VLSI DESIGN
2000, Vol. 11, No. 4, pp. 331 – 338
Reprints available directly from the publisher
Photocopying permitted by license only

© 2000 OPA (Overseas Publishers Association) N.V.
Published by license under
the Gordon and Breach Science
Publishers imprint.
Printed in Malaysia.

Design and Analysis of Radix-8/4/2 64b/32b Integer Divider Using COMPASS Cell Library*

CHUA-CHIN WANG†, CHENN-JUNG HUANG and I-YEN CHANG

Department of Electrical Engineering, National Sun Yat-Sen University, Kaohsiung, Taiwan 80424

(Received 5 June 1999; In final form 24 September 1999)

A high speed 64b/32b integer divider employing digit-recurrence division method and the on-the-fly conversion algorithm, wherein a fast normalizer is included, which is used as the pre-processor of the proposed integer divider. For the sake of enhancing throughput rate, the proposed divider uses a mixed radix-8/4/2 division instead of the traditional radix-2 division. On-the-fly remainder adjustment is also realized in the converter module of the divider. The entire design is written in Verilog HDL (hardware description language) employing COMPASS 0.6 μm 1P3M cell library (V3.0), and then synthesized by SYNOPSYS. The simulation results indicate that our design is a better option than the existing long divider designs.

Keywords: Integer divider; On-the-fly conversion; Fast normalizer

1. INTRODUCTION

Integer division is a critical operation in the CPU design, since the number of clock cycles to complete an integer is probably very long and unpredictable [1 – 3]. The role of division is becoming more and more critical owing to the requirement of signed computer arithmetic, the modulus computation, the calculation of encryption keys, and so on. Division algorithms can be roughly classified into two categories: namely, digit-recurrence methods [4, 5], and functional iteration techniques [4, 6], while the former is commonly used. Regarding the digit-recurrence method, traditionally there are two types of division schemes, *i.e.*, restoring and non-restoring schemes. However, they both require multiple operation steps to derive a quotient bit. Not only is the efficiency drastically poor, but also a long adder/subtracter is needed to execute the remainder bit adjustment. These difficulties lead to the degradation of the entire microprocessor. Although high-radix division algorithm has been proposed to overcome the mentioned problems [5, 7], there are a few things left unsolved. First, how to efficiently normalize the dividend and the

* This research was partially supported by National Science Council under grant NSC 88-2219-E-110-001.
† Corresponding author. Tel.: 886-7-525-2000 ext. 4144, Fax: 886-7-5254199, e-mail: ccwang@ee.nsysu.edu.tw

divisor. Second, how to correctly adjust the final quotient and remainder without paying too many H/W overheads. In addition, though many research works has been proposed to either enhance the speed or the throughput [4–6], [8–10], the real hardware realization of a long divider is still a challenging task. The difficulties involved in the hardware realization include how to meet the minimal clock period, how to rapidly normalize given data words, how to control the operation sequence of different modules such that no racing problem occurs, and so on.

In this work, we thoroughly complete the VLSI implementation of a long 64b/32b signed integer divider wherein a pipelined fast normalizer, radix-8/4/2 digit-recurrence algorithm, and on-the-fly conversion method [6]. The proposed design methodology can also be applied to a longer divider, *e.g.*, 128b/64b signed integer divider. All of these works are physically implemented by using Verilog code integrated with COMPASS 0.6 μm 1P3M cell library in the Cadence cadtool environment. The simulation results show that our design is better than the existing long divider designs.

2. CELL-BASED DESIGN OF 64B/32B SIGNED INTEGER DIVIDER

2.1. Digit-recurrence Theory

Assume x, d, q, rem to be the dividend, the divisor, the quotient, and the remainder in the division operation. We also denote the radix of the division is r. Define a residual (partial remainder) w so that in the jth step of division is

$$w[j] = r^j(x - d \cdot q[j]). \tag{1}$$

According to [5], the digit-recurrence algorithm is described as follows:

- One digital arithmetic left-shift of $w[j]$ to produce $r \cdot w[j]$ except the first step;
- Determination of the quotient digit q_{j+1} by the quotient-digit selection function;
- Generation of the divisor multiple $d \cdot q_{j+1}$;
- Subtraction of $d \cdot q_{j+1}$ from $r \cdot w[j]$,

where

$$-d < w[j] < d. \tag{2}$$

$$\text{rem} = \begin{cases} w[n] \cdot r^{-n} & \text{if } w[n] \geq 0 \\ (w[n] + d) \cdot r^{-n} & \text{if } w[n] < 0. \end{cases} \tag{3}$$

Figure 1 shows the data flow of a division step.

Although the above algorithm has been well written in literature [5], the following unsolved problems still appear during the implementation:

(a) Fast normalization of the dividend and the divider is ignored.
(b) A long adder is needed at the adjustment of the remainder.
(c) Extra adjustment actions are required when the last cycle of the division contains non-multiple digits of the radix.

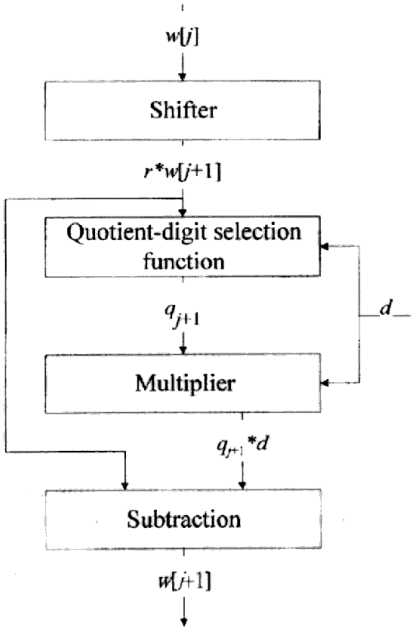

FIGURE 1 The data flow of a division step.

(d) The adjustment of the remainder is missing when the signed division is executed.
(e) A data flow control unit is required, which provides correct timing control such that the results of the division can be correctly placed on the output ports.

In short, the above problems will occur during the realization of a long signed divider. If these problems are not resolved efficiently, the hardware divider will be large and slow.

2.2. Design of the 64b/32b Signed/Unsigned Integer Divider

In this work, we present an improved design of a 64b/32b signed/unsigned integer divider, where the long ignored implementation problems mentioned above are all resolved. The key design issues of our integer divider are enumerated as follows:

2.2.1. Fast Normalizer

Binary data normalizer is one of the major time bottlenecks in dividers [5, 6]. If the sequential style of normalizers is used, the average time for a dividend or divisor normalization will be very long. The task of normalizer is to find the bit position of the first leading "1" of the given binary data. Since the data is unknown, the worst case of the time complexity will be $O(N)$, [8, 9]. From the viewpoint of data flow, the combinational design will be faster than the sequential design. Hence, we adopt a fast and scalable design methodology to normalize the binary data with the time expense $\approx O(\log N)$.

Assume the length of the data word is N, which is the power of 2. The entire word is divided into subwords with the length n, which is also the power of 2. Hence, the number of subwords is N/n. We can utilize modified priority encoders to locate the leading "1" in a subword.

The bit position of the leading "1" can be detected by an n-bit priority encoder (PE). The output of the PE is the binary representation of the position of the leading "1" in the subword. The length of the output representation is, then, $k = \lceil \log_2 n \rceil$. The function table of the PE is shown in Table I:

We still can not figure out where the global leading "1" is at this stage, even though the respective leading "1" is known in each subword. A total of N/n n-input OR gates and another PE, called the high-level PE, are required to generate the select signals telling which subword the leading "1" is located. This high-level PE and the PEs used in the subwords are arranged in a hierarchical format. The output of the high-level PE is the selection signals of a total of k N/n-way-to-1 MUXs. The architecture of the entire fast normalizer is shown in Figure 2 where $N = 64$, and $n = 4$. Notably, the outputs of these PEs are utilized for two tasks:

(1) computing the required number of cycles to generate the correct quotient and the remainder;
(2) instructing a barrel shifter to shift the original data word properly.

TABLE I The function table of the priority encoder (PE)

Input (n bits)	Output (k bit)	Decimal notation
1XXX...X	11...11	0
01XX...X	11...10	1
001X...X	11...01	2
⋮	⋮	⋮
000...0X	00...00	$n-1$

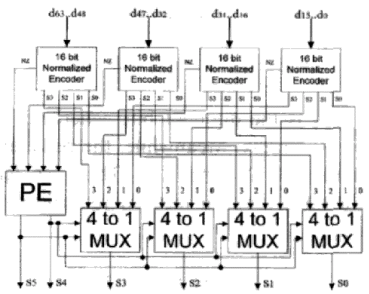

FIGURE 2 The architecture of fast normalizer ($N = 64$, $n = 4$).

2.2.2. Radix-8 Division with Radix-4 and Radix-2 Selection Functions

The next problem that we like to resolve is the redundant step occurring at the last step of the division. Since the radix-8 is used in the division, there is a possibility that the last stage of division has only one or two bits left in the dividend to be processed. If only one radix-8 selection function [5] is used at this stage, an extra adjustment step will be needed to correct the result. This introduces additional delays and hardware cost, *e.g.*, long adders. We thus integrate the radix-4 and radix-2 selection functions in the division to overcome this difficulty. The control unit will monitor the number of bits to be computed in the fixed-point quotient. The radix-4 division will be executed at the last stage when the number of bits to be computed in the fixed-point quotient is two, whereas the radix-2 division will be executed when the number of bits left in the quotient is one. Moreover, in our design we can take advantage of that the positions of leading "1" in the dividend and the divider can be detected in the normalizer such that the total number of division steps is well determined before the iterative digit-recurrence mechanism.

2.2.3. Radix-8 (High Radix) Quotient Selection Function Table

It can be shown that the residual is computed basing on the following equality.

$$w[j+1] = r \cdot w[j] - D \cdot q_{j+1}. \tag{4}$$

where q_{j+1} is the quotient bits generated at step $j+1$, r is the radix. Meanwhile, the residual must be bounded, $-D < w[j] < D$. Thus, we tend to utilize a table look-up method to realize such a function,

$$q_{j+1} = \mathrm{SEL}(w[j], D) \tag{5}$$

The SEL(·) in the above function is called "quotient selection function" [5].

2.2.4. Hardware Consideration of Signed Division

Notably, the sign of the remainder should be the same as that of the dividend. This results in an adjustment problem of the remainder at the last stage of the division. Usually a full wordlength adder is required to handle this problem. In our design, both the dividend and the divider are converted into positive numbers before the normalization. Their sign information is then kept and used to select the result generated by the 37-b carry save adder (CSA) for the remainder adjustment. This will simplify the entire design and have not loss regarding speed.

2.2.5. Data Flow Control Unit

Our cell-based design for the 64b/32b signed/unsigned integer divider is given in Figure 3. The detailed flow control is described as follows:

(1) Convert the dividend and the divider into positive numbers. Then use the fast normalizer to execute the normalization.
(2) Compute the required cycles for radix-8, radix-4, and radix-2 division by the positions of the leading "1" of the dividend and the divider which can be generated by the normalizer.
(3) In each radix-8 division cycle, use the radix-8 selection function to generate 3 bits for the quotient.
(4) In the radix-4 (radix-2) division cycle, use the radix-4 (radix-2) selection function to generate the remaining bit(s) for the quotient.
(5) A radix-8/4/2 on-the-fly converter is used to generate the quotient and avoid any possible carry ripple. This converter is controlled by multiplexers such that it can be used by the radix-8, radix-4 and radix-2 selection functions.
(6) Use a carry-save adder to filter out the carry ripple produced in every quotient generation step. Notably, the error will be absorbed in the next usage of the selection function.
(7) The last stage is to adjust the remainder by a fast adder, whose bit length is $64 + \log_2 r + 1 = 68$. Meanwhile the barrel shifter in the normalizer is used to produce the final remainder.

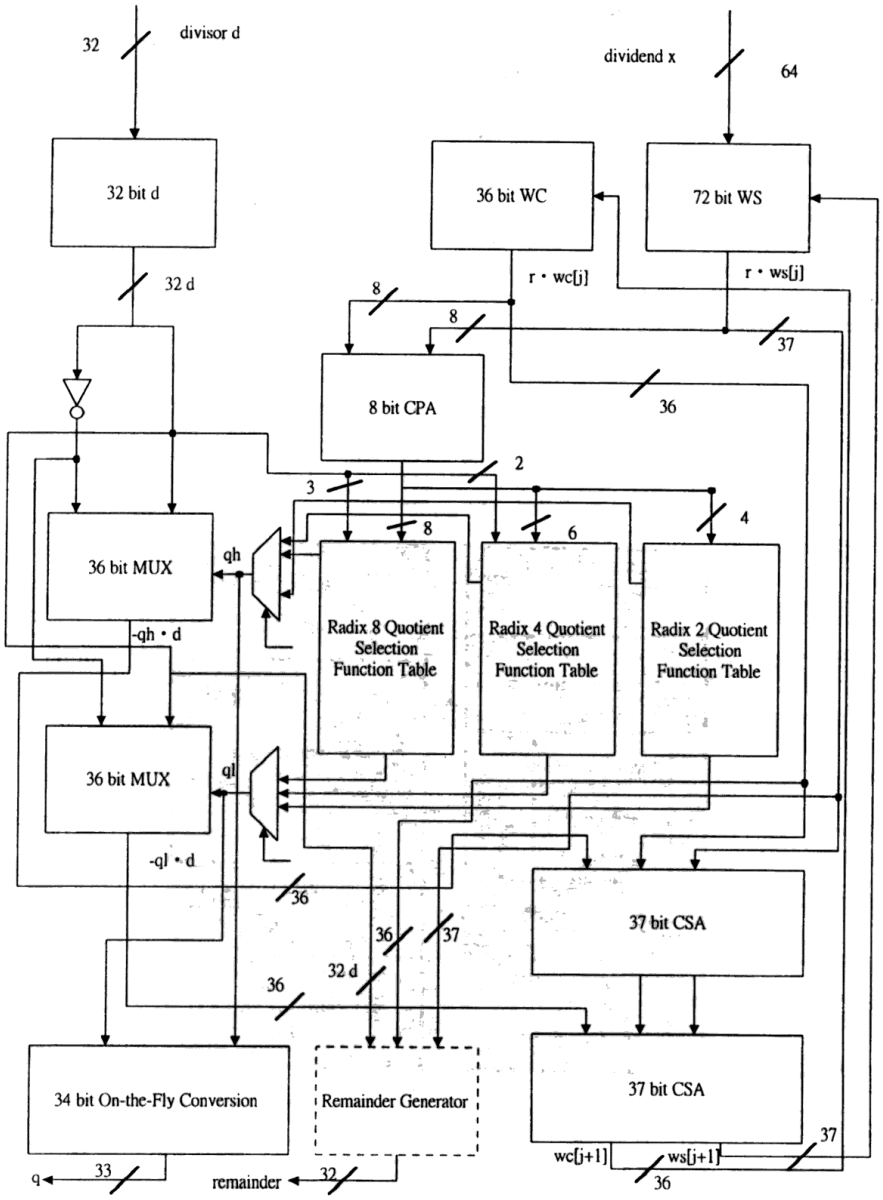

FIGURE 3 The design architecture of the 64b/32b signed/unsigned integer divider.

3. SIMULATION AND ANALYSIS

In order to compare with currently available design methodologies for long integer dividers, we use the Verilog HDL incorporated with COMPASS 0.6 μm 1P3M cell library (version 3.0) to synthesize the 64b/32b signed/unsigned divider by SYNOPSYS. Figure 4 shows the circuit layout of the integer divider, while the synthesis results of every submodule is given in Table II. We also use the TimeMill to execute the full-chip-scale post-layout simulation. The test patterns produced by the Verilog behavioral code are fed into TimeMill to test under different clock rates.

At this stage, we find that the chip functions correctly up to a 66 MHz clock.

For the sake of realizing the performance improvement of the proposed design in the long integer division, we compare our work with currently available CPUs' integer divider, including [2, 3], to present the superior design of our divider chip, as shown in Table III. Note that the entry in Table III is the number of clock cycles.

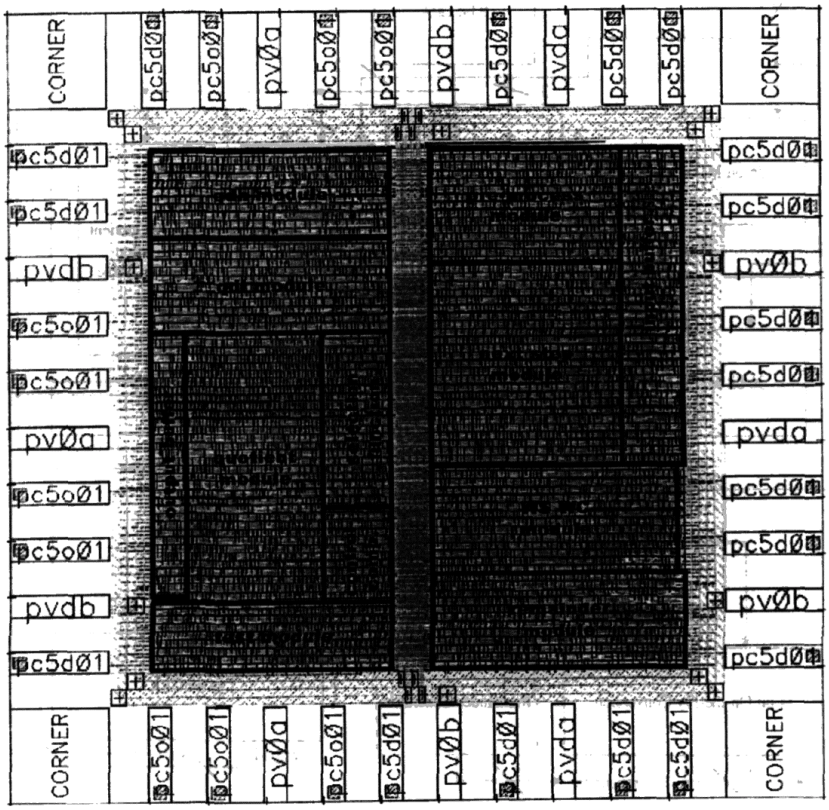

FIGURE 4 The circuit layout of the integer divider.

TABLE II Synthesis results of every submodule of the chip by SYNOPSYS

Component	Area (cell unit)	Area percent	Max delay
Pre-process module	1033.29	5.9291%	4.47 ns
Next step module	3063.85	17.5808%	6.73 ns
WS WC module	2679.38	15.3746%	4.93 ns
Selection function module	970.66	5.5698%	7.78 ns
qdh and qdl module	2122.26	12.1778%	1.24 ns
Quotient module	2489.78	14.2867%	4.81 ns
Remainder module	2757.53	15.8231%	16.07 ns
Control component	166.39	0.9548%	4.30 ns
Input buffer	1093.76	6.2762%	2.66 ns
Output buffer and test module	1050.35	6.0271%	4.14 ns
Total	17468.76	100.00000%	

TABLE III Cycle-based performance comparison of 64b/32b integer dividers

	Pentium	Cyrix 6 × 86MX	Radix-4/2 divider [11]	Our divider
Integer division	42–4 (longest–shortest)	45–13 (longest–shortest)	23–3 (longest–shortest)	18–3 (longest–shortest)

4. CONCLUSION

In this work we present an improved design of a 64b/32b signed/unsigned integer divider. Not only we show the feasibility of using the mixed radix-8/4/2 method, those long ignored implementation problems are also resolved. The simulation results indicate that our design is a better option than the existing long divider designs. Notably, this design can be integrated in the ALU unit of a 64-bit microprocessor.

References

[1] "*Pentium Pro Family Developer's Manual*", Intel, 1996.

[2] Gwennap, L., "Intel's P6 uses decoupled superscaler design", *Microprocessor Report*, **9**(2), Feb., 1995.

[3] Gwennap, L., "Klamath extends P6 family", *Microprocessor Report*, **11**(2), Feb., 1997.

[4] Bashagha, A. E. and Ibeahim, M. K., "Two's complement high radix division", *1997 IEEE Inter. Symp. on Circuits & Systems (ISCAS'97)*, pp. 2088–2091, Hong Kong, June, 1996.

[5] Ercegovac, M. D. and Lang, T., "*Division and square root – digit-recurrence algorithms and implementations*", Reading: Kluwer Academic Publishers, 1994.

[6] Hwang, K., "*Computer arithmetic: principles, architectures, and designs*", Reading: John Wiley & Sons, 1979.

[7] Ercegovac, M. D., Lang, T. and Montuschi, P., "Very-high radix division with prescaling and selection by rouding", *IEEE Trans. on Computers*, **43**(6), 909–917, Aug., 1994.

[8] Cavanagh, J. F., "*Digital computer arithmetic*", McGraw-Hill, Inc., 1984.

[9] Hayes, J. P., "*Computer architecture and organization*", McGraw-Hill, Inc., 1988.

[10] Sirnivas, H. R. and Parhi, K. K., "A fast radix-4 division algorithm", *IEEE Inter. Symp. on Computer Arithmetic*, pp. 311–314, Santa Monica, 1994.

[11] Wang, C.-C., Huang, C.-J., Lin, G.-C. and Wu, C.-F., "A chip design of Radix-4/2 64b/32b signed/unsigned integer divider using Compass cell library", *1999 IEEE Inter. Symp. on Circuits and System*, **I**, 439–442, June, 1999.

Authors' Biographies

Chua-Chin Wang was born in Taiwan, in 1962. He received the B.S. degree in electrical engineering from National Taiwan University, Taiwan, in 1984 and the M.S. and Ph.D. degrees in electrical engineering from State University of New York, Stony Brook, in 1988 and 1992, respectively. Currently he is a Professor in the Department of Electrical Engineering, National Sun Yat-Sen University, Taiwan. His research interests include low-power logic and circuit design, VLSI design, and neural networks and implementations.

Chenn-Jung Huang was born in Hualien, Taiwan, in 1961. He received the B.S. degree in electrical engineering from National Taiwan University, Taiwan, in 1984 and the M.S. degree in computer science from University of Southern California, Los Angeles, in 1987. He is currently completing requirements for the Ph.D. degree in electrical engineering at National Sun Yat-Sen University, Taiwan. His research interests are computer arithmetic, computer communication networks, and neural networks.

I-Yen Chang was born in Taiwan, in 1975. He received the B.S. degree in information engineering from Chung Yuan Christian University, Taiwan, in 1997 and the M.S. degree in electrical engineering from National Sun Yat-Sen University, Taiwan, in 1999. His research interests include VLSI design and computer arithmetic.