DESIGN AND NMOS IMPLEMENTATION OF PARALLEL PIPELINED MULTIPLIER

A Thesis Presented to
The Faculty of the College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
Chao-Wu Chen
June, 1988

Ohio University Library
TABLE OF CONTENTS

I. INTRODUCTION .................................................. 1
   A. Multiplier in Digital Signal Processing [DSP] ......................... 2
   B. Existing Multipliers ........................................... 5
      1. Modified Booth's Multiplier .................................. 6
      2. Parallel Array Multiplier ................................... 8
      3. Wallace Type Multiplier ................................... 10
      4. Other Multipliers .......................................... 12

II. DESCRIPTION OF PROPOSED MULTIPLICATION .................... 15
   A. Algorithm - Iterative and Pipelined Process ..................... 15
   B. Combinational Counter [CCT] .................................. 18

III. DESIGN METHODOLOGY ........................................... 22
   A. NMOS Circuit Basic Concepts .................................. 22
   B. Design and Simulation Tools ................................... 26
      1. Layout Tools .............................................. 26
      2. Physical Verification Tools ................................ 27
      3. Behavioral Verification Tools .............................. 27

IV. THE DESIGN CONSIDERATIONS .................................. 30
   A. System Considerations ....................................... 30
      1. Three pipelining Stages ................................. 30
      2. Floor Plan .............................................. 31
CHAPTER I INTRODUCTION

Since the advent of VLSI and the progress made in CAE, multiple cellular design has been the main method in digital hardware design [13]. Multiple usage of the same subcircuits to achieve greater system performance becomes feasible and cost effective [4]. Many parallel algorithms have been developed under this design trend. Digital hardware multipliers are no exception. A number of parallel multiplication algorithms have been presented during the last two decades.

The objective of this thesis is to present the design and implementation of a parallel pipelined multiplier, that is well suited for digital signal processing applications such as the finite impulse response (FIR) digital filters and matrix-vector multiplication [13]. The type of application requires fast signal multiplication within reasonable limits on the area. Our goal will be to design a compact layout with the main emphasis on the processing speed.

In the following sections of this chapter, two types of digital filters and several existing multipliers are examined. Chapter II introduces the multiplication algorithm used. Chapters III & IV present the design methodologies and implementation considerations. Actual physical layouts and simulation reports are revealed in chapter V. In the last chapter, discussion and future modification are given.
A. Multiplier in Digital Signal Processing [DSP]:

The hardware multiplier is fundamental and is the most important building block used in DSP [6]. Because the speed of the multiplier is much lower than that of the adder, the performance of DSP is restrained by the multiplier. In other words, the bottleneck in DSP performance is the propagation delay of the multiplier. Thus speeding up the multiplier becomes one of the major objectives in DSP. The aim of this thesis is to present a pipelined multiplier to improve the performance of FIR filters in DSP. To understand why pipelining techniques are suitable for FIR digital filter we refer to organization of the basic structures of digital filters.

There are two categories of digital filters: one is the finite impulse response (FIR) filter, and the other is the infinite impulse response (IIR) filter. Fig. 1.1 illustrates the structure of the FIR filter, which has the transfer function of:

\[ H(Z) = \sum_{k=0}^{n} b_k z^{-k} \]

Fig. 1.2 shows the structure of the IIR digital filter with transfer function [2], [18]

\[ H(Z) = \sum_{k=0}^{n-1} b_k z^{-k} / (1 + \sum_{k=0}^{m-1} a_k z^{-k}) \]

Advantages and disadvantages of these two digital filters are summarized in Table 1.1 [2].
Fig. 1.1 Nonrecursive (FIR) digital filter

Fig. 1.2 Recursive (IIR) digital filter
### Comparison of FIR & IIR Digital Filters

<table>
<thead>
<tr>
<th>NO.</th>
<th>Property</th>
<th>FIR filters</th>
<th>IIR filters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>analog equivalent</td>
<td>no meaningful analog filter</td>
<td>linked to analog filter via bilinear transformation</td>
</tr>
<tr>
<td>2</td>
<td>phase linearity</td>
<td>linear phase</td>
<td>non-linear</td>
</tr>
<tr>
<td>3</td>
<td>stability</td>
<td>inherently stable</td>
<td>must be designed with great care</td>
</tr>
<tr>
<td>4</td>
<td>adaptivity</td>
<td>well suited</td>
<td>not well suited</td>
</tr>
<tr>
<td>5</td>
<td>sensitivity to coefficients</td>
<td>low</td>
<td>high</td>
</tr>
<tr>
<td>6</td>
<td>speed</td>
<td>low</td>
<td>high</td>
</tr>
<tr>
<td>7</td>
<td>stages</td>
<td>longer</td>
<td>shorter</td>
</tr>
</tbody>
</table>

Table - 1.1

Because of its inherent stability, linearity and adaptivity, the FIR filter is now and will continue to be important [17]. Because the FIR filter has vector input and no feedback, the multiplication in the FIR filter can be realized in the pipelining process to achieve higher speed. The objective of this thesis is to design and implement a parallel pipelined multiplier suited for FIR filters and vector operations.
B. Existing Multipliers:

There are usually two cost functions, AT and AT^2, used to evaluate the performance of VLSI algorithms. A represents the silicon area or the hardware cost. T is the operation propagation delay. Better VLSI algorithm leads to lower order cost function. Although multiple cellular design is the trend of VLSI design, it is not always the most optimal solution, in terms of cost function AT. For example with this cost function, the conventional shift-and-add multiplier is the best of all hardware multipliers [7]. Thus AT^2 is useful if one wants to emphasize the importance of both timing at the expense of silicon area. In following subsections, both cost functions will be used to appraise the performance of different parallel multipliers.

Two kinds of hardware multipliers are the most commonly used. The first kind is a standard recoded multiplier, which uses a string recoded scheme to reduce the needed additions and is often in semi-parallel form. One such example is the multiplier in the IBM/360 Model 91 [1]. The second kind is an iterative cellular array multiplier which makes use of parallelism to achieve higher speed [4]. A modified Booth's multiplier belongs to the first kind; the parallel array and the Wallace type multiplier are the second kind of multipliers. Furthermore, two table look-up multipliers are also discussed in less detail.
1. Modified Booth's Multiplier:

The modified Booth's multiplier uses a string recoded scheme to reduce the needed additions. This recoded scheme is based on the fact that a string of consecutive 1's can be replaced by a 1 and a minus 1. For example, 1111 = 8 + 4 + 2 + 1 = 15 = 16 - 1 = 1000\_2, where minus 1 is noted as $\_$. A multiple scanning rule derived from the string recoded scheme is shown in Table 1.2. This multiple scanning rule reduces the additions by half and doesn't depend on the pattern of input string [4] [11] [12].

<table>
<thead>
<tr>
<th>Multiplier bits $X_{i+1} \times X_{i-1}$</th>
<th>Multiples of multiplicands to be added</th>
<th>Reasoning by string property</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0</td>
<td>no string</td>
</tr>
<tr>
<td>0 0 1</td>
<td>2</td>
<td>end of string</td>
</tr>
<tr>
<td>0 1 0</td>
<td>2</td>
<td>isolated 1</td>
</tr>
<tr>
<td>0 1 1</td>
<td>4</td>
<td>end of string</td>
</tr>
<tr>
<td>1 0 0</td>
<td>-4</td>
<td>beginning of string</td>
</tr>
<tr>
<td>1 0 1</td>
<td>-2</td>
<td>beginning and end of string</td>
</tr>
<tr>
<td>1 1 0</td>
<td>-2</td>
<td>beginning of string</td>
</tr>
<tr>
<td>1 1 1</td>
<td>0</td>
<td>center of string</td>
</tr>
</tbody>
</table>

Table - 1.2

One possible implementation of such a multiplier is illustrated in Fig. 1.3. This implementation is semi-parallel in form, thus the time delay depends mainly on how many modules are built in. If the number of modules is $m$,
Fig. 1.3 Schematic logic diagram of modified Booth's multiplier.
then for an N-bit multiplier, \((m+1)N\) full adders are needed; due to one-bit of \(m\)-input addition can be realized by \((m-1)\) bits full adders and there are two shift-back partial products in such multiplier. And the delay time is of the order \((N^2/2m)\), if the CSA (carry save adder) delay is neglected since it is small compared with the CPA (carry propagate adder) delay. Therefore, cost functions \(AT\) and \(AT^2\) are \(O((m+1)N^3/2m)\) and \(O((m+1)N^5/4m^2)\), respectively. Wiring is another aspect which should be taken into account. Since \(mN\) signals should be merged and the required wiring area is increasing in proportion to \(mN^2\). Therefore it is not recommended to implement the desired parallel multiplier in this structure. Thus modified Booth's multiplier is not suitable for the proposed parallel pipelined multiplier.

2. Parallel Array Multiplier:

A parallel array multiplier is the most straightforward iterative cellular multiplier. Its algorithm is directly derived from the matrix shape of binary number multiplication. An implementation block diagram of a 4-bit parallel array multiplier is shown in fig. 1.4. Each processing element contains one AND gate and one full adder. The multiplier and the multiplicand bits are passed horizontally and diagonally along the dotted lines. Summands and carries are transferred in the vertical and the diagonal directions.
Fig. 1.4 The logic diagram of The Parallel array multiplier.
along the solid lines respectively [4] [16]. In this 4 bit case, the worst case delay is about 9 adder delay [16] and 16 PEs are needed. Therefore cost function $AT$ is $O(N^2(2N+1))$ and $AT^2$ is $O(N^2(2N+1)^2)$. The design of such a multiplier can be very compact and allows for expansion vertically or horizontally as desired. Braun [4] replaced the bottom line adders with a look-ahead carry adder. The multiplier's main defect is that the delay time is in proportion to order $N$ and it is also not suitable for pipelining structure. Because of these two defects the parallel array multiplier is also not suitable for the proposed pipelined multiplier.

3. Wallace Type Multiplier: [4] [14] [17]

The Wallace Type Multiplier is the multiplier which uses Wallace trees to realize multiple input addition. A Wallace tree is the most efficient connection of multiple input addition. A 7-input Wallace tree, designated as $(7,3)$. is shown in Fig. 1.5 (a). Different weighted multiple-input additions can also be formed by Wallace trees. Fig. 1.5 (b) illustrates a two-bit three-input Wallace tree, shown as $(3,3,4)$. These Wallace trees can also be called counters. For example a 7-input Wallace tree can be called a $(7,3)$ counter.

Two Wallace-type multipliers will be discussed next. Both use 4-bit multipliers as the basic iterative cells.
Fig. 1.5 (7,3) and (3,3,4) Wallace trees
First, an 8-bit Wallace multiplier which requires eight (3.2) counters and one 12-bit CPA is illustrated in Fig. 1.6 (a). In Fig. 1.6 (b) a 16-bit Wallace multiplier is shown. In such a 16-bit multiplier, one 28-bit CPA and eight (7.3), eight (5,3), and sixteen (3,2) counters are needed. Altogether, a 16-bit Wallace multiplier needs 108 full adders and delay time equals four times (3,2) counter delay plus one 28-bit CPA delay. However, the related interconnection network involves connections of 224 signals and the merging of 120 lines. So the interconnections become a critical issue while designing a big multiplier.

4. Other multipliers:

There are two table look-up multipliers. The first one is a ROM multiplier which simply uses memory to store the products and treats multiplier and multiplicand bits as address bits to select the desired product. It can be very fast because it requires only one memory access time; however, it needs too many hardware investments. For instance, to implement an eight-bit multiplier, 128K bytes of memory are required [4]. The second table look-up multiplier is a logarithm multiplier, which uses the addition property of the logarithmic function to perform the multiplication. It requires one addition time and two memory access times to perform one multiplication. The required
Fig. 1.6 8-bit and 16-bit multipliers formed by 4-bit multiplier slice.
memory depends on the product size and product accuracy. For example, with an 0.3 percent accuracy, an 8-bit multiplier needs 22016 bits of memory to generate an 11-bit product. Although the logarithm multiplier saves huge memory, it is limited to those application requiring less accuracy [3] [4]. Thus it is also not appropriate for the proposed multiplier.

After reviewing these existing multipliers, one may conclude that most of them are not suitable for implementing a fast pipelined multiplier. Modified Booth needs too large a wiring area, look-up table multipliers require too many memories to be accurate enough. Although parallel array multipliers have simple interconnections, they have the largest delay and cannot be executed in pipelined structure to achieve higher speed. Thus, only one multiplication scheme, the eight-bit Wallace type iterative scheme, is left. The proposed multiplier uses such an iterative scheme to build the pipelined stages and therefore reduces delay time to the range of addition delay. The detailed proposed multiplication algorithm is presented in Chapter II.
The proposed multiplication algorithm is described in this chapter. As mentioned in Chapter I, the proposed iterative multiplication algorithm is similar to the 8-bit Wallace multiplier. A detailed description of this iterative scheme and of the pipelining structure is presented in section A. Section B contains the discussion of a combinatorial counter (CCT) introduced to achieve higher speed.

A. Algorithm - Iterative and Pipelined Process

For a fast hardware multiplier, there are two requirements. The first one is to generate all the needed summands or partial products at once, and the second is to follow summing all of these summands or partial products in a certain optimal way [7]. After describing the proposed algorithm, it will be known whether the proposed multiplication meets these two requirements and as well as what the potential is for being the fastest hardware multiplier.

The proposed iterative multiplication scheme is very well known in assembly programming and is illustrated in Fig. 2.1. This iterative scheme is an N-bit multiplication that can be formed by combining four (N/2)-bit multiplications. Successively using this iterative scheme, a full multiplication can be realized. For example, an 16-bit
Fig. 2.1
The scheme of multiplication expansion.

Fig. 2.2 The connections of Wallace and purposed multipliers
multiplication can be divided into 4 iterative steps if 2-bit multipliers are used as the final steps. One possible design of such an iterative scheme is shown in Fig.2.2. Each iterative scheme contains four submultipliers, N-bit (3.2) counters and (3N/2)-bit CPA.

As the figure shows, data flow is in one direction, and hence each iteration is independent of the other. This is the necessary condition for the pipelining process. Therefore, if registers are inserted between some iterative steps, the pipelining structure is constructed. The longest delay is the final stage delay (3N/2-bit CPA delay). In the next section, one modification (CCT) will be introduced which reduces the longest stage delay to N-bit CPA delay. Thus the multiplication delay is reduced to the order of the addition delay. Further, if a lookahead carry adder is used, the N-bit multiplication delay will be of the order (logN).

To implement a (logN)th stage iterative step, 2N full adders are needed. Therefore, the whole N-bit multiplier requires 2N(N-1) adders. Thus, the cost function AT and AT^2 can be 2N(N-1)logN and 2N(N-1)log^2N, respectively.

Although it is not the best performance multiplication algorithm, it has the potential to be the fastest multiplier. The only disadvantage of this algorithm is the rather large silicon area consumption. However, it is the essence of parallelism to achieve timing performance at the expense of hardware cost.
B. Combinational Counter (CCT):

The combinational counter is introduced to replace the upper \((N/2)\)-bit CPA which is used for passing carries. Such carries can only be of three values: 0, 1, or 2 as the result of the three-input addition needed in the discussed iterative scheme. Therefore, if a device is able to up-count 0, 1, or 2, then this device can substitute the upper \((N/2)\)-bit CPA. Furthermore, if the device accomplishes this operation while \((3,2)\) and \(N\)-bit CPA are processing, then one-third of the original time can be saved.

The possible function diagram of CCT is shown in Fig.2.3a. The add-two and add-one blocks are used to up-count two and one and the multiplexer is used to select the actual final result. This selection is made after the carries are generated by \((3,2)\) counters and \(N\)-bit CPA. Therefore, for each iteration, only one \((3,2)\) counter delay and \(N\)-bit CPA delay plus one multiplexer delay are needed. However, such a function is too complicated to be implemented. Another approach is illustrated in fig. 2.3b which uses two up-count-1 CCT to perform the desired function. Although it is easier to implement, the propagation delay is twice the sum of up-count-1 and multiplexer delay.

According to these two approaches, it would not be reasonable to make such modification. However, if carefully examining the carries at every bit position, one can find only one-bit-carry is needed in the above position \((3N/2)+1\).
Fig. 2.3 Two function diagrams of CCT.
Therefore, only one up-count-1 CCT is required to perform this task, and it also guarantees the CCT delay will be less than the sum of the (3,2) and the N-bit CPA delay. Hence, the iteration step delay is only the above sum and one multiplexer delay. The full iterative scheme is illustrated in Fig. 2.4. Design and implementation details will be discussed in chapter IV and V.
Fig. 2.4 The proposed iterative scheme.
CHAPTER III DESIGN METHODOLOGY

The purpose of this chapter is to introduce the background material for implementing the proposed multiplier. Section A describes the basic NMOS circuits concepts. Some basic principles and electrical properties of NMOS, such as pass transistors, ratioed logic, restoration of signal and problem of fan in and fan out, will be discussed. Section B introduces the layout design tools and the simulation tools. They include LED, DRC, EXTRACT, ESIM, CRYSTAL, SPICE, and POWEST.

A. NMOS Circuit Basic Concepts:

The advancement of CAE has made the fabrication and the design house more independent of each other. Mead and Conway [8] presented a full set of design rules for NMOS implementation as the interface between the design and the fabrication. The following description of basic NMOS circuit concepts are based on the material presented in Introduction to VLSI System [8] by Mead & Conway and Basic VLSI Design: Principles and Application by Pucknell [10].

Unlike the TTL or other bipolar logic, the MOS circuits are voltage-controlled, hence, the MOS transistor can be used as the switch in series with line carrying signal in a similar manner as in the use of relay contact. Therefore
complicated functions can be constructed within a small silicon area. Such an MOS application is called pass transistor logic. Fig. 3.1 shows an MOS implementation of a four to one multiplexer. Such a multiplexer, if implemented by TTL, requires at least nine gates.

However, there is a disadvantage in employing a pass transistor in the NMOS circuit: the high level voltage, when passing a pass transistor, is reduced by a $V_T$ (threshold voltage of NMOS transistor). This problem needs to be solved by increasing the $k$ ratio of the next stage NMOS gates. This $k$ ratio is the ratio of pull up and pull down resistances of the NMOS gate. NMOS logic is a ratioed logic. For a normal NOR gate and inverter, the $k$ ratio is four.

Normally, the $k$ ratio is made in a geometrical way, that is, by widening the pull down transistor or lengthening the pull up transistor to increase such ratio. However, widening the pull down transistor not only reduces the resistance but increases the input capacitance, and hence the gate needs more time to be driven. Generally, if a gate has a large fan out or a large capacitive load, it is better to widen its pull down transistor.

As we mentioned before, MOS circuits are voltage controlled, that is there is no static current flowing into their inputs. The load of an NMOS circuit is purely capacitive. The equivalent circuit diagram just looks like a voltage divider driving a capacitor. Fig. 3.2 shows two
Fig. 3.1 Multiplexer implemented by pass transistor.
Fig. 3.2  NMOS inverters and equivalent circuit diagram.
inverters and their circuit equivalent diagram. As shown in the figure, the rise time and the fall time of NMOS circuit are not the same.

B. Design and Simulation Tools:

To design a large and complicated IC, it is essential to have computer aids so that the job can be completed within a reasonable time or, in some cases, can be completed at all. These computer design tools should include [10]:

1. Physical design layout and editing tools: They can use either textual or graphic input.

2. Physical verification tools: They must include design rule checker, circuit extractors, ratio rule and other static check.

3. Behavioral verification tools: They are the simulators of logic (switch level) function verification, timing behavior simulation and the power consumption estimation.

In our case, they are the layout editor (LED), DRC, and EXTRACT in VALID SCALDstar System. The simulation tools are the MEXTRA, ESIM, CRYSTAL, SPICE and POWEST in UNIX System. In the following paragraphs, brief description of these design tools will be presented.

1. Layout Tools: Layout Editor (LED)

LED is a graphic layout editor, which handles design
directories (files) hierarchically, and therefore multiple cellular design is very easy in this environment. It offers many other functions such as rotating, zoom in, cutting, pasting, ... and so on. Its layout can be easily transferred to a standard IC layout format the CIF (Caltech Intermediate Format) for further processing.

2. Physical Verification Tools: DRC / EXTRACT

DRC (Design Rule Checker) is the script available in VALID SCALDstar System for checking if there is any design rule violation. If there is, the DRC will mark the violation spot in the layout file for correction. EXTRACT also resides in VALID SCALDstar System. In our case, it combines parts of the design rule checking and circuit extracting. Usually, the circuit extractor is for extracting circuit information, such as network lists, transistors and capacitance values from the layout file, that is, preparing the input files for behavioral verification.

3. Behavioral Verification Tools:

In our case, the behavioral verification is done in the UNIX system. The simulators, in the UNIX system, don't accept the circuit files generated by EXTRACT, thus another circuit extractor, MEXTRA, is needed. MEXTRA accepts layout files in CIF format. Therefore the layout file, not the circuit files, is transferred to the UNIX system via a
serial port for simulation.

a. **MEXTRA** --- It is a circuit extractor and resides in the UNIX system. In our case, it is the actual circuit extractor, not EXTRACT. It reads the CIF layout file, and generates a .SIM file for simulation.

b. **ESIM** --- It is an "event-driven-switch" level simulator of NMOS transistor circuits used for the verification of combinational or sequential logic function. It can accept commands in interactive mode or from a command file, and reads .Sim file.

c. **CRYSTAL** --- It is also a semi-interactive program for analyzing the timing characteristics of large integrated circuits. As well as ESIM, it reads .sim file. It reports the worst case delay obtained under certain input patterns.

d. **SPICE** --- It is a general-purpose circuit simulation program for nonlinear dc, nonlinear transient, and linear ac analysis. Its transistor model can be modified as required. It does the circuit simulation work very well, but takes a lot of computing time to simulate. Therefore, it is only suitable for small circuits; for large circuits, it has to be replaced by ESIM and CRYSTAL.
e. POWEST --- It is a program which reads .sim file and estimates the power dissipation. It reports the typical and maximum power dissipation of the design.
CHAPTER IV THE DESIGN CONSIDERATIONS

In the following two chapters, the design of the proposed eight-bit multiplier is discussed. In this design, the 8-bit multiplication is divided into 8-bit by 8-bit, 4-bit by 4-bit, and 4-bit by 2-bit three iterative pipelined stages. It is enough to give a full picture of the proposed iterative multiplication scheme. This chapter is divided into two parts: Section A contains chip level consideration and system implementation while Section B contains function cells. The actual physical layouts and simulation results are presented in the next chapter.

A. System Considerations:

1. Three Pipelining Stages:

As mentioned above, this eight-bit case is divided into three stages and chooses 4-bit by 2-bit multipliers as its terminal stage. This choice is based on the consideration of hardware requirements which consists of the interconnections and the devices needed. For example, if 2-bit by 2-bit multipliers are used as the terminal stage, it not only increases the complexity of interconnections but also needs more adders. To implement a 4-bit by 4-bit multiplier in such a way, one needs two full iterative steps
and 16 adders plus about 20 gates. However, if it is implemented using 4-bit by 2-bit cells, only 12 adders and 20 gates are needed and the iterative scheme is much simpler. Another way to implement a 4-bit by 4-bit multiplier is through engaging the parallel array multiplier. Although, its connections are simple, its delay time is rather large and can even be larger than the 8-bit stage delay. Hence, the reason for using the 4-bit by 2-bit multiplier as the terminal stage is very clear.

The pipelined stages do not have to be the same as the iterative steps, although, in our case, they are the same. For instance, it is possible to combine 4 by 2, and 4 by 4, two iterative steps into only one pipelined stage. However, in our eight-bit case, this arrangement is not a proper solution due to the propagation delay which will be larger than the 8-bit stage delay for these two iterative steps. But if the desired implementation is a 16-bit case, then including 4-bit by 2-bit cells and 4-bit by 4-bit cells into one pipelined stage would be better.

2. Floor Plans:

There are two choices in decision of the floor plan of the iterative multiplication scheme. The first choice is a two dimensional one like the iterative scheme used by the 8-bit Wallace type multiplier (fig. 1.6). The second is one dimensional as illustrated in fig. 2.4. Next paragraph will
discuss how to make a selection between these two floor plans.

Fig. 4.1 is a floor plan of the proposed iterative scheme in the two dimensional form. The arrows show the directions in which the data flows. The four submultiplier directions and the iterative stage direction are perpendicular to each other which leads to the jumping of the Vdd and the Gnd plane. Furthermore, the interconnections of two dimensional iterative scheme is more complicated than that of the one dimensional and require more area. Fig 4.2 shows the floor plan of a one dimensional iterative scheme. In both figures, the shaded regions are the required wiring areas. After comparing these two figures, the reason why the two dimensional floor plan is not used will be very understandable. Actually, the layout of the two dimensional iterative scheme has been done unsuccessfully before and required about 30-50 percent more silicon area than the one dimensional floor plan.

3. Design Specifications:

The objective of this design is to produce a single chip parallel pipeline multiplier. Thus, chip essential signals such as "chip select" (CS) and "reset" (RES) should be included in the design. Two interleaved timing signals, "latched enable" (Le) and "output enable" (Oe), are used for pipelining control. However, as these two signals are both
Fig. 4.1
Two-dimension iterative scheme.

Fig. 4.2
One-dimension iterative scheme.
held high, the pipelining circuit will operate in combinational mode. It is designed to meet the non-vector type input multiplication. Therefore, the chip can operate both in pipelining and in combinational modes. The chip select (CS) is designed to control these two timing signals (Le & Oe) to govern the input and output data flow. The reset signal (RES) controls the channels between the input bus and the ground plane, and so as the reset becomes high, all of the inputs are pulled down to zero. Besides these four control signals, 16 output product bits, two 8-bit inputs, Gnd and Vdd are needed. Therefore, this eight-bit monolithic multiplier has 38 pins.

The final stage is an eight-bit iterative stage. It reads the inputs from the registers of the 4-bit by 4-bit multipliers as out enable (Oe) is active and passes them to the (3,2) counters, N-bit CPA, and combinational counter. Fig. 4.3 shows the function cells of this stage and their relative active timing diagram.

The intermediate stage is a four-bit by four-bit stage. Since it only has two submultiplier cells, it needs a 4-bit CPA and a two-bit CCT to perform the summation. Fig. 4.4 shows the function cells of this stage and its relative active timing diagram. Besides the above two cells, eight-bit noninverting registers are needed to store the eight-bit product.

The first stage is the 4-bit by 2-bit terminal stage;
Fig. 4.3 Eight-bit pipelined stages and its relative timing
Fig. 4.4 4-bit stage and its relative timing
not an iterative scheme, it is a 4-bit by 2-bit multiplier. Fig. 4.5 (a) illustrates the function diagram of this 4-bit by 2-bit multiplier which contains 8 AND gates used to generate summands of the multiplication and a 4-bit CPA used to perform the summing. The six-bit output is stored in a 6-bit noninverting register. For NMOS implementation, the AND gates are changed to NOR gates and the inputs are complemented. This modifications is shown in Fig. 4.5 (b). Therefore, inverting registers are needed to negate the inputs. Although, there are six inputs, if considering the inputs of two adjacent 4-bit by 2-bit multipliers, four inputs can be shared by them, thus, only 4 inverting input registers are needed in the 4-bit by 2-bit multiplier. Altogether, there are one 4-bit input inverting register, one 6-bit output noninverting register, 8 NOR gates and one 4-bit CPA are needed to implement this cell.

In the final layout, one 8-bit iterative cell, four 4-bit by 4-bit iterative cells and eight 4-bit by 2-bit multipliers are needed. Hence, any minor increases in length of a 4-bit by 2-bit cell causes an eight time length increase in the final layout. Therefore, the general rule in designing the 4-bit by 2-bit multiplier is to make it as short as possible. On the contrary, the rule for designing the third stage cell is to make it as thin as possible.
Fig. 4.5 Two function diagrams of 4-Bit by 2-Bit multiplier.
B. Function Cell Implementation:

According to the above section, 9 function cells are needed. They are an 8-bit (3,2) counter, an 8-bit CPA and a 3-bit CCT in final stage, a 4-bit CPA and a 2-bit CCT in intermediate stage, and an NOR gate array and a 4-bit CPA in the first stage, plus the noninverting register and the inverting register. All of them are described in the following subsections.

1. The Adders:

As mentioned above, there are three types of adders in this eight-bit design. They are the 8-bit CPA, the 4-bit CPA, and the 8-bit (3,2) counter. Because the 8-bit and the 4-bit CPAs can be implemented using the same functional cells, only two types of adder cells are needed.

a. (3,2) Counters:

The (3,2) counter is a one-bit full adder. The only difference between a (3,2) counter and a full adder is the three inputs of the (3,2) counter are present at the same time. Thus, there is no carry propagation delay and the delay of the N-bit (3,2) counter is the same as the one-bit (3,2) counter delay. The logic diagram of the (3,2) counter delay is shown in Fig. 4.6. The required function is implemented in the selective logic form. The input "C" is used to select the actual results. For example, as C is "0"
Fig. 4.6 Logic diagram of one-bit (3,2) counter.

Fig. 4.7 Carry generation logic diagram.
the carry-out function is AND of A, B and the sum bit, is XOR of A, B, or if C is "1", then the carry-out function becomes the OR of A, B and the sum bit becomes XNOR of A, B. The worst case delay of such a function cell is about three gate delays.

**b. Carry Propagate Adder (CPA):**

Unlike the (3,2) counter, the most important factor of CPA performance is the carry propagation delay. If the lookahead carry adder is used, the delay time will be of order log\(N\), however, in our eight-bit design, it is not necessary. First, the carry function is reviewed and shown as below:

\[
C_{i+1} = A_iB_i + (A_i+B_i)C_i
\]

\[
= A_iB_i + (A_i+B_i)(A_{i-1}B_{i-1} + (A_{i-1}+B_{i-1})C_{i-1})
\]

\[
= A_iB_i + (A_i+B_i)(A_{i-1}B_{i-1} + (\ldots(A_0B_0))\ldots).
\]

Therefore, the carry function can be shown as Fig. 4.7, and for NMOS implementation, the logic function is modified as shown in Fig. 4.8. The one-bit carry function is implemented by four NOR gates and the one-bit carry propagation delay is two NOR gate delays. The full-bit adder and the terminal adder (bit0) cells are illustrated in Fig. 4.9.
Fig. 4.8 Logic diagram of carry generator.

Fig. 4.9 The iterative cell and terminal cell of CPA.
2. The Combinational Counter (CCT):

There are two CCTs in this eight-bit case; one is a three-bit upcount-1 CCT and the other is a two-bit upcount-1 CCT. Considering upcount-1, we know that the least significant bit must be changed to its complement and higher bits depend on those bits below them. If all of the lower bits are 1, then this bit changes to its complement; otherwise it retains the same logic value. Fig. 4.10 illustrates the schematic diagram of a two-bit CCT. As mentioned in Chapter II, the carry-in is used to select the actual results. In our case, in order to be compatible with the CPA carry-out, the complement of carry-in is used as the select input. The XNOR gate is used to transport or complement the upcount 1 result. Instead of using the original input, the CCT used the upcount 1 result to decide whether to transport or to complement this certain bit. In this manner NOR gates, not NAND gates, are used to perform the decision logic. Fig. 4.11 illustrates the three-bit CCT implemented in such a way.

3. The NOR Gate Array and the Registers:

The NOR gate array is used with the inverting registers to generate all the summands of the multiplication. It is shown in fig. 4.12. Such a cell should be as short as possible, because 64 summands are needed next to the input registers in this 8-bit multiplier, so as to minimize the overall
Fig. 4.10
Two-bit CCT

Fig. 4.11
Three-bit CCT
Fig. 4.12  NOR gate array Cell

Fig. 4.13  Inverting register cell

Fig. 4.14  Non-inverting register cell
height of the design.

There are two register cells in this eight bit case; one is of the inverting type, and the other is noninverting. The first one is illustrated in Fig. 4.13. Two timing controls, "Le" & "Oe", are used to regulate the data flow. These two timing signals are two nonoverlapped waves, both governed by the CS (chip select). If the CS is not true, then both two timing signals will be low and there will be no data flow in the machine. The RES (reset) controls the channels between input bus and the Gnd plane, thus if RES is high then all the inputs are pulled down to zero. Because there are three stages in this machine, at least, three clocks are needed to reset the whole multiplier.

The noninverting register is shown in Fig. 4.14. It is simpler than the inverting one and uses the same two timing signals. The two register cells use dynamic latches to store or maintain signal and have very simple structures (two or three shift registers), hence the register delay is very small compared to the stage operation delay. Therefore, the two timing signals (Le & Oe) should be asymmetrical to prevent waste of timing performance, that is, the duty cycle of Le would better be smaller than that of Oe to achieve higher speed.
CHAPTER V SIMULATION RESULTS

The simulation results and actual physical layouts are presented in this chapter. This chapter is divided into two parts. The first part contains the simulation results of each functional cell. The second part contains the stage level simulation results and layouts.

A. Functional Cells Simulation Results:

The basic functional cells used include a 4-bit CPA, a (3,2) counter, a 2-bit CCT, a NOR gate array, and registers. The simulation reports of each functional cell are divided into two portions, the first being the function description and the behavioral simulation report, and the second being its physical layout. In the first portion, the layout size, interfacing properties, brief function description, timing simulation result, and the power dissipation information are presented. The interfacing properties contains the information of the driving ability and the easiness of being driven. This information is the output equivalent resistance and the input capacitance in the NMOS implementation. The timing simulation report and the power dissipation information are obtained from the simulators of CRYSTAL and POWEST respectively. The actual copy of these two simulation reports and that of ESIM (function verification simulator)
are to be founded in the Appendix.
NAME : LAD4

SIZE : 210 * 121

INTERFACING PROPERTIES :

A0, B0 : 4 Cg * k=8
A1-A3, B1-B3 : 6 Cg k = 4 (nonswitched)
S0 : 2R/ 1/2 R ** (switched) ***
S1-S3 : 4R/ R (switched)
C4 : 2R/ 1/2 R (nonswitched)

FUNCTION DESCRIPTION :

The LAD4 is a 4-bit carry propagate adder. It accepts nonswitched input signals and produces a 4-bit sum and one inverting carry out. The carry propagation travels along the second NOR gates column. Thus, for each carry bit, only two NOR gate delays are required. The sum bit is generated in a selective method, if Cin is high XNOR of A and B is selected, otherwise XOR of A and B is selected.

WORST CASE DELAY : 6.2 ns

POWER DISSIPATION : 7.2 mW (avg.)
10.5 mW (max.)
* $C_g$ is the unit gate to channel capacitance. (input capacitance)

** $R$ is the equivalent transistor resistance. (output resistance)

*** The "switched" and "nonswitched" signals mean the high voltage level is degraded or not, respectively.

There is an 8-bit CPA, which uses the same cell as lad4 uses, in the third stage. Thus the simulation report of this type of cell is omitted in this chapter.
Fig. 5.1 Layout of LAD4.
Fig. 5.2 Layout of LAD4.
NAME: PAT

SIZE: 157.5 * 75

INTERFACING PROPERTIES:
A, B, C: 6 Cg K = 4
00, 01: 2R/ 1/2 R (nonswitched)

FUNCTION DESCRIPTION:
The PAT is a one-bit full adder which is used as the cell of (3,2) counter (CSA). It accepts nonswitched inputs and produces nonswitched outputs. It is implemented in a selective manner to simplify the required function.

WORST CASE DELAY: 2.69 ns

POWER DISSIPATION:
1.8 mW (avg.)
2.8 mW (max.)
NAME: CCT2

SIZE: 72 * 54

INTERFACING PROPERTIES:

<table>
<thead>
<tr>
<th>Input</th>
<th>Property</th>
<th>k Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>I0</td>
<td>2 Cg</td>
<td>k = 4</td>
</tr>
<tr>
<td>I1</td>
<td>1 Cg</td>
<td>K = 4</td>
</tr>
<tr>
<td>C</td>
<td>4 Cg</td>
<td>k = 4</td>
</tr>
<tr>
<td>00</td>
<td>2R/ 1/2 R (switched)</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>4R/ R    (switched)</td>
<td></td>
</tr>
</tbody>
</table>

FUNCTION DESCRIPTION:

The CCT2 is a 2-bit combinational counter. It accepts nonswitched input signals and produces switched outputs. It is able to transmit inputs to the output port or up-count-one of the inputs to the outputs. This function selection is made by the input which is produced by the carry propagate adder of the second stage.

WORST CASE DELAY:

- 0.89 ns (from I0 to O1)
- 0.9 ns (MUX delay)

POWER DISSIPATION:

- 0.86 mW (avg.)
- 1.30 mW (max.)
Fig. 5.4 Layouts of CCTs.

Layout of CCT2

Layout of CCT3
NAME: CCT3

SIZE: 114.5 * 61

INTERFACING PROPERTIES:

<table>
<thead>
<tr>
<th>I0</th>
<th>4 Cg</th>
<th>k = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>3 Cg</td>
<td>k = 4</td>
</tr>
<tr>
<td>I2</td>
<td>1 Cg</td>
<td>k = 4</td>
</tr>
<tr>
<td>C</td>
<td>5 Cg</td>
<td></td>
</tr>
<tr>
<td>O0</td>
<td>2R</td>
<td>(switched)</td>
</tr>
<tr>
<td>O1</td>
<td>4R</td>
<td>(switched)</td>
</tr>
<tr>
<td>O2</td>
<td>4R</td>
<td>(switched)</td>
</tr>
</tbody>
</table>

FUNCTION DESCRIPTION:

The CCT3 is a three-bit combinational counter and is used in the third pipelined stage. It accepts nonswitched inputs and produces switched outputs. Its function is similar to that of the CCT2.

WORST CASE DELAY:
- 2.32 ns (from I0 to O2)
- 1.87 ns (MUX delay)

POWER DISSIPATION:
- 1.0 mW (avg.)
- 1.4 mW (max.)
NAME : INP

SIZE : 52 * 50

INTERFACING PROPERTIES :

RES : 1 Cg (control signal)
Le : 1 Cg (control signal)
Oe : 1 Cg (control signal)
i/p : 2 Cg k = 8 (switched)
o/p : 2R/ 1/2 R (nonswitched)

FUNCTION DESCRIPTION :

The INP is an one-bit inverting register with reset control used in the first stage. The INP accepts switched input and produces nonswitched output. Le and Oe are the two timing signals to latch or to enable the data flow. RES is the reset signal of the whole multiplier, and as RES is high, all of the inputs are reset to zero.

WORST CASE DELAY : 2.35 ns
(from i/p to o/p as LE and OE are both high)

POWER DISSIPATION : 1.7 mW (avg.)
0.9 mW (max.)
Layout of INP
(inverting register)

Layout of NONREG
(noninverting register)
NAME : NONREG

SIZE : 31 * 43

INTERFACING PROPERTIES :

OE, LE : 1 Cg (control signal)
i/p : 2 Cg (switched)
o/p : 2R/ 1/2 R (nonswitched)

FUNCTION DESCRIPTION :

The NONREG is an one-bit noninverting register and is used in the 2nd and 3rd stages. It is controlled by the same timing signals as the INP.

WORST CASE DELAY : 1.96 nS

POWER DISSIPATION : 0.33 mW (avg.)
0.55 mW (max.)
B. Stage Simulation Results:

As mentioned earlier, there are three pipelined stages in this design. The first and the second stages perform the function of a 4-bit multiplier and generate an 8-bit product. In the design, the above mentioned function contains two register latching, two 4-bit CPA operation, one NOR gate, and one 2-bit CCT operation. On the other hand, the third stage performs the summing of 8-bit iterative step which contains one (3,2) counter operation, one register latching, one 8-bit CPA delay, and one 3-bit CCT operation. The longest delay is the third stage, therefore the pipelining clock rate is determined by the final stage delay. In the following subsections, two simulation reports will be presented. The first reports is for the first two stages, and the second report is for the third stage.
NAME: 4B4

SIZE: 422 * 510

INTERFACING PROPERTIES:

A0-A3: k = 8 (switched)
B0-B3: k = 8 (switched)
LE, OE, RES: (control signal)
P0-P7: (switched)

FUNCTION DESCRIPTION:

The 4B4 cell is a four-bit by four-bit multiplier which has two pipelined stages. The first is an four-bit by two-bit multiplier, which generates a six-bit product which is then stored in the second stage register array. The second stage performs the remaining summation to produce the eight-bit product. The INP and NOR gate array in the first stage are used for generating 16 summands of this four-bit multiplication. The CCT2 is used to pass the carry to the most significant two bits. The two LAD4 (CPA) is for the required summation in each stage.
WORST CASE DELAY: 41.7 ns

The worst case delay is measured from feeding the LSB input to the cell until the appearance of the MSB product at the output terminals. During the timing simulation, both timing controls (Le & Oe) are maintained high. Therefore, this delay is the combinational delay of 4B4 (not the single stage delay).

POWER DISSIPATION: 49 mW (avg.)
71 mW (max.)
Fig. 5.6 Layout of 4B4.
Fig. 5.7 Layout of 4B4.
NAME : 8B8

SIZE : 1688 * 735

INTERFACING PROPERTIES :

A0-A7 : k = 8 (switched)
B0-B7 : k = 8 (switched)
LE, OE, RES : (control signal)
P0-P15 : (switched)

FUNCTION DESCRIPTION :

The 8B8 is a three stage 8-bit multiplier which uses four 4B4 cells as its first two stages. The third stage contains a 32-bit register array, an eight-bit (3,2) counter array (PAT), an eight-bit CPA, and a three-bit CCT (CCT3). The 8-bit CPA is constructed by rotating the one-bit cell of LAD4. Therefore, the summing function is similar to that of LAD4. The one-bit carry delay is also two NOR gate delays.

WORST CASE DELAY : 102 ns

This worst case delay is the whole 8-bit multiplication delay, therefore, the third stage delay can be obtained by subtracting
this value from the 4B4 worst delay. Thus the third stage delay is about 60 ns which means the pipelining clock rate can no more than 16 MHZ. If 8B8 is operating in pipelining mode, no more than 16 M products can be obtained per second. However, if 8B8 is operating in combinational mode, about 10 M products are generated each seconds.

POWER DISSIPATION:

260 mW (avg.)

310 mW (max.)

Because the maximum power dissipation is about 310 mW, the maximum DC current is 62 mA. Due to the metal migration phenomenon, the maximum current density is about 1.00 mA/\lambda^2. The required current can be supplied to the multiplier cell using total power lines of 62 lambda wide.
Fig. 5.8 Layout of 8E8.
The final eight-bit parallel pipelined multiplier is illustrated in Fig. 5.9. As mentioned before, the chip requires 38 pins. Fig. 5.10 shows the pin configuration of this 38-pin parallel pipelined multiplier. "A0-A8" and "B0-B8" are the input buses (multiplicand and multiplier), "P0-P15" are the product bits, "Le" and "Oe" are the two non-overlapped timing signals, and "RES" and "CS" are the two chip signals for reset and chip-select. The total chip area is about 2200 * 1700 lambda^2. Although, the effective area is very thin, for future pin connection, the chip is made wider.
Fig. 5.9 Final chip layout.
Fig. 5.10 Pin configuration of the 8-bit parallel pipelined multiplier.
CHAPTER VI DISCUSSION AND FUTURE MODIFICATION

This chapter presents the performance discussion and the future modification of the designed parallel pipelined multiplier. Section A contains the analysis of the performance of this NMOS implementation 8-bit multiplier. Section B presents a three-cell iterative algorithm which could possibly lead to a 50 percent silicon area reduction and also reduce the operation delay time.

A. Performance Discussion:

As mentioned in chapter one, the designed algorithm is potentially the fastest multiplier and can reduce the multiplication delay to the order of addition delay. However, the simulation results are quite different from this optimistic timing estimation. The simulation results show that, for the first two stages, the operation delay is about 40 ns, and the final stage requires another 60 ns to complete the remaining function. That means at most the pipelining process could save 40 percent \( \frac{40}{40+60} \) of the delay time. However, if the timing control of pipelining process is taken into account, the pipeline timing improvement will be very limited.

The functions performed by the first two stages and the third stage are almost the same, hence the delay should not
have such difference of 20 nS. The extra 20 ns delay of the third stage must be contributed by the interconnections. Because these interconnections are mainly made with metal layer, such delay must be caused by the relative area capacitance which is about 0.075 times of gate to channel area capacitance [10] (1 Cg is the capacitance of 2 lambda * 2 lambda enhancement transistor). In our design, one-bit carry propagates about 160 lambda along 3 lambda metal lines, therefore about 480 lambda square metal lines are used. This large area is about 120 times of minimum gate area, hence the interconnection capacitance is about 120 * 0.075 Cg = 9 Cg. Thus, the original 4 Cg carryout capacitive load becomes 13 Cg. This explains why the third stage needs extra 20 ns operation delay.

From the information above, we deduce that it is very important to have shorter interconnections in NMOS circuits. Normally, high regularity leads to a smaller silicon area or shorter interconnections. The regularity is the ratio of total number of required functions and the number of functions should be designed in detail [10]. The regularity provides a way to estimate the effort needed to implement a certain task. Knowing the definition of regularity, we then understand why high regularity leads to smaller silicon area consumption and shorter interconnections. Therefore, the regularity must be taken into consideration in choosing a VLSI algorithm.
Since NMOS or MOS logic have capacitive loads, the interconnection capacitance may cause serious effects in timing. Thus the regularity is the most important factor in MOS implementation. The designer of this 8-bit parallel pipelined multiplier has learned this too late to design an optimal pipelined structure. At this point it seems to be of no practical interest to discuss the feasibility of expanding such a multiplier in NMOS implementation.

B. Future Modification

This section will present a three-cell iterative scheme. The previous statements show that, to expand N/2-bit multiplication to N-bit multiplication, four N/2-bit multipliers are needed. However, for certain applications, a 2N-bit product is not required for N-bit multiplication. For example, it is enough to have a 32-bit precision product for an 32-bit floating point multiplication. The three-cell iterative scheme is based on such a requirement. In the following paragraphs, this three-cell scheme is explained.

The three-cell iterative is derived from the prediction of the product of the least significant submultiplier (LS submultiplier). If only N-bit precision is required for an N-bit by N-bit multiplication, the least significant N-bit will be omitted after all. Therefore the main function of all the bits below bit position N are just used for passing
carry. Those bits are composed of all the partial product bits produced by the LS submultiplier and half of two intermediate submultipliers. Such a waste of hardware investment can be prevented by predicting the possible results. For example, the most significant two bits of LS submultiplier can be predicted by some simple hardware logic, then, in the end, at most a 1/4-LSB error is incurred for this N-bit result. In carefully examining the binary multiplication properties, this predicting function can possibly be done with 7 simple gate functions and only uses 4 MSB of the original inputs of the LS submultiplier. If such predicting function is used, about twenty-five percents of hardware investment is saved. If further usage is made of such predicting functions in the generation of two intermediate partial products, about 37.5 percent of hardware investment can be saved with an error less than 3/4-LSB. These two kinds of three cell iterative schemes are illustrated in fig. 6-1 and fig. 6-2.

Therefore, for half-size product requirement, the three-cell iterative scheme is suitable, and the error is moderate and can be tolerated for large size multiplication. Such iterative scheme can also decrease the operation delay, due to the reduction of required addition size and the decrease of silicon area which will lead to the decrease of interconnections. From studying of such a three-cell iterative, we know that ultimately half of the hardware
Fig. 6.1 Three cell iterative scheme.

Fig. 6.2 Two stages three cell iterative scheme.
investment and half of the operation delay can be saved. However the inputs of such a multiplier should be normalized before processing them.
REFERENCES


I.LAD4

a. ESIM command file and simulation result:

```
I
W A n3 a2 a1 a0
W B b3 b2 b1 b0
W S s3 s2 s1 s0
W c4
h a3 a2 a1 a0 b3 b2 b1 b0
s
I a3 a1 b2 b0
s
h a3 c1 b2 b0
1 b3 b1 a2 a0
s
quit
```

53 transistors, 44 nodes (30 rulled wr)
initialization took 61 steps
step took 47 events
S=1110 14
B=1111 15
A=1111 15
c4=0
step took 43 events
S=1111 15
B=1010 10
A=0101 5
c4=1
step took 25 events
S=1111 15
B=0101 5
A=1010 10
c4=1

b. POWEST simulation result:

<table>
<thead>
<tr>
<th>#devs</th>
<th>Pdc_avg (W)</th>
<th>Pdc_max (W)</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00000000</td>
<td>0.00000000</td>
<td>enhancement pullups</td>
</tr>
<tr>
<td>30</td>
<td>0.007169</td>
<td>0.010486</td>
<td>depletion pullups</td>
</tr>
<tr>
<td>0</td>
<td>0.00000000</td>
<td>0.00000000</td>
<td>special depletion pullups</td>
</tr>
<tr>
<td>30</td>
<td>0.007169</td>
<td>0.010486</td>
<td>TOTAL</td>
</tr>
</tbody>
</table>
c. CRYSTAL simulation results:

```
Crystal, v.2
: build lad4.sim
[0:00.4u 0:00.2s 23k]
: input a3 a2 a1 a0 b3 b2 b1 b0
[0:00.0u 0:00.1s 28k]
: output c4 s3 s2 s1 s0
[0:00.0u 0:00.0s 28k]
: del a0 0 -1
Marking transistor flow...
Setting Vdd to 1...
Settings GND to 0...
(31 stages examined.)
[0:00.2u 0:00.1s 36k]
: cr
Node 178 is driven high at 3.83ns
   ...through fet at (27, 53) to s3
   ...through fet at (26, 60) to 199
   ...through fet at (30, 57) to Vdd after
195 is driven high at 2.54ns
   ...through fet at (29, 64) to Vdd after
136 is driven low at 2.19ns
   ...through fet at (13, 29) to GND after
111 is driven high at 1.78ns
   ...through fet at (4, 21) to Vdd after
70 is driven low at 1.30ns
   ...through fet at (13, 1) to GND after
45 is driven high at 0.89ns
   ...through fet at (4, -7) to Vdd after
12 is driven low at 0.40ns
   ...through fet at (11, -25) to 19
   ...through fet at (13, -25) to GND after
a0 is driven high at 0.00ns
[0:00.1u 0:00.0s 36k]
: cl
[0:00.0u 0:00.0s 36k]
: del a0 -1 0
Marking transistor flow...
Setting Vdd to 1...
Settings GND to 0...
```
(27 stages examined.)

Node 178 is driven high at 6.23ns
...through fct at (27, 53) to s3
...through fct at (26, 60) to 199
...through fct at (30, 57) to Vdd after 136 is driven high at 4.37ns
...through fct at (4, 31) to Vdd after 111 is driven low at 3.63ns
...through fct at (8, 18) to GND after 70 is driven high at 3.27ns
...through fct at (4, 3) to Vdd after 45 is driven low at 2.52ns
...through fct at (8, -11) to GND after 12 is driven high at 2.05ns
...through fct at (8, -25) to Vdd after a0 is driven low at 0.00ns

: quit

Crystal done.
II. PAT

a. ESIM command file and simulation result:

```
14 transistors, 15 nodes (7 pulled ur)
initialization took 21 steps
test took 14 events
S=11 3
S=11 b=1 a=1
test took 7 events
S=10 2
c=1 b=1 a=0
test took 7 events
S=01 1
c=1 b=0 a=0
test took 6 events
S=00 0
c=0 b=0 a=0
test took 6 events
S=10 2
c=0 b=1 a=1
```

b. POWEST simulation result:

<table>
<thead>
<tr>
<th>#devs</th>
<th>Pdc_avg (W)</th>
<th>Pdc_max (W)</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000000</td>
<td>0.000000</td>
<td>enhancement pullups</td>
</tr>
<tr>
<td>7</td>
<td>0.001785</td>
<td>0.002840</td>
<td>depletion pullups</td>
</tr>
<tr>
<td>0</td>
<td>0.000000</td>
<td>0.000000</td>
<td>special depletion pullups</td>
</tr>
<tr>
<td>7</td>
<td>0.001785</td>
<td>0.002840</td>
<td>TOTAL</td>
</tr>
</tbody>
</table>
c. CRYSTAL simulation result:

Crystal: v.2
: build pat.sim
[0:00.1u 0:00.1s 17k]
: input a b c
[0:00.0u 0:00.0s 22k]
: output co s
[0:00.0u 0:00.0s 22k]
: clear
[0:00.0u 0:00.0s 22k]
: delay a 0 -1
Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
(15 stages examined.)
[0:00.1u 0:00.0s 23k]
: cri
Node 24 is driven high at 1.46ns
...through fet at (-5, 22) to 21
...through fet at (-3, 15) to Vdd after
15 is driven low at 0.04ns
...through fet at (-12, 11) to b after
a is driven high at 0.00ns
[0:00.0u 0:00.0s 23k]
: cl
[0:00.0u 0:00.0s 23k]
: del a -1 0
Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
(17 stages examined.)
[0:00.1u 0:00.0s 23k]
: cr
Node 24 is driven high at 1.96ns
...through fet at (-15, 25) to 15
...through fet at (-13, 15) to Vdd after
a is driven low at 0.00ns
[0:00.0u 0:00.0s 23k]
: cl
[0:00.0u 0:00.0s 23k]
Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
(14 stages examined.)
[0:00.0u 0:00.0s 23k]
: cr
Node 15 is driven high at 2.69ns
...through fet at (-15, 25) to 24
...through fet at (-5, 22) to 21
...through fet at (-3, 15) to Vdd after
c is driven high at 0.00ns
[0:00.0u 0:00.0s 23k]
: c]
[0:00.0u 0:00.0s 23k]
: del c -1 0
Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
(14 stages examined.)
[0:00.1u 0:00.0s 23k]
: cr
Node 23 is driven high at 1.87ns
...through fet at (-37, 22) to 30
...through fet at (-35, 14) to Vdd after
29 is driven high at 0.22ns
...through fet at (16, 15) to Vdd after
c is driven low at 0.00ns
[0:00.0u 0:00.0s 23k]
: quit
[0:00.6u 0:00.2s 23k] Crystal done.
III. CCT3
   a. ESIM command file and simulation result:

I
W IN i2 i1 i0
W OUT o2 o1 o0
w c
l i2 i1 i0 c
s
h c
s
h i0
s
l c
s
h i1
s
h c
s
h 12
s
l c
s
quit

14 transistors, 15 nodes (5 pulled up)
initialization took 12 steps
step took 10 events
OUT=001 1
IN=000 0
c=0
step took 8 events
OUT=000 0
IN=000 0
c=1
step took 4 events
OUT=001 1
IN=001 1
c=1
step took 5 events
OUT=010 2
IN=001 1
c=0
step took 4 events
OUT=100 4
IN=011 3
c=0
step took 7 events
OUT=011 3
IN=011 3
c=1
step took 0 events
OUT=011 3
IN=011 3
c=1
step took 4 events
OUT=100 4
IN=011 3
c=0
b. CRYSTAL simulation result:

Crystal  v.2
: build cct3.sim
[0:00.2u 0:00.1s 17k]
: input i2 i1 i0 c
[0:00.0u 0:00.0s 22k]
: output o2 o1 o0
[0:00.0u 0:00.0s 22k]
: del i0 0 -1
Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
More than 5 transistors in series, see o1 (see source at 6,12)
(14 stages examined.)
[0:00.1u 0:00.0s 23k]
: cr
Node 34 is driven high at 2.32ns
...through fet at (2, -4) to i1 after
25 is driven high at 0.53ns
...through fet at (8, 12) to o0
...through fet at (16, -2) to i0 after
10 is driven high at 0.00ns
[0:00.0u 0:00.0s 23k]
: cl
[0:00.0u 0:00.0s 23k]
: del i0 -1 0
Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
More than 5 transistors in series, see o1 (see source at 6,12)
(19 stages examined.)
[0:00.1u 0:00.0s 23k]
: cr
Node o2 is driven high at 1.42ns
...through fet at (-17, 12) to 33
...through fet at (-24, -1) to 24
...through fet at (-14, 0) to Vdd after
10 is driven low at 0.00ns
[0:00.0u 0:00.0s 23k]
Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
(22 stages examined.)

Node 00 is driven high at 1.55ns
...through FET at (8, 12) to 25
...through FET at (-2, -1) to 34
...through FET at (1, 0) to Vdd after
35 is driven low at 0.04ns
...through FET at (20, -3) to GND after
c is driven high at 0.00ns

Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
(24 stages examined.)

Node 00 is driven high at 1.87ns
...through FET at (8, 12) to 25
...through FET at (-2, -1) to 34
...through FET at (6, 12) to 01
...through FET at (-5, 9) to 11 after
35 is driven high at 0.48ns
...through FET at (20, 2) to Vdd after
c is driven low at 0.00ns

POWEST simulation result:

<table>
<thead>
<tr>
<th>#devs</th>
<th>Pdc_avg (W)</th>
<th>Pdc_max (W)</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000000</td>
<td>0.000000</td>
<td>enhancement pullups</td>
</tr>
<tr>
<td>5</td>
<td>0.001018</td>
<td>0.001389</td>
<td>depletion pullups</td>
</tr>
<tr>
<td>0</td>
<td>0.000000</td>
<td>0.000000</td>
<td>special depletion pullups</td>
</tr>
<tr>
<td>5</td>
<td>0.001018</td>
<td>0.001389</td>
<td>TOTAL</td>
</tr>
</tbody>
</table>
IV. INP

a. ESIM command file and simulation result:

```
 6 transistors, 11 nodes (3 pulled up)
initialization took 13 steps
step took 5 events
  o/p=0 i/p=1 res=0 oe=1 le=1
step took 4 events
  o/p=1 i/p=0 res=0 oe=1 le=1
step took 4 events
  o/p=1 i/p=1 res=0 oe=0 le=1
step took 6 events
  o/p=0 i/p=1 res=0 oe=1 le=0
```

b. POWEST simulation result:

<table>
<thead>
<tr>
<th>#devs</th>
<th>Pdc_avg (W)</th>
<th>Pdc_max (W)</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000000</td>
<td>0.000000</td>
<td>enhancement pullups</td>
</tr>
<tr>
<td>3</td>
<td>0.000893</td>
<td>0.001691</td>
<td>depletion pullups</td>
</tr>
<tr>
<td>0</td>
<td>0.000000</td>
<td>0.000000</td>
<td>special depletion pullups</td>
</tr>
<tr>
<td>3</td>
<td>0.000893</td>
<td>0.001691</td>
<td>TOTAL</td>
</tr>
</tbody>
</table>
c. CRYSTAL simulation result:

Crystal: v.2
: build in. sim
[0:00.1u 0:00.1s 16k]
: input i/p le oe res
[0:00.0u 0:00.0s 21k]
: output o/p
[0:00.0u 0:00.0s 21k]
: clear
[0:00.0u 0:00.0s 21k]
: delay i/p 0 -1
Marking transistor flow...
Setting Vdd to 1...
Settings GND to 0...
(6 stages examined.)
[0:00.1u 0:00.0s 22k]
: critical
Node o/p is driven low at 1.43ns
...through fet at (13, -5) to GND after
13 is driven high at 1.34ns
...through fet at (6, 2) to Vdd after
12 is driven low at 0.95ns
...through fet at (0, 3) to 18
...through fet at (-5, -5) to GND after
9 is driven high at 0.24ns
...through fet at (1, -11) to i/p after
i/p is driven high at 0.00ns
[0:00.0u 0:00.0s 22k]
: clear
[0:00.0u 0:00.0s 22k]
: delay i/p -1 0
Marking transistor flow...
Setting Vdd to 1...
Settings GND to 0...
(6 stages examined.)
[0:00.1u 0:00.0s 22k]
: critical
Node o/p is driven high at 2.35ns
...through fet at (14, 2) to Vdd after
13 is driven low at 2.23ns
...through fet at (5, -5) to GND after
12 is driven high at 2.12ns
...through fet at (0, 3) to 18
...through fet at (-5, -1) to Vdd after
9 is driven low at 0.12ns
...through fet at (1, -11) to i/p after
i/p is driven low at 0.00ns
[0:00.0u 0:00.0s 22k]
: quit
[0:00.3u 0:00.3s 22k] C r y s t a l done.
V. 4B4
  a. ESIM command file and simulation result:

1
W A a3 a2 a1 a0
W B b3 b2 b1 b0
W F r7 r6 r5 r4 r3 r2 r1 r0
w le oe res
h a3 a2 a1 a0 b3 b2 b1 b0 1r oe
1 res
s
1 a3 a1 b2 b0
s
1 a2 a0
s
1 b3
h a2 a0 a3
s
quit

297 transistors, 226 nodes (159 pulled up)
initialization took 172 steps
step took 140 events
F=11100001
B=1111 15
A=1111 15
res=0 oe=1 le=1
step took 339 events
F=00110010
B=1010 10
A=0101 5
res=0 oe=1 le=1
step took 66 events
F=00000000
B=1010 10
A=0000 0
res=0 oe=1 le=1
step took 61 events
F=00011010
B=0010 2
A=1101 13
res=0 oe=1 le=1
b. CRYSTAL simulation results:

Crystal, v.2
: build 4b4.sim
[0:02.2u 0:00.3s 59k]
: input a3 a2 a1 a0 b3 b2 b1 b0 le oe res
[0:00.0u 0:00.0s 64k]
: output p p6 p5 p4 p3 p2 p1 0
p isn't in table!
[0:00.0u 0:00.0s 64k]
: del a0 0 -1
Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
(2231 stages examined.)
[0:15.1u 0:00.8s 150k]
: cr
Node 1074 is driven high at 39.85ns
...through fet at (99, 73) to p6
...through fet at (113, 81) to 1125
...through fet at (100, 91) to 1171
...through fet at (113, 84) to p7
...through fet at (110, 94) to 1173
...through fet at (56, 90) to Vdd after

105 is driven high at 0.24ns
...through fet at (-87, -79) to a0 after
a0 is driven high at 0.00ns
[0:00.2u 0:00.1s 150k]
: cl
[0:00.1u 0:00.0s 150k]
: del a0 -1 0
Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
(2762 stages examined.)
[0:16.1u 0:00.6s 167k]
: cr
Node 1074 is driven high at 41.70ns
...through fet at (99, 73) to p6
...through fet at (113, 81) to 1125
...through fet at (100, 91) to 1171
...through fet at (113, 84) to p7
...through fet at (110, 94) to 1173
...through fet at (56, 90) to Vdd after
Node 1074 is driven high at 40.38ns
...through fet at (99, 73) to p6
...through fet at (113, 81) to 1125
...through fet at (100, 91) to 1171
...through fet at (113, 84) to r7
...through fet at (110, 94) to 1173
...through fet at (56, 90) to Vdd after

Node 1074 is driven high at 41.70ns
...through fet at (99, 73) to p6
...through fet at (113, 81) to 1125
...through fet at (100, 91) to 1171
...through fet at (113, 84) to r7
...through fet at (110, 94) to 1173
...through fet at (56, 90) to Vdd after
...through $fet$ at $(-82, -33)$ to GND after 353 ns when driven high at 2.12 ns
...through $fet$ at $(-74, -24)$ to $V_{dd}$ after 411 ns when driven low at 0.12 ns
...through $fet$ at $(-77, -22)$ to $V_{dd}$ after 411 ns when driven low at 0.12 ns
...through $fet$ at $(-87, -27)$ to $V_{dd}$ after 411 ns when driven low at 0.12 ns

[0:00.2u 0:100.0s 180k]
: quit
[1:23.7u 0:103.5s 180k] CrystaL done.

c. POWEST simulation results:

<table>
<thead>
<tr>
<th>$devs$</th>
<th>$P_{dc_avg}$ (W)</th>
<th>$P_{dc_max}$ (W)</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000000</td>
<td>0.000000</td>
<td>enhancement pullups</td>
</tr>
<tr>
<td>159</td>
<td>0.049098</td>
<td>0.070775</td>
<td>depletion pullups</td>
</tr>
<tr>
<td>0</td>
<td>0.000000</td>
<td>0.000000</td>
<td>special depletion pullups</td>
</tr>
<tr>
<td>159</td>
<td>0.049098</td>
<td>0.070775</td>
<td>TOTAL</td>
</tr>
</tbody>
</table>
VI. 8B8

a. CRYSTAL simulation result:

Crystal, v.2
: build 8b8_sim
[0:14.2u 0:01.6s 251k]
: input le oe res a7 a6 a5 a4 a3 a2 a1 a0 b7 b6 b5 b4 b3 b2 b1 b0
[0:100.0u 0:100.1s 256k]
: output r15 r14 r13 r12 r11 r10 r9 r8 r7 r6 r5 r4 r3 r2 r1 r0
[0:100.0u 0:100.0s 256k]
: del a0 0 -1
Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
More than 5 transistors in series, see 6952
(see source at 150,395).
More than 5 transistors in series, see r15
(see source at 163,388).
More than 5 transistors in series, see r13
(see source at 149,355).
More than 5 transistors in series, see 6534
(see gate at 145,384).
More than 5 transistors in series, see 6931
(see drain at 145,384).
More than 5 transistors in series, see 6746
(see gate at 142,384).
More than 5 transistors in series, see r14
(see source at 163,366).
More than 5 transistors in series, see 6952
(see gate at 150,395).
More than 5 transistors in series, see r15
(see source at 163,388).
More than 5 transistors in series, see 6952
(see source at 150,395).
More than 5 transistors in series, see 6952
(see source at 150,395).
No more messages of this kind will be printed.....
(78829 states examined.)
[7:41.8u 0:02.4s 569kJ]
: cr
Node 6952 is driven low at 102.34ns
...through fet at (160, 398) to r15
...through fet at (163, 388) to 6972
...through fet at (150, 395) to 6919
...through fet at (145, 384) to 6931
...through fet at (142, 384) to GND after
6534 is driven high at 100.79ns
a0 is driven high at 0.00ns
[0:100.5u 0:100.1s 569kJ]
: cl
[0:100.2u 0:100.0s 569kJ]
: del a0 -1 0
Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
(72731 states examined.)
[7:00.1u 0:00.5s 657kJ]
: cr
Node 6952 is driven low at 103.03ns
...through fet at (160, 398) to r15
...through fet at (163, 388) to 6972
146 is driven low at 0.12ns
...through fet at (-194, -417) to a0 after
a0 is driven low at 0.00ns
[0:00.5u 0:00.1s 657k]
: cl
[0:00.2u 0:00.1s 657k]
: del b0 0 -1
Markins transistor flow...
Setting Vdd to 1...
Setting GND to 0...
(86461 stages examined.)
[7:56.4u 0:06.2s 696k]
: cr
Node 6952 is driven low at 102.53ns
...through fet at (160, 398) to #15
...through fet at (163, 388) to 6972
...through fet at (150, 395) to 6919
...through fet at (145, 384) to 6931
...through fet at (142, 384) to GND after
6534 is driven high at 100.98ns

...through fet at (-194, -365) to b0 after
b0 is driven high at 0.00ns
[0:00.5u 0:00.2s 696k]
: cl
[0:00.2u 0:00.0s 696k]
: del b0 -1 0
Markins transistor flow...
Setting Vdd to 1...
Setting GND to 0...
(89053 stages examined.)
[8:09.7u 0:03.7s 753k]
: cr
Node 6952 is driven low at 102.99ns
...through fet at (160, 398) to #15
...through fet at (163, 388) to 6972
...through fet at (150, 395) to 6919
...through fet at (145, 384) to 6931

423 is driven high at 2.12ns
...through fet at (-180, -364) to 513
...through fet at (-184, -360) to Vdd after
489 is driven low at 0.12ns
...through fet at (-194, -365) to b0 after
b0 is driven low at 0.00ns
[0:00.5u 0:00.1s 753k]
: quit
[31:04.8u 0:17.0s 753k] Crystal done.
b. ESIM command file and simulation result:

```
1
W P r15 r14 r13 r12 r11 r10 r9 r8 r7 r6 r5 r4 r3 r2 r1 r0
W A a7 a6 a5 a4 a3 a2 a1 a0
W b7 b6 b5 b4 b3 b2 b1 b0
h le oe
l res
s
h a0 b0
l oe
s
h oe
l le
s
h a2 b2
l oe
h le
s
h oe
l le
s
h a4 b4
l oe
h le
s
h oe
l le
s
h a6 b6
l oe
h le
s
h oe
l le
s
h oe
l le
s
h oe
l le
s
h oe
l le
s
h quit
```
1570 transistors, 1171 nodes (833 pulled up)
initialization took 2699 steps
step took 2970 events
B=00000000 0
A=00000000 0
P=0000000000000000 0
step took 227 events
B=00000001 1
A=00000001 1
P=0000000000000000 0
step took 344 events
B=00000001 1
A=00000001 1
P=0000000000000000 0
step took 329 events
B=00000101 5
A=00000101 5
P=0000000000000000 0
step took 354 events
B=00000101 5
A=00000101 5
P=0000000000000000 0
step took 331 events
B=00010101 21
A=00010101 21
P=0000000000000000 0
step took 362 events
B=00010101 21
A=00010101 21
P=0000000000000000 0
step took 333 events
B=01010101 85
A=01010101 85
P=0000000000000001 1
step took 383 events
B=01010101 85
A=01010101 85
P=00000000000011001 25
step took 331 events
B=01010101 85
A=01010101 85
P=00000000000011001 25
step took 428 events
c. POWEST simulation results:

<table>
<thead>
<tr>
<th>#devs</th>
<th>Pdc_avg (W)</th>
<th>Pdc_max (W)</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>833</td>
<td>0.000000</td>
<td>0.000000</td>
<td>enhancement pullup</td>
</tr>
<tr>
<td>833</td>
<td>0.222570</td>
<td>0.314264</td>
<td>depletion pullup</td>
</tr>
<tr>
<td>833</td>
<td>0.222570</td>
<td>0.314264</td>
<td>special depletion pullup</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TOTAL</td>
</tr>
</tbody>
</table>