CASE STUDY OF A VLSI DESIGN PROJECT: A SIMPLE INNER PRODUCT MACHINE*

R.A. Rutenbar and Y.E. Park

Department of Electrical and Computer Engineering
and
Program in Computer, Information and Control Engineering
The University of Michigan
Ann Arbor, Michigan 48109

Abstract

We present a case study of the application of recently evolved structured VLSI design methodologies to the design and implementation of a simple VLSI quasi-serial inner product machine.

1. Introduction

The recent introduction of structured VLSI design methodologies, such as the Mead and Conway discipline [1], has made VLSI implementation available to a much larger group of digital designers. In particular, the effective decoupling of architecture and fabrication inherent in these disciplines makes feasible the low-volume production of special-purpose chips for application in research and university environments. The advantages to the arithmetic designer of being able to design, build, and benchmark a large, novel or application-directed design are significant.

As an example of the design process involved, this paper presents a case study of the implementation of a simple VLSI inner product machine. The inner product chip was designed by the authors as part of a course project for a VLSI design class, first offered in the fall of 1980 at the University of Michigan. We will discuss the design process starting with an initial set of specifications, and then describe the top-down descent through three design hierarchies: arithmetic design, register-transfer design, and VLSI mask design.

2. Specifications

The original choice to implement the inner product function arose both from its utility as a fundamental buiding block, and from experience in using it in some high-speed image reconstruction applications [2],[3]. After this initial selection, the two constraints most influencing the specification process were (1) the fact that this was our first fabrication run, and (2) the restricted time frame (about 6 weeks) in which to complete the design. Accordingly, we adopted a conservative approach to the entire project. The following sections describe some of the relevant constraints considered in arriving at the set of specifications shown in Table 1.

Table 1. Chip Specifications

Function:	$P \leftarrow \sum_{i=1}^{N} A_i B_i$
Inputs:	A_i, B_i, 8-bit 2's compl. integers
Output:	P, 16-bit 2's complement integer
Structure:	2's complement quasi-serial multiply/add
Timing:	Externally supplied counter

2.1. Basic Operation

Given two vectors $\underline{A}=(A_1,A_2,\ldots,A_N)$ and $\underline{B}=(B_1,B_2,\ldots,B_N)$, we wish to compute their inner product P, where

$$P = \sum_{i=1}^{N} A_i B_i .$$

The inner product chip has two inputs, A and B, and one output P. The chip is initialized by setting P=0. When two inputs A_k and B_k are presented at the A and B inputs, the chip performs the computation $P \leftarrow P + A_k B_k$. P is always available as an output after this operation, i.e., P is a partial inner product.

2.2. Operand Length

Maximal operand length was an initial goal; unfortunately, simplicity required the assignment of a unique bonding pad to each operand bit to avoid any time multiplexing of the pins on the integrated circuit package. Because of the practical limitation of having at most a 40 pin package, we chose to use 8 bit integer operands. This committed at least (8+8+2x8)=32 pins of the chip to operand I/O and left a few for control signals.

* This work is supported in part by the National Science Foundation, Division of Mathematics and Computer Science, under Grant no. MCS-8007298.

2.3. Operand Sign

We resolved at an early stage to use two's complement operands--if at all possible. The relevant problem here was to find an arithmetic structure with low enough circuit complexity to be implementable in our time frame.

2.4. Arithmetic Structure

We considered a broad spectrum of available, realizable structures, ranging from fast, parallel, combinational-array multiplier/adders to the more mundane shift/add approaches. However, because our primary goal was to obtain a rapidly implementable design, and not necessarily to meet any arbitrary speed criteria, we immediately ruled out the more hardware intensive parallel structures. Of the remaining sequential approaches, it was essential to choose a structure that handled signed operands without several special cases or end effects; i.e., we needed a datapath that was easy to control. We chose a structure based on a two's complement quasi-serial multiplier [4] which, because of its uniform handling of signed operands, and its decision-free control-path, fit well with our requirements and our time frame.

2.5. System Controller

From the previous requirement for a simply controlled datapath, we posed the following question: do we really need a controller on the chip? The Mead and Conway methodology advocated the use of PLA-based finite state machines as system controllers. Although this approach was attractive because it was conceptually straightforward and cleanly implemented, we chose not to put a controller on the chip; control timing will be supplied externally by the user. We did this for the following two reasons:

(1) The quasi-serial structure requires only a counter for its control, thus it is not inconvenient for the user to supply the timing signals.

(2) Counters tend to require notoriously large PLA's. Hence, rather than spend time on some clever alternate structure (e.g., an FIR filter implemented with shift register feedback [5]), and because of some serious concern about the silicon area available to this project, we decided to omit it entirely.

The only disadvantage of this decision was that the inclusion of overflow bits in the result, i.e., the extension of the length of the sum P beyond the 16 bits of each product A_iB_i, required the user to supply some awkward timing signals. Because these extra bits were not essential to a prototype chip such as ours, they were omitted.

3. Arithmetic Design Level

For the first level of the design hierarchy, we will consider the algorithmic basis of the arithmetic chosen to implement our previously specified inner product function. The central algorithm of this section, quasi-serial multiplication was first developed by Swartzlander [6] for sign-magnitude operands, and was later extended to the two's complement case by McDonald and Guha [4]. High-speed, parallel inner product implementations based on the use of N sign-magnitude quasi-serial multipliers for N-element operand vectors have been previously considered [7]. Our low-speed design uses only a single two's complement quasi-serial multiplier and processes the N pairs of input elements in two operand vectors in N sequential multiply/add steps. We first briefly review the development of the two's complement quasi-serial multiplication algorithm presented in [4], and then show that the extension to an inner product algorithm is trivial.

Consider two n-bit two's complement numbers A and B where

$$A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

$$B = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i.$$

Let P be the 2n-bit product AB. By simple sign extension, increase the length of A to k bits, where $k > 2n$; call this longer version A'. If we let $P' = A'B$, then obviously $P' = P$; i.e., the lower 2n bits of the two's complement representation of P' are identically the bits of P. After multiplying the expressions for A' and B and rearranging we find that

$$P' = -a_{k-1}\sum_{i=0}^{n-2} b_i 2^{k-1+i} + \sum_{i=0}^{k-2}\sum_{j=0}^{n-2} a_i b_j 2^{i+j} \qquad (1)$$

$$- 2^{n-1}(-a_{k-1}b_{n-1}2^{k-1} + \sum_{i=0}^{k-2} a_i b_{n-1} 2^i).$$

Recognizing that we are really interested in P means that we can omit any terms in P' involving powers of 2 beyond 2^{2n-1}. Thus, because $k > 2n$, the first term of (1) can be omitted entirely, and the double sum can be simplified to

$$\sum_{j=0}^{n-2}\sum_{i=0}^{2n-1-j} a_i b_j 2^{j+i}. \qquad (2)$$

If we now complement the parenthesized part of the last term in (1), we may replace the subtraction of that term with the addition of

$$2^{n-1}(\overline{-a_{k-1}b_{n-1}}2^{k-1} + \sum_{i=0}^{k-2}\overline{a_i b_{n-1}}2^i + 1)$$

which, after the omission of unwanted higher order terms, simplifies to

$$\sum_{i=0}^{n}\overline{a_i b_{n-1}}2^{n-1+i} + 2^{n-1}. \qquad (3)$$

Thus, the 2n-bit product P can be computed from (2) and (3) as

$$\sum_{j=0}^{n-2}\sum_{i=0}^{2n-1-j} a_i b_j 2^{j+i} + \sum_{i=0}^{n}\overline{a_i b_{n-1}}2^{n-1+i} + 2^{n-1} \qquad (4)$$

where $a_i = a_{n-1}$ for $i \geq n$.

$$
\begin{array}{cccccccccccccccc}
a_7b_0 & a_7b_0 & a_7b_0 & a_7b_0 & a_7b_0 & a_7b_0 & a_7b_0 & a_7b_0 & a_7b_0 & a_6b_0 & a_5b_0 & a_4b_0 & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 \\
a_7b_1 & a_7b_1 & a_7b_1 & a_7b_1 & a_7b_1 & a_7b_1 & a_7b_1 & a_7b_1 & a_6b_1 & a_5b_1 & a_4b_1 & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & 0 \\
a_7b_2 & a_7b_2 & a_7b_2 & a_7b_2 & a_7b_2 & a_7b_2 & a_7b_2 & a_6b_2 & a_5b_2 & a_4b_2 & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & 0 & 0 \\
a_7b_3 & a_7b_3 & a_7b_3 & a_7b_3 & a_7b_3 & a_7b_3 & a_6b_3 & a_5b_3 & a_4b_3 & a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 & 0 & 0 & 0 \\
a_7b_4 & a_7b_4 & a_7b_4 & a_7b_4 & a_7b_4 & a_6b_4 & a_5b_4 & a_4b_4 & a_3b_4 & a_2b_4 & a_1b_4 & a_0b_4 & 0 & 0 & 0 & 0 \\
a_7b_5 & a_7b_5 & a_7b_5 & a_7b_5 & a_6b_5 & a_5b_5 & a_4b_5 & a_3b_5 & a_2b_5 & a_1b_5 & a_0b_5 & 0 & 0 & 0 & 0 & 0 \\
a_7b_6 & a_7b_6 & a_7b_6 & a_6b_6 & a_5b_6 & a_4b_6 & a_3b_6 & a_2b_6 & a_1b_6 & a_0b_5 & 0 & 0 & 0 & 0 & 0 & 0 \\
\overline{a_7b_7} & \overline{a_7b_7} & \overline{a_6b_7} & \overline{a_5b_7} & \overline{a_4b_7} & \overline{a_3b_7} & \overline{a_2b_7} & \overline{a_1b_7} & \overline{a_0b_7} & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
\end{array}
$$

Figure 1. Bit Matrix for n=8

This implies that, for example in the desired case n=8, the rows of the bit matrix in Figure 1 sum to P. Note, however, that this simple example shows that we may rewrite (4) to index down the columns of the associated matrix:

$$P = 1 + \sum_{i=0}^{2n-1} \sum_{j=0}^{n-1} (a_{i-j} * b_j) 2^i \qquad (5)$$

where

$$a_i * b = \begin{cases} a_i b & i \neq n-1 \\ \overline{a_i b} & i = n-1 \end{cases}$$

$$a_i = \begin{cases} a_{n-1} & n \leq i \\ a_i \text{ in } A & 0 \leq i < n \\ 0 & i < 0. \end{cases}$$

This new column-sequential formulation is the basis of quasi-serial multiplication. If we let s_i be the sum of the ith column associated with (4), then we have

$$s_i = \sum_{j=0}^{n-1} a_{i-j} * b_j$$

and (5) simplifies finally to

$$P = 1 + \sum_{i=0}^{2n-1} s_i 2^i. \qquad (6)$$

If the product P is represented as

$$P = -p_{2n-1} 2^{2n-1} + \sum_{i=0}^{2n-2} p_i 2^i,$$

then algorithm MPY computes P <- AB from (6). Note that the variable "carries" accounts for the fact that column-sum s_i may exceed one bit and carry over into higher order columns. It is initialized to 1 to account for the (1 + ...) in (6).

```
MPY: DO
     carries <- 1
     DO FOR i = 0 TO 2n-1
          compute $s_i$
          $p_i$ <- ($s_i$ + carries) mod 2
          carries <- ($s_i$ + carries - $p_i$)/2
     END
END
```

The extension of MPY to an inner product algorithm is immediate. If we wish to compute P <- P + AB we simply include the bit p_i of P in the addition step for column-sum s_i; i.e., P is included as another row in the matrix of Figure 1. Algorithm IP shows this minor extension.

```
IP: DO
     carries <- 1
     DO FOR i = 0 TO 2n-1
          compute $s_i$
          $p_i^+$ <- ($s_i$ + carries + $p_i$) mod 2
          carries <- ($s_i$ + carries + $p_i$ - $p_i^+$)/2
          $p_i$ <- $p_i^+$
     END
END
```

Substitution of n=8 into algorithm IP formally defines the arithmetic used in our inner product realization.

4. Register-Transfer Design

Our goal at this level of the design hierarchy was to translate algorithm IP, derived previously, into a set of registers, logic elements, and timing signals. As shown in [4], the quasi-serial formulation of (6) has a simple realization with shift registers, gates, and adders; a block diagram is shown in Figure 2. The key point is the arrangement of shift registers and gates at the far left of Fig. 2. After A and B are loaded into the appropriate registers with the orientations shown, each right shift of the A register produces at the outputs of the NAND/AND gates the constituent bits of one column of the matrix of Fig. 1; in particular, the sum of these gate outputs after the ith shift is precisely s_i.

However, the right side of Fig. 2 differs slightly from previously described designs in which s_i is computed with a parallel counter [8], and the add/shift inner loop of algorithm MPY is implemented with an auxiliary register and fast adder. Our design uses a single, minimum-size adder tree to produce both one bit of P, the inner product being updated, and carries into the higher order columns of the bit matrix associated with the multiplication. (Actually, we have deviated slightly from algorithm IP in that carries are separated into two components: carries resulting from s_i, and a single carry from adding the newly formed product bit to the corresponding bit in the inner product P).

The main result of this phase of the design was an APL based register-transfer simulation of the structure in Fig. 2 which, assuming idealized timing, verified the functional correctness of the implementation.

Unfortunately, real timing is not so idealized. The Mead and Conway methodology recommends the use of a two-phase, non-overlapping clocking scheme. This approach eliminates many potential pitfalls, but must still be handled cautiously. One general caution arises because, unlike the innocuous black-box registers of Figure 2, real registers can be complex entities composed of several subunits. Further, a design decision was made to minimize the number of different "component" types by using just a single shift-register type to realize all required registers. These constraints forced us to evaluate in detail the design of any proposed clocking scheme. We attempted to answer the following two questions:

(1) Was the restriction to one register type feasible? Would it be easier just to design several different registers?

(2) What were the timing/synchronization problems introduced by the use of shift registers in the feedback paths to the adder in the implementation of Figure 2?

Our solution to these problems was to write an APL based timing simulator in which each register was represented as a composite of logic-gates and ideal-switches (e.g., inverters and pass transistors); the combinational logic of the adder tree was simulated at a high level to avoid unwanted detail. The results of this exercise showed that (1) no single register type would suffice, but two similar register types would work adequately, and (2) successful two-phase timing could be designed to accommodate the restricted register types and still avoid any synchronization problems. Thus, at least for idealized two-phase timing, we validated our register-transfer design.

5. VLSI Mask Design

Armed with the results of the register-transfer simulators, we were now ready to confront the final task of the project: the creation of a program in CalTech Intermediate Form (CIF) [1] to describe the masks necessary to fabricate the final VLSI chip. Much to the credit of the Mead and Conway approach, the transition from dealing with register-transfer building-blocks to transistors was quite painless; we simply learned a small set of design rules and some general structures for a transistor, a gate, a register, etc. With a little practice (and owing much to the pattern recognition capabilities of humans) design at the VLSI mask level became more and more manageable.

Writing the CIF program was approached in a top-down form like any other programming project. Roughly speaking, we divided the problem into several subproblems, wrote a subroutine to handle each, and then wrote a main program to connect the partial results together. Because CIF deals with geometric primitives like boxes and polygons, these "subroutines" turn out to be the mask descriptions of the register transfer building blocks, e.g., registers and adders. In CIF, these are called symbols or cells; they can be defined once and then copied to all the physical locations on the chip at which they may be required. The "main" program locates all symbols and provides code for how to connect them together.

Much to our delight, our conservative design required only nine symbols (excluding I/O pads supplied in a symbol library [9]). Of these, only three were designed from scratch; the remaining six were all registers isomorphic to one of the two types verified by the timing simulator of section 4. Of all symbols, the largest and most complex was the carry-save adder. It used an 18 transistor implementation of the logic equations

$$C_o = AB + (A + B)C_i$$

$$S = \overline{C_o}(A + B + C_i) + ABC_i.$$

Measured in the normalized CIF units of λ, the adder was $90\lambda \times 255\lambda$ which is large in comparison to the rest of the chip; this was the result of very conservative electrical design for large drive capability. Actually, the value $\lambda = 4\mu$ chosen for all projects in the fabrication run was itself conservative. Our minimum feature size will be 8μ as compared with the more typical 3-4μ for NMOS in commercial implementations.

With all cells designed, the final step in the design process was to complete a hand-layout for the entire chip, and then complete the CIF code to realize that layout. By layout we mean deciding exactly where all symbols are located and how they are connected. This required primarily patience, a modicum of cleverness for cell placement and routing, and an extravagant amount of graph paper. Coding in CIF proved straightforward although arduous. The reason is that CIF is a low level mask specification: it deals extensively with absolute coordinates on the chip, and with simple geometric constructs. Writing CIF is reminiscent of writing machine language; unfortunately, it is analagously tedius to read, write and debug. Lacking interactive design aids, (e.g., graphics) our approach was to translate the entire layout into CIF, and then debug with the aid of checkplots generated from the CIF. After about six passes through the debug/plot cycle, we converged to a final design. The final chip measured $1100\lambda \times 1280\lambda$ which translates to 4.4mm x 5.12mm. The layout is shown, along with a general floorplan, in Figure 3. Table 2 sums up some of the pertinent statistics relating to this phase of this design.

We should mention in closing that the use of design-rule check and static-check software [10] and an MOS logic simulator [11] developed at MIT proved useful in diagnosing possibly harmful design-rule violations and connectivity problems in the individual cells and in the entire design.

6. Conclusions

We have presented a case study of the implementation of a simple VLSI inner product machine. The basic goals for the project were

(1) to implement a function of moderate complexity;

(2) to complete the design rapidly;

(3) to produce a simple VLSI implementation.

We hope that we have demonstrated that, with a structured methodology such as that of Mead and Conway, one can quickly and effectively design nontrivial systems.

Table 2. Vital Statistics

Process:	NMOS, $\lambda = 4\mu$
Size:	1100 x 1280 = 4.4mm x 5.12mm
Pinout:	38 pads -- A0-A7, B0-B7 inputs; P0-P15 outputs; MCLR, LD control; ø1, ø2 timing; VDD, GND power.
Cells:	40 registers 38 pads 9 gates 8 adders 2 misc.
Active Devices:	503 transistors, 403 circuit nodes
Design Time:	5-6 man-weeks

The potential impact of such a methodology on the implementation of computer arithmetic is significant. Consider, for instance, the ability to design a large arithmetic structure with VLSI components that fit naturally and precisely into the designer's hierarchical view of the system. VLSI may unburden designers from the constraints and compromises that often arise from the unavailability of highly specialized off-the-shelf components. We feel that the application of VLSI technology to arithmetic problems per se merits further study. Indeed, shortly after we completed the mask designs for this project, Mead reported on similar bit-serial approaches to inner products for VLSI [12].

At the time of this writing, our design is being delivered to General Motors Research Laboratories for fabrication using an electron-beam mask generator. We are confident that we can correct any errors found in the prototypes to be returned to us later this year. We believe that larger operands, perhaps 16-bits, could also be accommodated using a quasi-serial structure, an appropriately scaled-down λ, and some pin multiplexing.

Acknowledgments

We would like to thank Professor R. J. Lomax for his tireless efforts in support of this project. We greatly appreciate the cooperation and generosity of General Motors Research Laboratories, and in particular we wish to thank E. F. Weller Jr. and J. W. Hile for their support.

References

[1] C. Mead and L. Conway, Introduction to VLSI Systems. Reading: Addison-Wesley, 1980.

[2] B. K. Gilbert, L. M. Krueger, E. E. Swartzlander Jr., D. E. Atkins, and E. L. Ritman, "Application of optimized parallel processing digital computers and numerical approximation methods to ultra high-speed three dimensional reconstruction of the intact thorax," Biomedical Computing, vol. 10, pp. 317-329, 1979.

[3] B. K. Gilbert, A. Chu, D. E. Atkins, E. E. Swartzlander Jr., and E. L. Ritman, "Ultra high-speed transaxial image reconstruction of the heart, lungs, and circulation via numerical approximation methods and optimized processor architecture," Computers and Biomedical Research, vol. 12, no. 1, pp. 17-38, Feb. 1979.

[4] T. G. McDonald and R. K Guha, "The two's complement quasi-serial multiplier," IEEE Trans. Comput., vol. C-24, pp. 1233-1235, Dec. 1975.

[5] R. L. Rivest, "A description of a single chip implementation of the RSA cipher," LAMBDA, Vol. I., No. 3, pp. 14-18, Fourth Quarter 1980.

[6] E. E. Swartzlander, Jr., "The quasi serial multiplier," IEEE Trans. Comput., vol. C-22, pp.317-321, Apr. 1973.

[7] E. E. Swartzlander, Jr., B. K. Gilbert, and I. S. Reed, "Inner product computers," IEEE Trans. Comput., vol. C-27, no. 1, pp. 21-31, Jan. 1978.

[8] E. E. Swartzlander, Jr., "Parallel counters," IEEE Trans. Comput., vol C-22, pp. 1021-1024, Nov. 1973.

[9] R. W. Hon and C. H. Sequin, A Guide to LSI Implementation, 2nd ed. Xerox Palo Alto Research Center, Jan. 1980.

[10] C. M. Baker and C. Terman, "Tools for verifying integrated circuit designs," LAMBDA, Vol. I., No. 3, pp. 22-30, Fourth Quarter 1980.

[11] R. E. Bryant, "An algorithm for MOS logic simulation," LAMBDA, Vol. I., No. 3, pp. 22-30, Fourth Quarter 1980.

[12] M. R. Buric and C. A. Mead, "Bit-serial inner product processors in VLSI," Proc. Second Caltech Conference on Very Large Scale Integration, 1981.

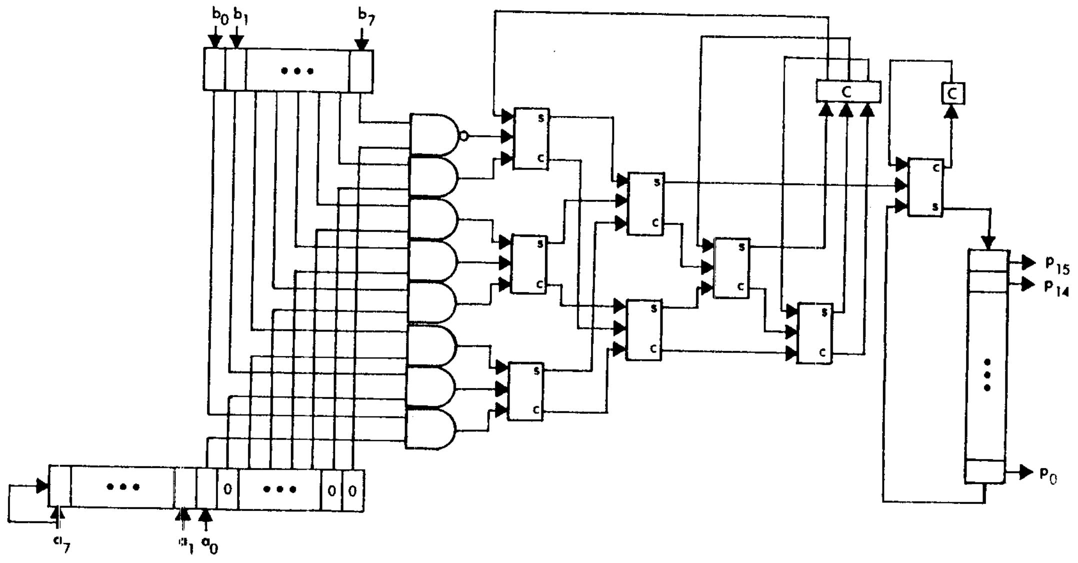
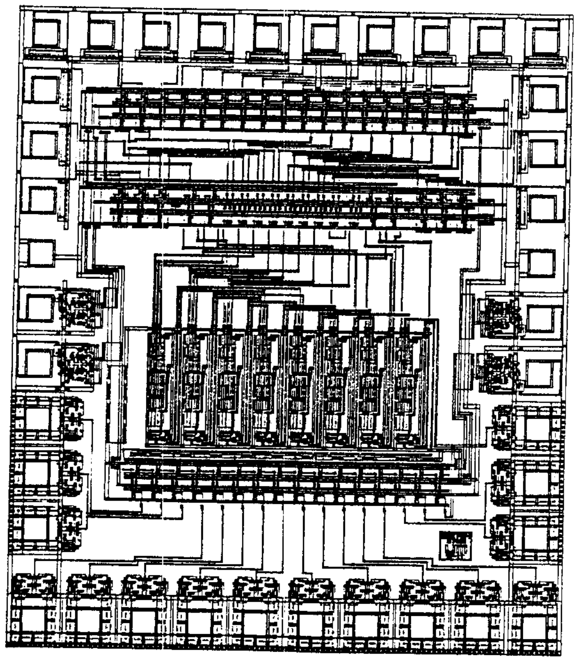
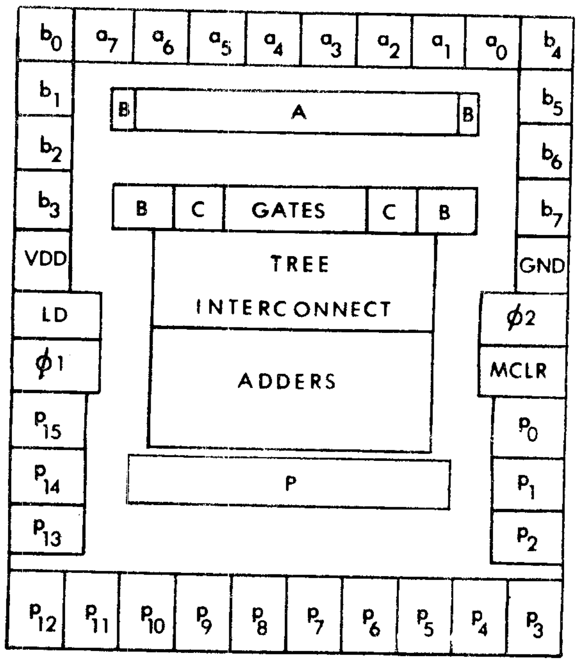


Figure 2. System Block Diagram

(a)

(b)

Figure 3. (a) VLSI Mask from CIF Specification, (b) General Chip Floor-plan