A CHIP DESIGN OF RADIX-4/2 64B/32B SIGNED AND UNSIGNED INTEGER DIVIDER USING COMPASS CELL LIBRARY

C.-C. Wang, C.-J. Huang, & G.-C. Lin

Department of Electrical Engineering

National Sun Yat-Sen University

Kaohsiung, Taiwan 80424

ABSTRACT

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 radix-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. At last, a real chip is fabricated and fully tested. The test results turn out to be very impressive. Besides, a performance evaluation of a 128b/64b signed integer divider using the same design methodology is also included in this study.

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][2][3]. The role of division is becoming more and more critical owing to the requirement of signed computer arithmetics, 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 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][5][6][8][9][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-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 final chip layout has been tapeout and delivered to the TSMC (Taiwan Semiconductor Manufacturing Company) to produce the real product. At last, the real chip in DIP package is fabricated and fully tested by IMS digital tester of ATS. The test results verify the correctness of our design

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

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

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

Fig. 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:

0-7803-5471-0/99/$10.00©1999 IEEE

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.

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.

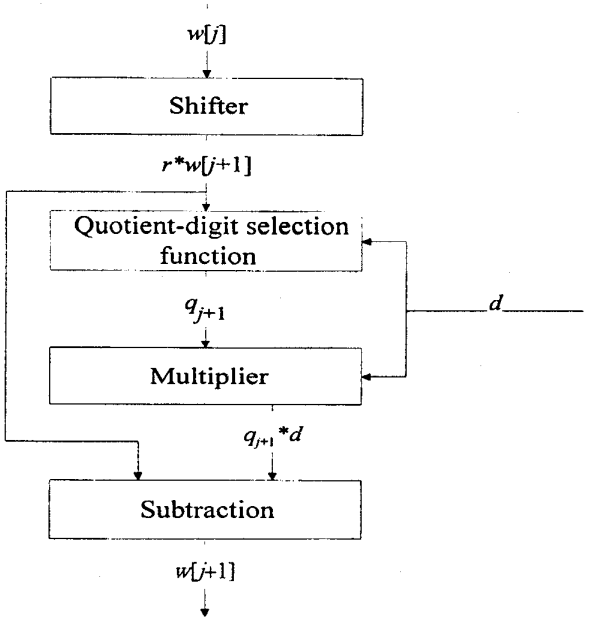

Fig. 1 The data flow of a division step.

2.2 Design of the 64b/32b Signed/unsigned Integer Divider

In this work, we present an improved design of a long 128b/64b signed/unsigned integer divider and a physical 64b/32b signed integer divider chip implementation, where the long ignored implementation problems mentioned above are all resolved. The key design issues of our integer divider are enumerated as follows:

- 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 the following:

Input (n bits)	Output(k bit)	Decimal Notation
1XXX...X	11...11	n
01XX...X	11...10	n - 1
001X...X	11...01	n - 2
⋮	⋮	⋮
0000...0	00...00	0

Table 1 : The function table of the priority encoder (PE).

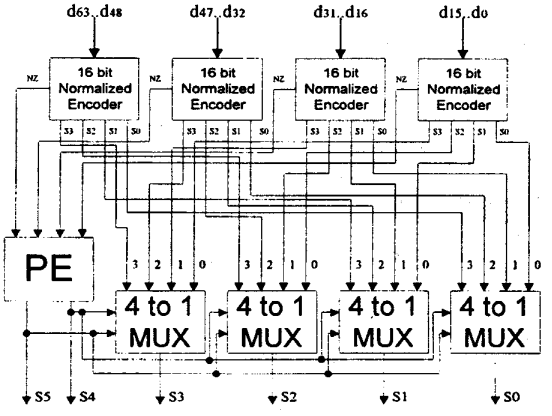

Fig. 2 The architecture of fast normalizer ($N = 64$, $n = 4$).

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 Fig. 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.

- Radix-4 Division with a Radix-2 Selection Function

The next problem that we like to resolve is the redundant step occurring at the last step of the division. Since the radix-4 is used in the division, there is a possibility that the last stage of division has only one bit left in the dividend to be processed. If only one radix-4 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-2 selection function in the division to overcome this difficulty. The control unit will monitor the number of bits left in the dividend such that the radix division will be executed at the last stage when the number of bits of the dividend is odd. 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.

- Radix-4 (High Radix) Quotient Selection Function Table

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

$$w[j+1] = r \cdot w[j] - D q_{j+1} \quad (4)$$

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,

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

The $SEL()$ in the above function is called "quotient selection function", which is shown in Fig. 3.

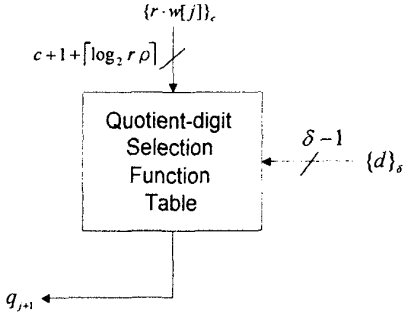

Fig. 3 The quotient-digit selection function table with inputs and outputs.

- 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 35-b carry save adder (CSA) for the remainder adjustment. This will simplify the entire design and have not loss regarding speed.

- Data Flow Control Unit

Our cell-based design for the 64b/32b signed/unsigned integer divider is given in Fig. 4. Notably, the hardware penalty of using the radix-4/2 division is that of a total of 133+65+4+67 2-to-1 MUXs plus a few simple primitive gates are needed in addition to the original H/W of a pure radix-4 selection function. However, the gain regarding the timing to handle the adjustment for 1 bit quotient is worthwhile.

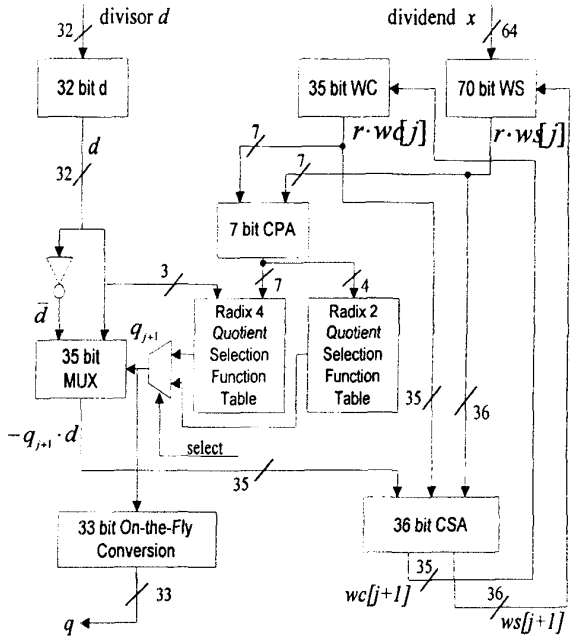

Fig. 4 The design architecture of the 64b/32b signed/unsigned integer divider.

3. PERFORMANCE EVALUATION & CHIP IMPLEMENTATION

In order to compare with currently available design methodologies for long integer dividers, we extend our design approach by using the Verilog HDL incorporated with COMPASS 0.6 μm 1P3M cell library (version 3.0) to synthesize such a 128b/64b signed/unsigned divider by SYNOPSYS. The detailed numerical report is shown in Table 2.

Although the longest delay is almost 18 ns, it doesn't imply that the shortest period of the working clock has to be the same value. The reason is that the division by the digit-recurrence method requires many "steps"; while each step is triggered by the

working clock. We will find out what the maximum working frequency is later by simulations. In fact, the digit-recurrence is based on the clock cycles. We, thus, randomly generate over 30,000 test vectors to find out the possible shortest period of the clock. The result shows that a clock with 12 ns period will provide correct division result.

Part name	Total area (gate count)	critical delay (ns)
combinational	17946.667969	17.56
control unit	347.993408	7.25

Table 2 : Performance evaluation by SYNOPSYS.

In order to realize the performance improvement of the proposed design in the long integer division, the following table shows the comparison with other published designs.

Design	Area (gate count)	longest delay (gate delays)	longest delay (cycles)
rpl model of [11]	≈ 100,000	≥ 12,000 ns	-
bla model of [11]	≈ 100,000	≥ 1,200 ns	-
divider of P-II [1]	(NA)	(NA)	33 (64b/32b)
Our design	≤ 20,000	41×12=492 ns	41

Table 3: Performance comparison of 128b/64b integer dividers.

The above results show that our design indeed possesses the advantages regarding the area and the speed.

We also 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 4. Note that the entry in Table is the number of clock cycles.

	Pentium	Cyrix 6x86MX	Our divider
Integer Division	42 - 4 (longest - shortest)	45 - 13 (longest – shortest)	23 -3 (longest - shortest)

Table 4: Cycle-based performance comparison of 64b/32b integer dividers.

The real 64b/32b chip has been tested by IMS digital tester of ATS Co. The die photo of the chip is shown in Fig. 5. The chip has been tested by 1000 groups of dividend and divider pairs, which are randomly generated, and the results of corresponding quotient and remainder pairs are all correct.

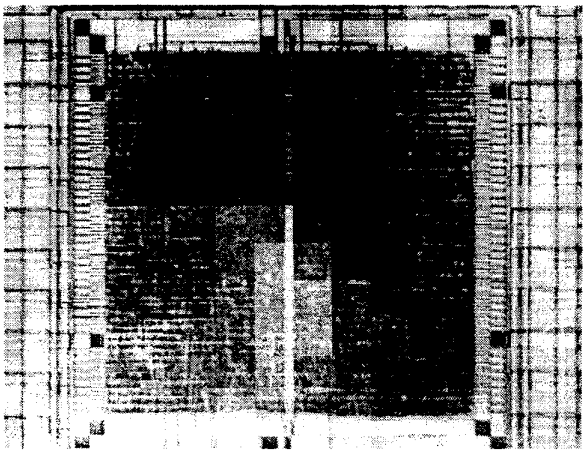

Fig. 5 The die photo of the chip.

4. REFERENCES

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

[2] L. Gwennap, "Intel's P6 uses decoupled superscaler design," *Microprosessor Report*, vol. 9, no. 2, Feb. 1995.

[3] L. Gwennap, "Klamath extends P6 family," *Microprocessor Report*, vol. 11, no. 2, Feb. 1997.

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

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

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

[7] M. D. Ercegovac, T. Lang, and P. Montuschi, "Very-high radix division with prescaling and selection by rouding," *IEEE Trans. on Computers*, vol. 43, no. 6, pp. 909-917, Aug. 1994.

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

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

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

[11] K. Baty, "Design Ware," Reading : pp. B-3 to B-12, Synopsys, Inc. 1996.