Zeitschrift: Bulletin des Schweizerischen Elektrotechnischen Vereins, des Verbandes Schweizerischer Elektrizitätsunternehmen = Bulletin de l'Association suisse des électriciens, de l'Association des entreprises électriques suisses

Herausgeber: Schweizerischer Elektrotechnischer Verein ; Verband Schweizerischer Elektrizitätsunternehmen

Band: 78 (1987)

Heft: 11

Artikel: Computers for VLSI Design

Autor: Lamb, P.

DOI: https://doi.org/10.5169/seals-903871

Computers for VLSI Design

P. Lamb

The choice of computer systems for VLSI Computer-Aided Design (CAD) is enormously wide. In this article, the selection of computer systems is looked at from the point of view of the typical CAD tasks which are performed in VLSI design and what sort of hardware can be effectively used in the major areas of the design problem. Finally, the problems of maintaining a mixed computer environment, and the tradeoffs which must be made in such an environment, are briefly addressed.

Die Auswahl an Computersystemen, die sich für den Entwurf (CAD) von VLSI-Schaltungen eignen, ist gross. In diesem Beitrag werden Auswahlkriterien besprochen, die sich aus den typischen Hard- und Softwareaufgaben beim VLSI-Design ergeben. Es werden auch kurz die Probleme einer gemischten Computerumgebung sowie die Kompromisse, welche diese verlangt, besprochen.

Le choix des systèmes d'ordinateurs, qui conviennent pour l'assistance à la conception de circuits VLSI, est considérable. Discussion de ses critères, qui résultent des tâches typiques concernant le matériel et le logiciel. Bref aperçu des problèmes d'un environnement mixte de l'ordinateur et des compromis à admettre.

Address of the author
Peter Lamb, dipl. Ing., Institute for Integrated Systems, ETH-Zentrum, 8092 Zürich.

In deciding on the computing resources needed for VLSI design, it is important first to look at the tasks which are commonly done in VLSI computer-aided design (CAD), and the computing resources which are desirable to carry out the task. In the following the CAD area is divided into three major groups: VLSI circuit and layout design, simulation and layout verification.

1. VLSI circuit and layout design

Table I shows this area divided in two major applications: graphic entry programs and programs for conversion of symbolic (so-called "sticks") layout into VLSI mask geometry.

1.1 Graphic entry programs and graphics devices

Graphic entry programs are typically conventional two-dimensional drawing programs, with drawing on multiple layers. The ability to represent multiple layers is less important for schematic capture than for layout, although it may be desirable to handle and display logically distinct parts of the schematic on different layers and in different colours, or in different line styles on monochrome devices. Schematic capture applications tend towards line graphics, while layout programs use filled areas far more, especially filled and pattern-filled rectangles.

True vector graphics devices have almost disappeared from the market for VLSI CAD applications, so we will concentrate here on the characteristics of raster graphics devices. The resolution of a displayed image is a result of the digital resolution of the stored raster image, the quality of the digital/analog converters and video amplifiers in the graphics interface and monitor and the quality of the monitor tube itself. Low cost is usually an indicator that the quality of one or more of the key components is lower than desirable. High cost does not always indicate the reverse.

The quality of color monitors varies considerably. It is a deciding factor on the strain placed on a designer working with such a system. The main variables are the resolution of the final image, the amount of flicker, the size of the screen and environmental factors such as the type of lighting, sources of reflection, and noise produced by the monitor and any other associated hardware.

The type of image displayed by VLSI layout editors is also such that low quality in any of these areas is accentuated. Typically, the image displayed contains large numbers of vertical and horizontal *edges*, so that any *distortion* of the image shape is imme-

Table I
VLSI circuit and layout design

Problem area	Algorithm style	Computation needs
Schematic capture	Graphics	Moderate CPU-screen bandwidth
Layout editing	Graphics	Moderate to high CPU-screen bandwidth, colour.
Symbolic layout compaction	Large, complex data structures	Fast integer arithmetic, array indexing and pointer manipulation; large real and virtual memory spaces.

diately noticeable. The edges delineate areas of high colour contrast, so that any small misalignment of the red, green and blue components of the image results in bands of the incorrect colour at these boundaries. Finally there are often large areas which should be of constant colour (such as the background of the image), which sometimes are not constant when displayed. All of these factors are relatively unimportant for displays of natural scenery, where there are fewer long, straight edges, weaker colour contrast and usually no large areas of constant colour.

In order to display a *flicker-free* image the monitor should have a high scan rate (70 Hz or more) and not use interlacing. Displays with a scan rate which is a multiple of the mains power supply should be avoided, since their flicker is particularly bad in rooms with artificial lighting. Interlacing should be avoided, since this can cause extremely bad flicker for particular types of pattern. A pattern which is made up of horizontal lines on every second scan line of an interlaced display is most disturbing. Monochrome displays can reduce this problem by using screen phosphors with a long time constant, but this is usually not possible for color screens.

In general, smaller monitors render a sharper and better aligned picture than larger monitors. However, as the cost of memory decreases, and more resolution becomes available from the digital hardware, smaller monitors begin to approach the limits of the visibility of small graphic objects (lettering in particular).

Graphics application software is now demanding higher data bandwidths between the CPU and the graphic display. Graphic entry programs become annoying when there is an appreciable delay between a user action (typing a key, or "pointing" using a mouse or graphics tablet) and the reaction of the system. This is often the case when the user communicates over a serial line at limited speed, and the application is running on an overloaded timesharing system. It is in this area where graphics workstations should excel. In such a workstation, the CPU is local, so there is no competition for the CPU from other users, the raster graphics device is usually memory-mapped so that it can be accessed at full CPU speed, and there is often hardware assistance for graphics operations.

The most popular devices for *graphic input* are the graphic tablet and the "mouse". The graphic tablet consists of a special surface and either a stylus or a small box which can be moved over the surface in order to indicate position. The stylus or box (puck) has buttons to indicate when the coordinates shown should be transmitted to the application program. Mice, in the computer graphics sense, come in two varieties: one uses a mechanical movement sensor, and can be used on any surface that is not too smooth, and the other variety uses an optical sensor which must be used over a special surface. The main difference between the mouse and the graphic tablet is that the same position on the graphic tablet always indicates the same position on the graphics screen, where the mouse returns only relative position. The two devices are quite similar in their use apart from the fact, that a tablet can be used with an off-screen command menu (by using an overlay on the tablet surface) and a mouse can be effectively used in a much smaller space.

1.2 Symbolic layout compaction

Symbolic layout is becoming a more widely used mechanism for constructing a VLSI mask layout. This method requires that the user lay out an approximate view of the final mask layout. This approximate or symbolic layout consists of transistors, wires and contacts only, laid out on an undimensioned grid. The final mask details and the spacing between the elements is then calculated by program, a symbolic layout compacter.

When an entire chip is constructed in this manner, the amount of data which is processed by the compacter is enormous. Compaction of the Flintstone[1] datapath e.g. takes approximately 17½ hours, requires more than 100 Mbyte of virtual memory space on a computer with 8 Mbyte of physical memory, and, not surprisingly, pages[2] heavily (about 600 000 page faults). Similar experiences have been noted for other chips using this technique.

The computation requirements for this task are for a large virtual address space, in order to deal with the problem at all, a large physical memory space in order to reduce the amount of paging activity and so use the CPU speed effectively, and fast disks for when the paging occurs. For the compaction mentioned above, a Sun 3/160C workstation with 8 Mbyte memory, running SunOS (Sun's version of the Unix operating system) was used. Paging took place remotely over Ethernet on a Sun file server. This configuration falls somewhat short of the ideal, because of the relatively small amount of memory available, and the indirect access to the paging device.

2. Simulation

Simulation plays an important role in VLSI design. The requirements on analytical and event-driven simulators are listed in Table II.

[1] This 20 000 transistor microprocessor, which is being developed in the Institute for Integrated Systems, ETHZ, is described by the article by *T. von Eicken*, also in this issue.

[2] Since the virtual memory used is so much larger than physical memory, program pages must be frequently exchanged between main memory and disk.

Table II
Simulation

Problem area	Algorithm style	Computation needs
Process	Numerical analysis	Fast floating point processing, vector processing.
Device	Numerical analysis	Fast floating point processing, vector processing.
Circuit	Numerical analysis	Fast floating point processing, vector processing.
Gate/switch	Event-driven	Fast integer arithmetic, array indexing and pointer manipulation.
RTL/architecture	Event- or clock-driven	Fast integer arithmetic, array indexing and pointer manipulation.

2.1 Floating-point intensive simulation

VLSI wafer processing simulation, device modelling in two and three dimensions and circuit analysis, particularly transient analysis are large consumers of CPU time in VLSI design. The algorithms used in these simulators are generally linear algebraic manipulations of large sparse matrices[3], and so are typically heavy users of floating-point arithmetic. These problems are often amenable to calculation on vector processors such as the Cray, or less expensive "near-supercomputers" like the Alliant and Convex.

However, careful analysis of the application programs with program flow tracing software is necessary (in order to find the computationally intensive parts of a program) before deciding on one of these machines, since the performance of vector processors heavily depends on the existence of easily vectorizable operations on long vectors or arrays. If the problem does not have these characteristics, the performance of the machine may be disappointing in comparison to conventional computers of similar cost.

Most programs written for conventional processors will need some rewriting in order that a vector processor can approach its maximum performance. This will usually consist of inserting compiler directives in the code in order to permit the compiler to vectorize loops which in the most general case are not strictly vectorizable. For example:

```
      SUBROUTINE MULT (A, B, C, N)
      REAL A(N), B(N), C(N)
      DO 10 I = 1,N
10        A(I) = B(I) * C(I)
      RETURN
      END
```

At first sight this FORTRAN subroutine looks perfect for vectorization; but consider a call of this subroutine as:

```
REAL X(100)
CALL MULT (X(3), X(2), X(1), 98)
```

Since the formal parameters of MULT now correspond to actual parameters which are parts of the same array and therefore overlapping in memory, the result returned by a vector processor will be different from that returned by a conventional computer. The compiler, then, should not automatically vectorize MULT. A programmer must decide that MULT is not used in such a peculiar manner anywhere in the program which will use the subroutine. In a large program, and for routines less simple than MULT, this is not always an easy matter.

Occasionally, a program may run faster if selected pieces of code are not automatically vectorized. Since there is usually a significant setup time for the vector hardware (filling and emptying the vector pipeline), small loops may run faster if they are not vectorized. The compiler should recognize this automatically if the length of the loop is known at compile time and not vectorize if the loop length is too small. If the loop length can only be determined at run time, then the decision must be made by the programmer.

A third change that may need to be made is inversion of some program loops:

```
One way to loop:
      DO 10 I = 1,10000
        DO 10 J = 1,4
10          A(I,J) = B(I,J) + C(I,J)
The other way...
      DO 10 J = 1,4
        DO 10 I = 1,10000
10          A(I,J) = B(I,J) + C(I,J)
```

The second loop will vectorize far better than the first (it will also have better paging behavior on a conventional computer).

Deciding between a conventional and a vector processor depends on the amount of computation used by programs which can be vectorized, on the extent of modifications which need to be made to the applications in order to use the power of the vector processor and, of course, the relative costs of conventional machines and vector processors.

2.2 Event-driven simulation

Event-driven simulation generally has different characteristics from analytical simulation. By simplifying the model used in the simulation to transistor switches or to gate level or higher abstractions, the concentration on floating-point arithmetic and on solution of the whole circuit at each time point moves to solution of only those parts of the circuit which are changing, to discretized values for nodes and often discretized time intervals.

As the level of simulation gets higher, the main computational load moves from iterative floating-point operations on large sparse arrays to event list manipulation and discrete evaluation of node and device lists, now effectively randomly accessed under the control of the event manager. Vectorization of such programs is difficult, special floating-point hardware helps little (even if it helps speed up integer multiplication and division, too). What is needed with current compiler technology is simply the fastest integer operations possible on conventional computers.

Virtual address space is also not such a great problem: switch-level simulation of the Flint Stone processor mentioned earlier causes no particular problems for our computer systems, apart from Speed. Our testing of low-cost vector machines so far has been disappointing for problems like this and the symbolic compaction mentioned earlier.

3. VLSI layout verification

Before fabrication, VLSI Layouts must be checked for consistency against design rules. This includes checking the layout geometry, extracting the circuit netlist and checking this netlist for electrical consistency (Table III).

Table III
Layout verification

Problem area	Algorithm style	Computation needs
Geometrical check	Polygon algebra	Fast sorting of large files, fast disk I/O, fast integer/bit manipulation.
Circuit extraction	Polygon algebra	Fast sorting of large files, fast disk I/O, fast integer/bit manipulation.
Electrical consistency check	Complex data structures	Fast integer arithmetic, array indexing and pointer manipulation.

[3] matrices with a large number of zeros.

Geometrical checking and circuit extraction for VLSI handle large amounts of data. Most algorithms for these tasks require that the layout data is sorted in at least one axis, but once this is done, the amount of input data in the program at any one time is simply all objects crossing a narrow band which moves across the chip: as it moves, objects are read in and discarded as needed. The sort operation is often one of the more time-consuming parts of this process, and requires an efficient sorter, capable of handling large data collections, and fast disk I/O support on the computer. Electrical consistancy checking requires the program to trace through the electrical netlist to find transistors which are not properly connected to the circuit. This requires similar processing capabilities to digital simulation of the circuit.

4. Miscellaneous

Format conversion between various data formats used in VLSI design tools is an uninteresting but vital part of any design system built up from parts obtained from different suppliers (and sometimes from the same supplier). In general it requires fast disk I/O, since the files are often fully expanded representations of the design, and so quite large. Fast conversion is needed between representation in the input file and a common internal representation and between this internal form and the output form. This is doubly important if the conversion goes through a common external representation in a two-stage conversion process. This two-stage conversion between formats reduces the number of conversion programs required to cover all possible format conversions, but at the cost of a double conversion each time.

5. Conclusion

A the moment it appears that there is no single computer which recommends itself to the solution of all the processing problems of VLSI CAD. Workstations are effective for graphic input, moderate-size digital simulations and small analog simulations. Conventional mainframe computers can cover large digital simulations, analog simulations of subsystems and tasks which require high I/O bandwidth either for file operations or paging. Super-computer architectures do not always provide a cost-effective coverage of computing tasks which are not vector based, such as symbolic layout compaction and event-driven simulations. Once a mixed solution is needed, the problems of communication and compatibility between the computers become the main impediment. This can be attacked only by covering the three following problems

a. compatibility of communications hardware and protocols,
b. reduction of training and other overheads by the use of a common operating environment,
c. the use of portable high-level languages and language environments in software written for use on more than one type of machine.

The third problem here has been so much discussed that I hesitate to add more to this almost-won argument. Important here is that the language environment, especially its run-time libraries are portable, as well as the language constructs themselves. The other two problems can be approached in two ways. One is to restrict the choice of computers to one manufacturer, in the knowledge that while the machines may today be a cost-effective choice, they may not remain so, and the cost of a changeover to another manufacturer may be very high. The other is to select a mixed hardware solution and a common operating and networking system. At the moment this is reasonable only with the Unix operating system and TCP/IP networking protocols on Ethernet hardware. This allows a very wide choice of hardware possibilities from personal computers through to the fastest supercomputers; Unix is available for almost all computer systems, either as the only, or as an alternative operating system.

Sometimes Unix, because of its portability, does not use the full hardware capability fo operating system support available for a machine—this is similar to the lower speed obtainable from a high-level language compiler in comparison to assembly coding. Portability in computer systems is rarely without a small efficiency penalty.

The tendency in the workstation market is clearly towards Unix. The cost of developing a new operating system is so high that it cannot be undertaken by the relatively small companies involved in this market, especially when they are under strong pressure for continuous hardware improvement.

An example of a mixed computing system for VLSI design

The hardware spectrum in the Institute for Integrated Systems, ETH Zurich, is not very broad—there are machines from two manufacturers—DEC and Sun Microsystems. Currently, the upgrade of the VAX 11/785 to some high-performance machine is being considered. Computers from the two manufacturers already represented come into consideration, of course, but also "near-supercomputers" and high-performance conventional computers from other suppliers. Experience with benchmarking some of these alternative machines, and of porting software between VAX and Sun (both running Unix) has been convincing that there are relatively few problems to be expected in introducing another computer manufacturer.

[4] 3 VAX 11/785, 2 microVAX, 35 Sun workstations, 5 Sun file Servers and 2 Symbolics LISP machines.