A Quasi-Delay-Insensitive Microprocessor Core Implementation for Microcontrollers

CHANG-JIU CHEN, WEI-MIN CHENG, HUNG-YUE TSAI AND JEN-CHIEH WU
Department of Computer Science
National Chiao Tung University
Hsinchu, 300 Taiwan

Microcontrollers are widely used on simple systems; thus, how to keep them operating with high robustness and low power consumption are the two most important issues. It is widely known that asynchronous circuit is the best solution to address these two issues at the same time. However, it's not very easy to realize asynchronous circuit and certainly very hard to model processors with asynchronous pipeline. That's why most processors are implemented with synchronous circuit. There are several ways to model asynchronous pipeline. The most famous of all is the micropipeline; in addition, most micropipeline based asynchronous systems are implemented with single-rail bundled-delay model. However, we implemented our 8-bit microprocessor core for asynchronous microcontrollers with an alternative – the Muller pipeline. We implemented our microprocessor core with dual-rail quasi-delay-insensitive model with Verilog gate-level design. The instruction set for the target microprocessor core is compatible with PIC18. The correctness was verified with ModelSim software, and the gate-level design was synthesized into Altera Cyclone FPGA. In fact, the model we used in this paper can be applied to implement other simple microprocessor core without much difficulty.

Keywords: asynchronous circuit, microcontroller, micropipeline, dual-rail, Muller pipeline, quasi-delay-insensitive

1. INTRODUCTION

It is widely known that synchronous circuits have some problems that have to be carefully dealt with such as clock skew problem, difficulty in clock distribution, worse case performance, not modular, sensitive to variations in physical parameters (temperature, voltage, and process), synchronization failure, and noise (EMI). All these problems derive from the "clock" signal [1]! As the VLSI based systems become larger, more complex, and work with higher clock rate, these problems also become more serious than ever before. However, because of several complex historical and practical reasons, almost all systems today are still implemented with fixed clock period based design. While synchronous design may introduce lots of problems with systems growing up larger and larger, asynchronous design may overcome these problems via avoiding the use of clock signal. Furthermore, how to accomplish IP reuse easier becomes one of the most important issues for SoC design. Asynchronous circuits may be one of the best solutions to address this issue. Without the influence of the "clock" signal, asynchronous circuits make software OOP style design for hardware design possible. All things that the designers need to know are the handshaking protocol interface [1]. It also makes each designed component or IP more reusable. With growing up mobile device and embedded

Received May 10, 2007; revised November 6, 2007 & March 17, 2008; accepted July 17, 2008.
Communicated by Yao-Wen Chang.

system markets, all these issues need to be seriously considered. Thus, it's time to implement these systems with asynchronous circuits.

Microcontrollers are widely used on a variety of different simple systems. Most of these systems do not need very complex processor core. Instead of very complex processor core, most of them needs processor core with some other special requirements such as adaptation to operating variations, low power consumption, and low EMI. Asynchronous circuits naturally meet these requirements. That's why we focus on designing processor core for microcontrollers with asynchronous circuits. 8051, AVR, and MicroChip PIC family microcontrollers are all popular 8-bit microcontrollers for embedded systems. There are several asynchronous 8051 compatible microprocessor implemented [2, 3]. However, because of its CISC nature, it's not very easy to implement it with pipeline directly. Thus, in this paper, we implemented our microprocessor core with PIC18 instruction set with quasi-delay-insensitive model. Though we implemented our processor core with PIC18 instruction set, we hope that our research could still be a generic model to implement such processor core for these simple microcontrollers.

2. RELATED WORKS

Asynchronous circuits have been studied since early 1950's; however, synchronous circuits have still dominated the mainstream of digital circuit design. Recently, some academic and commercial research shows that it's worth to implement real-life systems with asynchronous circuits. But, because of lack of tools and standardization of implementation and design models, there is still not much research on it and just limited commercial applications.

Without clock signal, asynchronous circuits rely on handshaking protocols to make sure the correctness of the circuit operations. The protocols can be divided into control signaling and data encoding. Fig. 1 shows the 4-phase handshaking protocol. In this protocol, only the rising edge is the valid active transition; thus it's a level signaling or return-to-zero protocol. On the contrary, in the 2-phase handshaking protocol, the falling and rising edge of request and acknowledge are active signals; thus it's a transition signaling or non-return-to-zero protocol. However, it makes the control very complex and hard to implement. Fig. 2 shows the 2-phase handshaking protocol. Except control signaling, there are also choices for how to encode data (data signaling protocol). The Bundled Data or called Single Rail refers to separate request and acknowledge wires that bundles the data signals with them. Thus total $n + 2$ wires are required to send n-bit data. Fig. 3 shows the bundled-data model. Except bundled-data model, there are data encoding methods for DI circuits. However, because of implementation issue, dual-rail encoding is the most popular used DI data encoding scheme. To represent 1-bit data in dual-rail encoding method, two physical wires are used. For example, a valid data, D is represented by two physical data wires, d.0 and d.1. The following equation shows this encoding scheme. (1) $D = 0$; $(d.0, d.1) = (0, 1)$ (2) $D = 1$; $(d.0, d.1) = (1, 0)$. In particular, (0, 0) represents a space which allows us to identify consecutive 0's or 1's. (1, 1) state is not used. Data transferring starts from the (0, 0) state (called "null" or "empty" data). If a state is changed from $(d.0, d.1) = (0, 0)$ to (0, l)/(1, 0), which notices the arrival of valid data '0/l'. Thus total $2 \times n$ wires are needed to transfer n-bit data. Fig. 4 shows the dual-rail model [1].

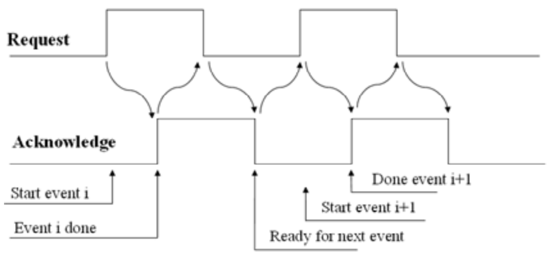


Fig. 1. The 4-phase protocol.

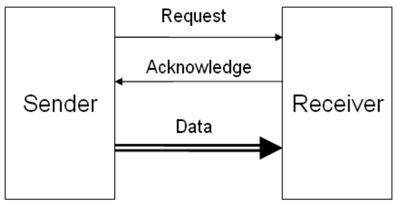


Fig. 3. Bundled-data signaling model.

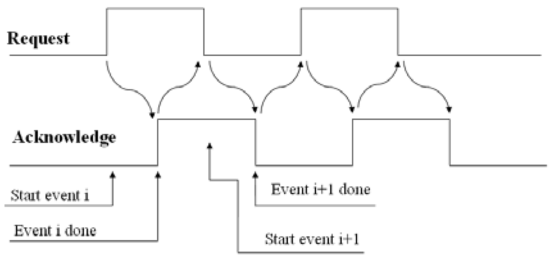


Fig. 2. The 2-phase protocol.

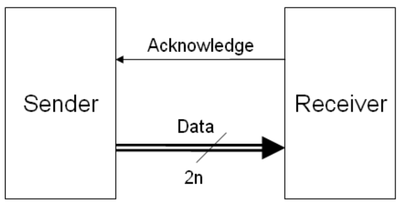
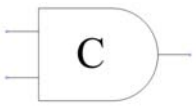


Fig. 4. Dual-rail data signaling model.

David Muller proposed his famous Muller *C*-element and Muller pipeline (aka Muller distributor) in 1959 [4, 5]. A Muller pipeline is a naturally simple and elegant handshaking control model. The simplest form of Muller pipeline mainly consists of *C*-elements and inverters. Fig. 5 shows the schematic symbol and truth table of a two-input *C*-element. If both inputs are high or low, the output will be high or low; otherwise, the previous value is kept. Fig. 6 shows the original Muller pipeline model. To understand its behavior, let's consider the *i*th *C*-element C_i. In the initial state, all *C*-elements are initialized to 0. The handshaking may be initialized. The *i*th *C*-element C_i can propagate a 1 from its previous stage the (i – 1)th *C*-element only if the next stage C-element (C_{i+1}) is 0. Thus, the signal can be propagated one stage to one stage. It should be notice that the original single-rail model is based on bundled-data model; thus the request signal must be propagated via a matching delay as shown is Fig. 6. In fact, the matching issue should be carefully handled on all bundled-data model. The pipeline model can also be constructed as 4-phase dual-rail model as shown in Fig. 7 [6]. The model can be considered as two Muller pipelines connected in parallel with a common acknowledge signal in per stage. The detailed behavior described in section 3.1.



a	**b**	**y**
0	0	0
0	1	No change
1	0	No change
1	1	1

Fig. 5. The muller *C*-element: symbol & truth table.

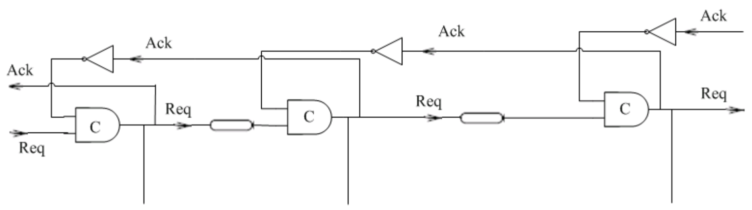


Fig. 6. The Muller pipeline.

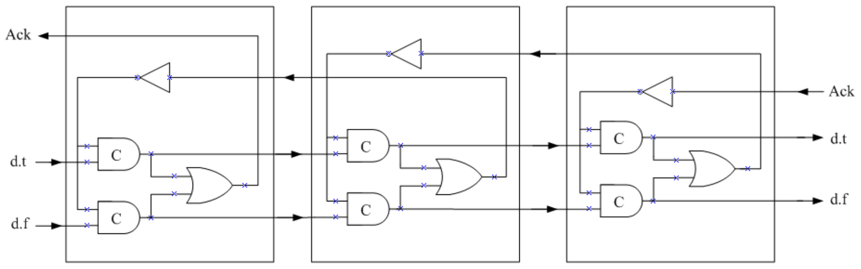


Fig. 7. A three-stage 1-bit wide 4-phase dual-rail pipeline.

Except the Muller pipeline, there are also several models were proposed. The most important of all is the micropipeline which was described by Ivan E. Sutherland is his famous Turing Award “Micropipelines” lecture in 1989 [7]. The approach is based on a two-phase bundled-data model with micropipeline as backbone control circuit. As the most well-known asynchronous circuit design model, lots of different asynchronous systems have been implemented based on it. It can be used to implement many kinds of different pipelined systems, even processors. For example, the NSR processor is a very simple 16-bit micropipeline based microprocessor with very simple RISC instructions (less than 20 instructions) [8]. In addition, the most famous of all is the Amulet series processors [9-12]. These processors are ARM compatible processors implemented with micropipeline architecture.

There are also some different models proposed for asynchronous processor design. Some try to modify the original “micropipeline” architecture. For example, a new control circuit for micropipeline was proposed by Choy *et al.* [13] and “Micronets” architecture tries to decentralize the control to the functional units [14]. Furthermore, there have been still several famous asynchronous processor implementation models proposed. Takashi Nanya *et al.* showed their QDI 8-bit microprocessor model called “TITAC” which uses Martin’s *Q*-element as control circuitry [15]. TITAC2 was proposed to show a new delay model called scalable-delay-insensitive (SDI) [16]. The delay model modified original DI or QDI unbounded gate and wire delay to bounded relative delay ratio between any two components. There are also some works that try to model processor with asynchronous circuits. Martin *et al.* in Caltech have already shown three generations of different asynchronous processor model [17]. Chen *et al.* showed an asynchronous RISC processor model in 2002 [18]. In addition, there are also several asynchronous superscalar processor models proposed, for example the Kin architecture [19], Hades project [20], and the most famous of all the counter flow pipeline (CFPP) [21]. However, all these superscalar models are not very easy to implement or just not realized ideas, and certainly not very suitable to be implemented for cores for simple microcontrollers.

3. NCTUAC18

There have been several asynchronous FIFO pipeline models proposed. However, most of them are based on bundled-data model, especially micropipeline or related models. But it is widely known that the DI circuit has the highest robustness and reliability. Moreover, almost all the research focuses on the original FIFO model, not to real system implementation. Thus, in this paper, we tried to implement a real microcontroller with QDI models. We implemented our asynchronous microprocessor core with 4-phase dual-rail pipeline model as shown in Fig. 7. However, because it's too hard to realize real DI circuits, we selected the QDI model to implement our asynchronous microprocessor core. In this section, the detailed implementation will be described.

3.1 The 4-Phase Dual-Rail Pipeline Model

The NCTUAC18 was implemented with 4-phase dual-rail pipeline model. Fig. 7 shows a 3-stage 1-bit wide 4-phase dual-rail pipeline model with completion detection [6]. An *N*-bit pipeline can be constructed via connecting *N* such pipelines in parallel with corresponding *N*-bit completion detector. Because of dual-rail nature, the behavior of four-phase dual-rail pipeline in Fig. 7 is very simple. Once the pair of *C*-element in one pipeline stage receives the empty (null) codeword {*d.t*, *d.f*} = {0,0}, it stores the empty code word and sends the acknowledge signal '0' out to its previous stage; otherwise, once the pair of *C*-element receives any one of the two valid codewords {*d.t*, *d.f*} = {0, 1} or {1, 0}, it stores the valid codeword and sends the acknowledge signal '0' out to its previous stage. Fig. 8 shows how the data items (*D#*) can pass through in a 4-stage 4-phase dual-rail pipeline. At time 0, all pipeline stages are initialized to null state and the first data item *D*0 are prepared to enter the pipeline. Then, at time *T*1, the first data item D0 with valid codeword enters the first stage and then the acknowledge signal can be generated to notify its left side environment to prepare for the next null codeword. At time *T*2, the data item *D*0 enters the second pipeline stage and the acknowledge signal is generated to inform the first stage to accept the null codeword. Then, the null codeword can enter the first pipeline stage. With 4-phase dual-rail protocol, the first data item *D*0

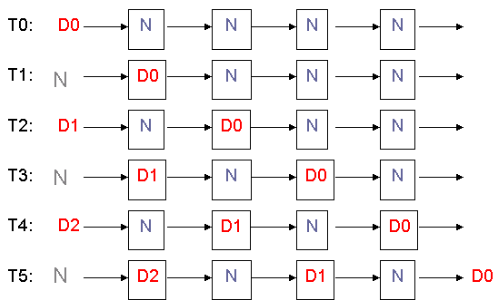


Fig. 8. Abstract view of data flow passing through 4-phase dual-rail pipeline.

can pass through the 4-stage pipeline at time *T*5. It clearly shows that any two pipeline stages with valid state are separated by a stage with null (*N*) state. We solved the data hazard problem in our pipeline design with this characteristic.

3.2 The NCTUAC18 Pipeline Model

The NCTUAC18 pipeline is based on the four-phase dual-rail pipeline model. It makes the NCTUAC18 is a core with high modularity, robustness, and reliability. The system block diagram of NCTUAC18 is showed in Fig. 9. Because of Muller pipeline model nature, the utilization of our 4-stage pipeline is 50%. Though it's not too hard to improve it to 100% via adding extra pipeline latches for each stage, we still implemented the original 50% model. There are several reasons for this selection. The first, with this 50% pipeline model, the OF stage and EX/WB stage cannot be executed simultaneously and thus data hazard problem can be easily resolved without very complex design. Second, this is just a microprocessor core for simple microcontroller, and thus simple pipeline model can reduce extra costs coming from extra pipeline latches and very complex control circuits. Finally, with this simple pipeline model, the QDI constraints can be easily kept in real processor design.

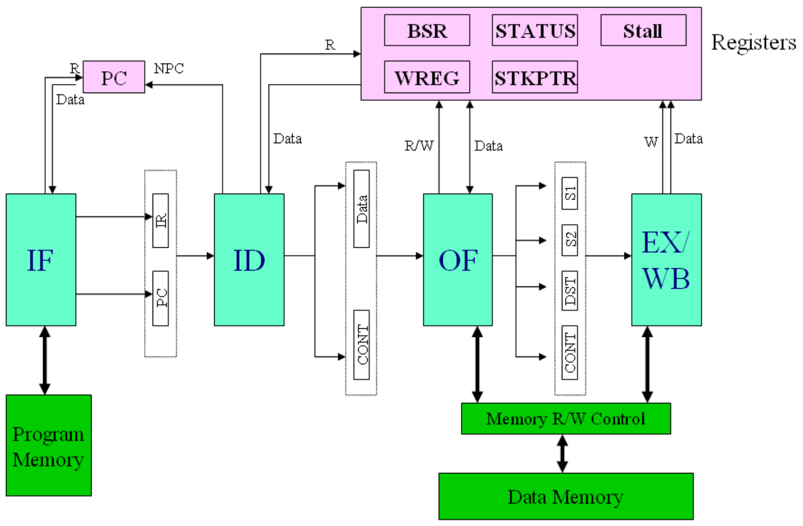
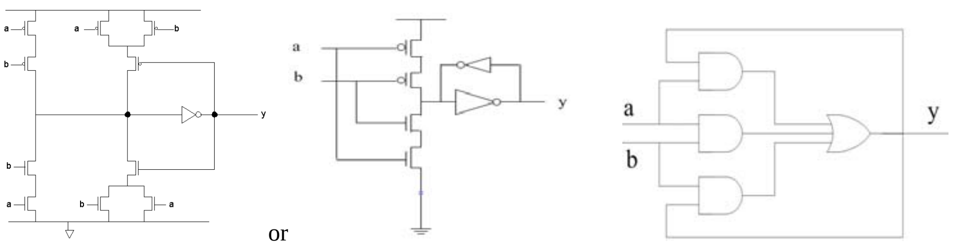


Fig. 9. NCTUAC18 system block diagram.

In order to reduce memory access time, we implemented the most frequently PIC18 registers – *PC* (Program Counter), *BSR* (Bank Select Register), *WREG* (Working Register), *STATUS*, *STKPTR* (Stack Pointer) and *Stall Register* defined by us. There are four stages in NCTUAC18 pipeline namely IF (Instruction Fetch), ID (Instruction Decode), OF (Operand Fetch) and EX/WB (Execution and Write Back). Following sections detailed describe our implementation.



(a) (b)

Fig. 10. (a) Generalized transistor-level *C*-element implementation; (b) Gate-level *C*-element implementation.

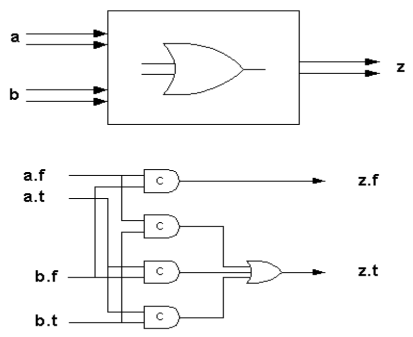


Fig. 11. Dual-rail OR gate symbol and gate-level implementation.

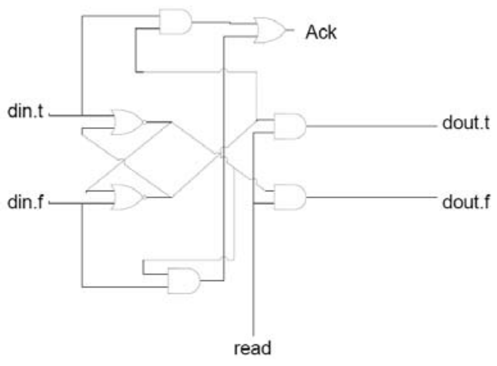


Fig. 12. 1-bit dual-rail register.

3.2.1 Basic building component construction

In order to implement our processor with dual-rail QDI model, the basic dual-rail QDI basic building components should be constructed first. We implemented all needed basic QDI dual-rail gates and constructed all building blocks with these QDI dual-rail gates. The most important of all is the *C*-element. The generalized transistor-level *C*-element implementation is shown in Fig. 10 (a); however, to provide synthesizable model for FPGA we also modeled it with gate-level design as shown in Fig. 10 (b). In addition, we also implemented *C*-element with reset for pipeline latch.

With *C*-element, other basic dual-rail components can also be constructed easily, for example dual-rail OR gate as shown in Fig. 11.

To satisfy the QDI constraint, the NCTUAC18 was implemented without any common bus. Data transfers from one block to the register can only use a direct connection. To meet our design requirement, we developed our own QDI register set. Fig. 12 shows a 1-bit dual-rail register. When a valid codeword is sent to *din.t* and *din.f*, the two NOR gates can correctly hold it. If the data item is written into the register, it issues acknowledgement to its previous stage to inform the written operation done. Because of the dual-rail nature, we designed the acknowledge signal simply through ORing two in/out signal of the two NOR gates. To read data from the register, just send read request signal to it and thus the dual-rail data can be correctly read out via *dout.t* and *dout.f*. We implemented this design for NCTUAC18 registers. Our register design does not deliver much higher cost than traditional register for synchronous systems.

Because lack of synthesis tool the design cannot be written in RTL model. Thus, the whole circuit should be carefully written in gate-level design. If the asynchronous design should be implemented with CMOS VLSI, some components had better to be created with full-custom design. For example, to implement efficient CMOS *C*-element, manually designed *C*-element cell is needed. Except modeling it with transistor-level, we also modeled it with gate-level as shown in Figs. 10 (a) and (b). Thus, it can be synthesized with Altera Quartus II software. In addition, because of DI nature, all components should be constructed carefully with DI model. Thus, the implementation cost is very high. The circuit should be optimized manually. Then, the constructed components can be easily put in that FIFO pipeline model.

3.2.2 Design of each pipeline stage

With those building blocks and components, the whole processor core can be built with them. In addition, the PIC18 compatible instruction set was implemented in our core. Table 1 shows the implemented instructions. Following section detailed describes the NCTUAC18 pipeline.

Table 1. Implemented instructions of NCTUAC18 core.

Operation	Byte-Oriented	Bit-Oriented	Literal	Control
Instruction	ADDWF	BCF	ADDLW	BC
	ADDWFC	BSF	SUBLW	BNC
	ANDWF	BTG	MULLW	BN
	ANDWF		MOVLB	BNN
	COMF		MOVLW	BOV
	DECF		IORLW	BNOV
	INCF		ANDLW	BZ
	IORWF		XORLW	BNZ
	MOVF			BRA
	MOVWF			GOTO
	MULWF			CALL
	NEGF			RETURN
	RLCF			PUSH
	RLNCF			POP
	RRCF			RCALL
	RRNCF			NOP
	SETF			
	SUBFWB			
	SUBWF			
	SUBWFB			
	XORWF			
	MOVFF			

IF stage:

In the IF stage, the “Read” signal controls the output for valid data or null data. When the *Read* signal is high, it reads PC value from the PC register and then retrieves the instruction from the program ROM. In addition, the current PC value is sent to ID stage for calculating the next PC value. After the output data has completely sent to the pipeline latch, the pipeline latch returns an acknowledgement signal. The signal is used to pull down the *Read* signal low, and finally the IF stage becomes null state.

ID stage:

It is widely known that dealing with conditional branches is very hard on pipeline processors; moreover, that’s especially much harder on such asynchronous pipeline without centralized control. Thus we develop our own model to deal with conditional branches. Because the NCTUAC18 pipeline is not very deep and the target processor core is very simple, the design should not be very complex. We believe that this model can be easily applied to all simple processor core which is implemented with these 4-phase dual-rail QDI asynchronous pipeline. In order to deal with conditional branch instructions simply, the conditional branch instructions are treated as two instruction cycle instructions. The conditional branch instructions are handled in the ID stage. The ID stage mainly consists of four parts. Fig. 13 shows the block diagram of the ID stage. The operations of the four parts are described in the following.

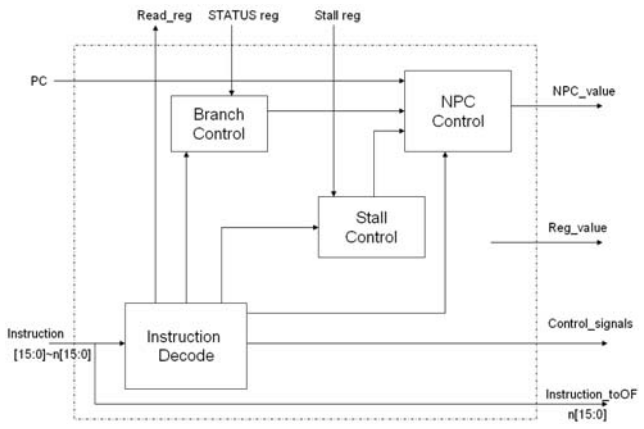


Fig. 13. Block diagram of ID stage.

***Instruction Decode*:** The *Instruction Decode* block decodes the input instructions and generates the control signals to the whole processor. It also requests *NPC* to generate next PC value directly if the input instruction is not a conditional branch.

***Branch Control*:** If the current input instruction is a conditional branch instruction, the *Instruction Decode* requests the *Branch Control* to deal with it. The *Branch Control* reads the value in the *STATUS* register to decide whether the branch is a taken or non-taken branch and then request the *NPC* to generate corresponding next PC value.

Stall Control**:** If the instruction is a conditional branch instruction and it is executed the first time, the *Stall Control* will request the NPC to generate the same PC value again in order to retrieve this instruction again. The mechanism can guarantee that the *STATUS* register can be correctly updated by the EX/WB stage.

NPC Control**:** The *NPC Control* is responsible to generate the correct next PC value corresponding to the input control signals.

OF Stage:

The OF stage is responsible for preparing source data and destination information for the EX/WB stage. In addition, because registers are implemented as memory addresses or in memory in the original PIC18 but WREG & several frequently used registers are implemented as real registers in our core, a remapping mechanism is implemented in the OF stage.

EX/WB Stage:

The EX/WB is the final stage of our pipeline and is responsible for computation and saving the result back in according to the input operands and control signals from the OF stage. It's an important design issue why EXE and WB are combined in one stage. That's because not all instructions need to pass through both EXE and WB. For example, "MOV" instruction only writes result back without any extra computation needed. Fig. 14 is the block diagram of EX/WB Stage. Because of the dual-rail nature, it's possible to design functional units that can complete each of its operations in variable length of time. We developed our own model that makes it possible to design a stage that can execute in variable length of time depending upon characteristics of the executing operation. Fig. 15 shows the concept of our proposed model. In Fig. 15, the block called "DeMUX" receives inputs from its previous stage and sends the inputs to the corresponding execution element or just even a bypass path depending on the control signal. Then, the block called "MERGE" sends the final result out into the next stage. In fact, all the *MERGE* circuit needs to do is to check if the result is valid via completion detection. Because of dual-rail nature, that's very easy to accomplish this goal. We implemented this model in all sub-stages of EX/WB stage. Though currently we only implemented this model in our

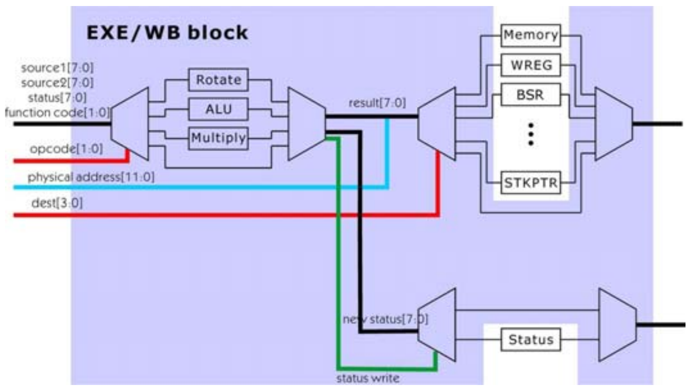


Fig. 14. Block diagram of EX/WB stage.

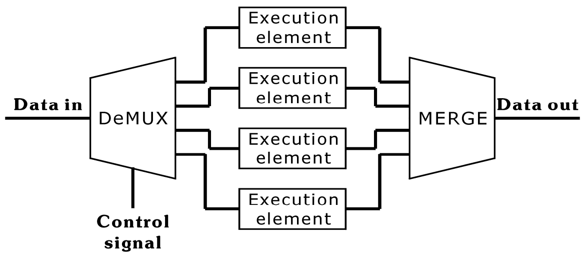


Fig. 15. The original concept of our EX/WB stage implementation model.

EX/WB stage, the proposed model can be implemented in all dual-rail circuits with several execution paths! With this "DeMUX-MERGE pair" model, the EX/WB stage may exploit data-dependent operations easily. Some operations of each of all the three sub-stages may even be directly bypassed without waste of time. It should be pointed out that the STATUS register will be updated in parallel with the WB sub-stage.

4. PERFORMANCE EVALUATION

In order to verify the correctness of the design, the ModelSim 6.0 was used to verify the correctness of the functionality. Because our current core design is so simple, we verified our design with random patterns and simple programs. Though it may be not enough for more complex design, it may be enough for our current design. In fact, circuit verification for the asynchronous system is very hard. It may be still a very good research topic. Furthermore, we also tried to synthesis our gate-level design with Altera Quartus II software. In order to fit the whole design, we selected Altera Cyclone EP1C20F400C8 as the target FPGA. Table 2 shows the maximum path delay for each stage. It's very interesting to find that the ID stage is the critical stage of the whole design. That's because the ID stage not only has to decode instructions but also to deal with the branch instructions. The most important of all is that our core use a not very efficiency method to deal with branch instructions. Furthermore, we may also find out that without global clock each stage can work in different speed. In fact, the path delay of each stage may vary with different operations. That's quite different from synchronous circuits which must work in worse case delay. That also means that you may replace any parts of the circuits with faster design without considering the "clock" issue. The DI/QDI circuit will automatically work fine with the new designed part.

Table 2. Maximum path delay of each stage.

Module	Maximum Path Delay (ns)
Latch	~ 34
IF stage	~ 27
ID stage	~ 455
OF stage	~ 157
EX/WB stage	~ 216
Register	~ 19

Another important implementation issue is area. In order to compare how much extra area needed in dual-rail implementation, we compared our design with AE18 CPU core which is an open source PIC18 compatible core implementation [22]. We synthesized both design with Altera Quartus II. Both of the target FPGAs are Altera Cyclone EP1C20F400C8. Table 3 shows the total logic elements needed of the both two designs. The results show that the NCTUAC18 core needs about 3.5 times extra LEs than AE18 core. That's because the NCTUAC18 core is a QDI implementation. However, it's an implementation tradeoff. It depends upon your implementation criteria!

Table 3. Comparison of LEs needed of both NCTUAC18 and AE18 core.

	NCTUAC18 Core	AE18 Core
Logic Elements	~ 13,800	~ 3,900

5. CONCLUSIONS AND DISCUSSIONS

Though there are several proposed asynchronous pipeline models, most asynchronous processors are still implemented with micropipeline or modified micropipeline models. That's not only because it's a Turing Award Lecture but also the implementation cost consideration. In addition, because of dual-rail nature and higher timing constraints coming from DI and QDI circuits, it's very hard to implement microprocessor core with DI or QDI circuits. Thus, most DI or ODI pipeline models are seldom used to implement microprocessors. However, it is widely known that DI circuit has the highest reliability and it is suitable to implement microcontrollers that may operate in variable environments. In addition, it does not need to consider the matching delay issue that may be encountered in implementation with bundled-data circuit such as micropipeline model. In this paper, we provide a methodology to model a QDI PIC18 compatible microprocessor core with dual-rail 4-phase pipeline in a reasonable cost. Though we just modeled PIC18 compatible core, the model can also be used on other simple microprocessor core. In fact, we show a clear flow to design a QDI microprocessor core for simple microcontroller with Verilog HDL and an easy implementation model.

Except the pipeline model itself, conditional branch handling is a very important design issue for pipeline processors. Furthermore, it's much harder for asynchronous processors without centralized control. In this paper, we show an easy way to deal with conditional branches for the dual-rail 4-phase pipeline microcontroller core. Though it's very simple, it's enough for simple microcontroller core with such short pipeline.

We also propose a special bypass or variable execution path model as shown in Figs. 14 and 15. It is implemented in EX/WB stage. Though it looks like a traditional implementation, each path can execute in variable length of time because of its asynchronous circuit nature. This proposed model really exploits data-dependent operations.

Furthermore, we also evaluate the maximum delay time of each stage and extra costs that may be introduced. Though QDI or even most asynchronous circuits may cause much extra cost, we still have to point out that it's also a design tradeoff. In fact, it depends upon your requirements. Asynchronous circuits may be a good solution for designing mobile devices or systems that needs high reliability, low power and low EMI.

Our work may provide a microprocessor core implementation model for these requirements. Based on current work, we'll focus on providing synthesizable 32-bit general purpose processor with asynchronous circuits in the future.

REFERENCES

1. A. Davis and S. M. Nowick, "An introduction to asynchronous circuit design," Technical Report No. UUCS-97-013, Computer Science Department, University of Utah, 1997.
2. J. H. Lee, W. C. Lee, and K. R. Cho, "A novel asynchronous pipeline architecture for CISC type embedded controller – A8051," in *Proceedings of the 45th Midwest Symposium on Circuits and Systems*, Vol. 2, 2002, pp. 675-678.
3. A. J. Martin, M. Nyström, K. Papadantonakis, P. I. Pénzes, P. Prakash, C. G. Wong, J. Chang, K. S. Ko, B. Lee, E. Ou, J. Pugh, E. V. Talvala, J. T. Tong, and A. Tura, "The Lutonium: A sub-nanojoule asynchronous 8051 microcontroller," in *Proceedings of the 9th International Symposium on Asynchronous Circuits and Systems*, 2003, pp. 14-23.
4. D. Muller and W. Bartky, "A theory of asynchronous circuits," in *Proceedings of International Symposium on the Theory of Switching*, 1959, pp. 204-243.
5. J. Gunawardena, "A generalized event structure for the Muller unfolding of a safe net," in *Proceedings of the 4th International Conference on Concurrency Theory*, 1993, pp. 278-292.
6. J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design – A Systems Prospective*, Kluwer Academic Publishers, London, 2001, pp. 11-25.
7. I.E. Sutherland, "Micropipelines," Turing Award Lecture, *Communications of the ACM*, Vol. 32, 1989, pp. 720-738.
8. E. Brunvand, "The NSR processor," in *Proceedings of the 26th Hawaii International Conference on System Sciences*, 1993, pp. 428-435.
9. J. V. Woods, P. Day, S. B. Furber, J. D. Garside, N. C. Paver, and S. Temple, "AMULET1: An asynchronous ARM microprocessor," *IEEE Transactions on Computers*, Vol. 46, 1997, pp. 385-398.
10. S. B. Furber, J. D. Garside, S. Temple, J. Liu, P. Day, and N. C. Paver, "AMULET2e: An asynchronous embedded controller," in *Proceedings of the 3rd International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 1997, pp. 290-299.
11. S. B. Furber, J. D. Garside, P. Riocreux, S. Temple, P. Day, J. Liu, and N. C. Paver, "AMULET2e: An asynchronous embedded controller," *Proceedings of the IEEE*, Vol. 87, 1999, pp. 243-256.
12. S. B. Furber, D. A. Edwards, and J. D. Garside, "AMULET3: A 100 MIPS asynchronous embedded processor," in *Proceedings of the International Conference on Computer Design*, 2000, pp. 329-334.
13. C. S. Choy, J. Butas, J. Povazanic, and C. F. Chan, "A new control circuit for asynchronous micropipelines," *IEEE Transactions on Computers*, Vol. 50, 2001, pp. 992-997.
14. D. K. Arvind, R. D. Mullins, and V. E. F. Rebello, "Micronets: A model for decen-

tralising control in asynchronous processor architectures," in *Proceedings of the 2nd Working Conference on Asynchronous Design Methodologies*, 1995, pp. 190-199.
15. T. Nanya, Y. Ueno, H. Kagotani, M. Kuwako, and A. Takmura, "TITAC: Design of a quasi-delay-insensitive microprocessor," *IEEE Design and Test of Computer*, 1994, pp. 50-63.
16. A. Takamura, M. Kuwako, M. Imai, T. Fujii, M. Ozawa, I. Fukasaku, Y. Ueno, and T. Nanya, "TITAC-2: A 32-bit asynchronous microprocessor based on scalable-delay-insensitive model," in *Proceedings of the International Conference on Computer Design*, 1997, pp. 288-294.
17. A. J. Martin, M. Nyström, and C. G. Wong, "Three generations of asynchronous microprocessors," *IEEE Design and Test of Computers*, 2003, pp. 9-17.
18. C. J. Chen, C. C. Shiu, and M. S. Wu, "The design of asynchronous processor," in *Proceedings of International Computer Symposium*, 2002.
19. R. Kol and R. Ginosar, "Kin: A high performance asynchronous processor architecture," in *Proceedings of the 12th International Conference on Supercomputing*, 1998, pp. 433-440.
20. C. J. Elston, D. B. Christianson, P. A. Findlay, and G. B. Steven, "Hades − An asynchronous superscalar processor," in *Proceedings of IEE Colloquium on Design and Test of Asynchronous Systems*, 1996, pp. 10/1-10/6.
21. R. F. Sproull, I. E. Sutherland, and C. E. Molnar, "The counterflow pipeline processor architecture," *IEEE Design and Test of Computers*, 1994, pp. 48-59.
22. S. Tan, "AE18 CPU CORE," http://www.opencores.org/projects/ae18, 2003.

Chang-Jiu Chen (陳昌居) received his Ph.D. degree in Computer Science from University of Oklahoma, in 1993, and his B.S. and M.S. degrees in Computer Engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1980 and 1986, respectively. He has been an Associate Processor of Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, since 1994. His research interests include asynchronous circuit design, computer architecture, digital system, microprocessor, and parallel processing.

Wei-Min Cheng (鄭緯民) received his B.S. and M.S. degrees in Computer Science from Tatung Institute of Technology, Taipei, Taiwan, in 1995 and 1997, respectively. He is currently a Ph.D. student in Computer Science in National Chiao Tung University, Hsinchu, Taiwan. His research interests include computer architecture, asynchronous system design, microprocessor and SoC.

Hung-Yue Tsai (蔡宏岳) received his B.S. degree in Computer Science from National Central University, Chungli, Taoyuan, Taiwan, in 2004, and M.S. degree in Computer Science from National Chiao Tung University, Hsinchu, Taiwan, in 2007. He is currently a Ph.D. student in Computer Science in National Chiao Tung University, Hsinchu, Taiwan. His research interests include asynchronous circuit & system design, SoC, VLSI design automation, network security, and computer architecture.

Jen-Chieh Wu (巫仁傑) received his M.S. degree in Computer Science from National Chiao Tung University, Hsinchu, Taiwan, in 2006. He is currently an R&D engineer in Wistron Corporation, Taipei, Taiwan. His research interests include VLSI de- sign, asynchronous circuit design, microprocessor, and SoC.