Pipelined Floating Point Divider
with Built-in Testing Circuits

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
Chung-nan Lyu
June, 1988
ACKNOWLEDGEMENTS

I would like to extend my deep gratitude to my advisor Dr. Janusz A. Starzyk for his guidance and patience throughout this project. I also extend a special note of gratitude to the members of my committee, Dr. Henry Lozykowski, Dr. Mehmet Celenk, and Dr. John Gillam for their assistance.

Finally, I want to express my great appreciation to my parents, sisters, and my close friend, whose endless love and encouragement made this possible.
# TABLE OF CONTENTS

Chapter 1: INTRODUCTION.................................................................1

Chapter 2: VLSI SYSTEM DESIGN AND HARDWARE IMPLEMENTATION........8
    2.1 VLSI System Design.......................................................8
    2.2 Division Algorithm.....................................................10
    2.3 Logic Design of Unit Processor.................................12
    2.4 Shift Register..........................................................16
    2.5 Design of Basic Cells..............................................21
    2.6 Clock Synchronization...............................................24
    2.7 VLSI Layout ............................................................24
    2.8 Routing and Current Requirements.........................35
    2.9 Timing Analysis.....................................................38

Chapter 3: TESTING.................................................................41
    3.1 Introduction............................................................41
    3.2 Conventional Testing Methods.................................42
    3.3 Design For Testability..............................................45
    3.4 Testing Algorithm..................................................47
    3.5 Timing Analysis.....................................................53

Chapter 4: SIMULATION AND DISCUSSIONS.................................56
    4.1 Design Strategy and CAD Design Tools.......................56
    4.2 SCALDstar System.....................................................59
    4.3 Simulation.............................................................60
    4.4 Discussions..........................................................63

Chapter 5: CONCLUSION...........................................................70
Chapter 1
INTRODUCTION

Over the past four decades, the computer industry has experienced four generations of revolution, from relays and vacuum tubes (1940-1950s), to discrete diodes and transistors (1950-1960s), to small- and medium-scale integrated (SSI/MSI) circuits (1960-1970s), and to large- and very-large-integrated (LSI/VLSI) devices (1970s-beyond) (Kai Hwang, 1984). The potential benefits to be gained through the use of LSI/VLSI devices are:

1. High reliability: This is achieved by the reduction in the number of interconnections between chips. Placing more gates, and hence more interconnections on the chip results in an inherently more reliable system due to longer mean time to failure of on-chip interconnections as compared to chip-to-chip interconnections. In addition, the mean time to repair can be minimized by using LSI/VLSI devices, since the fewer the number of chips in the system is, the easier and faster it is to isolate and replace a failing component.

2. High speed: This is gained by the ability of LSI/VLSI devices to put large number of elements on the chip. The capacitance of on-chip interconnections is significantly lower than that of off-chip interconnections. As a result, on-chip interconnections can be switched more rapidly than off-chip interconnections.
3. Small size (high density) and low parts count: This reduces the total length and number of interconnections between chips, which contributes to the high reliability and speed of LSI/VLSI (Guy Rabbat, 1983).

4. Low cost: The mass production of LSI/VLSI circuits and higher density of the IC packages reduce the power requirements, the number of PC boards and cabinets, resulting in smaller and cheaper system. The larger the IC package, the greater the cost savings.

Improvements in reliability and speed, as well as reduction in hardware cost and size have greatly enhanced computer performance. However, these are not the sole factors contributing to the high performance. Modern computer system uses the parallel processing strategy. "Parallel processing is an efficient form of information processing which emphasizes the exploitation of concurrent events in the computing process." (Kai Hwang, 1984) Parallel processing demands concurrent execution of many instructions (programs) in the computer and is a cost-effective mean to improve system performance. (Kai Hwang, 1984) There are three kinds of concurrent processing:

1. Parallel events may occur in multiple resources during the same time interval.

2. Simultaneous events may occur at the same time interval.

3. Pipelined events may occur in overlapped time or
intervals. (Kai Hwang, 1984)

The objective of this thesis is to present a design of a floating-point divider which may be used as an auxiliary processor of the CPU component in the computer. The divider is designed in nMOS technology using VLSI design tools available in the Electrical Engineering department.

The algorithm selected to realize this divider is suitable for LSI/VLSI implementation and allow testing circuits to be easily incorporated in the main design.

The CPU can do other tasks while the divider carries out its function. Since parallel processing and distributed processing are closely related, we may view the whole CPU as a form of parallel processing. Sometimes we have to use distributed techniques to gain parallelism (Kai Hwang, 1984). The divider itself also executes tasks using pipelining techniques.

The divider is a 24 pin IC with built-in testing circuits. In the operation mode, it generates outputs \( Q_i = A_i / B_i \), where \( A_i \) and \( B_i \) are input signals. In the testing mode, a functional test of the IC is performed. The pin configuration of the divider with its built-in testing circuits is shown in Fig. 1-1. Organization of the thesis is as follows.

Chapter two discusses the VLSI system design techniques and hardware implementation of the divider structure using nMOS technology. The divider and built-in testing circuits
Fig. 1-1 Pin configuration of the divider
are designed with full adders/subtractors and shift registers. The operation algorithm as well as logic level circuits of these basic cells are discussed in this chapter. The design layouts of these basic cells, the floor plan of the entire structure, the non-overlapped clock synchronizations, the calculation of current requirements, and the timing analysis are considered in this chapter. The floor plan of the entire structure is shown in Fig. 1-2. All the on-chip interconnections are put into the five basic cells, which simplifies its entire structure and allows upgrading of this design when more bits and higher accuracy are desired.

Chapter three discusses the testing algorithm and built-in testing circuits. "Testing, in its broadest sense, means to examine (the whole and all parts of) a product, to ensure that it functions and exhibits the properties and capabilities it is designed to possess." (Parag K. Lala, 1985) Two key points (controllability and observability) in design for testability are found in the testing circuits as independent of any fault-assumption. Scan-in and "AC" or "at-speed" testing techniques are employed. The number of testing vectors, the timing analysis, the clocking strategy, and the time required to finish the testing are also discussed in detail in this chapter.

Chapter four introduces the CAD design tools and presents the responses of the divider with its built-in
Routing area

Basic cells

Fig. 1-2 Floor plan of the divider
testing circuits. The simulation results are obtained by using the event driven simulator (ESIM) for the overall structure of the divider and show that this design is logically correct. The performance of this divider is deduced by running the CRYSTAL package on VAX. The timing analysis and worst case delay will also be presented in this chapter. The power dissipation is evaluated by POWEST package on VAX.

Chapter five summarizes the design of the divider with built-in testing circuits. The general trend, with emphasis on parallelism and "design for testability" in VLSI technology, is presented.
MOS technology may include pMOS, nMOS, and CMOS devices. The attractive property of pMOS is its easy manufacturing; of nMOS, its fast devices; of CMOS, its low power dissipation. Various designs and methodologies are considered in order to obtain minimum hardware and save design time.

2.1 - VLSI System Design

The design description for an integrated circuit may be described in terms of three domains, namely: 1) the behavioral domain, 2) the structural domain, and 3) the physical domain. These domains may be hierarchically divided into levels of design abstraction as below:

1. Architectural or functional level;
2. Register transfer level;
3. Logic level;

Fig. 2-1 illustrates different approaches for a typical design flow chart.

According to Principles of CMOS VLSI Design (Weste & Eshraghian, 1985), there are four design styles:

1. Structured design:
   1) hierarchy, 2) modularity, 3) regularity,
Fig. 2-1 LSI/VLSI typical design flow chart

(Weste & Esghragni, 1985)
4) locality;
2. Handcrafted mask layout design;
3. Gate array design;
4. Standard cell design.

nMOS technology and standard cell design are used because the divider itself has a very regular structure. All of these will be discussed in detail in the following sections.

### 2.2 - Division Algorithm

Considering two normalized floating-point numbers:

\[ N = n_0n_1n_2\ldots n_m \text{ (Dividend or Numerator)} \]
\[ D = d_0d_1d_2\ldots d_k \text{ (Divisor or Denominator)} \]

To perform the following division

\[ Q = N / D = q_0q_1q_2\ldots q_k \]

The operation to be performed in the first row is always subtraction (using 2's complement in a hardware implementation). The rest of the rows of the processors are either addition or subtraction depending on whether the sign of its previous row's output agrees with the dividend sign. When an unsuccessful subtraction occurs, the partial remainder changes its sign with respect to the dividend. (In the situation of 2's complement, the carry of this operation is "0".) A "0" quotient digit is generated and the partial remainder is first shifted left (in the hardware implementation) along the diagonals and then added to the
divisor in the next row. If the partial remainder is positive (in the situation of 2's complement, the carry is "1"), a "1" quotient digit is generated and the operation at the next row is subtraction. (The partial remainder will also be shifted left along the diagonals.) Thus, the quotient bit will be always the same as the carry out bit.

The example below is presented in four stages, each of which will generate one digit of the quotient. The more stages the divider has, the more accurate the result is. Following this algorithm, the reader can further iterate the algorithm to get more accurate results.

Example: Let us consider the division of \( N \) by \( D \) given by:

\[
N = (0.101001)_2 \\
D = (0.111)_2
\]

<table>
<thead>
<tr>
<th>Dividend N</th>
<th>0.101001</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subtract D</td>
<td>1.001</td>
</tr>
<tr>
<td>(in 2's comp.)</td>
<td></td>
</tr>
</tbody>
</table>

| 0 1.110001 | negative partial remainder |
| 1.10001   | quotient bit = 0            |
| Add D     | 0.111                     |

| 1 0.01101 | positive partial remainder |
| 0.1101   | quotient bit = 1            |
| shift left one bit |         |
subtract D 1.001
(in 2's comp.)

<table>
<thead>
<tr>
<th>0 1.1111</th>
<th>negative partial remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>quotient bit = 0</td>
</tr>
<tr>
<td>1.111</td>
<td>shift left one bit</td>
</tr>
<tr>
<td>Add D</td>
<td>0.111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1 0.110</th>
<th>positive partial remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>quotient bit = 1</td>
</tr>
</tbody>
</table>

So the result is

\[ Q = q_0q_1q_2q_3 = 0.101 \] (For this 4-bit result, the accuracy is about 85%).

### 2.3 - Logic Design of Unit Processor

The unit processor itself is an adder or a subtracter depending on the function required. The logic design of this unit processor is shown in Fig. 2-2. Due to the VLSI design using negative logic restriction, it is converted into another form as shown in Fig. 2-3. The control signal P is used to control the processor's functions of addition and subtraction (P = 0 for addition and P = 1 for subtraction). This P-controlled adder/subtractor is called unit processor PAS in this thesis. The block diagram of the divider is presented in Fig. 2-4. Since the quotient bit is always the same as the carry-out, the carry-out can be viewed as the quotient bit. On the other hand, the operation performed in the next row is determined by the sign of the current...
Fig. 2-2 Logic circuit of an adder/subtractor
Fig. 2-3 Revised logic circuit of an adder/subtractor
Fig. 2-4 Block diagram of the divider
output. When the carry is "0" (negative partial remainder), the operation below this row is addition. When the carry is "1" (positive partial remainder), the operation below is subtraction. So the carry-out (or the quotient) can be connected to the next row's P input. In hardware implementation, the carry-in of the very right processor of each row should be "0" for addition and "1" for subtraction. This can be achieved by simply connecting P to $C_i$ of the rightmost processor.

Because the quotient bit cannot be generated until the completion of the carry-out propagation horizontally through the serial connection of the adders/subtractors, the performance may be slow. The pipelined divider is formed by inserting proper clock synchronization and shift registers between rows (Fig. 2-5). This pipelined divider can generate one bit of quotient at the end of each clock period. Different sets of inputs can generate different quotient bits in overlapped time intervals. It is easy to see this property in Fig.'s 2-6 and 2-7.

2.4 - Shift Register

A latch is used to cause a unit delay. In each row, there are eight registers. The input sets and intermediate partial remainders need to be latched for a certain amount of time by these registers until the carry (quotient) bit is generated. Then this result is passed to the next row.
Fig. 2-5 Pipelined divider with shift registers
The sequence of data to be loaded in

<table>
<thead>
<tr>
<th>Clock</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sixth</td>
<td>A6 (A60 ' A63)</td>
</tr>
<tr>
<td>Fifth</td>
<td>A5 (A50 ' A53)</td>
</tr>
<tr>
<td>Fourth</td>
<td>A4 (A40 ' A43)</td>
</tr>
<tr>
<td>Third</td>
<td>A3 (A30 ' A33)</td>
</tr>
<tr>
<td>Second</td>
<td>A2 (A20 ' A23)</td>
</tr>
<tr>
<td>First</td>
<td>A1 (A10 ' A13)</td>
</tr>
</tbody>
</table>

Fig. 2-6 Data arrangements of the pipelined divider
Fourth clock \( A_4 \) \( \rightarrow \) \( B_4 \)
Third clock \( A_3 \) \( \rightarrow \) \( B_3 \)
Second clock \( A_2 \) \( \rightarrow \) \( B_2 \)

CLOCK

CLOCKT

PipeLine

DivideR

At the end of the first CLOCK \( Q_{10} \)

Fig. 2-7 (a)

Fourth clock \( A_4 \) \( \rightarrow \) \( B_4 \)
Third clock \( A_3 \) \( \rightarrow \) \( B_3 \)

CLOCK

CLOCKT

PipeLine

DivideR

At the end of the second clock \( Q_{20} \) \( Q_{11} \)
At the end of the first clock \( Q_{10} \)

Fig. 2-7 (b)
Fig. 2-7 Four consecutive steps of the pipelined divider
During the testing mode, these registers are used as the scan path. The logic circuit of the shift register is shown in Fig. 2-8. Because this design includes operation and testing modes, there are two input- and two output-paths for each of these registers. An OR gate is used as the paths for these two signals. The control signal T is "1" during the testing mode and "0" otherwise. The CLOCKT is a non-overlapped clock with respect to CLOCK. This will be discussed more thoroughly in section 2-9.

2.5 - Design of Basic Cells

Two shift registers, one adder/subtractor(PAS), and the necessary interconnections are included into one unit in order to simplify the overall structure. This is named CELL and the logic circuit is shown in Fig. 2-9.

The first row of this divider always executes subtraction. Since it is somewhat different from the rest of the rows, the units in the first row are named CELLU. Their P's are always "1".

In this design, four CELLs are used in a row (see Fig. 2-5). In total, there are four PAS's and eight shift registers in every row with shift registers used to latch inputs and partial remainders. The leftmost partial remainder of each row is discarded.

Since each CELL or CELLU needs control signals, clocking, and scan-in data function during the testing mode,
Fig. 2-8 Logic circuit of shift registers
Fig. 2-9 Logic circuit of a basic CELL
TEMPO, TEMPO1, and TEMPO2 are designed so that each of them provides the necessary interconnections to transfer all signals in and quotient bits out of this divider. TEMPO, TEMPO1 and TEMPO2 have been modified to suit the particular needs of the divider. The logic circuits of these units are shown in Fig. 2-10 while the overall floor plan of the divider, with paddings, is shown in Fig. 2-11.

2.6 - Clock Synchronization

Since the divider employs pipelining techniques, it has to use clock synchronization. In the operation mode, two non-overlapped clocks, CLOCK and CLOCKT, are used to synchronize the data in and out respectively. Both signals switch between zero volt (logic-0) and a voltage near VDD (logic-1). Note that both signals are asymmetric and do not have overlap. Both clocks are shown in Fig. 2-12. In each of these two clocks there are different periods in different modes (operation and testing).

2.7 - VLSI Layout

The VLSI layouts of CELLU, CELL, TEMPO, TEMPO1 and TEMPO2 are shown in Fig.'s 2-13, 2-14, 2-15, 2-16, and 2-17 respectively. Since the size of the active region of the divider is 1300 lambda x 1000 lambda, it is impossible to present the overall layout diagram in a single sheet. The reader can think of it as a combination of all these layouts
Fig. 2-10 (a) Logic circuit for TEMPO
Fig. 2-10 (b) Logic circuit for TEMPO1
Fig. 2-10 (c) Logic circuit for TEMPO2
Fig. 2-11 Floor plan of the divider with paddings
non-overlapped high time

Operation of a series of four adders/subtracters

Quotient bits released

Partial remainders latched by shift registers

Fig. 2-12 Configuration of CLOCK and CLOCKT in the operation mode
Fig. 2-13 Layout of CELLU
Fig. 2-14 Layout of CELL
Fig. 2-15 Layout of TEMPO
Fig. 2-16 Layout of TEMPO1
Fig. 2-17 Layout of TEMPO2
according to the overall floor plan.

### 2.8 - Routing and Current Requirements

This section discusses the routing and maximum current required for the power supply of the divider.

VDD and GND paths run through the metal layers because of low heat dissipation and low voltage drops along the lines due to low resistance and because of good speed due to low capacitance. Metal routing should be wide enough to supply sufficient current to each unit of the divider. VDD and GND paths form a set of interdigitated combs so that both of them are able to run through any cell. The configuration of VDD and GND paths running in between cells is shown in Fig. 2-18. Control lines are generally run by polysilicon perpendicular to metal wires. The VDD and GND lines are never run in polysilicon since it has a large resistance which causes considerable voltage drops. Diffusion wires are used for local computation and are never used to carry signals over a long distance because of their relatively large capacitance. Table 2.1 shows the resistance and capacitance of different layers.
Fig. 2-18 Interdigitized Vdd and GND routing
Resistance: ohm / square unit
Metal = 0.03
Diffusion = 10
Polysilicon = 15 - 100
Transistor = 10^4

Capacitance: pf / um^2
Gate channel = 4x10^-4
Diffusion = 1x10^-4
polysilicon = 0.4x10^-4
Metal = 0.3x10^-4

Table 2.1 Typical MOS electrical parameters (1978)
(from Introduction to VLSI System, 1980)

Because VDD and GND run in metal, the metal must be wide enough to drive the required current. The inverter employed in this design has the length to width ratio of pull up and pull down resistances 4:1 and 8:1. Therefore, according to Table 2.1, when the pull down transistor of this inverter is on, the current through it would be approximately 0.125 mA [4:1] and it is even smaller when using a 8:1 inverter. The NAND gates, NOR gates, and XNOR gates have similar physical structures to that of the inverter. The current through these gates would be about
0.125 mA or less. According to "nMOS & CMOS VLSI System Design", 1986, the current through a standard inverter with ratio 4 is 0.06 mA and the maximum current density of the metal is 1 mA/um. Taking the width of the metal equal 4 lambda (lambda = 2 microns), the number of gates supported by this metal line could be:

\[ 8 \times 1 / 0.06 = 133 \]

In this design, the number of gates between each pair of VDD and GND never exceeds 200. Supposing half of the gates will consume current at the same time, each pair of VDD and GND wires may support 100 gates. So the 4 lambda wide metal layers are sufficient to feed both the static and dynamic currents. On the other hand, estimating the total number of gates in the design at 700, and assuming that half of them are taking 0.06 mA, the minimum width of the global metal lines is

\[ 350 \times 0.06 / 2 = 10.5 \]

Thus a 15 lambda width of metal for global VDD and GND is enough.

2.9 - Timing Analysis

The input signals pass through five stages of adders/subtractors, each of which can be viewed as a series of four adders/subtractors. Every adder/subtractor has to wait until the carry to its right is generated in order to operate. The CLOCK period must be high during the operation
period of one stage. The CLOCKT is a non-overlapped clock with respect to CLOCK. When it is high (and the CLOCK is low), one bit of quotient is generated for each stage. When it is low, the CLOCK is high again. The operation of serial adders/subtractors continues in order to generate the next quotient bit for the next stage (Fig. 2-12). During the testing mode, the CLOCK and the CLOCKT are also non-overlapped, but their periods are very different from that in the operation mode. The worst case delay is determined by CRYSTAL and will be discussed in chapter four. The periods of clocks in different modes are also determined by CRYSTAL. In this design, every input of shift registers and quotient outputs have signals coming from two different paths; one for the operation mode and one for the testing mode. NOR gates, pass transistors, and discharged circuits are used in this design to include these signals and control their flow. Fig. 2-19 shows how these devices switch between the operation and the testing mode.
Signals of the testing mode

Signals of the operation mode

T = 0 in the operation mode, the signals in the testing mode will be discharged. T = 1 in the testing mode, the signals in the operation mode will be discharged.

Fig. 2-19 The discharged circuits.
Chapter 3

TESTING

Testing: A critical examination, observation, or evaluation, ... as a means of analysis or diagnosis.

- From Webster's Dictionary

If anything can go wrong, it will.

- Murphy's Law

3.1 - Introduction

"Electronic systems, especially digital computers, have become so versatile and useful that they are indispensable in the modern society. With the electronic systems continually progressing in significance and pervasiveness of its application, testing becomes increasingly important and, with LSI/VLSI, also more and more complex and costly." (Frank F. Tsui, 1987)

LSI/VLSI circuits and systems are technical products, and in most cases also commercial products. For a technical product, both the producer and the user have a real and justified interest in knowing for a short term: "Does it work?", and over a longer time period: "Will it work next week, next month or next year as well?" (Frank F. Tsui, 1987)

The former, the capability of functioning properly "now" or "when needed", is called availability or usability
(Frank F. Tsui, 1987). The latter, that of continuing to function properly "for a long time to come", is known as reliability (Frank F. Tsui, 1987). In addition, it is also important to know: "Is it worth getting, and economical to use?". This may be referred to as cost effectiveness (which includes also the operational efficiency).

These three (availability, reliability and effectiveness) are the main characteristics of a quality technical product. To be sure of achieving quality -"for product assurance"- the producer relies on testing.

Owing to the fast progress and wide application of LSI/VLSI circuits, circumstances of testing have changed in at least two aspects:

1. "Objects to be tested have become so complex and the data associated with them so voluminous that they can no longer be handled efficiently by single individuals. This has created problems in the planning and design for testing. To overcome such problems, more and more use of computer-aid-design tools has to be relied upon." (Frank F. Tsui, 1987)

2. Circuit speed has created problems in the execution of testing and has stimulated many efforts to refine the methods and equipment.

3.2 - Conventional Test Methods

In testing processes, there apparently exist two
different perceptions, depending on whether it is seen from the viewpoint of "captive manufacturers" (those who fabricate IC's) or from that of "IC users" (those who use vendor parts to build their systems). The captive manufacturers first do the chip testing, then subassembly (module, card, and board) testing, and finally system testing. The IC's users first do incoming inspection instead of chip testing, then subassembly testing, additional I/O's, and some extra points testing. The conventional test methods have three main characteristics in common:

"1. They are usable for testing system-parts only outside of a system.

2. They rely on feeding signals directly through the test-interface during testing.

3. They rely on the use of tester-driven-timing."

(Frank F. Tsui, 1987)

With circuits and packaging technologies rapidly advancing toward higher integration and speeds, the difficulties can be grouped together into three basic problems:

"1. Uncertainties in test-parameters conformity.

2. Limitations in tester technologies and test methods.

3. Uncertainty in timing synchronization across the test interface." (Frank F. Tsui, 1987) (Fig. 3-1)

The conventional test methods are inherently impeded by
Fig. 3-1 Difficulties and basic problems in conventional testing

(From LSI/VLSI Testability Design, 1987)
these basic problems because they rely on the feeding of signals directly through the test interface.

3.3 - Design For Testability

Rapid advances in LSI/VLSI technology have created the considerable problem of testing the packages by themselves. For example, test generation time increases due to package density as shown in Fig. 3-2. Also the incorporation of LSI/VLSI into larger designs have caused the cost of test generation to grow exponentially. One approach for solving these problems is to modify the design in a way that makes test generation and diagnosis easier (design for testability). Fig. 3-3 shows the improvement achieved from design for testability.

There are two key issues in designing for testability: controllability and observability (Parag K. Lala, 1985). Controllability refers to the ability to apply test patterns to the inputs of a subcircuit via the primary inputs of the circuit. Observability refers to the ability to observe the response of a subcircuit via the primary outputs of the circuit or at some other output points. Here we use the design for testability concept, and both the controllability and observability are employed in this design.
Fig. 3-2 Density vs. test generation time (from IEEE, 1979)

Fig. 3-3 Test pattern generation cost comparison (From IEEE, 1981)
3.4 - Testing Algorithm

The purpose of testing with a testing circuit independent of any fault-assumption is to determine if the device is functioning correctly. The test cannot locate the faults, nor can it prove that the device is fault free. But if the device passes the test, the possible faults in it can be viewed as within fault tolerance and they presumably have no influence on the overall function of the device.

We use the scan path technique to achieve total or near total controllability and observability in the divider. In this approach, the flip-flops and/or latches are designed to operate in either the parallel load mode or the serial shift mode (testing mode). In the normal mode of operation, flip-flops and latches are configured for the parallel load. For test purposes, the flip-flops are switched to the serial shift mode. In the serial mode, test values can be loaded by serially clocking in and testing results can be observed from the output pins.

A simple illustration of the scan path is presented in Fig. 3-4 with a multiplexer is placed ahead of each flip-flop. One input to this 2-to-1 multiplexer is fed by normal operation data and the other input is fed by the output of the previous flip-flop. For one of the multiplexers, its serial input is connected to a primary input pin. We use OR (NOR) gates instead of multiplexers to include both the parallel and serial modes and pass transistors, which
Fig. 3-4 The scan path
control the transfer of these signals (same design as in Fig. 2-19). The OR(NOR) gates then permit the parallel load for normal operation and can select serial shift to switch to the testing mode. When the serial mode is selected, there is a complete serial shift path from an input pin.

It should be noted that there is just one scan-in input pin for all the arrays of shift registers but each array of the testing object has its own output pin. So the desired values can be clocked into the circuit at the same time (controllability) and their testing values can be read respectively (observability).

Since the circuit can load arbitrary values into the registers by means of a shift path and read the testing values out of quotient-out pins, the circuit has, in effect, been converted into a combinational circuit for testing purposes. During the testing mode, each array of the processor (PAS) can be thought of as a separate circuit. Since there are eight inputs for each array of the processors and each array of the processors executes either serial additions or subtractions (simply set $P = 0$ or $P = 1$), then each array of the processors has $2^8$ testing vectors. However, for a normalized floating-point number $B$, it must be in this form $0.1xx$ and is transferred through every stage without any change. Thus there are just $2^6$ testing vectors for each array when executing exhaustive testing. By scanning in the desired values for each shift
register, we can do the exhaustive test for each array of the processors within 64 times of scan operation. Each set of testing vectors can be completed simply by scanning in, a single operation in the normal mode, and watching the quotient bit. For example, set the testing vector to 01110101 \((B_0A_0B_1A_1B_2A_2B_3A_3)\) for A and B sets of inputs and set \(P = 0\) for addition. At the end of this test, the quotient bit should be 1 in each array. By passing this test, the device can be proved to function correctly in this situation. If we finish testing each input set with proper results, we can say that the device is functioning correctly in all situations.

In Fig. 2-11, the T input pin is set to "1" during the testing mode and the desired values are scanned in during each clock-in. The hardware structures in Fig. 2-5 and Fig. 2-11 show that the partial remainders (intermediate results) and B are shifted diagonally. But during the testing mode, these partial remainders and B are no longer shifted diagonally. For scan-in, the circuit needs eight clock pulses to let the desired values arrive at their right positions and the CLOCKT is high for every eight pulses of CLOCK in order to let the values in the registers be fed into the processors to complete a single run. The carry-out bit observed at the quotient-out \((Q_1\bar{Q}_5)\) at the end of each CLOCKT represent the testing results. Fig. 3-5 is a simplified illustration of CLOCK and CLOCKT.
The values be fed to PAS's through Reg. 1 to 8. The testing values can be observed by Q0 & Q1

Fig. 3-5
Table 3-1 shows four out of 64 test vectors for testing this divider and their expected values after the test. These 64 testing vectors provide an exhaustive test of the divider.

Note that during the testing mode, the scan-in and P inputs are connected to each row. Therefore the shift registers in each row can simultaneously clock in data and the processors can also perform the addition or subtraction simultaneously. Since the processors in the first array always execute subtraction, the testing results for \( P = 0 \) are the same as for \( P = 1 \).

Compared to the conventional testing method, this way of testing is much easier. For example, if we want to do the exhaustive testing in the conventional method, the divider has \( 2^8 \) (combination for A) by \( 2^2 \) (combination for B) testing vectors. In total, there are \( 2^{10} \) possible testing vectors and even more when using more stages.

To sum up the design of this test, the procedures should be:

1. Switching the mode select to serial shift (the testing mode);
2. Serially clocking test data into these registers using the scan input;
3. Using CLOCKT to make the divider execute once in operation mode;
4. Evaluating the test responses at the output pins.
Table 3.1 List of four of the testing vectors and results

3.5 - Timing Analysis

As mentioned before, the non-overlapped clocks, CLOCK and CLOCKT, have different periods for different modes. In the testing mode, every eight pulses of CLOCK has one CLOCKT. Although the CLOCK is used to clock-in the data, its period is closely related to the period of CLOCKT. The function of CLOCKT , in the testing mode, is to make the divider operate a single step and have testing results at the quotient-out pins. So the period of high time for CLOCKT
in the testing mode is the same as for CLOCK in the operation mode. The time required is determined by CRYSTAL. CLOCK and CLOCKT do not overlap and the period of CLOCKT must be long enough to scan-in the testing vector as illustrated in Fig. 3-6. The exact intervals required for CLOCK and CLOCKT are presented in chapter four.
Fig. 3-6 Illustration of the CLOCK and CLOCKT in the testing mode
"When a system architecture or logic network is designed, performance and errors are checked by CAD programs." (Saburo Muroga, 1982) This is called logic simulation, since CAD programs are to check simulation to see whether the designed systems or networks are realized. In the 1st and 2nd section of this chapter, there is a brief introduction of the design strategy and CAD design tools as well as a description of the system. Section 3 explains the software packages of the simulation for verifying this design. The last section discusses these results.

4.1 - Design Strategy and CAD Design tools

Computers have been extensively used in all stages of design and development of an LSI/VLSI system, starting from system specifications to the test of prototypes. The objectives of developing computer-aided design (CAD) programs are to shorten the design and development time of LSI/VLSI chips, to minimize design errors, to facilitate design changes, and to shorten the time for design verifications and tests (Saburo Muroga, 1982). A basic design process (or strategy) used for LSI/VLSI chips is diagrammed in Fig. 4-1. The advantage of this design process is a library of predesigned (and hierarchical) chip parts or
INPUT LOGIC DESCRIPTION
AND TEST VECTORS

SIMULATION

LAYOUT CHIP

LAYOUT CHECKING

POST LAYOUT

TEST PROGRAM

(LIBRARY CELLS)

OUTPUT FOR MASKS, FOR FABRICATION & FOR TESTING

Fig. 4-1 A basic design process of LSI/VLSI circuits

(Guy Rabbat, 1983)
cells. This includes logic primitives and function cells for use by the logic designer and chip designer. The logic designer designs and simulates circuits in terms of the gate-level cells or functions. The resultant gate-level logic description maps directly into the library cell information for chip layout, for simulation, and for generation of mask information. The chip design process described produces a large number of well-designed cells and/or subcircuits.

The remaining post-layout steps (Fig. 4-1) of generating the mask data and the manufacturing information are not the design task. They are derived directly from logic description and chip layout. The post-layout steps are data processing to produce specific design outputs needed for the physical processing and fabrication (Guy Rabbat, 1983).

The chip design process aided with computers is independent of changes in silicon technology. For a new technology or a change in design rules, the new cells are first designed and verified and put in the library as new cells. The chip design process can be independently and continuously improved to reduce the chip design time and cost as well as the use of silicon technology (Guy Rabbat, 1983).

The evolving CAD design tools for all LSI/VLSI circuits are a set of programs that are primarily design task
oriented, independent of circuit form and of silicon technology. These CAD designers organize themselves and their work in a modular adaptive way to cope with the needs of increasing chip complexity, design volume, and the changing silicon and computer technologies.

4.2 - SCALDstar system

The CAD design tools use in this project are the SCALDstar system and VAX11/750 minicomputer, both having UNIX as their operating system. SCALDstar is an integrated system used for schematic design and layout of VLSI circuits. By incorporating design, layout, and validation tools on a single system, SCALDstar supports the IC designer from concept to mask design. SCALDstar uses two graphic CRT displays, a monochromatic display for operating and a color display for physical layout. The SCALDstar system comes with a complete set of sophisticated software tools for logic design. The system uses easy-to-understand and convenient menus, simple and effective commands, and on-line help facilities. Among software tools that we use are the followings:

1. LED: The layout editor (LED) creates the topological description of the design. By manipulating a trapezoidal cursor and painting colors under that cursor, the designer defines the layout of the circuit.

2. DRC AND EXTRACT: This single SCALDstar program
allows the designer to extract a connectivity of the layout and perform design rule checks (DRC) on that layout. When extracting the connectivity description, DRC locates transistors and determines their width and length. DRC then extracts gate and interconnect capacitance, taking into account area and fringe effects. Because cells are designed hierarchically, as the size of the layout extracted or checked by the DRC increases, the CPU time required increases linearly rather than exponentially.

3. ERC: The ERC program checks for electrical rule violations in the design. Using the extracted connectivity description, ERC detects problems such as transistors connected directly between supply and ground and improper sizing of transistors.

4. MAKEPLOT: This program is used to generate a plotter output of a cell created with the Layout Editor. The makeplot program can be used with HP pen plotters, Versatec and Benson black and white plotters, and the Versatec color plotter. Fig. 4-2 shows the SCALDstar programs and their interrelationship.

4.3 - Simulation

After layout has been implemented using SCALDstar computer-aided design Layout Editor, several verification steps are used to check this design. These steps and their functions are introduced as below and their results are
Fig. 4-2 Interrelationship of SCALDstar programs
presented in the next section.

1. DRC/EXTRACT: DRC and EXTRACT are two programs which use layouts as their source files. There are two main purposes for using DRC. One is to find structures that are difficult to be fabricated. The second purpose is to find the structures that are not practical even though they can be fabricated. EXTRACT is used to create a complete listing of all components of a layout. This list can drive several other simulation programs.

   The DRC/EXTRACT is executed in SCALDstar through a simple LED command by typing

   drc [command_file] &
   extract [command_file]

2. Creating database in CIF: The purpose of generating a CIF file, which describes the geometry of the layout, is to transfer this layout to VAX and to obtain data for chip fabrication. The CIF file is generated using the LED command: CIF or CIF -p, where -p lets the program provide information about signals.

3. Transferring file from the SCALDstar system to VAX: Some simulation programs like ESIM, CRYSTAL, and POWEST are not available in SCALDstar. The transfer of this design file becomes necessary. The steps to achieve this task are listed in Appendix I. Some other steps like modifying the transferred file before and after transformation, generating the simulation drive programs, and disconnection of the
SCALDstar and VAX are also included in Appendix I.

4. ESIM: ESIM, a logic simulator, is used to verify the design at the logic level. The basic model of this simulator consists of a set of nodes and transistors. Each node could be in one of the states (0, 1, x). The nodes are classified as input nodes which accept signals from outside the chip and observed nodes whose values can be used to check if the design is logically correct.

5. CRYSTAL—timing analysis: The purpose of the CRYSTAL program is to analyze the timing characteristic and performance of an LSI/VLSI circuit. CRYSTAL helps to find the paths that limit the clock speed and the worst-case delay. This simulator considers all possible input values in the analysis of a circuit configuration.

6. POWEST: This program helps to estimate the power consumption of an LSI/VLSI circuit by typing powest < [filename.sim].

4.4 - Discussions

An LSI/VLSI circuit needs to be verified by simulation programs after layout is completed. In this section, the simulation results and discussions are presented in a sequential order.

A design layout must conform to the design rules. A DRC software is used to check if all the design rules have been met by the design layout. A hierarchical structure is used
so that the time needed for DRC increases much slower than that of a flat structure when the size of the design layout becomes large. The results of the DRC are shown in Appendix II, proving that the design meets all the design rule requirements.

ESIM is an event-driven switch level simulator for nMOS transistors. ESIM verifies the system response on the logic level. The simulation is done in two parts, the operation mode and the testing mode as described below.

A. Operation mode simulation

(1) Basic process simulation: Exhaustive simulation is performed for a basic processor, PAS. There are two inputs, A and B, for PAS and a functional control input P. When \( P = 0 \), PAS executes addition, and when \( P = 1 \), PAS executes subtraction. The command file and results are shown in Appendix III part A (1).

(2) Overall simulation: The overall simulation is done by using the same example in page eight. This simulation generates 0.1011 while performing 0.1001001 / 0.111. The command file, results, and three other additional examples of overall simulation are shown in Appendix III part A (2). The divider generates 0.0100, 0.0101, and 0.1101 while performing 0.011011 / 0.111, 0.0101001 / 0.111 and 0.0110101 / 0.100 respectively.

B. Testing mode simulation

This simulation verifies the function of the testing...
circuit and the simulation results are shown for one stage.

(1) The scan path: In order to verify the function of the scan path, the values are first scanned in and then they are scanned out. The values scanned in are 00111011 in Reg.1 to Reg.8. The values in the scan-out should be 11011100 sequence. The command file and results are shown in Appendix III part B (1).

(2) Testing: Two examples of the simulation of the testing vectors are shown in this part. The scanned in binary strings are 00111011 and 01111111. Both addition and subtraction are executed in each case. The command file and results are shown in Appendix III part B (2).

CRYSTAL, a timing simulator, is used to determine the performance of an LSI/VLSI circuit. The simulation is done in one stage of adders/subtractors since the completion of a series of four additions/subtractions limits the clock speed. The command file and results are shown in Appendix IV. The worst-case delay is 67.57 ns and the critical path is from P to the Ci input of the rightmost PAS and to the propagation of the carry from rightmost to leftmost in one stage. The CLOCK needs to be high during this period. The CLOCKT is used to latch the intermediate results and is a non-overlapped clock with respect to CLOCK. Assuming the hold time of CLOCKT is 5ns and the non-overlapped high interval between CLOCK and CLOCKT is 10ns, the periods and frequencies of the CLOCK and CLOCKT are calculated as shown
\[ T_{\text{CLOCK}} = 67.57 \times 5 \times 10 + 10 = 92.57 \text{ ns} \]

\[ F_{\text{CLOCK}} = \frac{1}{92.57} = 10.8 \text{ MHz} \]

\[ T_{\text{CLOCKT}} = 5 \times 10 + 10 \times 67.57 = 92.57 \text{ ns} \]

\[ F_{\text{CLOCKT}} = \frac{1}{92.57} = 10.8 \text{ MHz} \]

Fig. 4-3 (a) Illustration of the clocks in the operation mode
\[ T_{\text{CLOCK}} = 5 \times 10 + 10 + 67.57 = 92.57 \text{ ns} \]

\[ F_{\text{CLOCK}} = \frac{1}{92.57} = 10.8 \text{ MHz} \]

\[ T_{\text{CLOCKT}} = 92.57 \times 8 = 740.56 \text{ ns} \]

\[ F_{\text{CLOCKT}} = \frac{1}{740.56} = 1.3 \text{ MHz} \]

Fig. 4-3 (b) Illustration of the clocks in the testing mode
In the testing mode, the required high interval of CLOCKT is the same as the CLOCK since the CLOCKT let the divider execute its function within one stage. By similar assumption of the non-overlapped high clocks and hold time, the required periods and frequencies of CLOCK and CLOCKT are calculated and shown in Fig. 4-3 (b).

Another more accurate timing simulator is SPICE, which is frequently used to determine the performance of small circuits. SPICE allows more detailed models of transistors than other timing simulators, but the execution time is almost two orders of magnitude proportional to the size of the circuit. It is unrealistic to use SPICE for verification of LSI/VLSI chips.

Of all the LSI/VLSI design styles, the stand cell design should have a high performance because of its highly regular structure and simple interconnections (or no interconnection). In this design, the performance of the divider is influenced by the built-in testing circuits. In order to prove this, the CRYSTAL package is run by using the divider without the discharged circuits ahead of the inputs of each NOR gate. The worst-case delay within one stage is 41 ns. This is 26 ns less than that of the divider with discharged circuits. If the divider is designed without any built-in testing circuits, the performance of operation is up to 15 MHz or more. Therefore, there is a trade off
between performance and built-in testing circuits.

The power consumption of this divider is 125 mW for the total design, which is evaluated by POWEST on VAX. By using this value, the global VDD and GND width are calculated. Since \( W \) (power) = \( V \) (voltage) \( \times I \) (current), 125 mW maximum power dissipation would cause 25 mA current runs through global metal line.

Taking \( \lambda = 2 \) \( \mu \)m,

\[
\text{current density of the metal} = 1 \text{ mA/\( \mu \)m}
\]

\[
25 / 2 = 12.5
\]

the global VDD and GND width should be 12.5 \( \lambda \). This number almost matches the required global metal width calculated on page fifteen.
This thesis presents a design of a floating-point divider which exploits a combination of parallelism, LSI/VLSI design, and built-in testing. Because of the regular structure and interconnections of the divider, it can be easily extended to perform more-bit division and obtain more precise results simply by using more cells in an array and having more stages in the perpendicular direction. This extension ability is an important factor in designing an LSI/VLSI system.

Since the built-in testing circuits are considered simultaneously with the designed function, it is easy to verify the whole circuit when system becomes large. If the system is extended, the performance may or may not be degraded depending on the type of the expansion. For example, to extend it to an 8 by 8 divider, the worst case delay will increase about two times since there are a series of eight PAS's in horizontal direction now. However, if the system with more precise results is desired, there is no influence on the worst case delay.

Implementing parallel algorithm in hardware is efficient by using LSI/VLSI technology for its homogeneity and modularity in cellular structure. As hardware cost declines and software cost increases, more and more hardware
systems are replacing the software algorithms. This trend is supported by the increasing demand for design for testability.

The performance of this divider is degraded by the testing circuits. The future development in testing may achieve the full automation test, then the system diagnosis, debug and maintenance without significant influence on the testing object itself.

The possible future upgrading of this design is to build a multifunction pipelined machine, which can perform multiply, division, squaring, and sqrt. operations and has similar structure and process elements (Kamal et al, 1974). These four functions can be performed by adders/subtractors with bypass signal lines (similar to PAS). The cells TEMPO's designed in this thesis can be extended to have function selection and boundary signal control. Some extra control lines need to be added. Testing circuits can be built in by using the technique discussed in this thesis. However the performance of this 4-function machine may be degraded because of the more complex cellular structure and interconnections.
APPENDIX

Appendix I COMMUNICATION BETWEEN THE SCALD SYSTEM AND VAX.73
Appendix II DESIGN RULE CHECK RESULTS.......................76
Appendix III SIMULATION RESULTS.................................78
Appendix IV CRYSTAL RESULTS......................................87
Appendix I

COMMUNICATION BETWEEN THE SCALD SYSTEM AND VAX

A. Transferring

1. The connection between SCALDstar system and VAX are accomplished using the script:

```
validcom 300 on VAX
```

Where 300 is the optional bits/sec baud rate, the baud rate defaults to 1200 bits/sec. The connection would not be achieved if the default was used.

2. Login to the user directory in SCALDstar system.

3. Up to this point, the user directory in SCALDstar system has been accessed. A design file can be transferred to VAX using the script:

```
t fnl fn2
```

(Notice after you have typed "" and then "t", the argument "take" appears on the screen.)

Where fnl is the design file which is sent.

fn2 is a given file name for fnl in VAX.

4. To disconnect the VAX from the SCALDstar system and get back to VAX, logout the SCALDstar and then type:

```
.
```

B. Modifying

1. Modifying the LED-generated CIF design file

The valid2vax program performs modifying the CIF design to
appropriate for the Mextra Circuit Extraction program before
the transferring. The following argument is used to invoke
the program:

    valid2vax in CSh%

Where fnl is the design file.

2. Eliminating a non-printable ascii character

The design files, which have been transferred to the
VAX, are always attached to each line by a non-printable
ascii character. However, the characters can be seen on the
screen if the design file is edited using "vi" command
(visual display editor). Eliminating the non-printable ascii
character can be attached using the script:

    validfix fn2

3. Circuit Extraction Using The Mextra Circuit
Extraction Program

Mextra (Manhattan Circuit Extraction for VLSI Simulation) performs by reading the design file basename.cif
and creating following files:

1) basename.log - contains a count of the number of
transistors, the number of nodes, and messages about
possible errors.

2) basename.nodes - is a list of node names and their
CIF locations.

3) basename.al - is a list of aliases which can be used
by esim (event switch level simulation).

4) basename.sim - is the circuit description, which is
a list of transistors and capacitors. It is applied with some simulation programs such as ESIM (a gate level simulator), CRYSTAL (a static timing verifier), POWEST (a dc power estimation program), and electrical rule checker (ERC).
Appendix II
DESIGN RULE CHECK RESULTS

DRC is a LED command by typing drc in led or typing drc filename > filename in csh%. My design employs hierarchical structure. The DRC checks all the subcircuits automatically. The computer print-out of the result is attached as follows:

Apr 2 1983 1988 drcresult Page 1

updatina /u0/lyu/tempo1/abstract.1.1
doina drc tests
done. Spent 198.204 seconds on TEMPO1, version 1
   ****************************  ****************************
   *    0 Errors    *  0 Edge Errors    *
   ****************************  ****************************
Total time: 207.050 seconds on TEMPO1 version 1

updatina /u0/lyu/tempo2/abstract.1.1
doina drc tests
done. Spent 200.203 seconds on TEMPO2, version 1
   ****************************  ****************************
   *    0 Errors    *  0 Edge Errors    *
   ****************************  ****************************
Total time: 209.183 seconds on TEMPO2 version 1

File /u0/lyu/tempo2/abstract.1.1 does not need to be updated
doina drc tests
done. Spent 104.525 seconds on TEMPO2, version 1
   ****************************  ****************************
   *    0 Errors    *  0 Edge Errors    *
   ****************************  ****************************
Total time: 112.305 seconds on TEMPO2 version 1

updatina /u0/lyu/cellu/abstract.1.1
doina drc tests
done. Spent 640.544 seconds on CELLU, version 1
   ****************************  ****************************
   *    0 Errors    *  0 Edge Errors    *
   ****************************  ****************************
Total time: 652.355 seconds on CELLU version 1
file /u0/lyu/cell/abstract.1.1 does not need to be updated
doing drc tests
done. Spent 679.811 seconds on CELL, version 1
* 0 Errors * 0 Edge Errors *
*************** *********************
Total time: 693.489 seconds on CELL version 1

updating /u0/lyu/thesis/abstract.1.1
doing drc tests
done. Spent 517.460 seconds on thesis, version 1
* 0 Errors * 0 Edge Errors *
*************** *********************
Total time: 2491.036 seconds on thesis version 1
Appendix III

SIMULATION RESULTS

A. Operation mode simulation

(1) PAS command file  

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Initialization took 77 steps</td>
<td></td>
</tr>
<tr>
<td>h TBAR CLK</td>
<td>step took 19 events</td>
<td></td>
</tr>
<tr>
<td>h Ci P</td>
<td>Ci+1=1 S/D=0 Ci=1 B=1 A=1 P=1</td>
<td></td>
</tr>
<tr>
<td>h A B</td>
<td>step took 14 events</td>
<td></td>
</tr>
<tr>
<td>w F</td>
<td>Ci+1=0 S/D=1 Ci=1 R=1 A=0 P=1</td>
<td></td>
</tr>
<tr>
<td>w A</td>
<td>step took 15 events</td>
<td></td>
</tr>
<tr>
<td>w B</td>
<td>Ci+1=1 S/D=0 Ci=1 B=0 A=0 P=1</td>
<td></td>
</tr>
<tr>
<td>w Ci</td>
<td>step took 12 events</td>
<td></td>
</tr>
<tr>
<td>w S/D</td>
<td>Ci+1=1 S/D=1 Ci=1 R=0 A=1 P=1</td>
<td></td>
</tr>
<tr>
<td>w Ci+1</td>
<td>step took 13 events</td>
<td></td>
</tr>
<tr>
<td>s l A</td>
<td>Ci+1=0 S/D=1 Ci=0 R=0 A=1 P=0</td>
<td></td>
</tr>
<tr>
<td>s l B</td>
<td>step took 13 events</td>
<td></td>
</tr>
<tr>
<td>s h A</td>
<td>Ci+1=0 S/D=1 Ci=0 B=0 A=0 P=0</td>
<td></td>
</tr>
<tr>
<td>s l Ci P</td>
<td>step took 10 events</td>
<td></td>
</tr>
<tr>
<td>s l A</td>
<td>Ci+1=1 S/D=0 Ci=0 R=1 A=1 P=0</td>
<td></td>
</tr>
<tr>
<td>s h B</td>
<td>step took 13 events</td>
<td></td>
</tr>
<tr>
<td>s h A</td>
<td>s quit</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** The commands are used to simulate different scenarios, and the results show the state transitions for each scenario.
(2) Overall simulation command file

<table>
<thead>
<tr>
<th>Results</th>
<th>Initialization took 3516 steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>step</td>
<td>took 805 events</td>
</tr>
<tr>
<td>step</td>
<td>took 368 events</td>
</tr>
<tr>
<td>step</td>
<td>took 0 events</td>
</tr>
<tr>
<td>Q0=0</td>
<td></td>
</tr>
<tr>
<td>step</td>
<td>took 104 events</td>
</tr>
<tr>
<td>Q0=0</td>
<td></td>
</tr>
<tr>
<td>step</td>
<td>took 157 events</td>
</tr>
<tr>
<td>Q0=0</td>
<td></td>
</tr>
<tr>
<td>step</td>
<td>took 360 events</td>
</tr>
<tr>
<td>Q0=0</td>
<td></td>
</tr>
<tr>
<td>step</td>
<td>took 168 events</td>
</tr>
<tr>
<td>Q0=0</td>
<td></td>
</tr>
<tr>
<td>step</td>
<td>took 0 events</td>
</tr>
<tr>
<td>Q1=1</td>
<td>Q0=0</td>
</tr>
<tr>
<td>step</td>
<td>took 93 events</td>
</tr>
<tr>
<td>Q1=1</td>
<td>Q0=0</td>
</tr>
<tr>
<td>step</td>
<td>took 157 events</td>
</tr>
<tr>
<td>Q1=1</td>
<td>Q0=0</td>
</tr>
<tr>
<td>step</td>
<td>took 206 events</td>
</tr>
<tr>
<td>Q1=1</td>
<td>Q0=0</td>
</tr>
<tr>
<td>step</td>
<td>took 167 events</td>
</tr>
<tr>
<td>Q1=1</td>
<td>Q0=0</td>
</tr>
<tr>
<td>step</td>
<td>took 0 events</td>
</tr>
</tbody>
</table>
| Q2=0    | Q1=1                          | Q0=0
| step    | took 88 events                |
| Q2=0    | Q1=1                          | Q0=0
| step    | took 157 events               |
| Q2=0    | Q1=1                          | Q0=0
| step    | took 292 events               |
| Q2=0    | Q1=1                          | Q0=0
| step    | took 168 events               |
| Q2=0    | Q1=1                          | Q0=0
| step    | took 0 events                 |
| Q2=0    |                                |
| step    | took 88 events                |
| Q3=1    | Q2=0                          | Q1=1 | Q0=0
| step    | took 156 events               |
| Q3=1    | Q2=0                          | Q1=1 | Q0=0
| step    | took 190 events               |
| Q3=1    | Q2=0                          | Q1=1 | Q0=0
| step    | took 168 events               |
| Q3=1    | Q2=0                          | Q1=1 | Q0=0
| step    | took 0 events                 |
| Q3=1    | Q2=0                          | Q1=1 | Q0=0
| step    | took 0 events                 |
| Q4=1    | Q3=1                          | Q2=0 | Q1=1 | Q0=0

quit
Three additional input sets for the simulation are shown in this part. The divider generates $0.0100$, $0.0101$, and $0.1101$ while performing $0.0110101 / 0.111$, $0.0101001 / 0.111$, and $0.0110101 / 0.100$ respectively.
<table>
<thead>
<tr>
<th>Command file</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>I h A3 A4 A6 A8 B2 B4 R3 I A1 A5 A7 B1 I A2</td>
<td>initialization took 3516 steps</td>
</tr>
<tr>
<td>h CLOCK</td>
<td>step took 691 events</td>
</tr>
<tr>
<td>s</td>
<td>step took 368 events</td>
</tr>
<tr>
<td>l CLOCK</td>
<td>step took 0 events</td>
</tr>
<tr>
<td>s</td>
<td>Q0=0</td>
</tr>
<tr>
<td>w R0</td>
<td>step took 106 events</td>
</tr>
<tr>
<td>s</td>
<td>Q0=0</td>
</tr>
<tr>
<td>h CLOCKT</td>
<td>step took 157 events</td>
</tr>
<tr>
<td>s</td>
<td>Q0=0</td>
</tr>
<tr>
<td>l CLOCKT</td>
<td>step took 303 events</td>
</tr>
<tr>
<td>s</td>
<td>Q0=0</td>
</tr>
<tr>
<td>h CLOCK</td>
<td>step took 167 events</td>
</tr>
<tr>
<td>s</td>
<td>Q0=0</td>
</tr>
<tr>
<td>l CLOCK</td>
<td>step took 0 events</td>
</tr>
<tr>
<td>s</td>
<td>Q1=0 Q0=0</td>
</tr>
<tr>
<td>w Q1</td>
<td>step took 93 events</td>
</tr>
<tr>
<td>s</td>
<td>Q1=0 Q0=0</td>
</tr>
<tr>
<td>h CLOCKT</td>
<td>step took 157 events</td>
</tr>
<tr>
<td>s</td>
<td>Q1=0 Q0=0</td>
</tr>
<tr>
<td>l CLOCKT</td>
<td>step took 514 events</td>
</tr>
<tr>
<td>s</td>
<td>Q1=0 Q0=0</td>
</tr>
<tr>
<td>h CLOCK</td>
<td>step took 169 events</td>
</tr>
<tr>
<td>s</td>
<td>Q1=0 Q0=0</td>
</tr>
<tr>
<td>l CLOCK</td>
<td>step took 0 events</td>
</tr>
<tr>
<td>s</td>
<td>Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>w Q2</td>
<td>step took 89 events</td>
</tr>
<tr>
<td>s</td>
<td>Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>h CLOCKT</td>
<td>step took 155 events</td>
</tr>
<tr>
<td>s</td>
<td>Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>l CLOCKT</td>
<td>step took 214 events</td>
</tr>
<tr>
<td>s</td>
<td>Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>h CLOCK</td>
<td>step took 171 events</td>
</tr>
<tr>
<td>s</td>
<td>Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>l CLOCK</td>
<td>step took 0 events</td>
</tr>
<tr>
<td>s</td>
<td>Q3=0 Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>w Q3</td>
<td>step took 85 events</td>
</tr>
<tr>
<td>s</td>
<td>Q3=0 Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>h CLOCKT</td>
<td>step took 156 events</td>
</tr>
<tr>
<td>s</td>
<td>Q3=0 Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>l CLOCKT</td>
<td>step took 173 events</td>
</tr>
<tr>
<td>s</td>
<td>Q3=0 Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>h CLOCK</td>
<td>step took 167 events</td>
</tr>
<tr>
<td>s</td>
<td>Q3=0 Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>l CLOCK</td>
<td>step took 0 events</td>
</tr>
<tr>
<td>s</td>
<td>Q4=0 Q3=0 Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>w Q4</td>
<td>step took 84 events</td>
</tr>
<tr>
<td>s</td>
<td>Q4=0 Q3=0 Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>s</td>
<td>step took 156 events</td>
</tr>
<tr>
<td>s</td>
<td>Q4=0 Q3=0 Q2=1 Q1=0 Q0=0</td>
</tr>
</tbody>
</table>
Command file

<table>
<thead>
<tr>
<th>h A8 A3 A5 R2 R3 B4</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>h A2 A4 A7 A6 A1 B1 T</td>
<td>initialization took 3516 steps</td>
</tr>
<tr>
<td>h CLOCK s</td>
<td>step took 702 events</td>
</tr>
<tr>
<td>l CLOCK s</td>
<td>step took 368 events</td>
</tr>
<tr>
<td>s</td>
<td>step took 0 events</td>
</tr>
<tr>
<td>w Q0</td>
<td>Q0=0</td>
</tr>
<tr>
<td>s</td>
<td>step took 157 events</td>
</tr>
<tr>
<td>h CLOCK T</td>
<td>Q0=0</td>
</tr>
<tr>
<td>s</td>
<td>step took 311 events</td>
</tr>
<tr>
<td>s</td>
<td>Q0=0</td>
</tr>
<tr>
<td>h CLOCK s</td>
<td>step took 167 events</td>
</tr>
<tr>
<td>s</td>
<td>Q0=0</td>
</tr>
<tr>
<td>l CLOCK s</td>
<td>step took 0 events</td>
</tr>
<tr>
<td>s</td>
<td>Q1=0 Q0=0</td>
</tr>
<tr>
<td>w Q1</td>
<td>Q1=0 Q0=0</td>
</tr>
<tr>
<td>s</td>
<td>step took 157 events</td>
</tr>
<tr>
<td>h CLOCK T</td>
<td>Q1=0 Q0=0</td>
</tr>
<tr>
<td>s</td>
<td>step took 169 events</td>
</tr>
<tr>
<td>s</td>
<td>Q1=0 Q0=0</td>
</tr>
<tr>
<td>h CLOCK s</td>
<td>step took 0 events</td>
</tr>
<tr>
<td>s</td>
<td>Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>l CLOCK s</td>
<td>step took 91 events</td>
</tr>
<tr>
<td>s</td>
<td>Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>w Q2</td>
<td>Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>s</td>
<td>step took 156 events</td>
</tr>
<tr>
<td>h CLOCK T</td>
<td>Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>s</td>
<td>step took 199 events</td>
</tr>
<tr>
<td>s</td>
<td>Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>l CLOCK T</td>
<td>step took 171 events</td>
</tr>
<tr>
<td>s</td>
<td>Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>h CLOCK s</td>
<td>step took 0 events</td>
</tr>
<tr>
<td>s</td>
<td>Q3=0 Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>l CLOCK s</td>
<td>step took 87 events</td>
</tr>
<tr>
<td>s</td>
<td>Q3=0 Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>w Q3</td>
<td>Q3=0 Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>s</td>
<td>step took 157 events</td>
</tr>
<tr>
<td>h CLOCK T</td>
<td>Q3=0 Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>s</td>
<td>step took 188 events</td>
</tr>
<tr>
<td>l CLOCK T</td>
<td>Q3=0 Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>s</td>
<td>step took 168 events</td>
</tr>
<tr>
<td>h CLOCK s</td>
<td>Q3=0 Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>s</td>
<td>step took 0 events</td>
</tr>
<tr>
<td>l CLOCK s</td>
<td>Q4=1 Q3=0 Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>s</td>
<td>step took 85 events</td>
</tr>
<tr>
<td>w Q4</td>
<td>Q4=1 Q3=0 Q2=1 Q1=0 Q0=0</td>
</tr>
<tr>
<td>s</td>
<td>step took 156 events</td>
</tr>
</tbody>
</table>
Command file

I
h A3 A4 A6 A8 B2
l A1 A2 A5 A7 B1 T B4 B3
h CLOCK
s
h CLOCK
s
l CLOCK
s
w Q0
h CLOCK
s
l CLOCK
s
w Q1
s
h CLOCK
s
l CLOCK
s
w Q2
s
h CLOCK
s
l CLOCK
s
w Q3
s
h CLOCK
s
l CLOCK
s
w Q4

Results

initialization took 3516 steps
step took 208 events
step took 1523 events
step took 227 events
step took 0 events
Q0=0
step took 249 events
Q0=0
step took 189 events
Q0=0
step took 168 events
Q0=0
step took 0 events
Q1=1 Q0=0
step took 84 events
Q1=1 Q0=0
step took 155 events
Q1=1 Q0=0
step took 174 events
Q1=1 Q0=0
step took 167 events
Q1=1 Q0=0
step took 0 events
Q2=1 Q1=1 Q0=0
step took 83 events
Q2=1 Q1=1 Q0=0
step took 155 events
Q2=1 Q1=1 Q0=0
step took 173 events
Q2=1 Q1=1 Q0=0
step took 167 events
Q2=1 Q1=1 Q0=0
step took 0 events
Q3=0 Q2=1 Q1=1 Q0=0
step took 83 events
Q3=0 Q2=1 Q1=1 Q0=0
step took 155 events
Q3=0 Q2=1 Q1=1 Q0=0
step took 173 events
Q3=0 Q2=1 Q1=1 Q0=0
step took 167 events
Q3=0 Q2=1 Q1=1 Q0=0
step took 0 events
Q4=1 Q3=0 Q2=1 Q1=1 Q0=0
step took 83 events
Q4=1 Q3=0 Q2=1 Q1=1 Q0=0
step took 155 events
Q4=1 Q3=0 Q2=1 Q1=1 Q0=0
B. Testing mode simulation

(1) Scan path

Scan path command file

Scan path simulation results

- Initialization took 3655 steps
- Step took 762 events
- Step took 129 events
- Step took 132 events
- Step took 127 events
- Step took 133 events
- Step took 152 events
- Step took 133 events
- Step took 177 events
- Step took 132 events
- Step took 177 events
- Step took 132 events
- Step took 132 events
- Step took 202 events
- Step took 132 events
- Step took 202 events
- Step took 132 events
- Step took 0 events
- SCANOUT1=1
- Step took 132 events
- SCANOUT1=1
- Step took 202 events
- SCANOUT1=1
- Step took 132 events
- SCANOUT1=1
- Step took 197 events
- SCANOUT1=0
- Step took 132 events
- SCANOUT1=0
- Step took 172 events
- SCANOUT1=1
- Step took 132 events
- SCANOUT1=1
- Step took 152 events
- SCANOUT1=1
- Step took 132 events
- SCANOUT1=1
- Step took 152 events
- SCANOUT1=1
- Step took 132 events
- SCANOUT1=1
- Step took 132 events
- SCANOUT1=1
- Step took 132 events
- SCANOUT1=0
- Step took 127 events
- SCANOUT1=0
(2) Testing

Testing vector 00111011

<table>
<thead>
<tr>
<th>Command File</th>
<th>Simulation Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>I 1 P</td>
<td>Initialization took 3655 steps</td>
</tr>
<tr>
<td>h T s</td>
<td>Step took 762 events</td>
</tr>
<tr>
<td>h SCANIN 1 CLOCKT s</td>
<td>Step took 129 events</td>
</tr>
<tr>
<td>h CLOCK s w Q1 s w P</td>
<td>Step took 132 events</td>
</tr>
<tr>
<td>1 CLOCK s h P</td>
<td>Step took 127 events</td>
</tr>
<tr>
<td>h CLOCK h CLOCKT s</td>
<td>Step took 133 events</td>
</tr>
<tr>
<td>1 CLOCK s h CLOCKT s</td>
<td>Step took 152 events</td>
</tr>
<tr>
<td>1 SCANIN 1 CLOCKT s</td>
<td>Step took 133 events</td>
</tr>
<tr>
<td>h CLOCK s</td>
<td>Step took 177 events</td>
</tr>
<tr>
<td>1 CLOCK s</td>
<td>Step took 132 events</td>
</tr>
<tr>
<td>1 SCANIN 1 CLOCKT s</td>
<td>Step took 177 events</td>
</tr>
<tr>
<td>h CLOCK s</td>
<td>Step took 6 events</td>
</tr>
<tr>
<td>1 CLOCK s</td>
<td>Step took 132 events</td>
</tr>
<tr>
<td>1 SCANIN 1 CLOCKT s</td>
<td>Step took 202 events</td>
</tr>
<tr>
<td>h CLOCK s</td>
<td>Step took 202 events</td>
</tr>
<tr>
<td>1 CLOCK s</td>
<td>Step took 241 events</td>
</tr>
<tr>
<td>1 CLOCK s</td>
<td>Step took 1 events</td>
</tr>
<tr>
<td>h CLOCK s</td>
<td>Step took 501 events</td>
</tr>
<tr>
<td>1 CLOCK s</td>
<td>Step took 0 events</td>
</tr>
<tr>
<td>P=0 Q1=0</td>
<td>$P=0 \ \ \ Q1=0$</td>
</tr>
<tr>
<td>1 CLOCK s</td>
<td>Step took 112 events</td>
</tr>
<tr>
<td>h CLOCK s</td>
<td>Step took 179 events</td>
</tr>
<tr>
<td>P=1 Q1=0</td>
<td>$P=1 \ \ \ Q1=0$</td>
</tr>
<tr>
<td>1 CLOCK s</td>
<td>Step took 177 events</td>
</tr>
<tr>
<td>1 SCANIN s h CLOCK s</td>
<td>Step took 177 events</td>
</tr>
<tr>
<td>1 CLOCK s h CLOCK s</td>
<td>Step took 177 events</td>
</tr>
<tr>
<td>1 CLOCK s h CLOCKT s</td>
<td>Step took 177 events</td>
</tr>
</tbody>
</table>
Testing vector 0111111

command file

I  w  Q1
h  T  w  P
h  SCANIN  s
h  CLOCK  h  P
s  h  CLOCKT
l  CLOCK  s
l  CLOCKT
h  CLOCK  s
l  CLOCK  s
h  CLOCK  s
l  CLOCK  s
h  CLOCK  s
l  CLOCK  s
h  CLOCK  s
l  CLOCK  s
h  CLOCK  s
l  CLOCK  s
h  CLOCK  s
l  SCANIN  h  CLOCK  s
l  CLOCK  s
h  CLOCKT  s
l  CLOCKT  s
l  P

initialization took 3655 steps

step took 762 events
step took 129 events
step took 132 events
step took 127 events
step took 132 events
step took 127 events
step took 132 events
step took 127 events
step took 132 events
step took 127 events
step took 132 events
step took 127 events
step took 132 events
step took 127 events
step took 133 events
step took 152 events
step took 290 events
step took 496 events
step took 1 events
P=0 Q1=1
step took 105 events
P=1 Q1=1
step took 177 events
P=1 Q1=1
Appendix IV

CRYSTAL RESULTS

CRYSTAL is used to determine the performance of this design. This simulation is done by a serial of four adders/subtracters (one stage). Once the inputs, outputs, and clocks are specified, CRYSTAL will consider all possible combinations and calculate the worst-case delay. The command file and results are shown in part A and B respectively.

A. CRYSTAL command file

```
inputs A1 A2 A3 A4 B1 B2 B3 B4 I P CLOCK CLOCKT
output Q
delay CLOCK 0 -1
critical dum2
clear
delay CLOCKT 0 -1
critical dum3
quit
```
B. Results

The results have two parts. First, CLOCK is driven high at 0.00ns, and second, CLOCKT is driven high at 0.00ns. The author chooses the first one because the CLOCK needs to be high first in this design.

Node 199 is driven low at 67.57ns through fet at (-329, 172) to GND after
192 is driven high at 67.26ns through fet at (-325, 195) to Q
through fet at (-290, 178) to Vdd after
217 is driven low at 66.36ns through fet at (-260, 172) to GND after
189 is driven high at 66.13ns through fet at (-263, 179) to 259
through fet at (-155, 177) to Vdd after
229 is driven low at 64.57ns through fet at (-29, 224) to 770

778 is driven high at 9.19ns through fet at (-73, 222) to 855
through fet at (-76, 222) to 653
through fet at (-82, 224) to Vdd after
801 is driven low at 5.44ns through fet at (-89, 223) to 808
through fet at (-96, 222) to GND after
550 is driven high at 4.73ns through fet at (-129, 224) to Vdd after
729 is driven low at 4.56ns through fet at (-138, 221) to GND after
777 is driven high at 3.90ns through fet at (-144, 222) to 853
through fet at (-147, 222) to 651
through fet at (-153, 224) to Vdd after
800 is driven low at 0.14ns through fet at (-160, 224) to 768
through fet at (-164, 219) to GND after
CLOCK is driven high at 0.00ns
Node 199 is driven low at 54.15ns
...through fet at (-329, 172) to GND after
192 is driven high at 53.85ns
...through fet at (-325, 195) to Q
...through fet at (-290, 178) to Vdd after
217 is driven low at 52.94ns
...through fet at (-260, 172) to GND after
189 is driven high at 52.72ns
...through fet at (-263, 179) to 259
...through fet at (-155, 177) to Vdd after
229 is driven low at 51.16ns
...through fet at (-148, 173) to 202
...through fet at (-148, 171) to GND after
74 is driven high at 50.24ns
...through fet at (-184, 173) to 46
...through fet at (-172, 175) to 17
...through fet at (-169, 173) to 70
...through fet at (-24, 177) to Vdd after
233 is driven low at 44.53ns
...through fet at (-19, 173) to 205
...through fet at (-18, 171) to GND after

550 is driven low at 4.04ns
...through fet at (-129, 219) to GND after
729 is driven high at 3.76ns
...through fet at (-138, 224) to Vdd after
777 is driven low at 3.03ns
...through fet at (-144, 222) to 853
...through fet at (-147, 222) to 651
...through fet at (-153, 221) to GND after
800 is driven high at 1.50ns
...through fet at (-160, 224) to 768
...through fet at (-166, 224) to Vdd after
497 is driven low at 0.26ns
...through fet at (-195, 197) to 421
...through fet at (-198, 212) to GND after
CLOKKT is driven high at 0.00ns
REFERENCES