A Gate-Array Realization of an Algorithm for Division

1984

Salman Y. Abbasi

University of Central Florida

Find similar works at: http://stars.library.ucf.edu/rtd

University of Central Florida Libraries http://library.ucf.edu

Part of the Engineering Commons

STARS Citation


This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
A GATE-ARRAY REALIZATION
OF AN ALGORITHM FOR DIVISION

BY

SALMAN YOUSEF ABBASI
B.S.E., Fairleigh Dickinson University, 1978

RESEARCH REPORT

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Engineering
in the Graduate Studies Program of the College of Engineering
University of Central Florida
Orlando, Florida

Spring Term
1984
ABSTRACT

A realization of a division algorithm suitable for high speed pipeline and realtime processors is presented. Implementation of the divide algorithm can be achieved by utilizing LSI / VLSI gate-array technology. The divider performs precision, high speed 9 bit sign magnitude division. The design consists of combinational logic, where input and output data are latched into input and output registers. Data propagates through 16 divide stages. The n'th stage generates the n'th quotient bit upon receiving the updated dividend and controls from the previous stage. A simulation program is developed to verify the algorithm, and an analysis for speed performance and cost is provided. Other division algorithms are discussed.
# TABLE OF CONTENTS

LIST OF FIGURES ........................................ iv

I. INTRODUCTION ........................................ 1
   Important Parameters ................................ 2
   Impact Of New Technology .............................. 3

II. DIVIDER GATE-ARRAY REALIZATION ............... 4
   Performance ........................................ 4
   Algorithm .......................................... 4
   Divider block Diagram ............................... 6
      Input/Output Registers ........................... 6
      Normalization Block .............................. 8
      Scaling Control .................................. 8
      Sign Generator .................................. 10
      Divide Block .................................... 12
         First Type Divide Cell ....................... 12
         Second Type Divide Cell ..................... 15
   Example ............................................ 18
   Propagation Delay .................................. 25
   LSI Implementation ................................ 25
   Software Simulation ................................. 27
   Other Divide Algorithms ............................ 27
      Algorithm 1 ..................................... 27
      Algorithm 2 ..................................... 33

III. CONCLUSION ......................................... 38

REFERENCES ............................................. 39
LIST OF FIGURES

1. Input/Output Data Format ................................. 5
2. Divide Block Diagram ..................................... 7
3. Normalized Logic Block Diagram ......................... 9
4. Scaling Control Diagram ................................... 11
5. First Type Divide Cell Diagram ......................... 13
6. 8 Bit Subtractor Block Diagram ......................... 14
7. 3:1 Mux Logic Diagram .................................... 16
8. Second Type Divide Cell Diagram ....................... 17
9. 2:1 Mux Logic Diagram .................................... 19
10. Software Simulation Program ............................. 27
11. Algorithm 1 Block Diagram ............................... 32
12. Algorithm 2 Pipeline Diagram ........................... 34
13. Pipeline Stage Divider Block Diagram .................. 35
I. INTRODUCTION

Many digital systems perform one or more of the binary operations of addition, subtraction, multiplication, and division. All arithmetic functions can be related to addition; subtraction is performed by adding complemented numbers, multiplication is repeated addition, and division is repeated subtraction. Thus, all these arithmetic operations could be performed in a central processor unit consisting of a binary adder and control logic.

These operations could be performed with either a serial or parallel circuit arrangement. A serial arrangement requires considerably less hardware than a parallel arrangement. However, if extremely fast arithmetic is required, parallel methods must be used. Multiplication and division, when performed by repeated additions and subtractions, consume several processing cycles. For fast systems, such as real time and pipeline processors, special hardware units are used to perform multiplication and division.

Arithmetic operations vary in their complexity of implementations. Addition and subtraction are fairly simple operations and are easy to implement. Multiplication and division are complicated functions and are difficult to implement. Multiplication and division are basically iterative processes. The first involves inspecting the multiplier digits one at a time, shifting one place to the left when the inspected
digit equals one, and then adding the individual partial products. The second involves a quotient digit selection, a multiplication, and a subtraction during each stage.

Computers generally break multiplication and division operations down into simpler functions such as add, subtract, or shift. Existing arithmetic logic units (ALUs) can perform these operations under program control. To accomplish division in minimum time and without a sizeable increase in hardware, the IBM 360/91 uses its fast pipeline multiply hardware for iterations in the division process. The Cray-1 computer performs division by forming the reciprocal of the divisor using its reciprocal pipe and then passing the reciprocal along with the dividend through the multiply pipe. Most logic designers use a method similar to the one used in the Cray-1 computer. A PROM look-up table is used to compute the reciprocal of the divisor, then the reciprocal is multiplied by the dividend.

**Important Parameters**

Some of the important parameters to be considered in implementing arithmetic algorithms are speed, cost, and accuracy. IBM 360/91 and Cray-1 computers share major components (pipelines and multipliers) to perform division. Other computers utilize ALUs. Computers and pipeline processors that multiply the reciprocal of the divisor by the dividend need special hardware to compute the reciprocal, and thus share a pipe between multiplication and division. The trade-off between speed and cost varies from one system to another and could be
optimized to meet system specifications. The throughput for a computer with a special division pipe is higher than the throughput for a computer which shares its multiply pipe. However, the hardware cost is higher when a special division pipe is added.

Impact of New Technology

Technological improvements in fabricating integrated circuits from silicon are being made every day. Today's technology produces smaller and smaller electronic components performing increasingly complex electronic functions at ever higher speed and lower cost. Circuit designers are leaving the microscopic world of PC-board-mounted MSI based logic and entering the microscopic realm of the system on a chip. Gate-array technology has been fully opened to designers. Manufacturers now offer comprehensive computer aided design (CAD) packages to help designers realize their gate-array requirements in any technology: ECL for speed, CMOS for efficiency, NMOS or TTL for anything in between. Gate-arrays exist in three primary configurations: 1) basic gate-array only, 2) semicustom and 3) custom arrays. Basic-gate-only populated chips are the easiest to employ, incorporating nothing more than NAND and NOR gates. Semicustom gate-arrays are for more complex designs, incorporating medium complexity modules such as JK and/or D flip flops and multiple input AND, OR, and XOR gates. Full-custom arrays can accomplish virtually any logic realization.
II. DIVIDER GATE-ARRAY REALIZATION

Performance

The gate-array design presented in this report emphasizes high speed performance and accurate division. The divider circuit accepts a sign magnitude dividend and divisor and produces a sign magnitude quotient. The design is configured for an 8 bit dividend plus 1 bit dividend sign, and an 8 bit divisor plus 1 bit divisor sign. The output quotient is 16 bits plus a sign bit. Figure 1 shows the input and output data formats.

Algorithm

The divider design is based on the hand calculation division algorithm. If there is a 4 bit dividend and a 4 bit divisor, the quotient can be calculated as follows:

\[
\begin{align*}
\text{dividend} & = 1001 \quad (4 \text{ bit binary number}) \\
\text{divisor} & = 0010 \quad (4 \text{ bit binary number}) \\
\hline
\end{align*}
\]

\[
\begin{array}{c}
0010 \\
1001 \\
10 \\
00 \\
00 \\
01 \\
00 \\
10 \\
10 \\
00 \\
\hline
\end{array}
\]

quotient = 100.1 (4 bit binary number)
Input Data Formats

Output Data Format

Figure 1. Input / Output Data Format
One important step in realizing the basic algorithm for division is to normalize the divisor, that is, to shift the divisor left until the MSB bit becomes 1. For example, the 8 bit binary number 00011010 becomes 11010000. Since the number of shifts for a given divisor is not known, the quotient scaling cannot be determined. The quotient scaling problem is corrected by monitoring all leading zero bits of the divisor before normalization and generating internal control signals to force quotient MSB bits to zero as explained later in this section.

Divider Block Diagram

The divider block diagram is shown in Figure 2 and consists of the following major blocks:

1. Input/output registers -- latches the input and output data.
2. Normalization block -- shifts the divisor until the MSB bit becomes 1.
3. Scaling control -- generates internal control signals to adjust for scaling problems resulting from divisor normalization.
5. Divide block -- stages of the process of division.

Input/Output Registers

Input and output registers are used as storage devices. Input and output data are latched simultaneously. The input and output maximum clock rate must be greater than the maximum propagation delay.
Figure 2. Divide Block Diagram
Normalization Block

The divisor is normalized by shifting left until the MSB bit becomes 1. Consider an 8 bit divisor D₀ through D₇, where D₀ is the LSB bit and D₇ is the MSB bit. The 8 bit normalized divisor D₀' through D₇', where D₀' is the LSB bit and D₇' is the MSB bit, will be determined by the following logic equations:

\[
D₀' = D₀ \cdot D₇
\]
\[
D₁' = D₁ \cdot D₇ + D₀ \cdot D₆ \cdot \overline{D₇}
\]
\[
D₂' = D₂ \cdot D₇ + D₁ \cdot D₆ \cdot \overline{D₇} + D₀ \cdot D₅ \cdot \overline{D₆} \cdot \overline{D₇}
\]
\[
D₃' = D₃ \cdot D₇ + D₂ \cdot D₆ \cdot \overline{D₇} + D₅ \cdot D₆ \cdot \overline{D₇} + D₀ \cdot D₄ \cdot D₅ \cdot \overline{D₆} \cdot \overline{D₇}
\]
\[
D₄' = D₄ \cdot D₇ + D₃ \cdot D₆ \cdot \overline{D₇} + D₂ \cdot D₅ \cdot \overline{D₆} \cdot \overline{D₇} + D₁ \cdot D₄ \cdot \overline{D₅} \cdot \overline{D₆} \cdot \overline{D₇}
\]
\[
+ D₀ \cdot D₃ \cdot \overline{D₄} \cdot \overline{D₅} \cdot \overline{D₆} \cdot \overline{D₇}
\]
\[
D₅' = D₅ \cdot D₇ + D₄ \cdot D₆ \cdot \overline{D₇} + D₃ \cdot D₅ \cdot \overline{D₆} \cdot \overline{D₇} + D₂ \cdot D₄ \cdot \overline{D₅} \cdot \overline{D₆} \cdot \overline{D₇}
\]
\[
+ D₁ \cdot D₃ \cdot \overline{D₄} \cdot \overline{D₅} \cdot \overline{D₆} \cdot \overline{D₇} + D₀ \cdot D₂ \cdot \overline{D₃} \cdot \overline{D₄} \cdot \overline{D₅} \cdot \overline{D₆} \cdot \overline{D₇}
\]
\[
D₆' = D₆ \cdot D₇ + D₅ \cdot D₆ \cdot \overline{D₇} + D₄ \cdot D₅ \cdot D₆ \cdot \overline{D₇} + D₃ \cdot D₄ \cdot D₅ \cdot \overline{D₆} \cdot \overline{D₇}
\]
\[
+ D₂ \cdot D₃ \cdot \overline{D₄} \cdot \overline{D₅} \cdot \overline{D₆} \cdot \overline{D₇} + D₁ \cdot D₂ \cdot \overline{D₃} \cdot \overline{D₄} \cdot \overline{D₅} \cdot \overline{D₆} \cdot \overline{D₇}
\]
\[
+ D₀ \cdot D₁ \cdot \overline{D₂} \cdot \overline{D₃} \cdot \overline{D₄} \cdot \overline{D₅} \cdot \overline{D₆} \cdot \overline{D₇}
\]
\[
D₇' = D₇ + D₆ + D₅ + D₄ + D₃ + D₂ + D₁ + D₀
\]

Figure 3 shows a diagram of the normalization logic.

Scaling Control

Scaling control signals are generated by monitoring the leading zero bits of the divisor and are used to force the associated quotient bits to zero. Also, the dividend for the following stage is selected
Figure 3. Normalized Logic Block Diagram
to be the same as the dividend for the current stage if the current scaling control signal is active. Consider a 16 bit quotient \(Q_7\) through \(Q_{-8}\), with \(Q_7\) as the MSB bit and \(Q_{-8}\) as the LSB bit and \(F_7\) through \(F_1\) are the associated controls for quotient bits \(Q_7\) through \(Q_1\). No control bits are needed for quotient bits \(Q_0\) through \(Q_{-8}\). \(F_7\) through \(F_1\) are generated using the following logic equations:

\[
F_1 = D_7
\]

\[
F_2 = \overline{D_7} \overline{D_6}
\]

\[
F_3 = \overline{D_7} \overline{D_6} \overline{D_5}
\]

\[
F_4 = \overline{D_7} \overline{D_6} \overline{D_5} \overline{D_4}
\]

\[
F_5 = \overline{D_7} \overline{D_6} \overline{D_5} \overline{D_4} \overline{D_3}
\]

\[
F_6 = \overline{D_7} \overline{D_6} \overline{D_5} \overline{D_4} \overline{D_3} \overline{D_2}
\]

\[
F_7 = \overline{D_7} \overline{D_6} \overline{D_5} \overline{D_4} \overline{D_3} \overline{D_2} \overline{D_1}
\]

Figure 4 shows a logic diagram of the scaling controls \(F_1\), through \(F_7\).

**Sign Generator**

Quotient sign is derived from dividend and divisor sign bits as shown in the following table:

<table>
<thead>
<tr>
<th>Dividend Sign</th>
<th>Divisor Sign</th>
<th>Quotient Sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

where: 0 is positive and 1 is negative

Note: The dividend, divisor, and quotient are sign magnitude numbers.
Figure 4. Scaling Control Diagram
Divide Block

Inputs to this block are an 8 bit dividend, $N_0$ through $N_7$, and an 8 bit normalized divisor, $D_0'$ through $D_7'$. The output is a 16 bit quotient $Q_8$ through $Q_7$. The divide block contains 16 divide cells, allowing one cell for every divide stage. The first 7 divide cells are associated with the 7 MSB bits of the quotient, the remaining 9 cells are slightly different from the first 7 and are associated with the 9 LSB bits of the quotient.

First Type Divide Cell. Figure 5 shows a block diagram of the first type divide cell. This type is used in the first seven divide stages. The terms shown on the block diagram are defined as follows:

- $n$ - Stage identifier, where $n$ goes from 7 to -8.
- $N_n$ - Dividend for the $n$'th stage.
- $F_n$ - Scaling control for the $n$'th stage.
- $D'$ - Normalized divisor.
- $OV_n$ - Overflow bit for the $n$'th stage.
- $B_n$ - Borrow bit resulted from the $n$'th stage.
- $Q_n$ - Quotient output from the $n$'th stage.

The following steps are performed in every stage:

Step 1: $D'$ is subtracted from $N_n$, the result is multiplied by 2 by shifting left 1 bit, and is forwarded to a mux logic to be used for selecting $N_{n-1}$ ($N_n$ for the next stage). Also, a borrow bit is generated if the result of the subtraction is negative. Figure 6 shows an 8 bit subtractor with borrow in and out bits.
Figure 5. First Type Divide Cell Diagram
Figure 6. 8 Bit-Subtractor Block Diagram
Step 2: A logical AND is performed between $B_n$ and $\overline{OV_n}$ and the result is used along with $F_n$ as a mux control to select $N_{n-1}$ as follows:

<table>
<thead>
<tr>
<th>$F_n$ ($S_1$)</th>
<th>$B_n$, $\overline{OV_n}$ ($S_0$)</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>$2(N_n - D')$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$2N_n$</td>
</tr>
<tr>
<td>0</td>
<td>1 or 0</td>
<td>$N_n$</td>
</tr>
</tbody>
</table>

The $OV_n$ bit in the above table is used to detect the overflow at the mux output as a result of the 1 bit left shift to generate $2N_n$. If the $OV_n$ bit is set and a $B_n$ bit is generated, then $OV_n$ will force $B_n$ to zero since it is not a true borrow. Whenever $F_n$ is logic 0 the mux selected output is $N_n$. Thus, as long as the $F_n$ flag is 0, the dividend will be passed to the next stage without any modifications, and the output $Q_n$ will be forced to zero. If $B_n$ is 0, $OV_n$ is 0, and $F_n$ is 1, then $2(N_n - D')$ is selected and $Q_n$ is set to 1. If $B_n$ is 1, $OV_n$ is 0, and $F_n$ is 1, then $2N_n$ is selected and $Q_n$ is set to 0. Figure 7 presents a logic diagram of the 3:1 mux along with the select table.

**Second Type Divide Cell.** Figure 8 presents a block diagram of the second type of divide cell. This cell is used in the remaining nine divide stages and functions in a fashion similar to the previous cell with the exception of an absence of the $f_n$ control bit.
Figure 7. 3:1 Mux Logic Diagram
Figure 8. Second Type Divide Cell Diagram
The mux selects one of the two values $2N_n$ or $2(N_n - D')$ as follows:

<table>
<thead>
<tr>
<th>Bn. OVn (S)</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$2(N_n - D')$</td>
</tr>
<tr>
<td>1</td>
<td>$2N_n$</td>
</tr>
</tbody>
</table>

If $OV_n$ is set and a $B_n$ bit is generated from the previous stage then the $OV_n$ bit will force $B_n$ to zero since it is not a true borrow.

$Q_n$ is determined as follows:

<table>
<thead>
<tr>
<th>Bn. OVn</th>
<th>$Q_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 9 shows a logic diagram of a 2:1 mux along with the select table.

**Example**

An example is helpful at this point to clarify the divide algorithm by going through it step by step.

**Input:**

- Dividend $(N) = 10011101 = (157)_{10}$
- Dividend Sign = 1 (negative)
- Divisor $(D) = 00001001 = (9)_{10}$
- Divisor Sign = 0 (positive)

**Output:**

- Quotient $(Q) = N/D$
Figure 9. 2:1 Mux Logic Diagram
Q sign = (Dividend Sign) XOR (Divisor Sign)

**Step 1:**
Determine Q sign = (1) XOR (0) = 1 (negative)

**Step 2:**
Normalize D to get D' according to the logic equations given previously (see "Normalization Block").
D' = 10010000

**Step 3:**
Generate scaling controls F₁ through F₇ according to the logic equations (see "Scaling Control").

\[
\begin{align*}
F_1 &= 1 \\
F_2 &= 1 \\
F_3 &= 1 \\
F_4 &= 1 \\
F_5 &= 0 \\
F_6 &= 0 \\
F_7 &= 0
\end{align*}
\]

**Step 4:**
Pass through 16 divide stages to compute Q

1st Divide stage, n=7
Q₇ = F₇. (B₇. OV₇) = 0 (F₇ = 0) MSB bit

2nd Divide stage, n=6
Q₆ = F₆. (B₆. OV₆) = 0 (F₆ = 0)

3rd Divide stage, n=5
Q₅ = F₅. (B₅. OV₅) = 0 (F₅ = 0)

4th Divide stage, n=4
N₄ = 10011101
D' = 10010000
\[
\begin{array}{c}
00001101 \\
\text{subtract}
\end{array}
\]
B_4 = 0
F_4 = 1
OV_4 = 0
Q_4 = F_4 \cdot (B_4 \cdot OV_4) = 1
N_3 = 00011010
OV_3 = 0

5th Stage, n=3
N_3 = 00011010

D' = 10010000 subtract
    10001010

B_3 = 1 , F_3 = 1 , OV_3 = 0
Q_3 = F_3 \cdot (B_3 \cdot OV_3) = 0
N_2 = 00110100 , OV_2 = 0

6th Stage, n=2
N_2 = 00110100

D' = 10010000 Subtract
    10100100

B_2 = 1 , F_2 = 1 , OV_2 = 0
Q_2 = F_2 \cdot (B_2 \cdot OV_2) = 0
N_1 = 01101000 , OV_1 = 0

7th Stage, n=1
N_1 = 01101000

D' = 10010000 Subtract
    11011000

B_1 = 1 , F_1 = 1 , OV_1 = 0
Q_1 = F_1 \cdot (B_1 \cdot OV_1) = 0
N_0 = 11010000 , OV_0 = 0
8th Stage, \( n=0 \)

\[ N_0 = 11010000 \]
\[ D' = 10010000 \quad \text{Subtract} \quad 01000000 \]
\[ B_0 = 0 \quad , \quad OV_0 = 0 \]
\[ Q_0 = (B_0 \cdot OV_0) = 1 \]
\[ N_{-1} = 10000000 \quad , \quad OV_{-1} = 0 \]

9th Stage, \( n=-1 \)

\[ N_{-1} = 10000000 \]
\[ D' = 10010000 \quad \text{Subtract} \quad 11110000 \]
\[ B_{-1} = 1 \quad , \quad OV_{-1} = 0 \]
\[ Q_{-1} = (B_{-1} \cdot OV_{-1}) = 0 \]
\[ N_{-2} = 00000000 \quad , \quad OV_{-2} = 1 \]

10th Stage, \( n=-2 \)

\[ N_{-2} = 00000000 \]
\[ D' = 10010000 \quad \text{Subtract} \quad 01110000 \]
\[ B_{-2} = 1 \quad , \quad OV_{-2} = 1 \]
\[ Q_{-2} = (B_{-2} \cdot OV_{-2}) = 1 \]
\[ N_{-3} = 11100000 \quad , \quad OV_{-3} = 0 \]

11th Stage, \( n=-3 \)

\[ N_{-3} = 11100000 \]
\[ D' = 10010000 \quad \text{Subtract} \quad 01010000 \]
\begin{align*}
B_{-3} &= 0 \quad , \quad OV_{-3} = 0 \\
Q_{-3} &= (B_{-3} \cdot OV_{-3}) = 1 \\
N_{-4} &= 10100000 \quad , \quad OV_{-4} = 0
\end{align*}

12th Stage, \quad n=-4

\begin{align*}
N_{-4} &= 10100000 \\
D' &= \overline{10010000} \quad \text{Subtract} \\
& \overline{00010000} \\
B_{-4} &= 0 \quad , \quad OV_{-4} = 0 \\
Q_{-4} &= (B_{-4} \cdot OV_{-4}) = 1 \\
N_{-5} &= 00100000 \quad , \quad OV_{-5} = 0
\end{align*}

13th Stage, \quad n=-5

\begin{align*}
N_{-5} &= 00100000 \\
D' &= \overline{10010000} \quad \text{Subtract} \\
& \overline{10010000} \\
B_{-5} &= 1 \quad , \quad OV_{-5} = 0 \\
Q_{-5} &= (B_{-5} \cdot OV_{-5}) = 0 \\
N_{-6} &= 01000000 \quad , \quad OV_{-6} = 0
\end{align*}

14th Stage, \quad n=-6

\begin{align*}
N_{-6} &= 01000000 \\
D' &= \overline{10010000} \quad \text{Subtract} \\
& \overline{10110000} \\
B_{-6} &= 1 \quad , \quad OV_{-6} = 0 \\
Q_{-6} &= (B_{-6} \cdot OV_{-6}) = 0 \\
N_{-7} &= 10000000 \quad , \quad OV_{-7} = 0
\end{align*}
15th Stage, \( n=-7 \)

\[ N_7 = 10000000 \]

\[ D' = 10010000 \quad \text{Subtract} \quad 11110000 \]

\[ B_7 = 1, \quad OV_7 = 0 \]

\[ Q_7 (B_7 \cdot OV_7) = 0 \]

\[ N_8 = 00000000, \quad OV_8 = 1 \]

16th Stage, \( n=-8 \)

\[ N_8 = 00000000 \]

\[ D' = 10010000 \quad \text{Subtract} \quad 01110000 \]

\[ B_8 = 1, \quad OV_8 = 1 \]

\[ Q_8 = (B_8 \cdot OV_8) = 1 \quad (\text{LSB}) \]

Result

\[
\begin{array}{cccc}
Q_7 & Q_0 & Q_{-1} & Q_8 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\
Q = & & & & \\
Q = 17.44 & & & & \\
\end{array}
\]

For comparison

\[ Q = \frac{N}{D} = \frac{157}{9} = 17.44 \]
Propagation Delay

The propagation delay in every stage is equal to the sum of delays through an 8-bit subtractor, 3:1 mux for the first seven stages and 2:1 mux for the last nine stages, and an AND gate. The total propagation delay through the gate-array divider could be calculated as follows:

1. Delay through the 8-bit subtractor (see "Subtractor Block Diagram").
   5 gate levels (XOR gate = 2 levels) \* 1 nsec \* 15 stages = 75 nsec.

2. Delay through the 3:1 mux (see "3:1 Mux Block Diagram").
   3 gate levels \* 1 nsec \* 7 stages = 21 nsec.

3. Delay through the 2:1 mux (see "2:1 Mux Block Diagram").
   3 gate levels \* 1 nsec \* 9 = 27 nsec.

4. Delay through last stage.
   7 gate levels (subtractor and gates) \* 1 nsec = 7 nsec.

5. Delay through input latches.
   2 gate levels \* 1 nsec = 2 nsec.

Total delay = 75 + 21 + 27 + 7 + 2 = 132 nsec.

Note: A 1 nsec internal gate propagation delay was used in the calculation. Applied Micro Circuits Corporation (AMCC) and Texas Instruments produce gate-arrays using ECL technology which achieves such speed. The propagation delay may be increased by a few nsec if internal drivers are added to buffer loaded signals.

LSI Implementation

To implement this algorithm for an 8 bit dividend, 8 bit divisor and a 16 bit quotient, a 2000 to 3000 gate gate-array is needed. Approximate gate number calculations are listed in table 1.
<table>
<thead>
<tr>
<th>Function</th>
<th>No. Gates</th>
<th>No. Of functions</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bit subtract</td>
<td>76</td>
<td>16</td>
<td>1216</td>
</tr>
<tr>
<td>9 bits (3:1 MUX)</td>
<td>54</td>
<td>7</td>
<td>378</td>
</tr>
<tr>
<td>9 bits (2:1 MUX)</td>
<td>36</td>
<td>8</td>
<td>288</td>
</tr>
<tr>
<td>D Normalizer</td>
<td>42</td>
<td>1</td>
<td>42</td>
</tr>
<tr>
<td>Registers I/O</td>
<td>160</td>
<td>1</td>
<td>160</td>
</tr>
<tr>
<td>Other gates</td>
<td>27</td>
<td>1</td>
<td>27</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>-</strong></td>
<td><strong>-</strong></td>
<td><strong>2111</strong></td>
</tr>
<tr>
<td>ADDRESS</td>
<td>INSTRUCTION</td>
<td>SOURCE CODE</td>
<td>COMMENTS</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>-------------</td>
<td>----------</td>
</tr>
<tr>
<td>2000</td>
<td>LDA 20B0</td>
<td>Get Divisor</td>
<td></td>
</tr>
<tr>
<td>2003</td>
<td>MOV B,A</td>
<td>MOVE A TO B</td>
<td></td>
</tr>
<tr>
<td>2004</td>
<td>STC</td>
<td>SET CARRY BIT = 1</td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td>CMC</td>
<td>COMPLEMENT CARRY BIT</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>RAL</td>
<td>SHIFT DIVISOR LEFT</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>JNC LOOP1</td>
<td>JUMP NO CARRY</td>
<td></td>
</tr>
<tr>
<td>200A</td>
<td>MOV A,B</td>
<td>MOVE B TO A</td>
<td></td>
</tr>
<tr>
<td>200B</td>
<td>STA 20B1</td>
<td>STORE A</td>
<td></td>
</tr>
<tr>
<td>200E</td>
<td>LXIH 20B8</td>
<td>LOAD P1</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>MVI D,07</td>
<td>INITIALIZE COUNTER</td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td>MVI B,01</td>
<td>MOVE 01 TO B</td>
<td></td>
</tr>
<tr>
<td>2015</td>
<td>LDA 20B0</td>
<td>LOAD DIVISOR</td>
<td></td>
</tr>
<tr>
<td>2018</td>
<td>CMA</td>
<td>COMPLEMENT A</td>
<td></td>
</tr>
<tr>
<td>201A</td>
<td>MOV C,A</td>
<td>MOVE C TO A</td>
<td></td>
</tr>
<tr>
<td>201B</td>
<td>RLC</td>
<td>GENERATE F CONTROLS</td>
<td></td>
</tr>
<tr>
<td>201C</td>
<td>MOV C,A</td>
<td>TO NORMALIZE QUOTIENT</td>
<td></td>
</tr>
<tr>
<td>201D</td>
<td>ANA B</td>
<td>MASK B WITH A</td>
<td></td>
</tr>
<tr>
<td>201E</td>
<td>ANA M</td>
<td>MASK M WITH A</td>
<td></td>
</tr>
<tr>
<td>201F</td>
<td>INX H</td>
<td>INCREMENT HL</td>
<td></td>
</tr>
<tr>
<td>2020</td>
<td>MOV M,A</td>
<td>MOVE A TO M</td>
<td></td>
</tr>
<tr>
<td>2021</td>
<td>LCR D</td>
<td>IS COUNT = 0</td>
<td></td>
</tr>
<tr>
<td>2022</td>
<td>JNZ LOOP2</td>
<td>NO GO TO LOOP 2</td>
<td></td>
</tr>
<tr>
<td>2025</td>
<td>NOP</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>2026</td>
<td>MVI C,8</td>
<td>MOVE 8 TO C</td>
<td></td>
</tr>
<tr>
<td>2026</td>
<td>MVI D,0</td>
<td>CLEAR D</td>
<td></td>
</tr>
<tr>
<td>202A</td>
<td>LDA 20B1</td>
<td>LOAD DIVISOR</td>
<td></td>
</tr>
<tr>
<td>202D</td>
<td>MOV B,A</td>
<td>MOVE A TO B</td>
<td></td>
</tr>
<tr>
<td>202E</td>
<td>LXIH 20BF</td>
<td>LOAD F CONTROL</td>
<td></td>
</tr>
<tr>
<td>2031</td>
<td>STC</td>
<td>CLEAR CARRY</td>
<td></td>
</tr>
<tr>
<td>2032</td>
<td>CMC</td>
<td>COMPLEMENT CARRY</td>
<td></td>
</tr>
<tr>
<td>2033</td>
<td>MOV A,M</td>
<td>MOVE M TO A</td>
<td></td>
</tr>
<tr>
<td>2034</td>
<td>SBI 0</td>
<td>IS F CONTROL =1</td>
<td></td>
</tr>
<tr>
<td>2036</td>
<td>JZ LOOP6</td>
<td>YES GO TO LOOP 6</td>
<td></td>
</tr>
<tr>
<td>2039</td>
<td>LDA 20B2</td>
<td>NO, LOAD DIVISOR</td>
<td></td>
</tr>
<tr>
<td>203C</td>
<td>SUB B</td>
<td>IS N-D NEGATIVE</td>
<td></td>
</tr>
<tr>
<td>203D</td>
<td>JC LOOP5</td>
<td>YES, GO TO LOOP 5</td>
<td></td>
</tr>
<tr>
<td>2040</td>
<td>MVI E,01</td>
<td>NO, MOVE 01 TO E</td>
<td></td>
</tr>
<tr>
<td>2042</td>
<td>LOOP4</td>
<td>MOVE A TO D</td>
<td></td>
</tr>
<tr>
<td>2043</td>
<td>STC</td>
<td>CLEAR CARRY</td>
<td></td>
</tr>
<tr>
<td>2044</td>
<td>CMC</td>
<td>COMPLEMENT CARRY</td>
<td></td>
</tr>
<tr>
<td>2045</td>
<td>RAL</td>
<td>MULTIPLY D BY 2</td>
<td></td>
</tr>
<tr>
<td>2046</td>
<td>STA 20B2</td>
<td>STORE DIVIDEND</td>
<td></td>
</tr>
</tbody>
</table>

Figure 10. Software Simulation Program
<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>INSTRUCTION</th>
<th>SOURCE CODE</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2049</td>
<td>7A</td>
<td>MOV A,D</td>
<td>MOVE D TO A</td>
</tr>
<tr>
<td>204A</td>
<td>07</td>
<td>RLC</td>
<td>ROTATE LEFT</td>
</tr>
<tr>
<td>204B</td>
<td>E6 01</td>
<td>ANI</td>
<td>MASK LSB BIT</td>
</tr>
<tr>
<td>204D</td>
<td>57</td>
<td>MOV D,A</td>
<td>MOVE A TO D</td>
</tr>
<tr>
<td>204E</td>
<td>3A B3 20</td>
<td>LDA 20B3</td>
<td>LOAD Q MSB</td>
</tr>
<tr>
<td>2051</td>
<td>17</td>
<td>RAL</td>
<td>ROTATE LEFT</td>
</tr>
<tr>
<td>2052</td>
<td>83</td>
<td>ADD E</td>
<td>ADD A TO E</td>
</tr>
<tr>
<td>2053</td>
<td>32 B3 20</td>
<td>STA 20B3</td>
<td>STORE Q MSB</td>
</tr>
<tr>
<td>2056</td>
<td>C3 6D 20</td>
<td>JMP LOOP7</td>
<td>GO TO LOOP 7</td>
</tr>
<tr>
<td>2059</td>
<td>5A LOOP5</td>
<td>MOV E,D</td>
<td>MOVE D TO E</td>
</tr>
<tr>
<td>205A</td>
<td>15 DCR D</td>
<td>IS D = 0</td>
<td></td>
</tr>
<tr>
<td>205B</td>
<td>CA 42 20</td>
<td>JZ LOOP4</td>
<td>YES GO TO LOOP 4</td>
</tr>
<tr>
<td>205E</td>
<td>3A B2 20</td>
<td>LDA 20B2</td>
<td>NO, LOAD DIVIDEND</td>
</tr>
<tr>
<td>2061</td>
<td>1E 00</td>
<td>MVI E,0</td>
<td>MOVE 0 TO E</td>
</tr>
<tr>
<td>2063</td>
<td>C3 42 20</td>
<td>JMP LOOP4</td>
<td>GO TO LOOP 4</td>
</tr>
<tr>
<td>2066</td>
<td>3A B3 20</td>
<td>LDA 20B3</td>
<td>LOAD Q MSB</td>
</tr>
<tr>
<td>2069</td>
<td>17 RAL</td>
<td>MULTIPLY BY 2</td>
<td></td>
</tr>
<tr>
<td>206A</td>
<td>32 B3 20</td>
<td>STA 20B3</td>
<td>STORE Q MSB</td>
</tr>
<tr>
<td>206D</td>
<td>2B LOOP7</td>
<td>DCX H</td>
<td>MOVE POINTER</td>
</tr>
<tr>
<td>206E</td>
<td>0D DCR C</td>
<td>IS COUNT = 0</td>
<td></td>
</tr>
<tr>
<td>206F</td>
<td>C2 31 20</td>
<td>JNZ LOOP3</td>
<td>NO, GO TO LOOP 3</td>
</tr>
<tr>
<td>2072</td>
<td>00 NOP</td>
<td>NO OPERATION</td>
<td></td>
</tr>
<tr>
<td>2073</td>
<td>00 NOP</td>
<td>NO OPERATION</td>
<td></td>
</tr>
<tr>
<td>2074</td>
<td>00 NOP</td>
<td>NO OPERATION</td>
<td></td>
</tr>
<tr>
<td>2075</td>
<td>0E 08 MVI C,8</td>
<td>YES, MOVE 8 TO C</td>
<td></td>
</tr>
<tr>
<td>2077</td>
<td>37 LOOP8</td>
<td>STC</td>
<td>CLEAR CARRY</td>
</tr>
<tr>
<td>2078</td>
<td>3F CMC</td>
<td>COMPLEMENT CARRY</td>
<td></td>
</tr>
<tr>
<td>2079</td>
<td>3A B3 20</td>
<td>LDA 20B2</td>
<td>LOAD DIVIDEND</td>
</tr>
<tr>
<td>207C</td>
<td>90 SUB B</td>
<td>SUBTRACT DIVISOR</td>
<td></td>
</tr>
<tr>
<td>207D</td>
<td>DA 99 20</td>
<td>JC LOOP10</td>
<td>BORROW=1 GO TO LOOP10</td>
</tr>
<tr>
<td>2080</td>
<td>1E 01 MVI E,01</td>
<td>BORROW=0 MOVE 1 TO E</td>
<td></td>
</tr>
<tr>
<td>2082</td>
<td>57 LOOP9</td>
<td>MOV D,A</td>
<td>MOVE A TO D</td>
</tr>
<tr>
<td>2083</td>
<td>37 STC</td>
<td>CLEAR CARRY</td>
<td></td>
</tr>
<tr>
<td>2084</td>
<td>3F CMC</td>
<td>COMPLEMENT CARRY</td>
<td></td>
</tr>
<tr>
<td>2085</td>
<td>17 RAL</td>
<td>MULT DIVIDEND BY 2</td>
<td></td>
</tr>
<tr>
<td>2086</td>
<td>32 B2 20</td>
<td>STA 20B2</td>
<td>STORE DIVIDEND</td>
</tr>
<tr>
<td>2089</td>
<td>7A MOV A,D</td>
<td>MOVE D TO A</td>
<td></td>
</tr>
<tr>
<td>208A</td>
<td>07 RLC</td>
<td>ROTATE DIVIDEND LEFT</td>
<td></td>
</tr>
<tr>
<td>208B</td>
<td>E6 01 ANI</td>
<td>MASK THE LSB BIT</td>
<td></td>
</tr>
<tr>
<td>208D</td>
<td>57 MOV D,A</td>
<td>MOVE A TO D</td>
<td></td>
</tr>
<tr>
<td>208E</td>
<td>3A B4 20</td>
<td>LDA 20B4</td>
<td>LOAD Q LSB</td>
</tr>
<tr>
<td>2091</td>
<td>17 RAL</td>
<td>ROTATE LEFT</td>
<td></td>
</tr>
<tr>
<td>2092</td>
<td>83 ADD E</td>
<td>ADD A TO E</td>
<td></td>
</tr>
<tr>
<td>2093</td>
<td>32 B4 20</td>
<td>STA 20B4</td>
<td>STORE Q LSB</td>
</tr>
</tbody>
</table>

Figure 10. (Continued)
<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>INSTRUCTION</th>
<th>SOURCE CODE</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2096</td>
<td>C3 A6 20</td>
<td>JMP LOOP11</td>
<td>GO TO LOOP 11</td>
</tr>
<tr>
<td>2099</td>
<td>5A LOOP10</td>
<td>MOV E,D</td>
<td>MOVE D TO E</td>
</tr>
<tr>
<td>209A</td>
<td>15 DCR D</td>
<td>IS D =0</td>
<td></td>
</tr>
<tr>
<td>209B</td>
<td>CA 82 20</td>
<td>JZ LOOP9</td>
<td>YES, GO TO LOOP9</td>
</tr>
<tr>
<td>209E</td>
<td>3A B2 20</td>
<td>LDA 20B2</td>
<td>NO, LOAD DIVIDEND</td>
</tr>
<tr>
<td>20A1</td>
<td>1E 00 MVI E,0</td>
<td>MOVE 0 TO E</td>
<td></td>
</tr>
<tr>
<td>20A3</td>
<td>C3 82 20</td>
<td>JMP LOOP9</td>
<td>GO TO LOOP9</td>
</tr>
<tr>
<td>20A6</td>
<td>OD LOOP11</td>
<td>DCR C</td>
<td>IS COUNTER =0</td>
</tr>
<tr>
<td>20A7</td>
<td>C2 77 20</td>
<td>JNZ LOOP8</td>
<td>NO, GO TO LOOP8</td>
</tr>
<tr>
<td>20AA</td>
<td>CF RS1</td>
<td>YES, STOP</td>
<td></td>
</tr>
<tr>
<td>20B0</td>
<td>Divisor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20B1</td>
<td>NDiv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20B2</td>
<td>Divid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20B3</td>
<td>QMSB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20B4</td>
<td>QLSB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20B5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20B6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20B7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20B8</td>
<td>01 F1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20B9</td>
<td>F2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20BA</td>
<td>F3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20BB</td>
<td>F4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20BC</td>
<td>F5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20BD</td>
<td>F6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20BE</td>
<td>F7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20BF</td>
<td>F8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 10. (Continued)
Software Simulation

The computer program in Figure 10 provides a software simulation to verify the design of the divide algorithm. The program was written for the SDK-85 microprocessor board made by Intel. Several combinations of the dividend (N) and the divisor (D) were used, and the results were recorded in Table 2.

Other Divide Algorithms

There are a number of interesting division algorithms presented in the referenced literature. These algorithms perform division in sequential or combinational logic. Two of these divide algorithms are highlighted below.

Algorithm 1

In an article titled "Analysis of Speed of a Binary Divider Using a Variable Number of Shifts per Cycle" by M. R. Patel and K. H. Bennett (1977), an analysis for a variable number of shifts per cycle divider is presented. The article indicates that the speed may be increased further by providing multiples of the divisor which are negative integral powers of two (± 0.5 x divisor, ± 0.25 x divisor). A dividend prefix number is used in this algorithm. The prefix number is inspected every cycle to decide whether to enter the ADD.SHIFT mode or the ADD.SHIFT mode. Algorithm implementation is by means of sequential logic. A block diagram of the algorithm implementation is shown in Figure 11.
# TABLE 2

RESULTS OF THE SOFTWARE SIMULATION

<table>
<thead>
<tr>
<th>N (HEX)</th>
<th>D (HEX)</th>
<th>Q (HEX)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0F</td>
<td>03</td>
<td>05.00</td>
</tr>
<tr>
<td>08</td>
<td>03</td>
<td>02.AA</td>
</tr>
<tr>
<td>09</td>
<td>02</td>
<td>04.80</td>
</tr>
<tr>
<td>AA</td>
<td>05</td>
<td>22.00</td>
</tr>
<tr>
<td>AB</td>
<td>05</td>
<td>22.33</td>
</tr>
<tr>
<td>05</td>
<td>45</td>
<td>00.12</td>
</tr>
<tr>
<td>05</td>
<td>0A</td>
<td>00.80</td>
</tr>
<tr>
<td>05</td>
<td>14</td>
<td>00.40</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>01.0C</td>
</tr>
<tr>
<td>7F</td>
<td>FF</td>
<td>00.7F</td>
</tr>
<tr>
<td>80</td>
<td>FF</td>
<td>00.80</td>
</tr>
</tbody>
</table>
Figure 11. Algorithm 1 Block Diagram
Algorithm 2

In an article titled "A Pipelined Processing Unit for On-Line Division" by M. J. Irwin (1978), a processing unit which could be used as one stage of a pipeline fractional processing during division is presented. The unit is capable of operating in a serial stand alone manner where the operands are supplied one digit per cycle and the result is generated one digit per cycle. The unit could be linked with other like units for high speed processing as shown in Figure 12. The unit block diagram is shown in Figure 13 and consist of the following major blocks:

1. Input/output registers -- latches input and output data.
2. Multi-input adder -- processes full precision operands.
3. Result digit selector -- selects quotient digits.
4. Residual adder -- calculates residual for the next processing unit.
5. Selection network 1 -- generates the required multiples of $D_{j-1}$ and $Q_j$.
6. Selection network 2 -- generates $n_j+\delta - Q_jd_j+\delta$.
7. Carry generation -- needed if radix complement representation of negative numbers is used.

The dividend and divisor are assumed to be in normalized form upon input to the unit. The first quotient digit, $q_1$, can be properly selected after delta leading digits each of the dividend and divisor are known. Thereafter, one new digit of the quotient can be determined upon the receipt of one new digit each from the dividend and divisor. The quotient digit selector is a Table look-up device which implements
Figure 12. Algorithm 2 Pipelined Diagram
Figure 13. Pipeline Stage Divider Block Diagram
the SELECT function. It examines alpha most significant digits of \( \text{rp}_j \) and beta most significant digits of \( D_j \) to select the appropriate quotient digit, \( q_{j+1} \). Alpha and beta are determined to give sufficient precision to the divide algorithm. The complexity of the selection network and adders increases for higher radices.

There are differences and similarities between Irwin's pipeline stage divider and the proposed gate-array divider. Some of these differences and similarities are listed below.

1. Both units utilize input and output registers.

2. Both units could be used on-line or as a stand alone divider.

3. Inputs to the pipeline stage divider are assumed to be in normalized form upon input to the unit. Inputs to the gate-array divider are not required to be in normalized form. Normalization for the divisor is done internally.

4. Pipeline stage divider requires delta operand digits to be present in the first divide cycle and an additional digit for each following cycle. Gate-array divider requires all operand digits to be present at the same time.

5. Pipeline stage divider selects quotient digits by utilizing a table look up. Gate-array divider selects quotient digits by performing a full comparison between the dividend and divisor (the comparison is done by means of operands subtraction and by generating control bits which determine the quotient digits).

6. Pipeline stage divider could be implemented for different radix values. Gate-array divider could be implemented for different number of bits.

A pipeline stage divider with radix \( r=2 \) is functionally equivalent to one gate-array divide cell and 8 of these pipeline stage dividers are functionally equivalent to an 8 bit gate-array divider. However a pipeline divider with \( r=8 \) is not equivalent to an 8 bit
gate-array divider because one digit consists of 3 bits. For a pipeline stage divider with $r=2$, $\delta=5$, $\alpha=4$, $\beta=0$, and a precision of 8 bits, each stage divider will consist of the following:

1. $P_0$ input register (8 bits).
2. $D_j$ input register (8 bits).
3. Selection network to provide $q_jD_{j-1}$, since $q_j$ could only be 1 or 0. An 8 bit 2:1 mux circuit could be used to perform this selection.
4. Selection network to provide $r_{P_{j-1}} - q_jD_{j-1}$ for the next stage.
5. Multi-input adder to compute $P_j$, where:
   \[ P_j \leftarrow r^\delta (n_{j+\delta} - Q_{j+\delta}) + r_{P_{j+1}} - q_jD_{j-1}. \]
6. Residual adder to calculate $r_{P_{j-1}} - q_jD_{j-1}$ for the next stage.
7. Result digit selector which consists of a 32x1 table look up.
8. Output registers for full precision quotient and residual.
9. Carry generation hardware.

In comparison with a gate-array divide cell, this pipeline stage divider requires extensive hardware. The selection network gets more complicated for higher radices and hardware requirement increases as well.
A 9 bit sign magnitude divider design has been presented. The divider is suitable for high speed pipeline and stand alone processing. Size analysis indicates that the divider could be implemented on an LSI gate-array circuit, consisting of approximately 2111 gates. The Divider design consists of combinational logic, where input data is latched into input registers and output data is latched into output registers. The divisor is normalized and then passed to all the divide stages, while the dividend is passed only to the MSB divide stage. The division process starts in the MSB stage and propagates through to the LSB stage with the remainder from the MSB stage becoming the dividend for the next stage and so on. Speed analysis indicates that the maximum propagation delay through the divider stages is approximately 132 nsec. The divider is capable of handling any 9 bit sign magnitude numbers (integers and fractions). The design could be expanded to handle more bits (larger numbers). However, the number of gates, I/O pins, and the propagation delay will increase when more bits are added.

Algorithm 1 divider is a sequential logic implementation suitable for stand alone division. The mean division speed for algorithm 1 divider is 2.3 bits/cycle. Algorithm 2 is a combinational logic implementation suitable for pipeline and stand alone processing. There are some similarities and differences between the gate-array divider and the pipeline stage divider (see Algorithm 2 for details).
REFERENCES


