Proc. Natl. Sci. Counc. ROC(A)
Vol. 23, No. 6, 1999. pp. 744-750

The Design and Implementation of a Pipelined Multiplier Associated with a New Pass Transistor Asynchronous Control Unit

JEN-SHIUN CHIANG AND JUN-YAO LIAO

Department of Electrical Engineering
Tamkang University
Taipei, Taiwan, R.O.C.

(Received November 23, 1998; Accepted April 12, 1999)

ABSTRACT

Due to advances in very large scale integrated circuit (VLSI) technology, the chip area of integrated circuits (IC) has increased significantly. However, power efficiency and performance have not proportionally improved. The main reason is the very high capacitive load of the global clock. On the other hand, the asynchronous circuit needs no global clock and it can overcome the difficulties that are met in synchronous circuit design. This paper describes a technique for asynchronous circuit design which uses a new asynchronous control unit that is composed of pass transistors. This asynchronous control unit has the advantages of simplicity and ease of implementation. We used the Taiwan Semiconductor Manufacturing Company (TSMC) 0.6 μm single-poly double-metal process to design and implement an 8–b×8–b pipelined multiplier associated with an asynchronous control unit. HSPICE simulation results show that the feed through rate of the inputs can be as high as 250 MHz.

Key Words: asynchronous circuit, Booth decoder, bounded delay model, clock skew, delay insensitive, micropipiline, Muller C-element, pass transistor, pipelined multiplier, quasi delay insensitive

I. Introduction

Digital sequential circuits can be classified into two categories, synchronous circuits and asynchronous circuits. In synchronous circuits, a system clock signal is needed to force the states to transition correctly and properly. On the other hand, asynchronous circuits do not have system clock signals; state transitions are caused by changes in inputs (Hauck, 1995). The internal states of synchronous circuits are controlled by clock triggered flip-flops; therefore, synchronous circuits do not have state race problems (Unger, 1995). In traditional synchronous circuits, blocks of logic circuits are activated by a globally distributed clock signal. As the chip area increases, the route length of the global clock becomes very long; hence, the capacitor load becomes very significant, which may cause clock skew. Clock skew may degrade the performance of the system. In order to increase the speed of a circuit, dynamic circuits are used to implement the circuit. However, the procedures involved in charging and discharging a dynamic circuit may consume power; the higher the clock frequency, the more power is consumed (Hauck, 1995).

The asynchronous circuit does not have the problems encountered in the synchronous circuit. The events of asynchronous circuits are activated by input signals instead of the global clock (Hauck, 1995). Therefore, we do not need to worry about problems caused by the global clock. Since there is no global clock, there is no dynamic power consumption, and theoretically, asynchronous circuits consume less power than synchronous circuits (Hauck, 1995). Since activation of asynchronous circuits is triggered by input signals, an asynchronous system has “average-case” performance instead of the “worst-case” performance in a synchronous system. Therefore, an asynchronous system potentially has higher speed than a synchronous system. In synchronous circuits, in order to make the system work properly, delay elements have to be added (Hauck, 1995). Because the very large scale integrated circuit (VLSI) process technology is dependent prone, parameters of delay elements varies in different VLSI processes. Therefore, a given product may have to be redesigned to fit a new VLSI process. This characteristic may increase the design and implementation complexity, and may even shorten the life cycle of the product. On the other hand, the design methodologies

for asynchronous circuits are the same in all of the VLSI processes. Therefore, asynchronous circuits can be easily changed to a new VLSI process (Hauck, 1995).

Although asynchronous circuits have good characteristics, they also have their drawbacks. The most important one is in the design of the control scheme. In order for asynchronous circuits to work properly, asynchronous controllers must be embedded into the circuits. Therefore, the asynchronous control unit is the core of the asynchronous circuit design. Several approaches to this problem have been proposed by researchers (Hauck, 1995; Sutherland, 1989; Myers and Meng, 1993; Meng *et al*., 1991; Cho *et al*., 1992; Dean, 1992; Furber *et al*., 1993; Furber, 1993; Martin *et al*., 1994; Paver, 1994; Richardson and Brunvand, 1995; Burns, 1991). However, some of them are vary complicated, and some are impractical for implementation in a real circuit (Hauck, 1995; Burns, 1991).

Asynchronous circuits can be roughly classified into several categories (Hauck, 1995), including bounded delay models, micropipelines (Sutherland, 1989; Furber *et al*., 1993), self timed circuits (Dean, 1992; Richardson and Brunvand, 1995), delay insensitive circuits, and quasi delay insensitive circuits. The bounded delay models are used in the conventional approach. When designing an asynchronous circuit in the bounded delay model, we have to generate the flow table and assign states very carefully to prevent critical races. Due to unpredictable delays, much effort is required. Because of these uncertainties, basically, the bounded delay approach is not practical for real circuit design (Hauck, 1995). The other approaches, including micropipelines, self timed circuits, delay insensitive circuits, and quasi delay insensitive circuits, are classified as handshaking mechanism approaches and are commonly used in asynchronous circuit design (Cho *et al*., 1992; Dean, 1992; Furber *et al*., 1993; Furber, 1993; Martin *et al*., 1994; Paver, 1994; Richardson and Brunvand, 1995).

In synchronous circuits, a global clock signal is used to coordinate signals in between stages. Similarly, a handshaking mechanism is used in an asynchronous circuit to coordinate signals in between stages by using control signals (like "request" and "acknowledge") instead of the system clock signal. The handshaking mechanism is depicted in Fig. 1, which shows only three stages: stage $i-1$, stage i, and stage $i+1$. Activation of the control signal in stage $i-1$ causes the data (signal) of stage $i-1$ to flow to stage i. Completion of the data transition of stage i activates the control signal of stage i. Activation of the control signal of stage i deactivates the control signal of the previous stage (stage $i-1$), and causes the data (signal) of stage

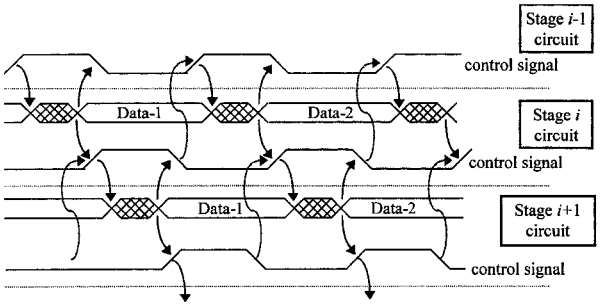

Fig. 1. The handshaking scheme used in asynchronous circuits.

i to flow to stage $i+1$. The data flow of stage $i+1$ may activate the control signal of stage $i-1$, and completion of the data transition of stage $i+1$ activates the control signal of stage $i+1$. Therefore, if we can insert a handshaking mechanism block in each stage of the digital system, the system can transition to the right state and work properly.

A new asynchronous control unit similar to handshaking is proposed in this paper. It has a simple architecture and can be implemented in real VLSI circuits easily. Using this new control unit, we have designed and implemented a pipelined multiplier. This pipelined multiplier verifies the usefulness of the proposed asynchronous control unit.

This paper is arranged as follows. Section I is the introduction. In Section II, the novel asynchronous control unit and the asynchronous pipelined scheme are presented. Section III discusses the architecture of the multiplier. In Section IV, we discuss the design of an 8–b×8–b asynchronous pipelined multiplier. Finally, we conclude this paper in Section V.

II. New Control Unit and the Structure of Asynchronous Circuits

The control unit is the core of an asynchronous circuit design. Recently, asynchronous circuit design has used event logic to control the system (Hauck, 1995; Sutherland, 1989). Muller C-element uses the approach of event logic (Sutherland, 1989). This control unit is used to control the changes of handshaking signals in order to enable or disable the function of internal logic blocks. The Muller C-element is useful, but it is not practical (Hauck, 1995). For practical design of asynchronous circuits, we propose another architecture for the asynchronous control unit.

The basic structure of an asynchronous circuit is composed of several stages. Except for the first and last stages, each stage is connected by two stages, the preceding stage and the succeeding stage. There is a control block and an executing logic block in each

	GC0	Φ1	GC1	Φ2	GC2	Φ3	GC3	Φ4	GC4	Φ5	· · ·
start	0	0	0	0	0	0	0	0	0	0	· · ·
step1	1	0	0	0	0	0	0	0	0	0	· · ·
step2	1	0	1	0	0	0	0	0	0	0	· · ·
step3*	0	1	1	0	1	0	0	0	0	0	· · ·
step4	1	0	0	1	1	0	1	0	0	0	· · ·
step5	1	0	1	0	0	1	1	0	1	0	· · ·
step6	0	1	1	0	1	0	0	0	1	0	· · ·
.	.	.	.	.	.	.	.	.	.	.	· · ·

Fig. 2. State flow of the asynchronous control cheme for the normal case.

stage. In order to find the proper control signals, we use differential logic, such as complementary pass logic (CPL) (Bellaouar and Elmasry, 1995), to implement the circuits. The executing logic block is composed of latches and differential logic. Similar to the protocol mentioned in Fig. 1, we use a "generate complete" (GC) signal to represent the control signal. The executing logic block will generate a GC signal to indicate that the function generated in that stage has finished. Our proposed control unit tries to use the characteristics of the GC signal. In our asynchronous system (as shown in Fig. 6), the GC signal is sent to both the previous stage and the succeeding stage to indicate that the job has finished in the current stage. When the previous stage receives the GC signal (GC_i), this signal acts like an acknowledge signal in that it tells the previous stage that the current stage has finished the job. The previous stage can then send new data to the current stage. When the succeeding stage receives the GC_i signal, this GC_i signal acts like a request signal in that it tells the succeeding stage that the current stage has finished the job, and that the data are on the way to the succeeding stage, and it also activates the executing logic of the succeeding stage. Therefore, a good handshaking protocol (in which GC_i is both an acknowledge signal sent to the previous stage and a request signal sent to the next stage) is formed and functions normally and smoothly.

Let us use a symbolic graph to describe the function of GC_i. Figure 2 shows several stages of an asynchronous circuit. GC_i and Φ_i are the handshaking signal "generate complete" and the reset signal for the ith stage, respectively. As long as Φ_i equals 1, GC_{i-1} is reset. At the very beginning, the system is reset, and all the GC_j's and Φ_i's are zeroes (start step). Suppose that GC_0 is activated, and that after some delay time, stage 0 finishes executing and sends the complete signal GC_1 to stage 1 (step 1). After some delay time, stage 1 finishes its logic operation, and the finished state activates Φ_1 and GC_2 (step 2). The activation of Φ_1 thus resets GC_0 (step 3). After Φ_1 becomes 0, GC_0 can again accept additional data or signals from the external circuits (step 4). At the same time, GC_2 continues working, propagating signals and data to the following stages (step 4). Stage by stage, the data or signals are propagated to the final stage, and the job is finished. If the second piece of data is fed into the system with some delay time, our architecture can work properly. The symbolic graph for this kind of situation is shown in Fig. 3. In step 3 of Fig. 3, data input stops. However, we find that the process continues working, propagating to stage 3 in step 4. Finally, the input data are fed into the system in step 5. The previous process continues on to the next stage regardless of the present data input. On the other hand, if one of the stages takes more time than some other stages, this architecture can still work. Figure 4 shows an example. Suppose stage 2 needs more time to execute data. In step 3 of Fig. 4, we find that GC_2 has not been activated, and that the system stops there to wait for the execution of stage 2. In step 4, stage 2 finally finishes executing, and GC_2 is activated; then the system continues operating, and the next batch of data is fed into the system. From Figs. 2-4, we find that the system can coordinate well and work smoothly.

Based on the protocol mentioned in the previous paragraph, a control unit has been designed and is shown in Fig. 5. In Fig. 5, GC_i is the "generate complete" signal of the current stage, GC_{i-1} is the "generate complete" signal from the previous stage, and GC_{i+1} is the "generate complete" signal in the following stage. This control unit can operate the same protocol mentioned in the previous paragraph. GC_{i-1}, GC_i, and GC_{i+1} all together activate the "enable" signal to trigger

	GC0	Φ1	GC1	Φ2	GC2	Φ3	GC3	Φ4	GC4	Φ5	· · ·
.	.	.	.	.	.	.	.	.	.	.	· · ·
Step 3*	0	1	1	0	1	0	0	0	0	0	· · ·
Step 4	0	0	0	1	1	0	1	0	0	0	· · ·
Step 5	1	0	0	0	0	1	1	0	1	0	· · ·
Step 6	1	0	1	0	0	0	0	1	1	0	· · ·
.	.	.	.	.	.	.	.	.	.	.	· · ·

Fig. 3. State flow of the asynchronous control scheme for discontinuous data feed.

	GC0	Φ1	GC1	Φ2	GC2	Φ3	GC3	Φ4	GC4	Φ5	· · ·
.	.	.	.	.	.	.	.	.	.	.	· · ·
Step 3*	0	1	1	0	0	0	0	0	0	0	· · ·
Step 4	0	1	0	0	1	0	0	0	0	0	· · ·
Step 5	1	0	0	1	1	0	1	0	0	0	· · ·
.	.	.	.	.	.	.	.	.	.	.	· · ·

Fig. 4. State flow of the asynchronous control scheme for slower stage 2.

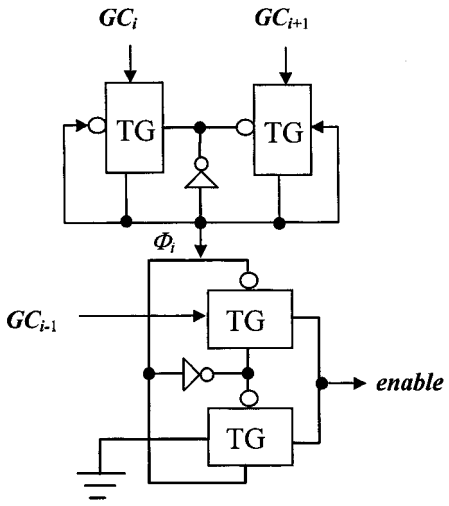

Fig. 5. The new asynchronous control unit.

the latches in the current stage. The architecture of the asynchronous control unit is very simple. The components of this control unit consist of only transmission gates and inverters, and the unit can be implemented very easily.

We will next discuss in more detail the function of this control unit that is embedded into asynchronous circuits. Figure 6 shows part of an asynchronous circuit. In each stage, there are two multiplexers which form the control unit. The circle on the upper multiplexer indicates the GC_{i+1} complement. The logic part is implemented using differential logic circuits. Initially, both of the outputs of the differential logic circuits are reset to '0' (or '1'). The GC circuit consists of a two-input exclusive-or gate, and both of the outputs of the differential logic circuits are fed to the inputs of the GC circuit. When the logic function finishes, the outputs of the differential logic generate two exclusive outputs that set GC to 1 and thus generate the complete signal. Initially, we set all the GC circuits to 0 and reset the enable signal. As soon as GC_{i-1} is set to '1', stage *i* is enabled, the latch in the logic block latches the data from stage *i*–1, and at the same time, both of the differential outputs are reset to '0' (or '1'). After the logic block finishes its operation, the GC becomes '1', forces the upper multiplexer to select $\overline{GC_{i+1}}$, forces stage *i*+1 to accept data from stage *i* and activates stage *i*+1. Only when stage *i*+1 finishes its operation can it be activated by stage *i*–1. Based on this handshaking mechanism, the asynchronous circuits operate stage by stage and function normally. Based on this characteristic, the asynchronous unit can be applied to a pipelined system. We will describe a real circuit architecture which uses this control unit in the next section.

III. The Architecture of the Asynchronous Pipelined Multiplier

Because asynchronous circuits work stage by stage, they are very similar to the pipelined architecture. In the pipelined architecture, latches are needed to separate the stages. The latches in the asynchronous pipelined scheme have two important functions. One is to latch data for the current state, which is similar to the conventional synchronous mode. The other function is to modify the wave form of the signal. Here, we use CPL to implement the logic block of asynchronous circuits. Due to the characteristics of CPL, the output signal is not full swing, and non-full-swing signals may be restored by the latches to full swing wave forms. The block diagram of an asynchronous circuit with latches is shown in Fig. 7. From Fig. 7, we find that the GC signal is detected from the latch. In order to perform this function, the latch is designed in a complex mode, composed of dynamic and static latches, and the schematic diagram is shown in Fig. 8. The dual outputs of the CPL are fed into the dynamic latch to form the executing complete signal. When the CPL logic part has finished executing, the dual signals are latched by the dynamic latch and activate GC to 1. The GC signal can then be propagated to both the previous stage and the following stage. The GC signal sent to the previous stage will enable the system to

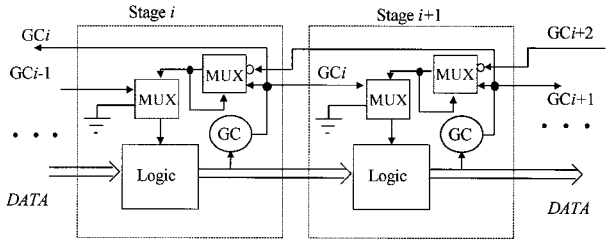

Fig. 6. The architecture of a system with asynchronous control units.

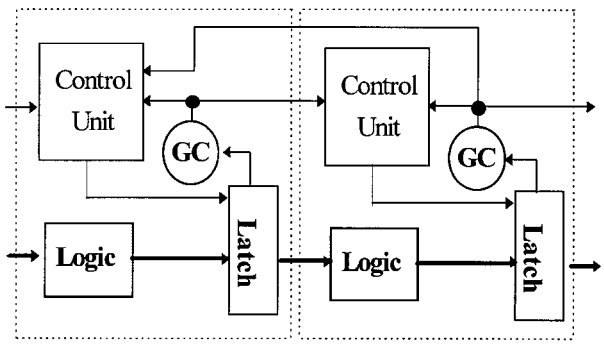

Fig. 7. Pipelined architecture of asynchronous circuits.

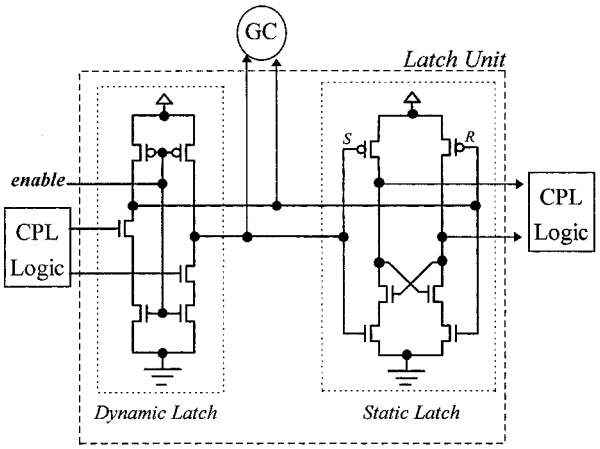

Fig. 8. The complex latch.

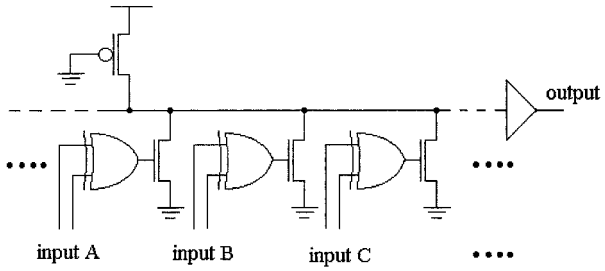

Fig. 9. The GC Circuits.

receive new data from the previous stage or the external world. The GC signal sent to the following stage can make the following stage ready to receive data from the current stage. When the GC signal of the current stage has been transmitted, GC itself should be clear, and the current stage can then receive a new set of data from the previous stage again. The dynamic latch can then automatically reset GC. The static latch is a set-reset (SR) latch. This SR latch can statically latch data to make sure that the next stage will have correct and stable data. If the fan-in is more than one, we can use the architecture shown in Fig. 9 to implement the GC signal.

Based on the design described in the previous paragraph, a pipelined multiplier has been designed. The architecture of the multiplier is shown in Fig. 10. The Booth algorithm (Koren, 1993) is applied to this multiplier. This multiplier has several components, including a Booth decoder, several 4-2 compressors (West and Eshraghhian, 1993), and a conditional-sum adder (Hwang, 1979). There are many partial products in a multiplier, and the Booth decoder can reduce the number of product terms and, thus, reduce the hardware cost. The 4-2 compressors can sum up to four partial product terms concurrently. Using the 4-2 compressor approach can prevent carries to propagate to the higher position and can increase the speed of the multiplier. Finally the conditional-sum adder can sum the result of the final two partial product terms to obtain the real multiplication product. The conditional-sum adder is very suitable for the pipelined architecture (Hwang, 1979); therefore, we used this architecture to design and implement the pipelined multiplier in order to demonstrate the validity of the proposed asynchronous architecture. The architecture shown in Fig. 10 can be fit to a pipelined multiplier of any length. Without loss of generality, we have designed an 8–b×8–b multiplier.

IV. Circuit Design and Implementation

The multiplication procedures for the multiplier are divided into ten stages to fit the pipelined architecture. Here, the Booth decoder is divided into two stages, the 4-2 compressors are separated into two stages, and the conditional-sum adder is implemented in five stages. We have used the CPL circuit to design and implement each stage. Since the design is asynchronous, we do not need to worry about the clock frequency. The multiplier keeps working as long as there are data in the multiplicand and multiplier. Based on this design arrangement, during every 4 ns, the system can feed another set of data to the multiplier.

The test chip was fabricated using Taiwan Semiconductor Manufacturing Company (TSMC) 0.6 μm single-poly-double-metal complementary metal-oxcide-

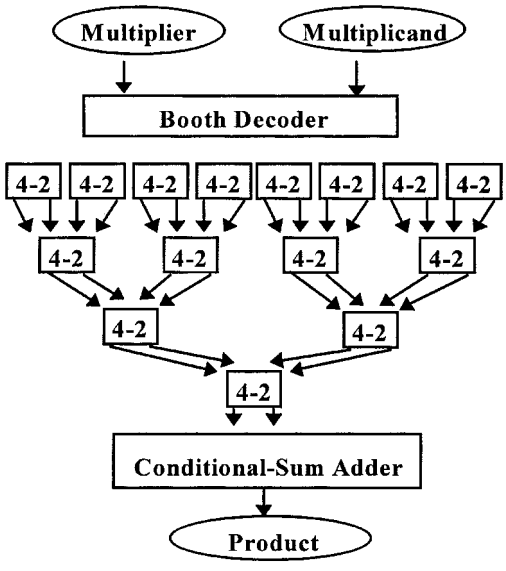

Fig. 10. Block diagram of the multiplier.

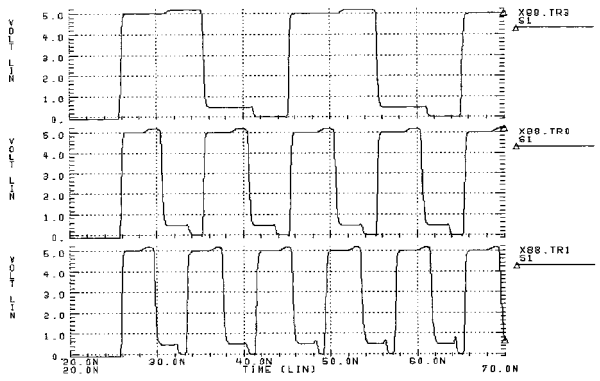

Fig. 11. HSPICE simulation results obtained under input data rates of 100/200/250 MHz.

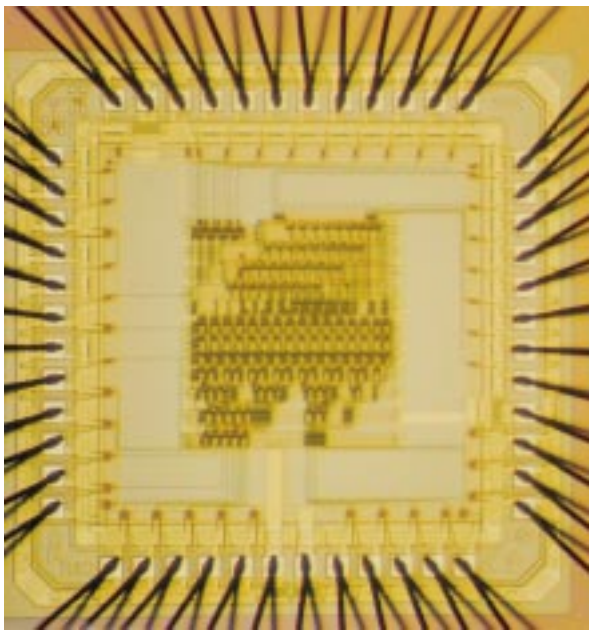

Fig. 12. The photograph of the die of the multiplier.

semiconductor field effect transistor (SPDM CMOS) technology. We used the fully custom design method to complete the circuit layout. The die size was about 970 μm×1048 μm, and 6428 transistors were used. The multiplication speed of the asynchronous pipelined multiplier can operate to a clock frequency of 250 MHz. Figure 11 shows the HSPICE simulation results for 100 MHz/200 MHz/250 MHz input data rates. The speed and power dispassion simulation results for the multiplier are summarized in Table 1. A photograph of the die of the multiplier is shown in Fig. 12.

V. Conclusions

In this paper, we have proposed an asynchronous controller with a simple architecture which is composed of transmission gates (pass transistors) and inverters. Based on this approach, the asynchronous circuit can obtain good performance. We have also tested the validity of the control unit by designing and implementing a pipelined multiplier. This pipelined multiplier can feed data every 4 ns and work very well. The design of the multiplier is based on the module scheme, and it can be expanded to any bit size. Besides the pipelined scheme, the asynchronous control unit can be applied to any kind of digital system.

Table 1. The Simulation Results

Frequency	Delay	Power
100 MHz	16.4 ns	43.75 mW
200 MHz	16.3 ns	65.35 mW
250 MHz	16.1 ns	73.5 mW

Acknowledgment

This work was supported by the National Science Council of the Republic of China under grant NSC 87-2215-E-032-004.

References

Bellaouar, A. and M. I. Elmasry (1995) *Low-Power Digital VLSI Design Circuits and Systems*. Kluwer Academic Publishers, New York, NY, U.S.A.

Burns, S. (1991) *Performance Analysis and Optimization of Asynchronous Circuits*. Ph.D. Dissertation. California Institute of Technology, Pasadena, CA, U.S.A.

Cho, K., K. Okura, and K. Asada (1992) Design of a 32-bit fully asynchronous microprocessor (FAM). 35th Midwest Symposium on Circuits and Systems, Washington, D.C., U.S.A.

Dean, M. E. (1992) *STRIP: a Self-timed RISC Processor*. Technical Report CSL-TR-92-543, Computer Systems Laboratory, Stanford University, Stanford, CA, U.S.A.

Furber, S. (1993) Computing without clocks. VII Banff Workshop: Asynchronous Hardware Design, Banff, Canada.

Furber, S., P. Day, J. D. Garside, N. C. Paver, and J. V. Woods (1993) A micropipelined ARM. VII Banff Workshop: Asynchronous Hardware Design, Banff, Canada.

Hauck, S. (1995) Asynchronous design methodologies: an overview. *IEEE Proceeding*, **83**, 69-93.

Hwang, K. (1979) *Computer Arithmetic Principles, Architecture, and Design*. John Wiley & Sons Inc., New York, NY, U.S.A.

Martin, A., S. Burns, T. Lee, D. Borkovic, and P. Hazewindus (1994) The design of an asynchronous microprocessor. CalTech Conference on VLSI, Pasadena, CA, U.S.A.

Meng, T. H., B. W. Brodersen, and D. G. Messerschmitt (1991) Asynchronous design for programmable digital signal processors. *IEEE Trans. on ASSP*, **39**, 939-952.

Myers, C. and T. H. Meng (1993) Synthesis of timed asynchronous circuits. *IEEE Trans. on VLSl Systems*, **1**, 106-119.

Paver, N. C. (1994) *The Design and Implementation of an Asynchronous Microprocessor*. Ph.D. Dissertation. University of Manchester, Manchester, U.K.

Richardson, W. F. and E. Brunvand (1995) Precise exception handling for a self-timed processor. 1995 International Conference on Computer Design: VLSI in Computers & Processors, Los Alamitos, CA, U.S.A.

Sutherland, I. E. (1989) Micropiplines. *Commun. ACM*, **32**, 720-736.

Unger, S. H. (1995) Hazards, critical races, and metastability, *IEEE Trans. Computers*, **44**, 754-768.

使用傳輸閘之新型非同步控制器與管線式乘法器之設計與實現

江正雄　廖俊堯

淡江大學電機工程學系

摘　要

隨著超大型積體電路技術的進步，IC是愈做愈大，但其功率消耗與效能卻未能有效改善，究其原因，主要是因為系統時脈所造成。傳統的數位電路都是屬於同步電路系統，在同步電路系統中，需要一個系統時脈來驅動整個系統，若時脈太慢則系統效果不彰，若時脈太快則會消耗過多能量，因此系統時脈常常是數位系統設計上的一個瓶頸。在數位系統設計上還有另一種設計方式，不用系統時脈來驅動系統，而只靠系統輸入信號的改變來驅動系統，也就是所謂的「非同步電路」，因為沒有系統時脈，所以也就沒有動態功率消耗，也克服了同步電路的難處。本論文主要就非同步電路作探討，提出一簡單實用使用傳輸閘的非同步控制單元，並將它應用在管線式乘法器上。經超大型積體電路設計（TSMC 0.6 *m*m SPDM製程）、模擬驗證、及實測，此非同步控制單元運作良好，所設計的管線式乘法器的頻率模擬可高達250 MHz。