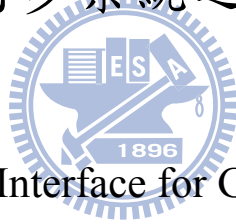


# 國立交通大學

## 資訊科學與工程研究所

### 碩 士 論 文

全域非同步區域同步系統之兩階段握手協定介面



Two-Phase Handshaking Interface for Globally Asynchronous Locally

Synchronous Systems

研 究 生：陳俊瑋

指 導 教 授：陳昌居 教授

中 華 民 國 九 十 八 年 七 月

全域非同步區域同步系統之兩階段握手協定介面

Two-Phase Handshaking Interface for Globally Asynchronous Locally  
Synchronous Systems

研究生：陳俊瑋

Student : Chun-Wei Chen

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



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

July 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年七月

# 全域非同步區域同步系統之兩階段握手協定介面

研究生：陳俊瑋

指導教授：陳昌居

國立交通大學資訊工程學系碩士班

## 摘要

現今大部分的數位系統還是以同步電路的方式來設計。但如同我們所知，隨著電路複雜度的增加，產生的問題也愈來愈多，如時脈時滯和功率消耗的問題。此外，SoC(系統單晶片)是現今的另一種設計趨勢，要將一個 SoC 設計中的多個 IP(知識產權)模組做整合並不是那麼的容易。所以，為了解決上述問題，GALS(全域非同步區域同步)是一個可以被期待的設計方式。

相較於傳統四階段握手協定、可延展時脈的 GALS 系統而言，我們提出了適用於可延展時脈 GALS 系統的兩階段握手協定介面。在我們的設計中，區域的同步模組可以有各自不同的時脈，並且可以正確的運算。並將這個新設計用 Synopsys Design Compiler 來做合成，使用的是 TSMC 0.13 微米的元件資料庫。

最後得到的結果顯示，我們提出的兩階段握手協定的新設計比起四階段握手協定的設計來說，有比較短的延遲時間；但如果就面積的觀點來看，兩階段握手協定的新設計所佔的面積要比四階段握手協定的設計來得大。

# Two-Phase Handshaking Interface for Globally Asynchronous Locally Synchronous Systems

**Student: Chun-Wei Chen**

**Advisor: Dr. Chang-Jiu Chen**

## Abstract

Most modern digital systems are based on synchronous circuit design nowadays. But as we know, with the increasing complexity of digital circuits, there are some problems such as clock skew and power consumption. In addition, system on chip (SoC) design is another trend today. To integrate several intellectual property (IP) modules in a SoC design is not an easy job. Globally asynchronous locally synchronous (GALS) design is a promising approach to solve these problems.

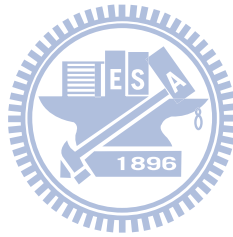
Compared with traditional four-phase handshaking, stretchable clocking based GALS systems, we propose a two-phase handshaking interface for stretchable clocking based GALS systems. In our design, the local synchronous modules can operate at different clock frequencies independently and work correctly. The design is synthesized with Synopsys Design Compiler with TSMC 0.13 $\mu$ m cell library.

The result shows that the new two-phase handshaking design has better latency than four-phase handshaking counterpart. But from the viewpoint of area, the new two-phase handshaking design is larger than four-phase handshaking counterpart.

# Acknowledgements

這篇論文能夠完成，首先要感謝陳昌居老師兩年來的指導，再來是口試委員鄭福炯、范倫達老師的寶貴建議，非常感謝。還要謝謝實驗室的學長、同學、學弟兩年來不論是在研究或生活上的幫助，在我趕火車時能載我到火車站，謝謝大家。另外要謝謝我的朋友們，每次與你們聊天都彷彿又回到當年一樣。最後要感謝兩年中幫助過我的許多人，謝謝。

當然，最重要的是家人的付出與支持，真的非常感謝你們。



# Contents

摘要 .....	i
Abstract .....	ii
Acknowledgements .....	iii
Contents .....	iv
List of Figures .....	vi
List of Tables .....	viii
Chapter 1 Introduction .....	1
1.1 Motivations.....	1
1.2 Asynchronous Circuits .....	2
1.2.1 Handshake Protocols.....	3
1.2.2 Data Encoding Methods.....	5
1.2.3 Muller C-Element.....	7
1.2.4 Advantages and Drawbacks.....	9
1.3 Organization of The Thesis.....	10
Chapter 2 Related Works.....	12
2.1 Overview.....	12
2.2 Asynchronous Wrapper .....	12
2.3 Pausible-Clock Generators .....	15
2.3.1 Pausible Clocking .....	15
2.3.2 Stretchable Clocking.....	18
2.3.3 Data-Driven Clocking .....	20
2.3.4 Comparisons Between Three Clock Generation Methods.....	21
2.4 Four-Phase Handshaking, Stretchable Clocking Based GALS Systems .....	22

2.4.1 Input Port .....	26
2.4.2 Output Port .....	28
2.4.3 Gate-Level Simulation.....	30
<b>Chapter 3 A New Design - Two-Phase Handshaking Interface for Stretchable Clocking Based GALS Systems .....</b>	<b>32</b>
3.1 Input Port .....	40
3.2 Output Port .....	41
3.3 Gate-Level Simulation.....	43
<b>Chapter 4 Results and Analysis .....</b>	<b>45</b>
4.1 Synthesis Result - Area.....	45
4.2 Synthesis Result - Latency .....	46
4.3 Adjacent Wrappers Operate at Different Clock Frequencies .....	47
<b>Chapter 5 Conclusions and Future Work .....</b>	<b>51</b>
<b>References.....</b>	<b>53</b>



# List of Figures

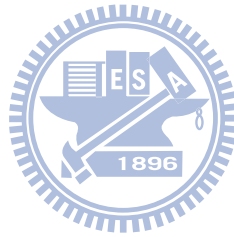
<b>Fig. 1</b>	<b>An abstract diagram of GALS systems</b> .....	<b>2</b>
<b>Fig. 2</b>	<b>Illustration of asynchronous systems</b> .....	<b>3</b>
<b>Fig. 3</b>	<b>Four-phase handshake protocol</b> .....	<b>4</b>
<b>Fig. 4</b>	<b>Two-phase handshake protocol</b> .....	<b>5</b>
<b>Fig. 5</b>	<b>Dual-rail data encoding systems</b> .....	<b>7</b>
<b>Fig. 6</b>	<b>Four-phase dual-rail protocol</b> .....	<b>7</b>
<b>Fig. 7</b>	<b>Muller C-element and its gate level implementation</b> .....	<b>8</b>
<b>Fig. 8</b>	<b>Gate level implementation of Muller C-element with reset</b> .....	<b>9</b>
<b>Fig. 9</b>	<b>General architecture of an asynchronous wrapper</b> .....	<b>14</b>
<b>Fig. 10</b>	<b>Implementation of a ring oscillator</b> .....	<b>15</b>
<b>Fig. 11</b>	<b>The pausable clocking based asynchronous wrapper</b> .....	<b>16</b>
<b>Fig. 12</b>	<b>Implementation of pausable clocking</b> .....	<b>16</b>
<b>Fig. 13</b>	<b>Mutual exclusion element</b> .....	<b>17</b>
<b>Fig. 14</b>	<b>The stretchable clocking based asynchronous wrapper</b> .....	<b>19</b>
<b>Fig. 15</b>	<b>Implementation of stretchable clocking</b> .....	<b>19</b>
<b>Fig. 16</b>	<b>Implementation of data-driven clocking</b> .....	<b>21</b>
<b>Fig. 17</b>	<b>Architecture of four-phase handshaking, stretchable clocking based GALS systems</b> .....	<b>23</b>
<b>Fig. 18</b>	<b>Timing diagram of Wra1 of four-phase handshaking, stretchable clocking based GALS systems</b> .....	<b>25</b>
<b>Fig. 19</b>	<b>(a) Input Port (b) STG of Input Port</b> .....	<b>27</b>
<b>Fig. 20</b>	<b>Circuit implementation of Input Port</b> .....	<b>28</b>
<b>Fig. 21</b>	<b>(a) Output Port (b) STG of Output Port</b> .....	<b>29</b>



<b>Fig. 22 Circuit implementation of Output Port .....</b>	<b>30</b>
<b>Fig. 23 Gate-level simulation .....</b>	<b>31</b>
<b>Fig. 24 Architecture of two-phase handshaking, stretchable clocking based GALS systems (wrong).....</b>	<b>33</b>
<b>Fig. 25 Capture-pass event controlled latch .....</b>	<b>34</b>
<b>Fig. 26 (a) Double edge triggered flip-flop (DET-FF) (b) Implementation of DET-FF ....</b>	<b>36</b>
<b>Fig. 27 Architecture of two-phase handshaking, stretchable clocking based GALS systems (correct) .....</b>	<b>36</b>
<b>Fig. 28 Timing diagram of Wra1 of two-phase handshaking, stretchable clocking based GALS systems .....</b>	<b>39</b>
<b>Fig. 29 (a) Input Port (b) STG of Input Port.....</b>	<b>40</b>
<b>Fig. 30 Circuit implementation of Input Port .....</b>	<b>41</b>
<b>Fig. 31 (a) The Output Port (b) The STG of Output Port.....</b>	<b>42</b>
<b>Fig. 32 Circuit implementation of Output Port .....</b>	<b>43</b>
<b>Fig. 33 Gate-level simulation .....</b>	<b>44</b>
<b>Fig. 34 LS1 runs at 213.95 MHz and LS2 runs at 73.55 MHz.....</b>	<b>48</b>
<b>Fig. 35 LS1 runs at 73.55 MHz and LS2 runs at 213.95 MHz.....</b>	<b>48</b>
<b>Fig. 36 The diagram of adjacent asynchronous wrappers with FIFO.....</b>	<b>49</b>
<b>Fig. 37 Four-phase bundled data pipeline .....</b>	<b>50</b>
<b>Fig. 38 Two-phase bundled data pipeline a.k.a. Micropipeline .....</b>	<b>50</b>

# List of Tables

<b>Table 1 Dual-rail data encoding .....</b>	<b>6</b>
<b>Table 2 Function of Muller C-element.....</b>	<b>8</b>
<b>Table 3 Function of capture-pass event controlled latch.....</b>	<b>34</b>
<b>Table 4 Area of each block in a wrapper of four-phase and two-phase designs .....</b>	<b>45</b>
<b>Table 5 Latency of four-phase and two-phase designs .....</b>	<b>47</b>



# Chapter 1 Introduction

## 1.1 Motivations

Most modern digital systems are based on synchronous circuit design nowadays. As clock frequency is getting higher and higher, more and more problems may be encountered. For a global clocking system, it is hard to distribute clock signals that has high clock rate but low clock skew. Another problem is power consumption. Clock distribution network takes the largest part of power consumption in synchronous systems [1]. As clock frequency is getting higher, the problem is getting worse. In addition, digital products focus on low power design today. So it is important to make a compromise between high performance and low power design.

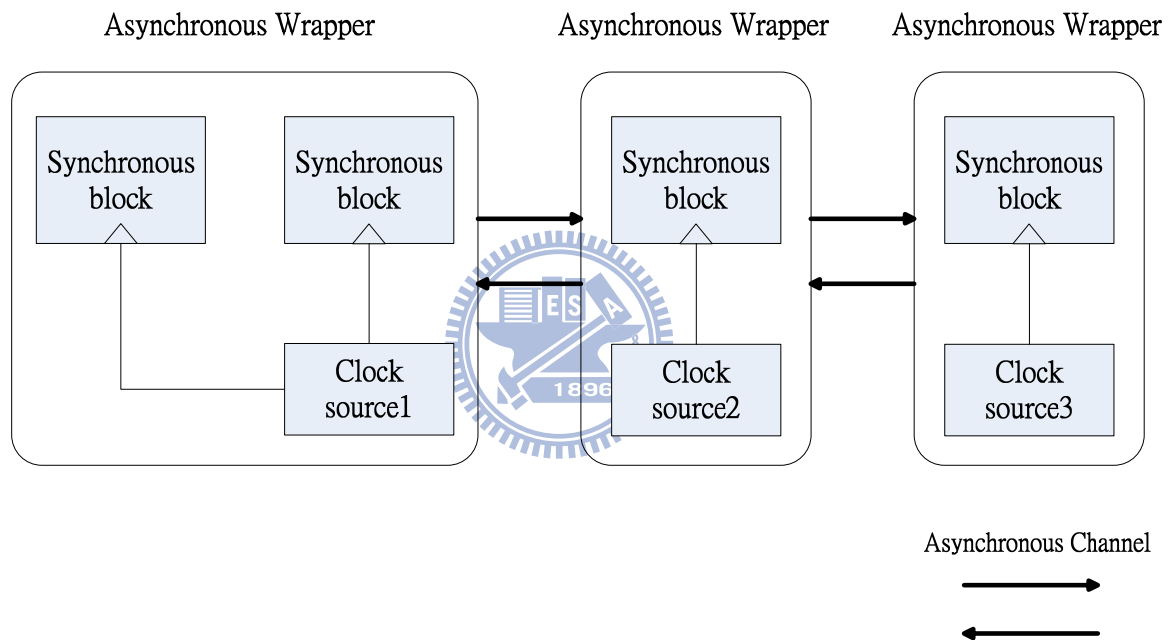


Another trend today is System-on-Chip (SoC) design. A SoC system may contain several IP modules. Each IP module may have its own clock source and operates at different clock frequencies. How to integrate these modules efficiently without data losing between multiple clock domains is a significant issue.

Based on above-mentioned reasons, Globally Asynchronous Locally Synchronous (GALS) design is a promising approach. As implied by the name, a GALS design means that the local modules operate synchronously and the communication between the local modules is asynchronous. Because of communicating via asynchronous approaches, the whole system

can be modularized more easily and consumes less power theoretically. Asynchronous circuits will be introduced in the next section. An abstract diagram of GALS systems is shown in Fig.

1. It can be seen that each synchronous system may have its own clock source and is wrapped by an asynchronous wrapper which will be introduced in the next chapter. The communication between these asynchronous wrappers is asynchronous.



**Fig. 1 An abstract diagram of GALS systems**

## 1.2 Asynchronous Circuits

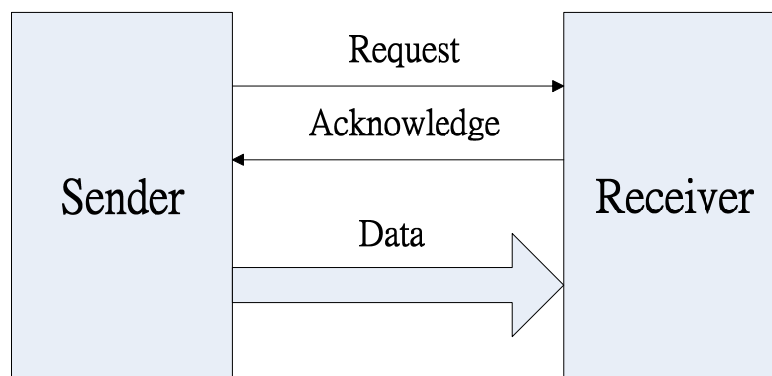
Synchronous circuit design is still a mainstream today and it does have many advantages to support it such as complete design flow and many CAD tools. But as mentioned earlier, with the clock frequency getting higher, the more problems will be faced. So asynchronous circuit

design is getting more and more attractive.

Compared with synchronous circuits, asynchronous circuits are clockless systems. Instead of a global clock signal, asynchronous circuits use handshake protocols between system components to ensure correct sequencing of events. Asynchronous circuits also have several ways to encode data. Of course, it does have its own advantages and drawbacks. These will be introduced in the following sections.

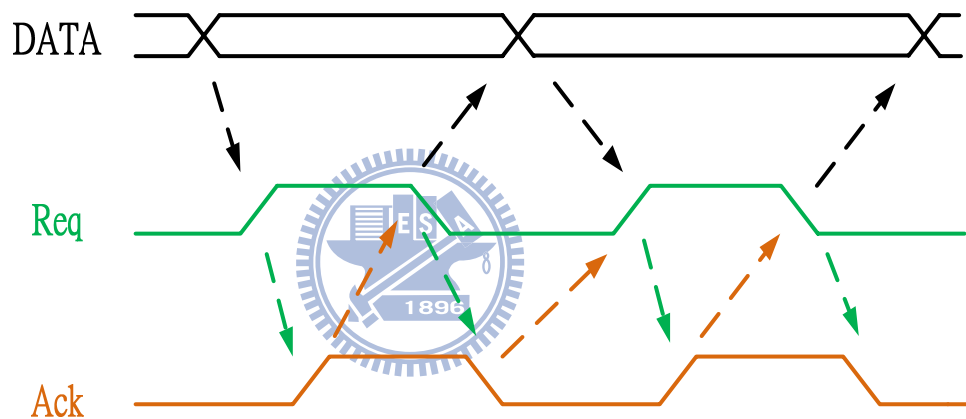
### 1.2.1 Handshake Protocols

Fig. 2 is an illustration of asynchronous systems. It can be seen that the communication between sender and receiver are using handshake protocols i.e. *Request* and *Acknowledge* signals. The data is sent out from sender to receiver. There are two main handshake protocols in asynchronous circuits [2]. One is called four-phase protocol and the other is called two-phase protocol.



**Fig. 2 Illustration of asynchronous systems**

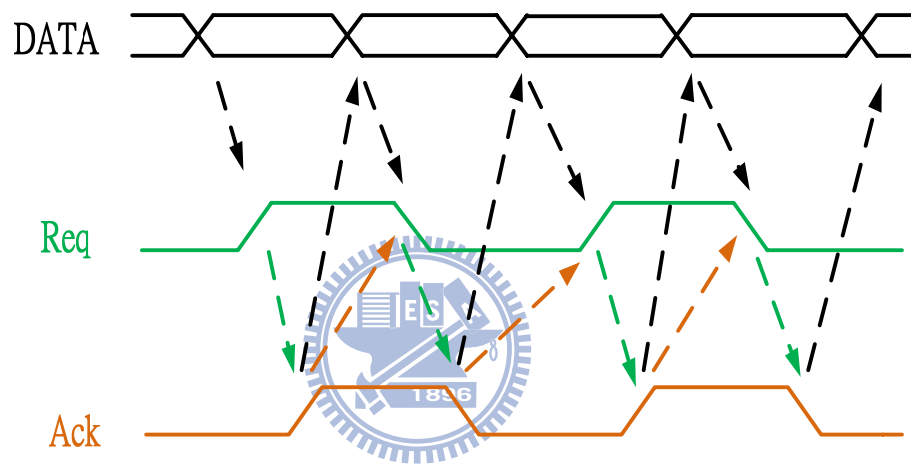
Four-phase protocol is shown in Fig. 3. If the data of sender is ready, sender asserts a request signal. Then receiver gets data and asserts an acknowledge signal as a response. Next, sender deasserts its request signal. Finally, receiver deasserts its acknowledge signal. When both signals return to logic 0, it means that handshaking is complete. So four-phase protocol is also called return-to-zero protocol. Some texts may use the term “level signaling” instead of four-phase protocol.



**Fig. 3 Four-phase handshake protocol**

Two-phase protocol is shown in Fig. 4. The difference between two-phase and four-phase protocols is the meaning of the signal edges. In four-phase protocol, only the rising edges can be active signals. The falling edges in four-phase protocol just mean reset. But in two-phase protocol, the rising edges and falling edges have no difference. In other words, both can represent active signals. Again, when the data of sender is ready, sender sets its request signal from 0 to 1. Then receiver gets data and also sets its acknowledge signal from 0

to 1 as a response. The end of this action means that handshaking is complete. If subsequent data is ready, sender will set its request signal from 1 to 0 to initiate a new process. Then receiver gets data and also sets its acknowledge signal from 1 to 0 to finish this process. So two-phase protocol is also called non-return-to-zero protocol. Some texts may use the term “transition signaling” instead of two-phase protocol.



**Fig. 4 Two-phase handshake protocol**

In general, the systems use two-phase protocol are more efficient and have better performance than the four-phase counterparts. However, using two-phase protocol may lead to more complex circuit theoretically.

## 1.2.2 Data Encoding Methods

There are several ways to encode data [2]. One is called “bundled-data” as known as

“single-rail” encoding. This method is as same as the data in normal synchronous systems.

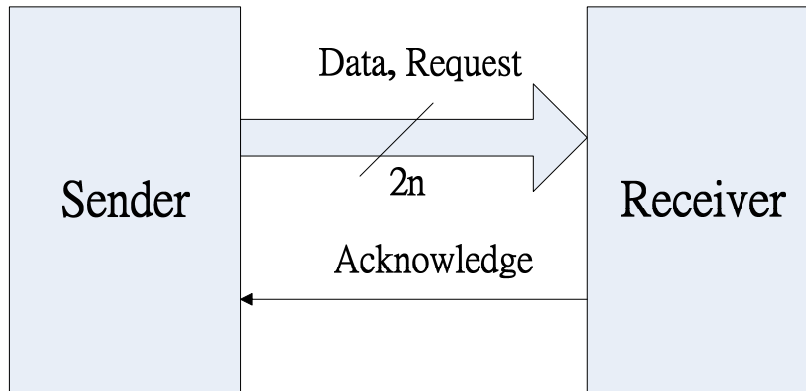
Every wire represents a bit of data. Another one is called “dual-rail” encoding. Dual-rail means that each data bit is encoded into two wires which called “d.t” and “d.f”. The encoding method is shown in Table 1. The pattern of {d.t, d.f} represents as following: {1, 0} means a valid data “1”; {0, 1} means a valid data “0”; {0, 0} means an empty token; {1, 1} is not used.

**Table 1 Dual-rail data encoding**

	d.t	d.f
Empty (“E”)	0	0
Valid “0”	0	1
Valid “1”	1	0
Not used	1	1

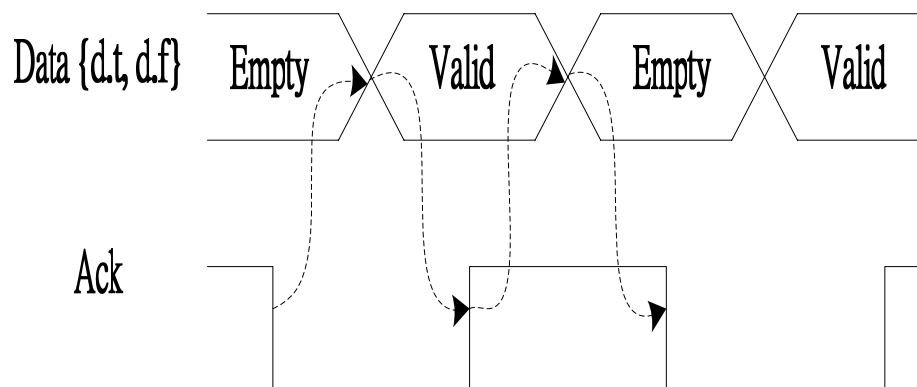
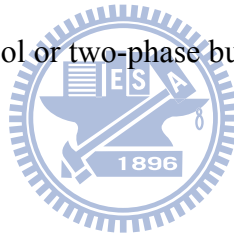
Fig.5 is a diagram of dual-rail data encoding systems. As Fig. 5 shown, the request signal of sender is encoded into one of two wires. So the communication based on dual-rail data encoding is delay-insensitive i.e. it works correctly regardless of the delays in gates and wires.





**Fig. 5 Dual-rail data encoding systems**

The using of handshake protocol and data encoding method depends on your design requirements. It can be any combinations of handshake protocols and data encoding methods such as four-phase dual-rail protocol or two-phase bundled-data protocol. Fig. 6 is an example of four-phase dual-rail protocol.



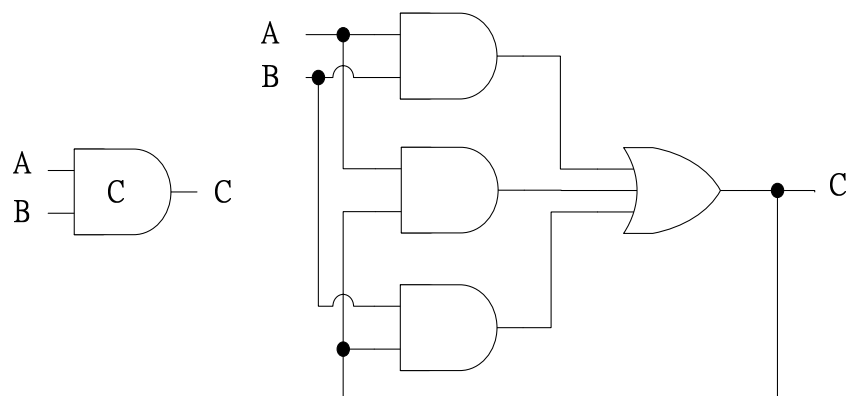
**Fig. 6 Four-phase dual-rail protocol**

### 1.2.3 Muller C-Element

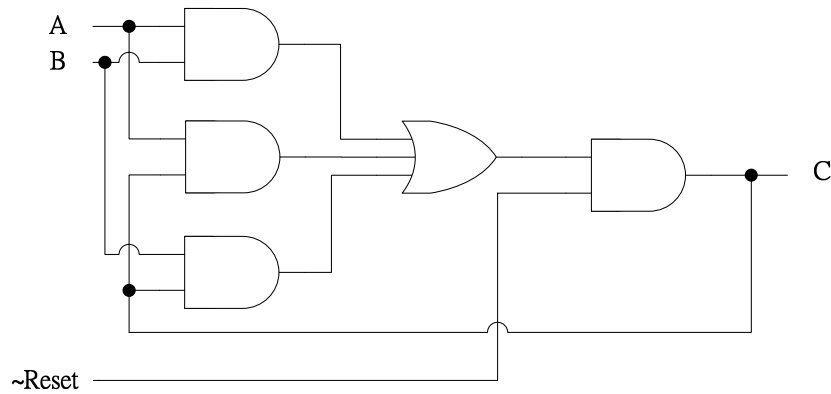
Muller C-element is a fundamental component in asynchronous circuits. It is a state-holding element just like an asynchronous set-reset latch. The function of Muller C-element is shown in Table 2. When the inputs are logic 1, the output is logic 1. When the inputs are logic 0, the output is logic 0. Otherwise, the output does not change. Fig. 7 shows the symbol of Muller C-element and the gate level implementation. Fig.8 shows the gate level implementation of Muller C-element with reset.

**Table 2 Function of Muller C-element**

Input 1	Input 2	Output
0	0	0
0	1	No change
1	0	No change
1	1	1



**Fig. 7 Muller C-element and its gate level implementation**



**Fig. 8 Gate level implementation of Muller C-element with reset**

## 1.2.4 Advantages and Drawbacks

Because of the inherent differences from synchronous circuits, asynchronous circuits have many advantages over synchronous counterparts [2]. The advantages are listed below:

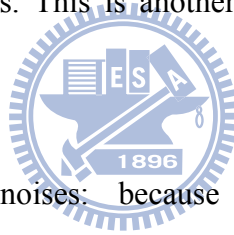
- (1) Low power consumption: because of no clock signals, asynchronous circuits eliminate the largest part of power consumption in the whole system – clock distribution networks [1]. Also, because the modules in asynchronous systems are active only when needed, they do not consume any standby power.
- (2) Average-case performance: in synchronous circuits, the maximum speed which the system can achieve depends on the slowest component. It is worst-case performance.

Compared with synchronous counterparts, asynchronous circuits have an average-case performance. Without a global clock signal, every component can operate at its own speed. As long as a component finishes its computation, the data

can be send out immediately.

(3) No clock skew problems: again, because of no clock signal, asynchronous circuits do not need to consider clock skew problem as synchronous counterparts will face. With the clock frequency getting higher, it is even harder to handle. GALS systems can reduce clock skew problems.

(4) Better modularity: As different modules operate at different clock frequencies, to integrate these modules into a system is not an easy job. Asynchronous circuits use handshake protocols to communicate between modules, so it is much easier to deal with modularity problems. This is another reason that why we use GALS design approach.

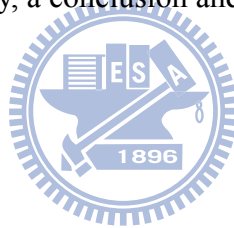


(5) Less electro-magnetic noises: because of no clock distribution networks, asynchronous circuits have less electro-magnetic noises.

Of course, asynchronous circuits do have its own drawbacks. One is few CAD tools to support so that asynchronous circuits are hard to design and are not as popular as synchronous circuits nowadays. Handshake protocols also increase design overheads such as area cost because of additional control signals. In addition, there is no hazard can be tolerated in asynchronous systems, or the whole system will malfunction.

## 1.3 Organization of The Thesis

This thesis proposes a two-phase handshaking, stretchable clocking based interface for GALS systems. Chapter 2 introduces some related works about GALS systems including some researches done in recent years, an asynchronous wrapper, pausable-clock generators, and commonly used four-phase handshaking, stretchable clocking based GALS systems. Chapter 3 illustrates the architecture, implementation and simulation of the new proposed two-phase handshaking, stretchable clocking based GALS systems. Chapter 4 shows the synthesis results of area and latency of two-phase and four-phase designs. It also proves that when adjacent asynchronous wrappers operate at different clock frequencies can work correctly in our new design. Finally, a conclusion and future works are discussed in chapter 5.



# Chapter 2 Related Works

## 2.1 Overview

The first GALS concept was presented by Chapiro in the 1980s [3]. In the mid-1990s, K.Yun and R. Donohue proposed pausable clocking scheme for heterogeneous systems [4]. Then J. Muttersbach et al. introduced the concept of an asynchronous wrapper in 2000 [5]. The asynchronous wrapper will be described in the next section. Since then, more and more studies were proposed. In 2002, Shengxian Zhuang et al. used standard cells and Muller C-elements to construct I/O ports of asynchronous wrappers [6]. Esmail Amini et al. introduced a clock gating technique for off-chip clock generators in 2006 [7]. In 2007, Jhao-Ji Ye et al. proposed a transmission method called “quasi-synchronous” for multiple-clock-domain IP modules [8]. Most of them are designed to improve latency and to reduce area. There are also some studies that make the taxonomy of GALS design styles [9], [10], [11]. According to [10], more and more GALS researches focus on the evaluations and applications of GALS systems over the years.

## 2.2 Asynchronous Wrapper

As mentioned earlier in chapter 1, a GALS system means to make an integration of synchronous systems and asynchronous environments. But how to adapt synchronous systems

into asynchronous environments safely is a big challenge. If the data comes from asynchronous environments is too close in time to the clock edges from synchronous circuits, it will cause synchronization failure i.e. the circuit may enter a metastable state.

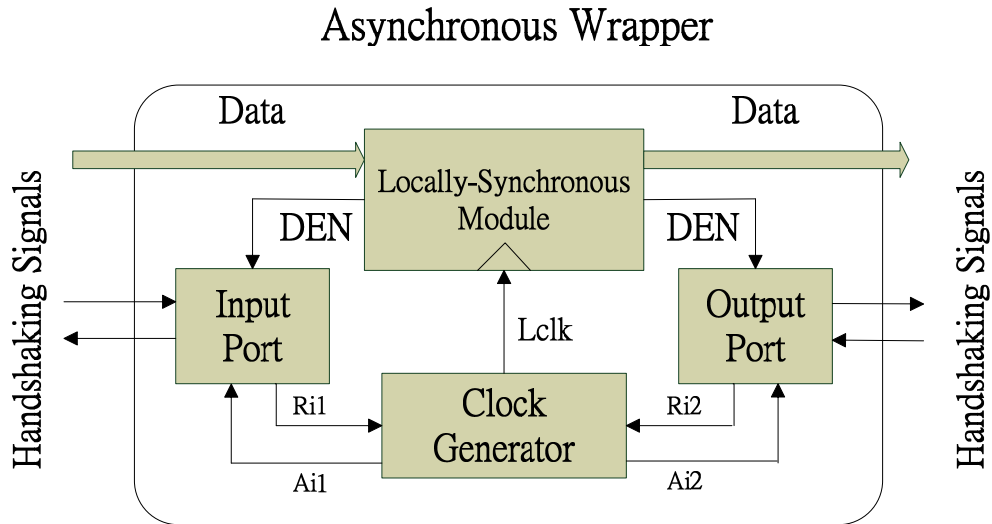
A metastable state is a stable state which the circuit is neither at logic 1 level nor logic 0 level. Instead, the circuit is at an uncertain level between logic 1 and logic 0. So the data may be interpreted as either logic 0 or logic 1 arbitrarily. This situation will cause the whole system malfunction. To avoid metastable problems, the data comes from asynchronous environments has to meet the setup time and hold time of the clock in synchronous systems.

A viable approach is making the clock stoppable so that secures the data transfer between asynchronous and synchronous systems. In order to achieve such a goal, a medium for the two very different systems is needed. This medium is called an asynchronous wrapper [5].

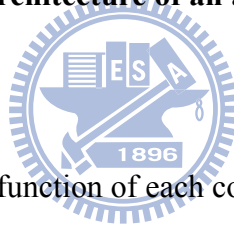
An asynchronous wrapper encapsulates synchronous modules in GALS systems and makes synchronous modules communicate with each other asynchronously. An asynchronous wrapper usually includes locally synchronous (LS) modules, a local clock generator and Input/Output port controllers. A local clock generator is used to generate stoppable clock for LS modules. I/O port controllers are responsible for producing handshaking signals and interfering local clock signals.

In short, an asynchronous wrapper manages all the data transfers in and out of LS modules and delivers locally generated clock signals to them. The general architecture of an

asynchronous wrapper is shown in Fig. 9.



**Fig. 9 General architecture of an asynchronous wrapper**

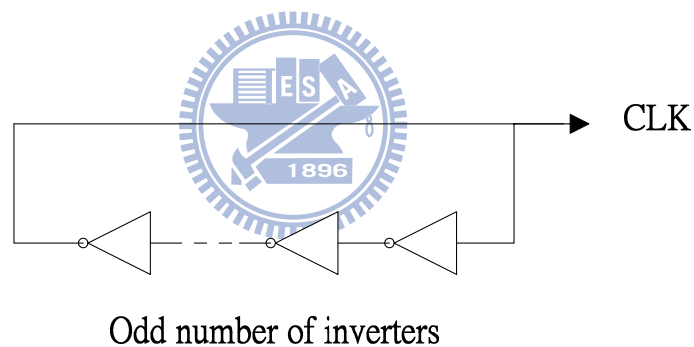


Here we briefly illustrate the function of each component of the asynchronous wrapper in Fig. 9. First, the LS module outputs data actively and accept data passively. As can be seen from Fig. 9, the LS module generates a *DEN* signal to inform I/O ports to stop the local clock *Lclk* when it is ready to accept/output the data from/to adjacent LS modules. Second, the I/O ports communicate with adjacent asynchronous wrappers by handshaking signals. The ways that I/O ports intervene the local clock generator depend on the ways how the local clock generator implements. Third, the local clock generator is used to generate stoppable clock signals for LS modules in order to avoid metastable problems mentioned earlier. The implementation ways of the local clock generator will be explained in the next section.



## 2.3 Pausible-Clock Generators

The local clock generators which generate stoppable clock are called pausable-clock generators. In general, pausable-clock generators can be divided into three categories: pausable clocking, stretchable clocking, and data-driven clocking [9], [10], [11]. All of three are based on ring oscillators. A ring oscillator constructs from an odd number of inverters to form clock signals. The Implementation of a ring oscillator is shown in Fig. 10. In order to stop the clock, additional control circuits are needed. The following introduces the three categories of pausable-clock generators.

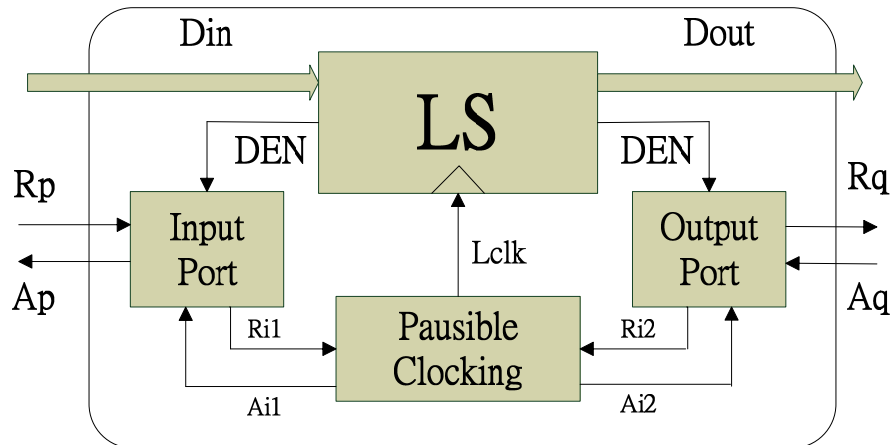


**Fig. 10 Implementation of a ring oscillator**

### 2.3.1 Pausible Clocking

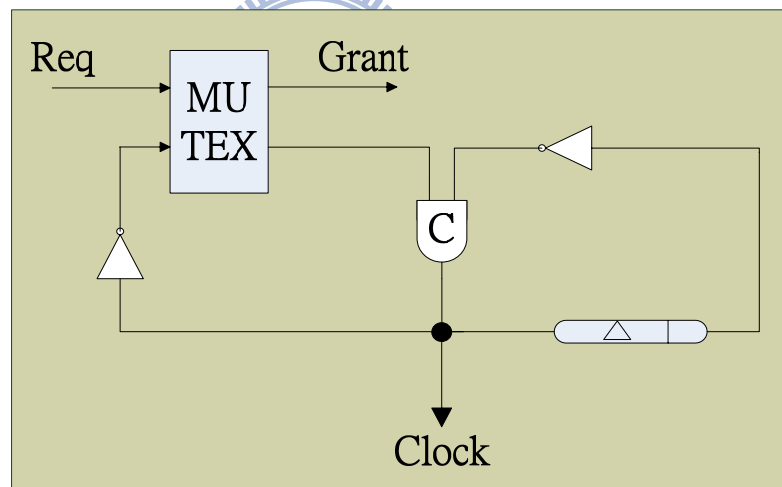
Fig. 11 shows a pausable clocking based asynchronous wrapper. The implementation of pausable clocking is shown in Fig.12. In Fig. 12, it can be seen that pausable clocking includes a mutual exclusion (MUTEX) element, a Muller C-element, and a ring oscillator.

## Asynchronous Wrapper



**Fig. 11** The pausable clocking based asynchronous wrapper

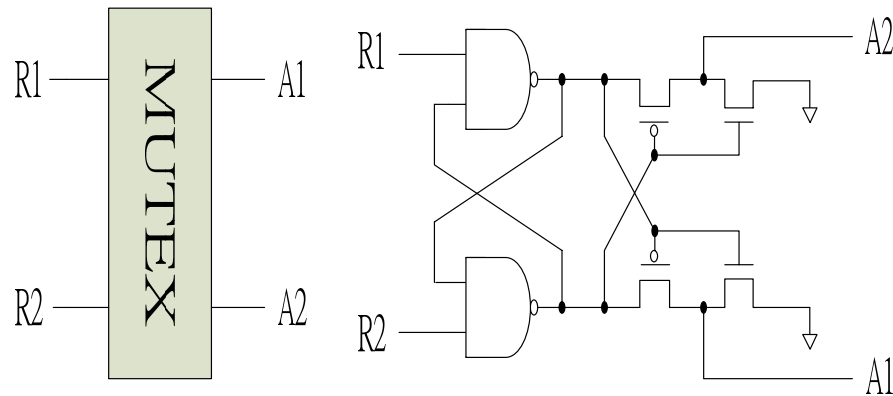
## Pausible Clocking



**Fig. 12** Implementation of pausable clocking

The MUTEX allows only one of two incoming requests to pass at a time [2]. If both requests arrive at the same time, it decides which one to pass by “tossing a coin”. The output of MUTEX is always mutually exclusive. The symbol and implementation of MUTEX is

shown in Fig. 13.



**Fig. 13 Mutual exclusion element**

In Fig. 12, the role of MUTEX is used to gate the clock. We can see that when the *Req* signal is asserted and the *Clock* is at low phase, the *Clock* will be stopped until the *Req* signal is deasserted. On the other hand, when the *Req* signal is not asserted, the *Clock* can operate normally. If both arrive simultaneously, it takes an uncertain time to decide which one to pass.

It means that the *Clock* may be gated or runs as normal.

Before illustrating the operation of pausable clocking based asynchronous wrapper in Fig. 11, there are two details should be reminded. One is that the LS module outputs data actively and accept data passively. The other is that the *DEN* signal issued by the LS module is using the transition signaling approach. It means that there is no difference between 0 -> 1 and 1 -> 0 transitions

Here we start to illustrate the operation of pausable clocking based asynchronous wrapper in Fig. 11. At first, when the LS module is ready to accept data (in other words, the LS module has finished its operation and can actively output data to the next wrapper), it will make a *DEN* signal transition (0 -> 1 or 1 -> 0) to inform I/O ports to stop *Lclk*. Then from the viewpoint of Input Port, it will assert *RiI* and wait for *AiI* from the clock generator. If the clock generator responds *AiI* as logic 1, it means that *Lclk* has been stopped. Next, Input Port can start the four-phase handshaking process with the outside environment (*Rp+* -> *Ap+* -> *Rp-* -> *Ap-*). After finishing the four-phase handshaking process, Input Port deasserts *RiI* and the clock generator deasserts *AiI* as a response. Finally, *Lclk* resumes running. The Output Port operates in the same way.

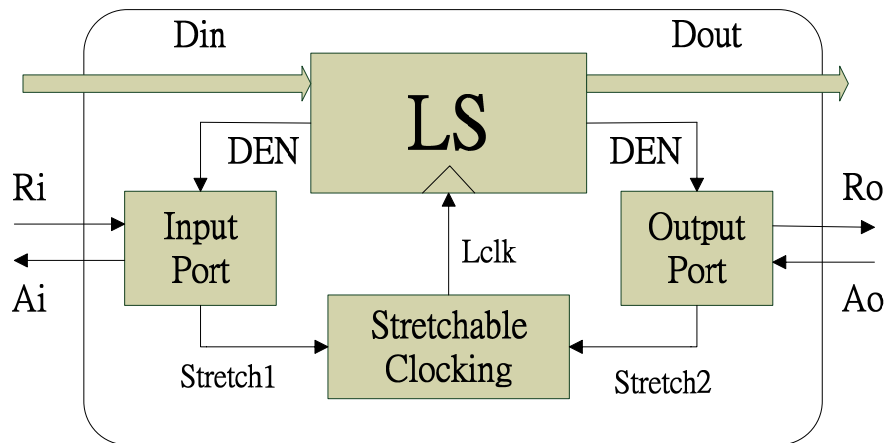


Therefore, in the pausable clocking based asynchronous wrapper, the I/O ports communicate with the clock generator by means of four-phase handshake protocol. There are some designs belong to pausable clocking scheme [4], [5], [12].

### 2.3.2 Stretchable Clocking

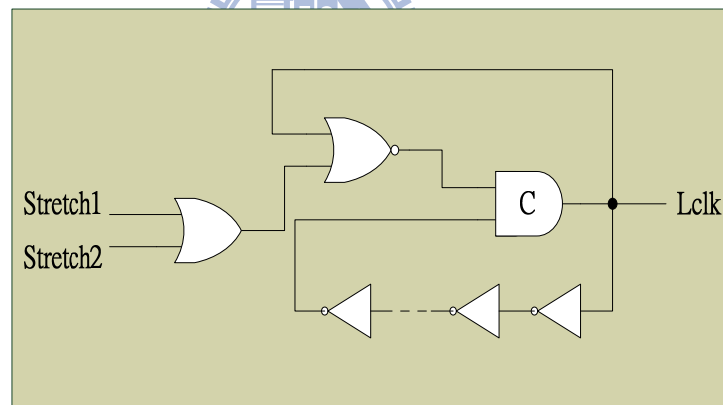
Unlike pausable clocking, stretchable clocking uses simpler circuits to achieve the same goal. It consists of a Muller C-element, a ring oscillator, some basic gates but except a MUTEX. The stretchable clocking based asynchronous wrapper and the implementation of stretchable clocking are shown in Fig. 14 and Fig. 15, respectively.

## Asynchronous Wrapper



**Fig. 14 The stretchable clocking based asynchronous wrapper**

## Stretchable Clocking



**Fig. 15 Implementation of stretchable clocking**

The communication between I/O ports and the clock generator is using the stretch signals.

If any stretch signals are asserted, the low phase of the clock will be stretched until all stretch signals are deasserted. Otherwise, the clock will be generated as normal. For example, in Fig.

15, if any of *Stretch1* or *Stretch2* are asserted, *Lclk* will be stretched until both of *Stretch1* and

*Stretch2* are deasserted. It can be extended to more than two stretch signals by just OR them together.

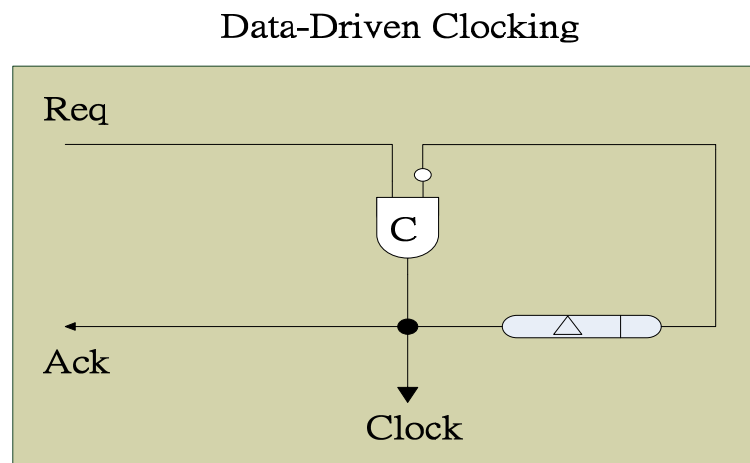
Like the pausable clocking based asynchronous wrapper, the LS module outputs data actively and accept data passively and the *DEN* signal issued by the LS module is using the transition signaling approach.

Here we start to illustrate the operation of stretchable clocking based asynchronous wrapper in Fig. 14. At first, when the LS module is ready to accept data (in other words, the LS module has finished its operation and can actively output data to the next wrapper), it will make a *DEN* signal transition (0 -> 1 or 1 -> 0) to inform I/O ports to stop *Lclk*. Then from the viewpoint of Input Port, it will assert *Stretch1* to the clock generator and *Lclk* is stopped. Next, Input Port can start the four-phase handshaking process with the outside environment ( $Ri+ \rightarrow Ai+ \rightarrow Ri- \rightarrow Ai-$ ). After finishing the four-phase handshaking process, Input Port deasserts *Stretch1*. Finally, if both of *Stretch1* and *Stretch2* are deasserted, *Lclk* resumes running. The Output Port operates in the same way. There are some designs belong to stretchable clocking scheme [6], [13].

### 2.3.3 Data-Driven Clocking

The intent of data-driven clocking is much different like pausable clocking and stretchable clocking although its implementation is not so different compared with the other

two. The implementation of data-driven clocking is shown in Fig. 16. The generation of each clock cycle responds to a complete four-phase handshake protocol. In other words, only when a new *Req* signal requests, the *Clock* signal will be produced. This design can avoid unnecessary clock switching activities only when data is available. There are some designs belong to data-driven clocking scheme [14], [15].



**Fig. 16 Implementation of data-driven clocking**

### **2.3.4 Comparisons Between Three Clock Generation Methods**

Let's do some comparisons between the previous three clock generation methods. There are some drawbacks in pausable clocking. First, the key component of pausable clocking is MUTEX. But MUTEX takes more area cost. Second, it may still have a metastable problem when the two inputs of MUTEX request at the same time before the output is decided. It would cause the whole system malfunction. Finally, because of using four-phase handshake

protocol between the I/O ports and the local clock generator, it may take much time and have worse performance.

Data-driven clocking also has its drawbacks. Each request signal produces a corresponding clock signal. For this reason, it may become a problem if the data needs more clock cycles to finish its operation. So the data-driven clocking needs additional circuits to generate additional clock cycles when this occurred.

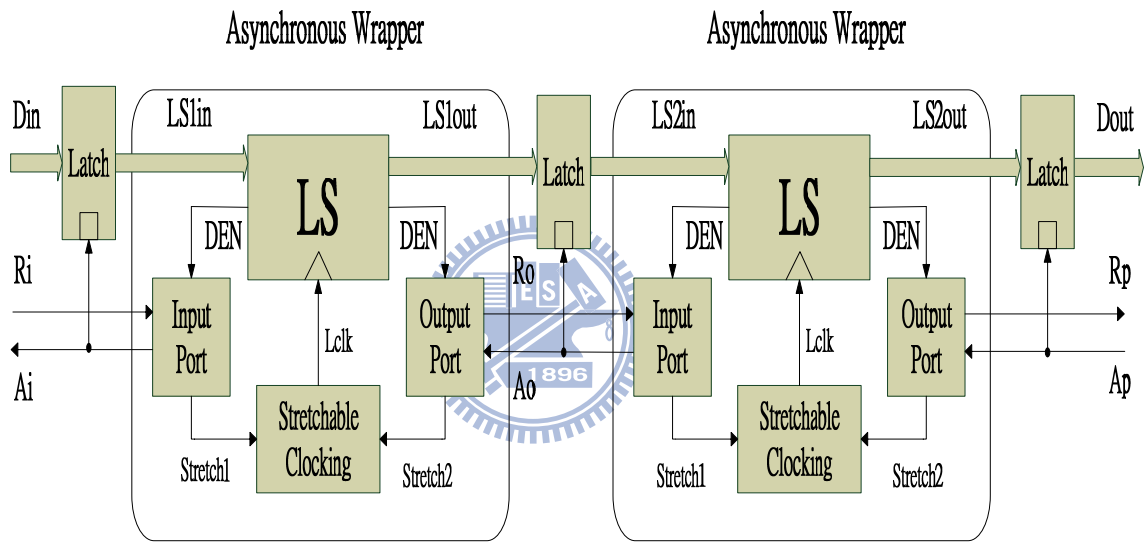
Stretchable clocking has some advantages. First, stretchable clocking can generate multiple clock cycles for data operations easily. Second, instead of requiring a MUTEX to stop clock signals, stretchable clocking uses simpler circuits to achieve the same goal. Finally, there is no handshaking between the I/O ports and the local clock generator. This can reduce the time that spends on stopping and resuming clock signals. Thus, stretchable clocking is a good option to implement local clock generators.

## **2.4 Four-Phase Handshaking, Stretchable Clocking Based GALS Systems**

So far most GLAS systems are based on four-phase handshake protocol. Fig. 17 shows a typical architecture of four-phase handshaking, stretchable clocking based GALS systems. It consists of two asynchronous wrappers. Each asynchronous wrapper is surrounded by two latches. The latch is used as a storage element to prevent data loss. It is controlled by the



acknowledge signal. When the acknowledge signal is asserted, it means that the latch becomes transparent so that data can pass through the latch. When the acknowledge signal is deasserted, it means that the data is latched by the latch. This can make sure that when the data accept by LS modules, the local clock of LS modules is stopped. In other words, adding latches avoid metastable problems.



**Fig. 17 Architecture of four-phase handshaking, stretchable clocking based GALS**

**systems**

Before illustrating the operation of the whole system in Fig. 17, we make some statement in order to facilitate the illustration. The former and latter asynchronous wrappers are called Wra1 and Wra2, respectively. In the same way, the former and latter LS modules are called LS1 and LS2, respectively.

To emphasize once again, the LS module outputs data actively and accept data passively and the *DEN* signal issued by the LS module is using the transition signaling approach.

At first, the whole system will be reset. After resetting, the output of LS1 and LS2 are zero (*LS1out* and *LS2out*). It means that *Wra1* and *Wra2* are ready to accept a new data from the outside environment. Then LS1 and LS2 both make their *DEN* signals activated.

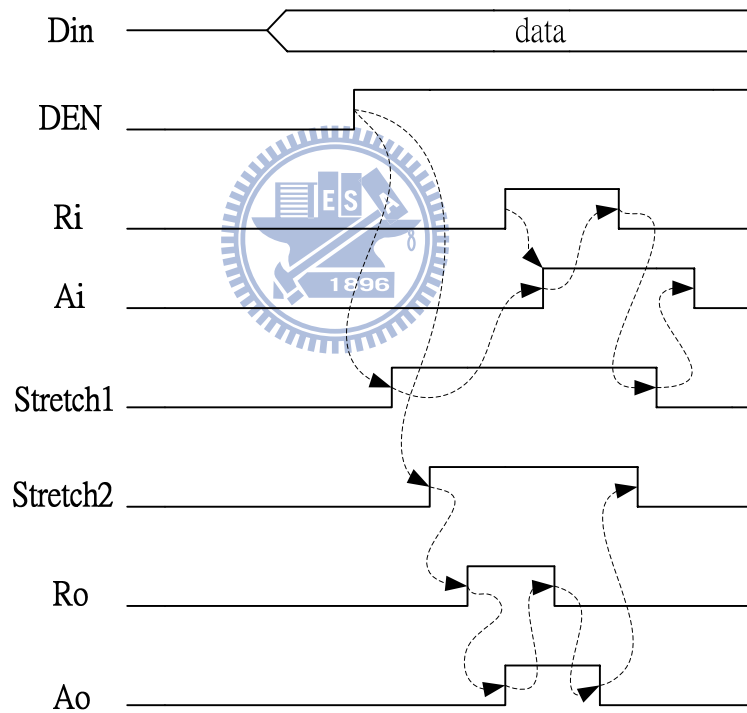
From the viewpoint of *Wra1*, after I/O ports receive a *DEN* signal transition, they pull up *Stretch1* and *Stretch2* to inform the clock generator. So *Lclk* is stretched. Then I/O Ports can start the four-phase handshaking process with the outside environment.

From the viewpoint of the Input Port of *Wra1*, when *Din* is ready, it receives *Ri+* from the outside environment. Then the Input Port responds *Ai+* so that *Din* can pass through the latch and is accepted by LS1. Next, the Input Port receives *Ri-* and pulls down *Stretch1*. If both *Stretch1* and *Stretch2* in *Wra1* are deasserted, *Lclk* resumes running. Finally, the Input Port responds *Ai-* to finish the four-phase handshaking process.

From the viewpoint of the Output Port of *Wra1*, when LS1 finishes its operation, it sends *Ro+* to the Input Port of *Wra2*. Then the Input Port of *Wra2* responds *Ao+* so that *LS1out* can pass through the latch and accept by LS2. Next, the Output Port sends *Ro-* and waits for the Input Port of *Wra2* responding *Ao-* to finish the four-phase handshaking process. Finally, the Output Port pulls down *Stretch2*. If both *Stretch1* and *Stretch2* in *Wra1* are deasserted, *Lclk* resumes running.

The operation of Wra2 is as same as Wra1 illustrating above. The latch between Wra1 and Wra2 is used to store data from Wra1 and prevent data from flowing into Wra2 until  $Ao$  is asserted i.e. make sure that data accepted by Wra2 meet the setup time and hold time of  $Lclk$  in Wra2.

Fig. 18 shows the timing diagram of Wra1 of four-phase handshaking, stretchable clocking based GALS systems.



**Fig. 18 Timing diagram of Wra1 of four-phase handshaking, stretchable clocking based GALS systems**

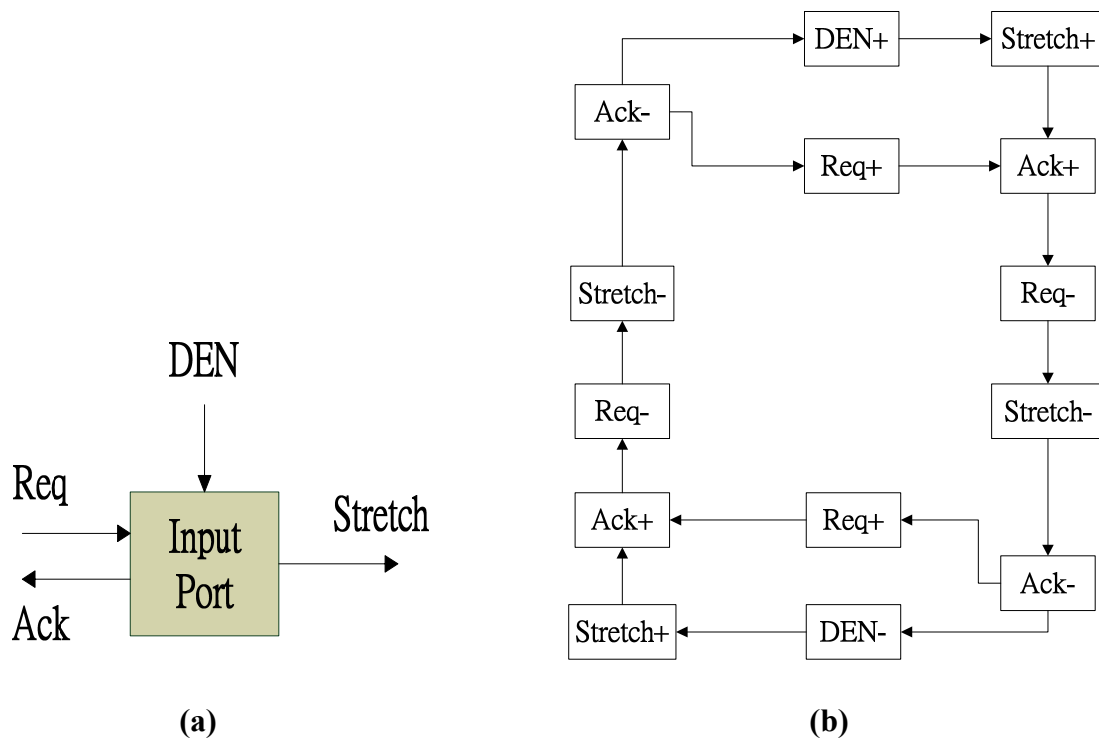
As mentioned earlier, the whole system in Fig. 17 will be reset first. After resetting, the

output of LS1 and LS2 are zero and then LS1 and LS2 both make their *DEN* signals activated.

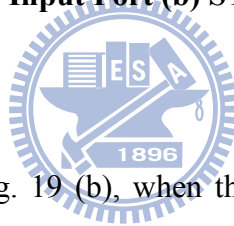
One thing should be noted is that the Output Port of Wra1 and the Input Port of Wra2 will do four-phase handshaking immediately after resetting. So *Stretch2* in Fig. 18 can be deasserted as soon as possible after resetting. Therefore, the first data of Wra1 can be calculated as long as the four-phase handshaking of the Input Port of Wra1 has finished since *Stretch2* has been deasserted.

### 2.4.1 Input Port

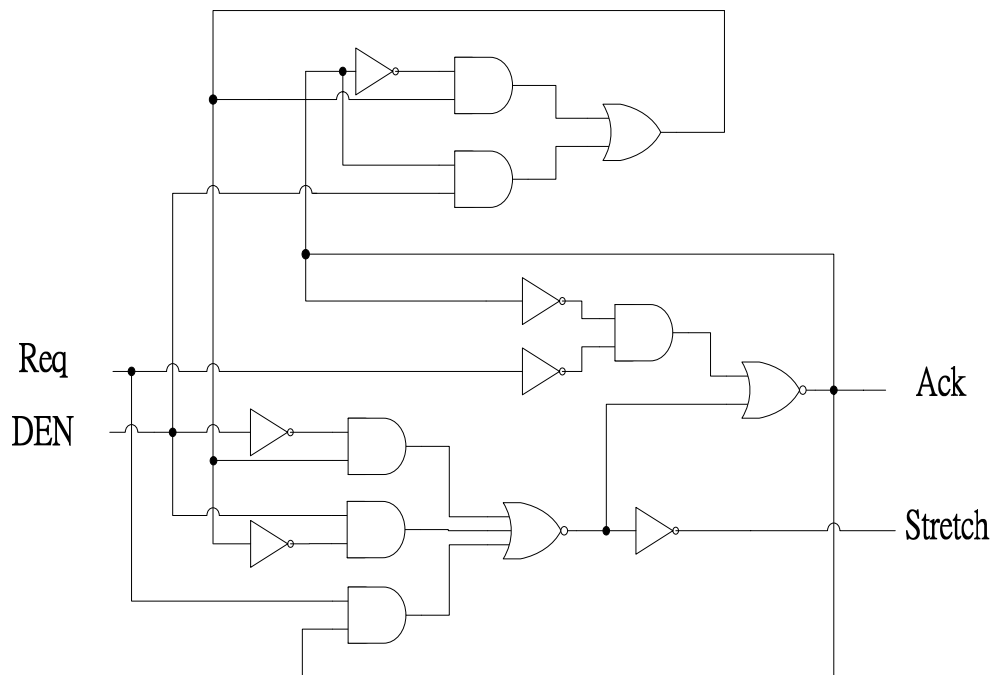
Fig. 19 (a) and (b) show the block diagram of Input Port and its signal transition diagram (STG). The Input Port has two inputs and two outputs. Inputs are *DEN* and *Req* signals. Outputs are *Stretch* and *Ack* signals. *DEN* is using the transition signaling approach. It means that there is no difference between 0 -> 1 and 1 -> 0 transitions.



**Fig. 19 (a) Input Port (b) STG of Input Port**



According to the STG in Fig. 19 (b), when the LS module is ready to accept data, it makes *DEN* do a 0 → 1 transition. Then *Stretch* is asserted. So the local clock is stretched. Next, Input Port waits for *Req+* from the outside environment to start handshaking. *Req+* means that the input data is valid. After receiving *Req+*, Input Port makes *Ack+* as a response. It denotes that the LS module is ready to accept data. Later, if Input Port receives *Req-*, it deasserts *Stretch* to let the local clock start running. Finally, Input Port responses *Ack-* to finish four-phase handshaking. This is a complete data transaction of the Input Port. If there is another data transaction, the LS module will make *DEN* a 1 → 0 transition to start it. Fig. 20 shows the circuit implementation of Input Port.



**Fig. 20 Circuit implementation of Input Port**



## 2.4.2 Output Port

Fig. 21 (a) and (b) show the block diagram of Output Port and its signal transition diagram (STG). The Output Port has two inputs and two outputs. Inputs are *DEN* and *Ack* signals. Outputs are *Stretch* and *Req* signals. *DEN* is also using the transition signaling approach.

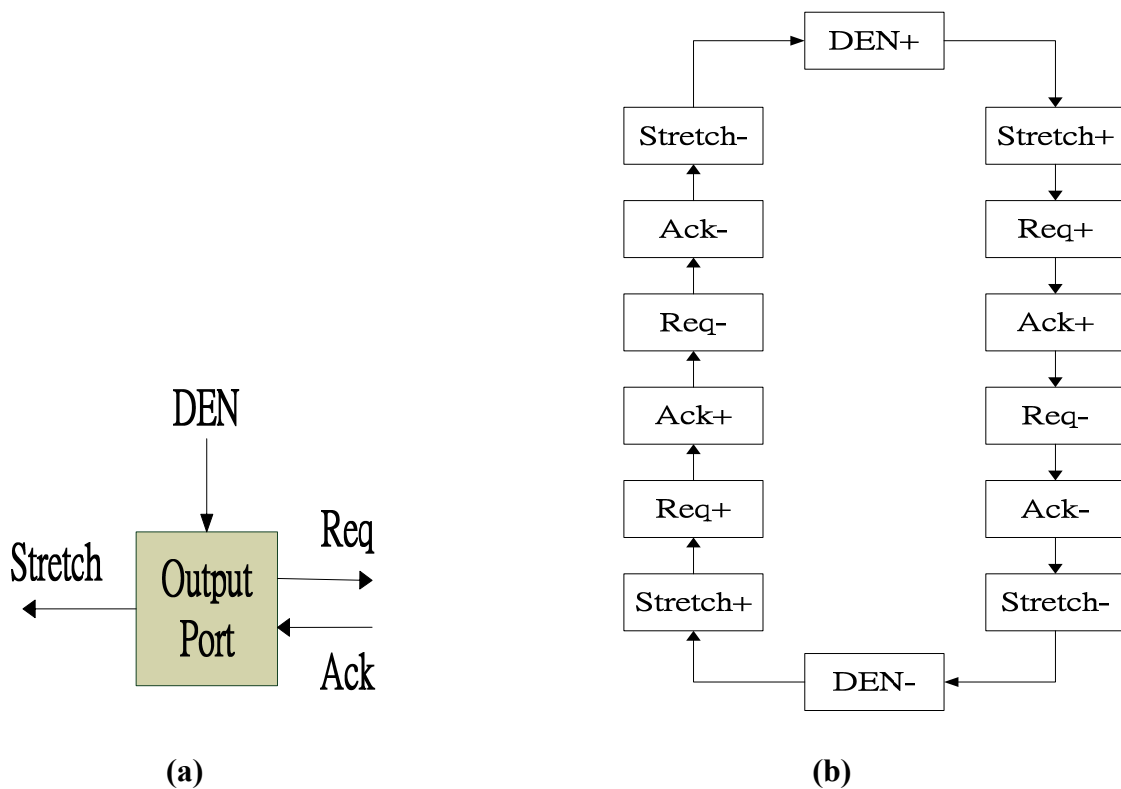
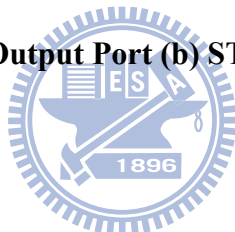
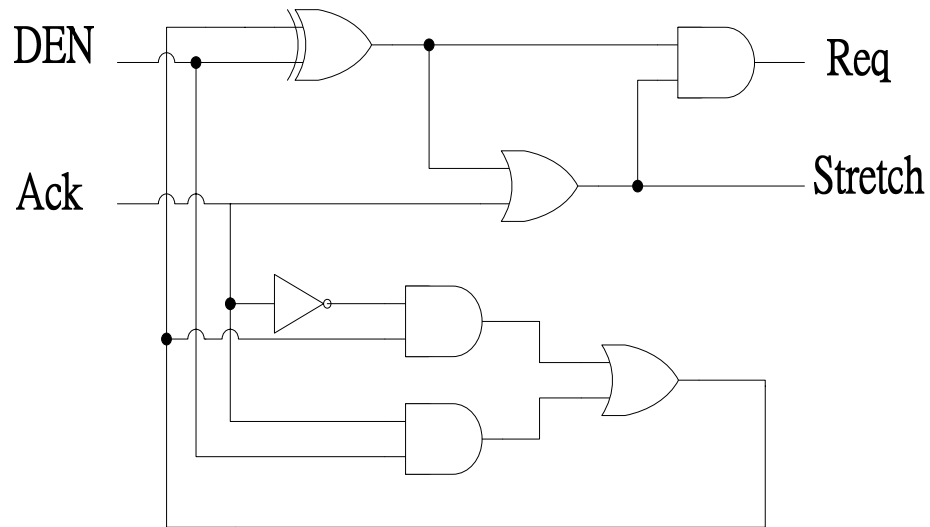


Fig. 21 (a) Output Port (b) STG of Output Port



From the STG in Fig. 21 (b), when the LS module is ready to send out data, it makes *DEN* do a 0 → 1 transition. Then *Stretch* is pulled up and the local clock is stopped. Next, Output Port will initiate *Req+* to start handshaking with the outside environment. After completing four-phase handshaking (*Req+* → *Ack+* → *Req-* → *Ack-*), *Stretch* is pulled down. This presents a complete data transaction of Output Port. If the subsequent data will be delivered out, it makes *DEN* do a 1 → 0 transition to start a new process. Fig. 22 shows the circuit implementation of Output Port.

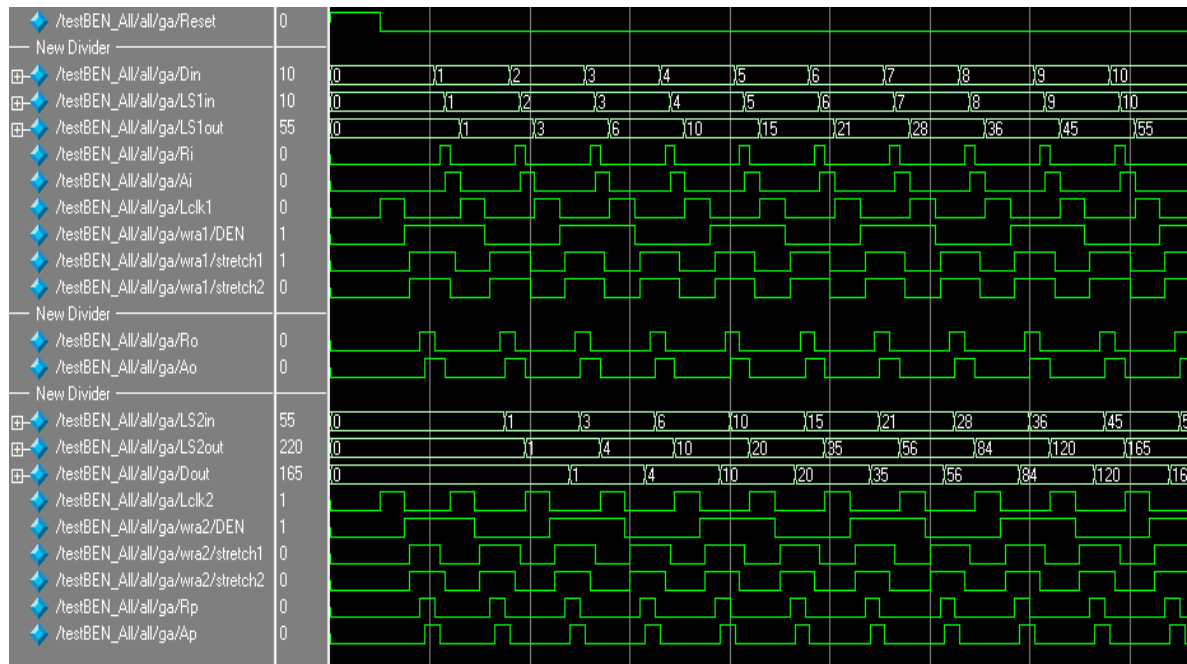


**Fig. 22 Circuit implementation of Output Port**

### 2.4.3 Gate-Level Simulation

To compare with the new design proposed in this thesis, we implement a four-phase handshaking, stretchable clocking based GALS system like Fig. 17 as the object of comparison. Two 10-bit accumulators are used as the LS modules in the simulation. The implementation is synthesized with Synopsys Design Compiler with TSMC 0.13 $\mu$ m cell library. The gate-level simulation is simulated by ModelSim6.0 and the function is proven correctly. Fig. 23 shows the gate-level simulation.



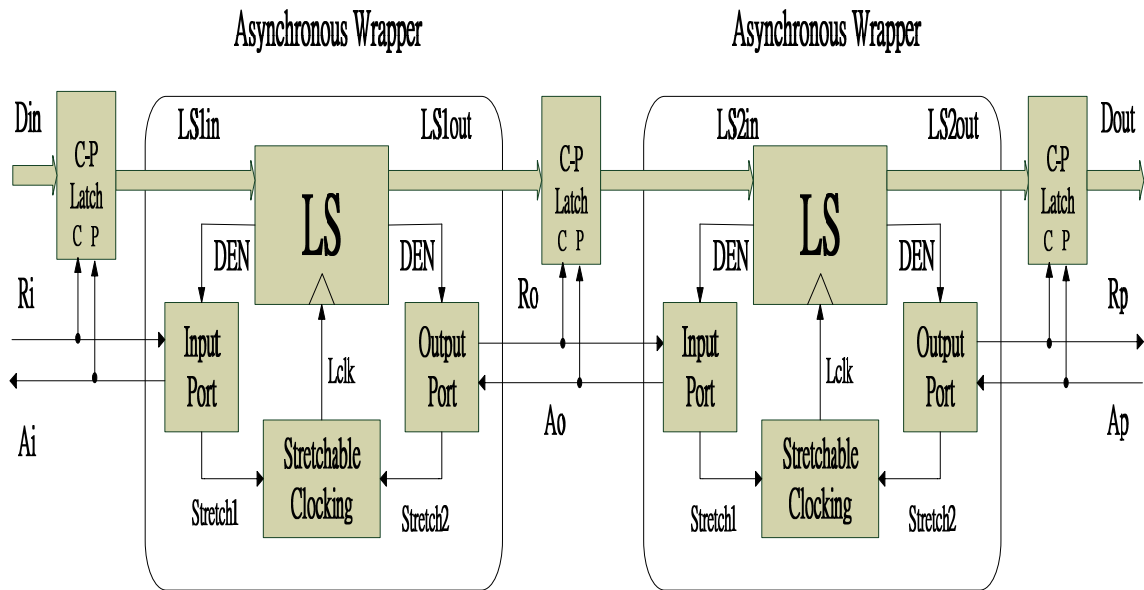


**Fig. 23 Gate-level simulation**



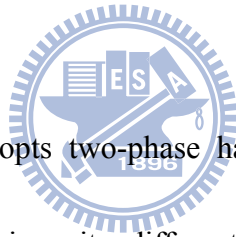
# Chapter 3 A New Design - Two-Phase Handshaking Interface for Stretchable Clocking Based GALS Systems

As mentioned earlier in Chapter 1, handshake protocols of asynchronous circuits can be divided into two categories – four-phase and two-phase handshaking. It is believed that two-phase handshake protocol has better performance than four-phase counterpart because of the property of non-return-to-zero [2]. But it will lead to a more complex implementation. That's why it is hard to find a GALS system that bases on two-phase handshake protocol. In this thesis, a two-phase handshaking interface for stretchable clocking based GALS systems is proposed. The implementations of Input and Output Ports are also delivered. The gate-level simulation is presented, too. The details will be described in the following sections. Fig. 24 shows an intuitive architecture of two-phase handshaking, stretchable clocking based GALS systems. But it has a problem that may cause the whole system malfunction. The problem and solution will be introduced later.

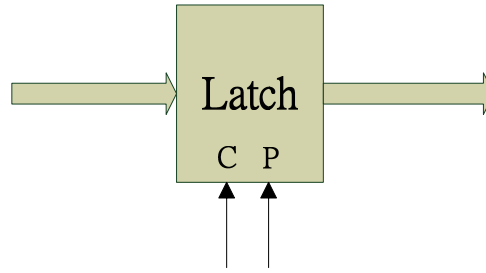


**Fig. 24 Architecture of two-phase handshaking, stretchable clocking based GALS**

**systems (wrong)**



Because the new design adopts two-phase handshake protocol, the storage element between asynchronous wrappers is quite different from the four-phase counterpart. The storage element is called capture-pass event controlled latch, C-P latch for short. Fig. 25 shows its block diagram and Table 3 presents its function. The C-P latch has two control signals – C and P. When both signals are same, C-P latch is transparent i.e. data can pass through it. On the other hand, when both signals are different, data will be captured. Normally, the signal changing flow of the pattern {C, P} is {0, 0} -> {1, 0} -> {1, 1} -> {0, 1}. It means that the behaviors of pass and capture are interleaved.



**Fig. 25 Capture-pass event controlled latch**

**Table 3 Function of capture-pass event controlled latch**

C	P	Behavior
0	0	Pass
1	0	Capture
1	1	Pass
0	1	Capture

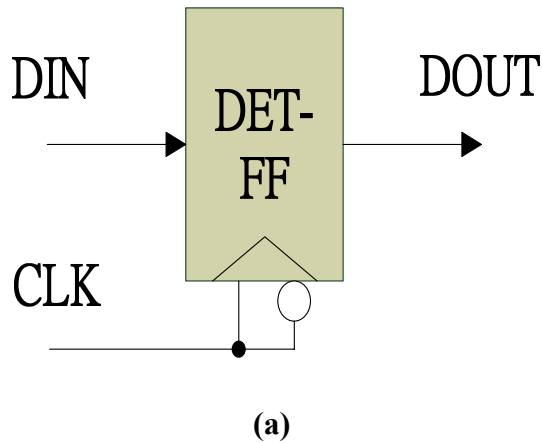
The initial state of C-P latch is  $\{0, 0\}$ . The role of C-P latch is as same as the latch in Fig. 17 i.e. they store data and prevent data from flowing into the next wrapper before the local clock of the next wrapper has been stopped. In Fig. 17, the latch is controlled by  $Ao$ . Before  $Ao$  is asserted, the data from  $Wr1$  is latched in the latch. However, in Fig. 24, the data from  $Wr1$  ( $LSIout$ ) will pass through C-P latch before  $Ao$  is asserted. This is because when  $Wr1$  finishes its operation and before  $Ro$  is asserted, C-P latch is at pass state (because  $\{C, P\}$  is at  $\{0, 0\}$  state). When the Output Port of  $Wr1$  asserts  $Ro$  to try to capture  $LSIout$ ,  $LSIout$  has

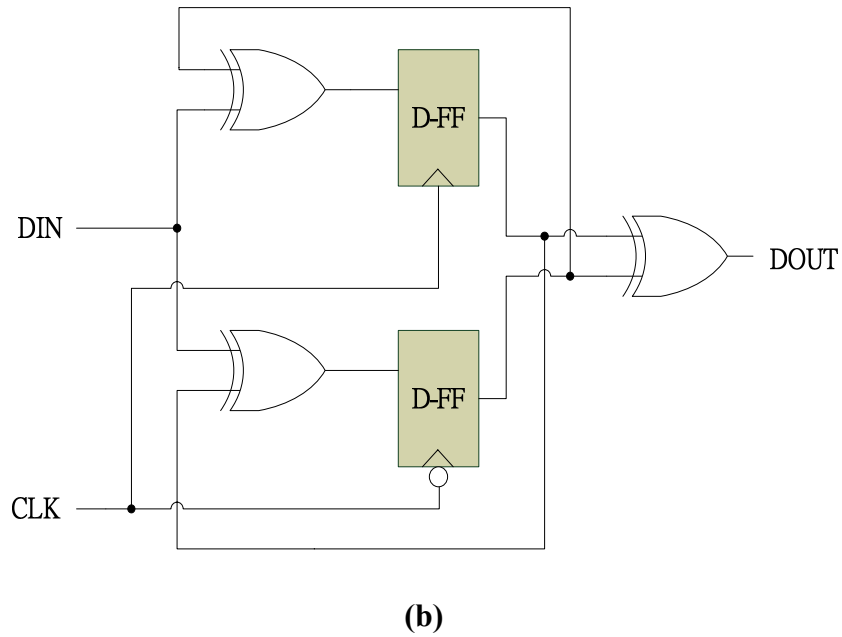
flowed into Wra2. This may cause the whole system malfunction.

To solve the problem mentioned above, a double edge triggered flip-flop (DET-FF) is needed. Because the statement “*always @(posedge clk or negedge clk)*” in Verilog is not synthesizable, we use an alternative implementation to construct double edge triggered flip-flops.

Fig. 26 (a) and (b) shows the block diagram of double edge triggered flip-flop and its circuit implementation. The DET-FF is triggered by both the rising and falling edges of clock signals. In our implementation, we use two D flip-flops and three XOR gates to construct it.

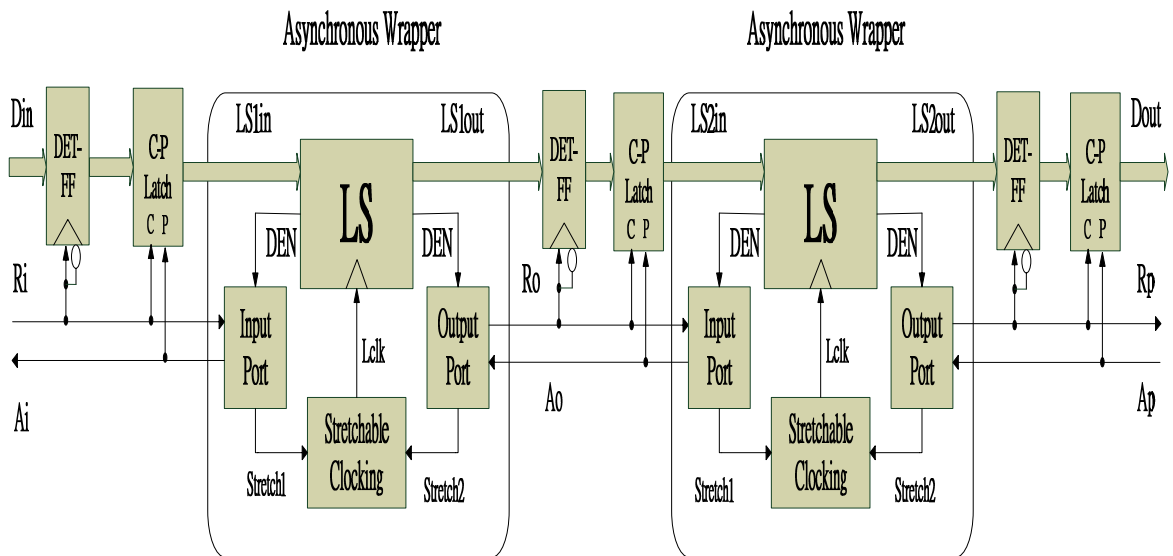
The implementation is synthesizable by Synopsys Design Compiler with TSMC 0.13 $\mu$ m cell library.





**Fig. 26 (a) Double edge triggered flip-flop (DET-FF) (b) Implementation of DET-FF**

So the correct architecture of two-phase handshaking, stretchable clocking based GALS systems is shown in Fig. 27.



**Fig. 27 Architecture of two-phase handshaking, stretchable clocking based GALS**

**systems (correct)**

Here we start to illustrate the operation of two-phase handshaking, stretchable clocking based GALS systems in Fig. 27. Once again, the LS module outputs data actively and accept data passively and the *DEN* signal issued by the LS module is using the transition signaling approach.

At first, the whole system will be reset. After resetting, the pattern  $\{C, P\}$  of C-P latches is  $\{0, 0\}$  and the output of LS1 and LS2 are zero (*LS1out* and *LS2out*). It means that Wra1 and Wra2 are ready to accept a new data from the outside environment. Then LS1 and LS2 both make their *DEN* signals activated.

From the viewpoint of Wra1, after I/O ports receive a *DEN* signal transition, they pull up *Stretch1* and *Stretch2* to inform the clock generator. So *Lclk* is stretched. Then I/O Ports can start the two-phase handshaking process with the outside environment.

From the viewpoint of the Input Port of Wra1, when *Din* is ready, *Din* is blocked by the DET-FF. Later, the outside environment makes *Ri+* so that *Din* can pass through the DET-FF. But *Din* is still blocked by the C-P latch until the Input Port makes *Ai+* so that *Din* can pass through the C-P latch and is accepted by LS1. After responding *Ai+*, the two-phase handshaking process is finished. Finally, the Input Port pulls down *Stretch1*. If both *Stretch1* and *Stretch2* in Wra1 are deasserted, *Lclk* resumes running.

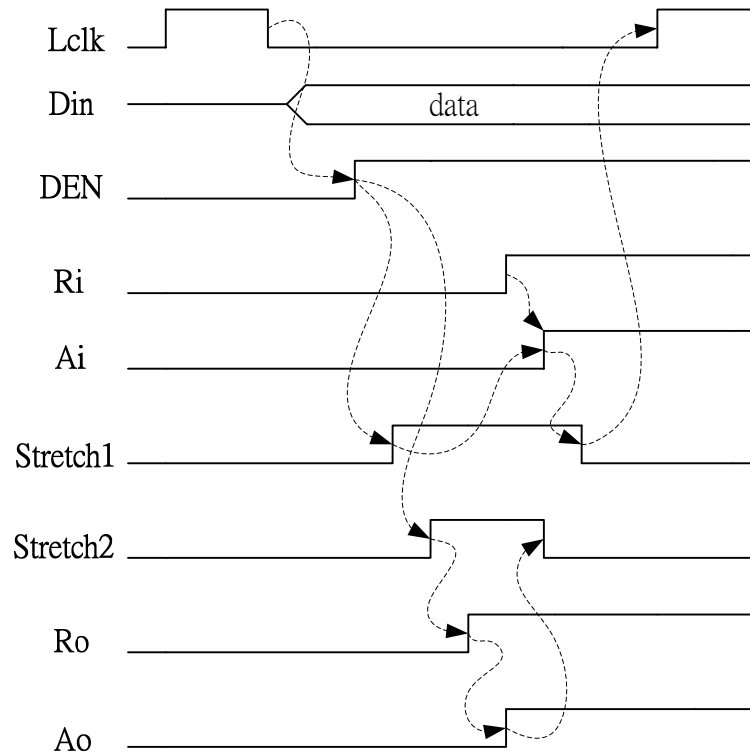
From the viewpoint of the Output Port of Wra1, when LS1 finishes its operation and before sending *Ro+*, *LS1out* is blocked by the DET-FF. Later, it makes *Ro+* to the Input Port

of Wra2 so that *LSIout* can pass through the DET-FF. But *LSIout* is still blocked by the C-P latch until the Input Port of Wra2 sends *Ao+* so that *LSIout* can pass through the C-P latch and is accepted by LS2. After the Input Port of Wra2 responds *Ao+*, the two-phase handshaking process is finished. Finally, the Output Port of Wra1 pulls down *Stretch2*. If both *Stretch1* and *Stretch2* in Wra1 are deasserted, *Lclk* resumes running.

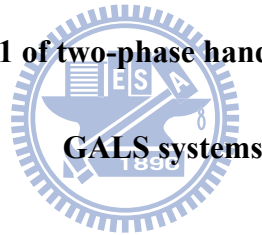
The operation of Wra2 is as same as Wra1 illustrating above. Using DET-FF and C-P latch as storage elements in the new design makes sure that data accepted by the wrapper meet the setup time and hold time of *Lclk* in that wrapper.

Another point needed to pay attention is that the transitions of *DEN* are activated at the falling edges of clock pulses in our design. It is used to make sure that the whole system can work correctly. If *DEN* is activated at the rising edges of clock pulses, some errors may happen like that *Stretch* signals are asserted but the local clock does not stop immediately. This may lead to a system malfunction. The timing diagram of Wra1 of two-phase handshaking, stretchable clocking based GALS systems is shown in Fig.28.





**Fig. 28 Timing diagram of Wra1 of two-phase handshaking, stretchable clocking based**



As mentioned earlier, the whole system in Fig. 27 will reset first. After resetting, the output of LS1 and LS2 are zero and then LS1 and LS2 both make their *DEN* signals activated. It should be noted that the Output Port of Wra1 and the Input Port of Wra2 in Fig. 27 will do two-phase handshaking immediately after resetting. So *Stretch2* in Fig. 28 can be deasserted as soon as possible. Therefore, the first data of Wra1 can be calculated as long as the two-phase handshaking of the Input Port of Wra1 has finished since *Stretch2* has been deasserted.

### 3.1 Input Port

The block diagram of Input Port and its signal transition diagram (STG) are shown in Fig. 29 (a) and (b). The Input Port has two inputs and two outputs. Inputs are *DEN* and *Req* signals. Outputs are *Stretch* and *Ack* signals. *DEN* is using the transition signaling approach as the same as four-phase counterpart.

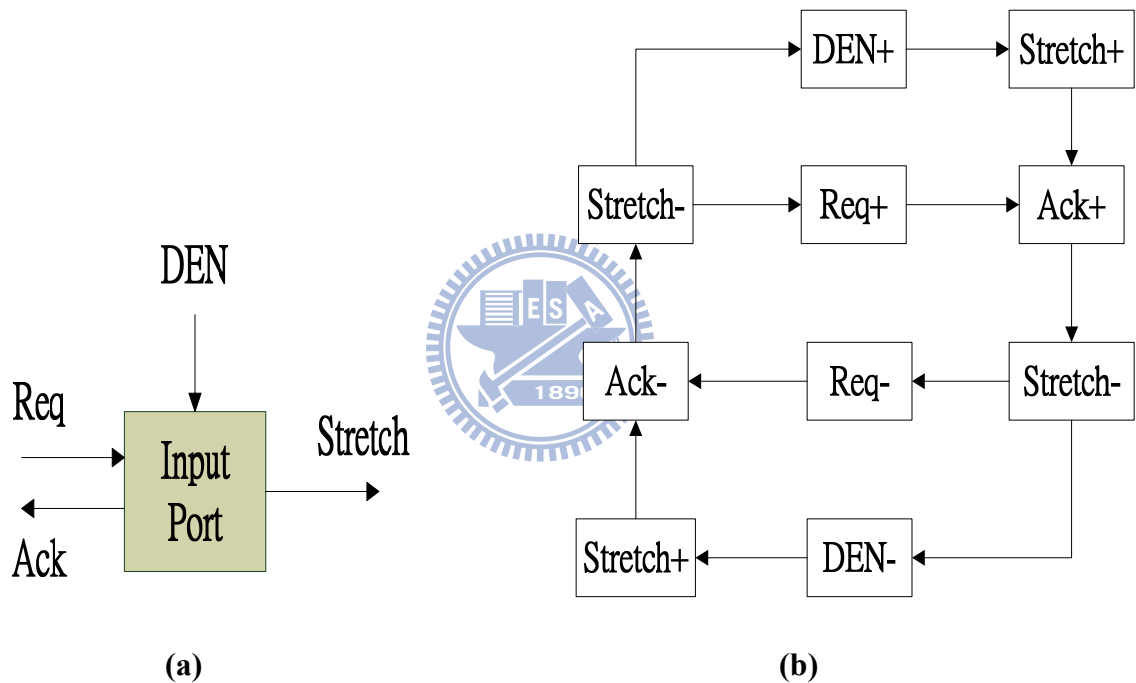
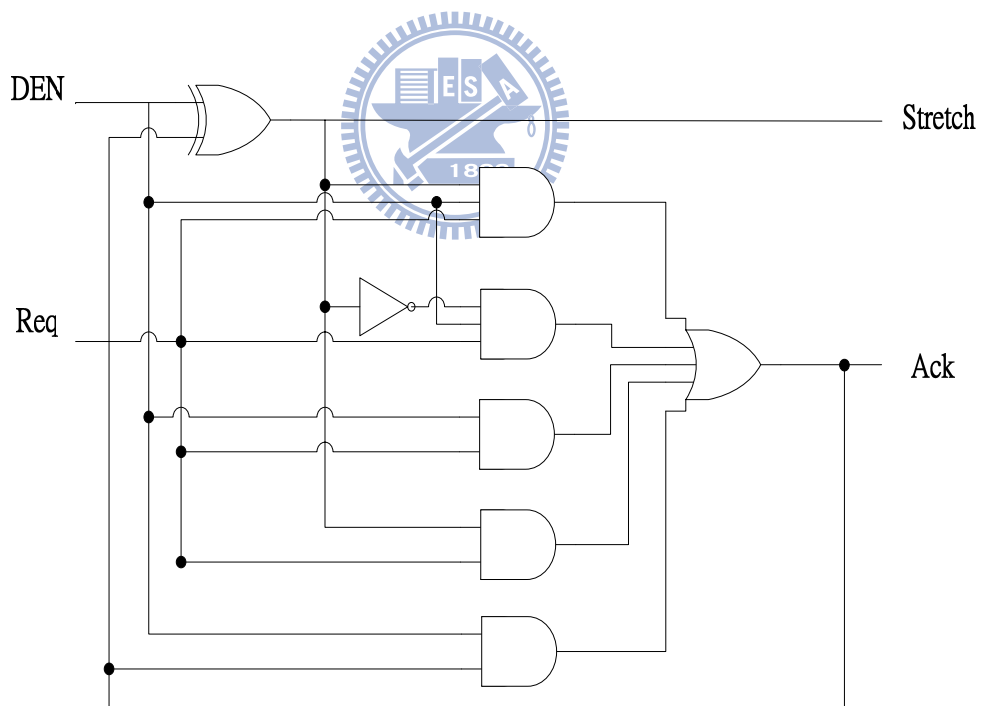


Fig. 29 (a) Input Port (b) STG of Input Port

From the STG in Fig. 29 (b), when the LS module is ready to receive data, it makes *DEN* do a 0 -> 1 transition. Then *Stretch* is asserted. So the local clock is stretched. Next, Input Port waits for *Req+* from the outside environment to start handshaking. *Req+* denotes that the input data is valid. After receiving *Req+*, Input Port makes *Ack+* as a response. So far,

two-phase handshaking is complete. After completing handshaking, *Stretch* signal is deasserted to let the local clock start running. Then the LS module starts its computation. This is a complete data transaction of the Input Port that adopts two-phase protocol. If there is another data transaction, the LS module will make *DEN* a 1  $\rightarrow$  0 transition to start it. The signal changing flow is similar to previous cycle except *Req+* and *Ack+* are substituted by *Req-* and *Ack-*.

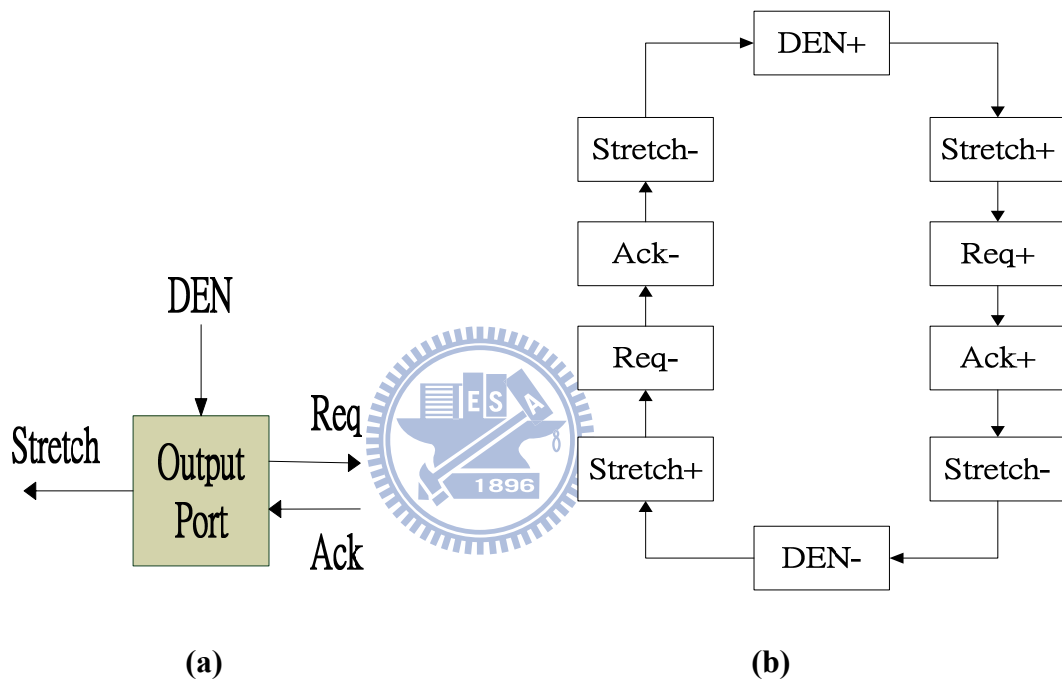
The implementation of Input Port can be converted from the STG in Fig. 29(b) by [2], [16]. Fig. 30 is the circuit implementation of Input Port.



**Fig. 30 Circuit implementation of Input Port**

## 3.2 Output Port

Fig. 31 (a) and (b) show the block diagram of Output Port and its signal transition diagram (STG). The Output Port has two inputs and two outputs. Inputs are *DEN* and *Ack* signals. Outputs are *Stretch* and *Req* signals. *DEN* is also using the transition signaling approach.

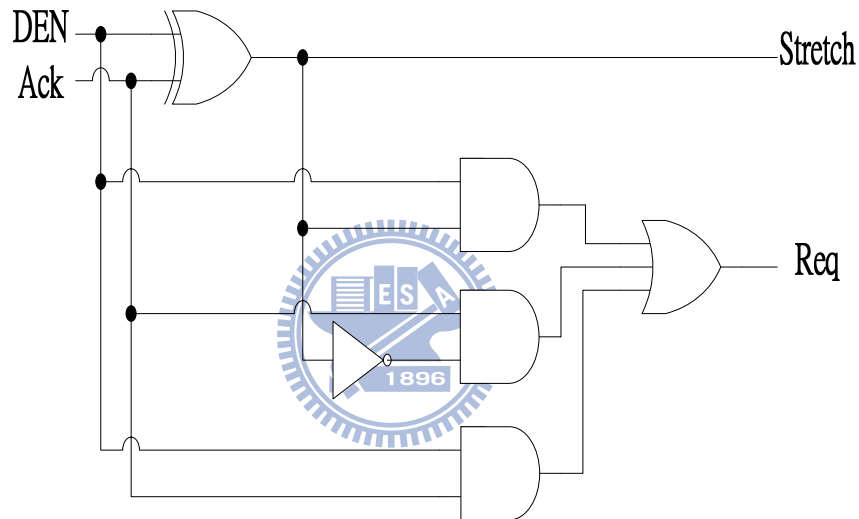


**Fig. 31 (a) The Output Port (b) The STG of Output Port**

According to the STG in Fig. 31 (b), when the LS module is ready to deliver out data, it makes *DEN* do a 0 -> 1 transition. Then *Stretch* is pulled up and the local clock is stretched. Next, Output Port will initiate *Req+* to start handshaking with the outside environment to denote that the data is valid to output. After receiving *Ack+* from the outside environment, *Stretch* is pulled down. Then the two-phase handshaking process is finished i.e. the data

transaction of Output Port is complete. If the subsequent data is ready to be send out, it makes *DEN* do a 1 -> 0 transition to start a new process and starts handshaking by initiating *Req*- then so on.

The implementation of Output Port also can be converted from the STG in Fig. 31(b) by [2], [16]. Fig. 32 is the circuit implementation of Output Port.

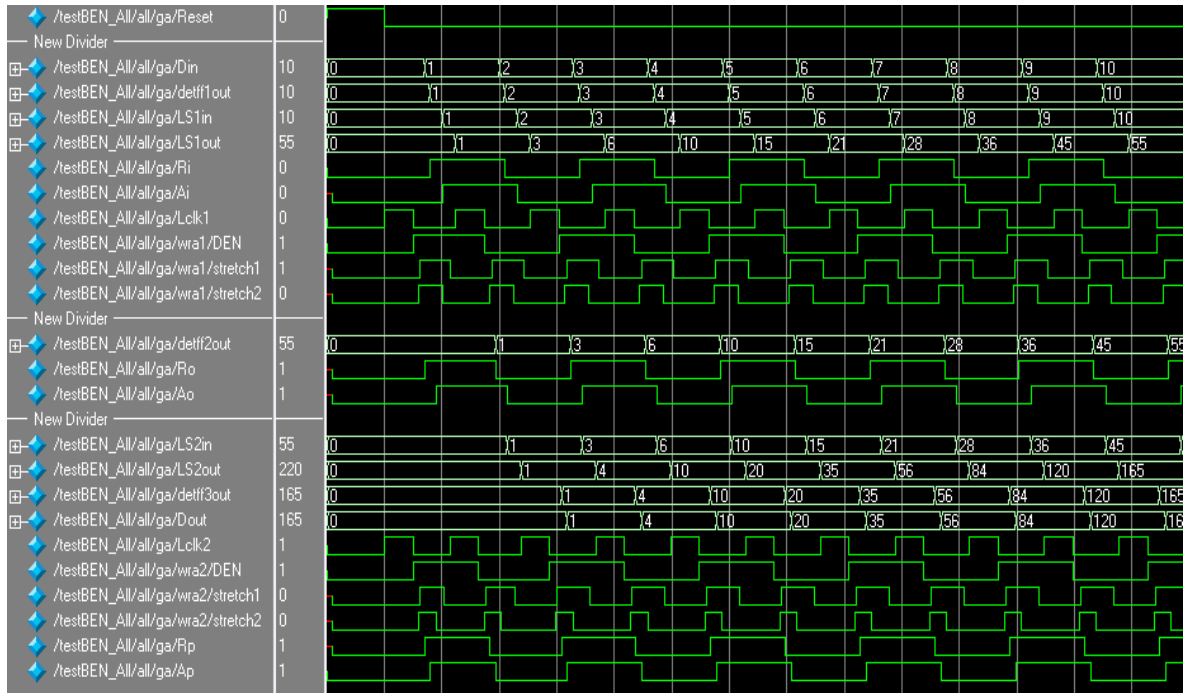


**Fig. 32 Circuit implementation of Output Port**

### 3.3 Gate-Level Simulation

A two-phase handshaking, stretchable clocking based GALS system like Fig. 27 is implemented. Like the four-phase counterpart mentioned in the section 2.4.3, two 10-bit accumulators are used as the LS modules in the simulation. The implementation is also synthesized with Synopsys Design Compiler with TSMC 0.13 $\mu$ m cell library. The gate-level

simulation is simulated by ModelSim6.0 and the function is proven correct. Fig. 33 shows the gate-level simulation.



**Fig. 33 Gate-level simulation**

Because the system adopts the two-phase protocol, it can be expected that the latency and the time to finish handshaking between adjacent wrappers can be improved. The synthesis results and analysis are discussed in the next chapter.


# Chapter 4 Results and Analysis

The implementations are synthesized with Synopsys Design Compiler with TSMC 0.13 $\mu\text{m}$  cell library.

## 4.1 Synthesis Result - Area

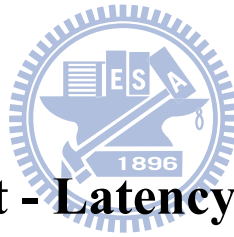
Table 4 shows the area of each block in a wrapper of four-phase and two-phase stretchable clocking designs.

**Table 4 Area of each block in a wrapper of four-phase and two-phase designs**



<b>Block</b>	<b>Area (<math>\mu\text{m}^2</math>)</b>	
	<b>4-phase</b>	<b>2-phase</b>
<b>Input Port</b>	71.290799	49.224600
<b>Output Port</b>	62.803799	67.895999
<b>Storage elements</b>	258.004804	1322.274589
<b>Stretchable clocking</b>	59.409000	59.409000
<b>One accumulator</b>	796.080604	796.080604
<b>One wrapper</b>	1653.267588	1636.293588

It can be seen that the area of the Input Port of two-phase design is smaller than the four-phase counterpart and the area of the Output Port of two-phase design is slightly larger than the four-phase counterpart. The area of stretchable clocking and an accumulator of both designs are the same. For one wrapper (not including the storage elements), the two-phase implementation is slightly smaller than the four-phase counterpart. The biggest difference of area between two designs is the storage elements. For four-phase design, the area of single latch is  $258.004804 \mu\text{m}^2$ . For two-phase design, the area of DET-FF and C-P latch is  $1322.274589 \mu\text{m}^2$ . The area of DET-FF and C-P latch is much larger than the area of single latch.



## 4.2 Synthesis Result - Latency

Before showing the result of latency, the definition of latency should be declared. The latency in our design is defined by the time between data into and out of the system i.e. the time between Din and Dout except the time spends on calculation in Fig. 17 and Fig. 27.

In this comparison, the latencies of four-phase and two-phase designs are 19488.3 (ps) and 18054.6 (ps), respectively. Obviously, the two-phase design takes less time than the four-phase design. It proves that two-phase handshake protocol has better performance than four-phase handshake protocol indeed. When two-phase handshaking interface is applied to larger GALS systems, it can be expected that the more latency are saving than four-phase



counterparts. Table 5 shows the latencies of four-phase and two-phase designs.

**Table 5 Latency of four-phase and two-phase designs**

	<b>4-phase</b>	<b>2-phase</b>
<b>Time (ps)</b>	19488.3	18054.6

The maximum clock frequency in our design is 213.95 MHz. It is limited to the delays of NOR gate and Muller C-element in Fig. 15.

### **4.3 Adjacent Wrappers Operate at Different Clock Frequencies**



In a SoC system, each IP module may run at different clock frequencies. So in this section, we present that our two-phase handshaking interface for stretchable clocking based GALS systems can work correctly when two adjacent wrappers operate at different clock frequencies.

As mentioned in the previous chapters, the former and latter LS modules are called LS1 and LS2, respectively.

Fig. 34 shows the case that LS1 is faster than LS2. We let LS1 run at 213.95 MHz and LS2 run at 73.55 MHz. Fig. 35 shows the case that LS1 is slower than LS2. In this case, we

let LS1 run at 73.55 MHz and LS2 run at 213.95 MHz. It can be seen that both case work correctly in our design.

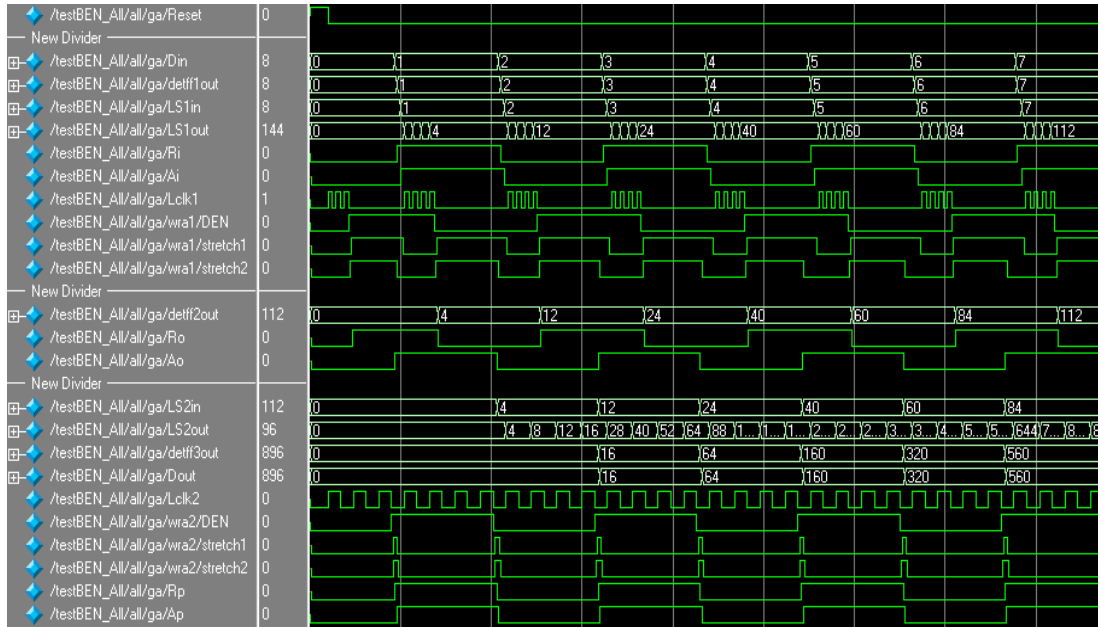


Fig. 34 LS1 runs at 213.95 MHz and LS2 runs at 73.55 MHz

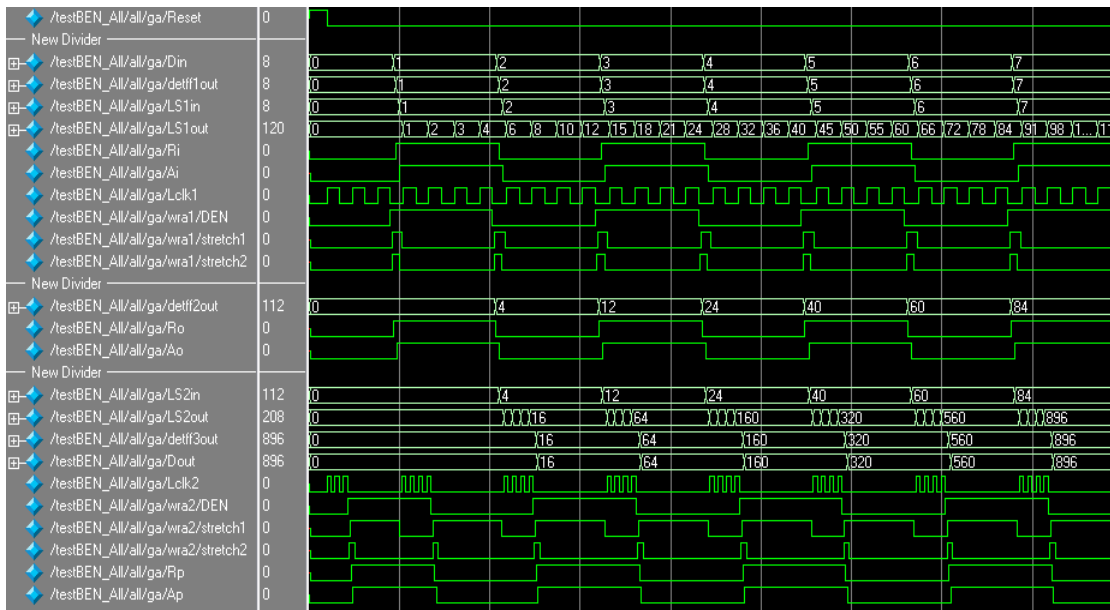
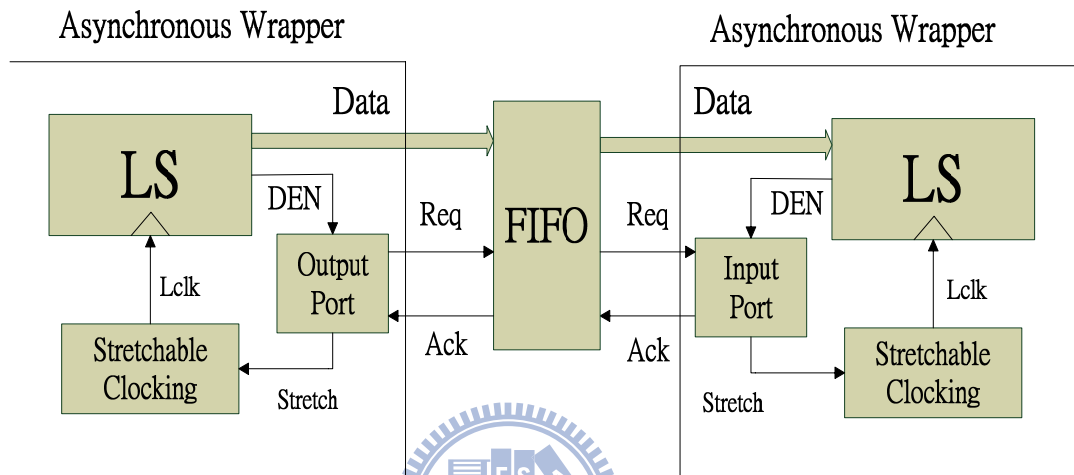


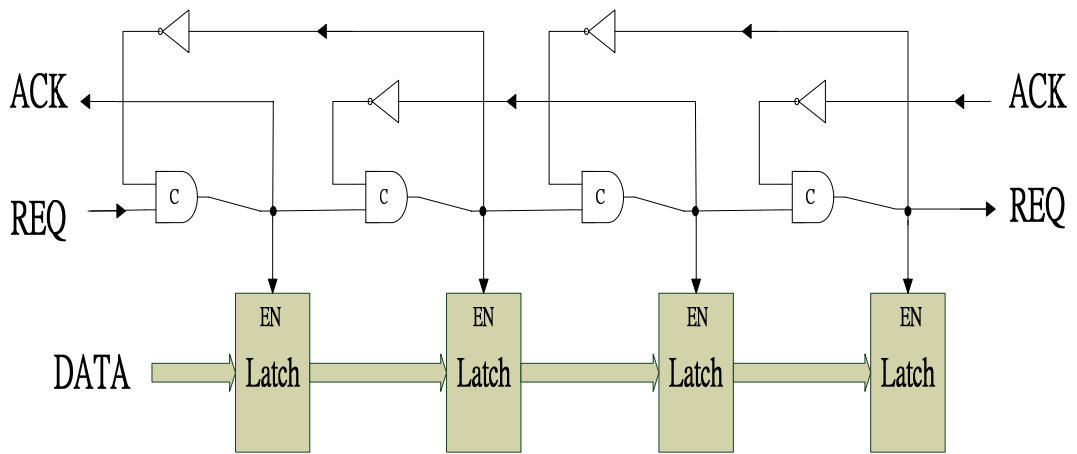
Fig. 35 LS1 runs at 73.55 MHz and LS2 runs at 213.95 MHz

When LS1 is faster than LS2, we can add a FIFO between adjacent asynchronous wrappers to smoothen the bursty data streams. Fig. 36 shows the diagram of adjacent asynchronous wrappers with FIFO.

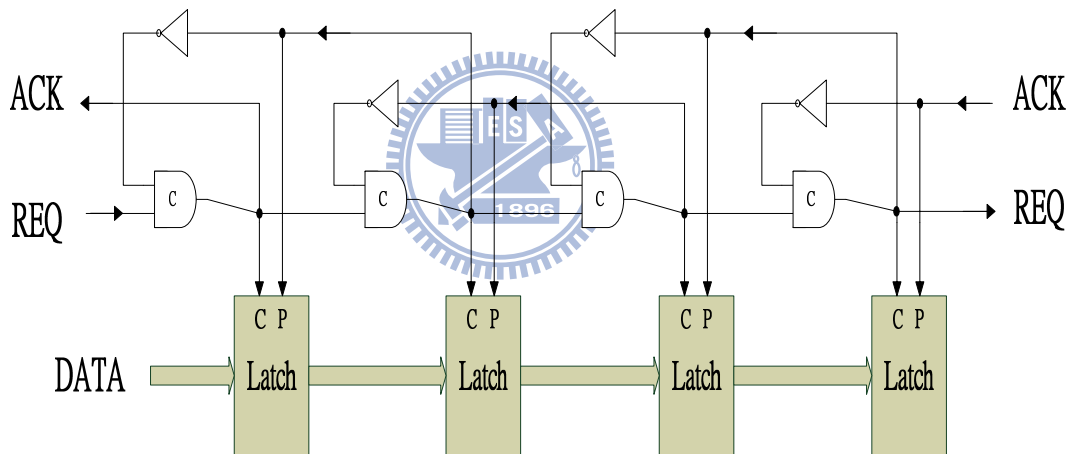


**Fig. 36 The diagram of adjacent asynchronous wrappers with FIFO**

In asynchronous circuits, the FIFO to be added depends on which protocols you used. Fig. 37 and Fig. 38 show a four-phase bundled data pipeline and a two-phase bundled data pipeline, respectively. Two-phase bundled data pipeline is as known as Micropipeline [17]. Because the handshake protocols are different, the storage elements used in the pipeline are also different. It can be seen that four-phase bundled data pipeline uses a normal latch and Micropipeline uses a capture-pass event controlled latch mentioned in chapter 3. Our new design is based on two-phase protocol, so Micropipeline should be used.




**Fig. 37 Four-phase bundled data pipeline**



**Fig. 38 Two-phase bundled data pipeline a.k.a. Micropipeline**

# Chapter 5 Conclusions and Future Work

In this thesis, a two-phase handshaking interface for stretchable clocking based GALS systems is proposed. The local synchronous modules can operate at different clock frequencies independently. In order to verify the feasibility, two 10-bit accumulators are implemented as the LS modules in our design. The synthesis and simulation is implemented by Synopsys Design Compiler with TSMC 0.13 $\mu$ m cell library and ModelSim6.0, respectively. The system function is proven reliably. The maximum clock rate is limited to the architecture of stretchable clocking.

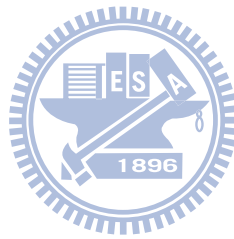


Compared with four-phase handshaking counterpart, the new two-phase handshaking interface has better latency but larger area. The reason is that two-phase handshake protocol is “no-return-to-zero”. Both rising and falling edges of signals can be active edges. Because of using DET-FFs and C-P latches as storage elements, two-phase design has larger area than four-phase counterpart.

The proposed two-phase handshaking interface can work well when adjacent LS modules operate at different clock frequencies.

In the recent years, GALS studies about applications and evaluations are increasing. So applying two-phase handshaking interface for stretchable clocking based GALS systems proposed in this thesis to large systems can be the future work, especially suitable for

Network-on-Chip (NoC) systems [10]. Because the asynchronous wrapper takes small area cost, it just accounts for small overhead in the large systems. Another important issue that should be studied is the evaluation of power consumption. In principle, two-phase handshake protocol consumes less power than four-phase protocol because of non-return-to-zero property. But the increased logic complexity may consume more power than the four-phase counterpart. Thus, the real power consumption should be evaluated after the proposed GALS interface is implemented in real SoC designs.



# References

- [1] A. Hemani, et al., “Lowering power consumption in clock by using globally asynchronous, locally synchronous design style”, In Proc. ACM/IEEE Design Automation Conference, pp. 873-878, June 1999.
- [2] JENS SPARSØ and Steve Furber, Principles of Asynchronous Circuit Design, Kluwer Academic Publishers, 2001.
- [3] D. Chapiro, “Globally-Asynchronous Locally-Synchronous Systems”, Dept. of Computer Science, Stanford Univ., doctoral dissertation, 1984.
- [4] K. Yun and R. Donohue, “Pausable Clocking: A First Step toward Heterogeneous Systems”, Proc. IEEE Int’l Conf. Computer Design: VLSI in Computers and Processors (ICCD 96), IEEE CS Press, pp. 118-127, 1996.
- [5] J. Muttersbach, et al., “Practical Design of Globally-Asynchronous Locally-Synchronous Systems”, Proc. 6th Int’l Symp. Advanced Research in Asynchronous Circuits and Systems (ASYNC 00), IEEE CS Press, pp. 52-59, 2000.
- [6] S. Zhuang, et al., “An asynchronous wrapper with novel handshake circuits for GALS systems”, in Proc. IEEE International Conference on Communications, Circuits and Systems and West Sino Expositions, vol. 2, pp. 1521–1525, June 2002.
- [7] Amini, E. et al., “Globally Asynchronous Locally Synchronous Wrapper Circuits based on

Clock Gating”, In Proceeding of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI), March 2006.

[8] Jhao-Ji Ye, et al., “Low-Latency Quasi-Synchronous Transmission Technique for Multiple-Clock-Domain IP Modules” Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium, pp. 869 – 872, May 2007.

[9] P. Teehan, et al., “A survey and taxonomy of GALS design styles”, IEEE Design and Test, vol. 24, no. 5, pp. 418–428, 2007.

[10] M. Krsti'c, et al., “Globally Asynchronous, Locally Synchronous Circuits: Overview and Outlook”, IEEE Design and Test of Computers, vol. 24, no. 5, pp. 430-441, September-October, 2007.

[11] R. Mullins and S. Moore, “Demystifying Data-Driven and Pausible Clocking Schemes”, Proc. 13th IEEE Int’l Symp. Asynchronous Circuits and Systems (ASYNC 07), IEEE CS Press, pp. 175-185, 2007.

[12] S. Moore, et al., “Point to point GALS interconnect”, In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, pp. 69-75, April 2002.

[13] D. S. Bormann and P. Y. Cheung, “Asynchronous wrapper for heterogeneous systems”, In Proc. International Conf. Computer Design (ICCD), Oct. 1997.

[14] M. Krstic and E. Grass, “New GALS Technique for Datapath Architectures”, In International Workshop on Power and Timing Modeling, Optimization and Simulation



(PATMOS), 2003.

[15] J. Kessels, et al., “Clock synchronization through handshaking”, In Proc. of the 8<sup>th</sup> Intl. Symp. on Advanced Research in Asynchronous Circuits and Systems (ASYNC), 2002.

[16] J. Cortadella, et al., “Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers”, In IEICE Trans. On Information and Systems, pp. 315-325. March 1997.

[17] I.E. Sutherland, “Micropipelines”, Comm. ACM, vol. 32, no. 6, pp. 720-738, June 1989.

