Cell-based implementation of radix-4/2 64b dividend 32b divisor signed integer divider using the COMPASS cell library

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

Abstract: A high-speed 64b/32b integer divider using the digit-recurrence division method and the on-the-fly conversion algorithm, is presented. A fast normaliser is used as the preprocessor of the proposed integer divider. To reduce maximum division time, the proposed divider uses radix-4/2 division, instead of the traditional radix-2 division. On-the-fly quotient adjustment is also realised in the converter module of the divider. The entire design is written in the Verilog hardware description language using the COMPASS 0.6 μm 1P3M cell library (V3.0), and then synthesised by SYNOPSYS. Finally a real chip is fabricated and fully tested. The test results are very impressive. 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 CPU design, since the number of clock cycles to complete an integer is usually very long and unpredictable [1–3]. The role of division is becoming more and more critical, owing to the requirement of signed computer arithmetic, modulus computation, the calculation of encryption keys, and so on. Division algorithms can be roughly classified into two categories: digit-recurrence methods [4, 5], and functional iteration techniques [6]. The former is most commonly used. In the digit-recurrence method, there are traditionally two types of division schemes, (i.e. restoring and nonrestoring 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/subtractor is needed to execute the remainder bit adjustment. These difficulties lead to the degradation of the entire microprocessor. Although a high-radix division algorithm has been proposed to overcome these difficulties [5, 7], there are a few things left unsolved. First, how to efficiently normalise the dividend and the divisor. Secondly, how to correctly adjust the final quotient and remainder without paying too many H/W overheads. In addition, although much research work has been proposed to either enhance the speed or the throughput [4–6], [8–10], the real hardware realisation of a long divider is still a challenging task. The difficulties involved in the hardware realisation include meeting the minimal clock period, rapidly normalising given data and controlling the operation sequence of different modules such that no racing problem occurs.

© IEE, 2000

IEE Proceedings online no. 20000160

DOI: 10.1049/ip-cdt:20000160

Paper first received 27th November 1998 and in revised form 14th September 1999

The authors are with the Department of Electrical Engineering, National Sun Yat-Sen University, Kaohsiung, Taiwan 80424

In this work, we complete the VLSI implementation of a long 64b/32b signed integer divider where a pipelined fast normaliser, radix-4/2 digit-recurrence algorithm, and on-the-fly conversion method are used [5]. The proposed design methodology can also be applied to a longer divider (e.g. 128b/64b signed integer divider). All of these works are implemented physically by using Verilog code integrated with the COMPASS 0.6μm 1P3M cell library in the Cadence cadtool environment. The final chip layout has been sent to the TSMC (Taiwan Semiconductor Manufacturing Company) to produce a real product. Finally, the real chip in a DIP package is fabricated and fully tested by an IMS digital tester of ATS. The test results verify the correctness of our design.

2 Cell-based design of 64b/32b signed integer divider

In this work, we combine the radix-4 and radix-2 division algorithms [5] to increase the throughput, and only small adders/subtracters are employed in the entire design. Meanwhile, we also present a fast normaliser so that the number of cycles to complete a correct division can be further reduced.

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, respectively. We also denote the radix of the division is denoted r. Define a residual (partial remainder) w so that in the jth step of division:

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

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

(i) one digital arithmetic left-shift of $w[j]$ to produce $r{\cdot}w[j]$ except the first step;

(ii) determination of the quotient digit q_{j+1} by the quotient-digit selection function;
(iii) generation of the divisor multiple $d \cdot q_{j+1}$;
(iv) subtraction of $d \cdot q_{j+1}$ from $r \cdot w[j]$,

where

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

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

Fig. 1 shows the data flow of a division step.

Although the above algorithm has been widely covered in the literature [5], there are many unsolved difficulties when it comes to realising such a divider in hardware. Further problems will occur when a higher radix is employed to increase the throughput because of increasing cycle times and circuit complexity. Many problems will appear during the implementation of the signed integer divisor, including the following.

(i) Fast normalisation of the dividend and the divisor is used as the preprocessor of the integer divider, since the iterative digit-recurrence mechanism mentioned above is only applied to fractional operands and quotient.
(ii) A long adder is needed at the adjustment of the remainder if a carry-save adder is employed at step (iv) of the digit-recurrence algorithm.
(iii) Extra adjustment actions are required when the last cycle of the division contains nonmultiple digits of the radix. (For instance, the radix is 4, but only one bit is left in dividend to be processed.)
(iv) The adjustment of the remainder, based on eqn. 3, is required when the signed division is executed.
(v) 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 realisation of a long signed divider. If these problems are not resolved efficiently, the hardware divider will be large and slow.

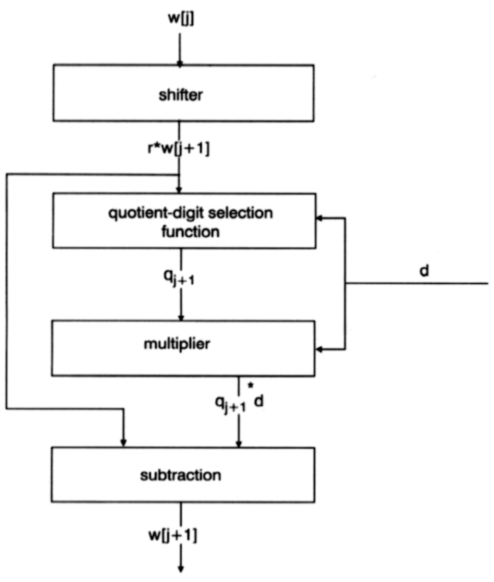

Fig. 1 *Date flow of division step*

2.2 Fast normaliser

A binary data normaliser is one of the major time bottlenecks in dividers [5, 6]. If the sequential style of normaliser is used, the average time for a dividend or divisor normalisation will be great. The task of normaliser 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], if no special design is utilised to resolve this problem. 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 normalise the binary data with a time expense of $\simeq O(\log N)$.

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 utilise modified priority encoders to locate the leading '1' in a subword.

2.2.1. Design of the fast normaliser:

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 1.

We still cannot 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, the high-level PE, is required to generate the select signals which indicate in which indicate in 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 a 16-bit normaliser is shown in Fig. 2 where $N=16$

Table 1: 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	$n-1$

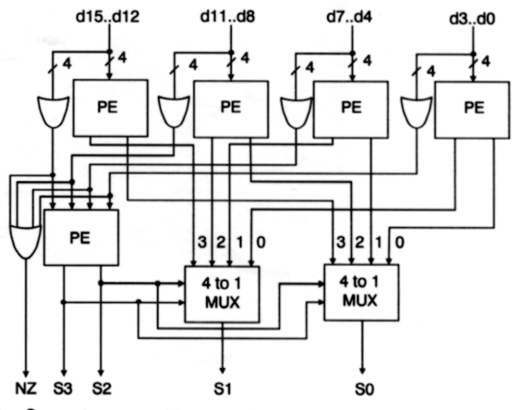

Fig. 2 *Architecture of fast normaliser*
$N=16$, $n=4$

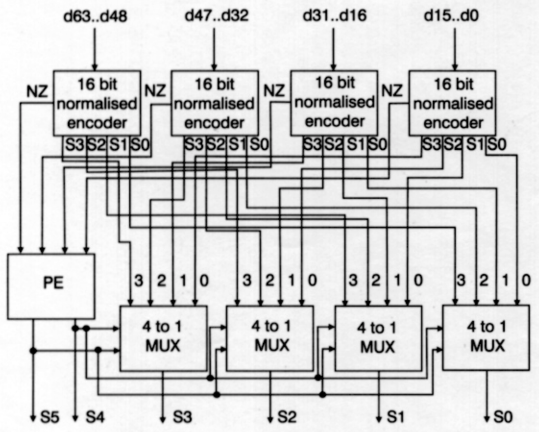

Fig. 3 *Architecture of 64-bit normaliser*

and $n = 4$. Based on 16-bit normalisers, a 64-bit normaliser can be further constructed as shown in Fig. 3. Notably, the outputs of these PEs are utilised for two tasks:

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

2.3 Radix-4 division with a radix-2 selection function

Next we resolve 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 the fact that the positions of the leading ‘1’ in the dividend and the divisor can be detected in the normaliser, such that the total number of division steps is well determined before the iterative digit-recurrence mechanism.

An analytic expectation of cycles required by such a radix-4/2 method is given as follows: recall that the following equation holds according to the definition of the integer division.

$$x = q \cdot d + rem \tag{4}$$

After being normalised by the PEs described in the previous Section, the dividend and the divisor become:

$$\begin{aligned} x &= X \cdot 2^{-(N-a)} \cdot 2^{-2} \\ d &= D \cdot 2^{-(N-b)} \end{aligned} \tag{5}$$

where X and D are the integer dividend and divisor, respectively, a and b are the number of bits that x and d shifts, respectively, and N is the bit length of the data words. Note that we need to multiply x by 2^{-2} at the beginning in order to make $x < d$. Hence, an interesting result will be obtained after a few essential mathematical manipulations.

$$\begin{aligned} x &= q \cdot d + rem \\ &\Rightarrow X \cdot 2^{-(N-a)} \cdot 2^{-2} = D \cdot 2^{-(N-b)} \cdot q + rem \\ &\Rightarrow X = D \cdot 2^{-(N-b)} \cdot q \cdot 2^{(N-a+2)} + rem \cdot 2^{(N-a+2)} \\ &\qquad = D \cdot q \cdot 2^{b-a+2} + rem \cdot 2^{N-a+2} \end{aligned}$$

Thus, we conclude that:

$$Q = q \cdot 2^{b-a+2} \tag{6}$$

$$\begin{aligned} REM &= rem \cdot 2^{(N-a)+2} = w[j+1] \cdot 2^{-((b-a)+2)} \cdot 2^{(N-a)+2} \\ &= w[j+1] \cdot 2^{N-b} \end{aligned} \tag{7}$$

where $w[j+1]$ is the residual generated at each iteration step of the division algorithm.

We derive that, since Q is an integer, we only need to compute the $b-a+2$ digits of q to generate the quotient, i.e. we do not have to calculate every digit of Q to obtain the correct result. This number of digits (i.e., $b-a+2$) has been computed and predicted in the normalisation phase such that the required cycles to compute the division result are drastically reduced. Also, the cycles required by the radix-4 division steps are $\lfloor b-a+2/2 \rfloor$, while the cycles for radix-2 division steps are $(b-a)\%2$. Then, the longest division which occurs at $2^{63}/(2^{31}+1)$ is 17 cycles per 2-bit quotient. If the initialisation and the adjustment of the remainder are both considered, the total cycles for such a division are $(17+3+3) = 23$. The shortest division requires only three cycles, which occurs when the dividend is zero.

Note that the redundant digit set $\{-2, -1, 0, 1, 2\}$ is employed in the generation of two digits of the quotient. To avoid the carry ripple effect produced by any negative number, we utilise the on-the-fly conversion method to produce the quotient digits such that a large adder/subtracter will not be needed.

2.4 Radix-4 (high radix) quotient selection function table

According to eqns. 1 and 6, the quotient digits that we will get after j steps are:

$$q[j] = \sum_{i=1}^{j} q_i \cdot r^{-i} \tag{8}$$

where q_i is the quotient bits generated at step i, and r is the radix. In each step of deriving the quotient digits, the residual will be computed based on the following equality:

$$w[j+1] = r \cdot w[j] - D \cdot q_{j+1} \tag{9}$$

Meanwhile, the residual must be bounded $-D < w[j] < D$. Thus, we tend to utilise a table look-up method to realise such a function.

$$q_{j+1} = SEL(w[j], D) \tag{10}$$

The SEL$(\cdot)$ in the above function is called ‘quotient selection function’ [5], which is shown in Fig. 4.

2.5 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

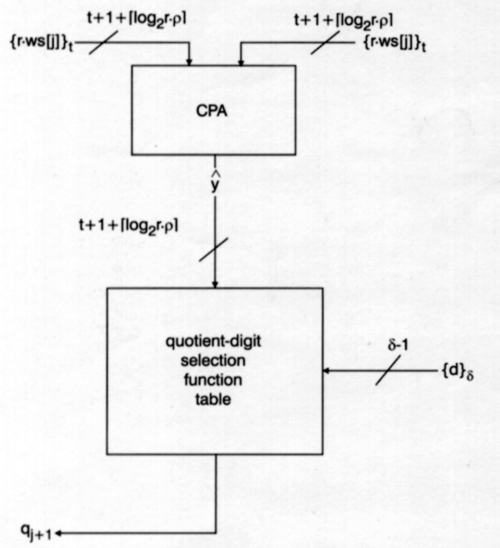

Fig. 4 *Quotient-digit selection function table with inputs and outputs*

full word-length adder is required to handle this problem. Although Bashagha *et al.* [4] proposed an algorithm to avoid using the long adder, one full word-length adder is still needed to generate a selection signal before the adjustment of the remainder in their design.

In our design, both the dividend and the divisor are converted into positive numbers before the normalisation as Fig. 5 shows. Their sign information is then kept and used to select the result generated by the 35-b carry-save adder for the remainder adjustment. This will simplify the entire design and incur no loss of speed.

2.6 Data flow control unit

In Fig. 6 we present our cell-based design for the 64b/32b signed/unsigned integer divider. The detailed flow control is described as follows.

(i) Convert the dividend and the divisor into positive numbers. Then use the fast normaliser to execute the normalisation.
(ii) Compute the required cycles for radix-4 and radix-2 division by the positions of the leading '1' of the dividend and the divisor which can be generated by the normaliser.
(iii) In each radix-4 division cycle, use the radix-4 selection function to generate 2 bits for the quotient. A 7-b carry propagate adder is required at this step.
(iv) In each radix-2 division cycle, use the radix-2 selection function to generate 1 bit for the quotient. A 4-b carry propagate adder is required at this step.
(v) A radix-4/2 on-the-fly converter is used to generate the quotient and avoid any possible carry ripple. This converter is controlled by multiplexers, so that it can be used by the radix-4 selection function and the radix-2 selection function.
(vi) Use a 35-b 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.
(vii) The last stage is to adjust the remainder by a fast adder, whose bit length is $64 + \log_2 r + 1 = 67$. Meanwhile, the barrel shifter in the normaliser is used to produce the final remainder.

Notably, the hardware penalty of using the radix-4/2 division is that 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 time gain in handling the adjustment for 1 bit quotient is worthwhile.

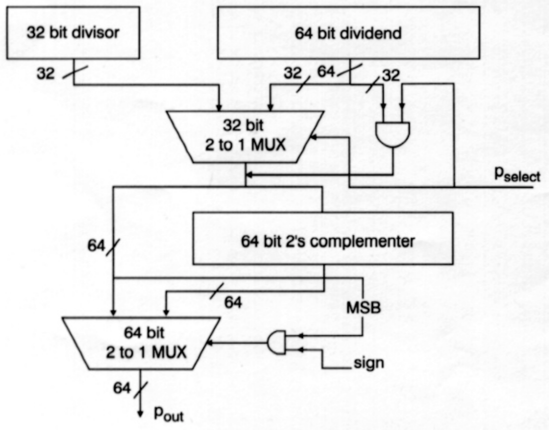

Fig. 5 *Conversion of dividend and divisor into positive numbers*

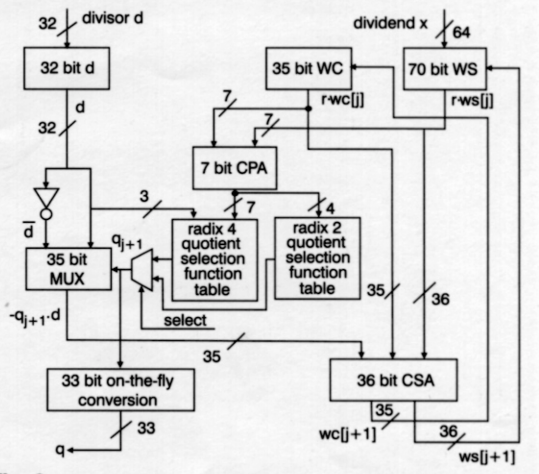

Fig. 6 *Design architecture of 64b/32b signed/unsigned integer divider*

3 Performance evaluation and chip implementation

3.1 Performance evaluation of a 128b/64b signed integer divider

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

Although the longest delay is almost 18 ns, it does not imply that the shortest period of the working clock has to be the same value. The reason is that the division by the

Table 2: Performance evaluation by SYNOPSYS

Part name	Total area (gate count)	Critical delay (ns)
Combinational	17947	17.56
Control unit	348	7.25

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

Design	Area (gate count)	Longest delay (gate delays)	Longest delay (cycles)
rpl model of [11]	$\simeq$100 000	$\geqslant$ 12 000 ns	–
bla model of [11]	$\simeq$100 000	$\geqslant$ 1200 ns	–
Divider of P-II [1]	(NA)	(NA)	33 (64b/32b)
Our design	$\leqslant$ 20 000	$41 \times 12 = 492$ ns	41

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. 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 a 12 ns period will provide the correct division result.

To realise the performance improvement of the proposed design in the long integer division, Table 3 shows the comparison with other published designs.

The above results show that our design possesses advantages both in area and speed.

3.2 64b/32b signed/unsigned integer chip implementation

Owing to budget limitations and performance comparison with current commercially available CPU integer dividers, we implemented a divider chip in a 64b/32b format. The entire design procedure was as follows.

Step 1. Respectively develop the Verilog synthesisable RTL code and nonsynthesisable behavioural code of the divider. Then, apply thousands of randomly generated test stimuli to compare the results of these two versions of Verilog code to verify function correctness.

Step 2. Use SYNOPSYS integrated with 1P3M COMPASS 0.6 μm cell library to synthesise the RTL code so that the gate-level Verilog code is generated. Repeat Step 1 to verify that the results of behavioural code, RTL code and the gate-level code are the same in the presence of thousands of test patterns. At this stage, we find that the maximum operating frequency is up to 150 MHz with pads, and up to 200 MHz without pads.

Step 3. Stream in the gate-level Verilog code in the Cadence design tool. Use Cell3 automatic place and route tools to generate the abstract view and the layout of the chip. At this stage, a total of 38 706 transistors are contained in the final layout, as shown in Fig. 7. The total area is $3351 \times 3312\,\mu m^2$, while the package is 40-pin DIP. We tended to reduce the I/O pins because of the budgetary problem. Hence, the I/O data are decomposed into bytes and transmitted in serial batch mode.

Step 4. After the final layout is generated, the TimeMill is utilised to execute the full-chip-scale post-layout simulation. The test patterns produced by the Verilog behavioural code are fed into the TimeMill for testing under different clock rates. At this stage, we find that the chip functions correctly up to a 50 MHz clock.

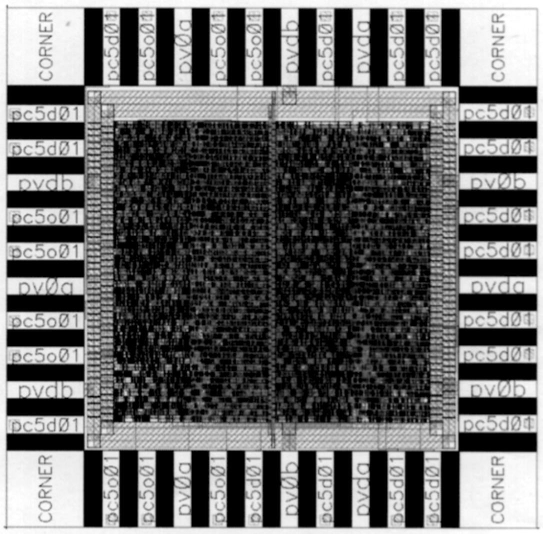

Fig. 7 *Final layout of divider chip*

To present the superior design of our divider chip, we compare our work with currently available CPU integer dividers, including [2, 3], as shown in Table 4. Note that the entry in the table is the number of clock cycles.

3.2.1. Pin-out assignment consideration: As we mentioned before, the pin count and chip area are strictly limited by a fabrication budget. The dividend and the divisor are partitioned into bytes and then sequentially read in the registers inside of the chip from low byte to high byte. They share a byte-wide input port, 'Operand' (8 bits). The selection of dividend or divisor is determined by a signal 'Wselect'. (1 = dividend, 0 = divisor).

Similarly, after the division is done ('OK' = 1), the results including the quotient and the remainder are output at 'Result' (8 bits). 'Wselect' determines which one is selected (1 = remainder, 0 = quotient). Another 3-bit 'Index' signals are used to indicate which byte the 'Result' is delivering. For instance. 'Index' – (000) denotes the lowest byte is presented at 'Result'; 'Index' = (001) denotes the second lowest byte appears at 'Result', and so on.

In addition to the above operands' I/O signals, there are a few signals required in integer division, including 'Sign' (1 = signed operation, 0 = unsigned operation), 'Start'

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

	Pentium	Cyrix 6 × 86MX	Our divider
Integer division	42 - 4 (longest - shortest)	45 - 13 (longest - shortest)	23 - 3 (longest - shortest)

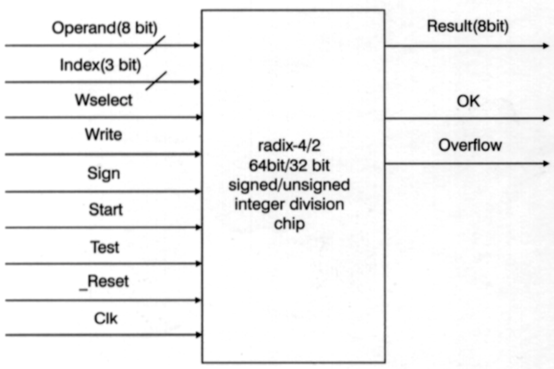

Fig. 8 *Entire pin-out of chip*
Excluding 12 power pins.

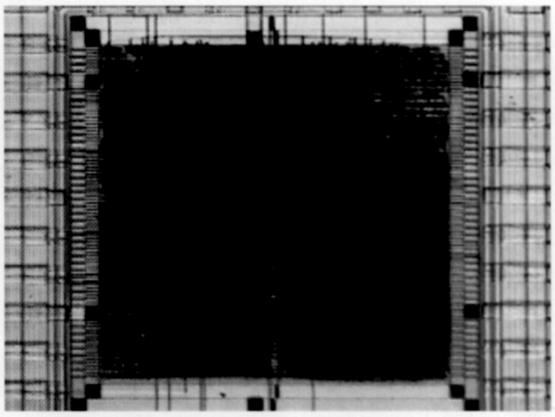

Fig. 9 *Die photo of chip*

(1 = start to execute the division), '_Reset'(asynchronous reset signal), 'OK' (1 = division is done), 'Overflow' (denotes either overflow or divided by 0), 'Clk' (clock), and 'Test' (1 = test mode, 0 = normal mode). Meanwhile, 12 power pads are required in the chip. Hence, the real chip is packaged in a 40-pin DIP. Fig. 8 shows the entire pin-out of the chip.

3.2.2. Testability design consideration: To observe the contents of internal registers should the chip malfunctions, a test mode is also taken into the chip implementation. When 'Test' is 1, bit2 to bit0 of 'Operand' are used as selection signals to dump the contents of the register, which we like to observe at 'Result'. Meanwhile, the 3-bit 'Index' signals are currently used to indicate which byte is placed at 'Result'.

3.3 Testing the real chip of 64b/32b signed/unsigned integer divider

The real 64b/32b chip has been tested by the ATS Company. IMS digital tester. The die photo of the chip is shown in Fig. 9. The chip has been tested by 1000 groups of dividend and divisor pairs, which are randomly generated, and the results of corresponding quotient and remainder pairs are all correct. The maximum operating clock for the chip is up to 50 MHz. The execution results of the testing patterns are summarised in Table 5.

Table 5: Execution of different cases of operands

Operands' cases	Execution results	Number of clock cycles
Normal case: dividend $\geqslant$ divisor, quotient is not overflow	If 'Sign' = 0 and quotient is not overflow ($\leqslant 4294967295_{10}$), quotient and remainder are both positive.	
	If 'Sign' = 1, remainder's sign is the same as that of dividend. If the signs of dividend and divisor are identical, the quotient is positive ($\leqslant 2147483647_{10}$); if different, the quotient is negative ($\geqslant -2147483647_{10}$)	7–23
Dividend $\geqslant$ divisor, quotient is overflow	If 'Sign' = 0 and quotient is overflow ($> 4294967295_{10}$), 'Overflow' = 1, and the quotient is set to $FFFFFFFF_{16}$.	
	If 'Sign' = 1 and quotient is overflow ($> 4294967295_{10}$ or $< -2147483647_{10}$), 'Overflow' = 1. If the signs of the dividend and the divisor are identical, the quotient is set to $7FFFFFFF_{16}$; if different, the quotient is set to 8000000_{16}.	5
Dividend < divisor	The quotient is 0, and the remainder is same as the dividend.	5
Divisor = 0	'Overflow' = 1. If 'Sign' = 0, the quotient is set to $FFFFFFFF_{16}$.	
	'Overflow' = 1. If 'Sign' = 1 and the dividend is positive, the quotient is set to $7FFFFFFF_{16}$. If 'Sign' = 1 and the dividend is negative, the quotient is set to 8000000_{16}.	3

4 Conclusions

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. Not only do we show the feasibility of using the radix-4/2 method, but all the implementation problems mentioned in Section 1 are also resolved. The simulation results indicate that our design is a better option than the existing long divider designs. The testing results of the real chip additionally verify the design methodology that we propose in this study. Notably, this design can be integrated in the ALU unit of a 64-bit microprocessor.

5 Acknowledgements

Our 64b/32b signed integer divider chip has been reviewed and approved by Chip Implementation Center (CIC) of National Science Council (NSC) of Taiwan to be fabricated by Taiwan Semiconductor Manufacturing Company (TSMC). The number of our chip is (CIC) T06-87C-09. We thank CIC, NSC, and TSMC. This research was partially supported by National Science Council under grant NSC 88-2219-E-110-001

6 References

1 'Pentium Pro Family Developer's Manual' (Intel, Reading, 1996)
2 GWENNAP, I.: 'Intel's P6 uses decoupled superscalar design, Microprocessor report, 1995, **9** (2), pp. 1–7
3 GWENNAP, L.: 'Klamath extends P6 family', Microprocessor report, 1997, **11**, (2), p. 1
4 BASHAGHA, A.E., and IBEAHIM, M.K.: 'Two's complement high radix division', 1997 International Symposium on *Circuits and Systems* (ISCAS'97), 1997, Hong Kong, pp. 2088–2091
5 ERCEGOVAC, M.D., and LANG, T.: 'Division and square root: digit-recurrence algorithms and implementations' (Kluwer Academic Publishers, Reading, 1994)
6 HWANG, K.: 'Computer arithmetic: principles, architectures, and designs' (John Wiley, Reading, 1979)
7 ERCEGOVAC, M.D., LANG, T., and MONTUSCHI, P.: 'Very-high radix division with prescaling and selection by rounding', *IEEE Trans.*, 1994, **C-43**, (6), pp. 909–917
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 International Symposium on *Computer Arithmetic*, 1994, Santa Monica, pp. 311–314
11 BATY, K.: 'Design ware' (Synopsys, Inc., Reading, 1996), pp. B-3 to B-12