

# A VLSI Design Management Environment

Steve Hodges and Peter Rounce

## 1. Introduction

Advances in technology are continually increasing the number of discrete devices which may be fabricated in a single integrated circuit. When we talk of VLSI chips today, they may well contain in excess of one million transistors - a figure which would have seemed almost unbelievable a few years ago. With this dramatic increase in complexity, it is more important than ever before to control the design process, thus maintaining the required quality, reliability and extensibility of a given design.

Equally, speed of development in a rapidly changing market, development costs and the cost of mistakes play important parts in a commercial environment. It is becoming increasingly important that the design process is as quick as possible, as cheap as possible and that mistakes are highlighted as early as possible (certainly before fabrication).

In a recent study of these problems in VLSI design, in particular with reference to the design methods and tools used in the Computer Science Department at UCL, a report was prepared, detailing the structure of a 'Digital Design Environment' (DDE)<sup>†</sup>. The issues brought to light by this study, and the solutions put forward are summarised below.

## 2. Problems with the Current System

### 2.1. Outline of the Facilities Within the Department

The Department has a large number of workstations, mainly Sun 3 and Sun 4 series machines, along with Hewlett Packard workstations, DecStations and VaxStations. They all run variants of the Unix operating system (SunOS, 4.3 Utah and Ultrix), and are all connected via a series of ethernet. Several machines are configured as file servers.

### 2.2. Use of Different Tools

Within the Computer Science Department at UCL, many different tools are used through the VLSI design process. These typically range from silicon layout editors, schematic capture tools and logic generators to logic optimisers and digital simulators. Whilst not every tool may be used in the creation of every design, it is likely that a large subset will, and it is the combination of these tools which causes a large number of problems. These are listed below:

- (i) A given tool may well be restricted to running on one machine architecture: another tool might only run on a different machine. This means that the designer must log in to the correct machine at each stage, before running the associated tool. There may also be problems accessing the correct files from different machines: not all machines have access to all file stores.
- (ii) As a design is processed, it must be passed from tool to tool. For example, the designer may use a schematic capture tool for initial input, then wish for their design to be minimised, and finally simulated. However, nearly every tool expects its input and produces its output in a unique format. Whilst many tools include options to read and write files in different formats, there is rarely one which is common to them all. This makes the passing of design data from one tool to the next very tedious and error prone; generally some form of file format conversion must be done at each stage.

---

<sup>†</sup> The DDE was conceived by Peter Rounce, and the study and associated report carried out by Paul Griffiths, Odiseas Mihanetzi, Grant Worsfold, Sumit Sahni, Flange Farnell and Steve Hodges.

- (ii) Each tool must be invoked manually: it is therefore quite easy for human error to creep in. This may take the form of a forgotten flag (perhaps with minor side-effects), but at the other extreme, it is quite easy (in Unix) to overwrite the source file with a mis-typed file redirection command. Of course, it is quite possible that the designer will forget to run the design through a certain tool altogether: a logic minimiser for example.
- (iv) Many of the tools used in the Department take a long time to run when the design is complicated - typically, a simulation of a large processor may be left running overnight. Some of these tools also have a tendency to crash, which often results in the loss of work. Of course, if the simulation which is left running overnight crashes, then it too is lost.

### 2.3. Design Management

Careful and comprehensive management of the design as it is developed is very important. It is all too easy to make changes to a design without documenting them, or without keeping backup versions in case references to these changes need to be made at a later date. In a similar way, it is equally possible to accidentally delete files with no means of retrieval. All of these inadequacies are highly undesirable.

Problems also arise with large designs. These are typically split into smaller blocks, which are designed and tested first, and then combined. This process may be applied at many levels, thus building up a hierarchy of blocks, with each level making use of the blocks in the levels below. However the various tools often do not support this system of hierarchy directly, and the descriptions of the blocks must be combined before the complete design can be processed. Once again there is a chance for human error to enter.

When separate blocks have information in common, for example the opcodes for a processor, this data must be replicated to facilitate individual simulation of each block. Any inconsistencies in the data obviously need to be avoided to prevent errors, and it is up to the designer to make sure everything is kept up to date.

### 2.4. New Tools

New VLSI design tools are continually being developed and released, many of them provide improvements over the old tools or give new facilities to the designer. The addition of the tool may well significantly alter the design process and introduce new file formats. At the same time, the designer may wish to apply the tool to the blocks already completed in a given design, as well as those yet to be designed. This is once again a fairly tedious manual process, and may take a considerable time.

## 3. Proposed Changes

### 3.1. User Interface

The proposal of the report was that a Digital Design Environment (DDE) should be developed. This would primarily consist of a user interface, which would be used by the designer to access all the tools. The interface would ideally be of the mouse/windows type, probably X windows; X is a widely accepted standard, and it seems likely that the designer would have access to a workstation with a bit-mapped display. The user interface provided by the DDE would have to be used to access the design tools - under no circumstances could the designer be allowed to use the tools directly. In this way, there would be a totally uniform interface to the various tools, with command line options and flags automatically taken care of.

### 3.2 Expandability and Adaptability

All aspects of the DDE should be totally configurable. This would enable the designer to incorporate new tools by specifying the file formats used, the function of the tool and so on. By including a list of machines on which the tool could be run, the associated problems could be automatically taken care of. The structure of the menus of the user interface could also be altered according to personal taste, or to suit a particular project. This might include adding frequently used Unix commands to the menus, or changing the orders of choices. Also, the user could easily set up their favourite editor to be invoked where necessary, configure different tools to be used for different blocks, and so on.

### 3.3. Process Management

The design process would be closely managed by the DDE, enforcing the designer to follow the design stages in the correct order. As a design is developed, the DDE would aid the designer by automatically applying the tools where appropriate. It would also be able to collate the test results and compare them with the last results; in this way, the effects of changes in the design would be readily highlighted. The test input patterns for a block could be generated from the block's logical description (e.g. finite state machine definition), to allow totally automatic testing.

When hurried, for example when nearing a deadline, it is all too easy to mis-type commands or forget to do certain things. With the process management provided by the DDE, this would not be possible.

There would also be an option to leave the DDE running through the design process unattended, if for example, a large design was to be simulated. The running of the various tools would be done automatically, and state information kept from stage to stage, so that following a machine failure and reboot, the minimum of processing would have to be repeated.

### 3.4. Design Management

The DDE would make extensive use of a revision control system (such as rcs in Unix). This would make sure that every version of every part of a design was available, along with the results of every associated simulation. The designer would have no choice but to use this, and would be forced to enter a comment for each change made. The system would also prevent unauthorised users from looking at or changing designs not created by themselves.

Hierarchical designs would be made easier by the introduction of simple descriptions outlining the relationships between different blocks in the hierarchy. Having entered these relationships, the structure of the design would be enforced. New blocks could be created easily by using a similar existing block as a basis, thereby setting up all the necessary files automatically. Shared data would ideally be held in a central area, with no replication, thus eliminating any problems of inconsistencies. Failing this, relationships between replicated data would be maintained, therefore allowing automatic propagation of changes through the system, and again taking the onus from the designer.

### 3.5. Invisibility

Finally, all the details not pertinent to the designer's use of the system would be hidden. For example, the file format conversion required between tools would happen automatically, without the knowledge of the designer. It would be possible to introduce a common format for use with all of the tools, along with filters to be used before and after each tool to perform any required conversion. In this way, any new tool requires at most two new filters, and it can then interface with any other tool. (The EDIF language is probably most suitable for this purpose.)

Similarly, if a certain tool runs only on certain machines, then this will be handled automatically. Even if the machine running the DDE is not able to run a certain tool, it will send the job to one which can, wait for completion and get the results, all totally invisibly to the designer.

## 4. Summary

The conclusion of the report is that together, the features outlined in the section above form a powerful and versatile environment to aid the VLSI design process. Whilst this study was carried out with the Computer Science Department at UCL, as the sole subject, it is likely that the DDE would provide features that would benefit the VLSI design process anywhere.

The key aspects of the DDE are to remove unnecessary, repetitive work from the designer, leaving them to concentrate on the design itself. This should in turn make VLSI products cheaper and quicker to make, and reduce errors. By also preventing the designer from making short-cuts, mistakes which are made are recoverable, and the designer has a comprehensive history of the design (along with comments) to refer to. Finally, by making the DDE highly configurable, it can be tailored to suit every designer's need, and has little chance of becoming obsolete, since new tools can easily be incorporated.