# A Quasi-Delay-Insensitive Microprocessor Core Implementation for Microcontrollers

Alka Pandey, Arfa Quamar, Apoorva Arya

*Abstract*- **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 micro pipeline; in addition, most micro pipeline based asynchronous systems are implemented with single-rail bundled delay model.**

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

## II. 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. 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. 1 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.2 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 dualrail 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. 3 shows the dual-rail model [1].

Fig.1. The 2-phase protocol.

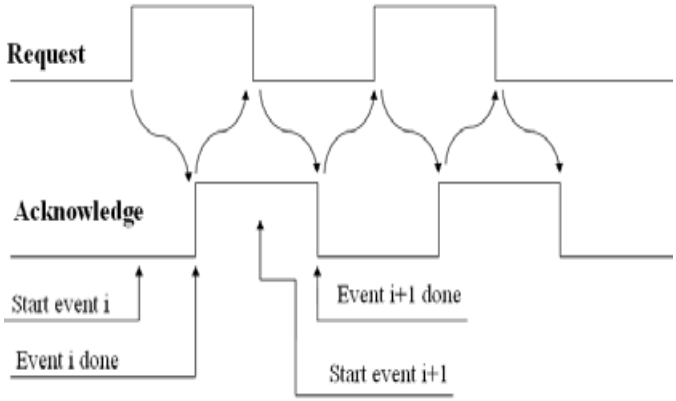



Fig.2. Bundled-data signaling model

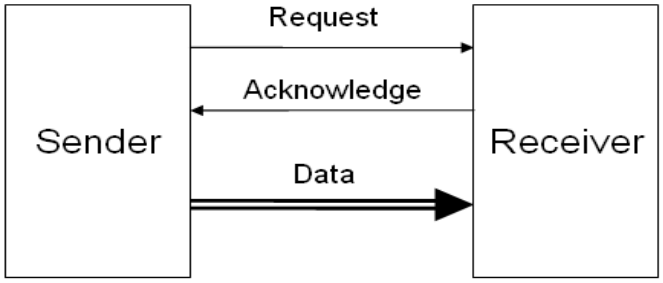
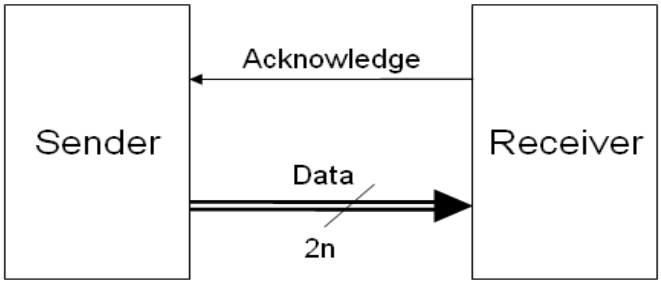
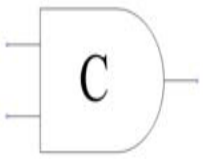



Fig. 3. Dual-rail data signaling model. .




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 *Ci*. In the initial state, all *C*-elements are initialized to 0. The handshaking may be initialized. The *i*th *C*-element *Ci* can propagate a 1 from its previous stage the $(i-1)$th *C*-element only if the next stage C-element ($Ci$+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.

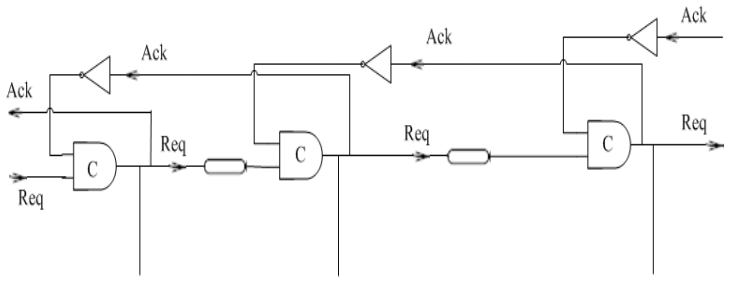



Fig. 6. The Muller pipeline.

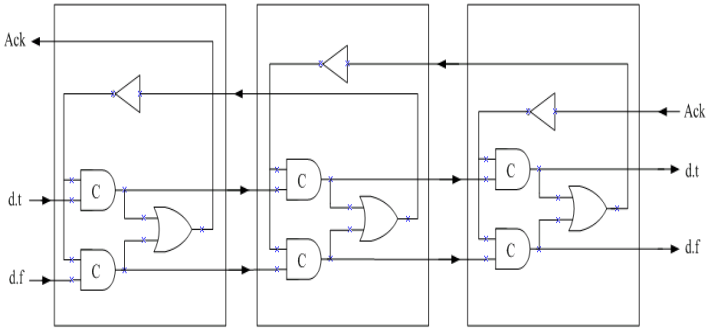



Fig. 7. A three-stage 1-bit wide 4-phase dual-rail pipeline.

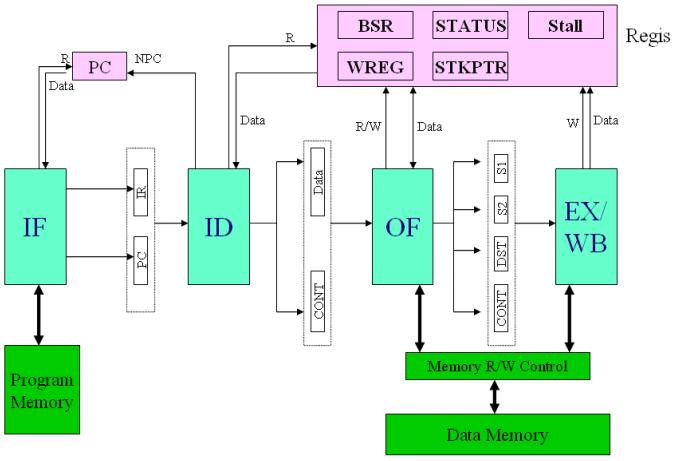
## III. NCTUAC18

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 dualrail 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.

## IV. 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.




**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.

**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 | | | |

## V. CONCLUSION

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.

ACKNOWLEDGMENT

We hereby acknowledge and thank the authors & websites listed in the references for the valuable information

REFRENCES

[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] I.E. Sutherland, "Micropipelines," Turing Award Lecture, Communications of the ACM, Vol. 32, 1989, pp. 720-738.

[3] E. Brunvand, "The NSR processor," in *Proceedings of the 26th Hawaii International Conference on System Sciences*, 1993, pp. 428-435.

[4] *J. Sparsø and S. Furber, Principles of Asynchronous Circuit Design – A Systems Prospective, Kluwer Academic Publishers, London, 2001, pp. 11-25.*

[5] *D. Muller and W. Bartky, "A theory of asynchronous circuits," in Proceedings o International Symposium on the Theory of Switching, 1959, pp. 204-243.*

[6] 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.

[7] 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.

[8] 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.