國立交通大學

資訊科學與工程研究所

碩 士 論 文

運用 GasP 之低功耗一對多對一結構
先進先出裝置

A Low Power 1-n-1 Structure FIFO implementation with GasP

研 究 生：孫銘澤

指導教授：陳昌居　教授

中 華 民 國 九 十 七 年 六 月

運 用 GasP 之 低 功 耗 一 對 多 對 一 結 構 先 進 先 出 裝 置

A Low Power 1-n-1 Structure FIFO implementation with GasP

研 究 生：孫銘澤　　　Student：Ming-Tse Sune

指導教授：陳昌居　　　Advisor：Chang-Jiu Chen

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年六月

運用 GasP 之低功耗一對多對一結構 先進先出裝置

研究生：孫銘澤　　　　　　　　　　　　　　　指導教授：陳昌居 教授

國立交通大學資訊學院資訊科學與工程研究所

摘　要

現今電路設計的趨勢朝向低耗電發展，但也常常會因此而喪失其原本電路的效能，所以擁有低功耗和高效能的電路更具優勢。在本篇論文中，我們提出一個低功耗且高效能一對多對一結構先進先出裝置之設計與實作。它是基於GasP的電路模組所延伸出來的系統，且為了實作此系統，我們提出一個方法可以將此系統的演算法轉換成相對應之GasP的電路模組。接著為了評估先前提出的系統演算法優劣，我們在實作電路前會概略評估此演算法對應出來的電路模組是否符合低功耗和高效能的實作目標。最後我們分別實作了多個先進先出裝置來做比較，其中它們分別為十級和十八級且皆為一個位元儲存空間，於每秒三十億筆資料下以TSMC 180奈米製程模擬。結果指出一對多對一結構先進先出裝置在耗電方面幾乎擁有最好的表現，特別在十八級時可以達到多一倍的節能結果，且在更多級數的先進先出裝置更可顯現出其優勢。

A Low Power 1-n-1 Structure FIFO Implementation with GasP

Student：Ming-Tse Sune **Advisor：Dr. Chang-Jiu Chen**

Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University

Abstract

The current trend of circuit design is towards low power, but the performance is often degraded. Therefore the circuits with power-efficiency and high performance are superior. This thesis presents a low power and high performance 1-n-1 structure FIFO implementation, based on GasP modules. In order to implement the system, we explain a method to transform the algorithm of systems into the corresponding GasP modules. Then we derived several equations to analysis the algorithm to conform our purpose before we really implement our design. Finally, we compared the proposed structure with other structures. The depths we compared are ten and eighteen, and the width is one bit. We assume that the environment sends three billion data items per second to the FIFOs and it is simulated with the TSMC 180nm process. The result indicates the 1-n-1 FIFOs almost have the best outcome. In particular, the 1-n-1 FIFO with eighteen stages has one time improvement more than the square FIFO, and the predominance is more obvious when the depth of FIFOs becomes larger.

Acknowledgement

首先我要感謝指導教授陳昌居老師這兩年給予的指導，接著要特別感謝博士班的鄭緯民學長、張元騰學長、和蔡宏岳學長給予我的論文非常多寶貴的意見，也感激實驗室的夥伴們給予我的幫助。感謝同在新竹的大學同學，讓我這兩年非常順利，幫我解決很多困擾。感謝我的室友們，陪我度過人生中很快樂的兩年。最重要的是感謝我的家人，有你們支持才是我完成論文的最大支柱。謝謝大家。

CONTENTS

摘　要 i

Abstract ii

Acknowledgement iii

CONTENTS iv

List of Figures vi

List of Tables viii

CHAPTER 1 INTRODUCTION 1

1.1 Advantages of Asynchronous Circuits 1

1.2 The Evolution of Modern Asynchronous Pipelines 2

1.2.1 Two-phase and Four-phase Handshaking Protocols *2*

1.2.2 Bundled-data and Dual-rail Handshaking Protocols *3*

1.2.3 Modern Asynchronous Pipelines (FIFOs) *3*

1.3 Organization of This Thesis 8

CHAPTER 2 RELATED WORKS 9

2.1 Synchronous FIFOs Implementation 9

2.2 The Design of GasP Circuits 10

2.2.1 A Design Based on GasP Modules *10*

2.2.2 Translating FSM Specification into GasP Circuits *14*

2.3 A Linear FIFO 17

2.4 A Square FIFO 18

2.5 The Brief Introduction about 1-n-1 FIFOs 26

CHAPTER 3 TRANSLATING DATA FLOW STRUCTURE INTO GASP CIRCUITS 27

3.1 Transforming Data flow Structure into Preliminary Control Circuits 27

3.2 Transforming Preliminary Control Circuits into Real Control Circuits 29

CHAPTER 4 THE 1-N-1 FIFOS 32

4.1 The Ideas about Reducing Data moves 32

4.2 The Basis of a 1-n-1 FIFO 32

4.2.1 The Data flow Structure and Algorithms for Branches and Mergence *33*

4.2.2 Preliminary Control Circuits for Branches and Mergence *34*

4.2.3 GasP Circuits for Branches and Mergence *35*

4.3 The 1-n-1 FIFO with nine stages 40
4.4 Power-Estimation for Linear, Square, Binary Tree, and 1-n-1 FIFOs 45
CHAPTER 5 EXPERIMENTAL RESULTS 48
5.1 The Cycle Time of These FIFOs 48
5.2 The Power Consumption of These FIFOs 50
5.3 The Transistors Counts 54
CHAPTER 6 CONCLUSIONS AND FUTURE WORKS 55
REFERENCES 56

List of Figures

Figure 1.1 : A four-phase bundled-data pipeline 4

Figure 1.2 : A two-phase bundled-data pipeline 5

Figure 1.3 : A 1-bit wide four-phase dual-rail pipeline 5

Figure 1.4 : The LP2/1 pipeline 6

Figure 1.5 : (a) A STFB stage (b) A GasP stage 7

Figure 2.1 : One stage of the linear FIFO with asynchronous reset (a) Symbol (b) Detail circuits 9

Figure 2.2 : An irregular FIFO 10

Figure 2.3 : Alternative control circuit of asP* 11

Figure 2.4 : GasP with self-resetting NAND 13

Figure 2.5 : The corresponding circuits between self-resetting NAND 13

Figure 2.6 : Translating FSM specification into GasP modules 14

Figure 2.7 : Translating GasP modules into detailed GasP circuits 16

Figure 2.8 : Translating GasP modules into detailed GasP circuits 16

Figure 2.9 : A two-stage linear FIFO, (a) usual stages, (b) optimized stages, (c) detailed circuits 18

Figure 2.10 : The square FIFO's data movements 19

Figure 2.11 : The control circuits of the square FIFO 22

Figure 2.12 : (a) The stages of the top row (b) The stages of the bottom row 22

Figure 2.13 : Stage 1 (a) Stage with data moves (b) The processing graph for FSM specification (c) The GasP modules of stage 1 24

Figure 2.14 : Stage 16 (a) Stage with data moves (b) The processing graph for FSM specification (c) The GasP modules of stage 16 25

Figure 2.15 : The 1-n-1 FIFO with 9 stages 26

Figure 3.1 : (a) The data flow structure of a linear FIFO (b) The preliminary diagram for the control circuits of the linear FIFO........28
Figure 3.2 : The processing diagram for coming data items........30
Figure 3.3 : The corresponding diagram between preliminary control circuits and real control circuits........31
Figure 4.1 : The data flow for branches and mergence........33
Figure 4.2 : The pointers for the data flow structure........34
Figure 4.3 : The preliminary control circuits for a branch........34
Figure 4.4 : The preliminary control circuits for the mergence........35
Figure 4.5 : A GasP module and its input and output ports........36
Figure 4.6 : The GasP control circuits for a branch........37
Figure 4.7 : The details of figure 4.6........37
Figure 4.8 : The GasP control circuits for the mergence........38
Figure 4.9 : The preliminary control circuits for a binary tree FIFO........39
Figure 4.10 : The GasP control circuits for a binary tree FIFO........40
Figure 4.11 : Continuous data items gotten from the input port........42
Figure 4.12 : Continuous data items sent out by the output port........43
Figure 4.13 : The preliminary control circuits for the 1-n-1 FIFO........44
Figure 4.14 : The GasP control circuits for the 1-n-1 FIFO........44
Figure 4.15 : Power-comparison for FIFOs........47
Figure 5.1: The timing constraint for 1-n-1 FIFOs........49
Figure 5.2 : The layouts of 1-n-1 FIFOs (a) eighteen stages (b) ten stages........53

List of Tables

Table 2.1 : All paths of the square FIFO 20

Table 4.1 : The paths of the binary tree FIFO in figure 4.9 40

Table 4.2 : Equations for average working GasP modules per data item 47

Table 5.1 : The cycle time of the FIFOs 49

Table 5.2 : 3 billion request signals per second (only GasP control circuits) 50

Table 5.3 : 3 billion monotonous data items per second (Full circuits) 51

Table 5.4 : 3 billion various data items per second (Full circuits) 52

Table 5.5 : The transistors counts 54

Chapter 1 Introduction

The goal of this thesis is to implement a fast and power-efficient asynchronous FIFO. In the first chapter, we introduce some advantages of asynchronous circuits. Then, we introduce the evolution of modern asynchronous pipelines(FIFOs) in this field. Finally, the organization of this thesis is described.

1.1 Advantages of Asynchronous Circuits

Synchronous design styles are used to most digital circuits today. It is simple to design their control circuits because all components just share and notice the clock signal which distributed over the whole circuit. However, the arrival time of the clock signal may be not the same at different parts of the circuit. It is a well-known problem called “clock skew”.

Asynchronous design styles are fundamentally different. The components communicate with each other by handshaking circuits so that there are no common and global signals. Compared with the synchronous design styles, the asynchronous design style has many benefits:

(1) NO clock skew problem : Asynchronous components communicates by handshaking circuits so that there is no globally distributed clock; thus the designer can ignore the clock skew problem.

(2) High operating speed : The worst-case timing assumption is needed in synchronous circuits. Asynchronous circuit often have computation detection mechanism to complete the operations, and it is local latencies rather than global worst-case latency.

(3) Low power-consumption : Synchronous clock signal is sent to every component, and all components must operate when it arrives, even if it is not necessary in portions of current computation. The fundamentality of synchronous systems results in worse power-efficiency. However, asynchronous circuits are just fired when it is actually necessary. Energy is only consumed for needed operations. The fundamentality of asynchronous systems causes better power-efficiency.

(4) Excellent EM emissions : Handshaking circuits exchange signals at random points in time. It is unlike synchronous clock signal ticks at the same time so that EM emissions is much better. Lexau et al. implemented both synchronous and asynchronous counterflow pipeline models [1], and the result showed a dramatic 9 dB reduction in peak EM emissions at essentially constant performance levels [2].

There are more potential advantages are discussed in [3] and [4] such as better modularity, robustness toward variations in supply voltage and temperature, etc. Modern high-speed asynchronous circuits tend to design with full-custom procedure because of few CAD tools support and lack of testing methods. Although it is more elastic to circuits, designer must spend more time on layout. For this reason, asynchronous circuits are not popular for modern design style.

1.2 The Evolution of Modern Asynchronous Pipelines

Modern asynchronous pipelines mainly use the { two-phase , four-phase } and { bundled-data , dual-rail } handshaking protocols, and there are many asynchronous processors implemented by them. Many asynchronous CAD tool and textbooks also use these protocols as their foundation. However different architectures of pipeline are discussed in the recent years. The Lookahead pipeline, a variation of domino pipelines, uses complex circuits and more signals to achieve shorter cycle time. On the contrary, the GasP and STFB use pulse-mode circuits to reduce the handshaking protocols but they still have high throughputs. Theses handshaking protocols and pipelines will be introduced in this section.

1.2.1 Two-phase and Four-phase Handshaking Protocols

In asynchronous circuits, the major handshaking protocols are two-phase and four-phase handshaking. A “signal event” in the two-phase handshaking protocol is defined when request and acknowledgement lines change, and it means handshaking signals exchanged when the "signal event" actives. Hence some articles use the terms "non-return-to-zero " or "transition signaling" instead of two-phase handshaking.

Compared with two-phase handshaking, four-phase handshaking is more complex. (1) When the communication cycle starts, the sender sends out data and then pull up the request line. (2) The receiver obtains the data and then pull up the acknowledgement line. (3) The sender pushes down the request line to respond the receiver. (4) The receiver pushes down the acknowledgement line, and the communication cycle completes. Some articles also use the terms "return-to-zero " or "level signaling" instead of two-phase handshaking according to its behavior.

1.2.2 Bundled-data and Dual-rail Handshaking Protocols

We can distinguish between bundled-data and dual-rail on how they transfer the data signals. The bundled-data protocol encodes data signals by using normal Boolean levels. It separates the request and acknowledgement lines from the data signals. In opposition to the bundled-data protocol, the dual-rail protocol encodes the request signals into the data signals; therefore two wires represent one bit information.

The dual-rail protocol is a special kind of one-hot encodings, and it can be extended as 1-of-n encodings. The 1-of-n encodings can also be extended to m-of-n encodings. If the circuits are designed for communication, m-of-n encodings are better choices. It can reduce the overhead of communication, but the computation circuits of m-of-n encodings are very complex.

1.2.3 Modern Asynchronous Pipelines (FIFOs)

Muller pipeline :

The Muller pipeline is presented in [3]. It is a four-phase bundled-data protocol and most asynchronous pipelines are variations of its control circuits. The behavior of the Muller pipeline is just like a local clock generator. In figure 1.1, the C-element of this stage propagates a 1 (request signal) if its predecessor sends a 1 (request signal) and its successor sends a 0 (acknowledgement signal). In the same way it propagates a 0 if its predecessor sends a 0 and its successor sends a 1. The signals are like clock pulse signals generated by C-elements.

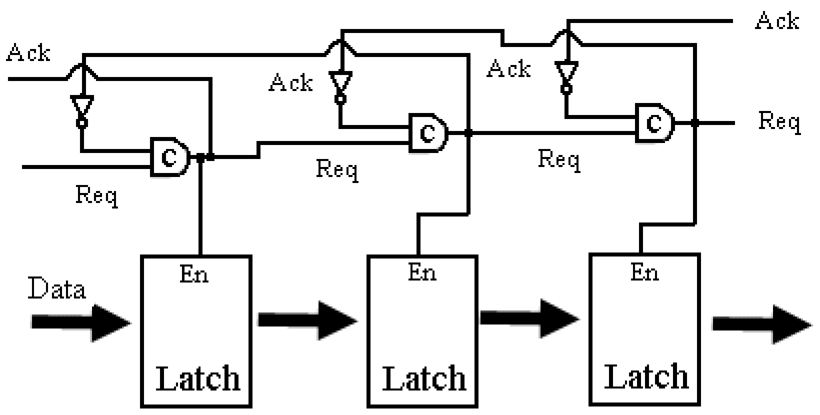


Figure 1.1 : A four-phase bundled-data pipeline

Micropipeline :

Ivan Sutherland introduced the Micropipelines in his 1988 Turing Award lecture [5]. The Micropipeline bases on the two-phase bundled-data protocol and it also use the foundation of Muller pipeline as its control circuit. In figure 1.2, the different point between control circuits of Micropipeline and Muller-pipeline is only the acknowledgement signal from its successor used as the control signal of latches. Therefore the latches of the Micropipeline are controlled by two signals, “capture” and “pass”. When “capture” and “pass” signals are in the same Boolean levels, the latches pass the data signals. Otherwise they capture the data signals.

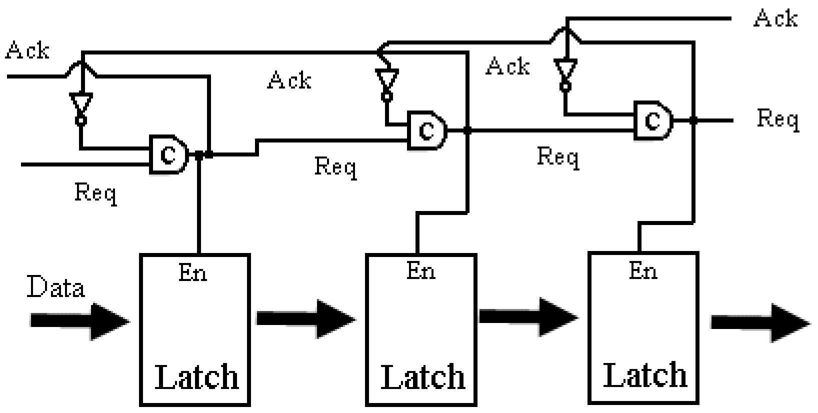


Figure 1.2 : A two-phase bundled-data pipeline

Four-phase dual-rail pipeline :

The RISC microcontroller, APIC18, uses the four-phase dual-rail protocol as its control circuits [6]. The request signal of the four-phase dual-rail protocol is encoded into the data signals; thus the receiver just detects whether the mixed signals is absorbed or not so that it can be delay-insensitive. Although this protocol is robust enough for any timing assumptions, it needs more logic gates to implement the functions with only at 50% utilization. In figure 1.3, the situations of pipeline stages are alternately “empty” and “valid”. If its situation is empty, {d.t,d.f}, the mixed signals, are {0,0}. Otherwise the {d.t,d.f} are {0,1} or {1,0}, and it represents the valid data signals are 0 or 1. It should be noticed that if {d.t,d.f} equals {1,1}, it is not a legal codeword. That is because it is not need to encode any valid information.

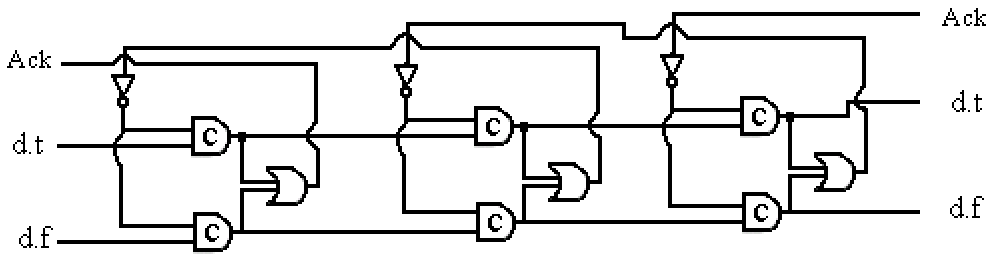


Figure 1.3 : A 1-bit wide four-phase dual-rail pipeline

Lookahead Pipelines :

In addition to common pipelines introduced above, Lookahead Pipelines are designed with special protocols. Their behaviors are similar to the domino pipelines. They extended PS0 [7] in order to gain better throughput and introduced several improved protocols [8,9]. The key points of those protocols are (1) early evaluation, (2) early done, and (3) combination of both. In figure 1.4, the LP2/1 pipeline in [7] combines "early evaluation" and "early done". Each stage receives information from two succeeding stages, the "early evaluation" protocol is used, and the "Eval" signal comes from the completion detector two stages ahead so that the current stage can evaluate early. The idea of "early done" let the previous pipeline receive the information whatever it evaluates or precharges. Because of these reasons, the early evaluation" and "early done" protocols can achieve a shorter cycle time.

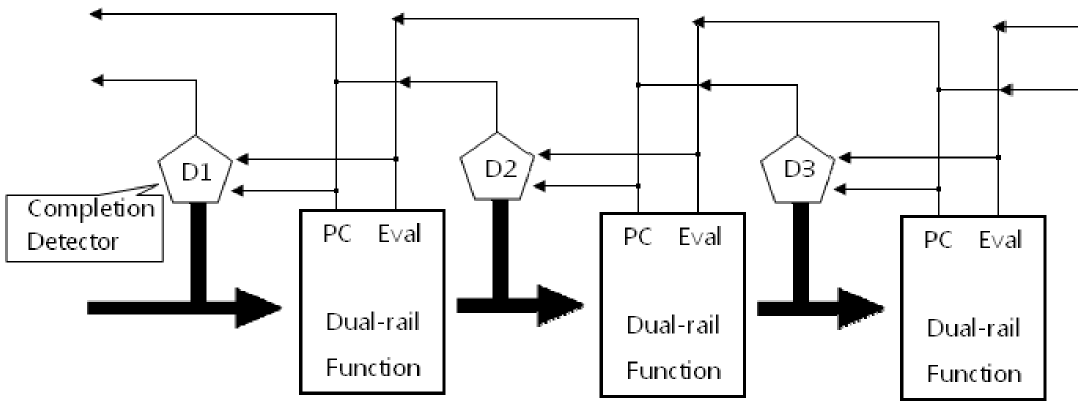


Figure 1.4 : The LP2/1 pipeline

***GasP* and *STFB* :**

In recent ten years, many researchers are interested in pulse-mode circuits, like GasP [10] and STFB [11]. The advantage of these pulse-mode circuits is high speed, but come with high noise sensitivity. In figure 1.5(a), a STFB stage is implemented in dual-rail protocol, and its data, request, acknowledgement signals are transferred via the same tri-state wires. A GasP stage is implemented in bundled-data protocol as shown in figure 1.5(b). The request and acknowledgement signals of GasP circuits are also transferred via the same tri-state wires, and the self-resetting mechanism let the cycle time be shorter. Both GasP and STFB circuits weakly keep states by states keepers on tri-state wires, and the wires must be susceptible to noise especially in smaller noise margins. Golani and Beerel presented high-performance noise-robust asynchronous circuits to mitigate sensitiveness to noise, including transistor sizing and wire spacing rules [12].

In particular, the request and acknowledgement signals of pulse-mode circuits are generated by the PMOS and NMOS which are marked with circles in figure 1.5. Because the handshaking protocol is simpler than the protocols introduced above, it can reduce the cycle time so that the pulse-mode circuits can operate in high speed.

The goal of this thesis is to implement a fast and power-efficient asynchronous FIFO with GasP circuits; thus the details of GasP circuits will be described in the next chapter.

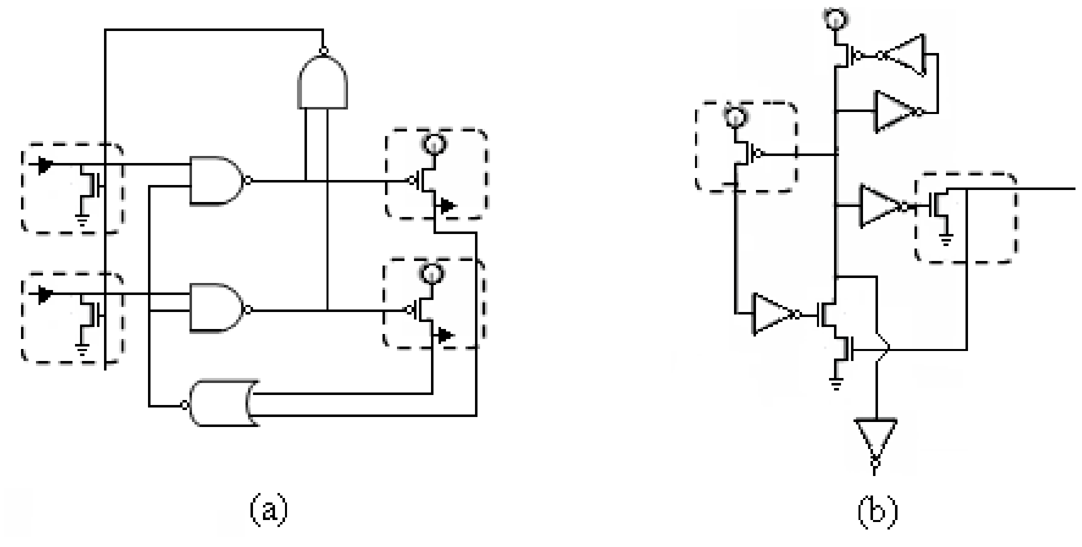


Figure 1.5 : (a) A STFB stage (b) A GasP stage

1.3 Organization of This Thesis

In the next chapter, we will discuss some issues in the design of GasP circuits. Then, in chapter 3 we provide an approach to transform a data flow structure into parallel compositions of GasP circuits. In chapter 4 we present the details of the FIFO architecture and estimate the power-consumption roughly. The implementation, verification and result are exhibited in chapter 5. Finally, we give conclusions in chapter 6.

Chapter 2 Related Works

In this chapter, we will introduce the synchronous FIFOs, and then the design of GasP circuits is introduced clearly. It includes the features of GasP circuits, the operations when it works, the FSM specification to describe them, and ways to translate FSM specification into GasP circuits, i.e. a linear FIFO and a square FIFO. Finally, a brief introduction about the proposed 1-n-1 FIFO will be mentioned.

2.1 Synchronous FIFOs Implementation

Although this thesis focuses on asynchronous systems which is implemented with GasP circuits, synchronous FIFOs are need to be compared with asynchronous FIFOs. Thus we introduce two major methods to implement the synchronous FIFOs. The first one is common linear FIFO with asynchronous reset, and one of its stage is shown in figure 2.1. The input port "D" receives data items from its predecessor and the output port "Q" sends data items to its successor, all data items are moved when the clock signal changes. When the clock signal is 0, a data item is delivered to the first part of this stage. Otherwise a data item is delivered from the first one to the output port "Q".

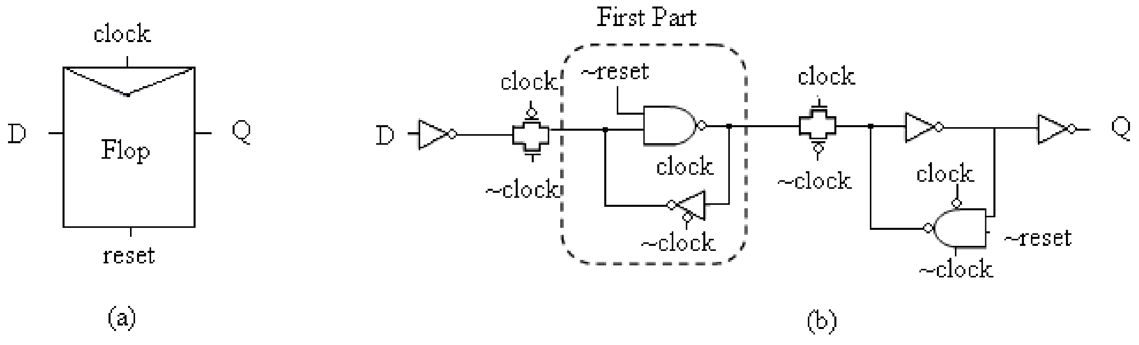


Figure 2.1 : One stage of the linear FIFO with asynchronous reset (a) Symbol (b) Detail circuits

The second one is the irregular FIFO, it uses two pointers to position the head and tail. The two pointers work in coordination with the control circuit as shown in figure 2.2, and the control circuit will send the control signals to let the irregular FIFO synchronize correctly. Because the synchronous irregular FIFOs are discussed fewer and it seems to have fewer benefits in synchronous systems, we only selected the linear FIFO to compare with our design.

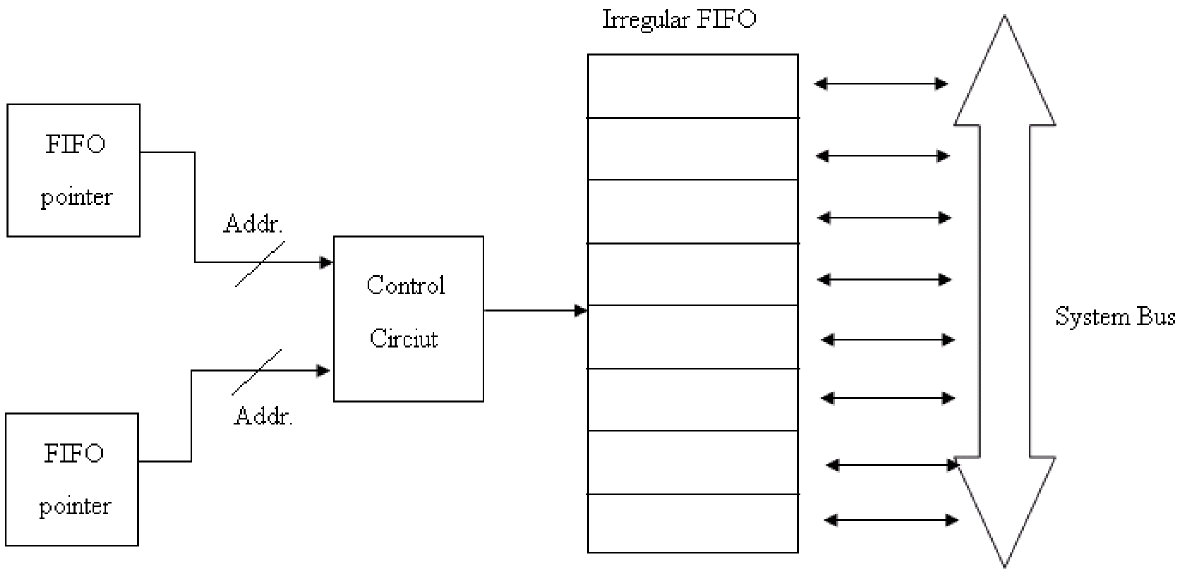


Figure 2.2 : An irregular FIFO

2.2 The Design of GasP Circuits

In this session, we will discuss the evolution of GasP circuits, its detailed design, and its timing constraints. Then we will introduce how to use the finite-state machines (FSM) specification to describe a GasP module, and translating GasP modules into GasP circuits.

2.2.1 A Design Based on GasP Modules

In [13], Molnar brought a basic control circuits for an asynchronous pipeline called Asynchronous Symmetric Persistent Pulse Protocol, “asP*”, and it can be regarded that the last three letters of GasP meant its asP* ancestry. “asP*” uses a series of flip-flops to control circuits and special transition latches. In figure 2.3, the asP* circuits are symmetric, and their forward latency and reversed latency are equal. The performance of asP* circuits is report in [14].

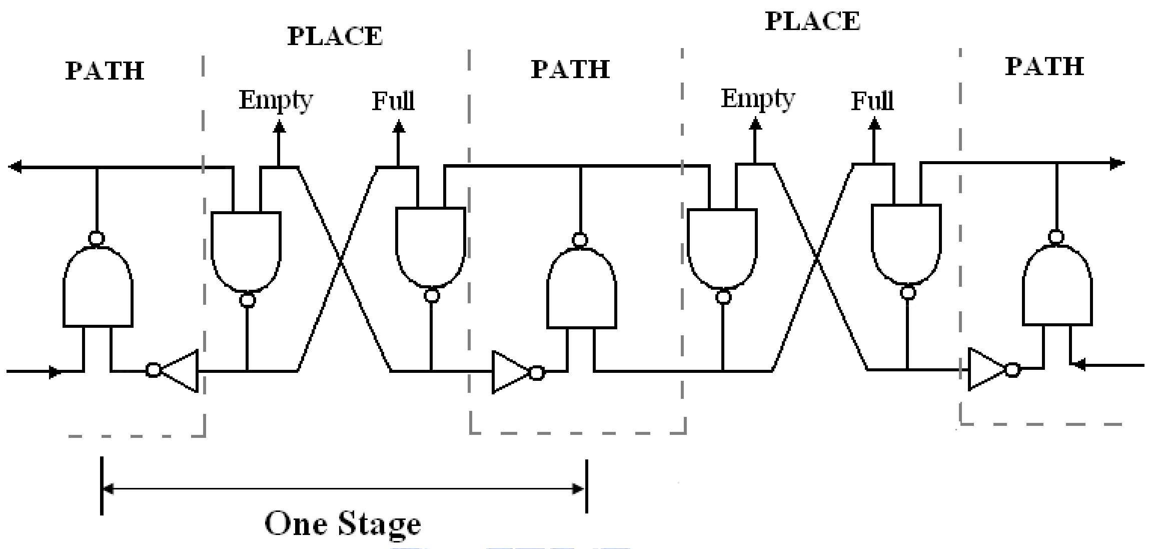


Figure 2.3 : Alternative control circuit of asP*

The GasP asynchronous circuits provide controls for simple pipelines. Its general idea is that "in very fast asynchronous circuits it is better to make the forward latency long and the reverse latency short". For this reason, GasP circuits take time to copy data forward through a latch, but none are moved backwards.

Let us define two key words "PLACE" and "PATH" to distinguish two kinds of circuits: "PLACE" represents circuits which hold the data item, "PATH" represents circuits which control the flow of data, and then PATHs and PLACEs alternate in the pipeline as shown in figure 2.3 and figure 2.4. Each PATH has a predecessor and successor PLACE, and each PLACE has a predecessor and successor PATH. PATHs must act only when both its predecessor PLACE is FULL and its successor PLACE is EMPTY.

Sutherland and Fairbanks pointed out that a PATH must get through below four things when it fires [10] :

(1) It must make data latches momentarily transparent.

(2) It must declare its successor stage FULL

(3) It must declare its predecessor stage EMPTY.

(4) It must reset the output of the series N-type transistors to the inactive or HI state.

In figure 2.4, the GasP circuits described here have a forward latency of four gate delays and a reverse latency of two gate-delays, and then the total cycle time is therefore six gate-delays (NO. 123456).

GasP circuits store each state with state conductor to indicate whether it is FULL or EMPTY. In particular, it uses the state encoding “HI=EMPTY” and “LO=FULL” for all state conductor, and the HI signal declares no data in the present latch at this moment. On the contrary, the LO signal declares that there is a data item in this latch. Because we implement GasP circuits with standard NAND symbol to replace the self-resetting NAND gate, the corresponding circuits between them will be shown in figure 2.5.

In order to implement correct functionality of GasP circuits, two conditions must be satisfied. The first one is meeting the setup and hold time, and the other one is the cycle time must be longer than the minimum delay of six gate-delays. To achieve the conditions, we would adjust the transistor size to reduce the logical effort with what explained in [15] and [16].

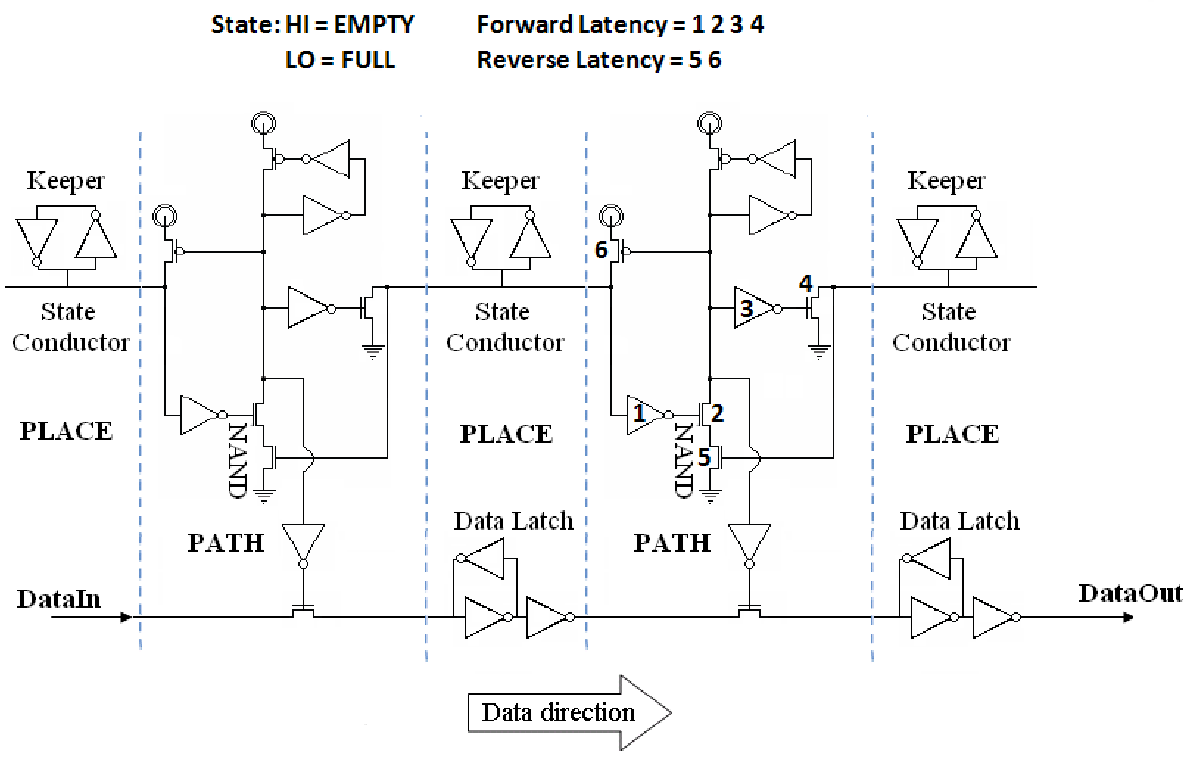


Figure 2.4 : GasP with self-resetting NAND

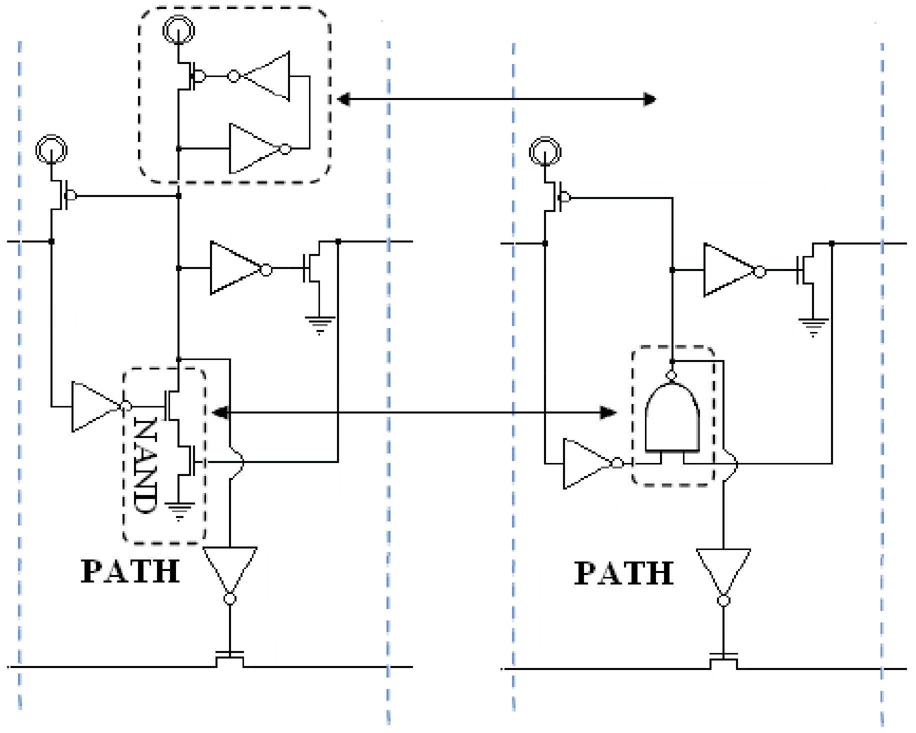


Figure 2.5 : The corresponding circuits between self-resetting NAND and standard NAND

2.2.2 Translating FSM Specification into GasP Circuits

The method of how to translate FSM specification into GasP circuits is introduced in [17] and [18], and it is often used to plan the PATHs. The basic state transition is shown below :

$$P = (a0 \rightarrow Q)$$

Where P is defined as an input state, Q is an output state, and a0 is a GasP module. When state P is set, module a0 will be fired. Then state Q is set, and at the same time state P is reset. The operations can be finished in one cycle. The mapping is shown in figure 2.6(a).

A more complex FSM with deterministic choice is in figure 2.6(b), its state transition is shown below :

$$I = (b1 \rightarrow J \mid b2 \rightarrow K)$$

Where I is still defined as an input state, J and K are output states, and b1, b2 are GasP modules. The environment can determine which modules should be fired. Therefore module b1 or b2 may be fired according to the deterministic choice. In other words, state J may be fired not only setting I but also the determination of this environment. The status of state K is the same as state J. Therefore module b1 and b2 may be fired by setting I and environment information, and then the corresponding state J and K are set.

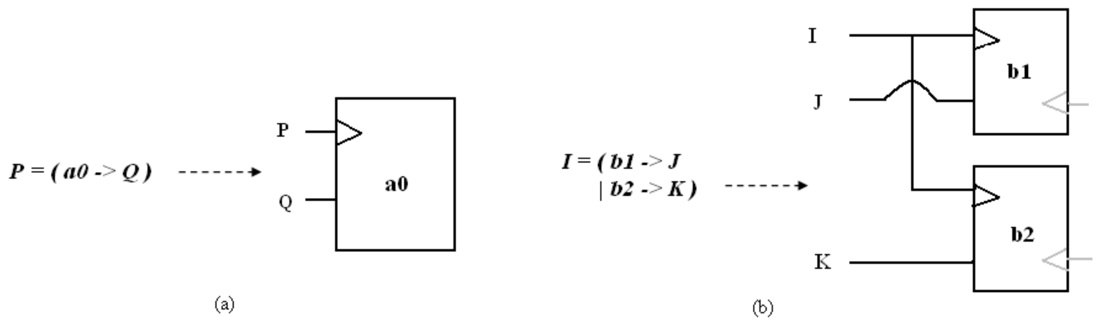


Figure 2.6 : Translating FSM specification into GasP modules

(a) a basic state transition, (b) deterministic choice

The following section describes how to translate GasP module into detailed GasP circuits. Figure 2.7 shows the relative mapping between modules and circuits, a0 and a1 can be regarded as an event is fired. P is one state stored in the state conductor, and it is full or empty. The label 4 refers the gate-delays between two modules when events are fired, and the numbers 1, 2, 3, and 4 means the successive gate-delays between events. Figure 2.8 is the same as figure 2.7. Q is one state between two modules, the label 2 also refers the gate-delays, and the successive gate-delays are number 1 and 2.

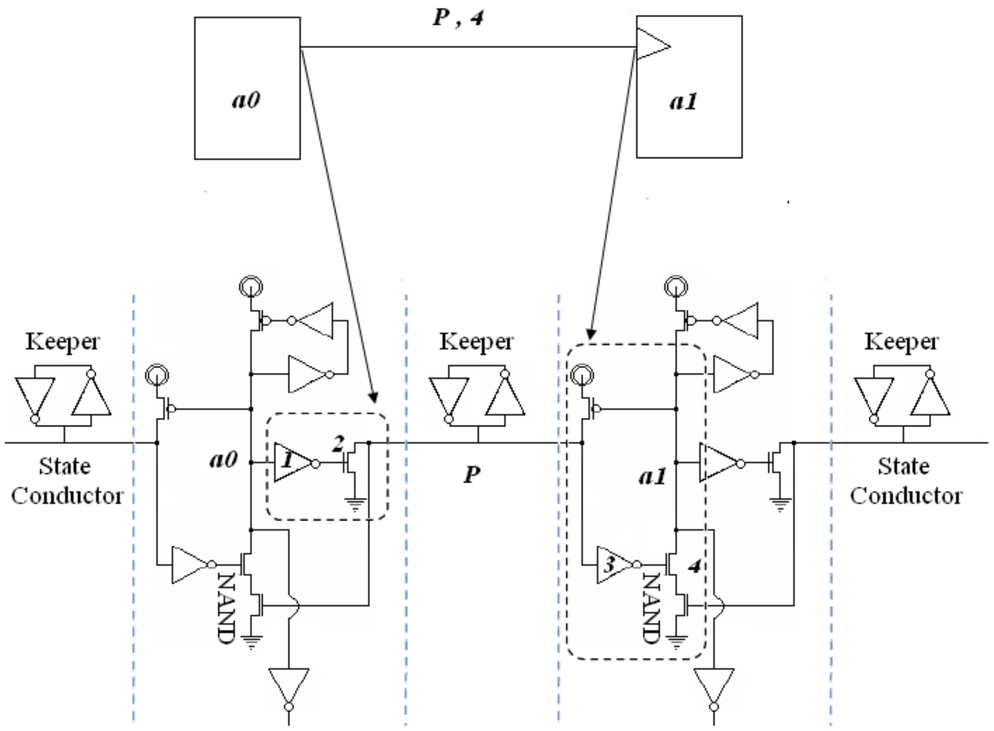


Figure 2.7 : Translating GasP modules into detailed GasP circuits

(4 gate-delays between 2 modules)

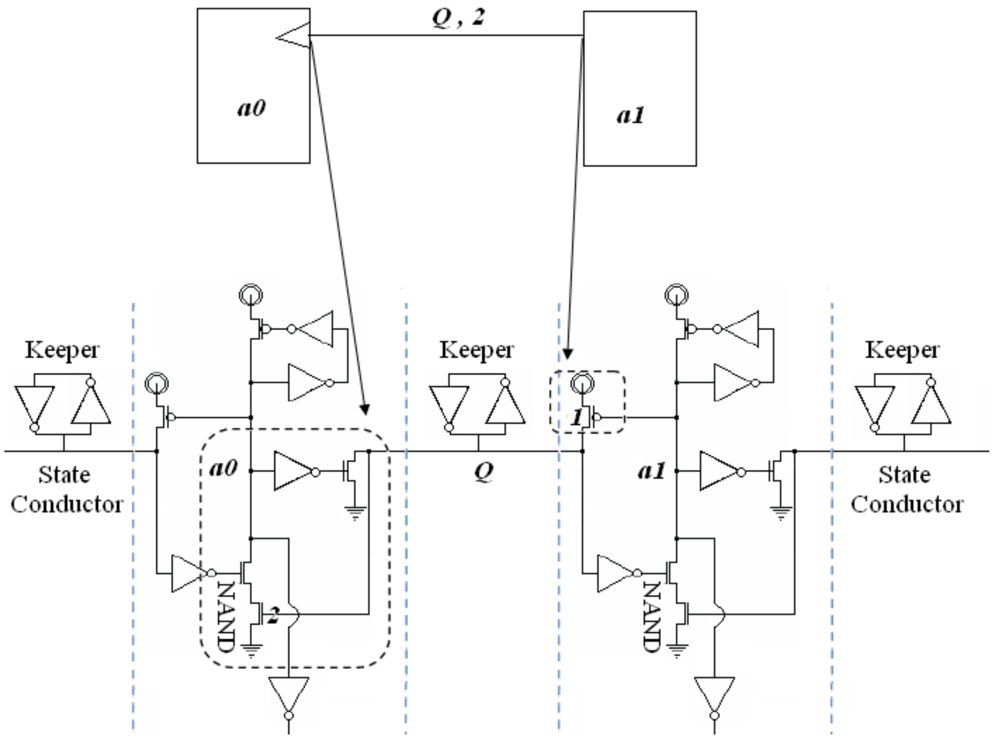


Figure 2.8 : Translating GasP modules into detailed GasP circuits

(2 gate-delays between 2 modules)

2.3 A Linear FIFO

The linear FIFO consists of GasP circuits shown in figures 2.7 and 2.8. We can use FSM specification to describe it, and give the corresponding modules. Considering the middle stages of a linear pipeline, the FSM is shown below :

```
Stage N =                          Stage N+1 =
   state E where                         state G where
      E = ( a0 -> F )                    G = ( a1 -> H )
      F = ( a1 -> E )                    H = ( a2 -> G )
   end                                   end
```

In figure 2.9(a), E, F G, and H are states and a0, a1, and a2 are modules. In particular, the shaded triangle of modules indicates where the initial states are. Otherwise the unshaded triangles represent the state are not the initial ones. Figure 2.9(b) is optimized from figure 2.9(a), and figure 2.9(c) is the detailed circuits of figure 2.9(b). One confusing key point is the difference between stages and modules. Figure 2.9 is a good example that a two-stage linear FIFO with three modules, and therefore one stage may be constructed with many modules in GasP circuits.

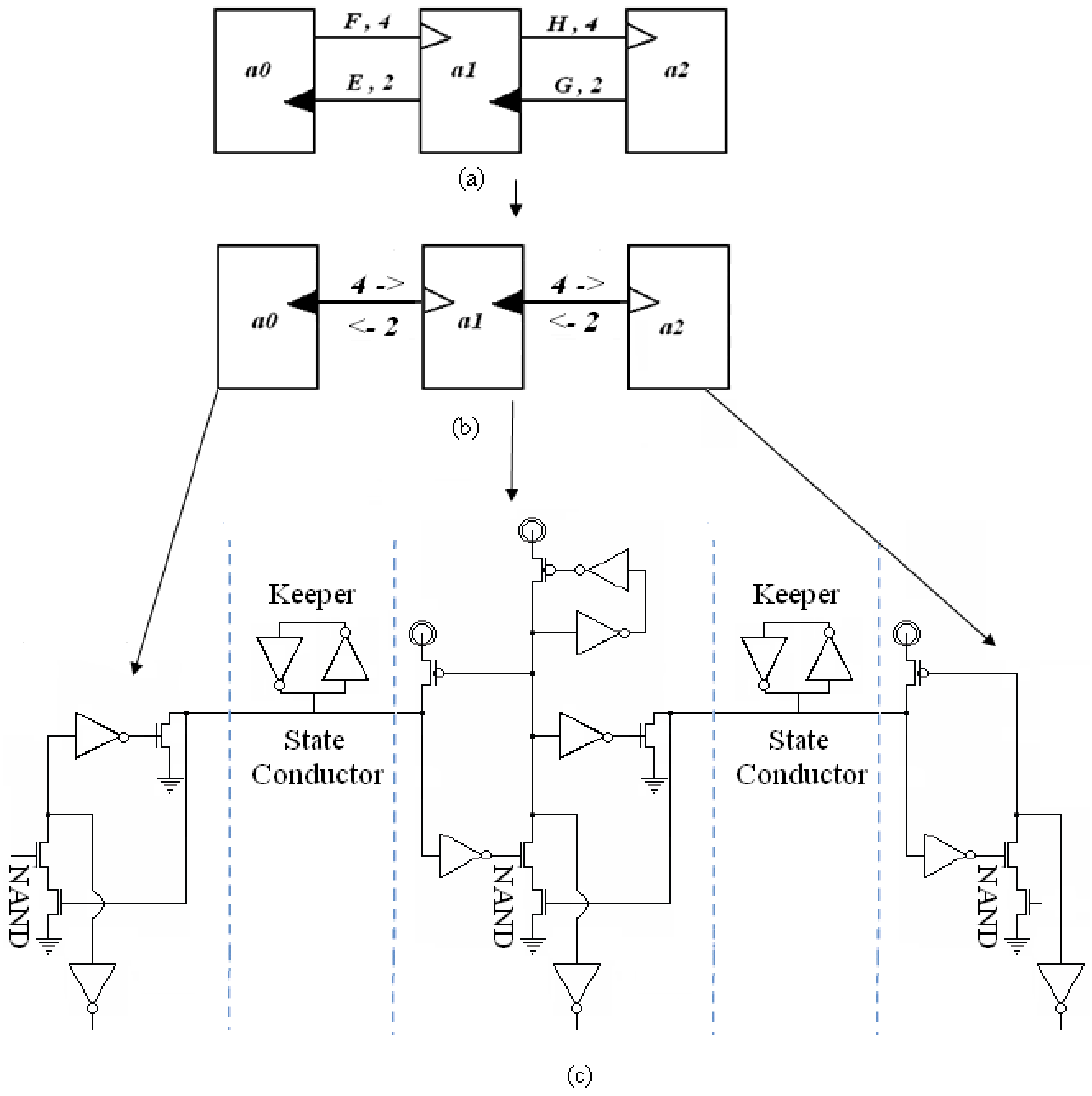


Figure 2.9 : A two-stage linear FIFO, (a) usual stages, (b) optimized stages, (c) detailed circuits

2.4 A Square FIFO

Ebergen presented a square FIFO implemented with GasP circuits [17]. The square FIFO is one of the low-latency and power-efficient FIFOs, and it consists of a two-dimensional array of stages. The route of one data item is schemed by the FSM specification of stages and the input-output behavior is the same as usual FIFOs.

Figure 2.10 illustrates the operation of a square FIFO with 16 stages. It consists of a top row of stages, some column FIFOs which are simple linear FIFOs, and a bottom row of stages. The top row sends out the data items to the column FIFOs in a round-robin fashion, and the bottom row also receives the corresponding data items in a round-robin fashion. Because every data item must go through stages of the square FIFO in order, the data items are distributed to their fixed route.

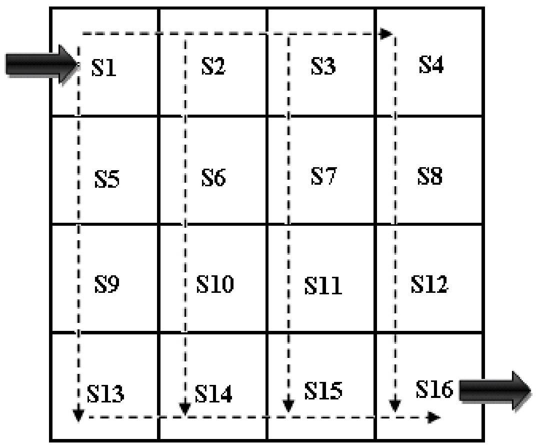


Figure 2.10 : The square FIFO's data movements

The data items goes through the square FIFO via four possible paths which are drawn by the dotted lines. The first data item goes through the square FIFO in the sequence S1, S2, S3, S4, S8, S12, and S16 stages. The routing path of the second data item is S1, S2, S3, S7, S11, S15, and S16. The routing path of the third data item is S1, S2, S6, S10, S14, S15, and S16. The routing path of the fourth data item is S1, S5, S9, S13, S14, S15, and S16. It needs the round-robin scheme and then repeats the same routing path. Therefore the routing path of fifth data item is the same as the first one and so on.

All data flow paths in figure 2.10 are listed in the table 2.1 where the variable *m* is an integer and *m* can be considered as the number of cycles. For example the routing path of the 21th data item is $4m + 1^{st}$ in the table 2.1.

Table 2.1 : All paths of the square FIFO

The *k*th data item	Through stages from input port to output port
4*m* + 1st	S1→S2→S3→S4→S8→S12→S16
4*m* + 2nd	S1→S2→S3→S7→S11→S15→S16
4*m* + 3rd	S1→S2→S6→S10→S14→S15→S16
4*m* + 4th	S1→S5→S9→S13→S14→S15→S16

The full control circuits of the square FIFO is shown in figure 2.11. The control circuits for data paths dictates data movements by letting the corresponding pass gates transparent. In figure 2.11, the stages 5~12 are simple linear pipeline shown in the above section. The control circuits for the top row are in the stages 1~4, and the bottom ones are in the stages 13~16. Therefore the FSM specification of stage 5~12 are the same as a linear FIFO in figure 2.8. However special FSM specification is needed to describe the top and bottom rows. Because the bottom row is easier to be described than the top row, the bottom row is presented first.

The problem for the bottom row is to determine when a data item can be received from the left-side or up-side. Most solutions involve counting mechanism, but the solution of [17] is different. Its rule for each stage of the bottom row is shown as follows :

"If the present data item passing through a stage comes from the left-most stage, the next data item for the stage must come from the stage above. Otherwise the next data item comes from the stage to the left. The first data item comes from the stage above. "

The rule of the bottom row can guarantee the order of data items because it totally corresponds the sequence of data items sent out by the top row. The rule of the top row is similar to the rule of the bottom row, and the problem for the top row is to determine when a data item can be sent to the right-side or down-side. The solution of the top row is also similar to the one of the bottom row. Each stage of the bottom row looks to a special data item moving right, and each stage of the top row looks to a special "bubble" item moving left. The interpretation of a bubble is a movement that involves both a movement of a data item to the right and a movement of a bubble item to the left. So the rule for each stage of the top row is shown as follows :

"If a data item moves into a stage of the top row by means of a swap with a bubble originating from the right-most stage, the next data item move out of the stage must be down; otherwise, the next data move item must be to the right. Initially, each stage of the top row is empty."

The rules for each stage of the top and bottom rows are explained above. Figure 2.12(a) illustrates the data moves of the top row. To keep track of whether a bubble comes from the right-most stage or not, it uses two types of arrows to distinguish. One is the dotted arrows which keep the track of a bubble coming from the right-most stage, and the other is the solid arrows recording all the other moves. Figure 2.12(b) is similar to figure 2.12(a). It shows the data moves of the bottom row. In order to keep track of whether a data item comes from the left-most stage or not, it also uses two types of arrows to distinguish. One is the dotted arrows which record a data item coming from the left-most stage, and the other is the solid arrows recording all the other moves.

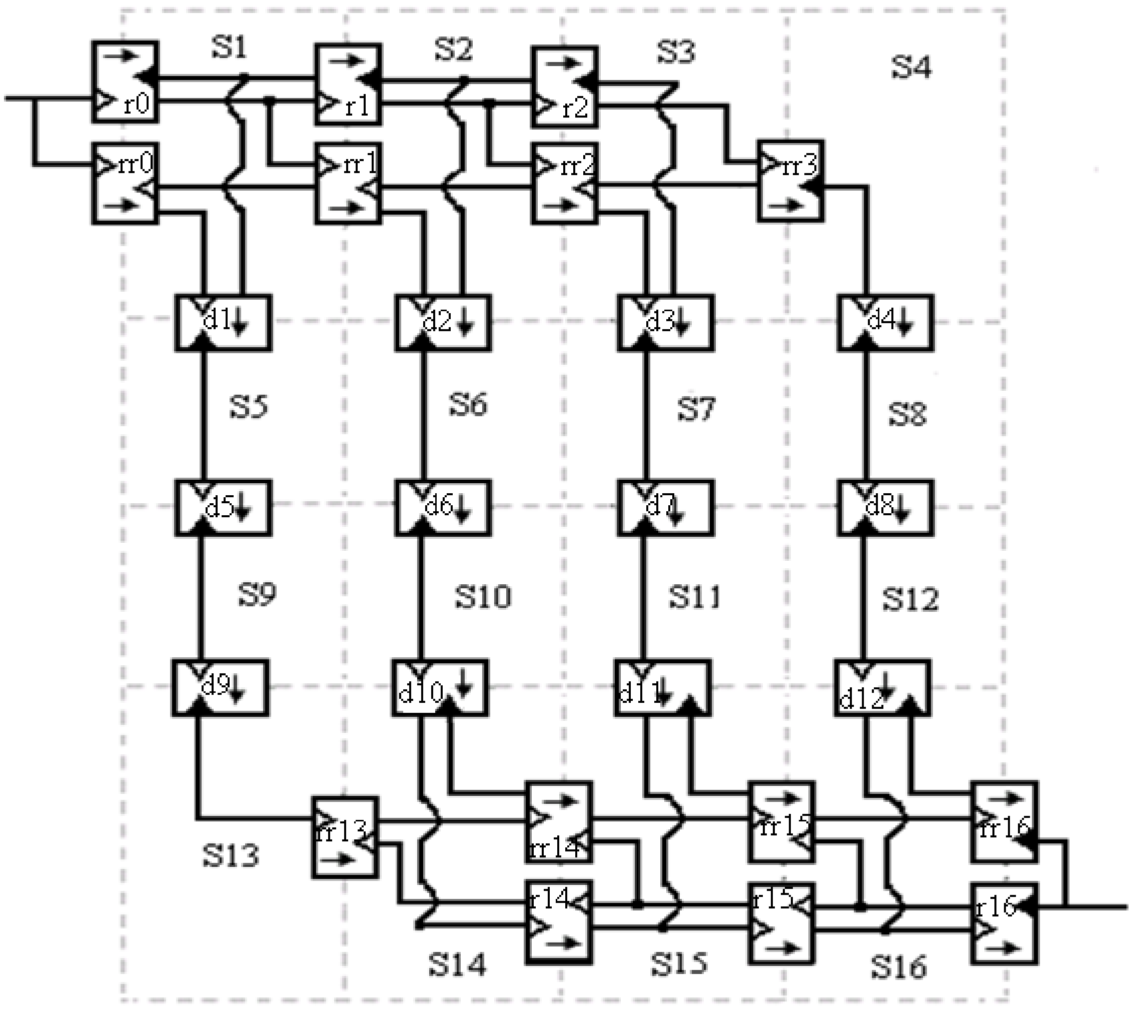


Figure 2.11 : The control circuits of the square FIFO

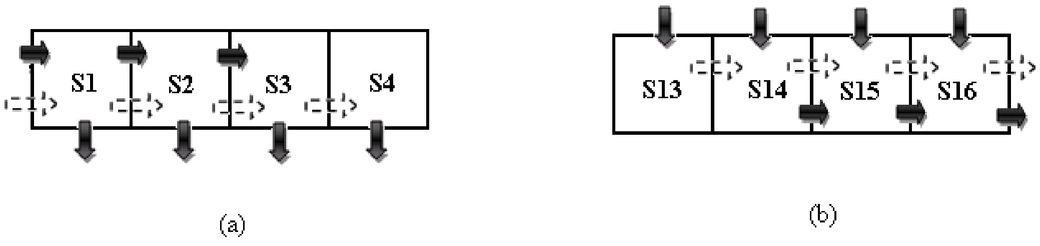


Figure 2.12 : (a) The stages of the top row (b) The stages of the bottom row

The rules for each stage of the top and bottom rows are explained clearly, and now we can give the FSM specification for the top and bottom rows. The FSM specification of stage 1 is shown as follows:

```
Stage 1 =
    state E0 where
        E0 = ( r0 -> F0)
        F0 = ( r1 -> E0 | rr1 -> E1)
        El = ( rr0 -> F1 )
        F1 = ( dl -> E0 )
    End
```

Let us translate FSM specification into GasP modules. Figure 2.13(a) shows the stage 1 with data moves, and then we use the FSM specification to draw the processing graph in figure 2.13(b). Finally, we translate the processing graph in figure 2.13(b) into the implementation of GasP modules in figure 2.13(c).

In figure 2.13(a) and (b), the dotted arrows represent the bubbles coming from the right-most stage of the top row, and they use prefix “rr” instead of “r”. Then we translate the FSM specification of stage 1 into figure 2.13(b). The states E0, E1, F0, and F1 are corresponded to the circles in figure 2.13(b). The modules r0, rr0, r1, rr1, and d1 are also corresponded to the lines in figure 2.13(b). Circles and lines connect with one another by the relationships of FSM specification. Finally, we translate the circles in figure 2.13(b) into tri-state wires in figure 2.13(c), and transforming the lines in figure 2.13(b) into the real GasP modules in figure 2.13(c).

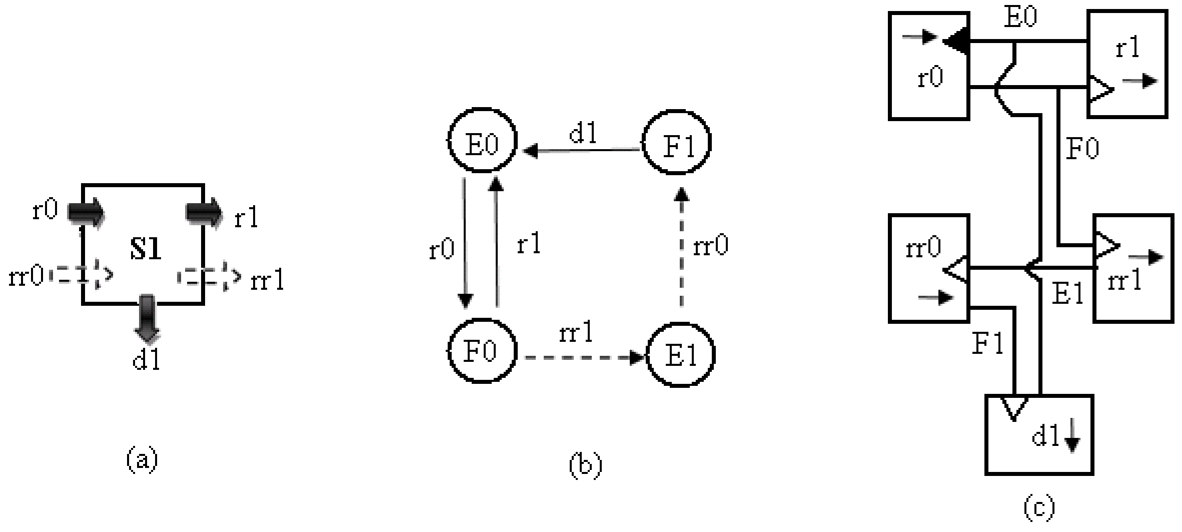


Figure 2.13 : Stage 1 (a) Stage with data moves (b) The processing graph for FSM specification (c) The GasP modules of stage 1

The FSM specification of stage 16 is similar to the stage 1, and it is shown as follows:

Stage 16 =

state E0 where

E0 = (d12 -> F0)

F0 = (r16 -> E1)

El = (r15 -> F0 | rr15 -> F1)

F1 = (rr16 -> E0)

end

Figure 2.14(a) also shows the stage 16 with data moves, and the processing graph of the stage 16 in figure 2.14(b). Finally, the implementation of GasP modules is in figure 2.14(c).

The dotted arrows in figure 2.14(a) and (b) stand for the data items coming from the left-most stage of the bottom row, and they also use prefix "rr" instead of "r". Then we transform the FSM specification of stage 16 into figure 2.14(b). The states E0, E1, F0, and F1 are corresponded to the circles in figure 2.14(b). The modules r0, rr0, r1, rr1, and d1 are also corresponded to the lines in figure 2.14(b). Circles and lines link with one another by the relationships of FSM specification for the stage 16. Finally, we transform the circles in figure 2.14(b) into tri-state wires in figure 2.14(c) and translate the lines in figure 2.14(b) into the real GasP modules in figure 2.14(c).

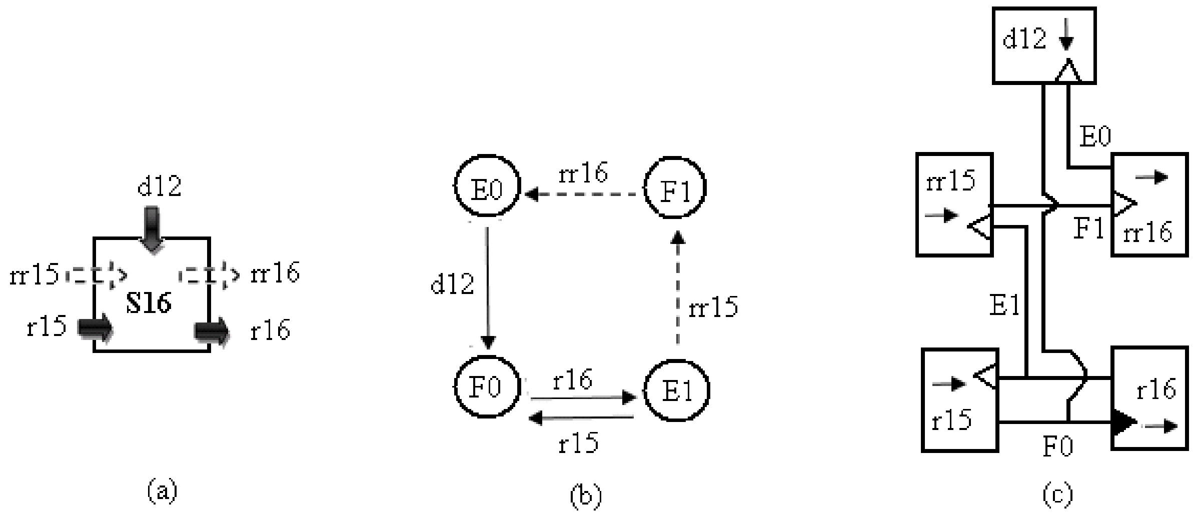


Figure 2.14 : Stage 16 (a) Stage with data moves (b) The processing graph for FSM specification (c) The GasP modules of stage 16

The stages 2 and 3 of the top row are similar to stage 1, and the stages 14 and 15 of the bottom row are similar to stage 16. The other stages are all simple one-stage linear FIFO. Since all stages of the square FIFO can be understood, the control circuits of the square FIFO can be implemented as shown in figure 2.11.

2.5 The Brief Introduction about 1-n-1 FIFOs

One goal of this thesis is to reduce the data moves in FIFOs. The architecture of the square FIFO in [17] presented fewer data moves than linear FIFO, but it still needs O($\sqrt{n}$) data moves where n is the number of stages. We hope to find some architecture of FIFOs that have high-throughput and fewer data moves with power-efficiency. From the viewpoint of algorithm, O(1) reaches the best beneficial result for work efficiency.

In order to achieve the goal, the 1-n-1 structure is considered the best architecture for the FIFOs. We should find the rule for the 1-n-1 FIFO and implement it with GasP circuits. A structure in figure 2.15 is O(1), and the solid arrows indicate the data moves. Each data item from the input port to the output port only crosses three stages, stage S1 to one of S2~S8 and one of S2~S8 to stage S9. In the general case, the 1-n-1 FIFO with n stages, each data move from the input port to the output port also goes through three stages, stage S1 to one of middle stages and from one of middle stages to stage Sn. More details of the 1-n-1 FIFO will be discussed in the chapter 4.

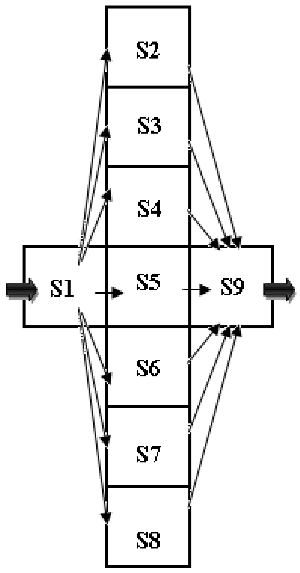


Figure 2.15 : The 1-n-1 FIFO with 9 stages

Chapter 3 Translating Data flow Structure into GasP Circuits

In this chapter, we will introduce how to translate data flow structure into corresponding GasP circuits. The method has two steps: the first step is to transform a data flow structure into preliminary control circuits, and the second step is to depict real GasP circuits by its preliminary control circuits. We will give an example of a linear FIFO to understand clearly.

3.1 Transforming Data flow Structure into Preliminary Control Circuits

One goal of this thesis is to transform a data flow structure into parallel compositions of GasP circuits. Giving unlike accounts of the incident with [17], we do not use FSM specification to describe a stage because it would be too complex to characterize a whole system. For example, the square FIFO in [17] and the efficient stack in [18] have complicated FSM specification for each stage, even the whole systems. Instead of finite-state machines, we prefer to plan a data flow structure first, and then translating it into the control circuits.

In order to explain the method clearly, we give an example of a linear FIFO with five stages, and translating it into GasP control circuits. The first step is to plan the data flow structure. The data flow structure of a linear FIFO is well-known, and the rule is receiving data items from its predecessor and sending them to its successor.

Figure 3.1(a) illustrates the data flow structure of a linear FIFO, the arrows indicate the direction of data moves. Since we know the rule and the data flow structure both, the corresponding graph of figure 3.1(a) can be depicted in figure 3.1(b). Figure 3.1(b) is a preliminary diagram for the control circuits of the linear FIFO. Its general idea is like the state transition graph (STG) diagram, and there are two rules to be obeyed. The first one is 1-bounded that no arc can ever contain more than one token, and the second one is no deadlock.

The circles in figure 3.1(b) can be regarded as switches and the dots can be regarded as tokens. A circle must send the dots to all of its output ports when it receives dots from all of its input ports. In other words, the switches are transparent when they get tokens from all of its input ports, and then producing new tokens to it all output ports.

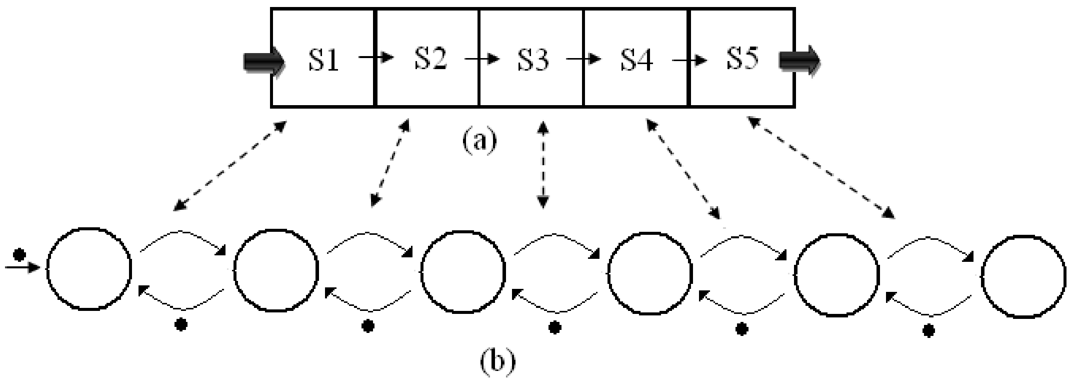


Figure 3.1 : (a) The data flow structure of a linear FIFO (b) The preliminary diagram for the control circuits of the linear FIFO

Let us analysis the preliminary diagram whether obeying the rule of the linear FIFO. Figure 3.2(a) shows the initial states of the preliminary diagram, and it must be properly set for correct operations. Figure 3.2(b) illustrates the first data item coming. When the first data item comes, the left-most circle, known as switch, produces a new token to it successor in figure 3.2(c). Then the second switch receives tokens from all of its input ports so that it becomes transparent. At this moment, the second switch produces new tokens to its predecessor and successor, and the second data item enters the linear FIFO in figure 3.2(d). The following operations are the same as above, the data items will move to the next stages in figure 3.2(e), and the operations will complete until all data items exit the FIFO.

3.2 Transforming Preliminary Control Circuits into Real Control Circuits

When the preliminary diagram is drawn and confirming that all operations are really correct, the next step is going to depict real GasP circuits by its preliminary control circuits. Figure 3.3(a) and (b) show the corresponding diagram between preliminary control circuits and real control circuits. In figure 3.3(a), the circled arrows are corresponded to the triangles of GasP modules in figure 3.3(b). If there is a dot beside an arrow, the corresponding triangle is shaded. Otherwise the triangle is unshaded. The output port of the circle in figure 3.3(a) is also corresponded to the output port of GasP modules in figure 3.3(b). Finally, the connections in figure 3.3(a) are relative to the corresponding connections in figure 3.3(b).

The optimal connections in figure 3.3(c) are on the basis of figure 3.3(b), it was mentioned in section 2.3. Finally, we can implement the real circuits in figure 3.3(d) with figure 3.3(c), and the procedures to translate data flow structure into GasP circuits are complete. In next chapter, we will use this method to implement our circuits instead of the FSM specifications.

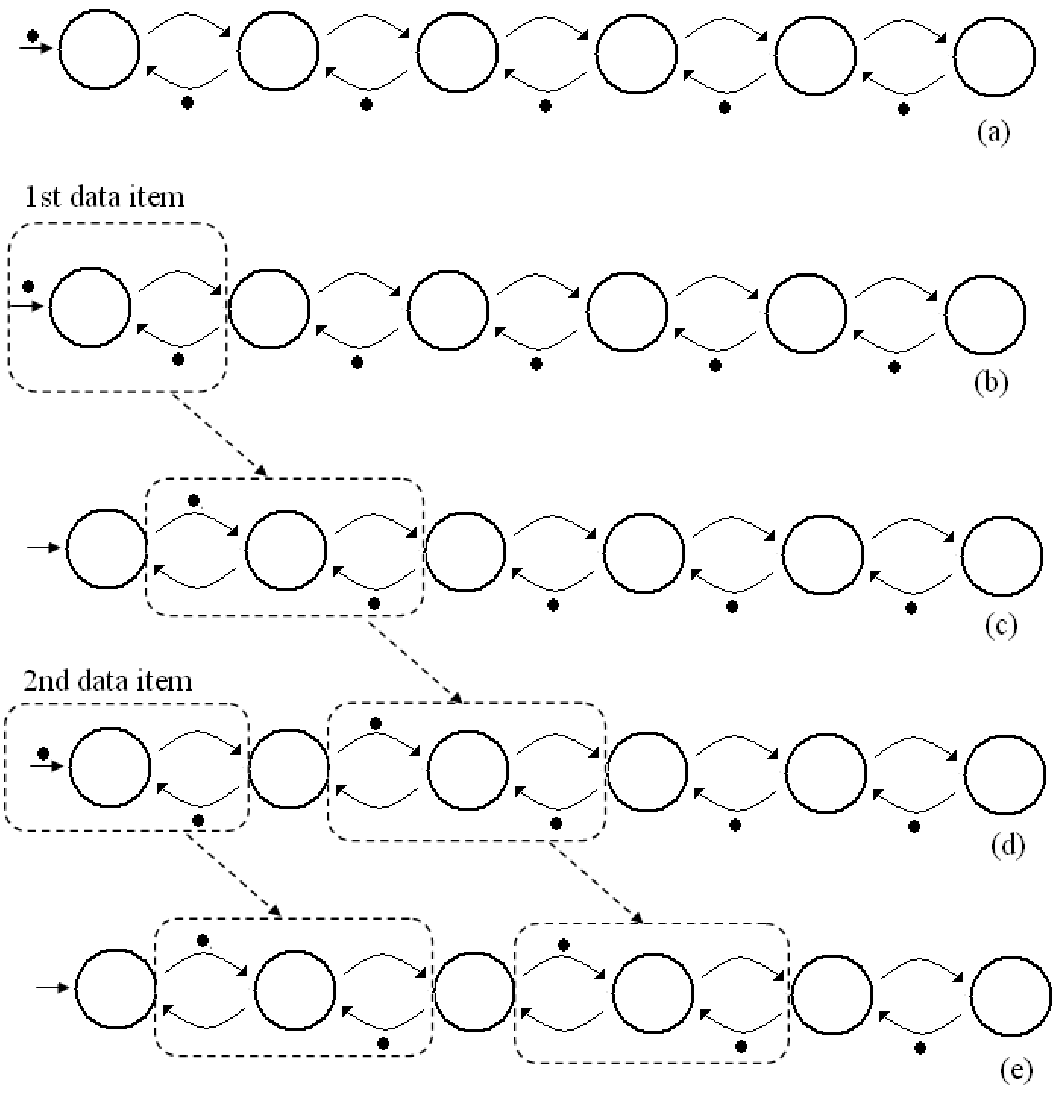


Figure 3.2 : The processing diagram for coming data items

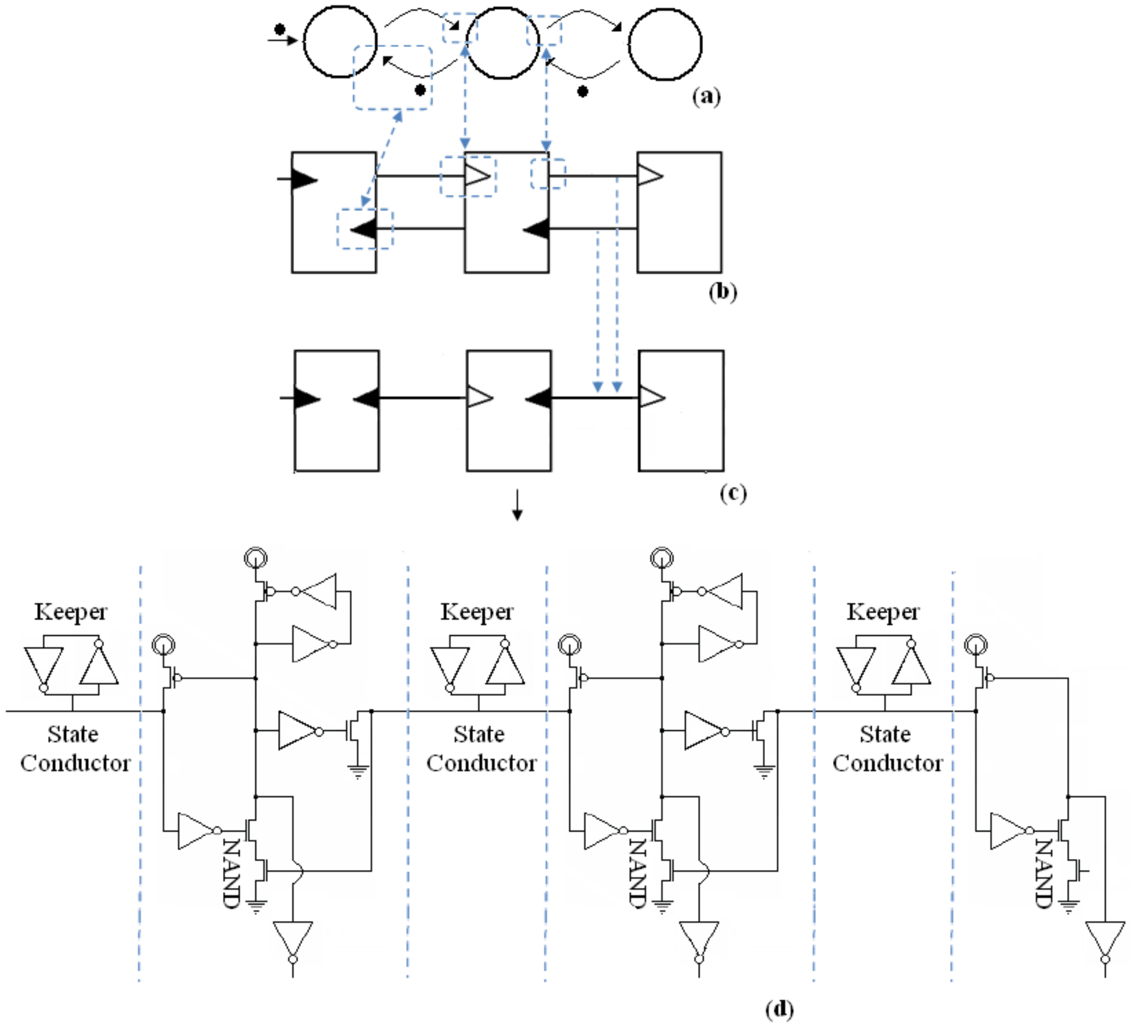


Figure 3.3 : The corresponding diagram between preliminary control circuits and real control circuits

Chapter 4 The 1-n-1 FIFOs

The first goal of this thesis is to reduce the data moves in the FIFO systems. This chapter will introduce how to reduce the data moves in the FIFO systems and use the presented method to implement our systems.

4.1 The Ideas about Reducing Data moves

The focal point for better power-efficiency is to reduce the data moves in systems. In synchronous systems, the clock signal triggered off all components to work even if they are unnecessary so that energy is consumed more. On the contrary, the asynchronous systems are just fired when the operations are actually necessary. The further focal point in asynchronous systems is to save more energy with different design of architectures. So the different kinds of architectures make different data moves leading to different power consumption.

Section 2.5 mentioned the brief introduction about these ideas, and 1-n-1 FIFOs are good choice for fewer data moves. From the viewpoint of algorithms, it is a better algorithm to reduce data moves because the structure provides constant data moves in FIFOs. For this reason, power consumption is also constant.

The following is the basis of the 1-n-1 FIFO, and we will construct our system from them. The same as the linear pipeline, we are going to use the method which was introduced in chapter 3 to implement our circuits.

4.2 The Basis of A 1-n-1 FIFO

In this section, we will introduce the basic operations for a 1-n-1 FIFO. In the beginning, the data flow structure and algorithms for branches and mergence are defined. Then we transform them into preliminary control circuits and verify whether they are correct or not. Finally, we translate the preliminary control circuits into GasP modules and the processes for GasP modules will be depicted step by step.

4.2.1 The Data flow Structure and Algorithms for Branches and Mergence

The basic operations for 1-n-1 FIFOs are branches and mergence. Now we will implement them with our procedures, and decide the data flow structure first. Figure 4.1 shows the data flow of branches and mergence. Many algorithms can achieve the behaviors, but the simplest one for them is using pointers to indicate which stage is the working one. We give a simple example to introduce how to use the pointers to make correct data flow. Figure 4.2 illustrates the pointers which point at the working stages. In figure 4.2(a), the pointer points at stage S2, so the first data item will pass through S2 and the sequence of the first data path are S1, S2, and S4. In figure 4.2(b), the pointer points at stage S3, therefore the second data item will go through S3 and its data path are S1, S3, and S4. The third data item is the same as the first one, the forth data item is the same as the second one, and so on.

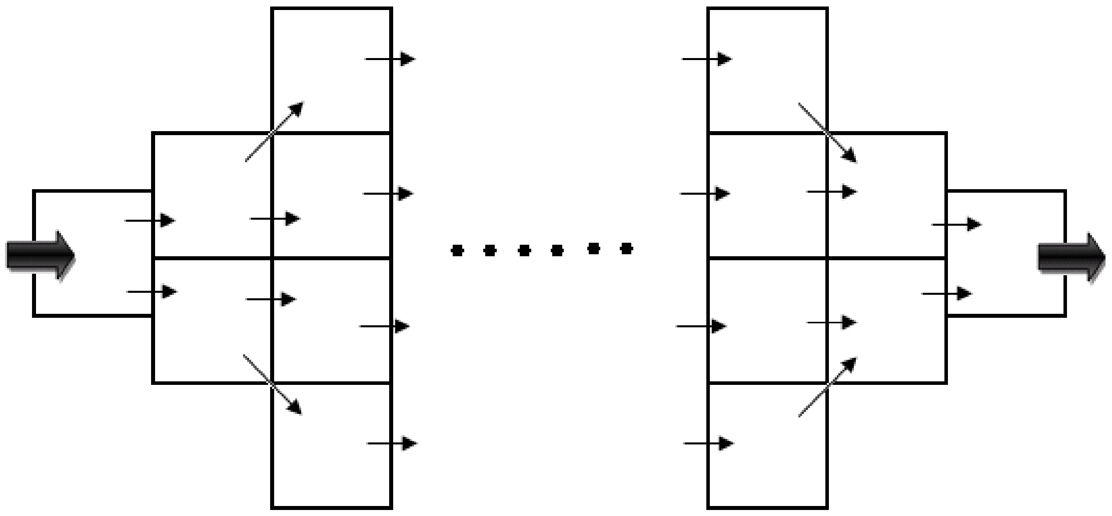


Figure 4.1 : The data flow for branches and mergence

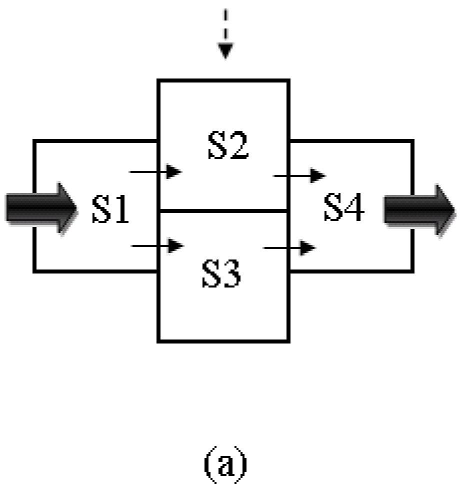


Figure 4.2 : The pointers for the data flow structure

4.2.2 Preliminary Control Circuits for Branches and Mergence

Since the data flow and its algorithms have been presented above, we are going to map the preliminary control circuits. Figure 4.3 is the preliminary control circuits for the stage S1 in figure 4.2 and the function of stage S1 is a branch. Figure 4.3(a), (b), and (c) illustrate the data item which is assigned to the upper switch, and then the pointer is passed to the lower switch. The next data item will goes through the lower switch in figure 4.3(d), (e), and (f), and the pointer returns to the upper switch.

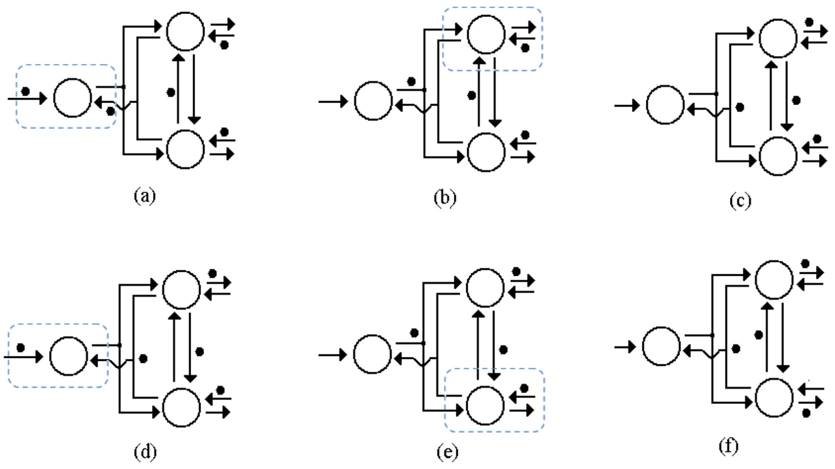


Figure 4.3 : The preliminary control circuits for a branch

Figure 4.4 is the preliminary control circuits for the stage S4 in figure 4.2 and its function is a mergence. In figure 4.4(a) and (b), the corresponding data item is received by upper switch and then the pointer is passed to the lower switch. Figure 4.4(c), (d), and (e) show that the next data item goes through the lower switch and the pointer returns to the uppers switch. Due to figure 4.3 and figure 4.4, we know that data items will pass through an 1-n-1 FIFO in the proper order.

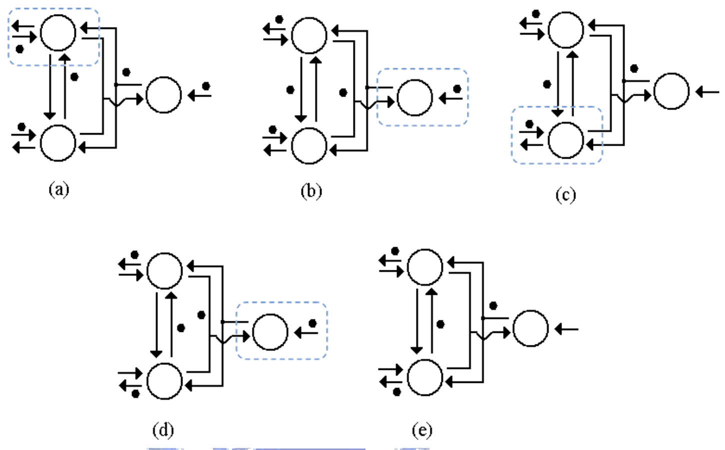


Figure 4.4 : The preliminary control circuits for the mergence

4.2.3 GasP Circuits for Branches and Mergence

In this section, we implement GasP circuits to branch and merge, and introduce their behavior clearly. One basic GasP module for these operations is drawn in figure 4.5 and it can achieve the round-robin fashion. Figure 4.5(a) shows a GasP module, the arrows indicate the direction of data flow, and the triangles mean self-resetting input ports. Hence “Pin a”, “Pin b”, and “Pin d” are self-resetting input ports, and “Pin c” is output port. In Figure 4.5(b) , cycle ”a → a1 → a2 → a” is self-resetting corresponding to “Pin a”, cycle ”b2 → b → b1 → b2” is self-resetting corresponding to “Pin b”, and ”d → d1 → d2 → d3 → d” is self-resetting corresponding to “Pin d” .

An input port of a GasP module is said to be set when it delivers a HI signal to the

NAND gate in the module; otherwise it is said to be reset. All GasP modules must be set or reset initially. In Figure 4.5(a), input port delivers a HI signal to the NAND when the triangles are shaded. Otherwise input port delivers a LO signal to the NAND when the triangles are unshaded. When all input ports are shaded, it means HI signals are delivered to all pins of the NAND gate in the module so that all output ports fire, and then self-resetting input ports are going to be unshaded. The details of GasP module in figure 4.5(a) is depicted in figure 4.5(b).

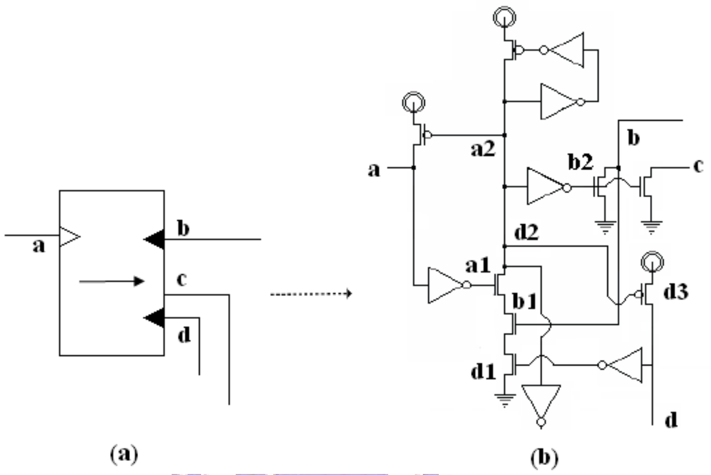


Figure 4.5 : A GasP module and its input and output ports

The GasP stage works in a round-robin fashion in figure 4.6 and it is composed of the GasP module in figure 4.5. Figure 4.6(a) shows all input ports of module M1 are set and module M1 is fired, and then left input ports of module M2 and module M3 are set in figure 4.6(b). In figure 4.6(b), all input ports of module M2 are set and the output port of module M2 are fired, and then HI signals are delivered to the NAND of module M3 in figure 4.6(c). When next data item is captured in module M1, it will go through module M3. The processes are shown in figure 4.6(d), (e), and (f) step by step. Hence modules M2 and M3 will be fired by turns. Figure 4.6 is relative to figure 4.3, we translate the preliminary control circuits in figure 4.3 into the GasP control circuits in figure 4.6. The details of modules M1, M2 and M3 are drawn in figure 4.7.

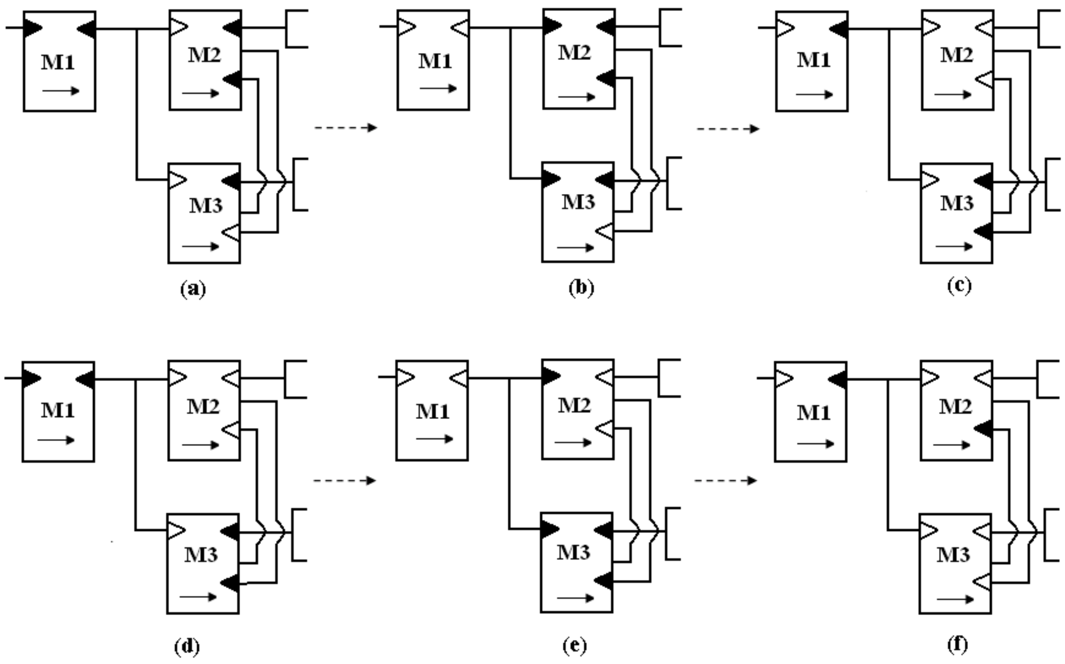


Figure 4.6 : The GasP control circuits for a branch

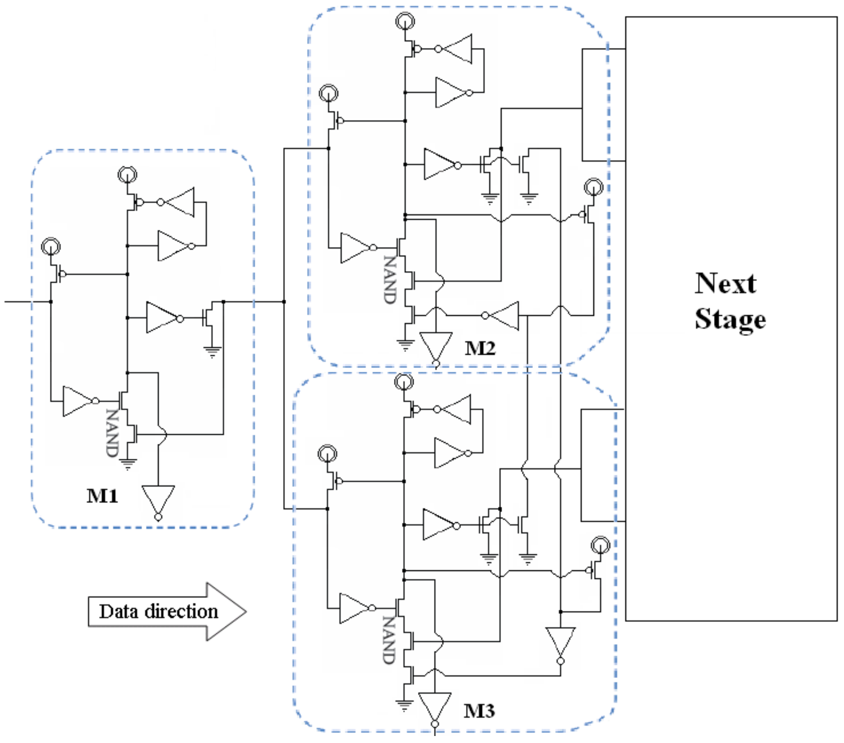


Figure 4.7 : The details of figure 4.6

Figure 4.8 is relative to figure 4.4, we also transform the preliminary control circuits in figure 4.4 into the GasP control circuits in figure 4.8. Figure 4.8(a) illustrates all input ports of module M1 are set and module M1 is fired, and then left input port of module M3 is set in figure 4.8(b). In figure 4.8(b), all input ports of module M3 are set and then it is fired, and then HI signals are delivered to the NAND of module M2 in figure 4.8(c). When next data item comes, it will pass through module M2 and M3. The processes are shown in figure 4.8(c), (d), and (e) step by step. So modules M1 and M2 send data items to module M3 by turns.

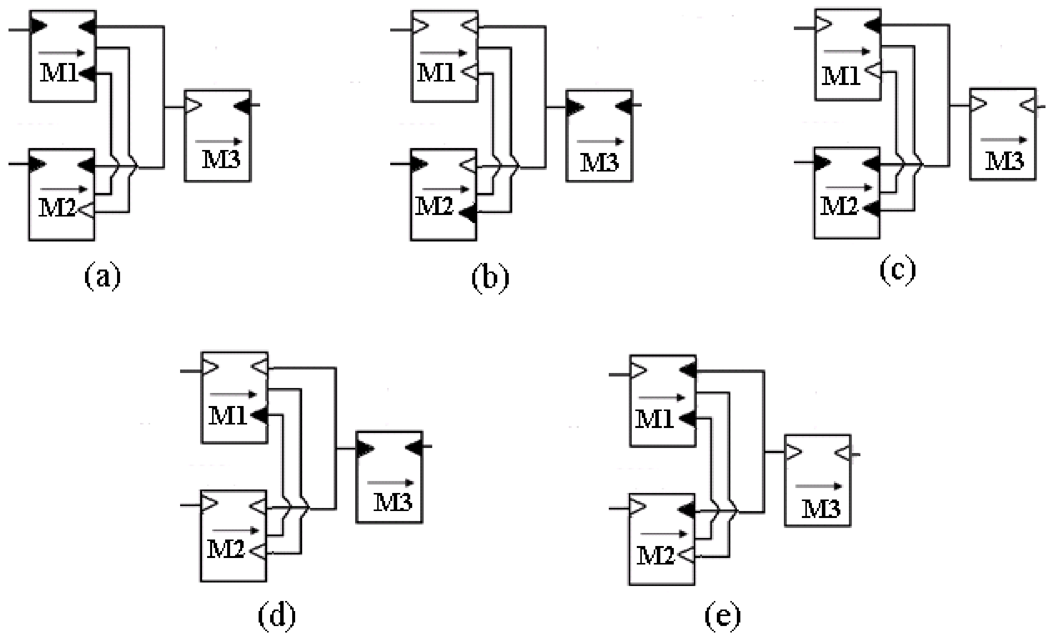


Figure 4.8 : The GasP control circuits for the mergence

Since the GasP control circuits for branches and mergence have been introduced above, we can use them to implement a binary tree FIFO. Figure 4.9 illustrates the preliminary control circuits for a binary tree FIFO with ten stages, and figure 4.10 shows the corresponding GasP control circuits. It has fewer data moves than square FIFOs and linear FIFOs. In addition to fewer data moves, it avoids some stages shouldering too heavy loading. In fact, we hope that the loading can be averagely shared by each stage, and it would be best that all stage have equal loading. Because of these reasons, the binary tree FIFO is a kind of better architecture.

The binary tree FIFO in figure 4.10 has ten stages which consist of GasP modules in figure 4.6 and figure 4.8. Data items go through the binary tree FIFO regularly because the GasP modules are built in a round-robin fashion. We list the data flow paths in figure 4.10 in the table 4.1 where the variable *m* is an integer. The data items can only route via four possible paths. The first data item goes through stages S1, S2, S4, S8, and S10; the second data item goes through stages S1, S3, S6, S9, and S10; the third data item passes through stages S1, S2, S5, S8, and S10; finally the forth data item passes through stages S1, S3, S7, S9, and S10. The routing path of fifth data item is the same as the first one, and so on.

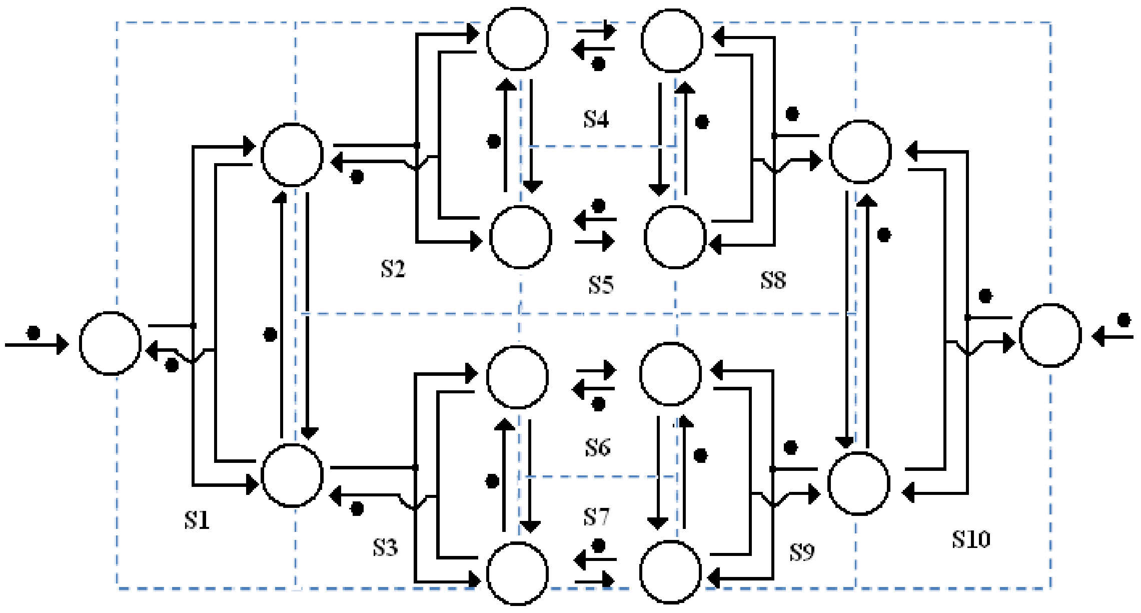


Figure 4.9 : The preliminary control circuits for a binary tree FIFO

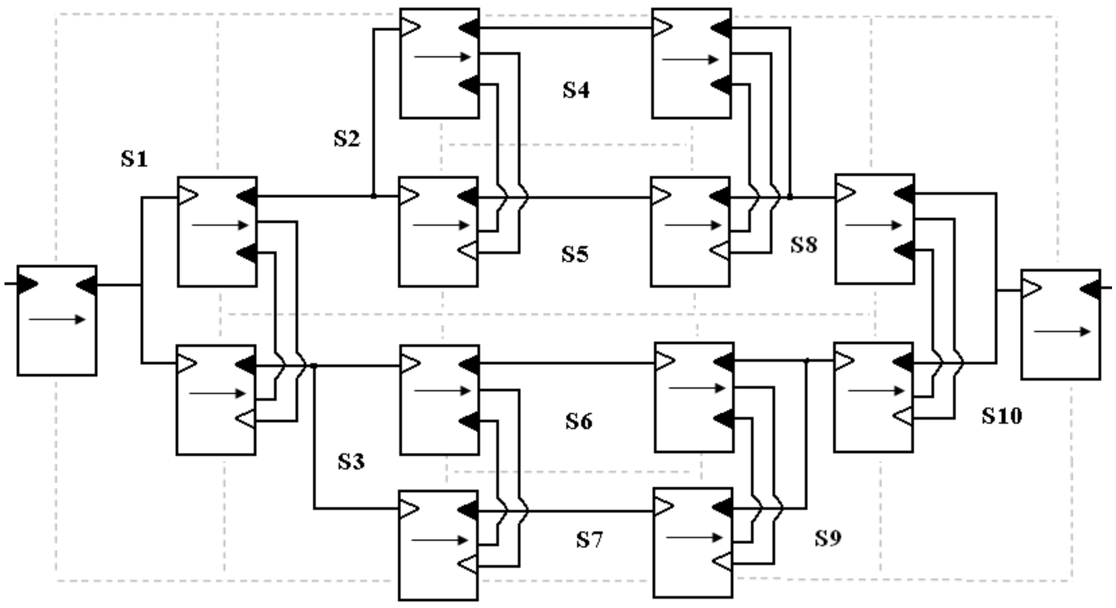


Figure 4.10 : The GasP control circuits for a binary tree FIFO

Table 4.1 : The paths of the binary tree FIFO in figure 4.9

The *k*th data item	Through stages from input port to output port
4*m* + 1st	S1→S2→S4→S8→S10
4*m* + 2nd	S1→S3→S6→S9→S10
4*m* + 3rd	S1→S2→S5→S8→S10
4*m* + 4th	S1→S3→S7→S9→S10

4.3 The 1-n-1 FIFO with nine stages

The timing complexity for the binary tree FIFO in figure 4.10 is O(logn). From the view of power-efficiency, we hope to find the best structure to achieve the goal. Therefore the 1-n-1 FIFO is our solution in our research.

The following is an example for the 1-n-1 FIFO with nine stages, and the data moves of the 1-n-1 FIFO are shown in figure 4.11. The 1-n-1 FIFO receives continuous data items from input port until it is overflow. Figure 4.11(a) shows the first data item is received from the input port of the 1-n-1 FIFO. Figure 4.11(b) illustrates the second data item is received and the first data item takes an internal move in the 1-n-1 FIFO, and so on. Its behavior is a round-robin fashion and the data items will pass through the FIFO in proper order. Figure 4.12 shows that all data items are sent out by the output port until nothing is in the FIFO. Figure 4.12(a) illustrates the first data item exits, and figure 4.12(b) shows the second data is sent out. At the same time, the third data item takes an internal move in the 1-n-1 FIFO, and so on. The algorithm of this data flow structure also uses pointers to point at which are the working stages. So we can translate the data flow structure into the preliminary control circuits in figure 4.13, and then implement the GasP control circuits in figure 4.14 by the preliminary control circuits.

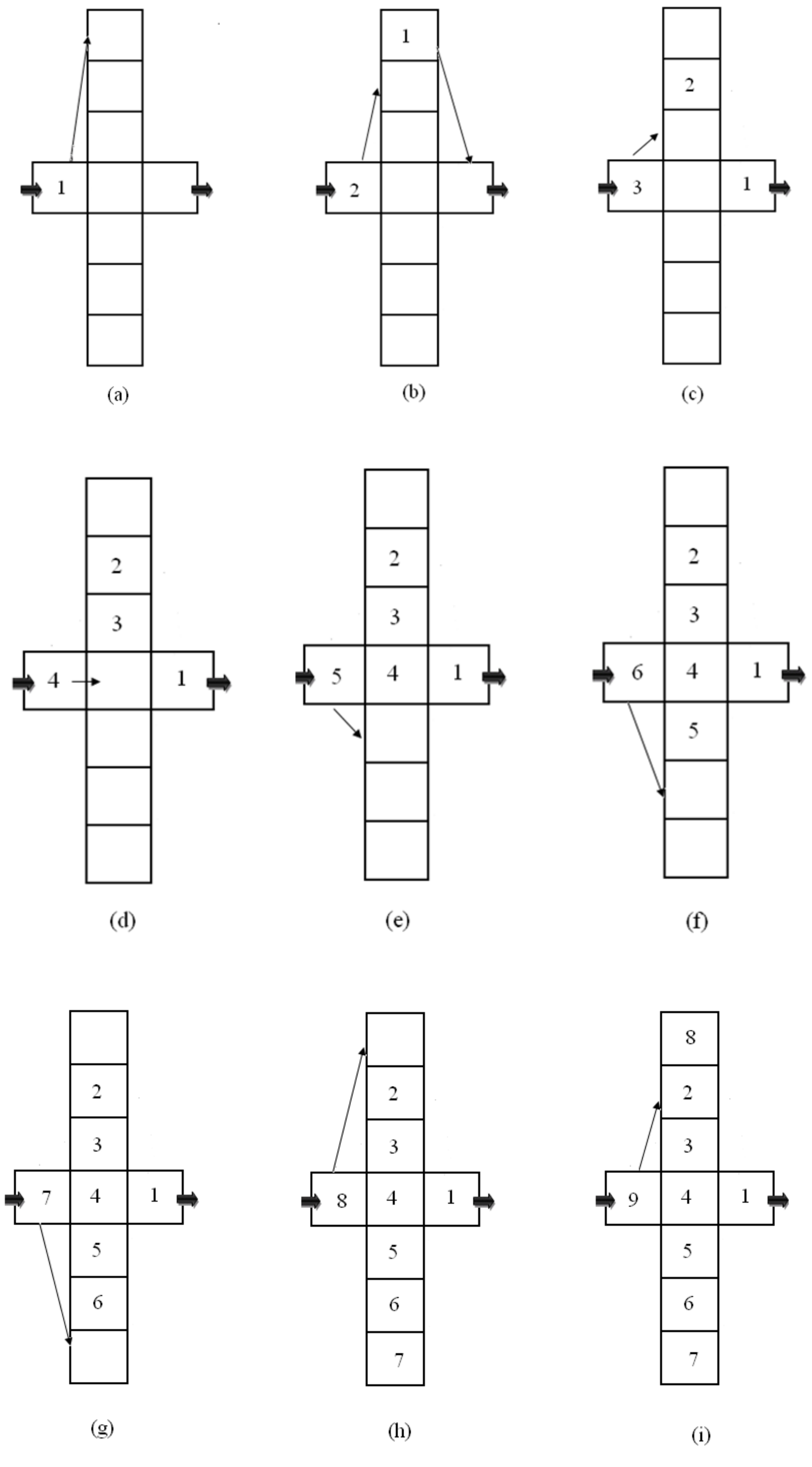


Figure 4.11 : Continuous data items gotten from the input port

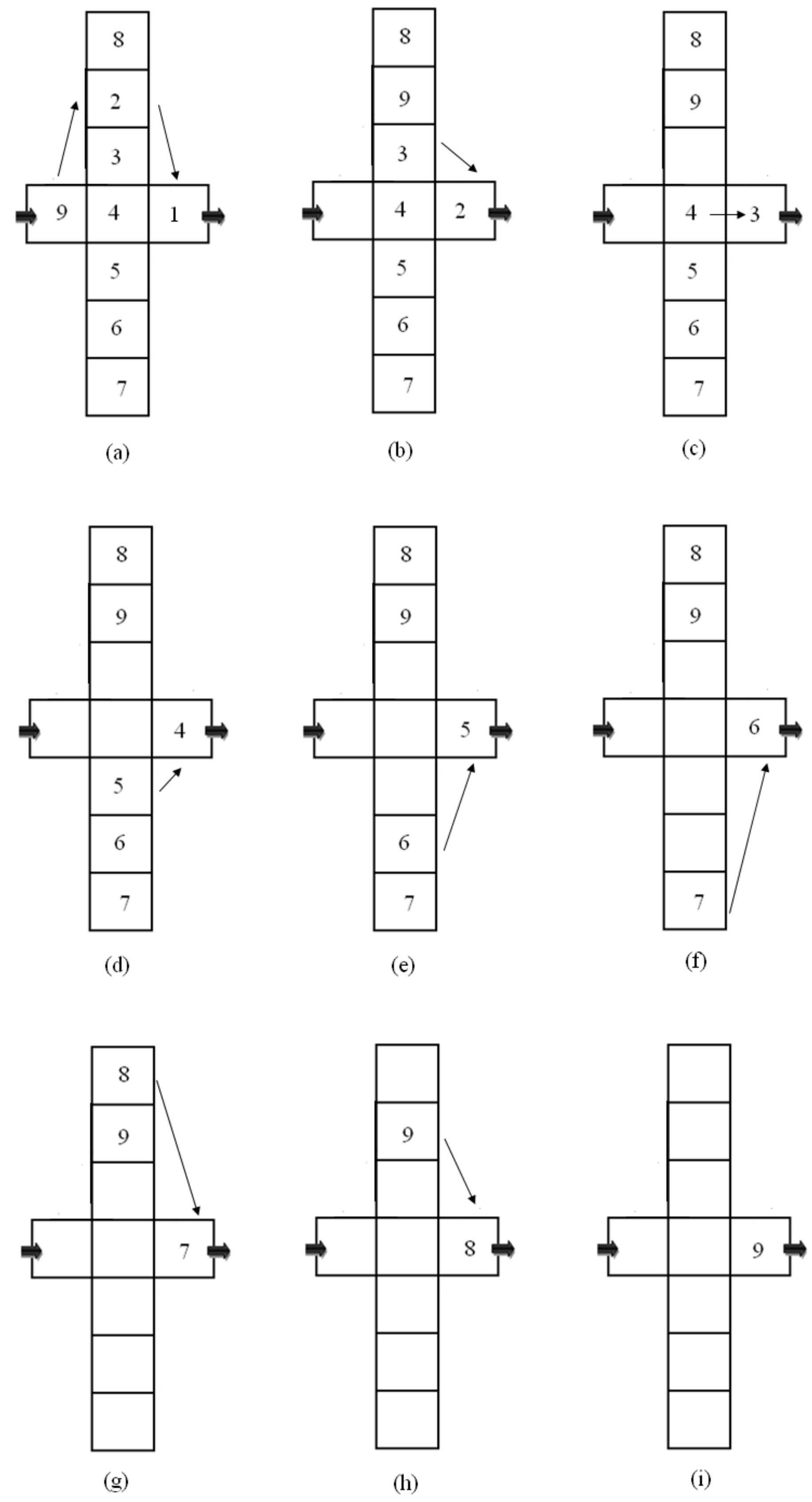


Figure 4.12 : Continuous data items sent out by the output port

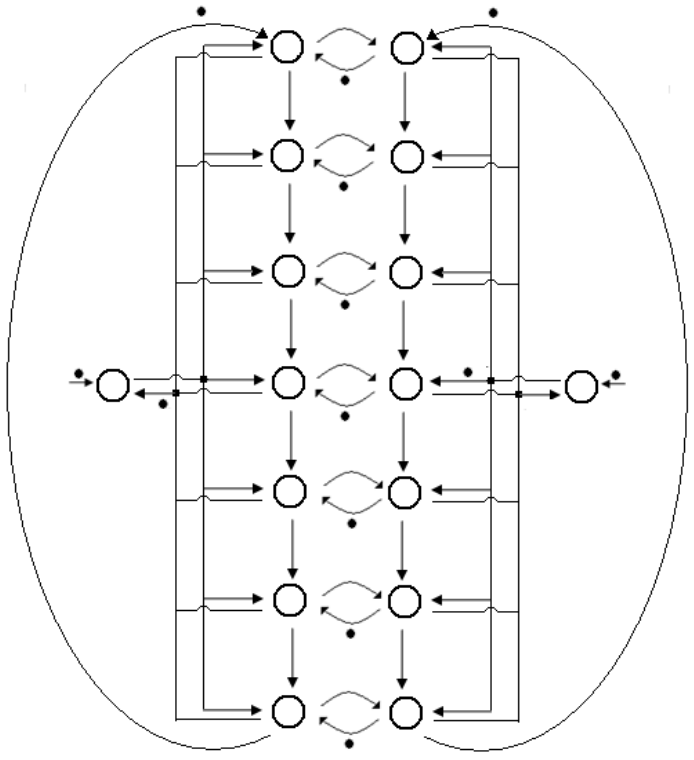


Figure 4.13 : The preliminary control circuits for the 1-n-1 FIFO

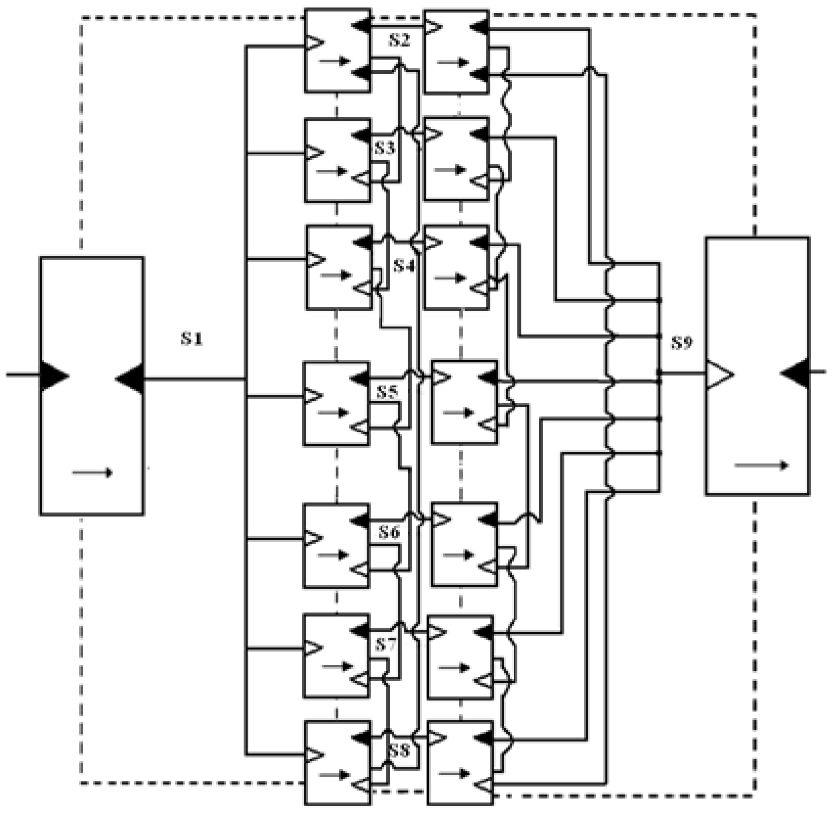


Figure 4.14 : The GasP control circuits for the 1-n-1 FIFO

4.4 Power-Estimation for Linear, Square, Binary Tree, and 1-n-1 FIFOs

Since all of the FIFOs have been introduced above, let us estimate their power-consumption and verify whether the 1-n-1 FIFOs have better efficiency. We present an idea to estimate power-consumption roughly; the idea is calculating how many GasP modules would work when a data item goes through the FIFOs.

First we want to know the average GasP modules per stage, and then computing how many stages will be passed when a data item goes though the FIFOs. Finally, we can calculate how many GasP modules work when a data item goes though the FIFOs. Because the linear, square, binary tree, and 1-n-1 FIFOs are both symmetric architecture, we can simply compute via above description and figures.

Linear FIFOs :

If the amount of stages in a linear FIFO is n, the total GasP modules in the linear FIFO are (n+1). So (n+1) / n are the average GasP modules per stage.

Because each data item will go through the linear FIFO via n stages, ((n+1) / n)*n= (n+1) is the number how many GasP modules will work when a data item goes though the linear FIFO.

Square FIFOs :

If the total number of stages in a square FIFO is n, the total GasP modules in square FIFO are $\sqrt{n}$ *($\sqrt{n}$ -1) + 2*($\sqrt{n}+\sqrt{n}$ -1), where $\sqrt{n}$ *($\sqrt{n}$ -1) are the number of column modules and $\sqrt{n}+\sqrt{n}$ -1 are top row of modules, and bottom row of modules are the same as top row of modules. So ($\sqrt{n}$ *($\sqrt{n}$ -1) + 2*($\sqrt{n}+\sqrt{n}$ -1)) / n = (n +3*$\sqrt{n}$ -2) / n are the average GasP modules per stage.

Because each data item will pass through the square FIFO via 2*$\sqrt{n}$ - 1 stages,

$(2*\sqrt{n} - 1)*((n + 3*\sqrt{n} - 2) / n)$ is the number how many GasP modules will work when a data item goes though the square FIFO.

Binary Tree FIFOs :

Suppose that data items are received from the root and its output port is the end point, the half height of a tree is h, then the total GasP modules in a binary tree FIFO are $(2^0+2^1+2^2\ldots2^{h-2} +2^{h-1}+2^{h-1}+2^{h-2}\ldots+2^2+2^1+2^0) = 2*(2^h-1)$, where $(2^0+2^1+2^2\ldots2^{h-2} +2^{h-1})$ are the number of half front modules and $(2^{h-1}+2^{h-2}\ldots+2^2+2^1+2^0)$ are the number of half back modules. The total number of stages in the binary tree FIFO are $(2^0+2^1+2^2\ldots2^{h-2}+2^{h-1}+2^{h-2}\ldots+2^2+2^1+2^0) = 2*(2^{h-1}-1) + 2^{h-1}$. So $2*(2^h-1) / (2*(2^{h-1}-1) + 2^{h-1}) = (2^h-1) / (2^{h-1}+2^{h-2}-1)$ are the average GasP modules per stage. Because each data item will pass through the binary tree FIFO via $2*h- 1$ stages, $(2*h- 1)* (2^h-1) / (2^{h-1}+2^{h-2}-1)$ are the number how many GasP modules will work when a data item goes though the binary tree FIFO.

1-n-1 FIFOs :

The following is the 1-n-1 FIFOs. If the number of stages is n, the total GasP modules are $(1+2*(n-2)+1)=2*n-2$. So $(2*n-2)/n$ are the average GasP modules per stage. Because each data item will go through the 1-n-1 FIFO via 3 stages, $3*((2*n-2)/n)$ is the number how many GasP modules will work when a data item goes though the 1-n-1 FIFO.

Table 4.2 lists all equations which were computed above, and then we calculate the number how many GasP modules will work when a data item goes through FIFOs in figure 4.15. Obviously the increasing ratio of average working GasP modules in the 1-n-1 FIFOs is lesser than the other FIFOs, especially when the numbers of stages are more than twenty-five. The main reason is the complexity of 1-n-1 FIFOs are O(1), tree FIFOs are O(log$n$), square FIFOs are O($\sqrt{n}$), and linear FIFOs are O(n). As our expectancy, 1-n-1 FIFOs have the better beneficial result and its power-comparison is almost close to a constant.

Table 4.2 : Equations for average working GasP modules per data item

Types of FIFOs	Average working GasP modules per data item
Linear FIFOs	n+1
Square FIFOs	$(2*\sqrt{n} - 1)*((n + 3*\sqrt{n} - 2) / n)$
Binary Tree FIFOs	$(2*h - 1)*(2^{h}-1) / (2^{h-1}+2^{h-2}-1)$
1-n-1 FIFOs	$3*((2*n-2)/n)$

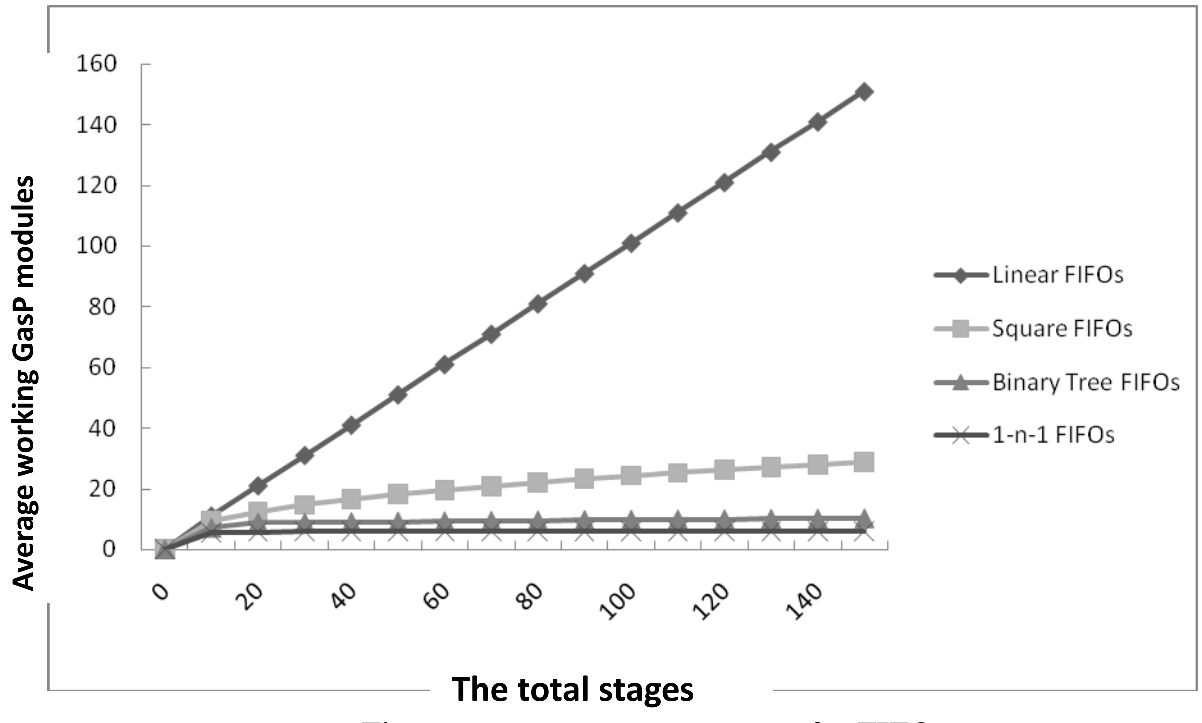


Figure 4.15 : Power-comparison for FIFOs

Chapter 5 Experimental Results

In this chapter, we implemented the FIFOs in transistor level and simulated them with the TSMC 180nm process. The simulator we used is HSPICE 2005.03. The experimental targets are introduced above. They are synchronous FIFOs in figure 2.1, linear FIFOs with standard NAND in figure 2.5, square FIFOs in figure 2.11, binary tree FIFOs in figure 4.10, and the 1-n-1 FIFOs in figure 4.14. We separately experimented on them and the depth of those FIFOs is respectively ten and eighteen stages which have one bit wide data storage.

5.1 The Cycle Time of These FIFOs

In order to analyze the performance of these FIFOs, we need to simulate them in the same conditions. The cycle time is the essential factor to determine how many data items could be received per second. Table 5.1 lists the cycle time of GasP circuits, it is obvious that the linear FIFOs are the fastest because its circuits are the simplest and each GasP module just connects to its successor and predecessor.

On the contrary, the 1-n-1 FIFOs are the slowest, its cycle time is more 1.5~1.6 times more than the linear FIFOs'. That is because the 1-n-1 FIFOs have a serious problem that the first stage and the last stage have too heavy loading. Its right self-resetting input ports of the first module has acute fan-out problem and left self-resetting input ports of the last module also has critical fan-out problem. It becomes the bottleneck of these circuits.

The secondary reason for the long cycle time is illustrated in figure 5.1. Figure 5.1(a) shows that the module M2 will be fired, and then it sends tokens to module M1 and M3 via path 1 and 2. When the path 1 is shorter than path 2, in the other words, the module M1 receives the token earlier and the operation will be correct as shown in figure 5.1(b). Otherwise if the path 2 is shorter than path 1, the operation will be wrong as shown in figure 5.1(c). In order to satisfy this timing constraint, the transistor size must be adjusted so that the cycle time is longer.

In order to let these FIFOs receive data in the same frequency, the faster cycle time needs to match up the slower one. The cycle time of the 1-n-1 FIFOs become the standard for our experiment. We let the GasP circuits receive 3 billion data items per second (333.33ps per data item), and the clock signal of the synchronous FIFOs is triggered at 3GHz in our simulation.

Table 5.1 : The cycle time of the FIFOs

Stages	**Linear FIFO**	**Square FIFO**	**Binary Tree FIFO**	**1-n-1 FIFO**
10	204.73ps	216.98ps	237.54ps	312.95ps
	100%	105.98%	116.03%	152.86%
18	205.54ps	228.52ps	236.90ps	332.25ps
	100%	111.18%	115.26%	161.65%

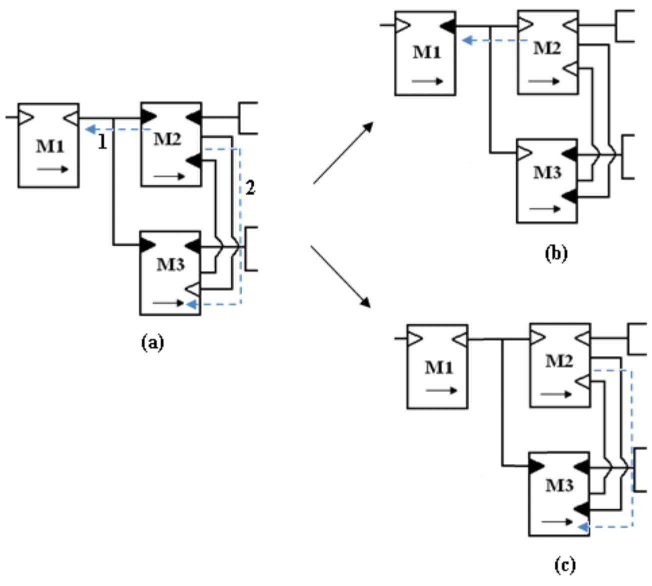


Figure 5.1: The timing constraint for 1-n-1 FIFOs

5.2 The Power Consumption of These FIFOs

In this section, we compare the power consumption between these FIFOs. We respectively experiment on three kinds of different conditions in order to analyze the result accurately.

The first condition :

We only compare the GasP control circuits with 3 billion request signals per second in order to avoid the different data patterns causing the discrepant results. In table 5.2, we can find that the lowest power consumption is 1-n-1 FIFOs regardless of the FIFOs' depth. When the depth of FIFO is ten stages, the linear FIFO consumes more 17.6% energy than the 1-n-1 FIFO and the square FIFO consumes even more 67.26% energy. When the FIFOs' depth is eighteen stages, the power consumption of the linear FIFO is 176.44% to the 1-n-1 FIFO and the square FIFO is 201.06% to the 1-n-1 FIFO. Moreover, the power consumption of binary tree FIFOs and 1-n-1 FIFOs is almost near because the number of stages is too small to reveal their difference.

Table 5.2 : 3 billion request signals per second (only GasP control circuits)

Stages	Linear FIFO	Square FIFO	Binary Tree FIFO	1-n-1 FIFO
10	7.9285mW	11.277mW	7.0327mW	6.7421mW
	117.60%	167.26%	104.31%	100%
18	13.830mW	15.760mW	8.4317mW	7.8385mW
	176.44%	201.06%	107.58%	100%

The second condition :

We compare the GasP full circuits and the synchronous FIFOs with 3 billion monotonous data items per second in table 5.3, the monotonous data items mean a series of 0 or 1 as input data pattern. These data patterns are gainful for synchronous FIFOs because they are implemented by simple flip-flops and there is none of control circuits except for the clock signal in synchronous FIFOs. A series of 0 or 1 make lower switching power consumption for data paths.

When the FIFOs' depth is ten stages, the synchronous FIFO has the lowest power consumption, and the 1-n-1 FIFO consumes 12.89% more than it because of lower switching power consumption and the simplest circuits. When the FIFOs' depth is eighteen stages, the 1-n-1 FIFO has the lowest power consumption, and the other FIFO consumes at least 22.32% more because of the fewer data moves are advantageous for it. In particular, the square FIFOs consumes the most energy whether the FIFOs' depth is ten or eighteen stages. That is because the algorithm of square FIFOs is so complex and thus its control circuits are implemented with many GasP modules. These modules consume much more energy, especially the modules of the top and bottom rows which have the most complicated control paths.

Table 5.3 : 3 billion monotonous data items per second (Full circuits)

Stages	**Linear FIFO**	**Square FIFO**	**Binary Tree FIFO**	**1-n-1 FIFO**	**Synchronous FIFO**
10	8.2398mW	16.104mW	8.0904mW	7.2038mW	6.381mW
	129.13%	252.37%	126.79%	112.89%	100%
18	17.524mW	18.437mW	9.9847mW	9.4003mW	11.498mW
	186.42%	196.13%	106.22%	100%	122.32%

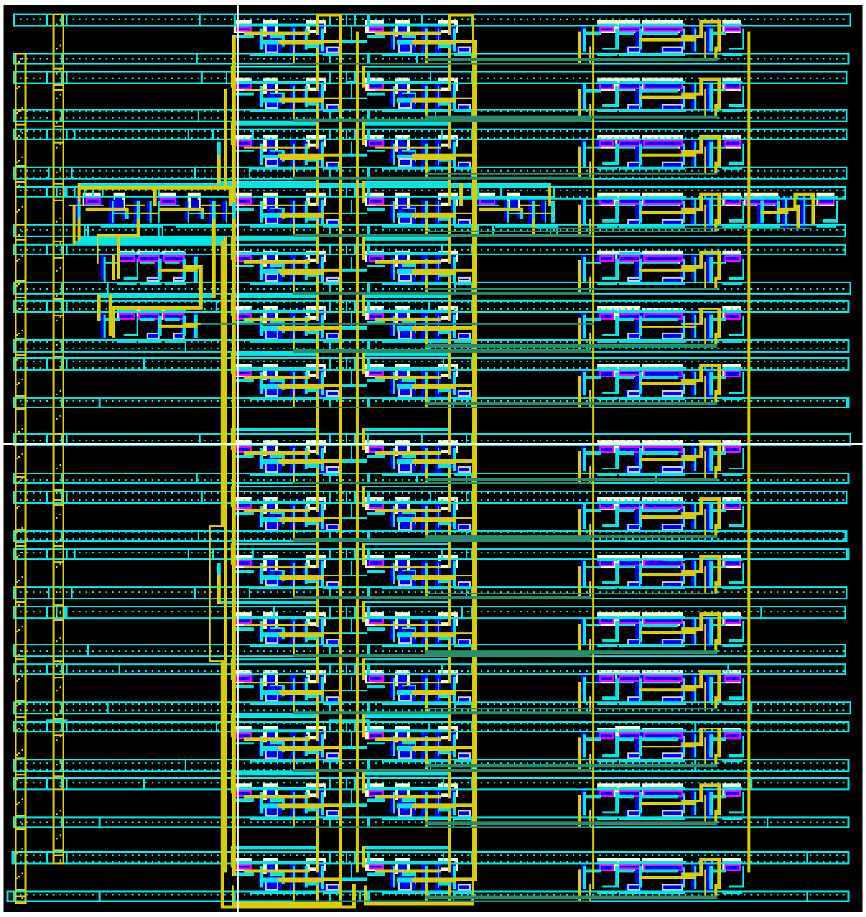
The third condition :

This condition is used to compare with the second condition. The test patterns in this condition are various, and the switching power consumption for data paths would be normalized. In table 5.4, we can find that the 1-n-1 FIFOs have the better results than the others because of the constant data moves. Through the comparisons, we can find that the synchronous FIFO with ten stages is 105.99% to the 1-n-1 FIFO and the linear FIFO needs even more 78.91% than the 1-n-1 FIFO when the depth of FIFOs is eighteen. Besides, the difference between linear FIFOs and square FIFOs become less when the depth of stages becomes larger because the square FIFOs have fewer data moves. This advantage can overcome the effect of the complex control circuits of square FIFOs.

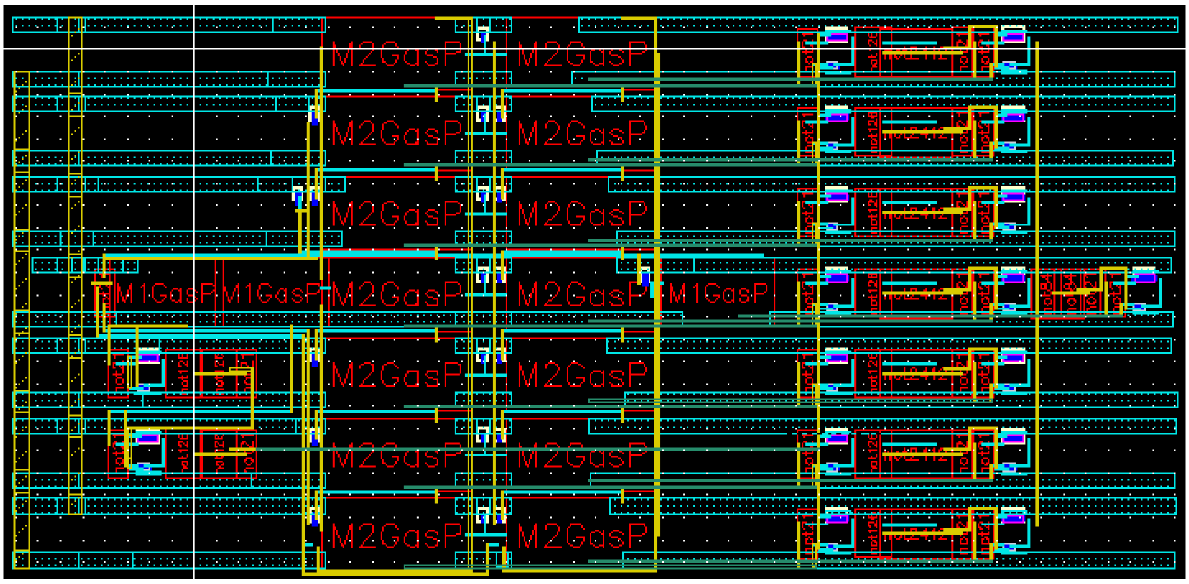
Table 5.4 : 3 billion various data items per second (Full circuits)

Stages	**Linear FIFO**	**Square FIFO**	**Binary Tree FIFO**	**1-n-1 FIFO**	**Synchronous FIFO**
10	10.238mW	17.083mW	9.0452mW	7.8242mW	8.2929mW
	130.85%	218.34%	109.07%	100%	105.99%
18	17.646mW	19.959mW	10.282mW	9.8628mW	14.865mW
	178.91%	202.37%	104.25%	100%	150.72%

In order to get more accurate result, we implement the layouts of 1-n-1 FIFOs in figure 5.2. Figure 5.2(a) and (b) show the 1-n-1 FIFOs with eighteen and ten stages respectively. Thus we can obtain more accurate results via these layouts. Finally, we make a summary of these kinds of different conditions. The 1-n-1 FIFOs almost have the best results, and their power consumption has one time improvement more than the square FIFOs. The predominance is more obvious when the depth of FIFOs becomes larger.



(a)



(b)

Figure 5.2 : The layouts of 1-n-1 FIFOs (a) eighteen stages (b) ten stages

5.3 The Transistors Counts

We compare the transistors counts between these FIFOs. In table 5.5, the synchronous FIFOs and the linear FIFOs are nearly equal, and they have the fewest transistors. The square FIFOs have better results than 1-n-1 FIFOs because the columns of square FIFOs are linear FIFOs so that it can reduce the counts of transistors.

The transistors of 1-n-1 FIFOs are 150%~170% to synchronous FIFOs and linear FIFOs because the middle stages of 1-n-1 FIFOs need a lot of transistors to implement the algorithm. It uses pointers to record the working switches so that most of transistors in 1-n-1 FIFOs are used for it.

Table 5.5 : The transistors counts

Stages	**Linear FIFO**	**Square FIFO**	**Binary Tree FIFO**	**1-n-1 FIFO**	**Synchronous FIFO**
10	287	320	348	443	280
	102.5%	114.29%	124.29%	158.21%	100%
18	503	602	571	837	504
	100%	119.68%	113.52%	166.07%	100.2%

Chapter 6 Conclusions and Future Works

Here is a summary of the most important conclusions in this thesis. (1) We explain a method on how to translate data flow structure into corresponding GasP circuits. The method has two steps: the first step is to transform a data flow structure into preliminary control circuits, and the second step is to depict real GasP circuits by its preliminary control circuits. Then we explain it with an example of constructing a linear FIFO. (2) We tried to find some architecture of FIFOs that have high-throughput and fewer data movements to be power-efficient. Thus we propose the 1-n-1 FIFOs with fewest data moves. (3) We present an idea to estimate power-consumption that calculates how many GasP modules would work when a data item goes through FIFOs. We also show the estimation results of linear, square, binary tree, and 1-n-1 FIFOs. Finally, we prove them via simulations, and the 1-n-1 FIFO has one time improvement more than the square FIFOs. But the transistors of the 1-n-1 FIFO are also more than the other FIFOs. It is approximately more than 60%.

Some FIFOs in this thesis do not include the wire delay information, and it may cause the results a little imprecise. Therefore the future work is to obtain that more precise results from simulations can be gotten with the layout of the designs, and more extension of tree FIFOs can be simulated in order to get more results to compare.

The other future work is the optimized problem. It can be used to determine how many degrees for each stage will be the most power-efficient. If we just consider the switching counts in those FIFOs, the 1-n-1 FIFO is the most power-efficient. However if we consider switching counts and the capacitance, the different processes will make different results. Therefore the optimized problem in different processes is a tough question.

References

[1] B. Coates, J. Ebergen, J. Lexau, S. Fairbanks, I. Jones, A. Ridgway, D. Harris, and I. Sutherland, "A Counterflow Pipeline Experiment," Advanced Research in Asynchronous Circuits and Systems, 1999. Proceedings., Fifth International Symposium on 19-21, April 1999, pp.161-172.

[2] J. Lexau, J. Will, and I. Jones "EM emissions of an asynchronous test chip," Proceedings of the 14th Int'l Zurich Symposium & Technical Exhibition of Electromagnetic Compatibility, February 2001, pp. 579-584.

[3] S. Hauck, "Asynchronous design methodologies: an overview," Proceedings of the IEEE, Vol. 83, Issue 1, Jan. 1995, pp.69-93

[4] J. Sparsø and S. Furber, Principles Of Asynchronous Circuit Design A Systems Perspective, Kluwer Academic Publishers, London, 2001.

[5] I. Sutherland, "Micropipelines," Communications of the ACM, vol .32, no. 6, June 1989, pp. 720-738.

[6] Jen-Chien Wu, "Decoder design of the asynchronous PIC microcontroller," National Chiao Tung University, 2006

[7] T.E. Williams, "Self-Timed Rings and their Application to Division," Stanford University, June 1991.

[8] M. Singh and S. Nowick, "High-throughput asynchronous pipelines for fine-grain dynamic datapaths," Advanced Research in Asynchronous Circuits and Systems, 2000. (ASYNC 2000) Proceedings. Sixth International Symposium on 2-6, April 2000, pp. 198-209.

[9] M. Singh and S. Nowick, "Fine-grain pipelined asynchronous adders for high-speed DSP applications," VLSI, 2000. Proceedings. IEEE Computer Society Workshop on 27-28, April 2000, pp. 111-118.

[10] I. Sutherland and S. Fairbanks, "GasP: A Minimal FIFO Control," Asynchronous Circuits and Systems, 2001. ASYNC 2001. Seventh International Symposium on 11-14, March 2001, pp. 46-53

[11] M. Ferretti, P.A. Beerel, "Single-track asynchronous pipeline templates using 1-of-N encoding Design," Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings 4-8, March 2002, pp. 1008-1015.

[12] P. Golani and P. Beerel, "High-performance noise-robust asynchronous circuits," In Proceedings of the 2006 Emerging VLSI Technologies and Architectures (ISVLSI06), 2006, pp. 6.

[13] C.E. Molnar, I.W. Jones, W.S. Coates, and J.K Lexau, "A FIFO ring performance experiment," Advanced Research in Asynchronous Circuits and Systems, April 1997, pp. 279-289.

[14] C.E. Molnar, I.W. Jones, W.S. Coates, J.K Lexau, S.M. Fairbanks, and I.E. Sutherland, “Two FIFO ring performance experiments,” Proceedings of the IEEE, Feb. 1999, pp. 297-307.

[15] I.E. Sutherland and J.K. Lexau, “Designing fast asynchronous circuits,” Asynchronous Circuits and Systems, 2001. ASYNC 2001 Seventh International Symposium on 11-14, March 2001, pp. 184-193.

[16] J. Ebergen, P. Cunningham, and J. Gainsley, “Transistor sizing with logical effort: How to control the speed and energy consumption of a circuit,” In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems.IEEE Computer Society Press, 2004.

[17] J. Ebergen, “Squaring the FIFO in GasP,” Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, March 2001, pp. 194-205.

[18] J. Ebergen, D. Finchelstein, R. Kao, J. Lexau, and D. Hopkins,”A fast and energy-efficient stack,” Asynchronous Circuits and Systems, 2004. Proceedings. 10th International Symposium on 19-23, April 2004, pp. 7-16.