2011 20th IEEE Symposium on Computer Arithmetic

Radix-16 Combined Division and Square Root Unit

Alberto Nannarelli
Dept. Informatics and Mathematical Modelling
Technical University of Denmark
Kongens Lyngby, Denmark
Email: an@imm.dtu.dk

***Abstract*—Division and square root, based on the digit-recurrence algorithm, can be implemented in a combined unit. Several implementations of combined division/square root units have been presented mostly for radices 2 and 4. Here, we present a combined radix-16 unit obtained by overlapping two radix-4 result digit selection functions, as it is normally done for division only units. The latency of the unit is reduced by retiming and low power methods are applied as well. The proposed unit is compared to a radix-4 combined division/square root unit, and to a radix-16 unit, obtained by cascading two radix-4 stages, which is similar to the one implemented in a state-of-the-art processor.**

***Keywords*-Floating-point, division, square root, digit-recurrence.**

I. Introduction

Although division and square root are much less frequent than addition and multiplication, most of multicore and embedded system processors have the two operations implemented in hardware.

In [1] it is reported that the consumers of division results are multiplications and additions, and therefore, a not efficient implementation of division will degrade overall performance.

The division and square root algorithms can be implemented for floating-point binary64 (formerly double-precision) by two classes of iterative algorithms.

The *digit-recurrence algorithms* require a number of iterations depending on the power-of-two radix chosen [2]. The main advantages are that the necessary hardware (and consequently, the power dissipated) is small and the rounding is easy as the remainder is computed as part of the algorithm.

The *multiplicative algorithms* (e.g. Newton-Raphson) are based on the computation of the reciprocal (or inverse square root) and then the result of the operation is obtained by multiplication [3]. With respect to the digit-recurrence algorithm, this requires less iterations (convergence is quadratic), but it requires a multiplier (larger than hardware necessary for digit-recurrence) that normally is not the one used for multiplication to not lose performance. Moreover, the rounding is not straightforward as for digit-recurrence.

Division (square root) is implemented by the radix-4 digit-recurrence algorithm, with some variants, in Intel Pentium CPUs [4], in ARM processors [5], in IBM FPUs [6] and by a radix-16 unit in the Intel Core2 processors [4].

On the other hand, division (square root) by iterative multiplication it has been chosen in AMD processors [7], NVIDIA GPUs [8], and in Intel Itanium CPUs [9].

In this paper, we focus on the digit-recurrence algorithm by combining radix-16 division and square root (div/sqrt in the following) in a single unit. This combination has been done in the past for radix-8 in [10], for radix-4 in [11] and recently in [5] and [12]. The Intel Core2 (code named Penryn) div/sqrt unit is implementing radix-16 by cascading two radix-4 stages similar to those of [5].

Here we propose the radix-16 combination of div/sqrt by overlapping, instead of cascading, two radix-4 stages [2]. In this way, we obtain a reduced operation latency with relatively small area and power dissipation overhead.

The results of the implementation are compared to a unit similar to the one of the Penryn, to a radix-4 combined unit, and to a radix-16 division only unit to see the overhead of square root combination.

II. Algorithm

Because of the similarities in the digit-recurrence algorithm, division and square root can be effectively implemented in the same unit. By defining the two operations as

$$\begin{array}{ll} \text{division:} & q = \frac{x}{d} + rem \\ \text{square root:} & s = \sqrt{x} \end{array}$$

where x is the dividend/radicand, d the divisor, q/s is the result (quotient/square root), and rem is the remainder, the generic radix-r division and square root, described in detail in [2], is implemented by the residual recurrence

$$w[j+1] = rw[j] + F[j] \qquad j = 0, 1 \ldots m \qquad (1)$$

in which $w[j]$ is the residual at iteration j (initialized by x),

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

The result digit (q_{j+1} for division and s_{j+1} for square root) are determined, at each iteration, by a selection function

$$\begin{array}{ll} q_{j+1} = SEL(\widehat{d}, \widehat{y}) & \text{(division)} \\ s_{j+1} = SEL(\widehat{S}[j], \widehat{y}) & \text{(square root)} \end{array}$$

where $\widehat{d}$ and $\widehat{S}[j]$ are respectively d and $S[j]$ truncated after the δ-th fractional bit, and $\widehat{y}$ is an estimate of $rw[j]$.

1063-6889/11 $26.00 © 2011 IEEE
DOI 10.1109/ARITH.2011.30

A. Radix-16 Division Recurrence

For higher radices, to avoid using multipliers to form $F[j]$ the quotient/result digit is usually decomposed in two parts. For example, for radix-16 division, a possible decomposition is

$$q_j = 4q_{Hj} + q_{Lj} \tag{3}$$

with $q_{Hj} \in \{-2,-1,0,1,2\}$, $q_{Lj} \in \{-2,-1,0,1,2\}$ and a corresponding digit set $q_j \in [-10,10]$.

With this decomposition, the retimed[1] radix-16 division recurrence is

$$\begin{array}{rcl} v[j] & = & 16w[j-1] - q_{Hj}(4d) \\ w[j] & = & v[j] - q_{Lj}d \end{array} \tag{4}$$

with $w[0] = x$ (eventually shifted to ensure convergence). and the selection function is also split into two parts

$$\begin{array}{rcl} q_{Hj} & = & SEL_H(\widehat{rw}, d_\delta) \\ q_{Lj} & = & SEL_L(\widehat{v}, d_\delta) \end{array} \tag{5}$$

where $y = \widehat{rw}$ and $\widehat{v}$ are truncated to a few MSBs (10 for division) and d_δ are the 3 bits of d ($0.5 \le d < 1.0$) of weight $2^{-2}, 2^{-3}, 2^{-4}$. However, because the two radix-4 stages are overlapped, q_{Lj} is computed speculatively for all the possible outcomes of q_{Hj} and the right one is selected once q_{Hj} is determined. That is

$$q_{Hj} = SEL_H(y, d_\delta)$$

$$q_{Lj} = \begin{cases} SEL_L(y + \widehat{2d}, d_\delta) & \text{if } q_{Hj} = -2 \\ SEL_L(y + \widehat{d}, d_\delta) & \text{if } q_{Hj} = -1 \\ SEL_L(y, d_\delta) & \text{if } q_{Hj} = 0 \\ SEL_L(y - \widehat{d}, d_\delta) & \text{if } q_{Hj} = +1 \\ SEL_L(y - \widehat{2d}, d_\delta) & \text{if } q_{Hj} = +2 \end{cases} \tag{6}$$

The residual(s) $w[j]$ ($v[j]$) are usually stored in carry-save representation (w_S, w_C, ...) to reduce the iteration time.

The recurrence of (4) is illustrated in Figure 1. The iteration is delimited by a dashed horizontal line in the figure, corresponding to the position of the registers in the hardware implementation.

The division unit is completed by a on-the-fly convert-and-round unit [2] which converts the radix-16 quotient-digit from the signed-digit representation to unsigned binary representation of the quotient q, and performs the rounding

B. Radix-16 Square Root Recurrence

The retimed radix-16 square root recurrence

$$w[j] = 16w[j-1] - s_j\left(S[j-1] + \frac{1}{2}16^{-j}s_j\right)$$

by introducing

$$P[j-1] = S[j-1] + \frac{1}{2}16^{-j}s_j$$

[1]In the hardware implementation the quotient-digit is computed at the end of the cycle (iteration) [13].

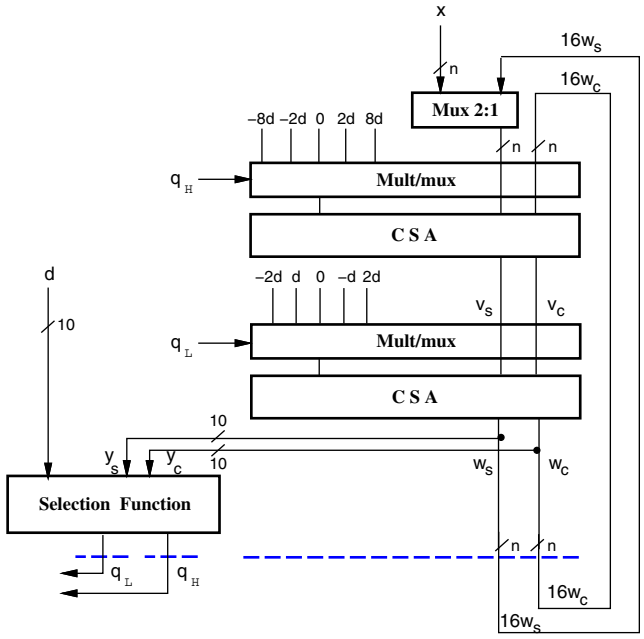

Figure 1. Radix-16 division recurrence.

can be rewritten as

$$w[j] = 16w[j-1] - s_jP[j-1]$$

which can be decomposed, by splitting the result digit $s_j = 4s_{Hj} + s_{Lj}$, into

$$\begin{array}{rcl} v[j] & = & 16w[j-1] - s_{Hj}(4P[j-1]) \\ w[j] & = & v[j] - s_{Lj}P[j-1] \end{array} \tag{7}$$

The selection function is:

$$\begin{array}{rcl} q_{Hj} & = & SEL_H(\widehat{rw}, \widehat{S}[j-1]) \\ q_{Lj} & = & SEL_L(\widehat{v}, \widehat{S}[j-1]) \end{array} \tag{8}$$

However, differently from division, the result-digit s_j and the residual $w[j]$ depend on the partial result $S[j]$ which is updated at each iteration by the on-the-fly conversion. This issue is addressed in Section III.

C. Radix-16 Combined Division/Square Root Recurrence

The radix-16 recurrences (4) and (7) can be combined as follows

$$\begin{array}{rcl} v[j] & = & 16w[j-1] + 4F_H[j-1] \\ w[j] & = & v[j] + F_L[j-1] \end{array} \tag{9}$$

where

$$\begin{array}{rcll} F_H[j-1] & = & \begin{cases} -q_{Hj}d & \text{(division)} \\ -s_{Hj}P[j-1] & \text{(square root)} \end{cases} & \\ F_L[j-1] & = & \begin{cases} -q_{Lj}d & \text{(division)} \\ -s_{Lj}P[j-1] & \text{square root)} \end{cases} & \end{array} \tag{10}$$

In the following, in the combined unit, we use q_j to refer to both the quotient and result digit. Moreover, in the

selection function we need to select among d and $S[j-1]$ for the two operations.

D. Selection Function

With the decomposition of (3) each radix-4 quotient-digit (result-digit) has a redundancy $\rho = \frac{2}{3}$ and the selection function of [2] for the radix-4 combined division and square root can be used.

The bounds for division are

$$\begin{aligned} U_k[j] &= d(k+\rho) \\ L_k[j] &= d(k-\rho) \end{aligned}$$

and for square root

$$\begin{aligned} U_k[j] &= S[j](k+\rho) + \tfrac{1}{2}(k+\rho)^2 4^{-(j+1)} \\ L_k[j] &= S[j](k-\rho) + \tfrac{1}{2}(k-\rho)^2 4^{-(j+1)} \end{aligned}$$

The selection constants for the combined radix-4 from [2] can be used.

The radix-4 result-digit q_{j+1} is determined by performing a comparison [14] of the truncated residual y (carry-save) with the four values (m_k) representing the boundaries to select the digit for the given d/$S[j]$. That is,

$$\begin{array}{rcccccl} & y & \geq & m_2 & \rightarrow & q_{j+1} = 2 & \\ m_1 & \leq & y & < & m_2 & \rightarrow & q_{j+1} = 1 \\ m_0 & \leq & y & < & m_1 & \rightarrow & q_{j+1} = 0 \\ m_{-1} & \leq & y & < & m_0 & \rightarrow & q_{j+1} = -1 \\ & & y & < & m_{-1} & \rightarrow & q_{j+1} = -2 \end{array} \qquad (11)$$

With respect to the selection constants of [2], when implementing selection by comparison, it is convenient to have symmetrical selection constants m_k's with respect to the positive/negative values of y. In this way, only one constant is stored per $\hat{d}/\widehat{S[j]}$ interval, and the other is obtained by a two's complement

$$m_{-1} = -m_2 \quad \text{and} \quad m_0 = -m_1$$

The modified constants m_k's are listed in Table I.

In the first row of Table I, $m_{-1} = -m_2 - 1 = -13$ because otherwise the algorithm is not converging. However, -13 is the one's complement of 12 and this exception can be easily overcome in the hardware implementation[2].

III. Architecture

In this section the architecture implementing the combined div/sqrt for binary64 (double-precision) is presented.

We assume the operands x (dividend/radicand) and d (divisor) normalized in $[0.5, 1)$. To ensure convergence ($|w[j]| \leq \rho d$), $w[0]$ is initialized to $x/4$ for division. For square root, the result is initialized to $S[0] = 1.0$ and $w[0] = x - 1.0$. Moreover, if the biased exponent of x is odd, the recurrence is initialized as $w[0] = \frac{x}{2} - 1.0$.

[2]Normally, two's complement is implemented in hardware by bit inversion (one's complement) and adding a '1' in the next available adder.

$d_\delta/\widehat{S[j]}$	m_2	m_1	m_0	m_{-1}
0.1000	12	4	-4	**-13**
0.1001	14	4	-4	-14
0.1010	16	6	-6	-16
0.1011	17	6	-6	-17
0.1100	18	6	-6	-18
0.1101	20	8	-8	-20
0.1110	22	8	-8	-22
0.1111	23	8	-8	-23

Values on m_k's are multiplied by 16.

Table I
SELECTION CONSTANTS FOR RADIX-16 DIVISION

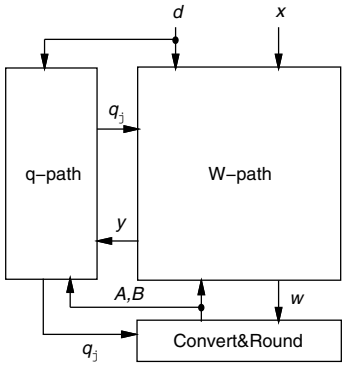

Figure 2. Structure of radix-16 combined div/sqrt unit.

To simplify the description of the architecture implementing the recurrence of (9) (10) and the selection function of (5) (8), we split the recurrence into two paths:

- ▷ **W-path** is the wide path (58 bits for binary64) implementing (9) and (10).
- ▷ **q-path** is the narrow path (max. 12 bits) implementing the overlapped selection function and the necessary speculation.

The on-the-fly conversion algorithm [2] produces the quotient of division and $S[j]$ to be used in the iterations. To avoid the use of a carry-propagate adder, two variables A and B are required. They are updated, in every iteration, as follows:

$$A[j] = S[j] \qquad \text{and} \qquad B[j] = S[j] - 16^{-j}$$

At the end of the iterations, $A[m]$ contains the rounded quotient of the division, or the square root.

The top-level unit (the computation of exponent and sign is straightforward and not covered in the paper) is sketched in Figure 2 and the detail of the implementation is given next. The combined div/sqrt unit is completed by a controller (not in the figure), containing a sequencer K keeping track of the iterations used for sqrt, plus some other signals to set multiplexers and enable registers. We assume that a signal (1-bit) OP sets the operation to div or sqrt.

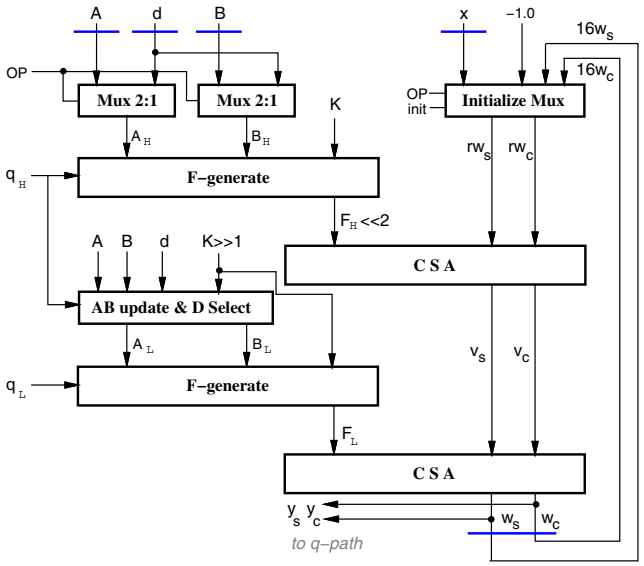

Figure 3. Radix-16 W-path.

A. Block W-path

The block W-path, shown in Figure 3, implements the expressions (w and v are carry-save):

$$\begin{aligned} v[j] &= 16w[j-1] + 4F_H[j-1] \\ w[j] &= v[j] + F_L[j-1] \end{aligned}$$

In the figure, a thicker (blue) line indicates the position of the registers. The datapath is 58 bits wide. The operations performed in the blocks of Figure 3 are described below.

- ▷ **Initialize Mux** is used to initialize the recurrence as follows:

$$\begin{aligned} OP = 0 \text{ (div)} : \quad & rw_s = x/4 \quad rw_c = 0 \\ OP = 1 \text{ (sqrt} : \quad & rw_s = x \quad\;\; rw_c = -1 \end{aligned}$$

  The multiplexer takes care of shifting x two (OP=0) or one (OP=1 and odd exponent) positions to the right.
- ▷ The two multiplexers at the top-left of Figure 3 select the inputs to block F-generate (FGEN) according to the operation:
  - division: divisor d;
  - square root: partial result $A[j]$ (and its diminished-by-1 value $B[j]$).
- ▷ **F-generate** (FGEN) generates the signals $F_H[j]$ and $F_L[j]$ as described in expression (10). For division, it is straightforward as it works as a multiplexer selecting $-2d, -d, 0, d, 2d$. For square root, the generation of F is more complicated as the bits of A and B are loaded in position (radix-4) according the sequencer K. More detail on FGEN operation can be found in [2] and [11].
- ▷ The block **AB update & D select** is necessary to update A and B according to the value of q_H. Because the conversion of A is delayed one cycle in the retimed implementation, it is necessary to perform the radix-4 update with q_H to avoid errors in the residual. The update is done by replacing B with A if $q_H > 1$, or vice-versa if $q_H < 1$, and by appending the converted q_H digit in the position indicated by the sequencer K. When division, d is selected.

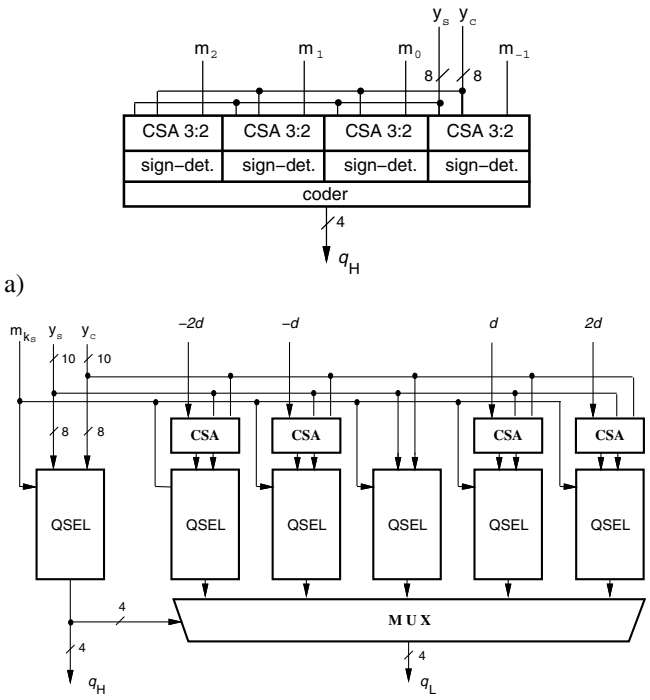

Figure 4. a) Implementation of QSEL. b) Overlapped radix-16 quotient-digit selection (division).

B. Block q-path

Before describing the q-path, we recall how selection-by-comparison [14] and radix-16 by overlapping two radix-4 stages are implemented in hardware for division.

The selection of (11) can be implemented with a unit (QSEL) similar to that depicted in Figure 4.a where four 8-bit comparators (sign-det.) are used to detect in which range y lies. The coder then encodes q_H in 1-out-of-4 code which is suitable to drive multiplexers.

In parallel, all five possible outcomes of q_L are computed speculatively (Figure 4.b), and then one of them is selected once q_H is determined. Therefore, the computation of q_L is overlapped to that of q_H, and q_L is obtained with a small additional delay.

Figure 5 (q-path) shows the additions to Figure 4.b necessary to implement square root. There are two main issues:

1) $\widehat{S}[j]$ (or $\widehat{A}$ in the figure) is not constant, as it is d in division, through the iterations.

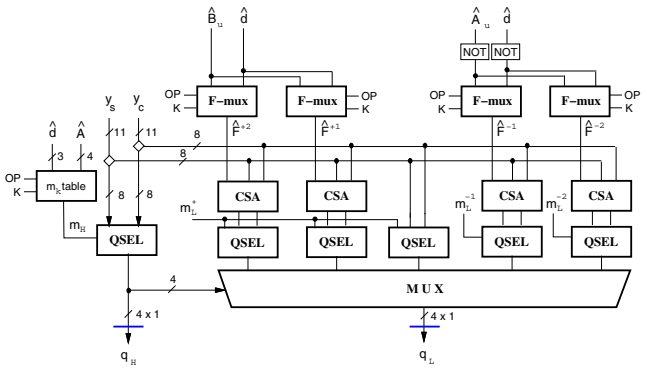

Figure 5. Radix-16 q-path.

bits of $\widehat{A}$			comment
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}.

Table II
BITS OF $\widehat{A}$ USED TO ACCESS m_k TABLE.

2) The computation of the speculative $\widehat{F}$'s is also depending on the iterations.

The main problem is that, although the few bits necessary for the q-path (12 bits for $\widehat{A}$ and $\widehat{B}$) can be determined in three radix-16 iterations, it might happen that if there is a sequence of $q_j = 0$, the update of A can occur at later iterations. Therefore, to be able to handle these cases, a 12 bit replica of the on-the-fly conversion is necessary to update $\widehat{A}$ (and $\widehat{B}$).

The solution for the changing $\widehat{A}$ has already been addressed in [2] for q_H by implementing the selection of the bits of $\widehat{A}$ according to Table II.

However, for the speculative computation of q_L, we might need to change the m_k's constants on-the-fly. This is illustrated with the following example.

For square root, the first non-zero radix-4 digit is either -1 or -2. If the first non-zero radix-4 digit is -2, then the following non-zero digit is positive. Consequently,

$$q_H[1] = \{0, -1, -2\}$$

and to select the constants of $q_L[1]$ we might have to update $\widehat{A}$ to $\widehat{A}_u$ (and $\widehat{B}_u$) as follows (see Table I and Table II):

$q_H[1]$		$\widehat{A}_u[1]$			m_k
0	$\rightarrow$	1.0000	$\rightarrow$	Table(111) :	$\{23, 8, -8, -23\}$
-1	$\rightarrow$	0.1100	$\rightarrow$	Table(100) :	$\{18, 6, -6, -18\}$
-2	$\rightarrow$	0.1000	$\rightarrow$	Table(000) :	$\{12, 4, -4, -13\}$

This situation can occur at each iteration, if there is an initial sequence of zero digits. By introducing a state ($zero = 1$) which is reset when the first non-zero digit appears, and by indicating with $m_k(a)$ the set of constants for the specific $\widehat{A}_u$, the example below can be generalized by

$$\text{if } zero = 1 \begin{cases} q_L^1 & : \ d.c. \\ q_L^2 & : \ d.c. \\ q_L^0 & : \ m_k(7) \\ q_L^{-1} & : \ m_k(7) \\ q_L^{-2} & : \ m_k(7) \end{cases} \quad \text{else} \begin{cases} q_L^1 & : \ m_k(\widehat{A}_u) \\ q_L^2 & : \ m_k(\widehat{A}_u) \\ q_L^0 & : \ m_k(\widehat{A}_u) \\ q_L^{-1} & : \ m_k(\widehat{B}_u) \\ q_L^{-2} & : \ m_k(\widehat{B}_u) \end{cases}$$

From the example above, it is also clear that three separate tables are necessary to fetch the m_k's for the q_L selection:

1) set m_L^+ when $q_H \geq 0$;
2) set m_L^{-1} when $q_H = -1$;
3) set m_L^{-2} when $q_H = -2$;

When the operation is division, these three tables are bypassed (mux) and the same set of m_k's is used throughout the whole division (see Figure 6).

Finally, the blocks **F-mux** select the speculative values to be added to y for the digit values $q_H = \{-2, -1, 1, 2\}$. This is realized with multiplexers selecting according to the operation (div/sqrt) and the iterations as explained in Figure 7.

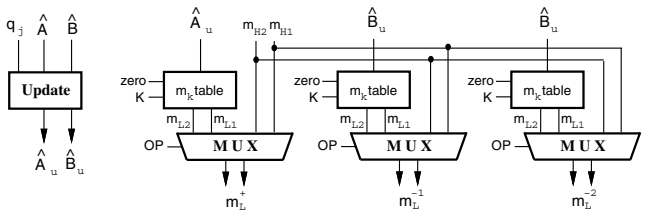

Figure 6. m_k's tables for speculative computation.

iter.		
1	$\widehat{F}^{+2}$ =	aa110000000
	$\widehat{F}^{+2}$ =	aaa11100000
	$\widehat{F}^{+2}$ =	bbb11100000
	$\widehat{F}^{+2}$ =	bb110000000
2	$\widehat{F}^{+2}$ =	aaaaaa11000
	$\widehat{F}^{+2}$ =	aaaaaaa1110
	$\widehat{F}^{+2}$ =	bbbbbbb1110
	$\widehat{F}^{+2}$ =	bbbbbb11000
3	$\widehat{F}^{+2}$ =	aaaaaaaaaa1
	$\widehat{F}^{+2}$ =	aaaaaaaaaaa
	$\widehat{F}^{+2}$ =	bbbbbbbbbbb
	$\widehat{F}^{+2}$ =	bbbbbbbbbb1
>3	$\widehat{F}^{+2}$ =	aaaaaaaaaaa
	$\widehat{F}^{+2}$ =	aaaaaaaaaaa
	$\widehat{F}^{+2}$ =	bbbbbbbbbbb
	$\widehat{F}^{+2}$ =	bbbbbbbbbbb
	...	

Figure 7. Operation in F-mux for square root (bits a are inverted).

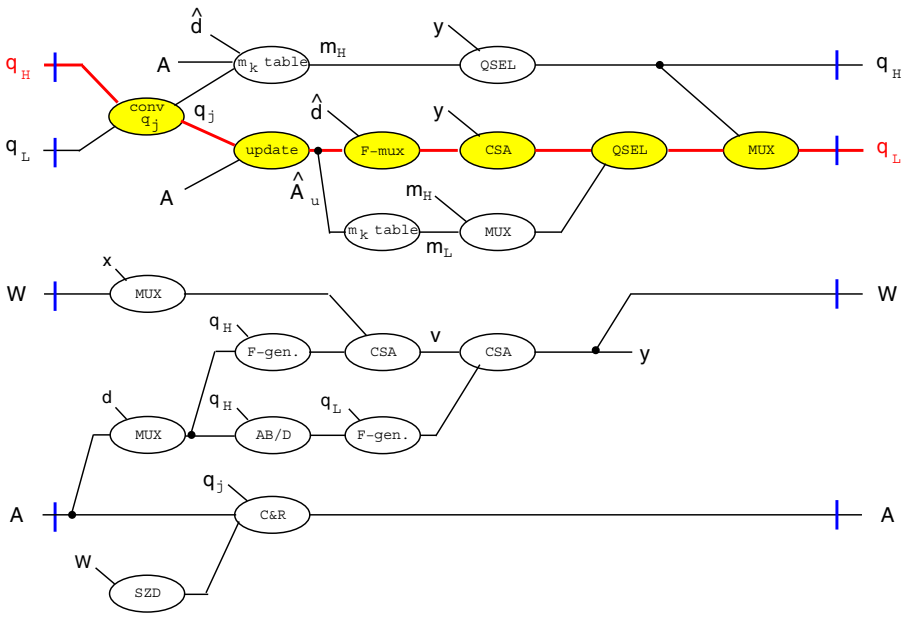

Figure 8. Timing paths (left to right) and critical path (highlighted).

C. Timing Diagram and Latency

Figure 8 shows the timings paths of the radix-16 combined div/sqrt unit. The numerical values obtained by implementing the unit (see detail in Section V) for these paths are reported in Table III. In the table, the delay in the paths from the starting node (register) to the arriving node are presented. The critical path (highlighted in Figure 8) is 1.083 ns and it is between register q_H and q_L.

The unit can be retimed by moving the position of the registers along the paths, or by moving the clock edges by controlled skew, to try and reduce the delay of the critical path.

However, no retiming will improve the delays in the paths between the same register (diagonal in Table III). By looking at the table, the longest delay on the diagonal is 1.082 ns for $q_L \rightarrow q_L$. This is basically the same value as the critical path (the FO4 delay in the library used in 45 ps) making any further retiming optimization useless.

On the other hand, due to the retimed implementation of the recurrence, the slack across register w (actually two registers for carry-save representation) is quite significant (1083-606=447 ps) and it can be utilized to design a large portion of the W-path (48 least-significant bits) for low power by trading off delay and reduced power.

As for the number of cycles, for both operations, 16 cycles, including initialization and rounding, are needed.

from / to	q_H	q_L	w	A
q_H	1.032	1.083	0.552	0.472
q_L	1.031	1.082	0.548	0.472
w	1.013	1.066	0.606	0.817
A	1.032	1.032	0.534	-

Table III
TIMING PATHS $[ns]$.

IV. OTHER ARCHITECTURES FOR COMBINED DIV/SQRT

The div/sqrt unit implemented in the Intel Core2 (Penryn) family is sketched in Figure 9 [4]. It implements IEEE binary32/binary64 compliant division and square root, plus extended precision (64 bits) and integer division. The unit consists of three main parts: the pre-processing stage necessary to normalize integer operands to ensure convergence; the recurrence stage; and the post-processing stage where the rounding is performed.

The recurrence is composed of two cascaded radix-4 stages synchronized by a two-phase clock to form a radix-16 stage (4 bits of quotient computed) over a whole clock cycle.

Each radix-4 stage is realized with a scheme similar to that of [5] shown in Figure 10. For each radix-4 stage the result-digit is determined by performing a comparison of the truncated residual $y = \widehat{4w[j]}$ (carry-save) with the four values (m_k) representing the boundaries to select the digit for the given d ($\widehat{S}[j]$). In parallel, all partial remainders $w^k[j+1]$ are computed speculatively, and then one of them is selected once q_{j+1} is determined. The carry-save output of the radix-4 stage is then hardwired to shift-left 2 bits (multiplication by 4).

This scheme was selected because of the reduced logical depth. However, the speculation on the whole w-word (68 bits for the Core2 format), is quite expensive in terms of area and power dissipation.

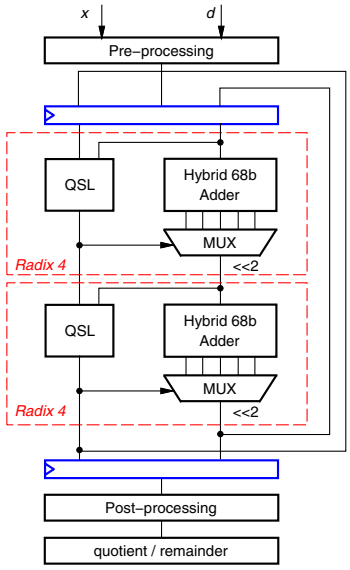

Figure 9. Architecture of Penryn div/sqrt unit.

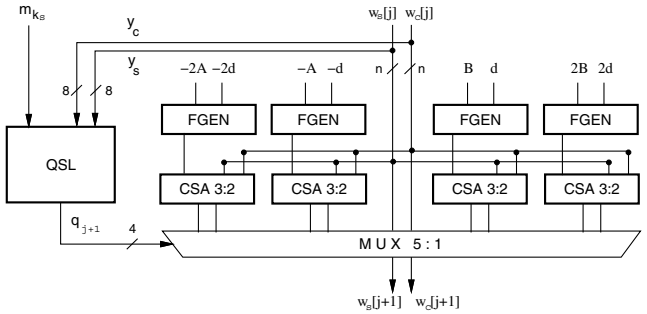

Figure 10. Single radix-4 div/sqrt stage.

V. Implementation and Comparison

To evaluate the performance in terms of delay, area and power dissipation of the proposed unit, we implemented for binary64 the following alternatives.

- **c16over** is the radix-16 combined div/sqrt unit with overlapped result-digit selection presented here.
- **cPenryn** is the combined unit similar to that of Figure 9 modified to handle binary64 only. That is, the recurrence is composed by two cascaded radix-4 stages, plus the same initialization and convert-and-round unit as the **c16over**.
- **d16over** is the radix-16 division only unit with overlapped selection function. This is considered our reference design.
- **dPenryn** is the unit of Figure 9 supporting division only.
- **r4comb** is a radix-4 combined div/sqrt unit similar to the one presented in [5].

The units are synthesized by Synopsys's Design Compiler with a 90 nm CMOS standard cell library providing two types of cells for each logic gate: a standard-V_t [3] cell and a high-V_t cell to obtain reduced power dissipation to be used in paths with a slack. For comparison purposes, the FO4 inverter delay is $45\ ps$ and the area of the NAND2 gate is $4.4\ \mu m^2$ in this library.

The units have been modeled in VHDL at RTL-level and synthesized to obtain the maximum speed. Because in our design flow we do not use two-phase clocks, for the ***Penryn** implementations we cascaded the two radix-4 stages of Figure 9 into a single clock cycle.

We used Synopsys's Power Compiler to estimate the power based on randomly generated input vectors by using an instruction mix (division/square root) in a ratio 5:1 to keep into account the different frequencies of the two operations. For all units, the average power dissipation was estimated at a normalized frequency of 100 MHz.

We also estimated the average energy to complete a full operation

$$E_{op} = P_{ave} \times \text{latency}\ .$$

The synthesis results are summarized in Table IV.

The fastest unit in terms of clock cycle is clearly the radix-4 combined. In our implementation the delay is 19 FO4 which is quite close to the value reported in [5] (17 FO4). However, the extra number of cycles makes its latency and E_{op} larger than those of **c16over**.

The overhead of combining div/sqrt is about 10% for **c16over** over **d16over** for delay/latency and higher for power and area (24% and 64% respectively).

For the **cPenryn** the overhead over **dPenryn** in delay is negligible, but the impact on area is quite large. As for the power dissipation, it is likely that for the ***Penryn** it is overestimated because the original clocking scheme (two-phase) prevents transitions generated in the upper radix-4 stage to propagate in the lower one. However, the speculation in the wide (w) path makes the unit dissipating more power than the ***16over** approach where the speculation is in the narrow path.

When comparing **c16over** and **cPenryn**, Table IV shows that the proposed unit is superior in all the metrics.

In summary, the results show that the proposed radix-16 combined div/sqrt unit has a shorter latency when compared to a unit similar to the one of Intel Core2 and consumes significantly less power. On the other hand, the overhead introduced by the square root is 11% for latency and 24% for power dissipation with respect to a radix-16 division only unit.

VI. Conclusions

In this work, we developed a radix-16 combined division and square root unit based on the digit-recurrence algorithm

[3] V_t is devices' threshold voltage.

Unit	Crit. Path $[ns]$	ratio	Cycles	Latency $[ns]$	Area NAND2	ratio	P_{ave} $[mW]$	ratio	E_{op} $[pJ]$
c16over	1.083	1.11	16	17.33	59,685	1.64	3.00	1.24	480
cPenryn	1.328	1.37	16	21.25	71,714	1.97	3.80	1.57	608
d16over	0.972	1.00	16	15.55	36,443	1.00	2.42	1.00	388
dPenryn	1.333	1.37	16	21.33	51,761	1.42	3.46	1.43	553
r4comb	0.857	0.88	28	24.00	42,911	1.18	2.10	0.87	588

P_{ave} is average power measured at 100 MHz.

Table IV
RESULTS OF IMPLEMENTATIONS.

by overlapping two radix-4 stages for the implementation of the selection function.

The main challenge is in updating the partial result of square root ($S[j]$), necessary for speculation, by keeping the latency low. To achieve this objective, the following design choices were made:

- The two radix-4 selection functions are overlapped to reduce the cycle time.
- As a consequence, we need to update the partial result ($\widehat{A}$ and $\widehat{B}$) in the narrow path (q-path) to compute q_L correctly.
- Due to speculation in computing q_L, different values of m_k's might be required in the initial iteration. This problem is solved by replicating the m_k-tables (three of them) for the selection of q_L (Figure 6). Having symmetrical values of m_k makes the tables quite small (52 NAND2 each).
- In the W-path, because $F_H[j]$ and $F_L[j]$ operate radix-4 for sqrt, it is necessary to update the values of A and B according to q_H to compute the right $F_L[j]$. This is done by the block **AB update & D select** in Figure 3.

With respect to the Penryn-like unit, the advantages of performing speculation on the narrow path, instead of the wide path, are evident by looking at the results of Table IV.

Clearly, the combination of div/sqrt has an overhead over the implementation of division only, but the increase in clock cycle is limited (10%) and the increase in power dissipation can be reduced by more aggressive design methods for low power. This is going to be addressed in future work.

In conclusion, we have presented a radix-16 combined division and square root unit for binary64 (significand computation) that has the lowest latency among some recently presented units of the same type (combined div/sqrt), including a unit similar to the one of the Intel Core2 processor.

ACKNOWLEDGMENTS

The author wishes to thank Tomás Lang for his suggestions and comments on the design of the unit.

REFERENCES

[1] S. Oberman and M. Flynn, "Design issues in division and other floating-point operations," *IEEE Transactions on Computers*, pp. 154–161, February 1997.

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

[3] ——, *Digital Arithmetic*. Morgan Kaufmann Publishers, 2004.

[4] H. Baliga, N. Cooray, E. Gamsaragan, P. Smith, K. Yoon, J. Abel, and A. Valles, "Improvements in the Intel Core2 Penryn Processor Family Architecture and Microarchitecture," *Intel Technology Journal*, pp. 179–192, Oct. 2008, http://www.intel.com/technology/itj/2008/v12i3/3-paper/1-abstract.htm.

[5] N. Burgess and C. N. Hinds, "Design of the ARM VFP11 Divide and Square Root Synthesisable Macrocell," *Proc. of 18th IEEE Symposium on Computer Arithmetic*, pp. 87–96, July 2007.

[6] G. Gerwig, H. Wetter, E. M. Schwarz, and J. Haess, "High performance floating-point unit with 116 bit wide divider," *Proc. of 16th Symposium on Computer Arithmetic*, pp. 87–94, 2003.

[7] S. F. Oberman, "Floating-point division and square root algorithms and implementation in the AMD-K7 microprocessor," *Proc. of 14th Symposium on Computer Arithmetic*, pp. 106–115, 1999.

[8] NVIDIA. "Fermi. NVIDIA's Next Generation CUDA Compute Architecture". Whitepaper. [Online]. Available: http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf

[9] H. Sharangpani and H. Arora, "Itanium processor microarchitecture," *IEEE Micro*, vol. 20, no. 5, pp. 24–43, Sep/Oct 2000.

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

[11] A. Nannarelli and T. Lang, "Low-Power Radix-4 Combined Division and Square Root," *Proc. of the International Conference on Computer Design*, pp. 236–242, Oct. 1999.

[12] N. Burgess, "Retiming the ARM VFP-11 Divide and Square Root Macrocell," *Proc. of 41st Asilomar Conference on Signals, Systems, and Computers*, pp. 363–366, Nov. 2007.

[13] E. Antelo, T. Lang, P. Montuschi, and A. Nannarelli, "Digit-recurrence dividers with reduced logical depth," *IEEE Transactions on Computers*, vol. 54, pp. 837–851, July 2005.

[14] N. Burgess and C. Hinds, "Design Issues in Radix-4 SRT Square Root and Divide Unit," *Proc. 35th Asilomar Conference on Signals, Systems and Computers*, pp. 1646–1650, 2001.