Low-Power Radix-4 Combined Division and Square Root

Alberto Nannarelli and Tomas Lang
Department of Electrical & Computer Engineering
University of California, Irvine, California 92697
e-mail : alberto@ece.uci.edu, tlang@uci.edu

Abstract

Because of the similarities in the algorithm it is quite common to implement division and square root in the same unit. The purpose of this work is to implement a low-power combined radix-4 division and square root floating-point double precision unit and to compare its performance and energy consumption with a radix-4 division only unit. Previous work has been done on reducing the energy dissipated in a divider. Here we apply the same techniques to the combined division and square root unit and consider modifications and tradeoffs. Results show that the energy dissipation for the combined division/square root unit can be reduced by about 35% without affecting the latency and an additional 20% reduction can be obtained using a dual voltage. Moreover, the unit is 5% slower than a divider and its energy dissipation is 15% higher.

1 Introduction

Energy consumption is becoming more important every day because of the increased densities on chip and faster clocks. Although division and square root are infrequent operations, compared to addition and multiplication, because of their longer latency they dissipates a substantial part of the energy in floating-point units.

Because of the similarities in the algorithm it is quite common to implement division and square root in the same unit [4, 5, 12]. The purpose of this work is to implement a combined radix-4 division and square root floating-point double precision unit and to compare its performance and energy consumption with a radix-4 division only unit.

Previous work [9, 8, 10] has been done on reducing the energy dissipated in a divider. Here we are interested in evaluating the impact of the addition of the square root functionalities to the architecture presented in [9].

Techniques for the reduction of the power dissipation applicable to a variety of CMOS circuits are presented in [11, 13, 2], while others, specific for the algorithm in question, are presented in [9]. We apply those techniques to the combined division and square root unit and consider modifications and tradeoffs.

Results show that the energy dissipation for the combined division/square root unit can be reduced by about 35% without affecting the latency. An additonal 20% reduction is possible by using dual voltage. With respect to the radix-4 divider, the combined unit has a cycle time about 5% longer and a 15% higher energy consumption.

The units are implemented using the Passport $0.6\mu m$, $3.3V$, three-metal layers, CMOS standard cell library [3] and the layout is obtained by automatic floor-planning. The critical path is obtained from simulations, the area from the layout and the energy dissipation is estimated using PET [7], a power evaluation tool which computes the power dissipated in a circuit from the netlist extracted from the layout (that includes the effect of interconnection capacitance), the standard cell library energy views, obtained by characterization, and the results of a logic-level simulation. The simulation is run on a set of random-generated test vectors and stopped when a sufficient accuracy is reached in a Monte Carlo approach similar to the one presented in [1].

2 Algorithm and Implementation

Because of the similarities in the algorithm, division and square root can be effectively implemented in the same unit. The radix-4 combined division and square root, described in detail in [4], is implemented by the residual recurrence

$$w[j+1] = 4w[j] + F[j] \qquad\qquad j = 0, 2, \ldots 26$$

in which

$$F[j] = \begin{cases} -q_{j+1}d & \text{(division)} \\ -(S[j]s_{j+1} + \frac{1}{2}\, 4^{-(j+1)}s_{j+1}^2) & \text{(square root)} \end{cases} \tag{1}$$

Since the partial result is initialized to $Q[0] = 1.0$ and $S[0] = 1.0$,

$$w[0] = \begin{cases} x - d & \text{(division)} \\ \frac{1}{2}(x-1) & \text{(square root)} \end{cases}$$

where x is the dividend/radicand, and d the divisor. Both d and x are normalized in $[0.5, 1)$ and $x < d$ for division. The result digit (q_{j+1} for division and s_{j+1} for square root) are determined, at each iteration, by a selection function

$$q_{j+1} = SEL(d_\delta, \hat{y}) \quad \text{(division)}$$
$$s_{j+1} = SEL(\hat{S}[j], \hat{y}) \quad \text{(square root)}$$

where d_δ and $\hat{S}[j]$ are respectively d and $S[j]$ truncated after the 4th fractional bit, and $\hat{y}$ is an estimate of $4w[j]$. The result digit is in signed-digit representation $\{-2, -1, 0, 1, 2\}$ with redundancy factor $\rho = \frac{2}{3}$ and the residual $w[j]$ is stored in carry-save representation (w_S and w_C) to reduce the iteration time. In order to use $S[j]$ in the iterations, we need to convert the result digits from signed-digit to conventional representation. The on-the-fly conversion algorithm [4] performs this conversion as the digits are produced and does not require a carry-propagate adder. For this purpose, two variables A and B are required. They are updated, in every iteration, so that:

$$A[j] = S[j] \qquad \text{and} \qquad B[j] = S[j] - 4^{-j}$$

The recurrence is implemented, as shown in Figure 1 with the selection function (SEL), the block to form F (FGEN), a block (DSMUX) which provides FGEN with the appropriate bit vector (depending on the operation), a carry-save adder (CSA), and two registers to store the carry-save representation of the residual. The conversion block performs the conversion from signed-digit to conventional representation and the rounding. The result is rounded in the last iteration according to the sign of the final residual and the signal that detects if it is zero, which are produced by the sign-zero-detection block (SZD).

The number of bits required in the recurrence is 55 fractional plus 1 integer for a total of 56 bits. To execute the operation 28 cycles are required for the iterations plus one additional cycle for the rounding, for a total of 29 clock cycles.

2.1 Implementation of block DSMUX

Block DSMUX selects the inputs to block FGEN according to the operation (division or square root) under execution. If the operation is a division, the value to provide to FGEN is d, while for square root the value to provide to FGEN is the partial result $S[j] = A[j]$. If in the conversion block we implement the algorithm described in [9], the value of B can be derived from A and the state in register C. In conclusion, in block DSMUX we utilize the inputs (d, A, and C) to obtain the desired outputs (A_{OUT}, B) according to Table 1.

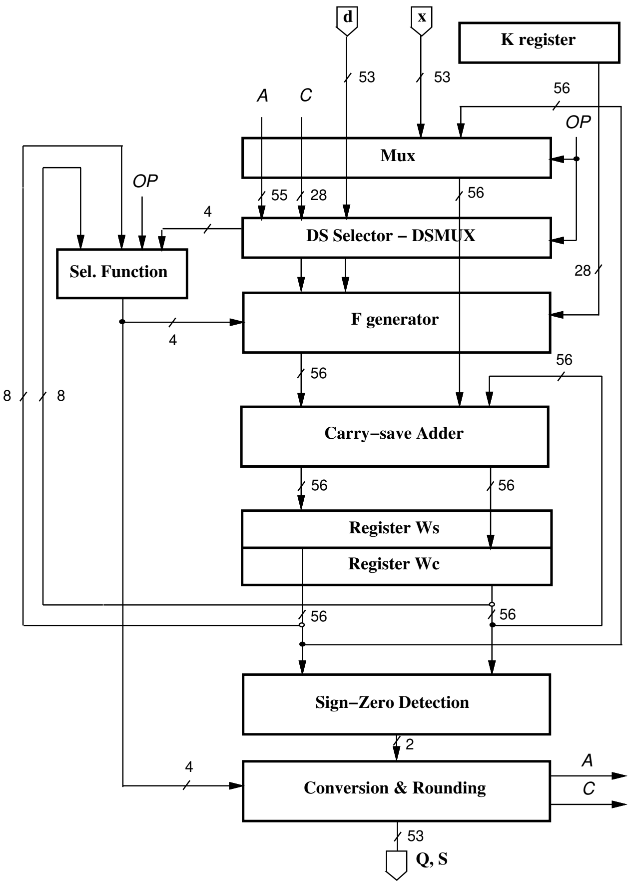

Figure 1. Radix-4 combined division/square root unit.

2.2 Implementation of Selection Function SEL

The selection function can be divided into two parts: an adder and a logic function. The adder is an 8-bit carry-propagate adder whose addends are the 8 MSBs of the carry-save representation of W. The 7 MSBs of the sum are used to generate the result digit at each iteration, along with 3 bits of either d or A chosen as follows:

- if the operation is division the 3 bits of d are those with weight 2^{-2}, 2^{-3}, and 2^{-4}.
- If the operation is square root, the 3 bits are chosen from A, as explained in [4], according to Table 2.

The selection logic function is described in Table 3. The digit h selected is the one satisfying the expression $m_h \leq \hat{y} < m_{h+1}$. The result digit is in the set $\{-2, -1, 0, 1, 2\}$ and four signals (M2, M1, P1, P2) are used to represent these five values in a one-out-of-four code (zero is coded as (0,0,0,0)). This representation makes the F-generator block (FGEN) simpler.

op.	A_{OUT}	B	
div	d	d	
sqrt	A	$\overline{C_i}A_{2i+1} + C_i(A_{2i+1} \oplus A_{2i})$	(odd bits)
		$\overline{C_i}A_{2i} + C_i\overline{A_{2i}}$	(even bits)
		$i = m, \ldots, 1, 0$	

Z_i refers to bit in position i in vector Z, being $i = 0$ the LSB.

Table 1. DSMUX operations.

1	0	-	first iteration ($j = 0$)
1	1	1	if ($A_{<0>} = 1$) and ($j > 0$)
$A_{<-2>}$	$A_{<-3>}$	$A_{<-4>}$	if ($A_{<0>} = 0$) and ($j > 0$)

$A_{<-k>}$ refers to bit in A with weight 2^{-k}. $A_{<-k>} = A_{55-k}$

Table 2. Bits of A used in SEL.

2.3 Implementation of block FGEN

Block FGEN generates the signal $F[j]$ as described in expression (1).

For square root the generation of F is quite complicated [4]. Table 4, where $a...aa$ and $b...bb$ represent bits of $A[j]$ and $B[j]$ respectively, shows the values of the bit-string for the different result digits. To keep track of the position, a 28-bit register (K) is used. The register K is initially loaded with 1 in the MSB and 0 in the remaining positions. At each iteration the 1 is replicated one position to the right according to the following expression.

$$K_i[j+1] = K_{i+1}[j], \quad K_{27}[j] = 1 \quad i = 0, 1, \ldots, 27$$

At the end of the square root all bits of K are 1. As can be seen, three zones are defined by the variables

$$K1_i = K_i K_{i-1}\ , \quad K2_i = K_i\overline{K_{i-1}}\ , \quad K3_i = K_{i+1}\overline{K_i}$$

and the relation between i and the bit h is $i = \lfloor h/2 \rfloor$. In terms of these variables we obtain

$$\begin{aligned} F_h \quad & (\text{odd } h) = \\ & K1_i(P2\overline{A_{h-1}} + P1\overline{A_h} + M1B_h + M2B_{h-1}) \\ & + K2_i(P2 + P1\overline{A_h} + M1B_h + M2) \\ & + K3_i(P1 + M1) \end{aligned}$$

$$\begin{aligned} F_h \quad & (\text{even } h) = \\ & K1_i(P2\overline{A_{h-1}} + P1\overline{A_h} + M1B_h + M2B_{h-1}) \\ & + K2_i(P2 + P1 + M1 + M2) + \ K3_i(P1 + M1) \end{aligned}$$

For division by setting all the bits of register K to 1 (all $K1_i = 1$ and $K2_i = K3_i = 0$), $F[j]$ can be generated by the same expression as for square root. This corresponds to

$$F_h = P2\overline{A_{h-1}} + P1\overline{A_h} + M1B_h + M2B_{h-1}$$

for $h = 0, 1, \ldots, 55$ and with $d = A = B$. Note that for division, when q_{j+1} is positive (subtraction), the carry-in in the adder must be set to 1.

m_h	$d_\delta, \hat{S}[j]$							
	$\frac{8}{16}$	$\frac{9}{16}$	$\frac{10}{16}$	$\frac{11}{16}$	$\frac{12}{16}$	$\frac{13}{16}$	$\frac{14}{16}$	$\frac{15}{16}$
m_2	12	14	15	16	18	20	20	22
m_1	4	4	4	4	6	6	8	8
m_0	-4	-5	-6	-6	-6	-8	-8	-8
m_{-1}	-13	-15	-16	-17	-18	-20	-22	-23

Values in table are multiplied by 16

Table 3. Selection function for radix-4 combined division/square root.

2.4 Minimum delay implementation

The minimum delay implementation, also referred as *standard*, is the one obtained with the constraint of minimum delay. The post-layout critical path is 7.3 ns. The critical path for the combined unit is 5% longer than the critical path for the division only unit of [9]. This is mainly due to the more complicated FGEN block.

3 Energy Reduction

In this section we apply the techniques described in [9] to the standard implementation. Because of the different conversion algorithm required for square root, with respect to the standard implementation of [9], in the combined unit a low-power conversion and rounding unit is already in place in the standard implementation. The other techniques applied are: retiming of the recurrence, change of redundant representation for the residual $w[j]$, disabling of SZD, clock-gating of flip-flops in register K, and voltage scaling.

3.1 Retiming of the recurrence

Retiming is the transformation that consists in repositioning the registers in a sequential circuit without modifying its external behavior [6]. For the recurrence, the retiming is done by moving the selection function of Figure 1 from the first part of the cycle to the last part of the previous cycle (see Figure 2). A new 4-bit register is needed to store the result digit. The new register is initialized to $q_0 = 1$ for division and $s_0 = 1$ for square root.

This retiming causes a problem for the square root operation. The result digit s_{j+1} is converted in the next clock cycle and, as a consequence, in the first few iterations the value $\hat{S}[j]$ is not available for the selection function (Figure 3). However, because the digit-selection is done in the last part of the cycle and the conversion of the previous digit is a short delay operation, we can forward the value of the converted digit from the digit-converter to the the selection

s_{j+1}	$F[j]$		bit-string							
			27		$i+1$	i	$i-1$	$i-2$		0
0	0		00	...	00	00	00	00	...	00
1	$-A[j] - 2 \times 4^{-(j+2)}$		$\overline{aa}$	...	$\overline{aa}$	$\overline{a}1$	11	00	...	00
2	$-2A[j] - 8 \times 4^{-(j+2)}$		$\overline{aa}$	...	$\overline{aa}$	11	00	00	...	00
-1	$B[j] + 14 \times 4^{-(j+2)}$		bb	...	bb	$b1$	11	00	...	00
-2	$2B[j] + 24 \times 4^{-(j+2)}$		bb	...	bb	11	00	00	...	00
		K	1	...	1	1	0	0	...	0
		K1	1	...	1	0	0	0	...	0
		K2	0	...	0	1	0	0	...	0
		K3	0	...	0	0	1	0	...	0

Table 4. Generation of $F[j]$.

function and determine the correct value for $\hat{S}[j]$. By indicating with $s_{<1>}s_{<0>}$ the converted digit and with s_{SIGN} its sign, Table 5 shows the modifications in the selection function.

By retiming the recurrence we reduce the switching activity in the F generator and in the CSA. Moreover, we change the critical path that is now limited to the eight most-significant bits, which allows the rest of the bits in the recurrence to be redesigned for low-power.

3.2 Changing the redundant representation

By using a radix-4 carry-save representation with two sum bits and one carry bit for each digit in the recurrence we only need to store one carry bit for each digit, instead of three. This can be done for the 48 LSBs that, after the retiming, are not on the critical path.

Furthermore, after the retiming, the eight MSBs, assimilated in the adder inside the selection function block, can be stored in w_S eliminating another eight flip-flops in w_C.

3.3 Switching-off not active blocks

The sign-zero-detection block (SZD), which is only used in the rounding step, is switched off by forcing a constant logic value at its inputs during the recurrence steps.

3.4 Reduction in register K

Register K is used to generate F[j]. In division the register is initialized to 1 in each bit, and the configuration is not changed for the whole operation. In square root the register is initialized to 1 in the MSB and to 0 in the remaining bits. Every iteration the 1 is propagated to the next bit, for a total of two transitions per flip-flop (one to set the bit to 1, one to reset at the end of the operation). It is convenient to disable the clock for those flip-flops that do not need to be changed in a specific cycle. This is the same modification done for registers A and C in the convert-and-round unit implemented by gating the clock of the flip-flops not used [9]. The enabling function (for the i-th flip-flop) is

$$f_i = OP\ K_{i+1}\ \overline{K_i} \qquad\qquad i = 27, \ldots, 0$$

where $OP = 1$ for square root and $OP = 0$ for division. By implementing this technique the energy dissipation in K is reduced virtually to zero for division and to about one third for square root.

3.5 Dual voltage

The power dissipated in a cell depends on the square of the voltage supply (V_{DD}) so that significant amount of energy can be saved by reducing this voltage [2]. However, by lowering the voltage the delay increases, so that to maintain the performance this technique is applied only to cells not in the critical path. For example, in our library, designed to operate with $V_{DD} = 3.3\ V$, for $V_{DD} = 2.0\ V$ the delay is doubled, and for voltages below $1.7\ V$ the delay increases in excess. In the recurrence the 48 least-significant bits can be redesigned for low voltage. By using two voltages we only need to level-shift when going from the lower to the higher voltage [14]. As a consequence, the voltage-level shifters are not needed until a specific digit moves into the 8 MSBs.

Our library was designed to operate with $V_{DD} = 3.3\ V$.

In order to evaluate the possible lower voltage V_2 to be used in a dual voltage implementation we need to determine the time slack available for the LSBs. The time slack is the difference between the delay in the paths through the MSBs and LSBs, and it gives the amount of time available for the delay of gates whose voltage is scaled to V_2 (Table 6).

The delay of the least-significant portion depends on the type of CSA adder used, since the delay of the radix-4 CSA is larger than that of the radix-2 CSA.

The critical path in the retimed implementation is 7.3 ns. By implementing the LSBs of the recurrence with radix-2 CSAs, the delay in the LSBs is 3.8 ns, resulting in a time

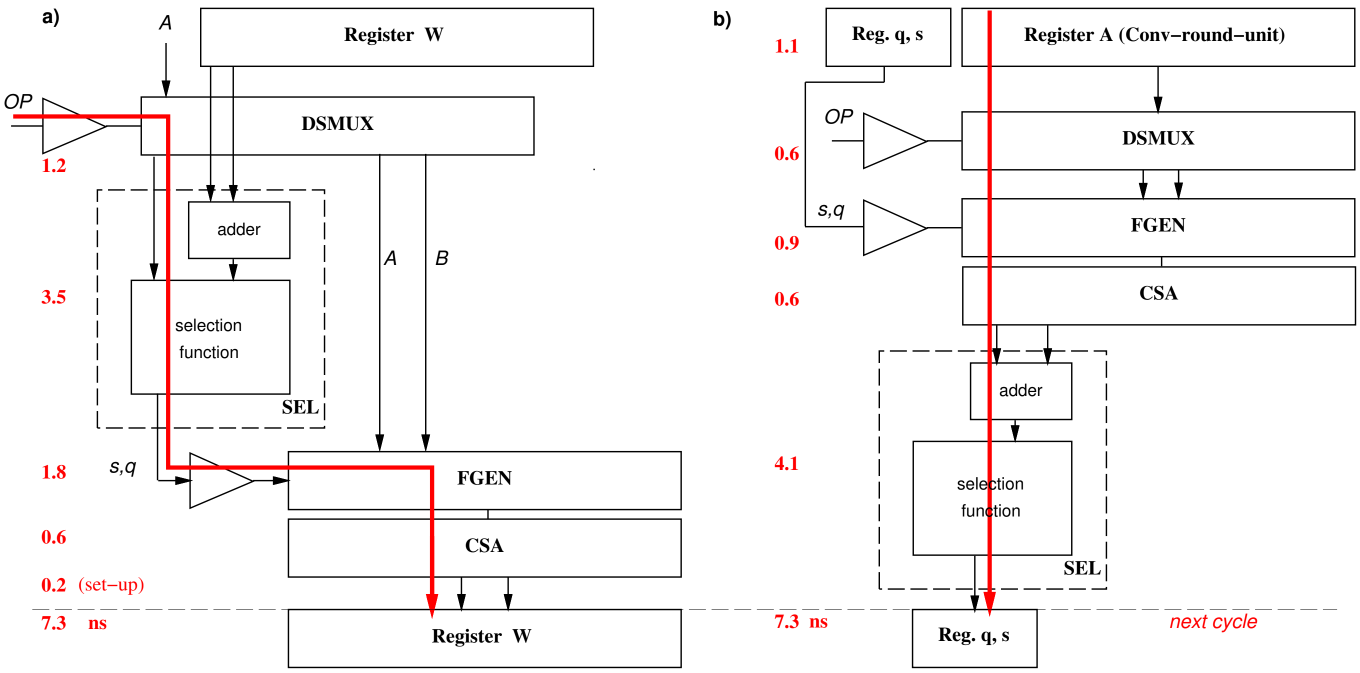

Figure 2. Retiming of the recurrence. a) before retiming. b) after retiming.

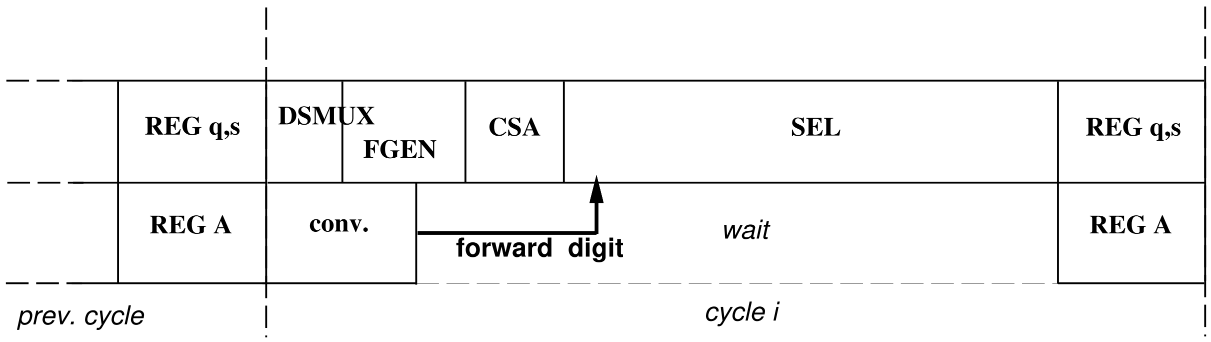

Figure 3. Digit forwarding.

j	$\hat{S}[j]$		
1,2	$A_{<0>}$	$A_{<-3>}$	$A_{<0>}$
3	$s_{<0>}$	0	0
4	$A_{<-2>} \oplus s_{SIGN}$	$s_{<1>}$	$s_{<0>}$
5	$A_{<-2>}\overline{(\overline{A_{<-3>}}\ \overline{A_{<-4>}} s_{SIGN})}$	$A_{<-3>}\overline{s_{SIGN}} + \overline{(A_{<-3>} \oplus A_{<-4>})} s_{SIGN}$	$A_{<-4>} \oplus s_{SIGN}$
others	$A_{<-2>}$	$A_{<-3>}$	$A_{<-4>}$

Table 5. Bits of A used in SEL (retimed).

MSBs:	DSMUX	FGEN	CSA	SEL	REG		
	0.6	0.9	0.6	4.1	1.1	=	7.3 ns
LSBs:	MUX	FGEN	CSA		REG		
	1.2	0.9	0.6		1.1	=	3.8 ns

Table 6. Paths for MSBs and LSBs in retimed recurrence.

slack of 3.5 ns. In this case $V_2 = 2.0\ V$ without affecting the latency of the unit. On the other hand, by opting for the use of radix-4 CSAs, the time slack is reduced to 2.6 ns and, consequently, V_2 can be lowered to $2.5\ V$.

Since the reduced voltage can be lower for radix-2 CSA, this might result in a reduction of the total energy. There is a tradeoff between the following:

- The voltage can be lower for radix-2 CSA
- There is a reduction in the number of flip-flops by using the radix-4 CSA.

Our estimation showed that the lowest energy is obtained by implementing radix-2 CSAs and $V_2 = 2.0\ V$ for the dual voltage implementation.

3.6 Low-power implementation

The implementation that consumes a reduced amount of energy is shown in Figure 4. Table 7 reports the average energy dissipation and area for the standard and low-power implementation. In the Table, entry *std* refers to the standard implementation, optimized for speed, and entry *l-p* is the low-power implementation with the same delay. It was not possible to implement dual voltage with our cell library so that entry *d-v* is an estimate of a possible implementation.

The energy dissipation for the square root operation is about 10% lower than for the division, on average. This is due to the fact that for square root in every iteration we add $F[j]$, which initially contains many zeros, to the residual $w[j]$.

Some of the low-power techniques used, such as the changed redundant representation, reduce the number of flip-flops in the registers and, consequently, the area.

4 Energy Comparison with Radix-4 Divider

In this section we compare the results obtained for the radix-4 combined division and square root with those obtained in [9] for a radix-4 divider.

Table 8 summarizes the results for the implementation *l-p* when performing division. Similar blocks are put on the same row. The combined unit dissipates 15% more than the divider, in the average. The largest differences are for the

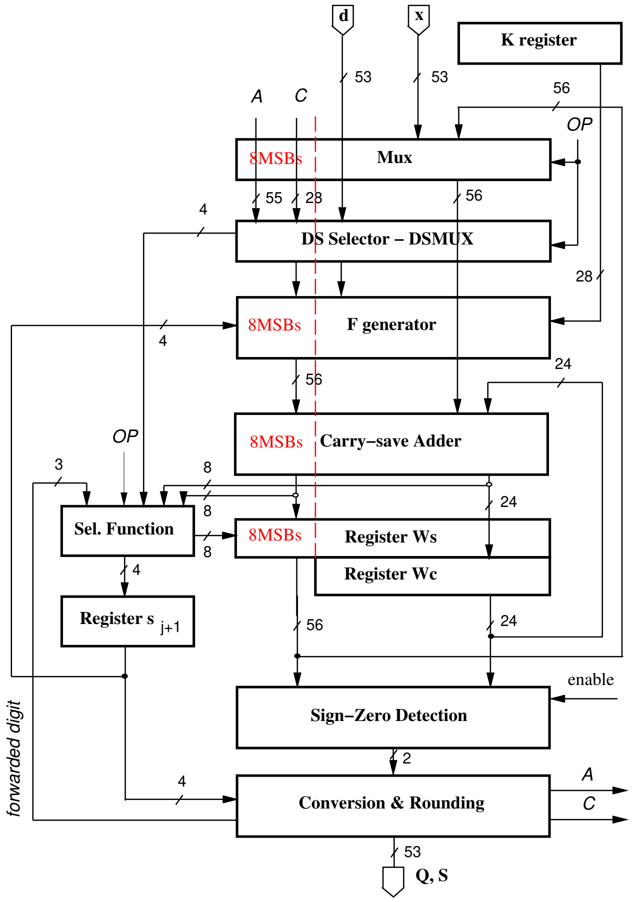

Figure 4. Low-power combined division and square root unit.

blocks FGEN and "mux". The implementation of FGEN is considerably more complicated than the corresponding unit in the divider and gates with large number of inputs (8-input NAND) have been used to keep the number of levels of logic (and the delay) low. As for the multiplexer, in the retimed implementation of the divider, it is moved out of the recurrence. However, in the divider an extra cycle is required because the first iteration is only used for initialization and no quotient-digit is produced. In the combined implementation in the first iteration we perform the subtraction $x - d$ using two inputs of the CSA and produce the first digit of the result.

5 Conclusions

We have implemented a low-power radix-4 combined division and square root unit and compared performance and energy consumption with a radix-4 divider. We were able to apply the techniques presented in [9] to the combined

blocks	division			sqr. root		
	std	l-p	d-v	std	l-p	d-v
	nJ	nJ	nJ	nJ	nJ	nJ
control	0.9	0.9	0.9	0.9	0.9	0.9
clk tree	0.8	0.8	0.8	0.8	0.8	0.8
mux	1.9	1.9	0.9	1.7	1.7	0.8
DSMUX	0.1	0.1	0.1	0.3	0.3	0.3
FGEN	7.3	4.9	2.4	5.8	4.3	2.0
CSA	8.8	5.2	3.8	4.7	3.9	2.3
sel. func.	1.5	2.0	2.0	1.2	1.8	1.8
register Ws	6.7	6.4	*3.7	6.3	6.1	*3.6
register Wc	6.7	2.7	*3.1	5.1	2.2	*2.5
register q	-	0.3	0.3	-	0.3	0.3
register K	1.3	0.0	0.0	1.6	0.5	0.2
SZD	5.8	0.6	0.6	4.6	0.6	0.6
C&R unit	3.7	3.7	*1.4	3.7	3.7	*1.4
E_{op} $[nJ]$	46.0	29.5	20.0	37.0	27.0	17.5
Ratio to std	1.00	0.65	0.45	1.00	0.75	0.50
E_{sqrt}/E_{div}	-	-	-	0.80	0.90	0.90
Area $[mm^2]$	1.9	1.8	-	1.9	1.8	-

Values marked * include level shifters.

Table 7. Summary of reductions for division and square root operations.

blocks	divider	combined
control	1.1	0.9
clk tree	0.9	0.8
mux	0.3	1.9
DSMUX	-	0.1
FGEN	2.8	4.9
CSA	4.8	5.2
sel. func.	1.6	2.0
register Ws	6.4	6.3
register Wc	3.5	2.7
register q	0.3	0.3
register K	-	0.0
SZD	0.6	0.6
C&R unit	3.9	3.7
E_{div} $[nJ]$	26.0	29.5
Ratio	1.00	1.15
Area $[mm^2]$	1.2	1.8
Ratio	1.0	1.5

Table 8. Comparison radix-4 divider vs. combined unit.

unit obtaining a reduction of the energy of about 35% without increasing the latency. An additional reduction of 20% would be possible by using dual voltage. Furthermore, results show that the cycle time of the combined unit is 5% longer than the radix-4 divider and the energy dissipation is about 15% larger for the combined unit.

Acknowledgments

This work was partially supported by NSF grant MIP 9314172 and by State of California and Sun Microsystems, through UC MICRO 97-084.

References

[1] R. Burch, F. Najm, P. Yang, and T. Trick. A Monte Carlo approach for power estimation. *IEEE Transactions on VLSI Systems*, pages 63–71, Mar. 1993.

[2] A. P. Chandrakasan and R. W. Brodersen. *Low Power Digital CMOS Design*. Kluwer Academic Publishers, 1995.

[3] Compass Design Automation. *Passport - 0.6-Micron, 3-Volt, High-Performance Standard Cell Library*. Compass Design Automation, Inc., 1994.

[4] M. Ercegovac and T. Lang. *Division and Square Root: Digit-Recurrence Algorithms and Implementations*. Kluwer Academic Publisher, 1994.

[5] J. Fandrianto. Algorithm for high-speed shared radix-8 division and radix-8 square root. *Proc. of 9th Symposium on Computer Arithmetic*, pages 68–75, Sept. 1989.

[6] J. Monteiro, S. Devadas, and A. Ghosh. Retiming sequential circuits for low power. *Proc. of 1993 International Conference on Computer-Aided Design (ICCAD)*, pages 398–402, Nov. 1993.

[7] A. Nannarelli. PET: Power Evaluation Tool. *http://www.eng.uci.edu/numlab/PET/*, Aug. 1996.

[8] A. Nannarelli and T. Lang. Low-Power Radix-8 Divider. *Proc. of the International Conference on Computer Design*, pages 420–426, Oct. 1998.

[9] A. Nannarelli and T. Lang. Low-Power Divider. *IEEE Transactions on Computers*, pages 4–17, Jan. 1999.

[10] A. Nannarelli and T. Lang. Low-Power Division: Comparison among implementations of radix 4, 8 and 16. *Proc. of the 14th Symposium on Computer Arithmetic, Adelaide (AUS)*, pages 60–67, April 1999.

[11] W. Nebel and J. Mermet editors. *Low Power Design in Deep Submicron Electronics*. Kluwer Academic Publishers, 1997.

[12] A. Prabhu and G. Zyner. 167 MHz radix-8 divide and square root using overlapped radix-2 stages. *Proc. of 12th Symposium on Computer Arithmetic*, pages 155–162, July 1995.

[13] J. M. Rabaey, M. Pedram, et al. *Low Power Design Methodologies*. Kluwer Academic Publishers, 1996.

[14] K. Usami and M. Horowitz. Clustered voltage scaling technique for low-power design. *Proc. of International Symposium on Low Power Design*, pages 3–8, Apr. 1995.