A Radix-2 Non-Restoring 32-b/32-b Ring Divider with Asynchronous Control Scheme

Jen-Shiun Chiang, Eugene Lai and Jun-Yao Liao

Department of Electrical Engineering
Tamkang University
Tamsui,251, Taiwan, R. O. C.
E-mail chiang@ee.tku.edu.tw

Abstract

Division operation is very important in the computer system. Nowadays people use a hardware module—divider to implement the division algorithm. Conventionally synchronous techniques are applied to implement the divider. The synchronous systems always need system clock signals to trigger the system. However, the system clock of the synchronous system may cause some problems, such as clock skew, dynamic power consumption, ..., etc. Compared to synchronous systems, asynchronous circuits do not need system clock signals and thus the asynchronous system does not have the shortcomings mentioned above. Here we will propose a new asynchronous architecture for the divider. In this asynchronous scheme, the architecture is of simplicity and is very easy for the VLSI implementation. By this asynchronous architecture, we use TSMC's 0.6um SPDM process to design a 32-b/32-b radix-2 non-restoring divider and the spice simulation proves it works. The HSPICE simulation shows that it can finish a 32-b/32-b division in between 3.7ns to 160.2ns.

Key Words: Asynchronous circuits, conditional carry-selection adder, divider, radix-2 non-restoring division algorithm, synchronous circuits, VLSI.

1. Introduction

The divider is a very important device in the computer central processing unit (CPU). Basically dividers are sequential circuits. Conventionally people use synchronous design technique to design the divider (Kuo et al., 1993). In the synchronous system, we need a global clock to activate the operation of the system. The global clock scheme used in the synchronous circuits makes the design of a system be easy (Sutherland, 1989). However, there are several drawbacks in the synchronous system with global clock signals (Unger, 1995; Hauck, 1995). First, the long path of the system clock signals may cause clock skew, and that may cause very severe problems and the system may thus be malfunctioned. Second, the clock frequency of the system clock is decided by the longest period among the states. In order to increase the clock frequency, people have to do a lot of efforts to separate the states to be almost equal operation time, and this may reduce the design efficiency and increase the design time. Third, nowadays people like to use dynamic devices to design circuits (Yuan and Svensson, 1989), and the charge and discharge of the circuits consume power. The higher the frequency of the clock is, the more the power consumes. Fourth, the VLSI technology progresses significantly nowadays. The implementation processes are dependent prone, and that means the circuit works in one VLSI implementation process may not function properly in another VLSI process (Hauck, 1995). Therefore, the same circuit has to be redesigned in different implementation processes. This characteristic reduces the life time of a product.

Besides the synchronous design technique, there is another design technique —asynchronous design technique. The asynchronous circuit (Jacob

and Brobersen, 1991; Pigeut, 1991; Williams and Horowitz, 1991; Renaudin et al. 1996) is activated by the input event, therefore, there is no need of system clocks. In the asynchronous system we do not need to worry about the problems which are caused by the system clock. There are several approaches to design an asynchronous circuit (Hauck, 1995), such as bounded delay models, micropipelines (Sutherland, 1989), delay insensitive circuits, and quasi delay insensitive circuits (Molina et al. 1996). The bounded delay models are the conventional approach; we have to generate the flow table and assign the internal states very carefully to prevent the critical race. Due to the unpredictable delays we have to do a lot of effort to make the state transitions properly, therefore it is not practical to the real circuit design (Hauck, 1995). The other approaches, such as delay insensitive circuits and quasi delay insensitive circuits, are classified as handshaking mechanism approaches which are the main stream of the asynchronous design recently. Since the circuit is activated by the input event and the handshaking signals in the asynchronous circuits, so we do not need to take special care of the system clock path routing to avoid the clock skew problems. Therefore, the asynchronous designs are implementation process independent (Hauck, 1995). That means the asynchronous design can be fit to any kind of VLSI process and do not affect the function of the design. There is no system clocks in the asynchronous circuits, the power consumption which we worry about in the synchronous circuits is thus prevented.

The divider proposed in this paper is in the asynchronous manner, and there is no system clock signal to activate the division operation. We use a single stage ring to finish the radix-2 non-restoring division (Hwang, 1979; Koren, 1993), so the hardware is small. About the asynchronous controller, we use the very commonly used devices, such as exclusive-or gates and latches, and pseudo NMOS technique (Wesete and Eshraghhing, 1993) to implement this 32-b/32-b divider. In order to increase the operation speed, the conditional carry-selection adder (CCSA) (Ohkubo and Suzuki, 1995) is adopted. This divider is designed in a parallel-in serial-out manner and takes only 4,051 MOS transistors. The HSPICE simulation shows this divider works well.

This paper is arranged as follows. Section I is introduction. Section II will describe the background of the radix-2 non-restoring division algorithm. Based on Section II we propose an architecture of the 32-b/32-b divider and is expressed in Section III. The circuit design is shown in Section IV. The VLSI layout is also shown in Section IV. The simulation and certain results are shown in Section V. Finally we will give the concluding remarks which are shown in Section VI.

2. Radix-2 Non-Restoring Division

Division is the most difficult operation in the computer arithmetic. Basically the division algorithm can be classified as multiplicative and subtractive approaches. In the multiplicative approach, we try to find the multiplicative inverse to calculate the quotient. On the other hand, we subtract the divisor form the partial remainder (dividend) recursively to find the quotient and remainder. Here we will concentrate ourselves to the subtractive approach. In the subtractive idea the algorithm is the same as the division methods that we were taught in the elementary school. Suppose that there are two n-digit numbers, X and D, which represent the dividend and divisor respectively. By the division operation we can find a n-digit quotient and a n-digit remainder denoted as Q and R respectively. The mathematical representations of X, D, Q, and R are as following (Koren, 1993),

$$R^{(j+1)} = r \times R^{(j)} - q_{j+1} \times D \qquad (1)$$

Where $j = 0, 1, 2, \ldots, n-1$ is the iteration number.

R_j is the partial remainder at iteration j.

r is the radix number.

q_{j+1} is the $j+1$th digit of the quotient.

The final quotient is represented as

$$Q = q_1 q_2 q_3 \ldots q_n$$

Due to the complexity and the hardware cost, we use radix-2, i.e., r=2, for our design. Therefore, equation (1) can be rewritten and represented in equation (2) as follows (Koren, 1993).

$$R^{(j+1)} = 2 \times R^{(j)} - q_{j+1} \times D \qquad (2)$$

In the hardware design we have to check the subtraction at each step to decide the quotient in that digit. There are two ways to find the quotient of the current digit. One is the restoring method, and the other is the non-restoring method. Without loss of generality let us discuss the two methods in the radix-2 number system. In the restoring approach, when the current partial remainder,

$R^{(j+1)}$, is positive, the current quotient bit is equal to 1. On the other hand, if the current partial remainder is less than 0, then the current quotient bit is set to 0, and then the partial remainder should be added with the divisor and restore back to the previous partial remainder, $R^{(j)}$, and it is so called the "restoring" method. In the non-restoring method, if the current partial remainder, $R^{(j+1)}$, is positive, the current quotient bit is equal to 1. On the other hand, if the current partial remainder is less than 0, then the current quotient bit is set to -1, and at the next step we have to add the divisor to the current partial remainder, $R^{(j+1)}$, to form the next partial remainder, $R^{(j+2)}$. The quotient map is shown in Fig. 1. By this method, there is no need to add divisor to restore the previous partial remainder. However, the quotient in the non-restoring scheme is represented in the signed bit (digit) format. Therefore, after we finish the division process, the non-restoring method needs an additional step to convert the signed bit format to the binary number representation. Since we do not need to check the polarity of the partial remainder to do the restoring of the partial remainder. Therefore, the speed of the non-restoring division algorithm is faster than the speed of the restoring division algorithm (Hwang, 1979; Koren, 1993).

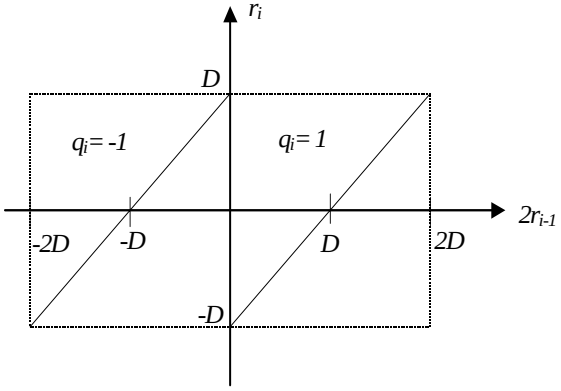

Fig. 1. The Quotient Map of Non-Restoring Division

In the radix-2 division, dividend and divisor are usually represented in the normalized format. A normalized binary number can be represented in the following format.

$$A = (0.1a_1a_2 \ldots a_n)_2$$

By the non-restoring division approach, we find the -1 of the quotient bit can be simply set to 0, and the quotient is the actual quotient that we want to find. We use a simple example to describe the normalized radix-2 non-restoring division algorithm. Suppose that we have two numbers, 011011111 and 01001, where the decimal points are abbreviated and the first bits of the dividend and divisor represent the sign bits. The division is executing as the following procedures.

Example:

011010000 / 01001 => *Q = 010111 &R = 0001*

011010000	
-) 01001	(+10111)
001000000	$q_1 = 1$
-) 01001	(+10111)
11111000	$q_2 = 0$
+) 01001	
0011100	$q_3 = 1$
-) 01001	(+10111)
001010	$q_4 = 1$
-) 01001	(+10111)
R = 0001 <= 00001	q_5 *=1 => Q = 010111*

That means in the radix-2 non-restoring division approach; we do not need to convert the signed digit quotient to the binary representation and the calculated quotient is the number that we want in the hardware design. For the benefit of the speed and hardware cost, we use normalized radix-2 non-restoring division algorithm to build our divider.

3. The Architecture of the Asynchronous Divider

The architecture of the divider is shown in Fig. 2. The basic components of this divider consists of multiplexers, counters, adders, shift registers, and latches. The operating procedures are described in next paragraph. From Fig. 2, we can find that the architecture has the characteristics of simplicity, and the hardware is small.

Suppose that the divisor and dividend are ready, we give a starting signal to start the divider. According to the non-restoring division algorithm, the multiplexer of the divisor will select the positive or negative divisor which depends on the sign of the result of the 32-b adder (partial remainder). If the partial remainder is positive the quotient bit is set to 1, and the multiplexer selects the negative divisor for the next iteration. On the other hand, the quotient bit is set to 0, and the

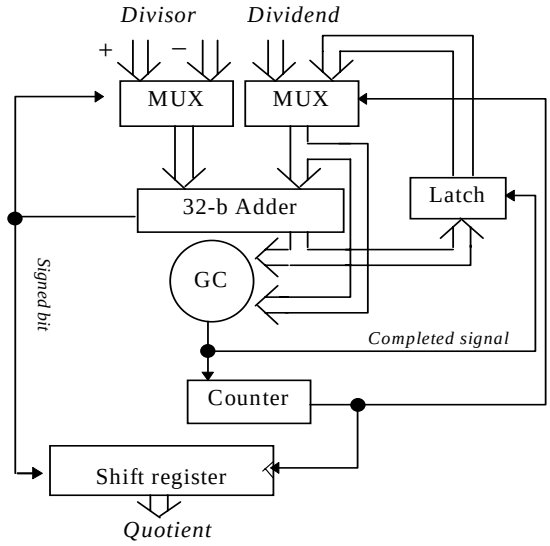

Fig. 2. The Architecture of the Divider

multiplexer will select the positive divisor for the next iteration. The GC (generate complete) gate has two sets of inputs, and each set is with 32-bit length. When both of the two sets are the same, the output of the GC gate is set to 0, otherwise, the output of the GC is set to 1. The detail circuit design of GC will be described in Section IV. The GC gate and the feedback latch together will generate a self-timed clock signal. The self-timed clock signal is used to trigger the counter and the feedback latch. Originally the output of GC is 0, as soon as the starting signal is triggered, then the output of GC becomes 1. This GC signal will activate the counter and make the counter to count 1 up. The feedback latch is designed as a level triggered latch. As soon as the counter counts, the multiplexer in the dividend side will select the result from the feedback latch. When the adder finishes summation operation, the two sets of inputs of GC are the same and cause the output of GC to be 0, and the feedback latch stops latching data. At this moment the 31st bit of the partial remainder will select either positive or negative divisor to the adder and do the addition operation. By the way, the 31st bit of the partial remainder is shifted into the MSB of the shift register to form the current bit of the quotient. As soon as the addition operation of the adder starts, the output of GC changes to 1, and the feedback latch starts latching the partial remainder, and the counter counts again. Recursively, the operation will repeat 32 times, and the counter will count 32 times and then set the ripple carry bit of the counter to 1 and the division procedure stops. Finally the quotient will be found in the output latch.

4. The Circuit Design and the Asynchronous Divider

The key components of this divider are the adder and the GC gate. In order to increase the speed of the division, the conditional carry-selection adder (CCSA) (Ohkubo and Suziki, 1995) is used. The block diagram of the CCSA is shown in Fig. 3. This adder combines the characteristics of carry look-ahead adder and conditional sum adder (Hwang, 1979). The first level of the 32-b CCSA consists of a series of 4-b CLA adders (8 sets). Each CLA adder will generate two sets of outputs, where one is without carry-in sum and carry, and the other is with carry-in sum and carry. The sums with carry or without carry will be selected by the CSS to form the exact sum. The carries of each CLA adder go into the CLA2 block to find the correct carries. Based on the correct carries, the CSS can select the right partial sums which form the CLA adders. The detail of CCSA please refer to (Ohkubo and Suziki, 1995). The speed of the CCSA is very fast but the hardware is smaller than the traditional conditional sum adder. This adder can satisfy our requirement.

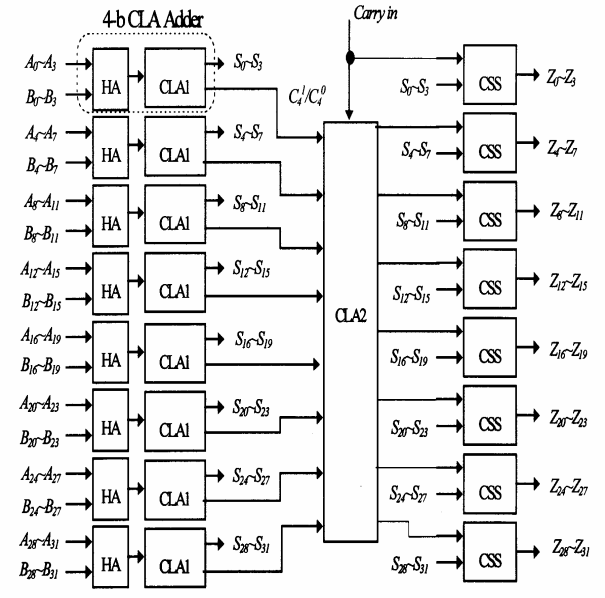

Fig. 3. The Block Diagram of the CCSA

The GC gate is designed by the pseudo

NMOS approach which is shown in Fig. 4. In the GC circuit, the exclusive-or gate is used to check the completion of the summation of the adder. One input of the exclusive-or gate is from the output of the feedback latch, and the other input is from the corresponding bit of the partial remainder. From Fig. 2 we find that the feedback latch is a level triggered latch, and the multiplexer in the dividend side selects the input from the feedback latch. When both inputs of GC are different, it means that the adder has not finished summation operation yet, and the feedback latch is still latching the output of the adder. As soon as the adder finishes summation, the exclusive-or gate of GC becomes 0 and makes the feedback latch stop latching data but hold the latched data. By the pseudo NMOS and exclusive-or gates arrangement as shown in Fig. 4, if one output of the exclusive-or gate is 1, the pseudo NMOS circuit will discharge and make the output of the GC to be 1. Only all the outputs of the exclusive-or gates are 0's, then the "output" of Fig. 4 becomes 0. This "0" signal of GC will cause the feedback latch to hold the latched data, and generate a series of self-timed clock signals.

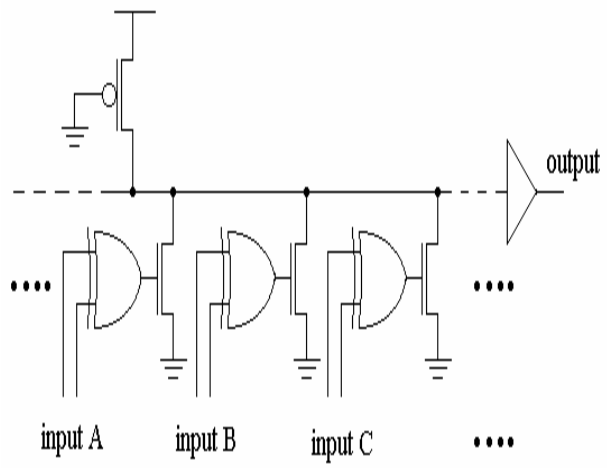

Fig. 4. The Structure of GC Gate

By the arrangement of GC and CCSA adder, the divider can work recursively. We need only one trigger signal ("starting") to tell the divider to do the division, and no external clock is needed. When the division operation is finished, the quotient is latched in the latch, and then the system stops there and idle to wait for the next sets of divisor and dividend, and the asynchronous operation mode is thus formed.

5. The Simulation and the VLSI Layout

By the architecture and the circuit design technique discussed in Section III and IV, we use TSMC's 0.6 μm SPDM process to design and implement a 32-b/32-b parallel-in serial-out divider. The HSPICE simulation of the GC gate is shown in Fig. 5. Because the speed of the addition depends on the amounts of the two inputs of the adder. In the asynchronous operation mode, the finished time for each operation should be different. From Fig. 5 we can clearly find that the period of the given cycles formed by the GC are different. However, it can work properly.

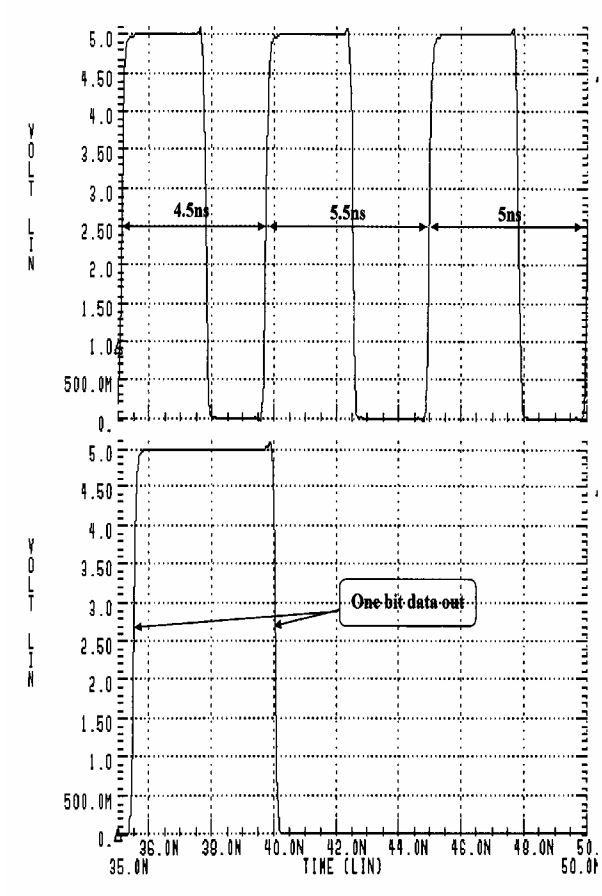

Fig. 5. The Period of Subtraction Operation

The fastest division needs only 3.7ns and the HSPICE simulation is shown in Fig. 6. If we need 32 division steps, it takes 160.2ns and the HSPICE simulation is shown in Fig. 7. The specification of this divider is shown in Table. 1. The VLSI layout of this 32-b/32-b radix-2 non-restoring divider is shown in Fig. 8.

Process	The fastest division	The complete division	Power consumption	Transistor count	Layout area
TSMC's 0.6μm SPDM	3.7ns	160.2ns	157.5mW	4051	835 μm × 840 μm

Table. 1. The Specification of the Divider

6. Conclusion

A new architecture of the 32-b/32-b radix-2 non-restoring asynchronous divider is proposed and implemented. This is a ring divider, and the asynchronous component, GC gate, is used to control the addition or subtraction procedures. The architecture of the divider is of simplicity and takes small hardware. From the simulation we find that even a very simple architecture the divider can operate asynchronously and properly. We design the divider by TSMC's 0.6μm SPDM process and this divider (32-b/32-b) is composed of 4051 MOS transistors.

Theoretically asynchronous circuits consume less power. However, it takes 157.5mW for the 32-b/32-b division. We find the reason is that pseudo NMOS architecture in the GC gate may consume most of the power. If the pseudo NMOS of the GC gate can be revised, the power consumption of this asynchronous divider may be reduced significantly.

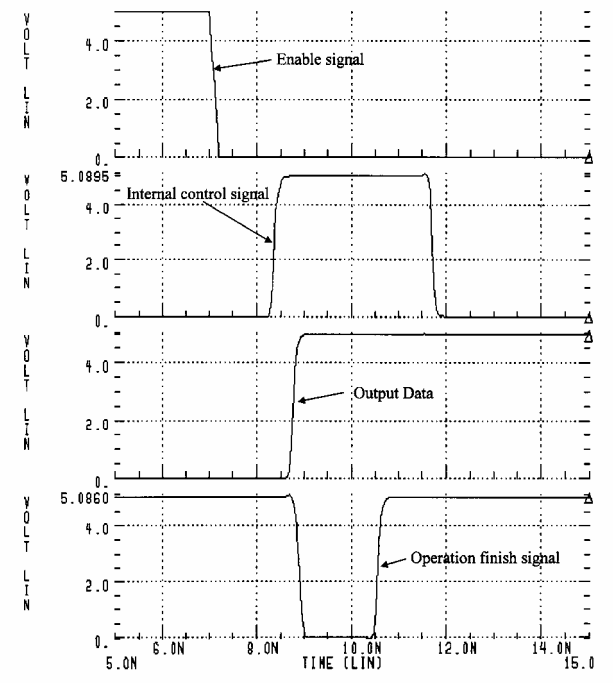

Fig. 6. The Fastest Division

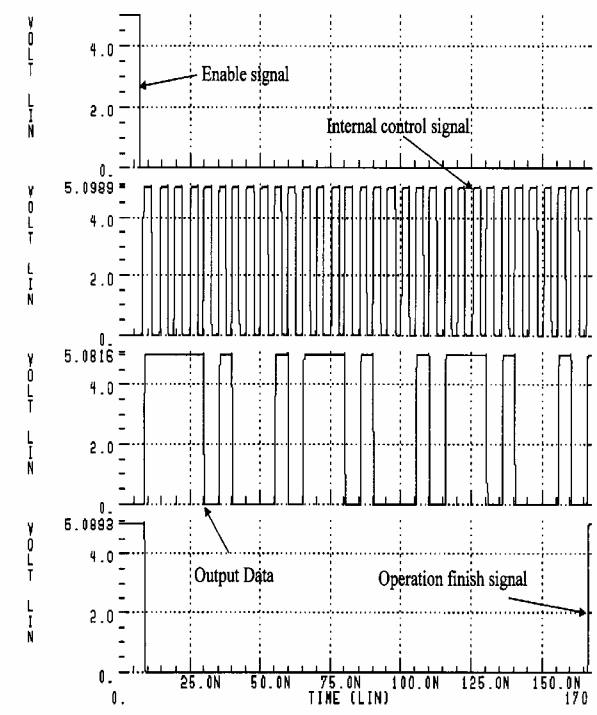

Fig. 7. Full 32 Division Steps

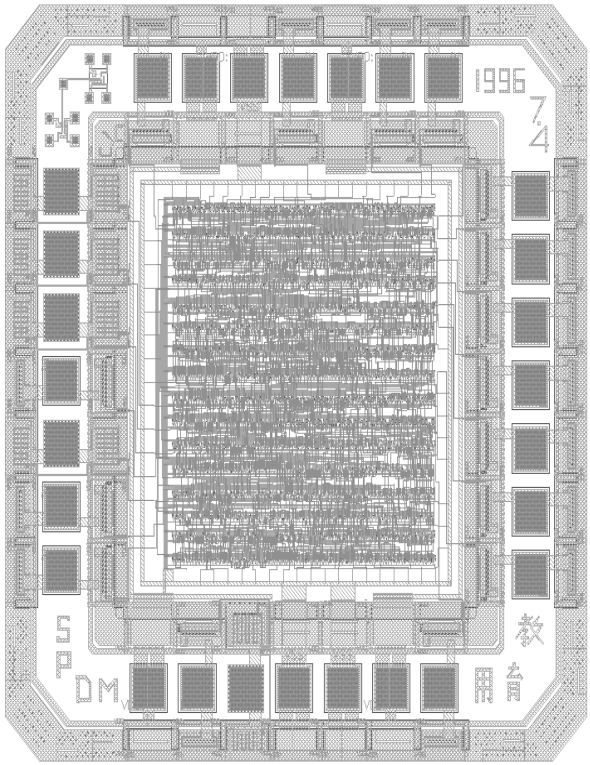

Fig. 8. The VLSI Layout of Divider

References

1. Hauck, S., "Asynchronous design methodologies: an overview," *IEEE Proceeding*, Vol. 83, No. 1, pp. 69-93 (1995).
2. Hwang, K. *Computer Arithmetic Principles, Architecture, and Design*, John Wiley & Sons Inc (1979).
3. Jacob, I. and Brobersen, R. W., "A fully asynchronous digital signal processor using self-timed circuits," *IEEE J. Solid-State Circuits*, Vol. 25, No. 11, pp. 1526-1537 (1991).
4. Koren, I. *Computer Arithmetic Algorithms*, Prentice-Hall Inc (1993).
5. Kuo, J. B., Chen, H. P. and Huang, H. J., "A BiCMOS dynamic divider circuit using a non-restoring iterative architecture with carry look ahead for CPU VLSI," *IEEE Int. Symposium on Circuits and Systems* (1993).
6. Molina, P. A., Cheung Y. K. and Bormann D. S., "Quasi delay-insensitive bus for fully asynchronous systems," pp. 189-192 (1996).
7. Ohkubo, K. and Suzuki, M., "A 4.4ns CMOS 54x54-b multiplier using pass-transistor multiplexer," *IEEE J. Solid-State Circuits*, Vol. 30, No. 3, pp. 251-256 (1995).
8. Piguet, C., "Logic synthesis of race-free asynchronous CMOS circuits," *IEEE J. Solid-State Circuits*, Vol. 26, No. 3, pp. 371-380 (1991).
9. Renaudin, M., Hassan, B. E., and Guyot, A., "A new asynchronous pipeline scheme: application to the design of a self-timed ring divider," *IEEE J. Solid-State Circuits*, Vol. 31, No. 7, pp. 1001-1013 (1996).
10. Sutherland K., "I. E. Micropiplines," *Commun. ACM*, Vol. 32, No. 5, pp. 720-736 (1989).
11. Unger, S. H., "Hazards, critical races, and metastability," *IEEE Trans. Computers*, Vol. 44, No. 6, pp. 754-768, (1995).
12. Weste, N. and Eshraghhian, K., *Principles of CMOS VLSI Design: A Systems Perspective, 2nd ed.*, Addison-Wesley Publishing Company (1993).
13. Willians, T. E. and Horowitz, M. A., "A zero-overhand self-timed 160-ns 54-b CMOS divider," *IEEE J. Solid-State Circuits*, Vol. 26, No. 11, pp. 1651-1661 (1991).
14. Yuan, J. and Svensson, C., "High-speed CMOS circuit technique," *IEEE J. Solid-State Circuits*, Vol. 24, No.1, pp. 62-70 (1989).

Manuscript Received: Apr. 12, 1999

Revision Received: Apr. 26, 1999
and Accepted: May. 31, 1999