Design of On-Line Division Unit

Paul K.-G. Tu and Miloš D. Ercegovac
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90024

Abstract

A gate array implementation of a radix-2 floating-point on-line division algorithm is presented. The design requires 111 equivalent gates per bit and has a cycle time of 24ns. For 8-bit exponent and 24-bit mantissa, the design requires 2497 equivalent gates and can fit on an LSI Logic LL9320P chip with a utilization factor of 78%.

1 Introduction

In this paper, we present a gate array implementation of on-line division. We include the derivation of binary level algorithm and design parameters, the gate-array design, and its performance characteristics. We choose the radix-2 floating-point on-line division algorithm, since division is the most complex among the basic on-line operations, and also because other on-line operations such as multiplication and square root can be realized with minor modifications of the division design. We choose radix-2 because it can serve as a basic measuring stick for higher radix on-line arithmetic designs.

2 The radix-2 on-line division algorithm

The radix-2 floating point on-line division algorithm described here is based on the algorithm and derivation presented by Trivedi and Ercegovac[4], and by Trivedi and Rusnak[5]. We modify the algorithm and derive key parameters, taking into consideration the following constraints, (i) the input operands in on-line arithmetic are not guaranteed to be normalized, and (ii), the recurrence computation is implemented in redundant form.

Assuming that the mantissa of the input divisor, denoted by D', is quasi-normalized[8], we have $D' \in [2^{-2}, 1)$, which means that it has at most one leading zero. After accumulating the first $\delta - 1$ input bits, where δ is the on-line delay, the initial input divisor, denoted as $D'_0 = \sum_{i=1}^{\delta-1} d_i 2^{-i}$, is conditionally shifted, and the initial value of the divisor actually used in the quotient computation, denoted by D_0, is obtained as

$$D_0 = \begin{cases} D'_0 & \text{if } D'_0 \geq 2^{-1} \\ 2 \cdot D'_0 + d_\delta 2^{-\delta+1} & \text{if } D'_0 < 2^{-1} \end{cases}$$

We then have, for all $j \geq 0$, $D_j \in (2^{-1} - 2^{-\delta+1}, 1)$.

Two related parameters affect the performance of on-line division. One is the on-line delay, δ, which is defined as the number of digits of each operand accumulated when the first digit of the output is generated. The other parameter is the precision of the digit selection function, k, defined as the number of fractional digit positions of the argument required by the selection function. There is a tradeoff between δ and k, and small values of both of them are desirable for best performance. Based on the analysis given in [6], we choose $\delta = 5, k = 4$.

In the following floating-point on-line radix-2 division algorithm e_n, e_d and e_q denote the exponents of the dividend, divisor, and quotient, n_{next}, d_{next} and q_j denote digits of their mantissas, and D_j and Q_j are accumulated values of j fractional digits of the divisor and quotient, respectively. The subscript *next* denotes the subscript of the next incoming digit. In step j, $next = j + 4$ if the divisor was not shifted during initialization. Otherwise, $next = j + 5$. The dividend mantissa is shifted right one bit position to avoid possible overflow of the result.

Algorithm *OLDIV* [Radix-2 on-line division]
step 1. [Initialization and shifting]
$e_q \leftarrow e_n - e_d + 1$
$A_0 \leftarrow \sum_{i=1}^{3} n_i 2^{-i-1}$
$D_0 \leftarrow \sum_{i=1}^{4} d_i 2^{-i}$

if $D_0 < 2^{-1}$ **then**
$D_0 \leftarrow 2 \cdot D_0 + d_{next}2^{-4}$; $e_q \leftarrow e_q - 1$
$Q_0 \leftarrow 0$; $q_0 \leftarrow 0$
step 2. [quotient generation]
for j = 1, $\cdots$, m **do**
$D_j \leftarrow D_{j-1} + d_{next}2^{-j-4}$
$A_j \leftarrow 2(A_{j-1} - q_{j-1}D_j) + n_{next}2^{-4} - d_{next}Q_{j-2}2^{-4}$
$q_j = select(A_j) \leftarrow \begin{cases} -1 & \text{if } A_j < -\frac{1}{4} \\ 0 & \text{if } -\frac{1}{4} \le A_j < \frac{3}{16} \\ 1 & \text{if } A_j \ge \frac{3}{16} \end{cases}$
$Q_j \leftarrow Q_{j-1} + q_j 2^{-j}$
end *OLDIV.*

3 Functional components of on-line division

The scheme implements exponent and mantissa calculation. Exponent calculation for division includes the following functions, which are performed during step 1 of the algorithm *OLDIV*:

1. $e_q \leftarrow e_n - e_d + 1$
2. **if** $D_0' < \frac{1}{2}$ **then** $e_q \leftarrow e_q - 1$

We assume that the exponents of the input and output are in parallel form, so that the exponent calculation is implemented by a conventional parallel adder. The mantissa calculation is performed in on-line fashion. During each time step, the following recurrence expression is evaluated to obtain intermediate result A_j,

$$A_j \leftarrow 2(A_{j-1} - q_{j-1}D_j + n_{next}2^{-5} - d_{next}Q_{j-2}2^{-5}) \quad (1)$$

The next quotient digit is calculated as a function of A_j.

The following functional components are to be realized in on-line division mantissa calculation. In this paper, a signed bit is referred to as a *sbit.*

Data accumulator The values of q_j and d_j need to be accumulated to provide Q_j and D_j for the recurrence evaluation. In each step, the following function is performed,

$$D_j \leftarrow D_{j-1} + d_{next}r^{-j-\delta} \quad (2)$$

Single sbit multiplier The terms $q_{j-1}D_j$ and $d_{next}Q_{j-2}$ in the recurrence are generated by multiplying a fractional number by a single sbit.

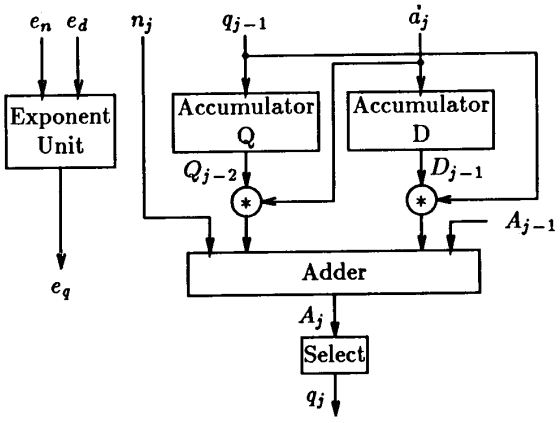

Figure 1: Functional components for on-line division

Adder A multiple input adder generates A_j as the sum of A_{j-1} from the previous time step, the product terms $q_{j-1}D_j$ and $d_{next}Q_{j-2}$, and input sbit n_{next}.

Selection The quotient digit selection function calculates the next quotient digit from A_j. Since

$$|A_j| < 2 - 2^{-3} - \frac{2}{3} \cdot 2^{-j-3} < 2$$

the input to the digit selection function will involve 1 sign bit position, 1 integer bit position, and 4 fractional bit positions, for a total of 6 bit positions.

Fig.1 illustrates the functional components of on-line division.

4 Binary level algorithm

In this section we specify binary level algorithms for each functional component of on-line division. The mantissa computation specified in *OLDIV* is modified such that the initialization step is incorporated as part of the quotient calculation step with simple control requirements. Then operations for each functional component are discussed in detail.

The $n_{next}2^{-5}$ term in (1) represents a shifted single sbit, and it is desirable not to let this single sbit cause extra delay to the overall computation. Since

$$|-d_{next}Q_{j-1}| < 1$$

and in 2's complement representation we obtain the terms $n_j 2^{-5}$, $P = -d_{next}Q_{j-1}2^{-5}$, and their sum as

$$\begin{array}{l} n_0 \,.\, n_0\ n_0\ n_0\ n_0\ n_1\ 0\ 0 \cdots 0 \\ \underline{p_s \,.\, p_s\ p_s\ p_s\ p_s\ p_s\ p_1\ p_2 \cdots p_{j-1}} \\ x \,.\, x\ x\ x\ x\ y\ p_1\ p_2 \cdots p_{j-1} \end{array}$$

where x and y are simple switching functions of n_0, n_1 and p_s,

$$x = n_0 + \overline{n}_1 p_s \quad (3)$$

$$y = n_1 \oplus p_s \quad (4)$$

The terms $n_{next}2^{-5}$ and $-d_{next}Q_{j-1}2^{-5}$ can be combined as one term for the multiple input adder.

The initialization step of algorithm *OLDIV* requires accumulating incoming sbits of N and D. In order to achieve an efficient design, we want to minimize the difference between computations performed during initialization and quotient generation. The expression for D_0 is the same as that for D_j in the quotient generation step. By keeping $q_j = 0$ in the recurrence expression (1), the expression for computing A_j during quotient digit generation becomes the same as that for computing A_0 during initialization. This simplifies the control mechanism for the recurrence evaluation. The modified mantissa algorithm (higher level) follows.

Algorithm *OLDIVM*
[Modified radix-2 on-line division algorithm]
Initialization
$A_{-4} = 0,\ D_{-4} = 0,\ Q_{-5} = 0$
$q_{-4} = 0,\ j = -3$
begin [recurrence]
$D_j \leftarrow D_{j-1} + d_{next}2^{-j-4}$
$A_j \leftarrow 2(A_{j-1} - q_{j-1}D_j + n_{next}2^{-5} - d_{next}Q_{j-2}2^{-5})$
$q_j \leftarrow \begin{cases} 0 & \text{if } j \le 0 \\ select(A_j) & \text{if } j > 0 \end{cases}$
$Q_{j-1} \leftarrow Q_{j-2} + q_{j-1}2^{-j+1}$
if $j = 0$ **and** $D_0 < 2^{-1}$ **then**
$D_0 \leftarrow D_0 \cdot 2$
else if $j \ge m$ **then stop**
else $j \leftarrow j + 1$
end [recurrence]
end *OLDIVM.*

4.1 Data accumulation and conversion

While the input and output of the on-line division computation are in signed-digit representation, the internal computation can be implemented in either redundant or non-redundant form. For speed consideration, it is advantageous to use redundant adders to avoid full precision carry propagation. We choose to implement the recurrence expression evaluation in 2's complement number system with carry-save adders, and to accumulate D_j and Q_j in 2's complement form. Another possibility is to implement this computation in signed-digit form with signed-digit adders.

init/ld	previous bit·flag	input 0	input 1	input −1
0/0	0c	0c	0c	0c
	0u	0u	0c	1c
	1c	1c	1c	1c
	1u	1u	1c	0c
1/*		0c	0c	0c
0/1		0u	1u	1u

Table 1: Conversion transition table

An on-the-fly conversion algorithm is given by Ercegovac and Lang[3] which converts a series of signed-digits into non-redundant parallel form. We adopt the modified version of this algorithm given by Tullsen[7], which is slightly simpler. Here we briefly describe Tullsen's algorithm. For more details of the data conversion algorithms, refer to [3] and [7].

In this algorithm, each bit position has a flag, which indicates whether the current bit is *confirmed(c)* or *unconfirmed(u)*. An *init* signal applies to all bit positions at the beginning of the process, and a *ld* signal is applied to each bit position in succession(i.e., the *ld* signal is applied to bit i when the ith sbit is available). For each bit, its value and flag are set according to the transition table given in Table 1.

In the following, we use superscripts to denote the bit position of a vector element within the vector, subscripts to denote the time steps, and use upper case latters to denote vectors, lower case letters to denote single bits of the vectors. At time step j, let x_j denote the input sbit, A_j the accumulated value of sbit series $x_1, x_2, \cdots, x_j$, C_j the vector of flags associated with A_j, and $\vec{Z}_j = (z_j^{(1)}, z_j^{(2)}, \cdots, z_j^{(p)})$ a shift register, where

$$z_j^{(l)} = \begin{cases} 1 \text{ if } & l = j \\ 0 \text{ if } & l \ne j \end{cases}$$

Then, the data conversion algorithm is the following.

$$(\vec{A}_j, \vec{C}_j, \vec{Z}_j) \leftarrow convert(\vec{A}_{j-1}, \vec{C}_{j-1}, \vec{Z}_{j-1}, x_j, init)$$

where the function *convert* is defined as

$$a_j^{(i)} \leftarrow f(x_j, a_{j-1}^{(i)}, c_{j-1}^{(i)}, z_j^{(i)}, init)$$
$$c_j^{(i)} \leftarrow g(x_j, c_{j-1}^{(i)}, z_j^{(i)}, init)$$
$$z_j^{(i)} \leftarrow z_{j-1}^{(i-1)}$$

The functions f and g are defined according to Table 1, with ld substituted by $z_j^{(i)}$.

4.2 Multiply by a single sbit

In the recurrence expression evaluation, we need to implement computations of the form $Z \leftarrow x_j \cdot Y$, where Z and Y are fractional numbers, and x is a single sbit. In 2's complement computation, this can be obtained by the following function,

$$Z = \begin{cases} Y & \text{if } x_j = 1 \\ 0 & \text{if } x_j = 0 \\ -Y & \text{if } x_j = -1 \end{cases}$$

where $-Y$ can be generated by bitwise complementing Y, and adding 1 to the right-most bit position to obtain its 2's complement. The binary level algorithm for the single sbit multiplier is the following.

$$z^{(i)} \leftarrow \begin{cases} y^{(i)} & \text{if } \vec{x}_j = \langle 01 \rangle \\ 0 & \text{if } \vec{x}_j = \langle 00 \rangle \\ \overline{y^{(i)}} & \text{if } \vec{x}_j = \langle 11 \rangle \end{cases}$$

4.3 Multiple input adder

Since the recurrence evaluation is performed in carry-save form, there are 4 input terms to the adder. Two levels of 3-to-2 reductions are used to realize the required 4-to-2 carry-save adder(Fig.2).

4.4 Quotient digit selection

Since the result A_j of the recurrence expression evaluation is in carry-save form, the quotient digit selection function is divided into two parts(Fig. 3). First the most significant portion of A_j is converted into non-redundant form through a carry-assimilation adder. Then the next quotient digit is generated.

As discussed in Section 3, the quotient digit selection function requires the input of 6 bit positions, hence a 6-bit adder is used. Carry-lookahead is used to reduce the time required for this function. The following is the algorithm for the carry-assimilation adder.

$$\vec{X} \equiv \langle x^{(0)}x^{(1)}x^{(2)}x^{(3)}x^{(4)}x^{(5)} \rangle$$
$$\vec{Y} \equiv \langle y^{(0)}y^{(1)}y^{(2)}y^{(3)}y^{(4)}y^{(5)} \rangle$$
$$\vec{Z} \equiv \langle z^{(0)}z^{(1)}z^{(2)}z^{(3)}z^{(4)}z^{(5)} \rangle$$
$$\vec{Z} \leftarrow cpa6(\vec{X}, \vec{Y})$$

where

$$z^{(i)} \leftarrow \begin{cases} x^{(i)} \oplus y^{(i)} & i = 5 \\ x^{(i)} \oplus y^{(i)} \oplus c^{(i)} & i = 0, 1, 2, 3, 4 \end{cases}$$

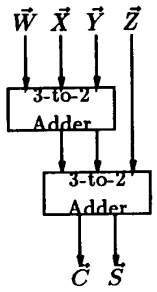

Figure 2: Structure of 4-to-2 adder

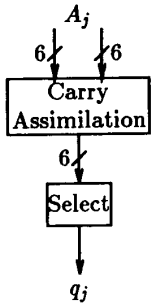

Figure 3: Quotient digit selection

$$\begin{aligned} c^{(4)} &= x^{(5)}y^{(5)} \\ c^{(3)} &= x^{(4)}y^{(4)} + (x^{(4)} + y^{(4)})x^{(5)}y^{(5)} \\ c^{(2)} &= x^{(3)}y^{(3)} + (x^{(3)} + y^{(3)})x^{(4)}y^{(4)} \\ &\quad +(x^{(3)} + y^{(3)})(x^{(4)} + y^{(4)})x^{(5)}y^{(5)} \\ c^{(1)} &= x^{(2)}y^{(2)} + (x^{(2)} + y^{(2)})c^{(2)} \\ c^{(0)} &= x^{(1)}y^{(1)} + (x^{(1)} + y^{(1)})x^{(2)}y^{(2)} \\ &\quad +(x^{(1)} + y^{(1)})(x^{(2)} + y^{(2)})c^{(2)} \end{aligned}$$

The quotient digit generation function is defined as

$$\begin{aligned} \vec{q} &\equiv \langle q^{(0)}q^{(1)} \rangle \quad a^{(i)} \equiv z^{(i)} \\ q^{(0)} &\leftarrow a^{(0)}(\overline{a^{(1)}} + \overline{a^{(2)}} + \overline{a^{(3)}} + \overline{a^{(4)}} \cdot \overline{a^{(5)}}) \\ q^{(1)} &\leftarrow a^{(0)}(\overline{a^{(1)}} + \overline{a^{(2)}} + \overline{a^{(3)}} + \overline{a^{(4)}} \cdot \overline{a^{(5)}}) \\ &\quad +\overline{a^{(0)}}(a^{(1)} + a^{(2)} + a^{(3)}) \end{aligned}$$

5 Pipelining

To minimize the step time of the computation, we use a two-stage pipeline scheme, as shown in Fig.4, where the shaded boxes indicate the buffers.

In stage 1, the following calculations are performed.

$$D_j \leftarrow D_{j-1} + d_{next}2^{-j-4}$$
$$A'_j \leftarrow 2(A'_{j-1} - q_{j-2}D_{j-1}) + n_{next}2^{-5}$$

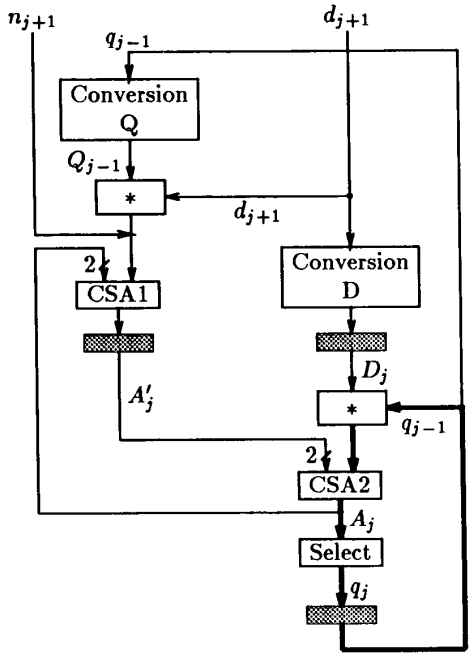

Figure 4: Pipeline scheme for on-line division

$$-d_{next}Q_{j-2}2^{-5}$$
$$Q_{j-1} \leftarrow Q_{j-2} + q_{j-1}2^{-j+1}$$

And in stage 2, q_j is generated.

$$q_j \leftarrow select(2(A'_j - q_{j-1}D_j))$$

Fig.5 illustrates the timing of the mantissa computation. The most time consuming part in the computation is the quotient digit selection function, which includes the 6-bit carry assimilation adder and quotient digit generation logic. The critical path, which is shown in thick lines in Fig.4, is that from the q_j buffer output to the signed digit multiplier to *CSA2*

Step	$\cdots$	$j-2$	$j-1$	j	$j+1$	$\cdots$
Input	$\cdots$	d_{j-1}	d_j	d_{j+1}	d_{j+2}	$\cdots$
	$\cdots$	n_{j-1}	n_j	n_{j+1}	n_{j+2}	$\cdots$
Values	$\cdots$	D_{j-1}	D_j	D_{j+1}	D_{j+2}	$\cdots$
Calculated	$\cdots$	Q_{j-3}	Q_{j-2}	Q_{j-1}	Q_j	$\cdots$
	$\cdots$	A'_{j-1}	A'_j	A'_{j+1}	A'_{j+2}	$\cdots$
	$\cdots$	q_{j-2}	q_{j-1}	q_j	q_{j+1}	$\cdots$

Figure 5: Illustration of timing for on-line division mantissa computation

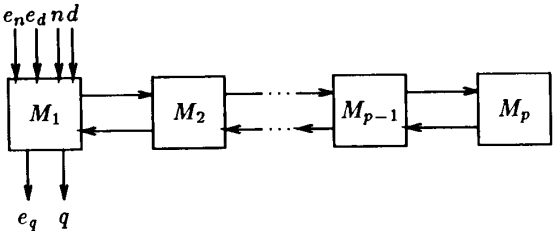

Figure 6: Modular structure of on-line division unit

to *Select* to the q_j buffer input. The pipeline scheme given here allows extra time in stage 1 for the distribution of q_j and d_j.

6 Organization and design of on-line division unit

The on-line division unit is organized as a linear array of modules, as shown in Fig.6. The first module M_1, shown in Fig.7, contains components for generating control signals($CTRL$), exponent calculation(EU), quotient digit selection($SELECT$), and bit-slices for the most significant portion of the recurrence calculation, which includes the sign, integer bit positions, and 5 fractional bit positions. The modules $M_2, \cdots, M_p$ are identical, each containing a number of bit-slices of the recurrence expression evaluation.

From the algorithm *OLDIVM*, the number of fractional bit positions involved in the on-line recurrence computation is $j+4$ at step j. To generate the jth quotient digit, only the 4 most significant fractional bit positions plus the sign and integer parts are needed, so it is not necessary to carry out the recurrence expression evaluation in full precision. Let n_{frac} denote the number of fractional bit-slices used in the selection function, n_{int} the number of bit-slices in the integer portion including the sign bit, and δ be the on-line delay. To generate m quotient bits, the number of bit-slices needed is

$$n = \left\lceil \frac{m}{2} + \frac{\delta + n_{frac}}{2} \right\rceil + n_{int} = \left\lceil \frac{m+9}{2} \right\rceil + 1 \quad (5)$$

For on-line division of 24-bit precision, 18 bit-slices are needed.

Assuming an 8-bit exponent, the main component of EU is an 8-bit parallel adder with carry lookahead. Fig. 8 shows organization of the exponent component.

Each bit-slice includes functional components for data conversion, signed digit multiplication, and

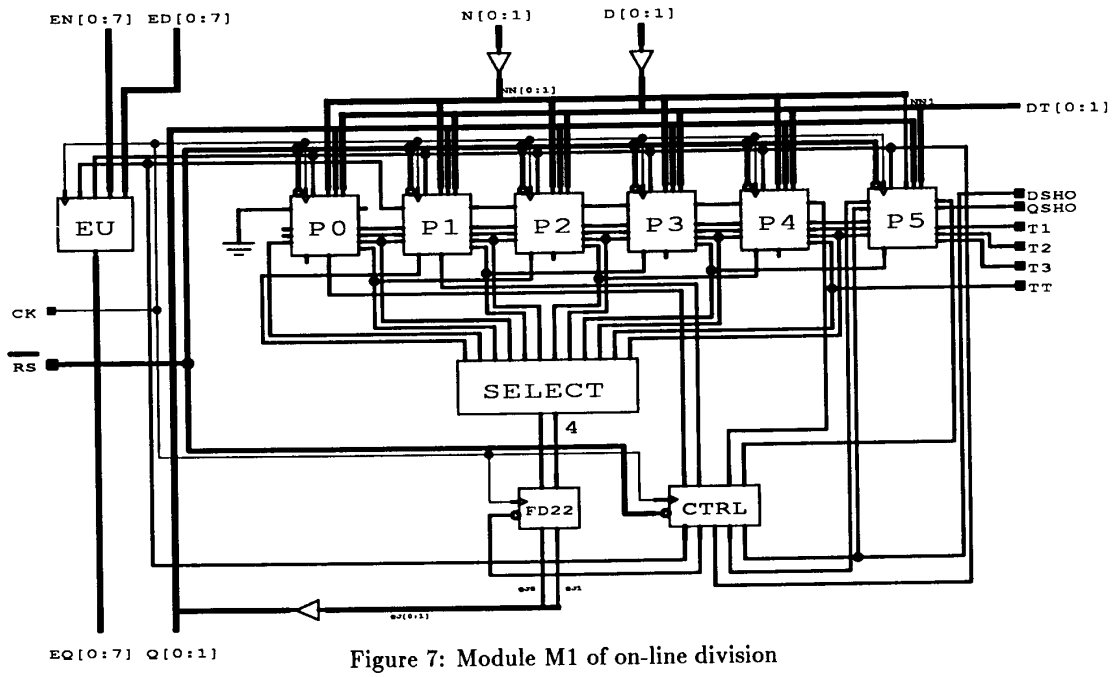

Figure 7: Module M1 of on-line division

carry-save addition. Fig. 9 shows the bit-slice organization.

Two slightly different designs of the bit-slices are used in module M_1 to incorporate the $n_{next}2^{-5}$ term in the recurrence expression. As was discussed in section 4.2, the most significant bit positions in the recurrence expression evaluation need to generate the values x and y, which are defined in (3) and (4).

The quotient digit selection function is composed of carry assimilation and quotient digit generation. The carry assimilation is realized by a 6-bit adder with carry lookahead.

7 Design characteristics

To estimate the implementation characteristics, the on-line division unit design is captured and simulated with the WORKVIEW[1] CAD tools[2] and the LSI[2] Design Kit[1] by Viewlogic Systems, inc.. The gate level complexity of the design is measured in terms of *equivalent gate*, which is equivalent to two n-channel and two p-channel transistors. The total gate count for the on-line division design is

$$G_{div} = G_{M1} + G_{bit\text{-}slice} \times n'$$

[1]WORKVIEW is a trademark of View*logic* Systems, Inc..
[2]LSI is a trademark of LSI Logic Corporation.

component	equiv. gate
EU	248
P0, P1, ..., P5	$6 \times 119 = 714$
SELECT	72
Control, Buffers	69
M_1 Total	1103

Table 2: Gate count of M_1

where n' is the number of bit-slices in modules M_2, $\cdots$, M_p. M_1 contains the most significant 6 bit-slices, so from (5) we have

$$n' = \left\lceil \frac{m+9}{2} \right\rceil + 1 - 6$$

Table 2 shows the equivalent gate counts of the components of module M_1. M_1 consists of components necessary for the generation of quotient mantissa digits and exponent, and has a fixed gate count regardless of the precision of the computation. Table 3 shows the component gate count of a bit-slice. An on-line division unit for single precision floating point operands with an 8-bit exponent and 24-bit mantissa has a total gate count of 2497, including the input/output drivers for the chip, and can be

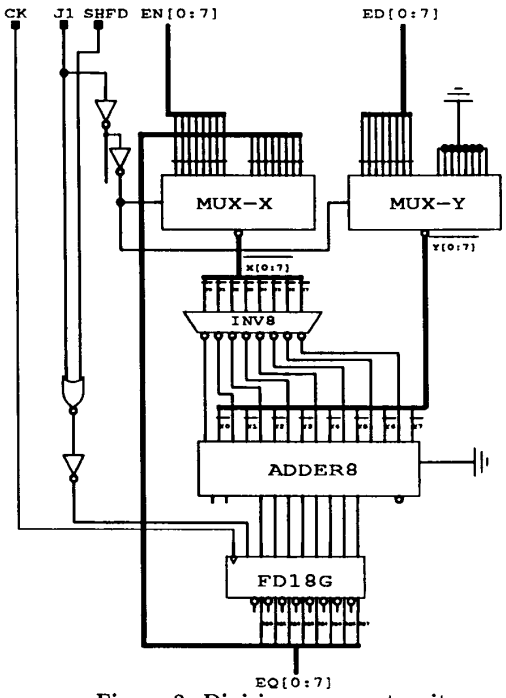

Figure 8: Division exponent unit

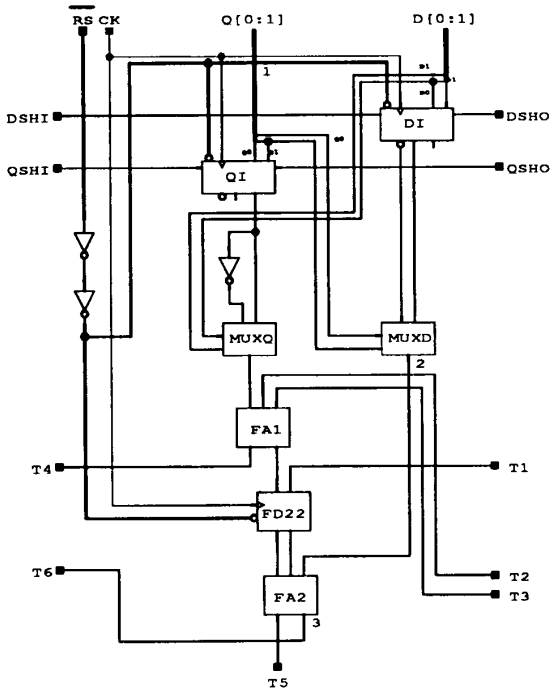

Figure 9: Bit-slice for on-line division

implemented on a LL9320P gate array chip with a utilization factor of 78%. The dimension of LL9320P is 6.19 x 5.83 mm.

Fig.10 shows the dependencies between components in the cycle period of the exponent and mantissa calculation. The time delays of the major components are shown in the timing diagram in Fig.11. The figures are estimates based on the load driven by each component, design parameters of the LSI LL9000 series gate array components, and average wire lengths without performing the actual routing and layout design. In Fig.11, *FD* denotes the transition of D flipflops, and *setup* is their setup time. *CONVERT* is the time for converting from signed digit form to 2's complement form. n_j *I/O* and d_j *I/O* are the input pad buffer delays for the dividend and divisor. Dashed lines in the diagram denote components that are connected in sequence but not in the critical path. The step time for the design is determined by the delays of the multiplier, the full adder, and the quotient digit selection unit, as is indicated by the critical path in Fig.4. The points *1*, *2*, *3* and *4* in Fig.11 refer to points along the critical path labeled in Fig.7 and Fig.9. The minimum step time, with the LSI LL9000 series components, is 24ns. This figure includes the input pad buffers for the dividend and divisor, but does not include the output pin drivers required for the quotient. The delay of the output drivers depends on the load of the output pins for the quotient.

component	equiv. gate
conversion	2 x 32 = 64
sbit multiply	2 x 6 = 12
FA	2 x 10 = 20
Buffers and drivers	15
bit-slice total	111

Table 3: Gate count of bit-slice

8 Summary

We have discussed the radix-2 floating-point on-line division algorithm from the implementation point of view. Under the assumption that the operands are guaranteed to be quasi-normalized, we selected the values of the on-line delay $\delta = 5$ and the precision of the quotient digit selection function $k = 4$.

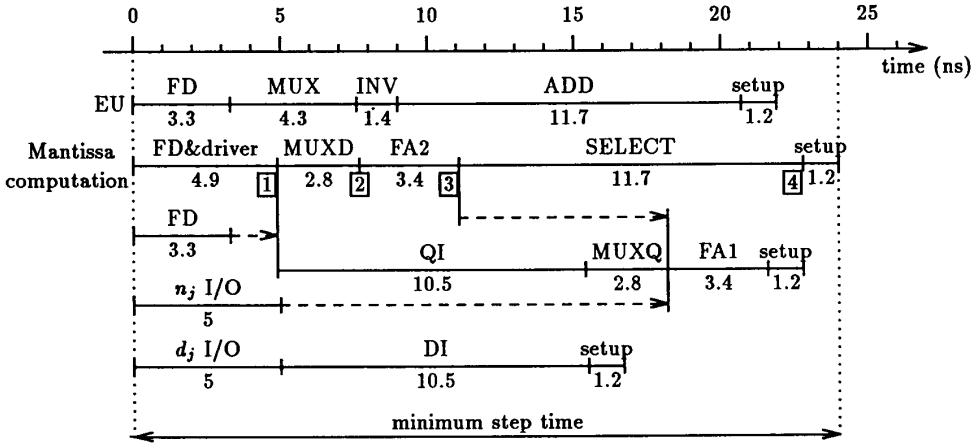

Figure 11: Timing of on-line division

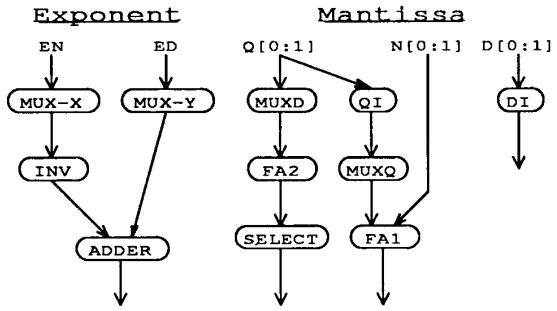

Figure 10: Dependency graph for on-line division step

We then presented a modular design of the on-line division unit, using the LSI Logic HCMOS gate array technology. To increase the precision of the computation will only require some more bit slices, and the step time will not be affected. Since full precision calculation of the recurrence expression is not needed in on-line division, it requires fewer bit slices than parallel schemes.

Acknowledgments This research has been supported in part by the NSF Grant No. MIP-8813340 *Composite Operations Using On-Line Arithmetic for Application-Specific Parallel Architectures: Algorithms, Design, and Experimental Studies.* The authors thank Dr. Tomas Lang for interest and constructive comments.

References

[1] *LSI Design Kit, Release 1.0.* VIEWlogic Systems, Inc., 313 Boston Post Road West, Marlboro, Massachusetts 01752, September 1987.

[2] *WORKVIEW Manual Set, Release 3.0.* VIEWlogic Systems, Inc., 313 Boston Post Road West, Marlboro, Massachusetts 01752, June 1988.

[3] M. D. Ercegovac and T. Lang. On-the-fly conversion of redundant into conventional representations. *IEEE Transactions on Computers*, C-36(7):895–897, July 1987.

[4] K. S. Trivedi and M. D. Ercegovac. On-line algorithms for division and multiplication. *IEEE Transactions on Computers*, C-26(7):681–687, July 1977.

[5] K. S. Trivedi and J. G. Rusnak. Higher radix on-line division. In *Proceedings of the Fourth Symposium on Computer Arithmetic*, pages 164–174, Santa Monica, CA, Oct 1978.

[6] P. K. Tu. *On-line arithmetic algorithms for efficient implementations.* PhD thesis, University of California, Los Angeles, 1989.

[7] D. M. Tullsen. *A Very Large Scale Integration Implementation of an On-Line Arithmetic Unit.* Master's thesis, University of California, Los Angeles, 1986.

[8] O. Watanuki. *Floating-point on-line arithmetic for highly concurrent digit-serial computation: application to mesh problems.* Technical Report CSD810529, Computer Science Department, UCLA, May 1981.