.4. Validation

Validation in a design data manager has two flavors:

- Design integrity.
- Data integrity.

![Diagram](image)
As different people work on different portions of the design, it is easy for the whole design to become inconsistent (i.e. each top level unit performs correctly, but the whole design does not, because of an interface mismatch between units). Therefore the validation software should check the consistency of a block’s interface in relation to other blocks in the design before, for instance, a check-in procedure is deemed successful. Similarly, if the boundaries of a block change, the validation software should ensure that all other views of this block also reflect the changes.

The second flavor of validation is data integrity. By this we mean that designers should be notified when upstream data is modified. In the simplest case, users should be mailed when changes occurred. Warnings should also be sent out when a CAD tool tries to use a file derived from an out-of-date source file. In the ultimate system, there would be links with the history controller so that it can be determined what has to be done to bring the data up-to-date.

However, the system should not be sensitive to system problems such as:-

- Crashes.
- Insufficient disk space.
- Program core dumps.

3.5. Notification

In large groups, it is imperative that there exists a mechanism which automatically informs designers of changes in the design, since verbal communication is often very poor. This task should be handled by a notification manager.

During a design, there are fundamentally two types of notification required; ECN (Engineering Change Notice) and ECO (Engineering Change Order). ECN usually exists for most of the project, and is simply a phase whereby interested designers are informed about changes made elsewhere in the design. Note that the changes are entered into the database, and designers are notified after-the-fact. In practice, a whole range of actions could be deemed worthy of notification, for example:-

- Data in the parent design changed.
- Someone created a new version of a block in the design.
- The blocks currently 'checked-out' by a designer.

As the design nears completion, and the database is verified as being correct, then the design tends to enter the ECO phase. In this phase, no-one can make any changes to the database until one or more designers have agreed to the specified change. In the past, ECO has typically been handled using hard-copy forms which are signed-off by the respective designers. Ideally, in the present world of remote workstations and electronic mail, the notification manager should be able to perform a similar function.

3.6. User Interface

With many CAD tools available for use by the designer, it is important that these tools are utilized and presented in a consistent manner. The design manager should provide such an interface to CAD tools. Note however that the design manager has no effect on the user interface once inside a CAD tool.

4. Implementation

At Sun, we have implemented a portion of the proposed design management system. Details of this implementation are the topic for a future paper, but briefly, the key points to the implementation are:-

- Hardware environment - The hardware requirements for the current project have been determined in light of the design management system. The main features of the environment are:-
  - Around 50 Sun workstations networked together
  - Design database is spread over 3 file servers
  - Network is split into 3 sub-networks (1 per file server). This helps minimize network traffic. Gateways connect sub-networks to other sub-networks etc.
  - Workstations are split into user workstations (workstations with monitors), and compute workstations. The compute workstations are distributed throughout the network, and will handle the majority of the compute intensive tasks (under control of the process controller).
  - Change control handled by NSE. NSE (Network Software Environment) is a tool developed at Sun which provides many of the features defined in the proposed design management system. The main features of NSE are:-

![Figure 3-3 Process control architecture](image-url)
5. Acknowledgements

This paper would not have been possible without the collective experiences of the engineers in the VLSI group at Sun, who have used and commented on this VLSI CAD environment development project.

Also, we would like to acknowledge the NSE team for helping us to crystallize our ideas while developing a design management system around the NSE.

6. References


---

Figure 4-1  NSE Browser and Menus