

การออกแบบวงจรคำนวณเลขอิงครรชณีโดยใช้ไดนามิกไปป์ไลน์แบบอสถวาร์



นางสาว เบญจวรรณ ตระบันพฤกษ์

สถาบันวิทยบริการ

จุฬาลงกรณ์มหาวิทยาลัย

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ภาควิชา วิศวกรรมคอมพิวเตอร์


คณะ วิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

ปีการศึกษา 2547

ISBN 974-53-1672-5

ลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย

DESIGN OF ASYNCHRONOUS DYNAMIC PIPELINED  
FLOATING POINT ARITHMETIC CIRCUITS



Miss Benjawan Trabenpreuk

สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

A Thesis Submitted in Partial Fulfillment of the Requirements  
for the Degree of Master of Engineering in Computer Engineering

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2004

ISBN 974-53-1672-5



นางสาวเบญจวรรณ ตรีบัณฑิต : การออกแบบวงจรคำนวณเลขเชิงตรรกะนี้โดยใช้ไดนามิกไปป์ไลน์แบบอสมวาร. (DESIGN OF ASYNCHRONOUS DYNAMIC PIPELINED FLOATING POINT ARITHMETIC CIRCUITS) อ. ที่ปรึกษา : ดร. อาทิตย์ ทองทักษ์, 100 หน้า. ISBN 974-53-1672-5.

วิทยานิพนธ์ฉบับนี้นำเสนอ วงจรคำนวณเลขเชิงตรรกะนี้โดยใช้ไดนามิกไปป์ไลน์แบบอสมวาร ซึ่งรองรับมาตรฐาน IEEE 754 ขนาด 32 บิต งานวิจัยนี้ประกอบด้วยสองส่วนหลักคือ ส่วนควบคุม และส่วนคำนวณเลขเชิงตรรกะนี้

ส่วนคำนวณเลขเชิงตรรกะนี้สามารถคำนวณได้ 5 รูปแบบ คือ บวก/ลบ คูณ การกลับค่าเป็นตรงข้าม การหาค่าสัมบูรณ์ และการเปรียบเทียบ ในขั้นตอนออกแบบได้แบ่งการทำงานของแต่ละรูปแบบออกเป็น ส่วนย่อยเพื่อให้ทำงานแบบไปป์ไลน์ และพบว่ารูปแบบการคำนวณมีการใช้ส่วนย่อยซ้ำกัน และสามารถรวมเป็นวงจรเดียวกันได้ ไดนามิกไปป์ไลน์จึงถูกนำมาใช้ในวิทยานิพนธ์นี้ เพราะเป็นไปป์ไลน์ที่สามารถรองรับการทำงานที่มีหลายรูปแบบในวงจรเดียวได้

ส่วนควบคุมถูกแบ่งออกเป็นสองส่วนย่อยคือ ตัวจัดการงานของไปป์ไลน์ และ ตัวควบคุมชั้นการทำงาน ส่วนควบคุมนำวิธีการจัดการการทำงานเพื่อให้ได้ประสิทธิภาพที่ดี และหลีกเลี่ยงการชนกันของชั้นการทำงานได้ โดยมี ตารางการจอง เมทริกซ์การชน และ แผนภาพแสดงสถานะ เป็นหัวใจสำคัญของการจัดการงาน ส่วนควบคุมที่นำเสนอสามารถนำไปใช้เป็นต้นแบบสำหรับออกแบบส่วนควบคุมของวงจรที่ใช้ไดนามิกไปป์ไลน์แบบอสมวารอื่นได้ โดยขั้นตอนการออกแบบจะมีลักษณะคล้ายกับงานที่นำเสนอ และส่วนวงจรควบคุมสามารถนำไปใช้ได้โดยไม่ต้องเปลี่ยนแปลง

ผลการจำลองการทำงานแสดงให้เห็นว่าวงจรที่นำเสนอสามารถทำงานและคำนวณเลขเชิงตรรกะนี้ตามมาตรฐาน IEEE 754 ได้อย่างถูกต้องและไม่เกิดการชนกันของชั้นการทำงาน

ภาควิชา .....วิศวกรรมคอมพิวเตอร์..... ลายมือชื่อนิสิต.....

สาขาวิชา.....วิศวกรรมคอมพิวเตอร์..... ลายมือชื่ออาจารย์ที่ปรึกษา.....

ปีการศึกษา 2547

## 4570394321 : MAJOR COMPUTER ENGINEERING

KEY WORD: FLOATING-POINT CIRCUIT / PIPELINE SCHEDULING / DYNAMIC PIPELINE/  
ASYNCHRONOUS PIPELINE

BENJAWAN TRABANPREUK : DESIGN OF ASYNCHRONOUS DYNAMIC PIPELINED  
FLOATING POINT ARITHMETIC CIRCUITS. THESIS ADVISOR: ARTHIT THONGTAK,  
100 pp. ISBN 974-53-1672-5.

This thesis proposes an asynchronous dynamic pipeline floating-point arithmetic unit which is compliant with single-precision (32 bits) IEEE 754 standard. It is composed of two main parts: the dynamic pipelined floating- point arithmetic unit and the control unit.

The arithmetic unit operates five functions: add/subtract, multiply, negate, absolute, and compare. The operation of each function is split into stages to work as a pipelined arithmetic unit. Moreover, five functions can be combined into one circuit to reduce the circuit's size. Then, the dynamic pipelining is considered because it is a type of pipeline that can perform multifunction.

The control unit is divided into two main parts: pipeline scheduler and stage controller. The control unit uses the pipeline-scheduling scheme to optimize performance and avoid stage collision. Reservation table, collision matrix, and state diagram are keys for this scheme. The proposed control unit can be used as a template to design the control unit of dynamic asynchronous pipelines. Steps of design method for any dynamic asynchronous pipeline are similar to our work, and control parts designed by STG will be applicable.

The simulation result shows that the circuit can operate and calculate floating-point number in IEEE 754 standard correctly without stage collision.

Department ...Computer Engineering.... Student's signature.....

Field of study..Computer Engineering..... Advisor's signature.....

Academic year .....2004.....

## Acknowledgement

I would like to thank my family for their pure love and support. They stand by me for my whole life. They bring me up when I am down. They do everything they can do for me. This thesis is dedicated to my family.

A special thank for my advisor, Dr. Arthit Thongtak for providing me guidance. I have learnt many new ideas from him.

Big thanks for my thesis committees: Dr. Sartid Vongpradhip, Dr. Somchai Prasitjutrakul, and Mr. Chumnarn Punyasai who gave me good comments to improve my thesis.

Thank you everyone in DSEL Lab for their helps and comments. They are willing helpers, although I need their helps on weekend.

Last but not least, I would like to thank my friends who have give me true friendships since the first day we met. They have never complained me even I call them after midnight or chat with them for a long time. They are my back-up because I can ask for their favours when I am troubled.

## Contents

	Page
Abstract (Thai) .....	iv
Abstract (English) .....	v
Acknowledgement.....	vi
List of figures.....	x
List of tables.....	xiii
Chapter	
I Introduction.....	1
1.1 Background .....	1
1.2 Objective .....	2
1.3 Scope .....	2
1.4 Methodology.....	3
1.5 Expected Result .....	3
1.6 Parts of the thesis book .....	3
1.7 Published paper .....	4
II Related researches and theorems.....	5
2.1 Preliminary work .....	5
2.1.1 Pipelined floating-point arithmetic circuits.....	5
2.1.1.1 MIPS R4000's FPU.....	5
2.1.1.2 Multi-Mode Pipelined Floating-Point Adder and Multiplier.....	7
2.1.1.3 Pipelined Packet-Forwarding Floating Point .....	7
2.1.1.4 Amphion Floating-Point Operators.....	9
2.1.2 Dynamic Pipeline Controller .....	10
2.1.3 An example of dynamic pipelined computer .....	11
2.2 Asynchronous pipeline.....	13
2.2.1 Two-phase signalling protocol.....	14
2.2.2 Four-phase signalling protocol .....	14
2.3 Pipeline scheduling .....	15
2.3.1 Reservation table.....	16

## Contents (continue)

	Page
2.3.2 Collision Matrix.....	17
2.3.3 State Diagram.....	17
2.4 Asynchronous circuit design.....	18
2.4.1 Environmental and delay Model .....	19
2.4.1.1 Environment model.....	19
2.4.1.2 Delay model.....	19
2.4.2 Dual-rail encoding .....	20
2.5 Synthesis of asynchronous circuits from signal transition graph.....	21
2.5.1 STG.....	21
2.5.2 Asynchronous circuit synthesis procedures from STG.....	22
2.5.2.1 Synthesis based on lock relations .....	23
2.5.2.2 Synthesis based on state assignment .....	26
2.6 Binary floating-point representation .....	28
 III Control Unit .....	 30
3.1 Overall of Control Unit .....	30
3.2 The pipeline scheduler.....	32
3.3. The stage controller.....	37
3.4 Circuit design and circuit synthesis .....	43
 IV Asynchronous Floating-point Arithmetic Unit.....	 46
4.1 Overall.....	46
4.2 Add/Subtract .....	48
4.3 Multiply .....	48
4.4 Negate.....	50
4.5 Absolute .....	50
4.6. Compare.....	50



## Contents (continue)

	Page
V Control Unit Implementation .....	51
5.1 Pipeline configuration.....	52
5.2 Reservation table generation.....	52
5.3 Collision Matrix generation .....	54
5.4 State Diagram generation.....	54
5.5 Hardware modification .....	54
VI Experiments and results .....	57
6.1 Experiments.....	57
6.1.1 Adding delays .....	57
6.1.2 Writing Testbench.....	57
6.1.3 Simulation .....	58
6.2 control unit results .....	60
6.3 Floating-point arithmetic unit results.....	61
VII conclusion.....	64
References.....	66
Appendices .....	69
Appendix A.....	70
Appendix B.....	87
Biography .....	100

## List of figures

	Page
Figure 2.1 MIPS R4000's floating-point unit .....	6
Figure 2.2 2 types of pipeline.....	8
Figure 2.3 Amphion's floating-point number format.....	9
Figure 2.4 Sign-magnitude format .....	10
Figure 2.5 Definition of function A.....	12
Figure 2.6 Definition of function B.....	13
Figure 2.7 Dynamic pipeline for function A and function B.....	12
Figure 2.8 An asynchronous pipeline structure .....	13
Figure 2.9 The four-phase micropipeline .....	14
Figure 2.10 Semi-decoupled control circuit.....	15
Figure 2.11 A four stages pipeline .....	15
Figure 2.12 The state diagram of the pipeline in fig.2.11 .....	18
Figure 2.13 An asynchronous system.....	19
Figure 2.14 Two-Input C-element.....	20
Figure 2.15 The quasi-delay insensitive model.....	20
Figure 2.16 A STG of two input C-element.....	22
Figure 2.17 Signal network N(t) for signal t.....	22
Figure 2.18 An example of synthesis based on lock relations.....	24
Figure 2.19 An example of synthesis based on state assignment.....	26
Figure 2.20 IEEE 754-1985 representation .....	28
Figure 3.1 Asynchronous Dynamic pipeline controller .....	31
Figure 3.2 Block Diagram of Pipeline scheduler.....	32

## List of figures (Continue)

	Page
Figure 3.3 Flow chart of Pipeline scheduler .....	33
Figure 3.4 Block Diagram of Initial collision matrix block.....	34
Figure 3.5 Block Diagram of shifter and recent_state block .....	35
Figure 3.6. Block Diagram of collision check block .....	36
Figure 3.7 Block Diagram of state generator block .....	37
Figure 3.8 Block Diagram of stage controller.....	38
Figure 3.9 Flow chart of Stage controller .....	39
Figure 3.10 Block Diagram of function selection .....	42
Figure 3.11 Block Diagram of stage control block.....	42
Figure 3.12 Block Diagram of stage complete block.....	43
Figure 3.13 STG of control block of the pipeline scheduler .....	44
Figure 3.14 a circuit for signal en_sh .....	45
Figure 4.1 Stage and Data transfer between stages .....	47
Figure 4.2 Block diagram of Add/Subtract function.....	48
Figure 4.3 Block diagram of Add/Subtract function.....	49
Figure 4.4 Block diagram of Compare function .....	50
Figure 5.1 Steps of Design methodology.....	51
Figure 5.2 A part of state diagram of the floating-point arithmetic unit .....	55
Figure 6.1 Changing Directory.....	58
Figure 6.2 Loading design.....	59
Figure 6.3 Viewing responses.....	59
Figure 6.4 Simulation result of asynchronous floating-point arithmetic unit .....	61

List of figures (Continue)

Page

Figure 6.5 Simulation result of asynchronous dynamic pipeline control unit ..... 62



สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

## List of Tables

	Page
Table 2.1 Reservation table for function X .....	16
Table 2.2 Reservation table for function Y .....	16
Table 2.3 Examples of state code .....	27
Table 2.4 State codes and trigger cube of crr .....	27
Table 4.1 Arithmetic function and stage usage .....	47
Table 5.1 The reservation table for function Add/Subtract .....	52
Table 5.2 The reservation table for function Multiply .....	52
Table 5.3 The reservation table for function Negate .....	53
Table 5.4 The reservation table for function Absolute.....	53
Table 5.5 The reservation table for function Compare.....	53
Table 6.1 Number of gates in control unit.....	61
Table 6.2 Average time for 1 cycle .....	63
Table 6.3 Number of gates in overall circuits .....	63

# CHAPTER I

## INTRODUCTION

### 1.1 Background

Asynchronous circuit design has been restudied because of clock skew in synchronous circuits since the last decade. Synchronous circuits that synchronize their operations by global clocks are used widely in industries because they are easy to design and verify. However, as size of devices become smaller, their speed are faster, and hardware systems become significantly complex, time spent to distribute signals between gates and wires causes the clock skew. The clock skew occurs when the global clock can not be distributed to entire system at a same frequency. It is a serious problem because the system may malfunction if the clock is not contributed evenly. It is very hard to avoid this problem when designers want to design high-speed circuits. As a result, clock skew limits speed of synchronous circuits.

Asynchronous circuits do not use the global clock to control their operations, so the clock skew is avoided. Circuit responses to signal transitions at any time, and output can be sensed as soon as the operation is completed. Accordingly, another advantage of the asynchronous circuit over the synchronous one is that its speed are not limited by the slowest part in order to synchronize all parts of the system. The speed of asynchronous circuit depends on the signal transition. Low power consumption is another advantage because the signal transitions are made only when necessary.

Nevertheless, asynchronous circuits are harder to design and verify than the synchronous ones. In addition, there are a few tools that support the asynchronous circuit design. Therefore, their obstructions cause asynchronous circuits study to be limited not only in human resources but also in areas.

The propose of this research is to expand the area of the asynchronous circuit study, especially for an increase of speed. Consequently, pipelining is chosen because it is a popular technique used to increase speed of a circuit. Its operation is split into parts called *Pipe stage* or *Pipe Segment* (called stage in this thesis). Each stage is overlapped to each other; every stage can operate simultaneously that is why the circuit is faster.

This thesis focuses on *Dynamic Pipeline* that can be reconfigured to perform multifunction: it performs variable function at different time. Therefore, the number of circuits in a system can be reduced, but control and scheduling part become dramatically more complex.

Floating-point Arithmetic Circuit is chosen for this thesis because it can operate a number of calculations in one circuit. The floating-point representation allows a range of very large and very numbers to be represented. Thus, it is more appropriate than integer representation for tasks that need high resolution such as scientific graphic tasks. Beside, this circuit can be used to demonstrate the dynamic pipeline obviously and is useful.

This thesis is asynchronous floating-point arithmetic circuit's design using dynamic pipeline to study the application of dynamic pipeline to asynchronous circuits. Moreover, effective dynamic pipeline controller design is also focused to be a choice to design faster asynchronous circuits.

## 1.2 Objective

To design asynchronous dynamic pipelined floating-point arithmetic circuits, and verify results by simulation.

## 1.3 Scope

1. A design of asynchronous dynamic pipelined floating-point arithmetic circuits that include these functions: Add/Subtract, Multiply, Negate, Absolute, and Compare.

2. The design is capable single precision (32 bits) of the IEEE 754-4985 floating-point number representation, precise exceptions, and rounding.

3. Scheduling scheme based on the reservation table, the collision matrix, and the state diagram.

#### 1.4 Methodology

The methodology consists of 9 steps as below:

1. Study of Pipeline and pipeline scheduling.
2. Study of floating-point arithmetic.
3. Study of asynchronous circuit design.
4. Study of asynchronous pipeline.
5. Top level design.
6. Component Design/Simulation and Testing
7. System Integration and Testing.
8. Research conclusion.
9. Thesis book writing.

#### 1.5 Expected Result

1. An implementation of asynchronous dynamic pipelined floating-point arithmetic circuits.

2. A way for asynchronous dynamic pipeline research.

3. A way for asynchronous dynamic pipeline scheduling research.

#### 1.6 Parts of Thesis Book

This thesis book is divided into 7 chapters. First, chapter 1 is an introduction that includes background, related work, objective, scope, methodology, expected results and published paper. Second, chapter 2 gives a briefly background of related researches and theorems used in this thesis. Third, chapter 3 is a design of control unit for asynchronous dynamic pipeline. Fourth, chapter 4 is about asynchronous dynamic pipelined floating-point arithmetic unit design. Then, chapter 5 is control unit application; it describes methods to apply control unit in chapter 3 to any asynchronous dynamic pipeline. Dynamic pipeline used in my floating-point arithmetic unit is used as an



example in chapter 5. Next, chapter 6 is results that consist of 2 sections: control unit results and floating-point arithmetic results. Finally, chapter 7 is conclusion including problems and future works.

### 1.7 Published Paper

A part of this thesis is published in “An Implementation of Asynchronous Dynamic Pipeline Controller” topic by Benjawan Trabanpreuk and Arthit Thongtak in “The first Thailand Computer Science Conference (Thcsc 2004)”. The conference is organized by Department of Computer Science, the faculty of Science, Kasetsart University. Thcsc 2004 is held at the faculty of Science, Kasetsart University, Bangkok, Thailand on December 16-17, 2004

A part of this thesis is published in “A Design of Asynchronous Dynamic Pipelined Floating-point Arithmetic Unit” topic by Benjawan Trabanpreuk and Arthit Thongtak in “The second ECTI Annual Conference (ECTI-CON 2005)”. The conference is organized by Electrical Engineering/ Electronics, Computer, Telecommunications and Information Technology (ECTI) Association. ECTI-CON 2005 is held at Asia Pattaya Beach Resort, Choburi, Thailand on May 12-13, 2005.

สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

## CHAPTER II

### RELATED RESEARCHES AND THEOREMS

The related idea and theorem is divided into 6 sections. First, preliminary works are discussed briefly in section 2.1. Second, section 2.2 describes operations and types of asynchronous pipeline. Third, section 2.3 gives some background about Pipeline scheduling that keep pipeline performance and stage collision avoidance. Then, section 2.4 is asynchronous circuit design because it is dramatically different from synchronous circuit design techniques. Next, synthesis of asynchronous circuits from signal transition graph is in section 2.5 because it is applied to control parts of our design. Finally, binary floating-point representation standard and format are briefly described in section 2.6.

#### 2.1 Preliminary works

The preliminary works is divided into 3 parts: Pipelined floating-point arithmetic circuits, Dynamic pipeline controller, and an example of dynamic pipelined computer.

##### 2.1.1 Pipelined floating-point arithmetic circuits

There are a number of Pipelined floating-point arithmetic circuits that have been presented, but all of them are synchronous circuits. Therefore, some modifications are necessary for asynchronous circuits. In this section, operations, application, advantages and disadvantages of each circuit are described briefly.

###### 2.1.1.1 MIPS R4000's FPU [1,2]

The MIPS floating point Unit, FPU, with associated system software fully conforms to the ANSI/IEEE standard 754-1985 (IEEE 754) which is the standard for Binary Floating-Point Arithmetic. The IEEE 754 includes 5 precise exceptions: Inexact, Overflow, Underflow, Division by zero and Invalid operation. If these exceptions are detected, numbers can not be represented in IEEE 754 standard. This FPU supports single format, 32 bits, and double format, 64 bits (IEEE 754 representation is described in section 2.6).

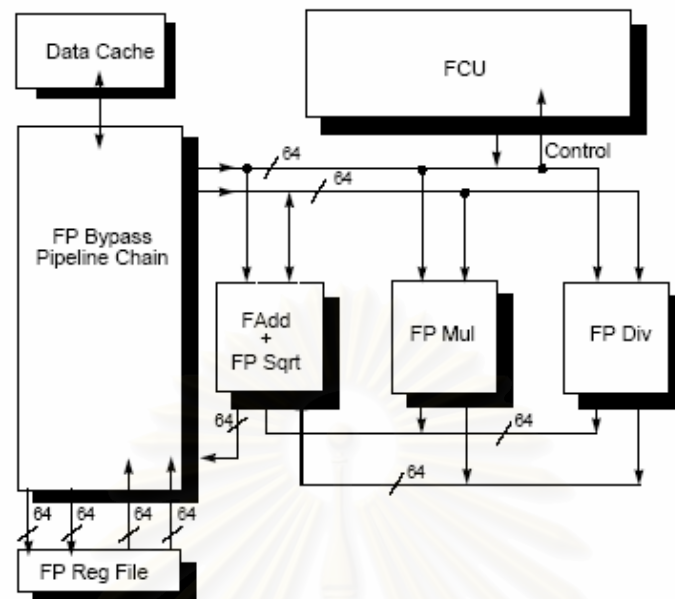


Figure 2.1: MIPS R4000's floating-point Unit [2]

This FPU that is shown in figure 2.1 is a part of MIPS R4000 microprocessor, and it operates as a coprocessor for CPU and extends the CPU instruction set to perform arithmetic operations on floating-point value. The FPU operations include Add/Subtract, Multiply, Division, Square root, Negate, Absolute, and Compare.

The FPU consists of 3 pipelined functional units.

1. FP Add + FP Sqrt is adder and square root circuit.
2. FP Mul is multiplier.
3. FP Div is divider.

Each functional unit consists of 2-3 pipelined stages, for example Mul1, Mul2, Div1, and Div2, to increase speed of circuit because circuit of each stage is smaller and faster. If two instructions need to use the same stage, a later instruction is stalled until a required stage is available.

This thesis refers some operations of MIPS R4000's FPU because it is fully supports floating-point operations.

#### **2.1.1.2 Multi-Mode Pipelined Floating-Point Adder and Multiplier [3,4]**

The multi-mode adder (FPA) and the multi-mode multiplier (FPM) are capable of both single and double precision of IEEE 754. It supports five formats conversion in four formats: single precision, double precision, integer, and block floating-point. The primary requirements to be met by the design of FPA and FPM were:

1. Single and double precision floating-point capabilities
2. 40 MHz clock capability
3. CMOS logic
4. IEEE 754-1985 compliance
5. Block floating-point to floating-point format conversion

These circuits consist of 2 main sections: Exponent processing and Mantissa processing circuits. Each section is pipelined, and both of them are paralleled.

The result showed that FPA and FPM met all of requirements. Moreover, hardware resources among the various modes are cleverly shared, so the area was significantly decreased. However, their control parts are very complicated because they are pipelined and paralleled in a circuit. Beside, two pipelined sections that are paralleled used some stages together, so one of them might be stalled. Thus, circuits might be slower than they should be.

In the future, researchers of FPA and FPM plan to combine them together to operate add and multiply in a circuit.

#### **2.1.1.3 Pipelined Packet-Forwarding Floating Point [5,6]**

This packet-forwarding floating-point circuit that operates as a cooperating adder and multiplier is capable the single precision of the IEEE 754. It employs a four-stage execution sequence with the latter two stages of each being the rounder phase as illustrated in figure 2.2. Moreover, conventional pipeline operation is also illustrated in figure.2.2 too.

The packet-forwarding technique is proposed to handle data hazard, the problem of the order of read/write accesses, in pipeline. Stage one accepts an operand in standard format at its start. Both pipelines accept the packet-forwarding operand in packet at the start of packet one and two, and output their results in packets after stage two and three. Stage four output is rounded and normalized to the standard format for retirement to a register. This design cuts the effective latency in half and reduces the stall cycle because outputs of each stage are forwarded to the next operand as illustrated in figure.2.2.

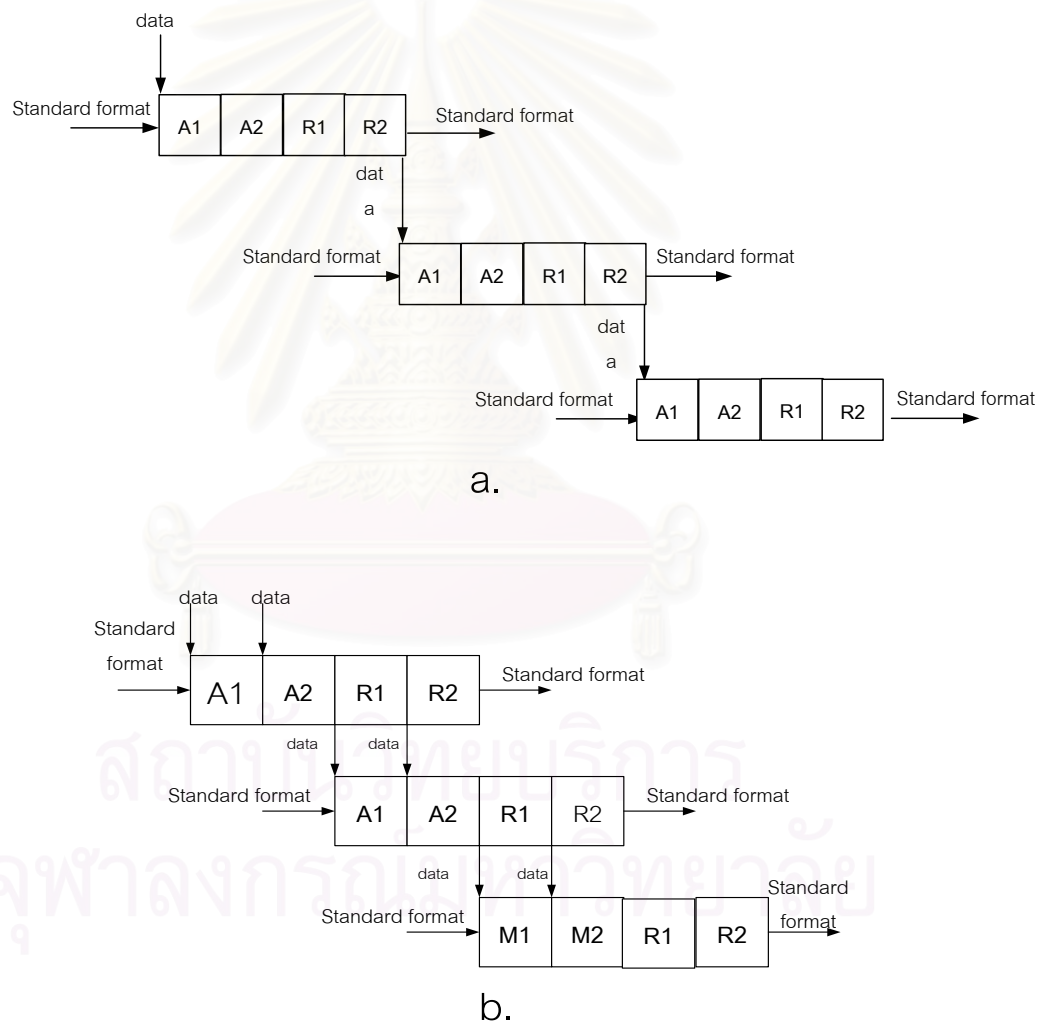


Figure.2.2: 2 types of pipeline

Conventional pipeline (a) Packet-forwarding pipeline (b)

However, this design doesn't have any shared stage despite some stages can be shared for both adder and multiplier. Therefore its area is too big.

#### 1.2.1.4 Amphion Floating-Point Operators [7]

Amphion proposed five 32-bit floating-point operators:

1. **Sm2FP** - sign magnitude to floating point operator; maximal clock frequency is 46 MHz

2. **FP2Sm** - floating point to sign magnitude operator; maximal clock frequency is 48 MHz

3. **FpSubAdd** - floating point adder and subtractor; maximal clock frequency is 34 MHz

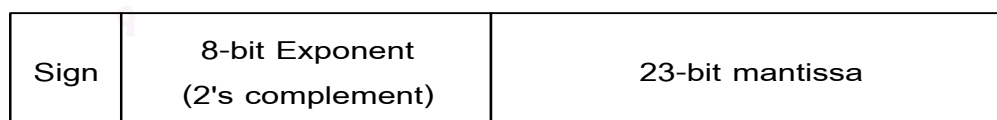
4. **FpMult** - floating point multiplier; maximal clock frequency is 32 MHz

5. **FpNrsDiv** - floating point divider; maximal clock frequency is 32 MHz

The floating-point functions have registers on both the inputs and outputs of the cores. Each function is a stage of pipeline in operators. They also have additional internal pipeline cuts to improve performance accordingly. Because maximal clock frequencies of operators are not equal, when two operators with different maximal clock frequency work together, the slower function will limit the maximal clock frequency of the whole system.

There are 2 styles of number formats in Amphion floating-point operators as below:

1. **Floating point number format:** a floating-point number in a core is represented by 32-bit `std_logic_vector` as illustrated in figure.2.3.



*Figure 2.3.: Amphion's floating-point number format*

The sign bit and the 23-bit mantissa are as same as IEEE 754 standard format. The 8-bit exponent is coded in 2's complement form, so it is easy to add or

subtract the exponent. However, an Amphion's floating point format number to IEEE 754 format convertor must be added when it cooperates with other systems that are capable IEEE 754. The most negative exponent (1000...000) indicates that the value of the floating-point number approach zero.

2. **Sign-magnitude format:** the fixed-point sign magnitude number in Amphion library is represented by a 32-bit `std_logic_vector` as illustrated in figure 2.4.

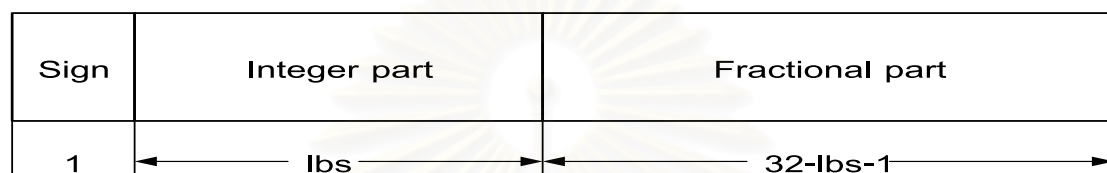


Figure 2.4: Sign-magnitude format

Parameter `lbs` describes the number of integer bits of the sign-magnitude number. This enables the user to control the number of integer bits in the sign-magnitude inputs or outputs to adjust the precision of numbers to the user's application. The `lbs` is provided as an input signal on `Sm2Fp` and `Fp2Sm`.

Nevertheless, the Amphion's floating-point operators are capable the precise exception of IEEE 754.

### 2.1.2 Dynamic Pipeline Controller [8.9]

A number of hardware controls have been proposed to keep performance and correctness of dynamic pipeline. There are 4 factors that must be included in the controller:

1. Generation of an "initiation complete" signal
2. Activation of logic within a stage
3. Route control
4. Function selection within a stage.

There are many ways to implement the controller to meet all four factors. A simple implementation of the first two involves extra bit added to a staging latch that is

clocked just as others in its group are. It is connected to extra bit in the next staging latch by “noncompute” logic that simply replicates the value. This implementation was proposed by cf. Cotton.

The route control and function selection are often similar. Their hardware mechanisms depend on the initiating mechanism, the complexity of the reservation table, and degree of reconfiguration. These mechanisms may fall between two control mechanisms: time-stationary control and data-stationary control.

A time-stationary control mechanism provides the route control and function selection signals for the entire pipeline from a single source external to the pipeline. This mechanism is centralized control.

A data-stationary control mechanism is extremely opposite with the time-stationary control. The control signal “follow” the data through the pipeline providing the control signals at each single as needed. There is no centralized control source. A code enters each stage must be decoded to determine what the logic should do, what path to take next.

The time-station control is simpler, but it is not flexible for operations while the data-stationary is more complex, but it is quite flexible.

### **2.1.3 An example of dynamic pipelined computer [10]**

This example is a design and implementation of a pipelined virtual computer of Eric Kasten from Michigan State University. The virtual computer is a distributed collection of processors connected by a network, and may operate in symphony to complete a task. Dynamic pipeline is a technique chosen to get the best throughput in a virtual computer because there are 2 functions, function A and function B, that use redundant stages in his design. They are image processing functions. The reservation table, the collision matrix, and the state diagram are used for scheduling the pipeline tasks (they is described in section 2.3). Figure 2.5 and 2.6 are definitions of function A



and B respectively. Function A and function B can be combined to a dynamic pipeline that can perform both functions as shown in figure 2.7. This virtual pipeline computer was implemented by PVM 3.4. The two functions were run separately and together by the TCP/IP protocol stack over both an Ethernet and atop ATM network. In addition, they were implemented and run in serial. The result presented that dynamic pipeline implementation provided dramatically performance over serial implementation.

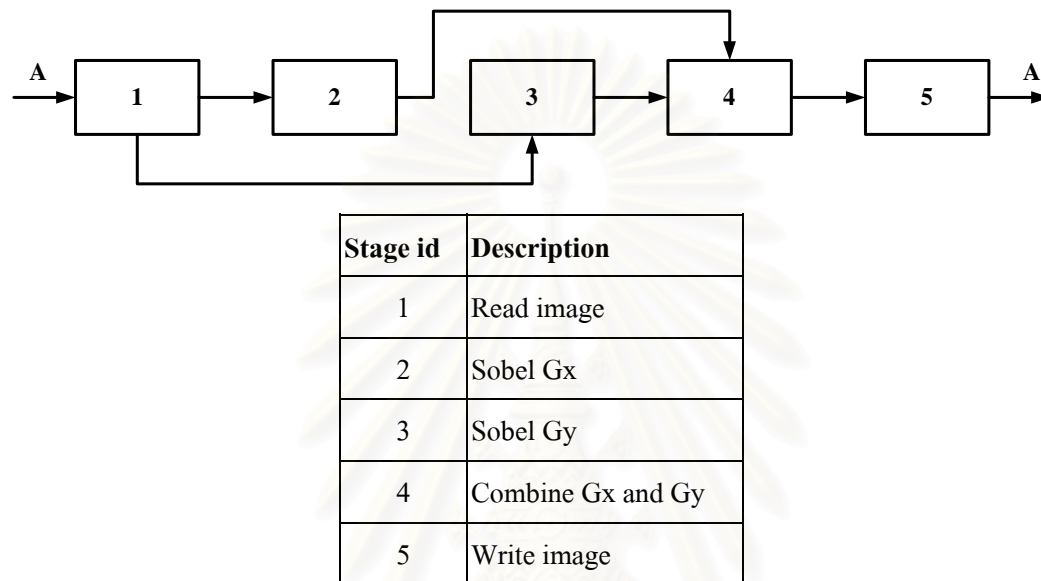


Figure 2.5: Definition of function A

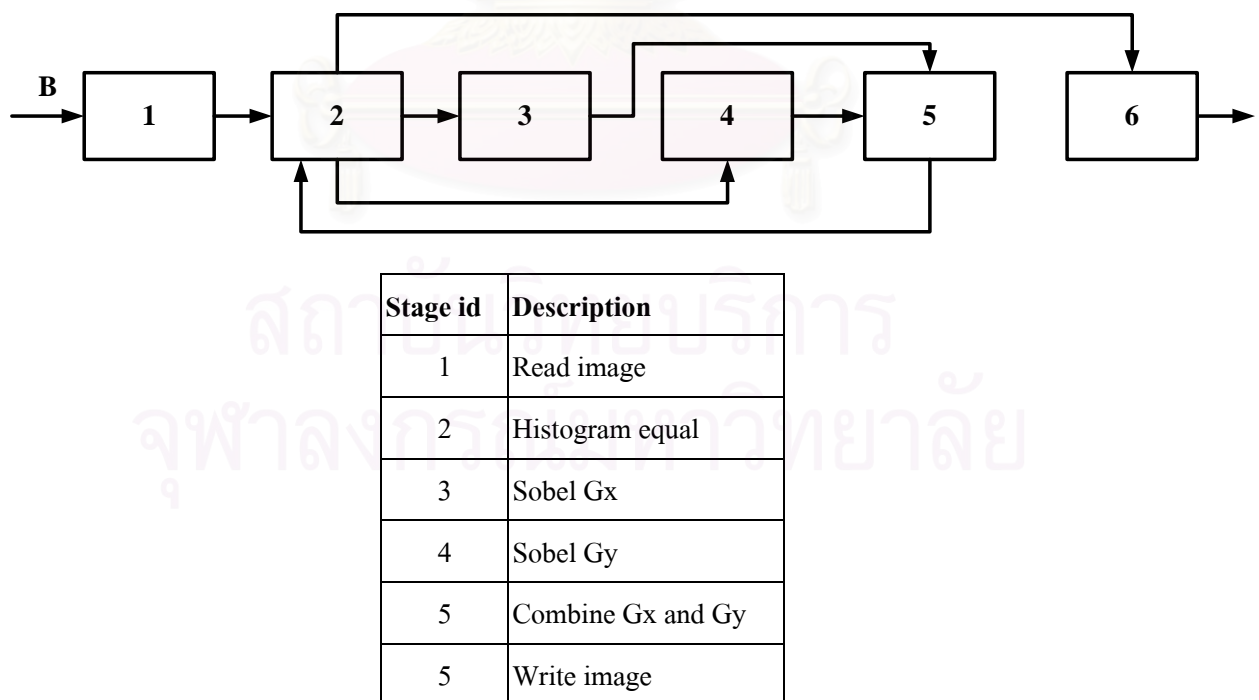


Figure 2.6: Definition of function B

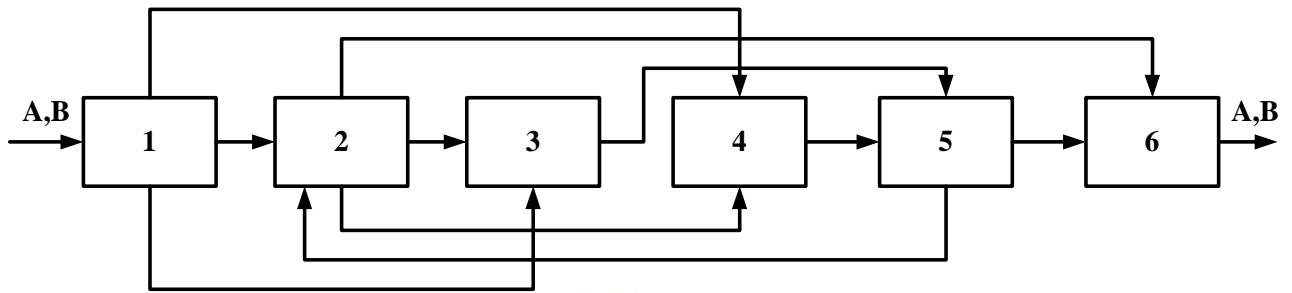


Figure 2.7: Dynamic Pipeline for function A and function B

## 2.2 Asynchronous pipeline

Asynchronous pipeline was first introduced in 1989 by Ivan E. Sutherland who named his pipeline “micropipeline” [11]. It is used to improve performance and speed of asynchronous circuits.

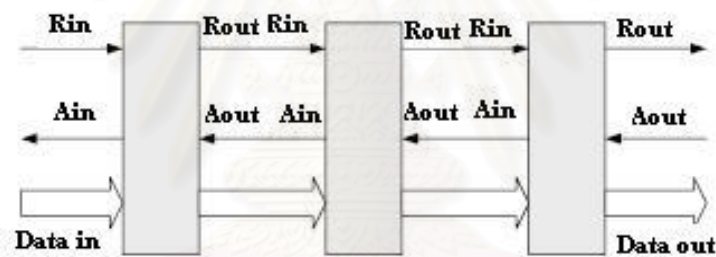


Figure 2.8: An asynchronous pipeline structure

Figure 2.8 represents the asynchronous pipeline structure used in this research. It uses bundle data channel that consists of two control signals, request and acknowledge, and data. The request signal is sent to the next state when data is valid, and the next stage send back the acknowledge signal. In addition, a stage of pipeline is composed of combination circuit (s) and latch (s). The request signal and the acknowledge signal labels depend on the direction of data; for example the request signal is called R out at the output of stage I and becomes Rin at input of stage i+1.

According to transitions of control signals, the asynchronous pipeline falls into 2 protocols.

### 2.2.1 Two-phase signalling protocol

This protocol used in Sutherland's micropipeline. Either rising or falling transition of control signals has the same meaning, so the signal level has no significance. Its event is indicated by a change in logic level (either '0' to '1' or '1' to '0'), and the next event is simply the opposite transition. The latch employed on two-phase micropipeline is edge-sensitive.

### 2.2.2 Four-phase Signaling Protocol

Here an event is indicated by a level (often logic '1') and before the next event the wire must return to zero [12] as represented in figure.2.9. The latter protocol was developed by the AMULET group [13, 14] to eliminate two-phase to four-phase converter that make circuits more complex because latches used their micropipeline are level-sensitive. Now, four-phase micropipeline is more popular and used by a number of researches such as [13, 14, 15] etc. In our design, the latter protocol is used because level-sensitive latches are used.

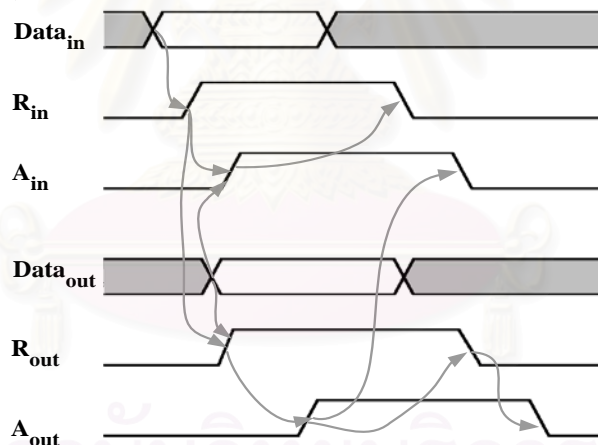


Figure.2.9: The four-phase micropipeline

Our design uses semi-coupled four-phase micropipeline of S.B. Furber and Paul Day [14] because latches in circuits are level-sensitive latches. Level-sensitive latch is a better choice because it is more popular than edge-sensitive latch. Moreover, synthesis of VLSI circuits from Signal Transition Graph (STG) used in this thesis supports four-phase signaling protocol. If edge-sensitive latch were applied to our circuits, converters for two-phase and four-phase protocol would be required. To increase the

decoupling between input and output side of the latch, and to allow the circuit to fill all its stages, two internal signals, A and Lt, are added to the semi-coupled control circuit as shown in figure. 2.10.

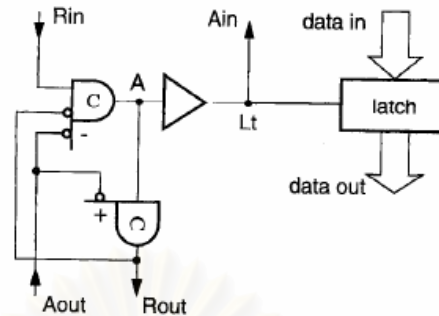


Figure.2.10: Semi-decoupled control circuit [14]

### 2.3 Pipeline scheduling

Pipeline scheduling is important for pipeline to guarantee its performance and collision avoidance. If pipeline is a purely linear, the scheduling is simply. Unfortunately, pipelines used in real circuits are more complex, and scheduling also becomes complicated.

The first thing should be known when studying about pipeline scheduling is types of pipeline. There are two types of pipeline.

1. **Static pipeline:** a static pipeline can be reconfigured to perform only one function (unifuction) at anytime.
2. **Dynamic pipeline:** a dynamic pipeline can be reconfigured to perform variable functions (multifunction) at different times.

Figure.2.11 represents an example of a four-stage pipeline. This pipeline can perform two functions; X and Y. If the pipeline can perform only one function, X or Y, at anytime, it is static pipeline; otherwise it is the dynamic pipeline.

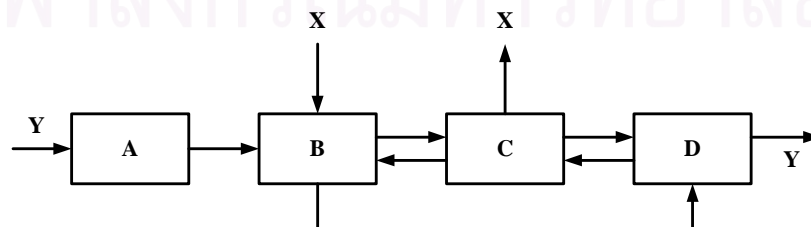


Figure. 2.11: A four stages pipeline

The most important for pipeline scheduling algorithms is two restrictions.

1. The execution time for all stages is a multiple of a stage (some basic clock for synchronous pipeline).
2. Once a computation starts through a pipeline, its time-pattern of stage usage is fixed.

The scheduling for dynamic pipeline is significantly more complex than static pipeline because of the complexity of stage usage.

The pipeline scheduling algorithms used in our design based on three keys: the reservation table, the collision matrix, and the state diagram.

### 2.3.1 Reservation table

Reservation table represents the dataflow of pipeline. It is described in a two-dimension tabular. Each row of the reservation table corresponds to the time usage of each stage. Each column is diagram of the internal usage of a pipeline at an instant of time. One reservation can represent the dataflow of only one function of the pipeline. The number of column is evaluation time that is the total time to complete a function. Reservations tables of three functions of the pipeline in figure.2.11 are shown in table 2.1 and 2.2.

*Table 2.1: reservation table for function X*

stage /time	0	1	2	3
A				
B	X			
C		X		X
D			X	

*Table 2.2 reservation table for function Y*

stage /time	0	1	2	3	4
A	Y				
B		Y		Y	
C			Y		
D					Y

### 2.3.2 Collision Matrix

An initiation of a reservation table occurs when a function started. The number of time unit between two initiations is latency. When two or more initiation try to use a same stage in a pipeline at the same time, the collision occurs. Forbidden latency is the latency that causes the collision.

Forbidden and non-forbidden (permissible) latencies in a reservation table are used to build a collision vector.  $M$  is the evaluation time of table; collision vector =  $(C_0, C_1, \dots, C_{m-1})$ .  $C_i$  is 0 if latency  $i$  is the permissible latency.  $C_i$  is 1 if latency  $i$  is the forbidden latency. For example, collision vector for function  $X$  is 10100; it is initial collision vector.

Collision vector is sufficient for static pipeline because it performs only one function. For dynamic pipeline, collision matrix is applied. A collision matrix  $C$  is an  $r \times t$  binary matrix where  $r$  is number of reservation tables in a pipeline, and  $t$  is a maximum evaluation time of tables[15]. The  $j$ th row of the  $i$ th matrix,  $CM_i$ , is the collision vector between an initiation of a reservation  $i$  and a later initiation of reservation  $j$ . In all cases the  $i$ th row of  $CM_i$  is the same as the initial collision vector for function  $i$ . This collision matrix is called initial collision matrix. For example, the initial collision matrix for the pipeline in figure 2.11 is:

$$CM_x = [10100 \quad 01000]$$

$$CM_y = [01110 \quad 10100]$$

### 2.3.3 State Diagram

A state diagram gives two information that are when to make new initiation without collision and the current state of a pipeline (current collision vector in static pipeline and collision matrix in dynamic pipeline). The initial collision matrix is states one. They are shift left until at least one of first bits in the matrices is 0. Then, they are ORed with OR results between initial collision matrices that their first bits are 0, and become the current states. The next state is the OR result between the current state and

the OR result of initial collision matrices. The procedure in detail is in [8] and [9]. The state diagram of the pipeline in figure.2.11 is shown in figure.2.12. Moreover, another example of a system that adopted the dynamic pipeline is Design and Implementation of a Pipelined Virtual Computer of E.Kasten [10].

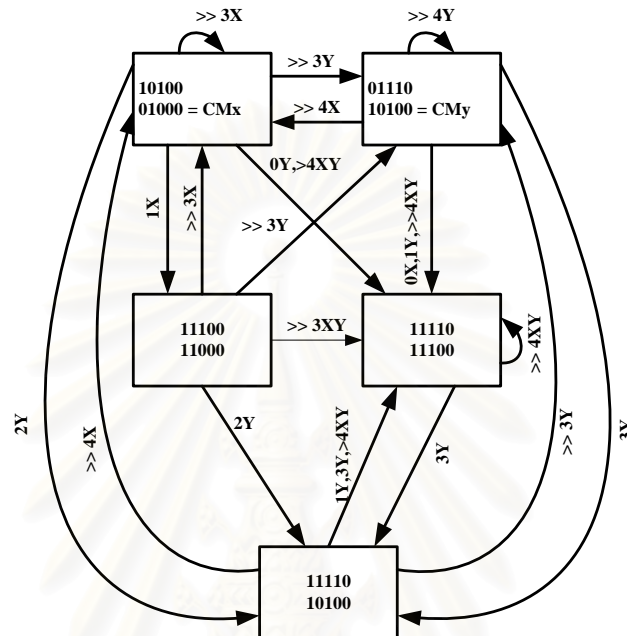


Figure.2.12: The state diagram of the pipeline in figure.2.11

## 2.4 Asynchronous circuit design

### 2.4.1 Environmental and delay Model

The two main things that define the asynchronous design style are environment model and delay model. They affect the asynchronous circuits directly because the environment model defines a protocol between them and their environment, and the delay model defines delay in their gates and wires. If designers use too pessimistic environment and/or delay model, their circuits can be very hard to design, but inefficient and expensive. On the other hand, if they choose too optimistic environment and/or delay model, their circuits are easy to design, but can't be guaranteed the correctness of circuit operations.

### 2.4.1.1 Environmental model

An asynchronous system is composed of a set of circuits and environment as shown in figure.2.13. An environment sends a set of inputs to a circuit, and the circuit sends a set of input to the environment.

There are three styles of environment model:

1. **The fundamental mode (Huffman mode):** all gates in all circuits should be stable before they receive a new set of inputs from the environment.
2. **The generalized mode operation:** all gates in a circuit should be stable it receives a new set of inputs from the environment.
3. **The input/output mode:** the environment can send a new set of inputs to circuits whenever it receives a set of outputs from a circuit.

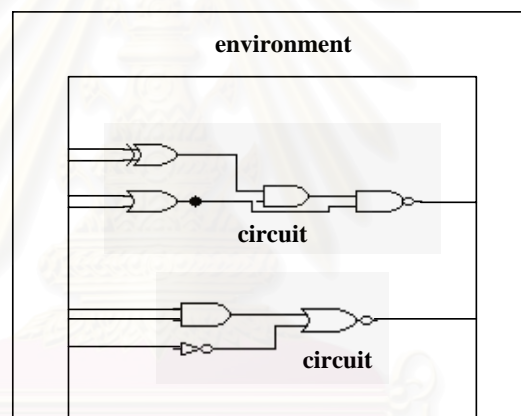


Figure 2.13: An asynchronous system

### 2.4.1.2 Delay model

According to delay variance in gates and wires in a circuit, delay model can be divided into three main styles:

1. **The Fundamental-mode Model (The Huffman Model):** gate and wire delay are bounded and the designer known the upper bound.
2. **The Speed-Independent Model (The Muller Model or SI):** gate delay is unbounded and no wire delay.
3. **The Delay-Insensitive Model (DI):** gate and wire delay are finite, but unbounded.



The DI model can guarantee the correctness of circuits, but they are hard to design and can include only C-element in order to ensure that their operations are correct. The C-element is a storage element that its size is quite big as shown in figure.2.14. Its output becomes 0 when all of its inputs are 0, and becomes 1 when all of its inputs are 1; otherwise its output is the previous output.

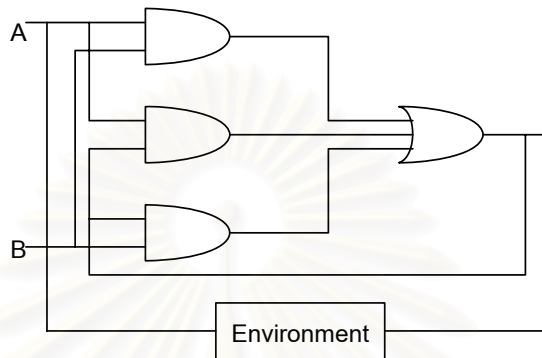


Figure 2.14: A Two-Input C-element

Isochronic fork is added to DI model to decrease the complexity and size of circuits; the new model called the Quasi-Delay insensitive model (QDI) [16]. The isochronic fork is a fork interconnection, and delay in all branches are the same as shown in figure 2.15. Signal d1, d2, d3, d4 are delay; d3 and the d4 are in the fork interconnection, so they are the same delay. QDI circuit is equivalent to the SI circuit if a sequence of inputs from an environment is sent to its circuits in order of its sequence.

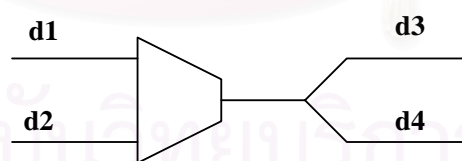


Figure 2.15: The quasi-delay insensitive model

For our design, the input /output mode model and QDI are used to design circuits.

## 2.4.2 Dual-rail encoding

As I described above that there is no global clock in asynchronous circuits, the single-rail implementation, a signal line for 1 bit-data, is not sufficient to represent the data in asynchronous circuit. If the current value of data is 1, and the new value is also 1,

we can not distinguish the arrival of the new data. Therefore, The dual-rail implementation is used to represent the 1-bit data: a signal line for logic '0', and the another for logic '1'. The following example is the dual-rail encoding presented by TITAC [16]: d0 for logic '0', d1 for logic '1'. At any time, only one of a pair of 1-bit data can be '1'; otherwise, it's an invalid data. Beside, (0,0) is a spacer code word used to distinguish between the current data and the new data. Both d0 and d1 must be reset to 0 before the arrival of the new data.

$$D = 0 \leftrightarrow (d1, d0) = (0, 1)$$

$$D = 1 \leftrightarrow (d1, d0) = (1, 0)$$

$$(0, 0) = \text{Spacer}$$

$$(1, 1) = \text{Invalid}$$

## 2.5 Synthesis of asynchronous circuits from signal transition graph

### 2.5.1 STG

STG is a graph-based method used to describe a behavior of a circuit. It is an interpreted Petri-Net introduced in 1987 by T.A. Chu [17]. It can describe concurrent and choice operation in a circuit. There are 3 types of signals: input, output, and internal signal. Input is distinguished by underline. Rising and falling transition of signal  $t$  represented by  $t+$  and  $t-$ . Arcs represent the relationship between signals. For one STG, it is a single cycle of STG if signals have only one rising and falling transition, otherwise it is multicycle. STG must satisfy 2 properties, which are live, and CSC (complete state coding). There is not any deadlock if the STG satisfy the first property. The second property prevents malfunction of a circuit.

**Definition 1 (CSC property) [18]:** A live STG satisfies the complete state coding (CSC) property if

1. Every state on its reachability graph has a different binary code, or
2. When two or more states have an identical binary code, all the internal and output signal transitions enabled in these states are the same

A single cycle STG of C-element in figure 2.14 shown in figure.2.16 , and it satisfy both two properties.

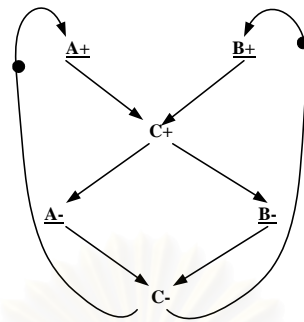


Figure.2.16: a STG of two input C-element

### 2.5.2 Asynchronous circuit synthesis procedures from STG

The circuit synthesis procedures from STG used in our design based on Park's [18]. Circuits operate correctly under the input/output mode and QDI. A signal network is a circuit structure used to implement every non-input signal. Each signal network is composed of four subnetwork; the set region network, the set acknowledgement network, the reset region network and the set acknowledgement network as shown in figure.2.17.

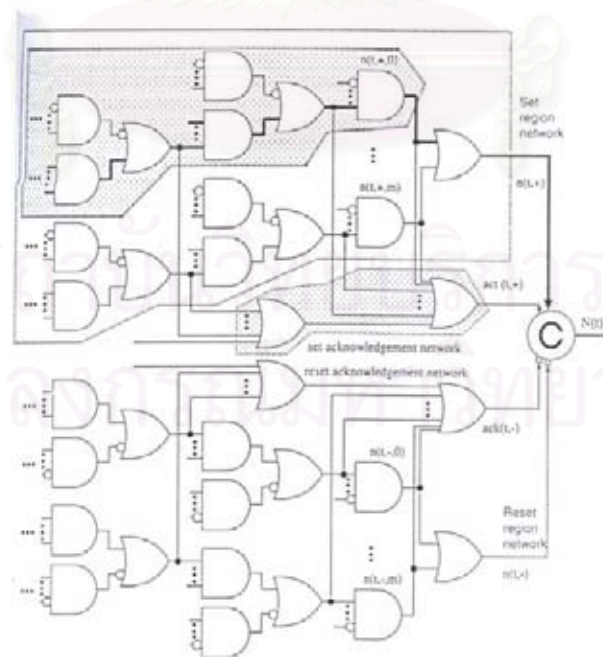


Figure.2.17: Signal network  $N(t)$  for signal  $t$  [18]

**Definition 2 (Region)[18]:** A maximal set of transitions which can be fireable from  $t^*$  to the immediately following  $t^*$ (not including  $t^*$ ) is in a region for a signal  $t$ .

**Definition 3 (Set/Reset Region)[18]:** if a region for signal  $t$  starts from transition  $t^{+/i}$  ( $t^{-/i}$ ), it is a set (reset) region of signal  $t$ .

The set (reset) region network  $n(t, +)$  ( $n(t, -)$ ) of signal  $t$  is constructed with a set of region network which implement all the rising (falling) transitions of signal  $t$ , and output to the C-element. Each region network  $n(t, *, i)$ ,  $i$  are the number rising (falling) transitions of signal  $t$ , is constructed by AND gate and OR gate. Region network for all rising (falling) transitions must be ORed before output to the C-element. The set (reset) acknowledgement network used to check if each region network completes its operations is constructed by a set of OR gate.

There are two methods for circuit synthesis: synthesis based on lock relations and synthesis based on state assignment.

### 2.5.2.1 Synthesis based on lock relations

This method can be used with single cycle STG which have no choice operations. There are 5 lock relations used for this method; semi,full, associate, transitive,and super lock [18]. The following definitions described lock relations are from Park's thesis.

**Definition 4 (semi-lock):** If there are two signal  $a$  and  $b$  such that  $a^* \rightarrow b^* \rightarrow (a^*)'$  or  $b^* \rightarrow a^* \rightarrow (b^*)'$  on a simple cycle, they are semi-locked.

**Definition 5 (full-lock):** If there are two signal  $a$  and  $b$  such that  $a^* \rightarrow b^* \rightarrow (a^*)' \rightarrow (b^*)'$  on a simple cycle, they are fully-locked.

**Definition 6 (associate-lock):** When a minimal set  $A$  of fully locked signals and signal  $b$  are such that  $\exists a_1, a_2 \in A : a_1 \rightarrow b^* \rightarrow a_2 \rightarrow (b^*)'$  on a simple cycle, then  $A$  and  $b$  are associatively locked, and  $A \cup b$  becomes a level-0 transitive-lock set.

**Definition 7 (super-lock):** When a set of fully locked signals and signal  $t$  are associatively locked, and  $t^*$  is concurrent with a transition in set  $A$ , then  $t^*$  has the super-lock relation with set  $A$ .

**Definition 8 (transitive-lock):** When a signal  $b$  and  $s$  level- $l$  transitive-lock set  $A$  are such that  $\exists a_1, a_2 \in A : a_1 \rightarrow b^* \rightarrow a_2 \rightarrow (b^*)'$  on a simple cycle, then  $A$  and  $b$  are a transitively locked, and  $A \cup b$  becomes a level- $(l+1)$  transitive-lock set.

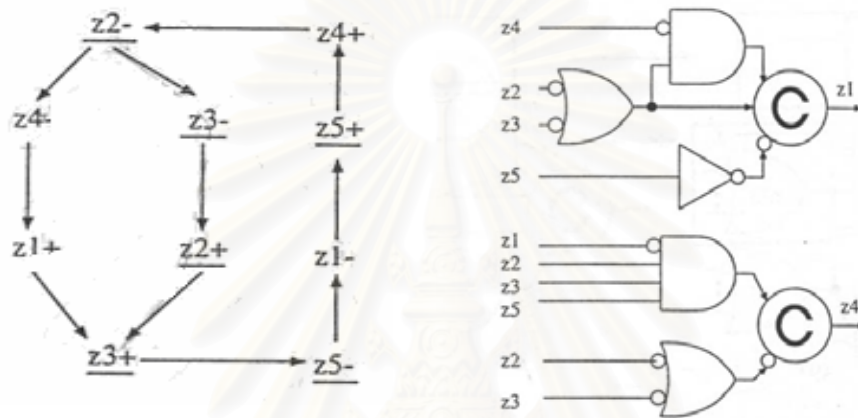


Figure.2.18: An example of synthesis based on lock relation. [18]

Semi-lock used to find full lock. Every relation, except semi lock, used to satisfy the four-phase handshake protocol of signals on a circuit. These lock-relations used to construct a signal network. Assume that  $a^+$  is a signal transition and  $b$  is a trigger signal of  $a$ . If  $a$  and  $b$  are fully-locked; it's sufficient, otherwise find another lock-relations between them and the others signal in order. If  $a$  and these signals are ordered, then put signals as inputs of AND gate in set region network. If  $a$  and these signals are concurrent, then put signals as inputs of OR gate. Input signal of OR gates must be super-lock.

Figure.2.18 is an example. First step is finding semi-lock relations . We find that a signal  $z1$  has the semi-lock relation with signal  $z2$ ,  $z3$ , and  $z5$ . More over, A signal  $z2$  has the semi-lock relation with  $z1$ ,  $z3$ , and  $z4$ . Next, we use semi-lock set to find full-lock relation. As a result, a signal  $z2$  has the full-lock relation with signal  $z3$ , and

signals  $z_1$  and  $z_5$  have the fully lock with each other. After that, we can find the associate lock between a signal and set of full-lock relation signals:  $(z_2, z_3)$  and  $(z_1, z_5)$  if, a signal is ordered with full-lock relation signals. A signal  $z_4$  has the associate-lock relation with a set  $(z_2, z_3)$  because of transition sequence of  $z_2^- \rightarrow z_4^- \rightarrow z_3^+ \rightarrow z_4^+$ . Furthermore, we can use sets of full-lock relation signals to find a super-lock relation, if a signal is concurrent with full-lock relation signals. Accordingly, a transition  $z_4^-$  is super-locked with the set  $(z_2, z_3)$  because  $z_4^-$  is concurrent with  $z_3^-$ . The next step is finding level 1 transitive-lock relations from sets of associate-lock signal:  $(z_2, z_3, z_4)$ . A signal  $z_1$  is transitively locked with a set  $(z_2, z_3, z_4)$  of associatively locked signal. After a set  $(z_1, z_2, z_3, z_4)$  is found, a signal  $z_5$  is a transitively locked with the set. As a result, all the signals in a given STG are included in level-2 transitive-lock signal.

After we get all of the lock relations in a given STG, we can synthesis circuits from those lock-relations. A circuit is synthesized for an output signal. For the example STG, there are two output signals, so we must synthesis two circuits for  $z_1$  and  $z_4$ . Each signal consists of two regions network: set region network and reset region network.

First, we will consider a circuit for  $z_1$ . For set region network  $n(z_1, +)$  for  $z_1^+$ ,  $z_4^-$  is its trigger, so we must find a full-lock relation between  $z_1$  and  $z_4$ . Fully-lock relation between  $z_1$  and  $z_4$  is not found. So, we must find an associate-lock relation between them. There is also no any associate-lock relation, but there exists a level1-transitive-lock relation with  $z_1$  and a set  $(z_2, z_3, z_4)$ . After this level-1 lock relation is found, finding a lock relation is over. Signal  $z_4$  is ordered to signal  $z_1$ , so it is assigned to the AND-gate for  $n(z_1, +)$ . Transitions  $z_2^+$  and  $z_3^-$  are concurrent with  $z_1^+$  so they are assigned to an OR-gate, and the output of the OR-gate inputs to the set acknowledgement network. Since there is only one OR-gate for the region network, the set acknowledgement network is constructed with a wire. For  $z_1^-$ , its trigger  $z_5^-$ , and  $z_5$  is fully-locked with  $z_1$ . As a result, only  $z_5$  is sufficient for reset region network.

Second we will consider a circuit for  $z_4$ . In case of set region network  $(z_4, +)$  for  $z_4$ . Its trigger transition  $z_5^+$  is transitively locked with a set  $(z_1, z_2, z_3, z_4)$ .

Thus, we  $z_1, z_2, z_3, z_4,$  and  $z_5$  are used to implement the set region network of  $z_4$ . They all are assigned to the AND-gate for the region network because there is no signal which has the super-lock relation with transition  $z_4^+$ . For reset region network  $n(z_4,-)$ , its trigger is  $z_2^-$ . A super-lock relation between  $z_4$  and full-lock relation set  $(z_2, z_3)$  is found. So,  $z_2$  and  $z_3$  are assigned to an OR-gate. The reset acknowledgement network consists of a wire because there is only one OR-gate for the reset acknowledgement network. Accordingly, the acknowledgement is not required because there is only one OR-gate and no- AND-gate.

### 2.5.2.2 Synthesis based on state assignment

The current state of a given STG is decided by a combination called a state code of values of all state variables. Let consider states code of figure.2.12 signals are ordered alphabetically, a state code position are  $crr, cs, eq, eqbar, mrr, trr, twa, xen$ . Examples of state code are in table 2.3. In addition, trigger cube must be considered to construct a circuit. The trigger cube  $TC(t^*/1)$  of  $t^*/1$  is a state code in which the value of state variables that are not trigger signals of  $t^*/1$  are changed to the don't care. Examples of state codes and trigger cubes of a signal  $crr$  are also shown in table 2.4. State code and trigger cube of every element in a circuit must be concerned, if there is any CSC in the given STG, it must be modified. Adding internal signal to the STG is a choice to modify the STG. An example of synthesis based on state assignment shown in figure.2.19

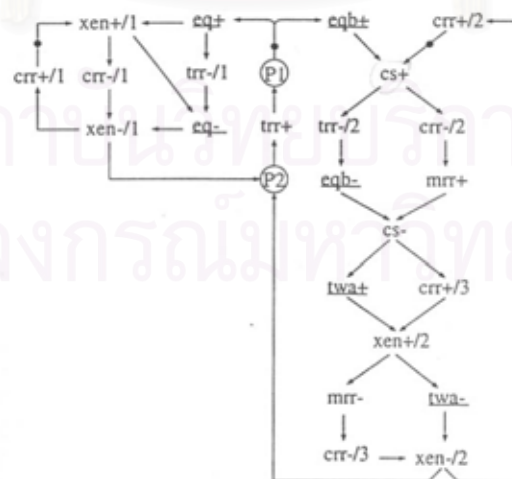


Figure.2.19: An example of synthesis based on state assignment.[18]

Table 2.3 Examples of state code [18]

Element	State code	Element	State code
p1	x0000100	p2	x0000000
crr-/1	10x00x01	crr-/2	110x0x00
crr-/3	100000x1	crr+/1	00x00x00
crr+/2	000x0x00	crr+/3	000010x0
cs-	01001000	cs+	10010100
eq-	x0100001	eq+	x0000100
eqbar-	x101x000	eqbar+	x0000100
mrr-	100010x1	mrr+	010x0x00
trr-/1	x0000000	trr-/2	x101x100
trr+	x0000000	twa-	x000x011
twa+	x0001000	xen-/1	00000001
xen-/2	00000001	xen+/1	10100x00
xen+/2	10001010		

Table 2.4 State codes and trigger cube of crr [18]

Transition	State code	Trigger Cube
crr-/1	10x00x01	xxxxxxx1
crr-/2	110x0x00	x1xxxxxx
crr-/3	100000x1	xxxx0xxx
crr+*1	00x00x00	xxxxxxx0
crr+/2	000x0x00	xxxxxxx0
crr+/3	000010x0	x0xxxxxx

The resulting circuits for signal crr are as follows.

$$n(crr,+) = \overline{cs.xen}$$

$$n(crr,-,1,3) = \overline{mrr.xen}$$

$$n(crr,-,2) = cs$$



## 2.6 Binary floating-point representation

Floating-point number can represent smaller and bigger number than integer, so very large number or very small fraction can be represented. It is called floating –point because it represents number in which the point is not fixed. The floating-point number must be normalized to a format that the most significant digit of the significand is nonzero. for decimal, numbers in front of point can be 1 to 9. For example,  $0.852 \times 10^6$  must be normalized to  $8.52 \times 10^5$ . For binary, numbers in front of point can be only 1, and the normalized number is one in the form.

$$(-1)^S \times 1.bbb\dots b \times 2^E$$

IEEE754-1985 is the standard for floating point representation and arithmetic as shown in figure.2.20.

Sign Bit (S): indicates the sign of number: 0 = positive, 1 = negative.

Exponent (E): the representation used is biased representation. A fixed value, bias, is subtracted from the field to get the true exponent. The bias equals  $(2^{k-1} - 1)$  where k is the number of bits in the binary exponent, so bias equals 127 for single format, 1023 for double format.

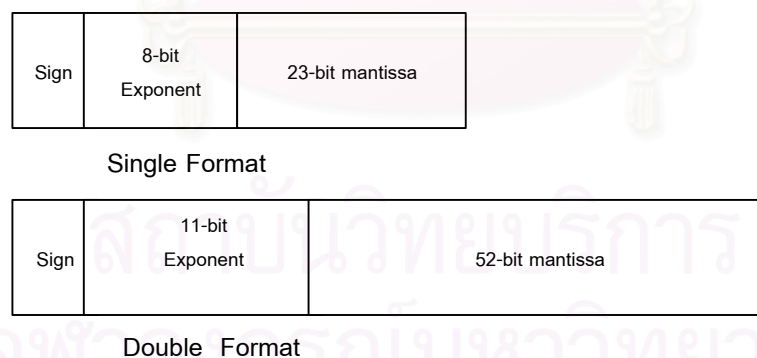


Figure 2.20: IEEE 754-1985 representation

Mantissa (b): (sometimes called Fraction or Significand) for binary, base b is always 2, so it is implicit and need not be stored. Similarly, 1 in front of the point is implicit and need not be stored in mantissa.

For my design, I use the single format (or single precision) of IEEE 754 standard including precise exception and rounding. The details of floating-point are in [20].



สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

## CHAPTER III

### CONTROL UNIT

This chapter is about the control unit used for dynamic pipeline scheduling and stage control. The control unit is described in 3 sections. Section 3.1 describes overall of the control unit composed of 2 main parts: pipeline scheduler and stage control. The first and second parts in details are discussed in section 3.2 and 3.3 respectively. Since the configuration of dynamic pipeline used in floating point arithmetic unit is quite complex, the simpler example is illustrated in figure. 2.11 is discussed in this chapter for better understanding. The last section 3.4 is about circuit design and circuit synthesis.

#### 3.1 Overall of Control Unit

The asynchronous dynamic pipeline control unit is shown in figure 3.1. As described above that the control unit consists of two main parts (two blocks in a dash block). They are interconnected with control signals. The pipeline scheduler handles all of pipeline scheduling and collision check, while stage controller controls function and data transfer of every stages in the dynamic pipeline.

Three signals:  $f$ ,  $n_f$ , and  $f_i$  are sent from the outside of the controller; such as CPU etc. Signal  $f$  goes high when one (or more) function is initiated; in contrast, signal  $n_f$  goes high when no function initiated in this cycle. Signal  $f_i$ , one bit for one function, used to indicate initiated functions: 0 function is not initiated, 1 function is initiated. In this research I assume that three signals are always valid, and will not be checked by the controller.

Two signals:  $p_c$  and  $f_s$  is sent to outside. Signal  $p_c$  sent from stage controller goes high when operations of a cycle are complete. Signal  $f_s$ , one bit for one function, indicates functions that are executed in a cycle. Moreover,  $f_s$  is also gotten by the stage controller for correct operation of every stages. These two signals are important for the outside unit to get correct inputs from the dynamic pipeline.

Two signals,  $s_o$  and  $n_o$ , interconnected between the two main parts are used to distinguish operations of the stage controller.  $S_o$  (same operation) is activated when there is not any new function that will be executed in a cycle. It will be active if  $nf$  is low or there is not permissible function because of the stage collision. If  $f$  is high, and any function can be executed without stage collision,  $n_o$  (new\_operation) will be activated.

Two sets of signals,  $ct_{si} - ct_{sj}$  and  $c_i - c_j$ , are interconnected between the stage controller and the dynamic pipeline. Set  $i-j$  is stages inside the dynamic pipeline: a, b, c, and d for the dynamic pipeline in figure. 2.3. Signal  $ct_{si} - ct_{sj}$  are used to indicate function and data transfer direction of each stage because a stage may executed more than one function. After stage operation completion,  $c_i - c_j$  are sent back to the stage controller

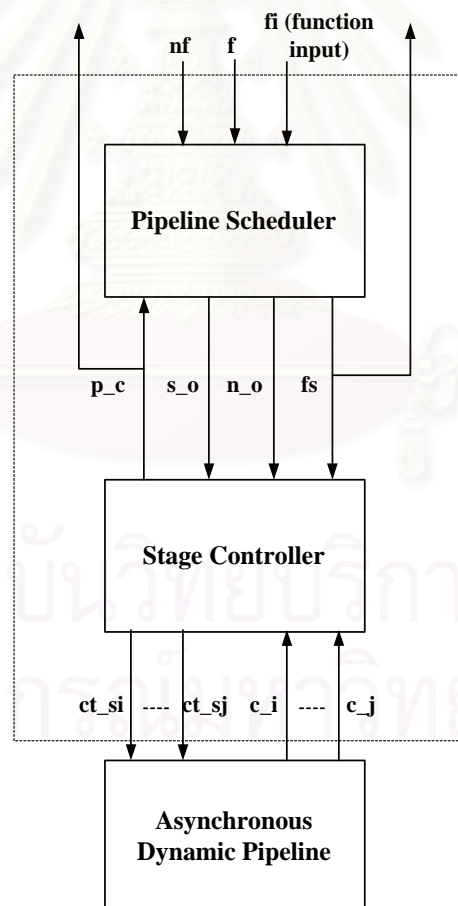


Figure 3.1: Asynchronous Dynamic pipeline controller

### 3.2 The pipeline scheduler

This block consists of four function blocks and a scheduler control as shown in figure 3.2. The design is based on pipeline scheduling scheme described in chapter 2.

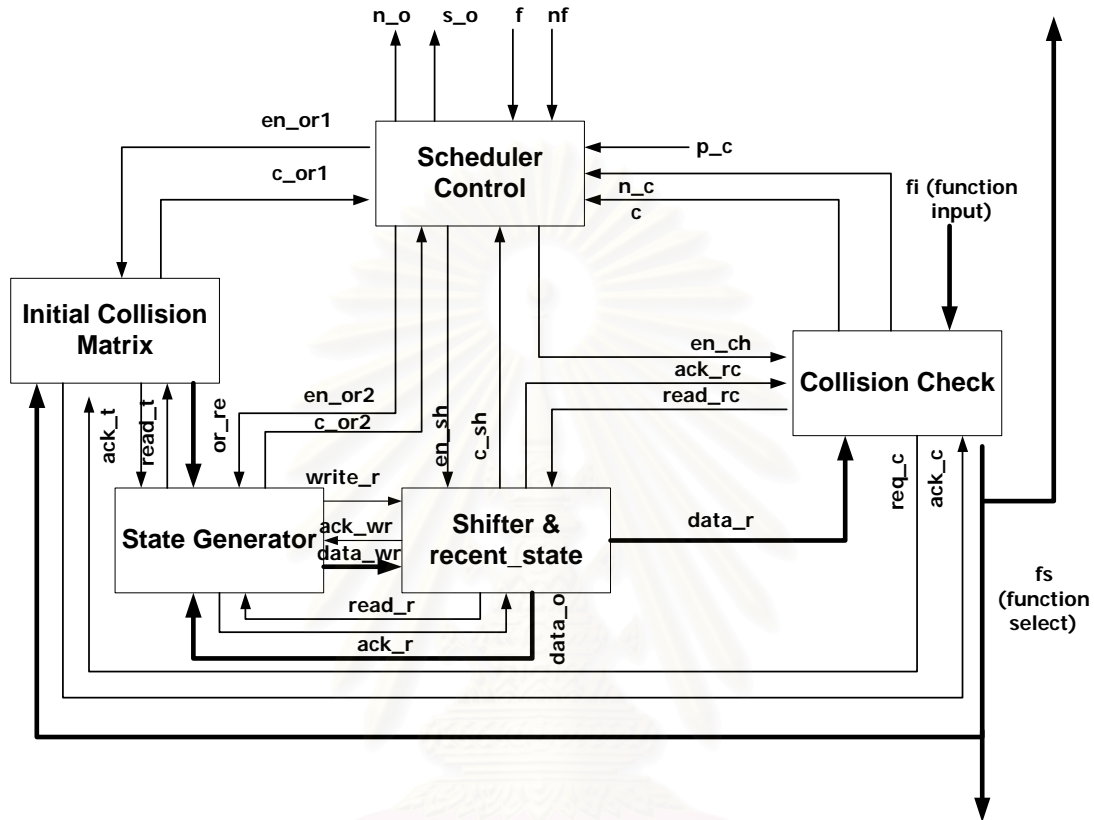


Figure 3.2: Block Diagram of Pipeline scheduler

Before discussing the operation of each block in pipeline scheduler, we discuss overall operation. The operation of pipeline scheduler are described below and illustrated by a flowchart in figure 3.3:

First, scheduler control block receives  $p_c$  signal that represent states operation completed from stage controller, and then shift left registers in Shifter and recent state Block.

Second, The scheduler's control block checks whether there is function comes into pipeline. If  $nf$  is high, there is no new operation for pipeline, and scheduler control block sends back  $s_o$  (same operation) to the stage controller and ends the operation of pipeline scheduler. The shift result becomes a current state of the pipeline. If  $f$  is high,

one or more new functions come into the pipeline, and then collision check block shown in figure 3.6 is enable. It requests current state, and uses that value and  $f_i$  value to check stage collision.  $C$  is high, if the pipeline can not operate those functions without collision, so functions will not be operated at this moment. The scheduler control block sets  $s_o$  high and ends the operation. If  $n_c$  (no collision) is high, the collision check block sends  $f_s$  to initial collision matrix block to select initial matrices that will be ORed together by state generator block that is shown in figure 3.7.

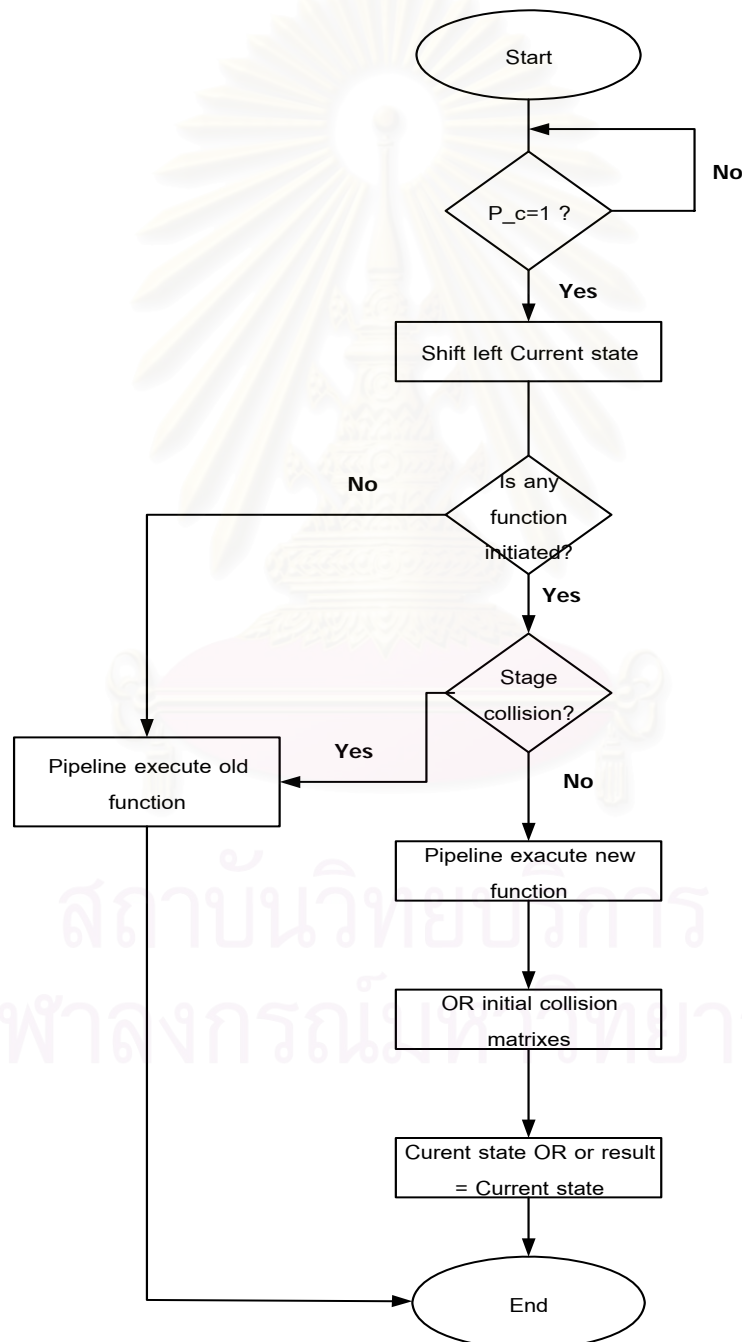


Figure 3.3: Flow chart of Pipeline scheduler

The first block is will be describe is Scheduler Control that is used to control another blocks and synchronize them. There is only a circuit inside because there is no combination circuit here. The circuit is a control circuit, and is generated from STG as we described in chapter2.

For initial collision matrix block Two initial collision matrices,  $CM_X = [101000101000]$ ,  $CM_Y = [011110101000]$  are stored in two registers inside as shown in figure.3.4. Moreover, there are four latches inside this block: two are in Or1 block, one in function selection block and another one is temp\_register. Data in every latch is used in combinations circuits, so 1-2 converter. The 2-1 converter is not required because we can send only data.t to temp\_register. A control block operation is similar to the main control block. First, function selection block receives fs that indicates the permissible function and store it in an inside latch. Then, Or function block uses data in this latch to capture data in CMX or CMY or both. After that, it generates or-result and sends it to temp\_register that is requested by State generator block. Finally, the control block gene

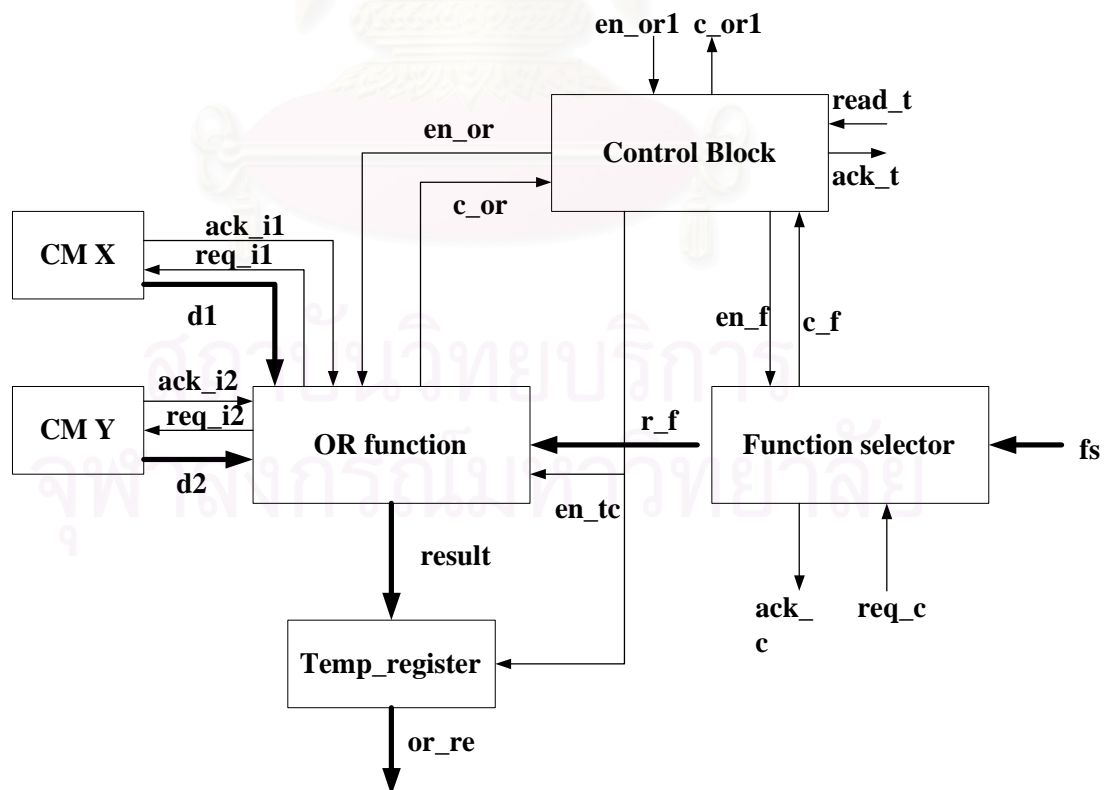


Figure 3.4: Block Diagram of Initial collision matrix block

All 0's are stored in the registers of shifter and recent state block that is illustrated in figure 3.5 at initial time. Size of shift register is as same as CMX. Beside, there is a latch in these blocks: sender\_c, sender\_o and receiver\_o. Data stored in shift register is shifted every cycle, and c\_sh goes high after shift operation is complete. If any function is initiated, collision check block requests first bits of shift register that stored in a latch inside sender\_c. Then, there are two cases of operation: 1. en\_q goes high to indicate that there is no initiated function and more operations are not requires, 2. en\_sh goes high again means functions are initiated so further operations are required. For case 2, data in shift register is requested by State generator block via sender\_o to or with or\_result from Initial Collision Matrix Block. Next, data that was ored by State generator is sent back via receiver\_o and becomes the recent state of dynamic pipeline. Finally, c\_sh goes high again as a completion signal. In this block, operations is shift, send and receive data,so 1-2 converter or 2-1 converter is not required.

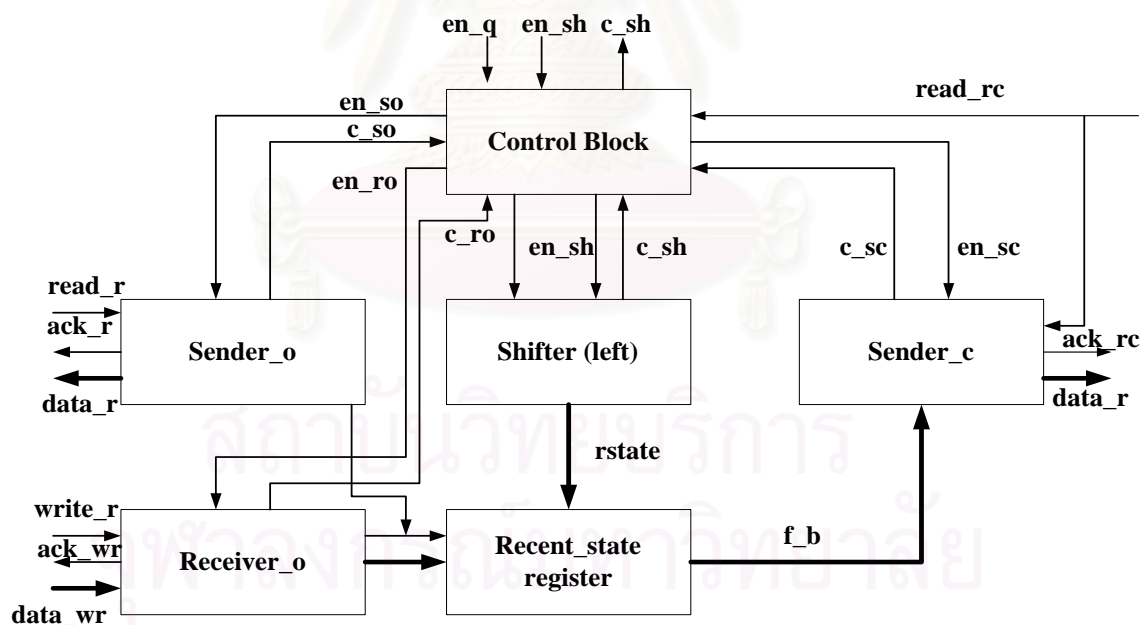


Figure 3.5: Block Diagram of shifter and recent\_state block

The Collision check block shown in figure 3.6 is enabled only when any new function is initiated. It requests first bits of recent state from Shifter and recent\_state



block and then compare with fi bit by bit. If fi is 1 (this function is initiated), collision is occurred when first bit is also 1. We must compare every bit; and then, c (collision) or n\_c (not collision) will go high depends on the characteristics of that dynamic pipeline.

1. If more than one function can be initiated at a same time, c goes high only when all functions cause stage collision.
2. If only one function can be initiated at a same time, c goes high when only one function cause stage collision.

If c goes high, operation is completed. Otherwise, fs is generated and sent to Initial collision matrix block via sender\_or1 block.

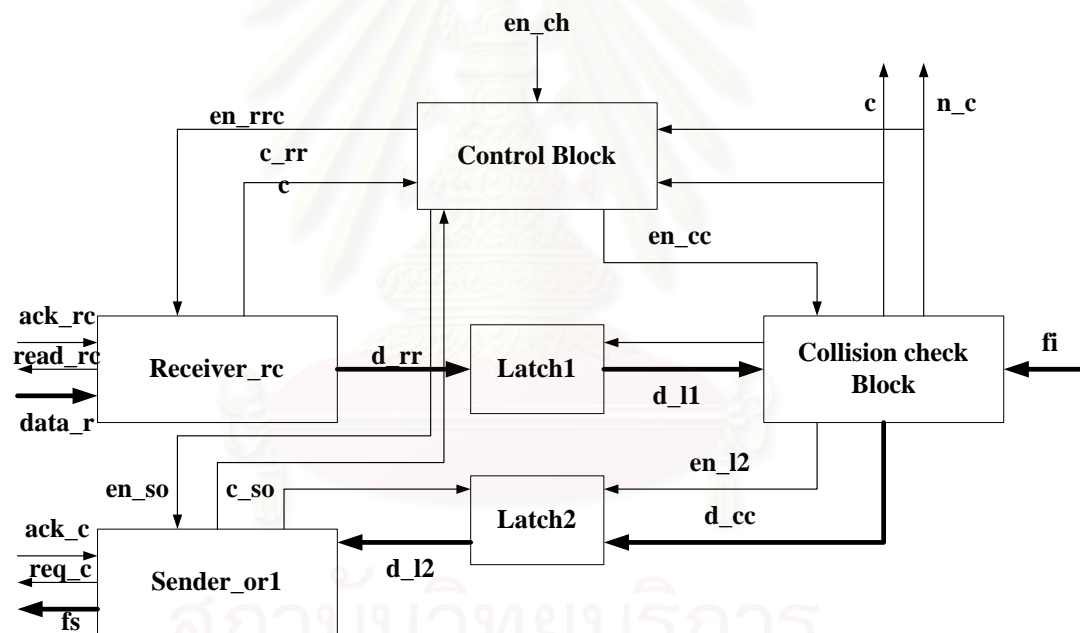


Figure 3.6: Block Diagram of collision check block

The last block of pipeline scheduler is State generator as shown in figure 3.7. State generator is enabled when new function is initiated and stage collision doesn't occur. It receives data from Initial Collision Matrix and Shifter and recent\_state, and stores them in latch 1 and latch 2 respectively. Next, it or them together and stores result in latch3 until Shifter and recent\_state requests it. A combination circuit in Or function is

dual-rail so 1-2 converters are required for latch1 and latch2. Finally, c\_or2 goes high after all operations are completed.

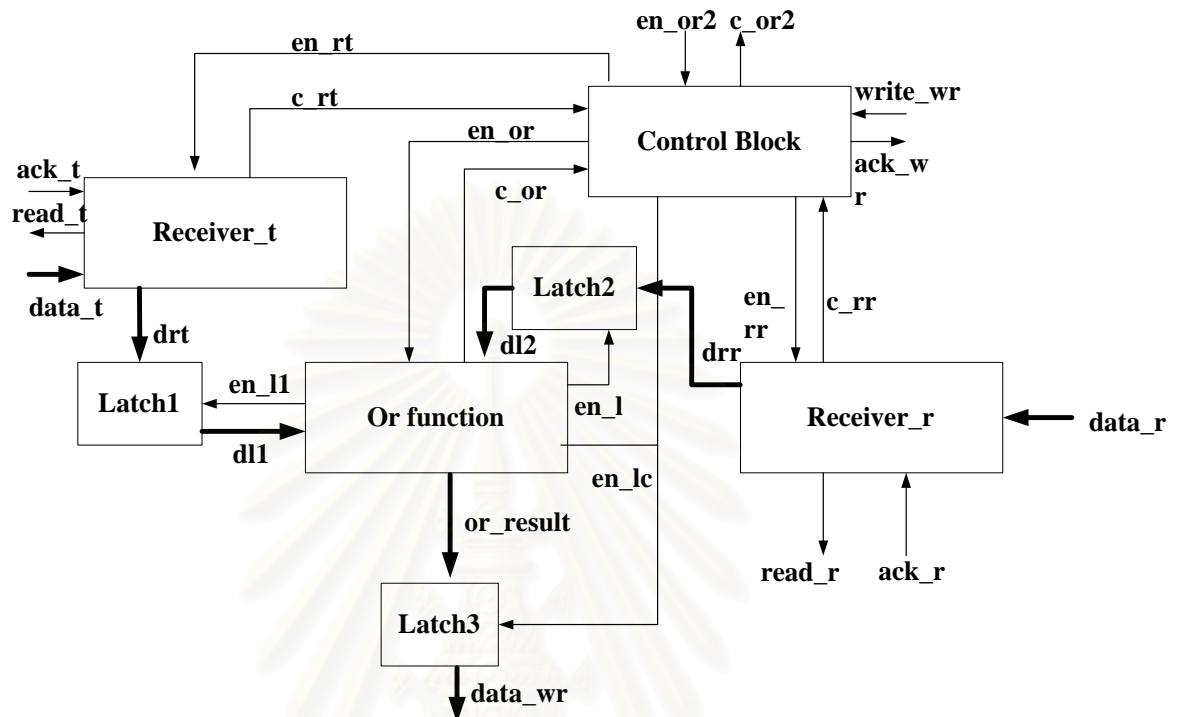


Figure 3.7: Block Diagram of state generator block

Next, ORed result is sent to state generator block that also requests current state. This block ors them together, and send it back as a current state to Shifter and recent state Block.

Finally, the scheduler control block ends the operation of pipeline scheduler by setting all enable signals to 0.

### 3.3 The stage controller

The stage controller is composed of three main parts and a stage control block as shown in figure 3.8.

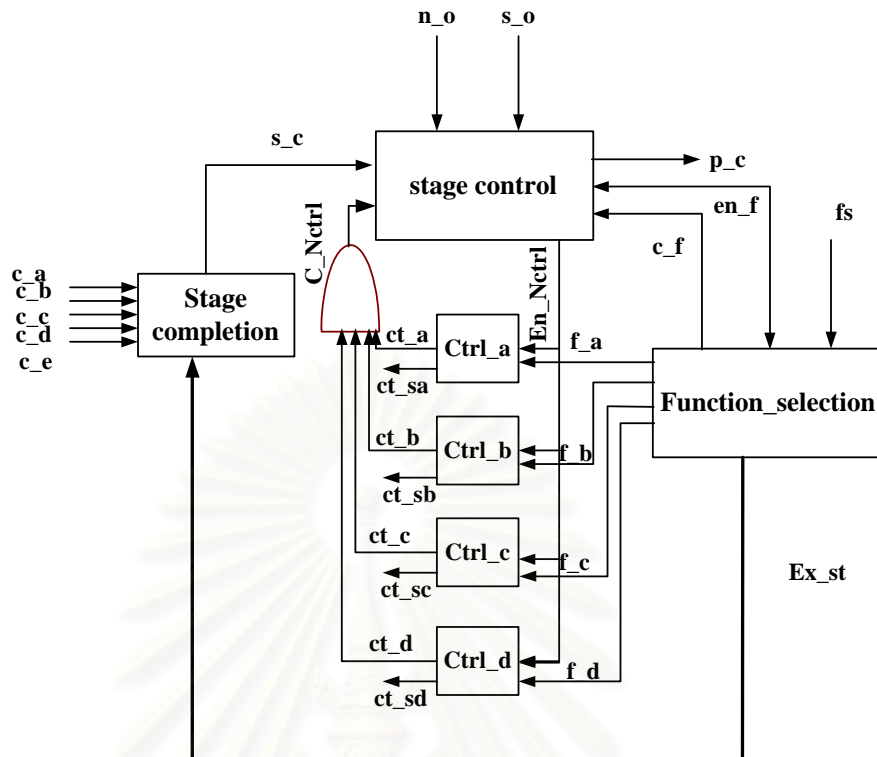


Figure 3.8: Block Diagram of stage controller

The operation of the stage controller is divided into two modes as shown in flowchart in figure 3.9

1. **n\_o is high:** The controller receives  $f_i$ , and then function selection block shifts the value of registers. Then function registers and direction registers are ored together depends on functions that are initiated. Then, ored results re or with the value in shift register and become the current functions and directions of stages. Then first rows of registers are sent to Ctrl\_block. Ctrl\_block send control signal to each pipe stage. After all execute stage (define by  $ex\_st$  signal) are completed,  $s\_c$  is high, and  $P\_c$  goes high.

2. **s\_o is high:** There is no changes in the first row of registers, and follows the steps when  $n_o$  is high.

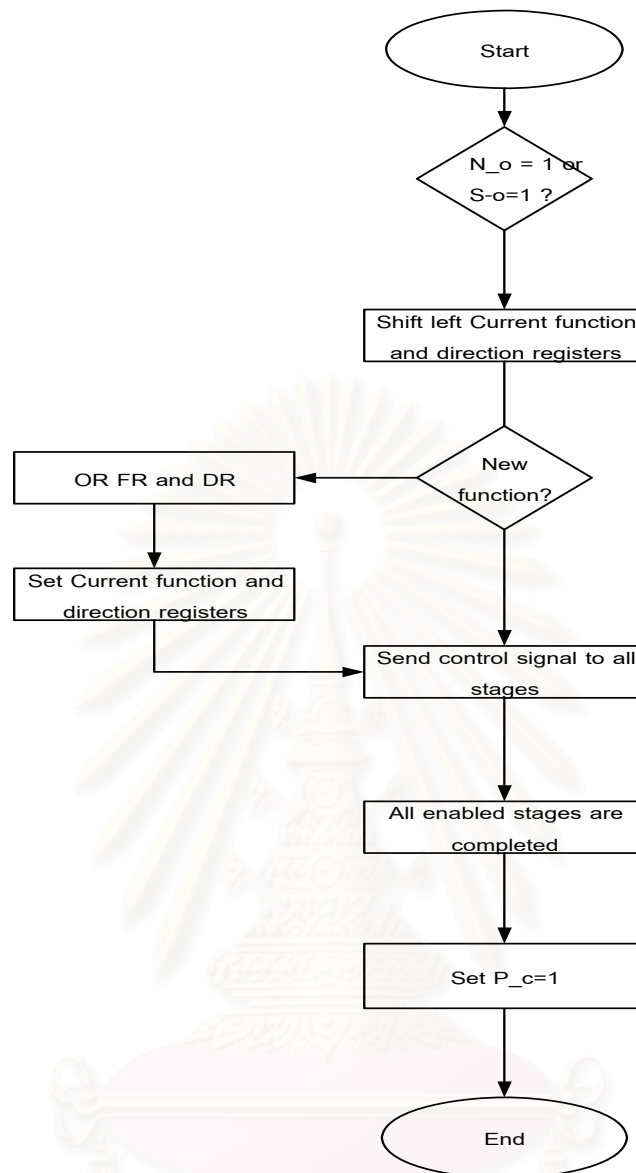


Figure 3.9: Flow chart of Stage controller

Next, we will move on to each module of stage controller.

The first part is function selection; its operation is to generate function and data transfer direction for every control\_stage block. Shift registers in function\_selection block is stored a set of functions and directions for each pipe stages in sequence as shown in figure 3.10. This shift register is shift every cycle. There are a three set of register in this block.

1. **An initial function register (FR):** It represents function that a stage executes at a unit time. Size of register,  $i \times j$ , depends on maximum evaluation time of stages and a maximum number of functions that a stage can execute. Where  $i$  = number of stages, and  $j$  = number of functions. For example, look back to figure. 2.5,  $i = 4$  and  $j=2$  because a stage can execute two functions at most. Each bit in a column of FR register represents a function: 0 is not execute, 1 is execute. For example, the following FR is FR of dynamic pipeline in figure 2.11. We get FR from the reservation tables. For pipeline in figure 2.11  $i = 5$ ,  $j = 2$ :  $j(0) = \text{function } x$   $j(1) = \text{function } y$ .

Stage A: FRax = [00 00 00 00 00]

FRay = [10 00 00 00 00]

Stage B: FRbx = [01 00 00 00 00]

FRby = [00 10 00 10 00]

Stage C: FRcx = [00 01 00 01 00]

FRcy = [00 00 10 00 00]

Stage D: FRdx = [00 00 01 00 00]

FRdy = [00 00 00 00 10]

2. **An initial direction register (DR):** It represents data transform direction that a stage executes at a unit time. It is similar to FR, but function is changed to direction. For pipeline in figure 2.11  $i = 5$ ,  $j = 2$ :  $j(0) = \text{first time direction}$   $j(1) = \text{second time direction}$ .  $J=2$  because the maximum direction of satges is 2.

Stage A: DRax = [00 00 00 00 00]

DRay = [01 00 00 00 00]

Stage B: DRbx = [01 00 00 00 00]

DRby = [00 01 00 10 00]

Stage C: DRcx = [00 01 00 10 00]

DRcy = [00 00 01 00 00]

Stage D: DRdx = [00 00 01 00 00]

DRdy = [00 00 00 00 01]

**3. Shift register:** There are two sets of shift registers: the first set represents the current function and the second set represents the current direction of stages. The scheme to generate the current state register is similar to the scheme to generate current state of pipeline scheduling scheme. The FR and DR of every stage is fetched into OR block. Stage A is an example, if function x is initiated, FR<sub>x</sub> and DR<sub>x</sub> are fetched; and then, they are ORED with values in the first set and the second set of shift registers respectively. After that, The ORed results become the current function and current direction of stage A. If function x and y are initiated, FR<sub>x</sub>, FR<sub>y</sub>, DR<sub>x</sub> and DR<sub>y</sub> are fetched; and then, FR and DR are ORED with each other before they will be or with the values in the first set and the second set of shift registers respectively.

FR and DR of each stage are combined together, and sent to each control block of stage (ctrl\_a – ctrl\_d) via f\_stage (f\_a – f\_d) signals.

For shift register, the register size of a stage is 4 x 4; 4 is the number of stages. Each row consists of four bits; the first two bit represent function, 00 = no function, 01 = function X, 10 = function Y, 11 is not used. A stage can be used more than one time for a function; for example stage B of Y is used in time 1 and time 3, and data transfer for time 1 and time 3 are different. Thus, the last two bits of row in the register is used to represent data transfer of stage; 01 for first data transfer, and 10 for later.

Function selection block get the value of function from shift register, and detect what stages will be enabled in this cycle. Then, ex\_st is sent to Stage\_complete. A number of ex\_st bits is equal the number of stage. For example, 0011 means stage A and stage B will be enabled in this cycle

The second part is Stage control block (ctrl\_a – ctrl\_d) as shown in figure 3.11. Control block function is similar to another control block. F\_a is split into two parts and sent to Function\_control and Direction\_control block. Combination circuit inside these two blocks depends on characteristics of stages. Outputs of these two blocks are control signal used to enable a stage and select a data transfer direction. After

completion of these two block, their results are combined together and become ct\_s(stage) signal.

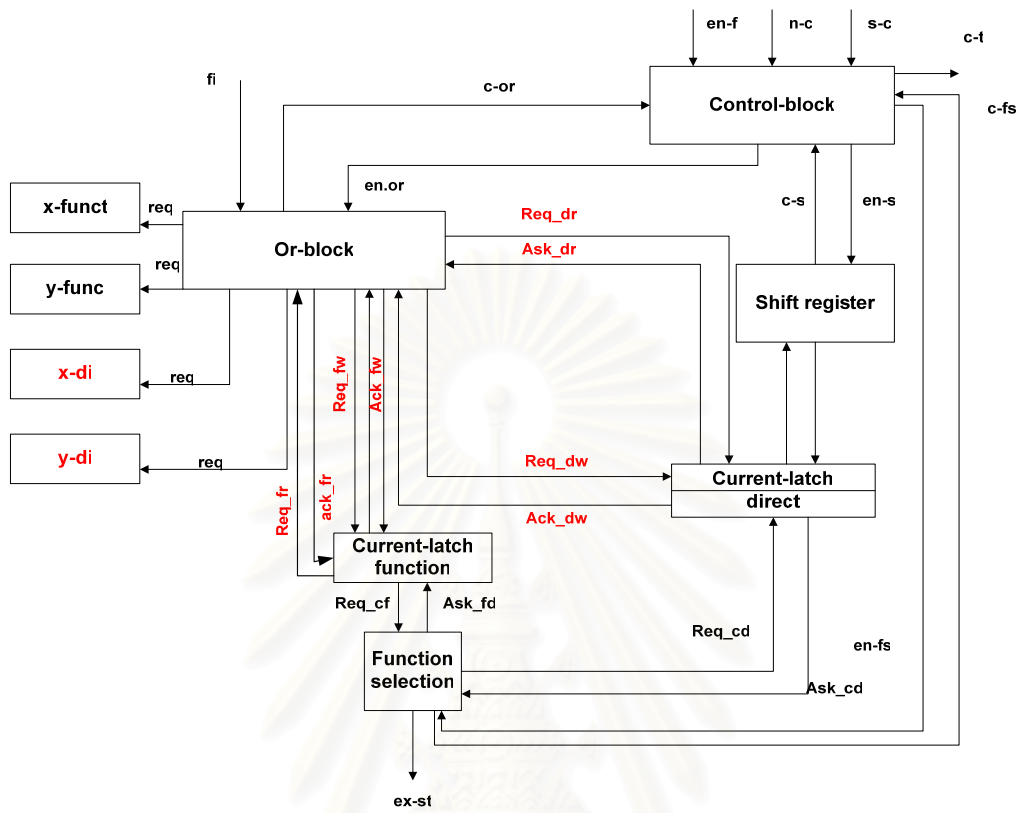


Figure 3.10: Block Diagram of function selection

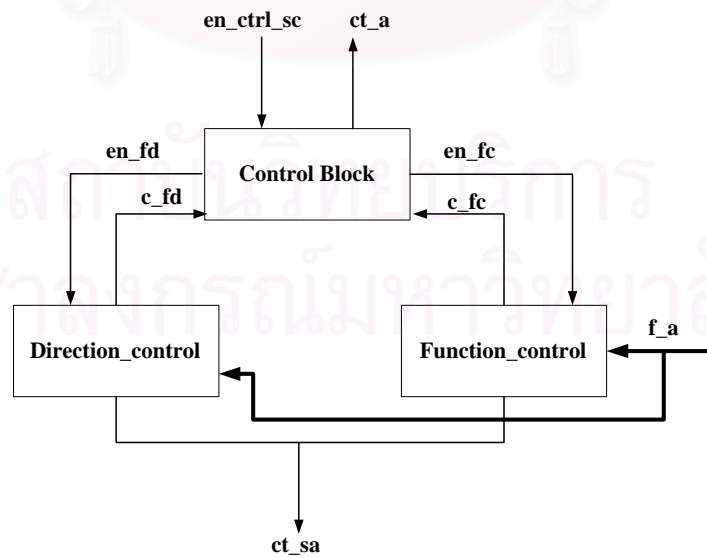


Figure 3.11: Block Diagram of completion block

The last part is Stage\_complete block as shown in figure 3.12.

The operation of the stage completion is divided into two operations. A compare block operates as a control block plus a compare block. Strobe signal is used to enable the 1-2 converter. After all enabled stages are completed, they send  $s_c(\text{stage})$  to this block. Next, compare block set  $s_c$  high to indicate that the operation of enabled stages in this cycle is completed.

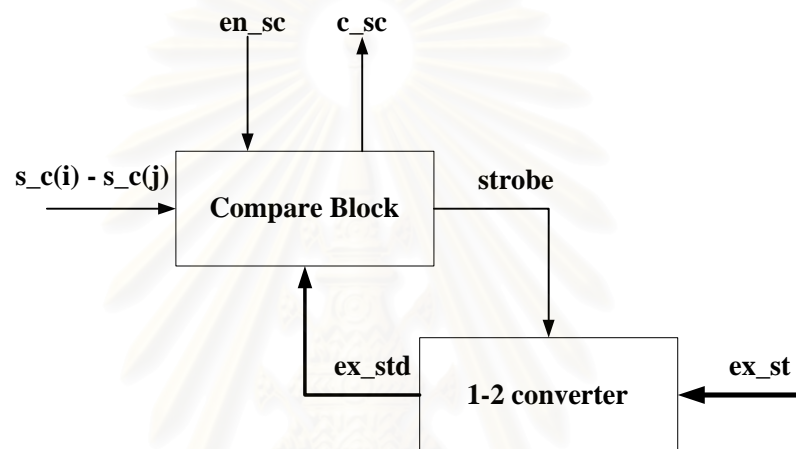


Figure 3.12: Block Diagram of Stage\_complete block

### 3.4 Circuit design and circuit synthesis

I started my design inside each main part by module design. Each part consists of 2 parts: control block and function blocks. There is only one main control block in the pipeline scheduler and the stage controller. A number of function blocks and their function depend on the operation in the pipeline scheduler and the stage controller. After I got all modules, I designed datapaths between each module. Then, submodules inside each module except control block are designed. The control block is a circuit that is synthesized from STG. Each module includes its control block and combination circuits in submodules. As a result, I got block diagram of each main part as shown in figure 3.2 and 3.7.



Next, I design STG of two main control blocks and module's control block. For each main part, STG is carefully design to avoid conflicts and confusions of control signals because an out put signal in a STG is an input signal in other STGs. Moreover, a signal might be used several times to generate other signals. When all STG in a main part is satisfy, we started circuit synthesis from STG based on Park's scheme as described in chapter 2. During synthesis, some STGs may be modified to satisfy Park's definations. An example of circuit synthesis from STG is shown in figure 3.13; it is a STG of the control block of the pipeline scheduler. Figure 3.14 is a circuit of signal en\_sh.

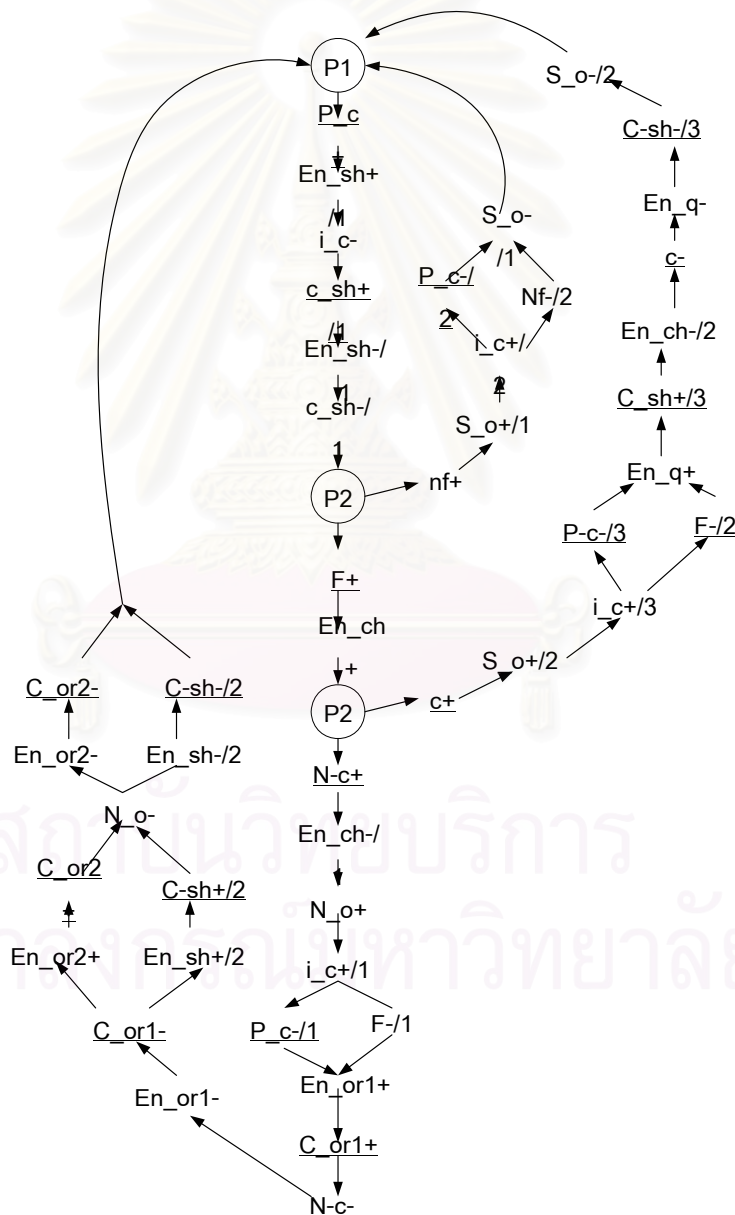


Figure 3.13: STG of control block of the pipeline scheduler

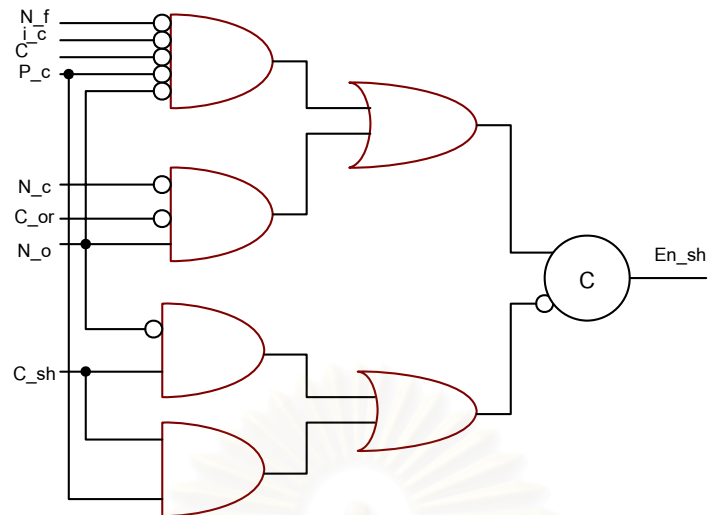


Figure 3.14: a circuit for signal *en\_sh*

Then, combinations circuits inside modules are designed. The design method based on asynchronous circuit design in chapter 2. Since data between each module is bundle data and it is single rail, 1-2 converter and 2-1 converter is necessary in some modules. Bundle data is applied to reduce data size and data wire. Furthermore, latches used in control unit are based on S-R latch.

สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

## CHAPTER IV

### ASYNCHRONOUS FLOATING-POINT ARITHMETIC UNIT

The proposed asynchronous floating-point arithmetic unit is discussed in this chapter. The arithmetic unit operates five functions: add/subtract, multiply, negate, absolute and compare. Overall of the arithmetic unit is described in section 4.1. The following section 4.2 - 4.6 discuss details of each function.

#### 4.1 Overall

This unit is based on single precision (32 bits) IEEE standard 754-1985 [22-24] including standard format and exceptions. The special bit, Guard Bit, is extended to the LSB of Mantissa in standard format to increase the precision of the arithmetic result. Then, the rounding technique is used to convert the result to the 32 bits standard format. The round to nearest is a rounding technique used in my design: under half is 0 and above half is 1. Beside, rounding to infinity and rounding to zero is included in my design.

The arithmetic unit operates four functions: add/subtract, negate, absolute and compare. The operation of each function is split into stages to work as a pipelined arithmetic unit. After consideration and comparison in terms of operation and stage, the arithmetic unit can then be divided into stages as follow:

- U (Unpack): get input: a number of input (1 for absolute and negate, 2 for Add/Subtract, Multiply ,and Compare). Next, this stage distributes input into 3 parts: Sign bit, Exponent and Mantissa.

- C (Check for zero/Change sign bit): check that input is 0 or infinity or precise exception or not, and change sign bit for negate and absolute function.

- E (Exponent Adder/Subtractor): add and subtract exponents of the input.

- A (Adder): add and subtract mantissa of the input. The complement circuit is in this stage.

- M (Multiplier): multiply mantissa of the input.
- Ch (Check for Exception): check whether the result is precise exception or not.
- N (Normalize): convert the result into formatted number used in circuit, not a standard format.
- R (Rounding): round the result and convert the result to 32 bits IEEE 754 standard format.

All functions use redundancy stages as shown in Table 1. The usage and data transfer stages in the dynamic pipeline used for this arithmetic unit is shown in Figure 4.1.

Table 4.1: Arithmetic function and stage usage

Arithmetic Function	Stage Usage
1. Add/Subtract	U, C, E, A, Ch, N, Ch,R
2. Multiply	U, C, E, M, Ch, N, Ch,R
3. Negate	U, C
4. Absolute	U, C
5. Compare	U, C, E, A, R

Stages Ch and N are considered to be a loop because some results need to be checked more than once and to format them to the standard format. In figure 4.1, we need to reserve Ch twice to avoid stage collision because we can not know that whether Ch will be used once or twice when inputs come into the arithmetic unit. It is time consumption that happens when the result uses Ch only once, but it is still better than stage collision.

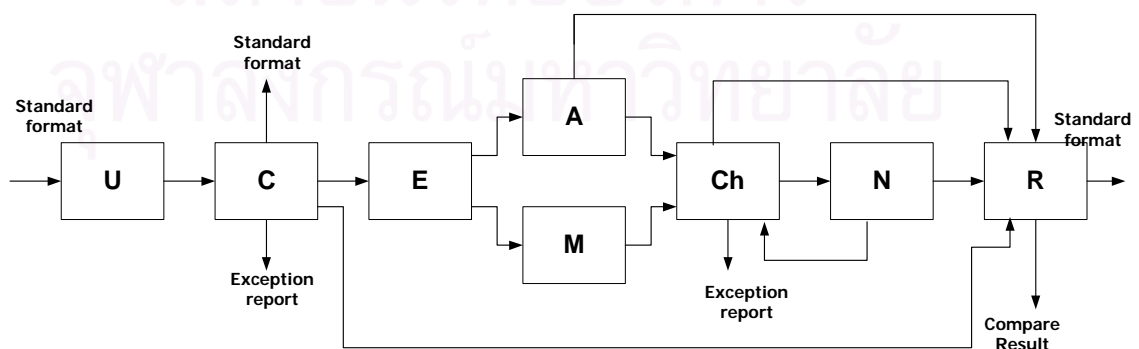


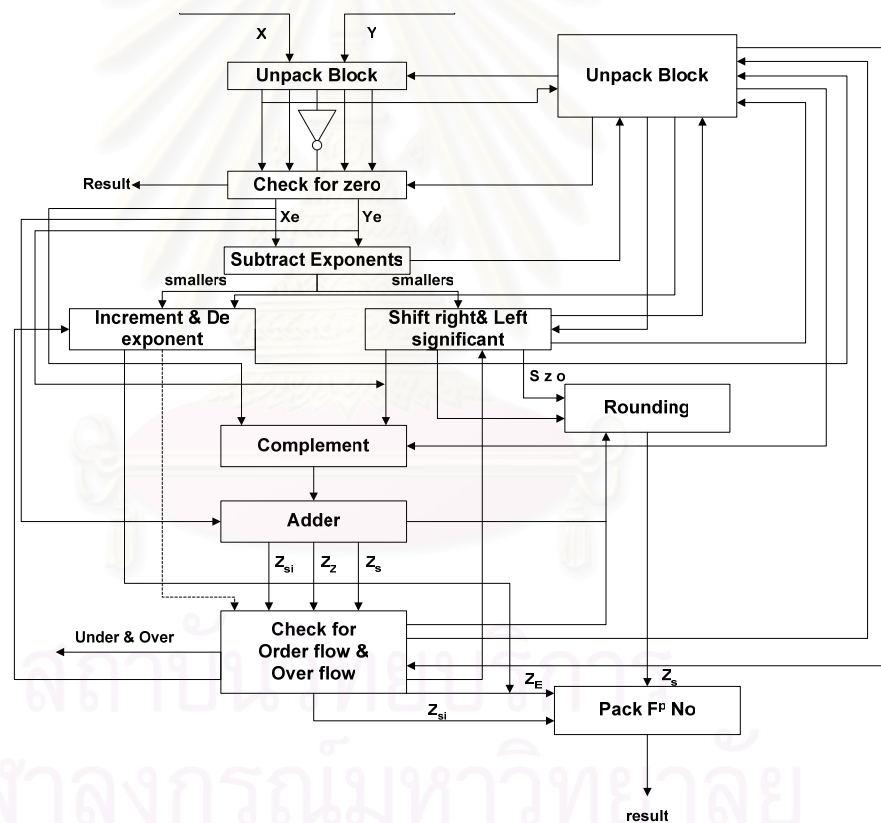
Figure 4.1: Stage and Data transfer between stages

## 4.2 Add/Subtract

There are five steps of Add/Subtract function after unpack as following.

1. Check for zero.
2. Align the significands.
  - Compare components
  - Shift right significand (smaller exponent)
3. Add or subtract the significand.
4. Normalize the result.
5. Round the result.

A block diagram of Add/subtract function is shown in figure 4.2.



Figur. 4.2: Block diagram of Add/Subtract function

## 4.3 Multiply

There are five steps of Multiply function after unpack as following.

1. Check for zero.

2. Add exponents
3. Subtract bias
4. Multiply the significand.
  - Check Q,Q-1
  - Add, subtract significands and count
  - Shift right A, Q,Q-1
5. Normalize the result.
6. Round the result

A block diagram of Add/subtract function is shown in figure 4.3.

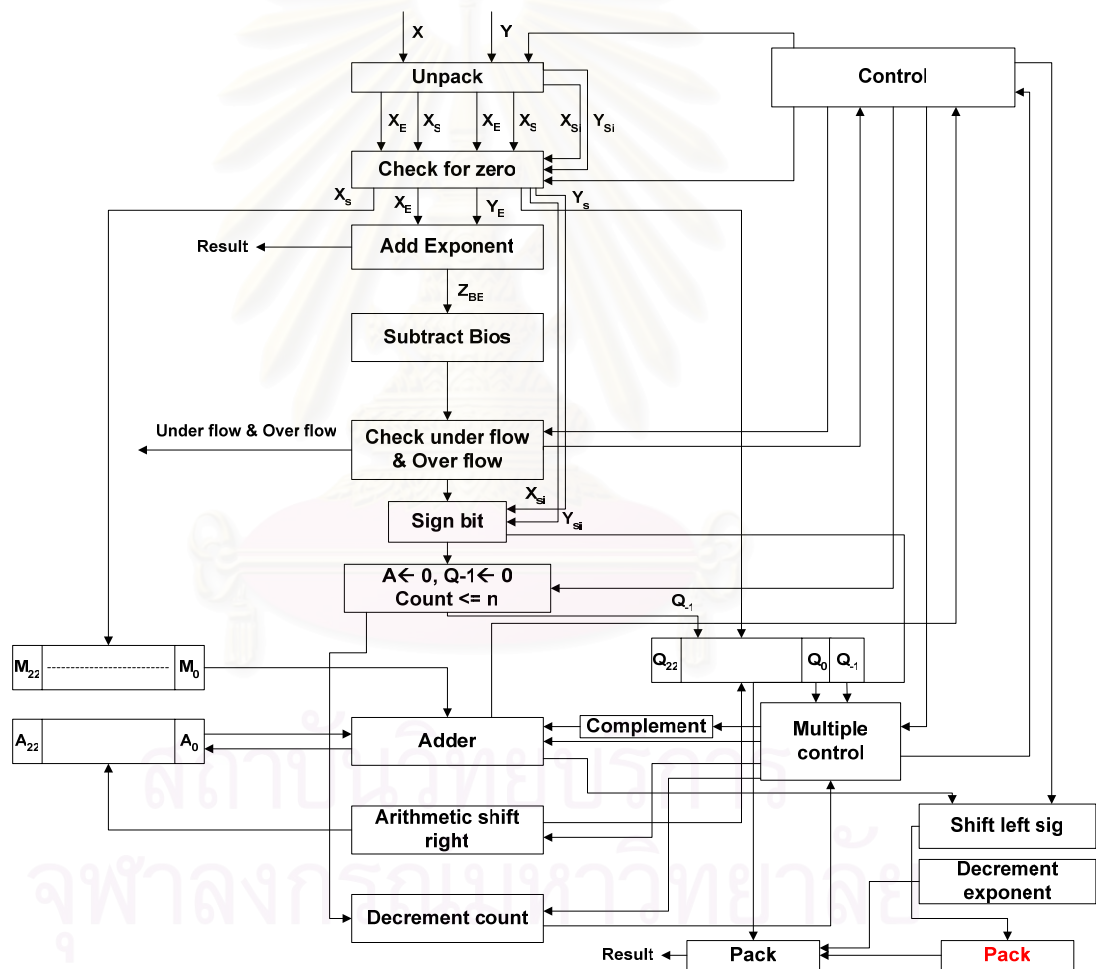


Figure 4.3: Block diagram of Add/Subtract function

#### 4.4 Negate

There is only one step after unpack: it is checking for exception and changing the sign bit.

#### 4.5 Absolute

There is only one step after unpack: it is checking for exception and assign 0 to the sign bit.

#### 4.6 Compare

There are five steps of Compare function after unpack as following.

1. Check for zero.
2. Compare sign bit.
3. Compare exponent.
4. Subtract significands.
5. Round the result
6. Return compare result.

A block diagram of Add/subtract function is shown in figure 4.4.

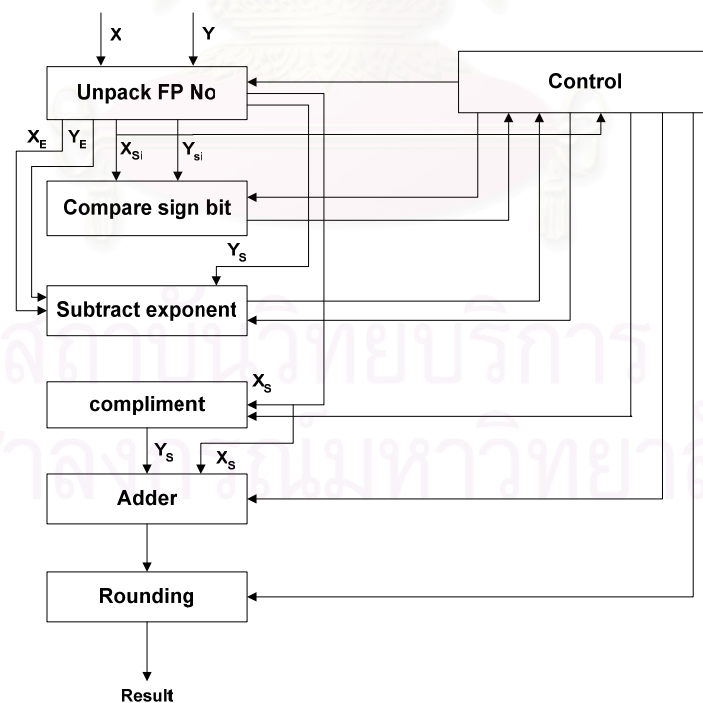


Figure 4.4: Block diagram of Compare function

## CHAPTER V

### CONTROL UNIT IMPLEMENTATION

This thesis presents the controller that can be used as a template to design the controller of dynamic asynchronous pipelines. Steps of implementation for any dynamic asynchronous pipeline are the same and described in section 5.1-5.5. However, the modification is necessary because of the characteristics of each pipeline; the number of stage, the number of function, and function configuration, but the main parts of their controllers are the same. An implementation for a dynamic asynchronous pipeline and its controller consists of five steps as presented in figure 5.1. In chapter 3, the control unit for a four stage dynamic pipeline is discussed. In this chapter, how to apply it to other dynamic pipeline is discussed. Each section gives an example: the proposed floating point arithmetic unit.

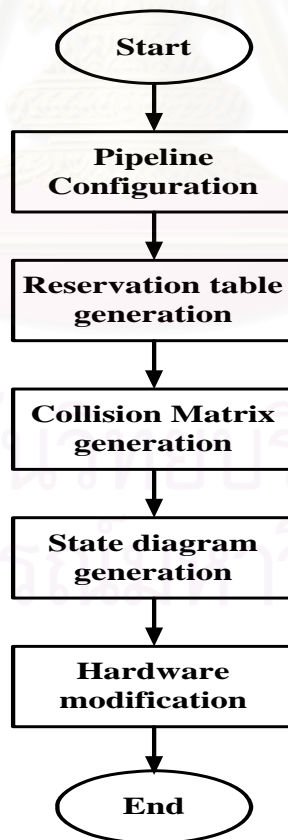


Figure 5.1: Steps of Control unit implementation





Table 5.3: the reservation table for function Negate (N)

stage /time	0	1
U	X	
C		X
E		
A		
M		
Ch		
N		
R		

Table 5.4: the reservation table for function Absolute (A)

stage /time	0	1
U	X	
C		X
E		
A		
M		
Ch		
N		
R		

Table 5.5: the reservation table for function Compare (C)

stage /time	0	1	2	3	4
U	X				
C		X			
E			X		
A				X	
M					
Ch					
N					
R					X

### 5.3 Collision Matrix generation

The collision matrix generated from this step will be store in registers of initial collision matrix block and is used to generate the current state of pipeline. A set of collision matrixes of the floating-point arithmetic unit is below.

$$CM_{ad} = [10100000 \ 10100000 \ 10000000 \ 10000000 \ 10010000]$$

$$CM_m = [10100000 \ 10100000 \ 10000000 \ 10000000 \ 10010000]$$

$$CM_n = [10000000 \ 10000000 \ 10000000 \ 10000000 \ 10000000]$$

$$CM_{ab} = [10000000 \ 10000000 \ 10000000 \ 10000000 \ 10000000]$$

$$CM_c = [10000000 \ 10000000 \ 10000000 \ 10000000 \ 10000000]$$

### 5.4 State Diagram generation

The state diagram is used to check correctness of the controller by checking the current state of pipeline generated from state generator block. The state diagram is not included in the controller circuit. Figure 5.2 is a part of state diagram of the floating-point arithmetic unit.

### 5.5 Hardware modification

Because of the configurations of the controllers for each pipeline are little different. Note that, scheduler control and stage control module will not be modified because they are the control circuits that can be used by any dynamic pipeline. Similarly, control block of every module of both pipeline scheduler and stage controller will not be modified. Combination circuits of modules must be modified to support each different dynamic pipeline.

For pipeline scheduler, the number and size of registers in initial collision matrix, state generator, shifter & current state module depends on the number of functions and pipeline stages. Furthermore, combination circuit in collision check module must be modified too, and can be improved to support the requirement of designers.

For stage controller, function selection, and ctrl\_block must be modified to support the pipeline configuration. The number and size of registers (FR, DR, and shift register) in function\_selection modules depends on the number of functions and pipeline stages. Similarly, the number of ctrl\_block is equal the number of pipeline stage that is

8. An example of FR and DR is below. A size of column in FR and DR is 3 because there are at most 3 functions that can be executed in a stage: Ch stage 001 = absolute, 010 = nagate, and 100 = another function. In addition, there are 3 data directons in a circuit at most: Ch stage and R stage.

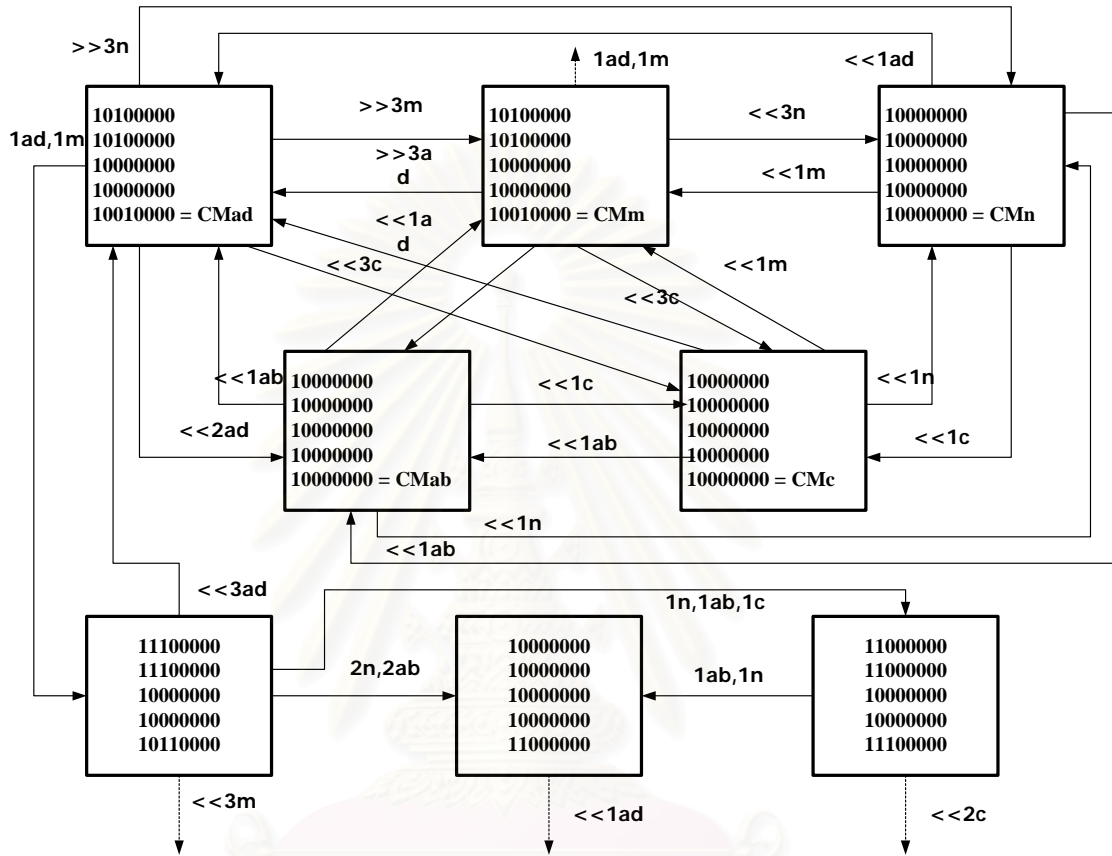


Figure 5.2: A part of state diagram of the floating-point arithmetic unit

Stage U: FRuad = [001 000 000 000 000 000 000 000]

Frum = [001 000 000 000 000 000 000 000]

FRuab = [001 000 000 000 000 000 000 000]

FRuab = [001 000 000 000 000 000 000 000]

FRuac = [001 000 000 000 000 000 000 000]

DRuad = [001 000 000 000 000 000 000 000]

DRum = [001 000 000 000 000 000 000 000]

DRuab = [001 000 000 000 000 000 000 000]

$$DRuab = [001\ 000\ 000\ 000\ 000\ 000\ 000\ 000]$$

$$DRuac = [001\ 000\ 000\ 000\ 000\ 000\ 000\ 000]$$

$$\text{Stage Ch : FRchad} = [000\ 000\ 000\ 000\ 001\ 000\ 001\ 000]$$

$$FRchm = [000\ 000\ 000\ 000\ 001\ 000\ 001\ 000]$$

$$FRchab = [000\ 000\ 000\ 000\ 000\ 000\ 000\ 000]$$

$$FRchab = [000\ 000\ 000\ 000\ 000\ 000\ 000\ 000]$$

$$FRchac = [000\ 000\ 000\ 000\ 000\ 000\ 000\ 000]$$

$$DRchad = [000\ 000\ 000\ 000\ 001\ 000\ 100\ 000]$$

$$DRchm = [000\ 000\ 000\ 000\ 010\ 000\ 100\ 000]$$

$$DRchab = [000\ 000\ 000\ 000\ 000\ 000\ 000\ 000]$$

$$DRchab = [000\ 000\ 000\ 000\ 000\ 000\ 000\ 000]$$

$$DRchac = [000\ 000\ 000\ 000\ 000\ 000\ 000\ 000]$$

Step 1 depends on the requirement of designers. Step 2 – Step 4 based on pipeline scheduling described in section 3. Step 5 is based on ideas presented in this thesis.

สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

## CHAPTER VI

### EXPERIMENTS AND RESULTS

This chapter is divided into 2 sections: Experiments, section 6.1, and experimental results. Moreover, the experimental results is composed of 2 sections: control unit result and asynchronous floating-point arithmetic unit. Section 6.2 shows comparison between results of two control units: figure2.11 and the floating-point arithmetic unit.

#### 6.1 Experiments

According to timing assumption used to design circuits in this thesis, the real circuits are not synthesized because the synthesis tools are not fully support asynchronous circuit design. Thus, simulation results were used to test and verify circuits. Experiments consist of 3 steps as follow.

##### 6.1.1 Adding delays

Delays are added to circuits to make circuits to be asynchrony. Delays in a gate is assumed to be 1. The following code demonstrates adding delays to circuits.

```
cs_ic  <= n_o or s_o after 1 ns;  
cr_ic  <= not (en_sh and p_c) after 2 ns;  
i_c    <= (cs_ic and cr_ic) or (cs_ic and i_c)  
        or (cr_ic and i_c) after 2 ns;
```

##### 6.1.2 Writing Testbench

A testbench is used to create an input sequence to a design, then observe the response. The following code is a fragment of testbench for control unit: it is initialize value before the circuits is started.

```

f   <= '0';
nf  <= '0';
fi  <= "00";

data_x <= ("10100", "01000");
data_y <= ("01110", "10100");

```

### 6.1.3 Simulation

The simulation results are generated by Model-SIM. First, we change the directory to the directory that program is. In figure 6.1, the directory to the directory in shaded box on screen.

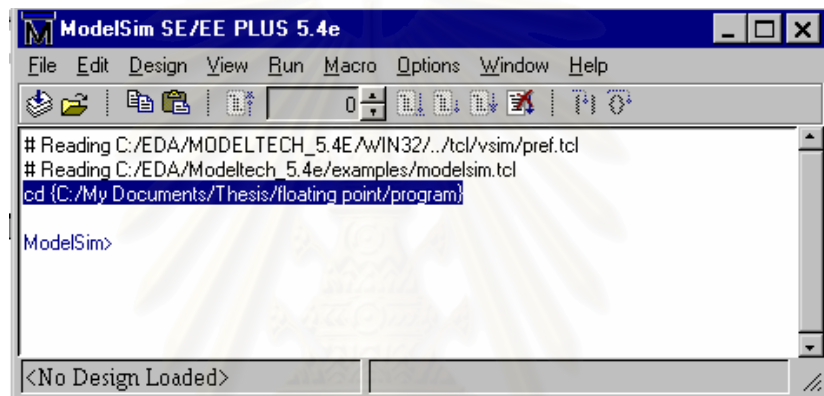


Figure 6.1 Changing Directory

Then, we load design that is an entity we want to test in our design as figure 6.2.

Next, we select view from menu bar, and select a thing we want to observe as shown in figure 6.3.

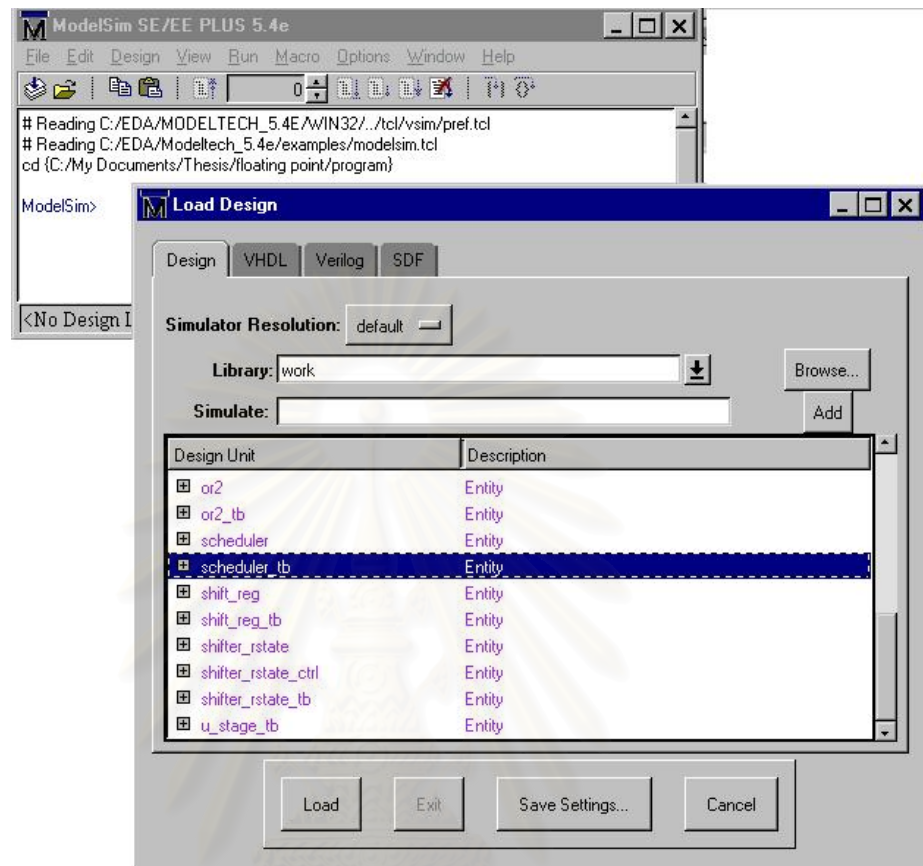


Figure 6.2 Loading design

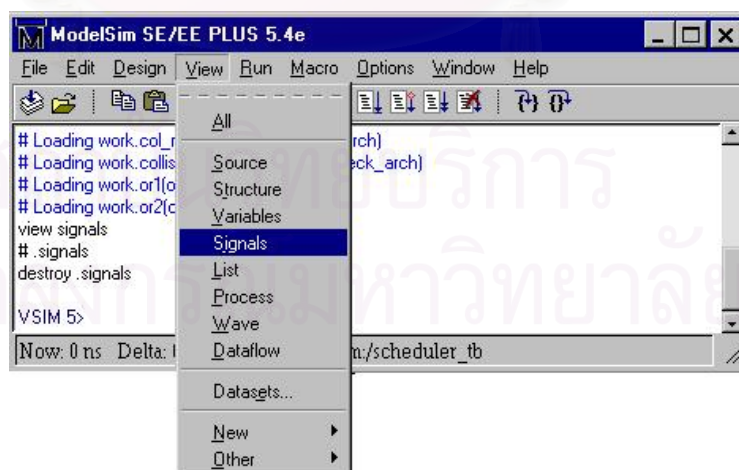


Figure 6.3 Viewing responses



## 6.2 control unit results

The circuit is simulated by Model Sim as shown in figure.6.4. P\_c to re\_st are signal of pipeline scheduler, and the rest are stage controller. Let consider the left hand side of figure 10, or\_rs and or2\_rs is the result of initial collision matrix and state generator block. Re\_st is the value of recent state register which is a shift register.  $Re\_st = [11000, 10000]$ . fi represents the function initiation ; it consists of two bits ,and each bit represent each initiation of function (Y,X). In the figure 10 , fi = "01"; X function are initiated. When p\_c goes high, shift register must be shifted left. And then, the collision check block is enable by setting en\_ch.

The first bit of the first column is 1 , so function X can not be executed without collision. Check collision block checked set c signal high to tell the control block that B function can not be executed at this moment. S\_o (same operation) goes high ; then the scheduler control block end the operation. For stage controller block, the operation begin after receiving s\_o. Function X can not be executed at that time and the old function (Y) is operated from time 2 to time 3 (reservation table in table 2.2). The value in registers become  $[000, 101, 000, 000]$  ; its mean only stage B are enable. Then en\_nctrl is fired ,and all ctrl\_i are enable. After ctrl\_a to ctrl\_d operation are completed c\_nctrl is fired. Then, stage B is completed and set s\_c high. Finally, stage control block end the operation of stage controller and set p\_c high.

Next, Function Y is initiated because fi is "11". After collision check, function X can be executed , n\_c is high and every block in pipeline scheduler are enabled in order. In addition ,n\_o is high to indicate that stage controller block must execute the new function (X). The operation are similar to the previous cycle except new function ,A and B, are executed [Look at time 0 table 2.2].At this cycle ,  $re\_st = [11100, 11000]$  , and registers value in stage control is  $[000, 010, 000, 100]$

For the next cycle ,right hand side of the picture, function X is initiated. Let consider re\_st, it indicates that function x can not be executed at this cycle. The operation are the same of the two previous cycle. Function X must wait until the first bit of re\_st (current state)is 0.



Figure 6.4: Simulation result of asynchronous dynamic pipeline control unit

Table 6.1 show a number of gates used in control unit for both dynamic pipeline in figure 2.11 that is a 2-function pipeline and dynamic pipelined floating-point arithmetic circuits that is a 5-function pipeline.

Table 6.1 number of gates in control unit

Block	2-function pipeline	5-function pipeline
Pipeline Scheduler	1846	5273
Stage Controller	5499	33787
Total	7345	39060

### 6.3 Floating-point arithmetic unit results

The simulation shown in figure.6.5 presents operations of two functions: Adder/Subtractor ( $0.5_{ten} + 0.75_{ten}$ ) and compare in the last four stages. p\_c is the signal

that starts the cycle of operations. It is high when every enabled stages completes their operation. `std_result` is the result of adder/subtrator function (fourth cycle). `Cmp_result` is the result of compare function (third cycle). `Out_type` is used to identify the result: 0 is adder/subtrator, 1 is compare.

For adder/subtractor, Ch stage is used twice (first and third bubble) because the output from A stage is not normalized. The third cycle in figure.6.2 shows that two functions are operated at the same time and it is an advantage of dynamic pipeline. Moreover, negate function can be initiated (fi in a rectangle) in this cycle because it will not cause the stage collision (consider from stage diagram).

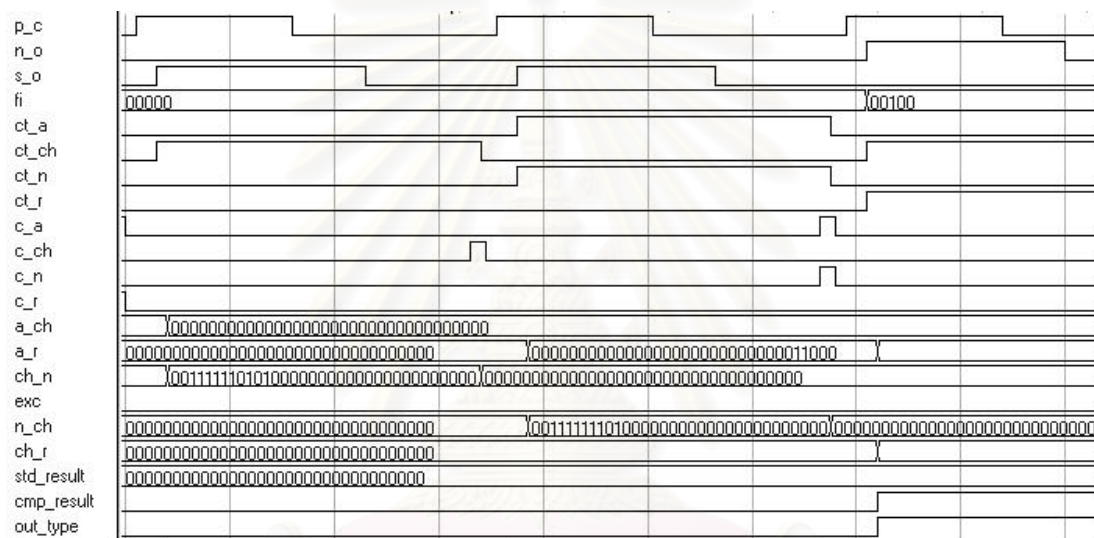


Figure 6.5: Simulation result of asynchronous floating-point arithmetic unit

Table 6.2 show average time spent for 1 cycle of each stage. Table 6.3 shows number of gates used in overall circuits.

Table 6.2 average time for 1 cycle

Stage	Time (ns)
U	157
C	204
E	187
A	238
M	451
Ch	197
N	161
R	181

Table 6.3 number of gates in overall circuits

Block	No. of gate	% of gate
Pipeline Scheduler	5273	1.93
Stage Controller	33787	12.35
Floating-point	234559	85.72
<b>Total</b>	<b>273619</b>	<b>100</b>

สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

## Chapter VII

### CONCLUSION

Clock skew problem in synchronous circuits can be avoided by an asynchronous circuit design. To increase speed and performance of asynchronous circuits, pipelining technique is adopted. In addition, this research focused on dynamic pipeline to expand the study area of asynchronous pipeline and apply it to multifunction circuits. The simulation result in section 10 showed that the circuit can work properly without stage collision.

This circuit can be applied by any multifunction circuits using dynamic pipeline with a little modification as described in design methodology (Chapter 5). The first thing is changing the initial matrix in OR1 block. Second, change a little operation in every block ,except control block, to support your pipeline. Notice, no need to change anything in control part of every block.

The challenge is to modify the controller to be a purely asynchronous circuit. There is not global clock in circuit, but stages must wait until the other stage are completed. To speed up pipeline this problem should be eliminated.

According to the number of gate in table 6.1, it shows that a number of gate in a control unit is increased when the complexity and number of functions and number of stage increase.

In this thesis, I have presented the asynchronous floating-point arithmetic unit using the dynamic pipelining technique. It can work properly without stage collision and cover the IEEE 754 standard. Dynamic pipelining reduces size and increases the speed of the arithmetic unit. This is the main advantage of my design. Furthermore, this arithmetic unit can be applied to other asynchronous processors or systems because I

use the four-phase micropipeline as the backbone of the arithmetic unit. This circuit applies the controller I designed for pipeline scheduling and stage control to maintain performance and stage collision avoidance.

The controller must know the data flow of functions before their inputs come to the pipeline. Thus, the configuration must be fixed; it is satisfactory for every function except the add/subtract function and multiplier. The stage usage of the add/subtract function and multiplier varies because it might be used more than once to check and normalize results. It cannot be predicted, so we must reserve stages for recursion in order to avoid stage collision, which causes time consumption. This point is a future challenge because the performance will be better if the time consumption can be eliminated.

Table 6.2 shows average time for a cycle of each stage, and it present that stage M is the slowest stage. Thus, stage M limit speed of throughput when Multiply function is initiated. A solution for this problem is splitting it into 2 circuits.

Table 6.3 presents that control unit is a small part when compared with floating-point arithmetic circuits. It does not increase too much complexity for circuits.

## References

1. J.L.Hennessy and D. A. Patterson, "Computer architecture a quantitative approach",Morgan Kaufman publishers,INC ,Sanfrancisco,California,1996.
2. J. Heinrich "MIPS R4000 Microprocessor User's Manual: Second Edition" [Online]. Available from <http://www.cag.lcs.mit.edu/raw/documents /R4400 Uman book Ed2.pdf> [2005,January 31]
3. Capt Kenneth R. Gilliam, "Design and Architecture for a Multi-Mode Pipelined, Floating- point Adder" ,Proceedings of the IEEE 1991 National on Aerospace and Electronics Conference, 20-24 May 1991 , Page(s): 73 –76
4. Enriquez, A.B.; Jones, K.R.; "Design and Architecture for a Multi-Mode Pipelined, Floating- point Multiplier" ,Proceedings of the IEEE 1991 National on Aerospace and Electronics Conference, 20-24 May 1991, Page(s): 77-81
5. D.W. Matula and A.M. Neilsen , "Pipelined packet-Forwarding Floating Point : I. Foundation and a rounder", 13th IEEE Symposium on Computer Arithmetic, 6-9 July 1997 , Page(s): 140 –147
6. D.W. Matula and A.M. Neilsen , "Pipelined packet-Forwarding Floating Point : II. An Adder",13th IEEE Symposium on Computer Arithmetic, 6-9 July 1997 , Page(s): 148 –155
7. AMPHION, "Floating Point Operator, Data Sheet" [Online]. Available from <http://www.quicklogic.com /images/amphion float pt op 1.pdf> [2005, January, 31]
8. P.M. Kogge, "The Architecture of Pipelined Computer". New York, New York Hemisphere Publishing Company, 1981
9. K. Hwang, "Advanced Computer Architecture: Parallesim, Scability, Programmability" New York, New York: Mcgraw-Hill,Inc.,1993

10. E. Kasten, "Design and Implementation of a Pipelined Virtual Computer", Technical Report, 1994
11. I.E. Sutherland, "Micropipelines", Communications of the ACM, vol.32,#6,pp: 720-738,June 1989
12. S.B. Furber "The return of Asynchronous Logic" [Online]. Available from [http://www.cs.man.ac.uk/async/background/return\\_async.html](http://www.cs.man.ac.uk/async/background/return_async.html) [2003,November 5]
13. Paul Day and J. Viv Woods. "Investigation into micropipeline latch style" . IEEE Transaction on VLSI systems, 3(2), June 1995.
14. S.Furber and P.Day, "Four-phasemicropipeline latch control circuits," IEEE Transactions on VLSI Systems , vol. 4, pp. 247-253,June 1996
15. K.Y. Yun, P.A. Beerel, and J. Arceo. "High-Performance Asynchronous Pipeline Circuits". Proceeding of the 1996 Symposium on Advanced Research in Asynchronous Circuits and Systems, pp: 17-28, March 1996.
16. Nanya, T.;Ueno Y.; Kagotoni, H.;Kuwako, H.;Takamura, A. TITAC: Design of a Quasi-Delay-Insensitive Microprocessor ,IEEE Design & Test of Computer Vol.11, No. 2, (1994):51-63
17. T.A.Chu, "Synthesis of Self-timed VLSI circuits from Graph-theoretic Specifications", Phd. Thesis,Messachusetts Institute of Technology, 1987
18. S.B. Park, "Synthesis of Asynchronous VLSI circuits from Signal Transition Graph Specifications" , Phd. Thesis, Tokyo Institute of Technology, 1996
19. Hauck,S. Asynchronous Design Methodologies: An Overview IEEE Transaction on Computer 83,1 (January 1995):69-93
20. David Glodberg, "What Every Computer Scientist Should Know About Floating-Point Arithmetic", Computer survey, March 1991



21. W.Stalling, "Computer Organization & Architecture ,Designing for Performance : Sixth Edition", New Jersey, Prentice Hall, 2003
22. D.A. Patterson and J.L. Hennessy, "Computer Organization & Design , Hardware/Software Interface : Second Edition", San Francisco, Morgan Kaufman Publishers, Inc, 1997



สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย



Appendices

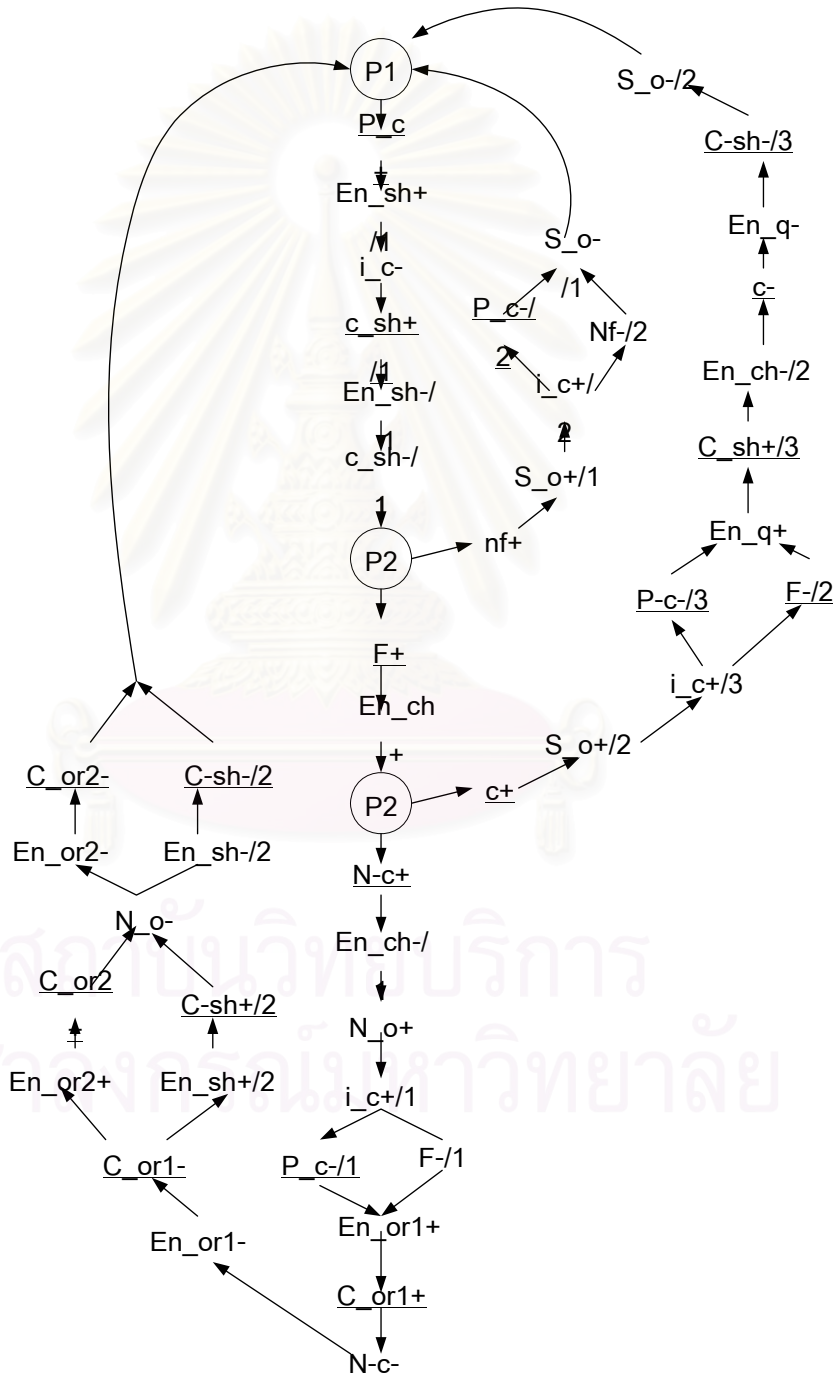
สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

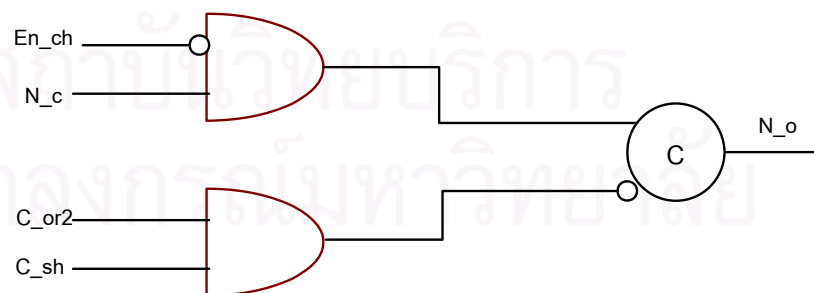
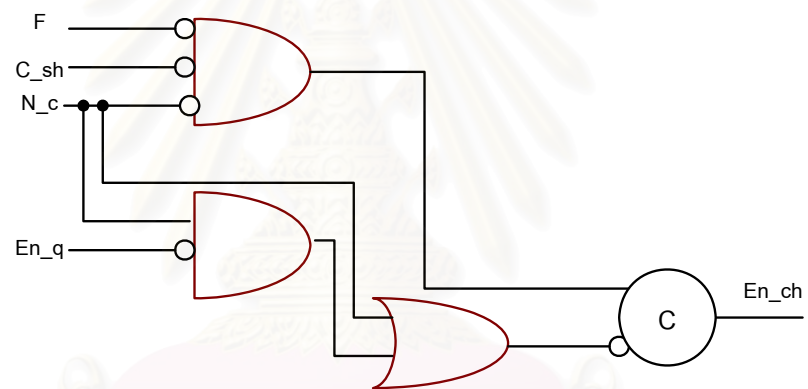
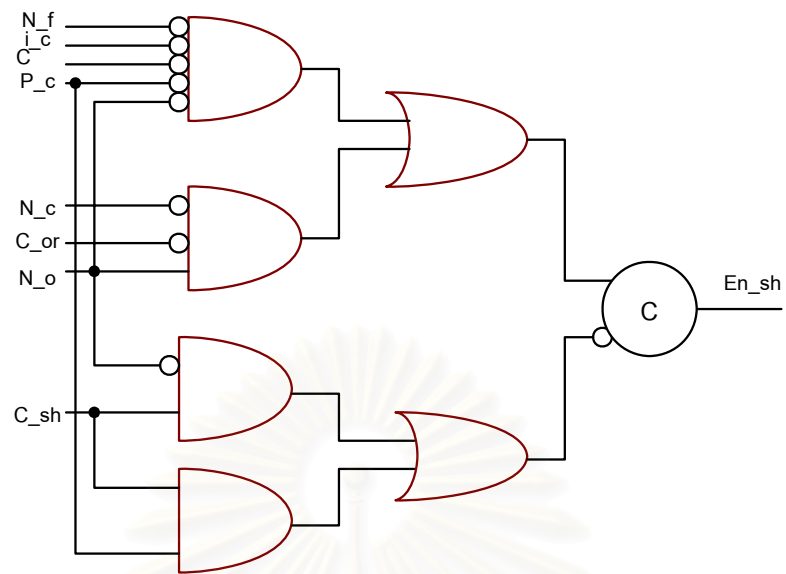
Appendix A

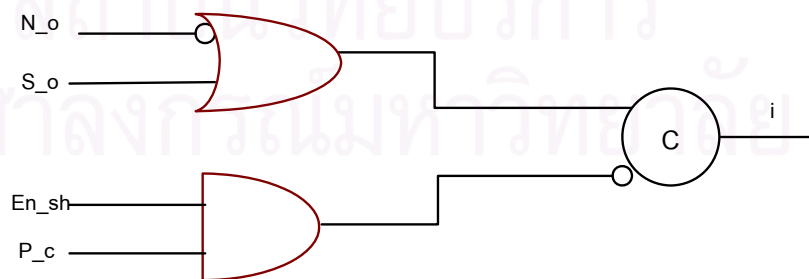
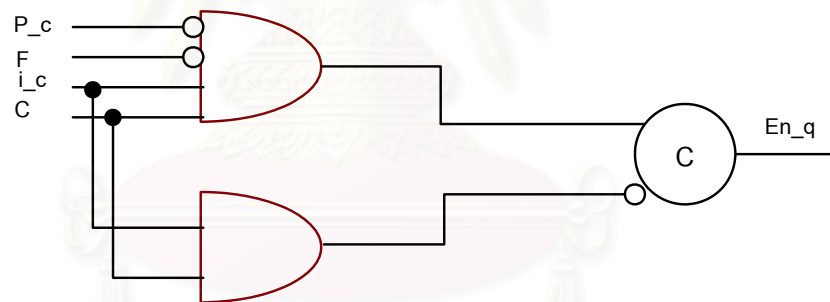
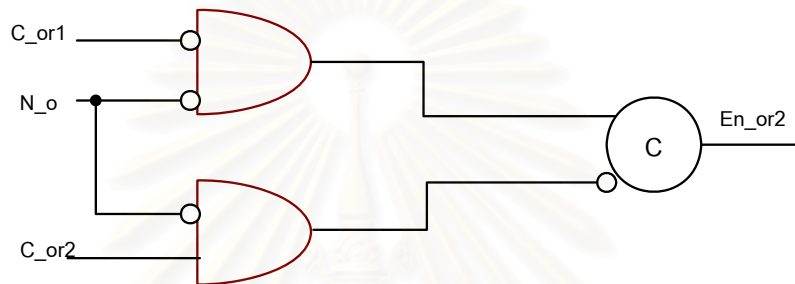
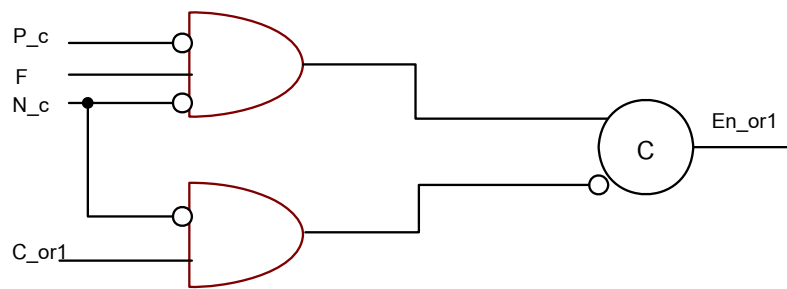
STG and Circuits

1 Pipeline Scheduler

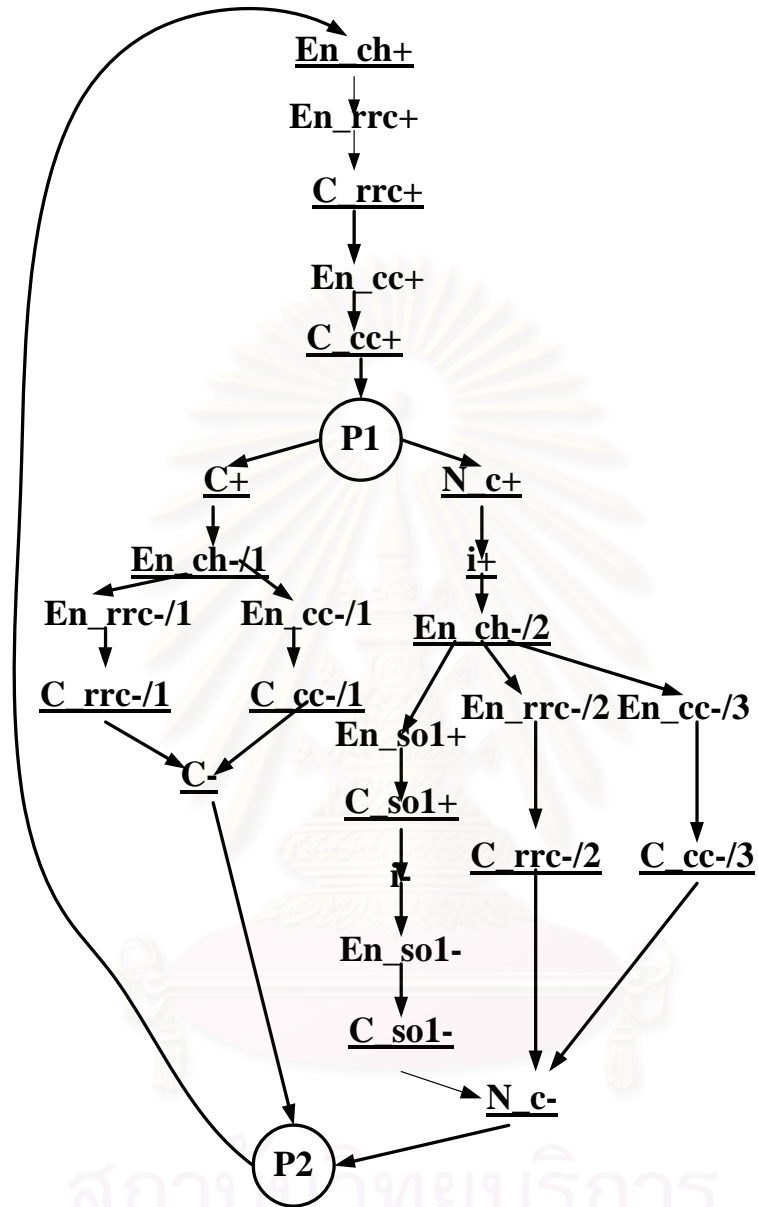
1.1 Control Block of Pipeline Scheduler

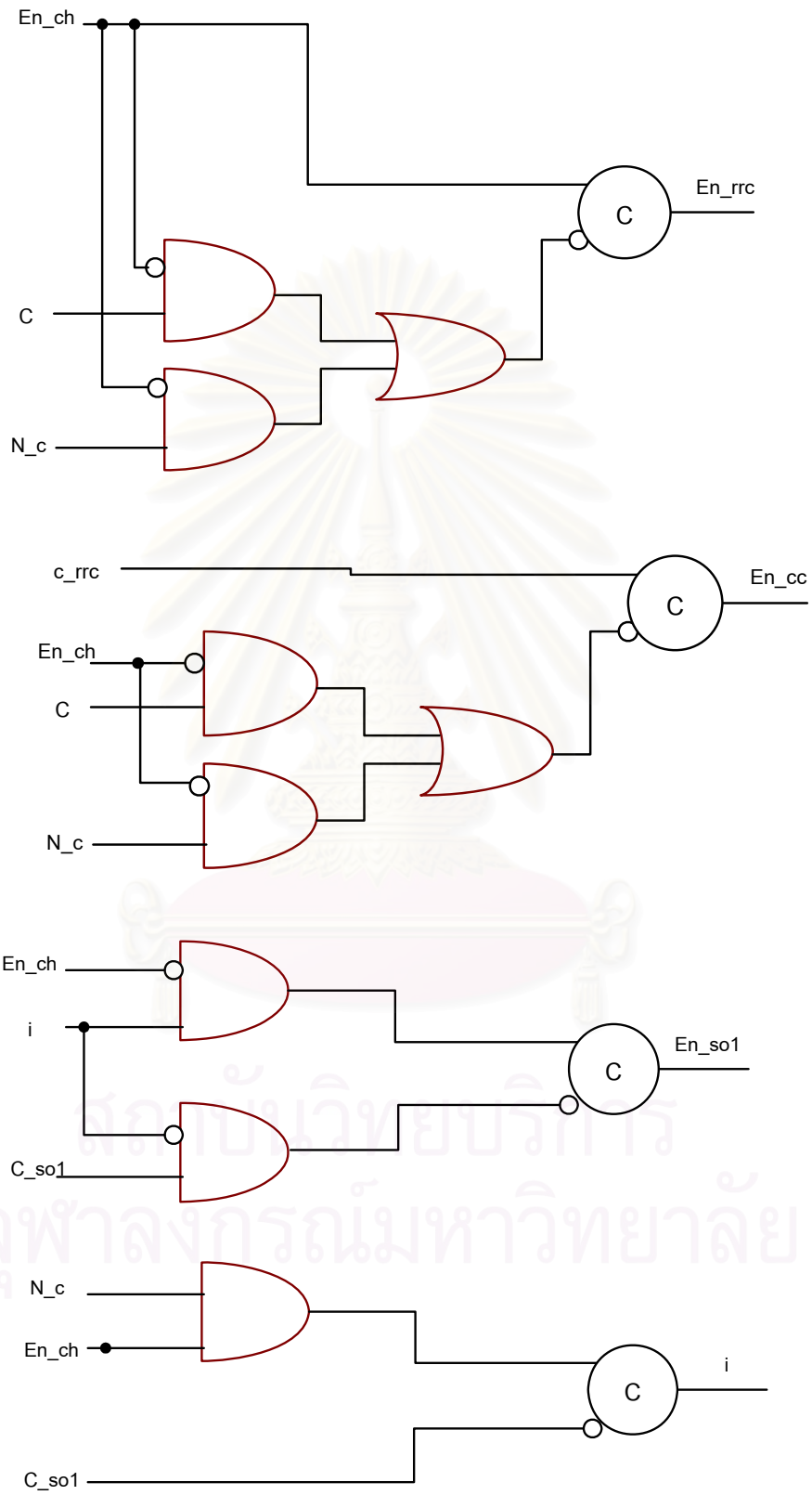




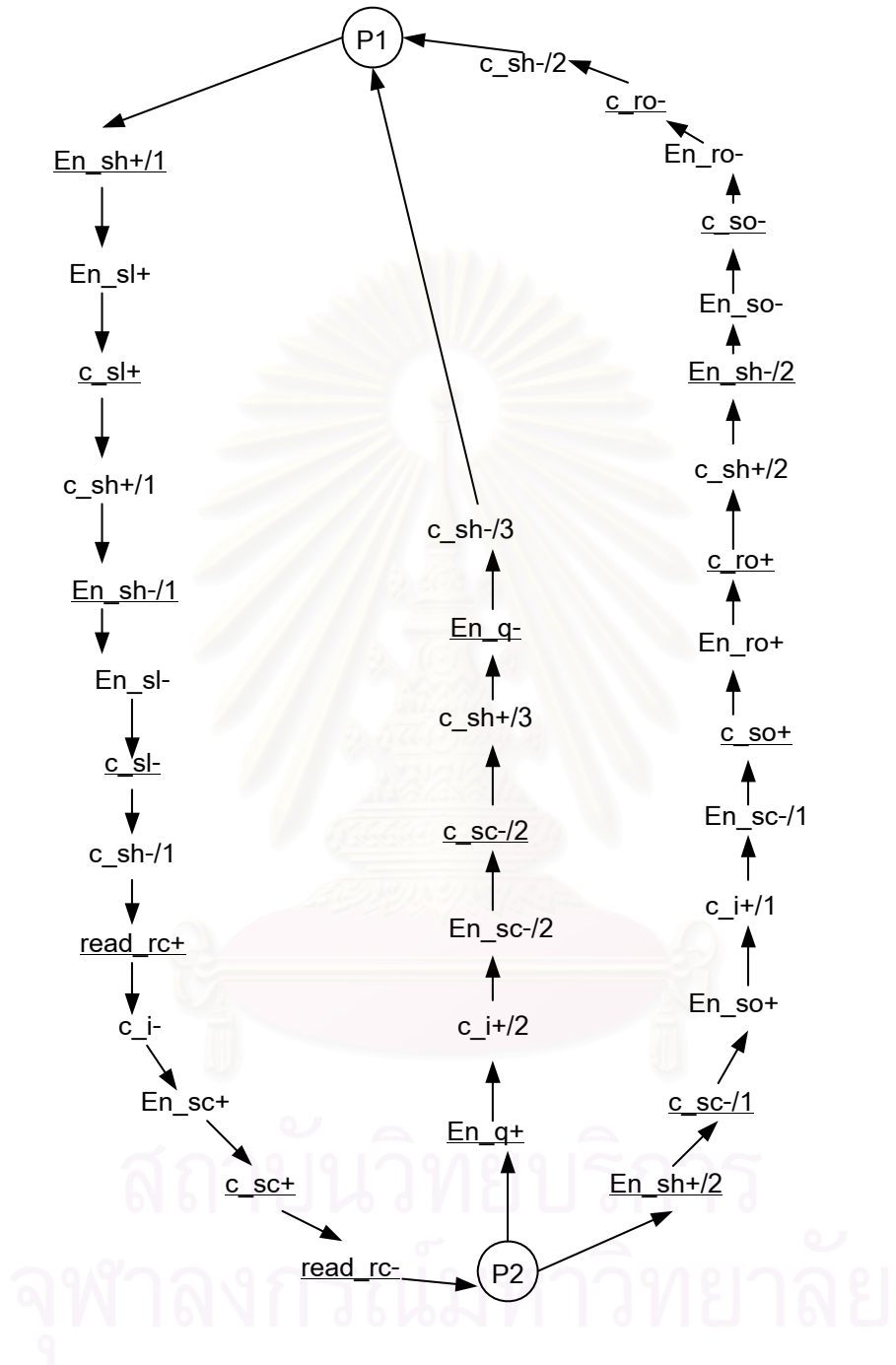


## 1.2 Collision Check Block

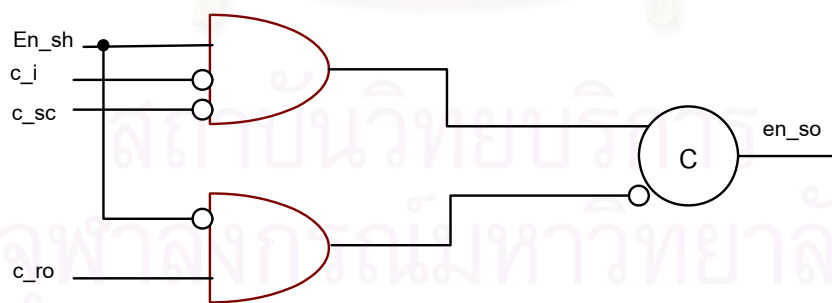
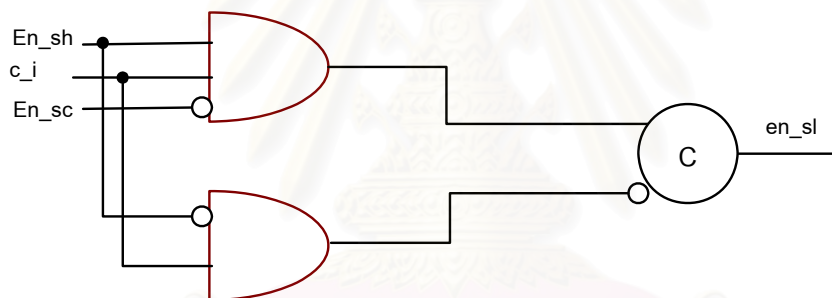
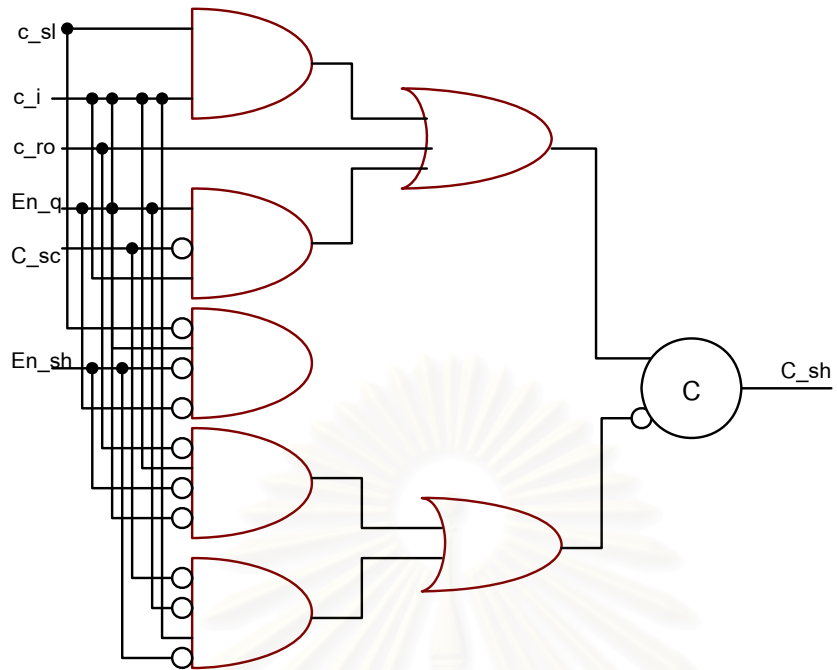


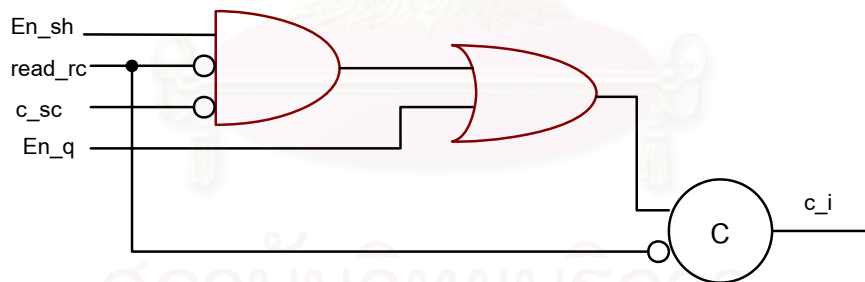
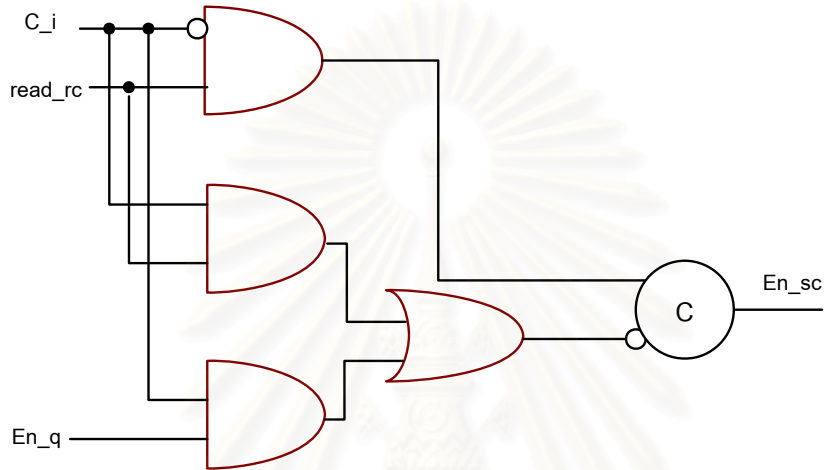
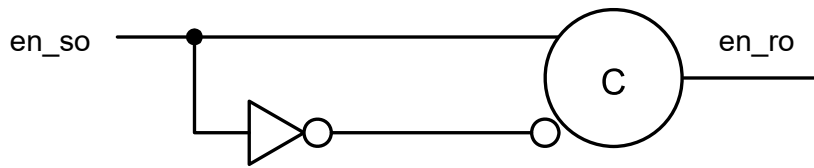


## 1.3 Shifter and Recent state Block



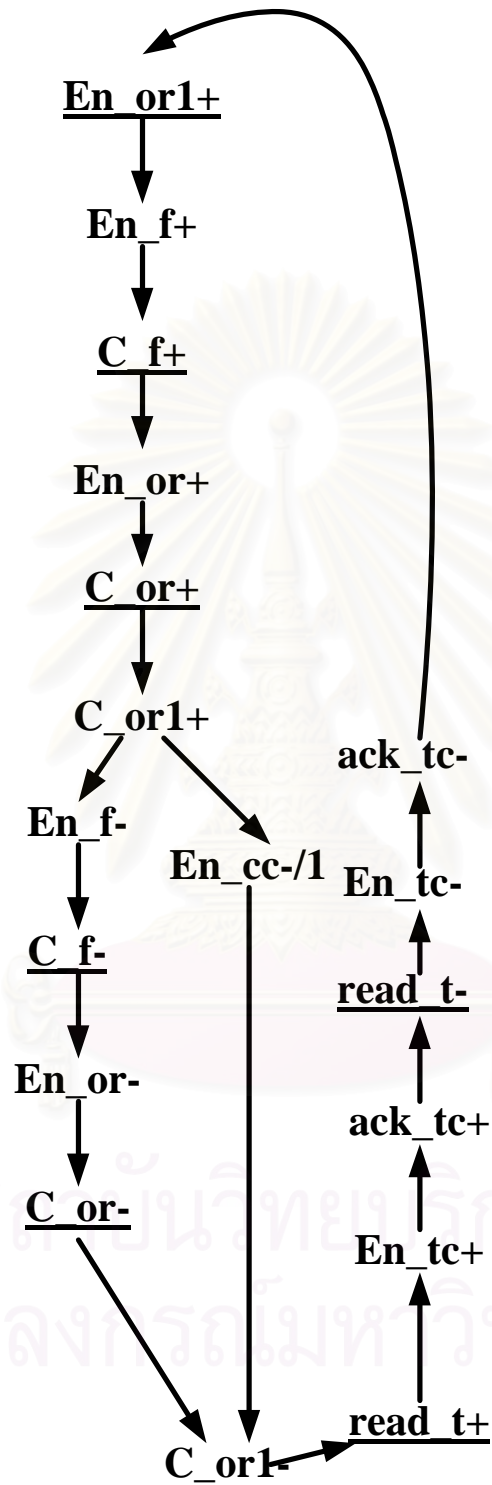


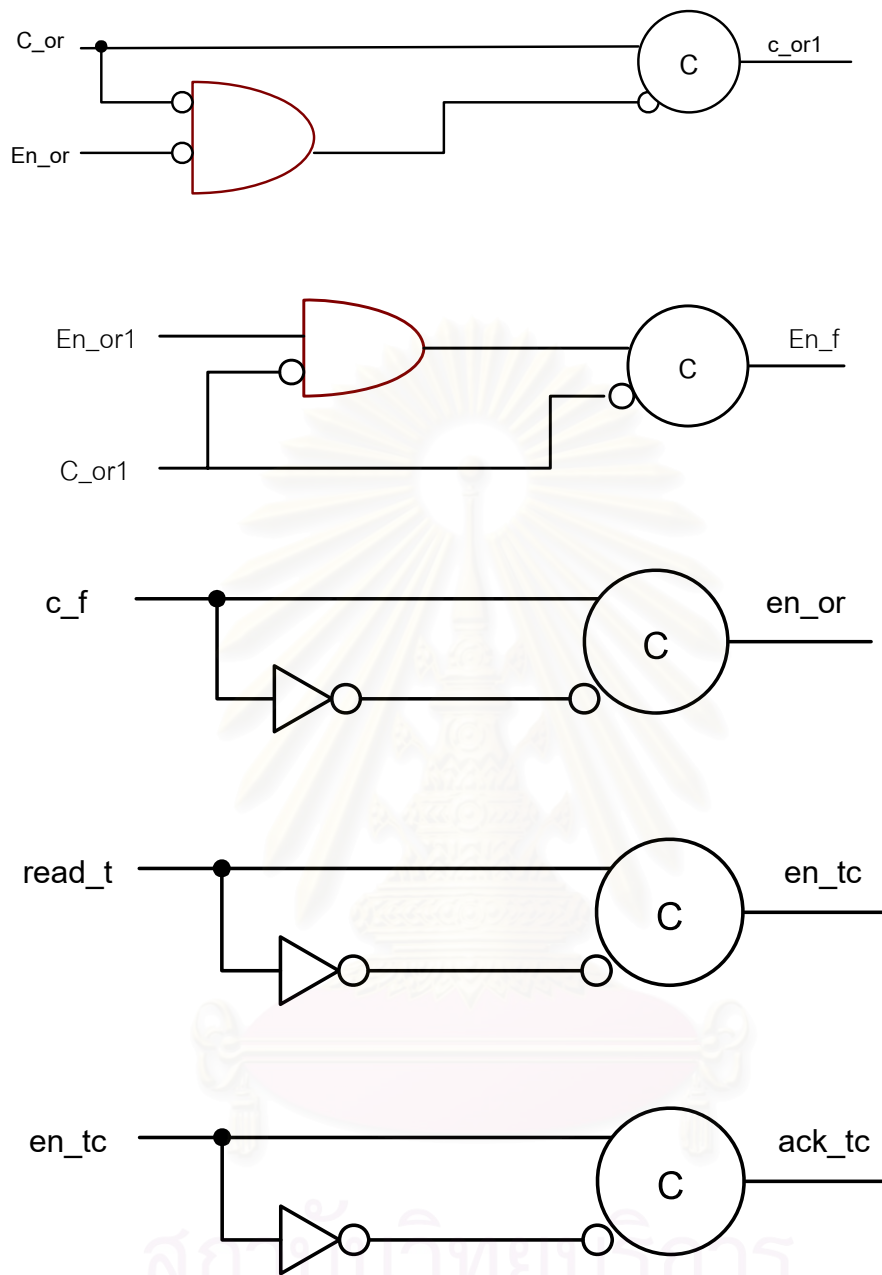




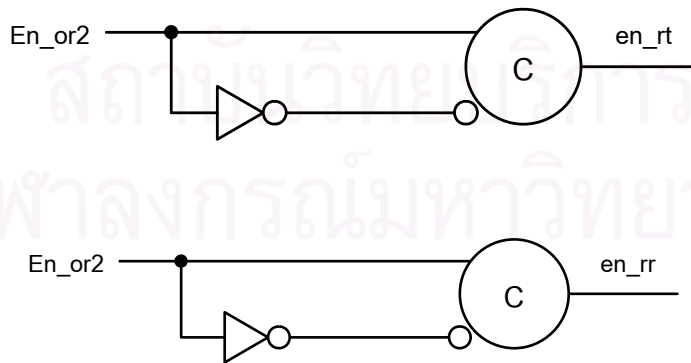
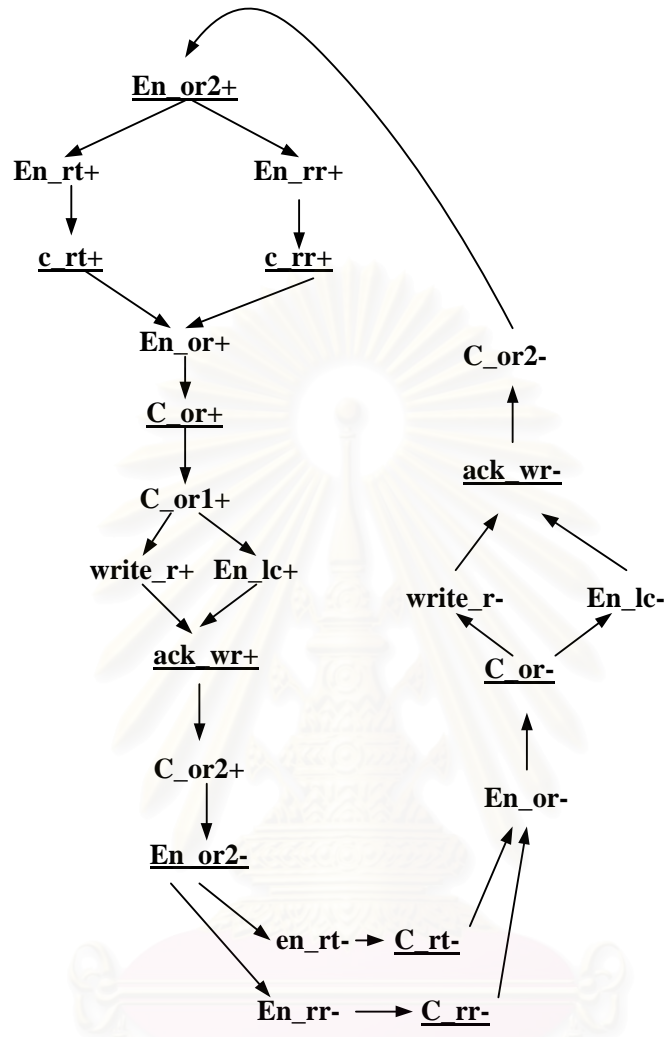
สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

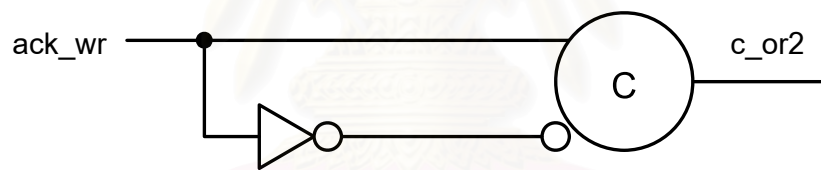
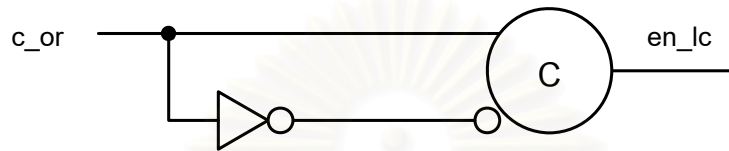
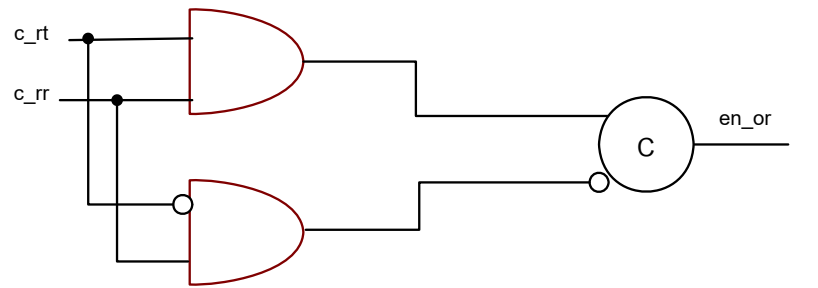
## 1.4 Initial Collision Matrix Block





1.5 State Generator

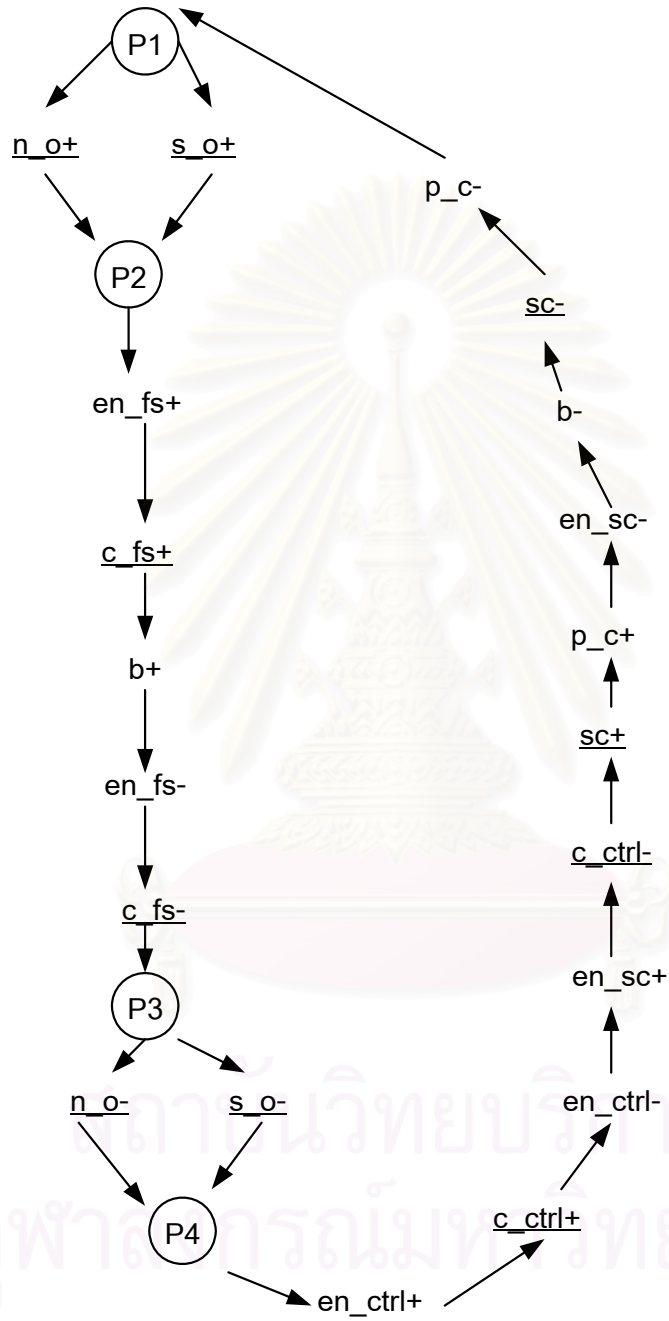


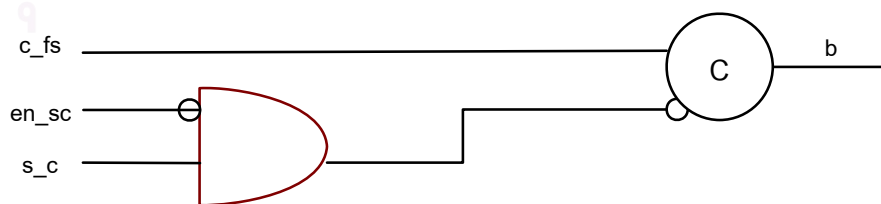
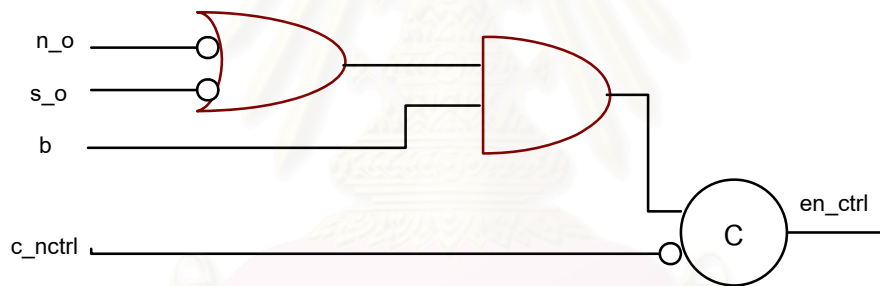
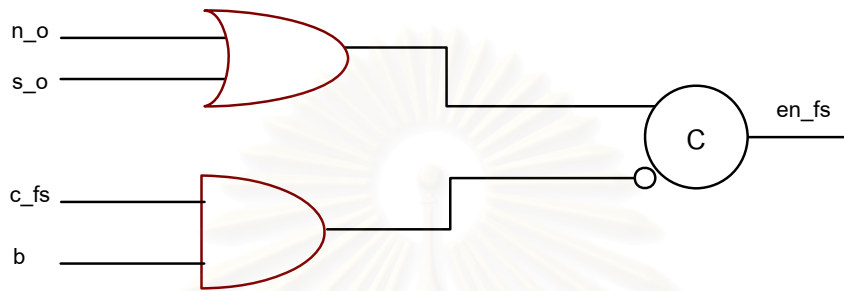
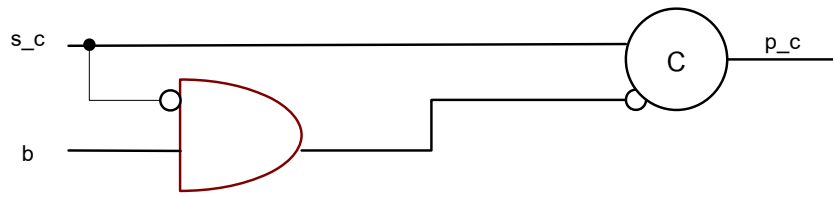


สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

## 2. Stage Controller

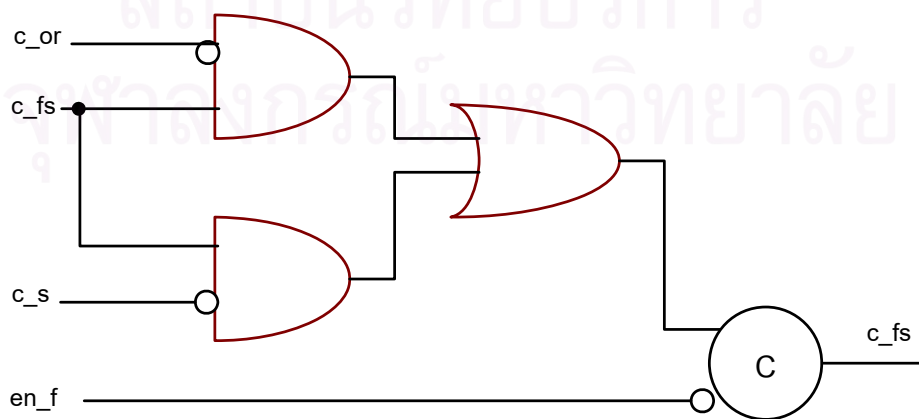
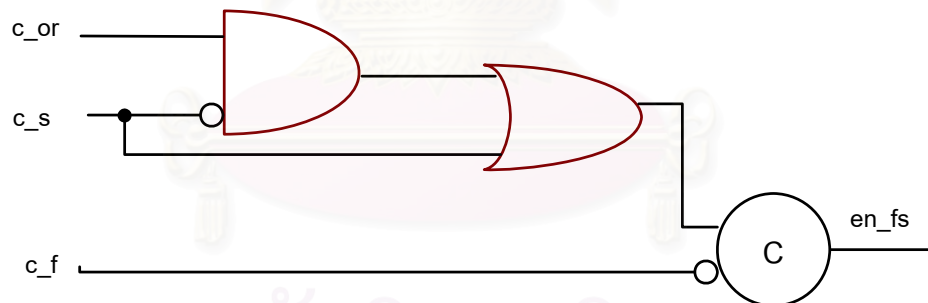
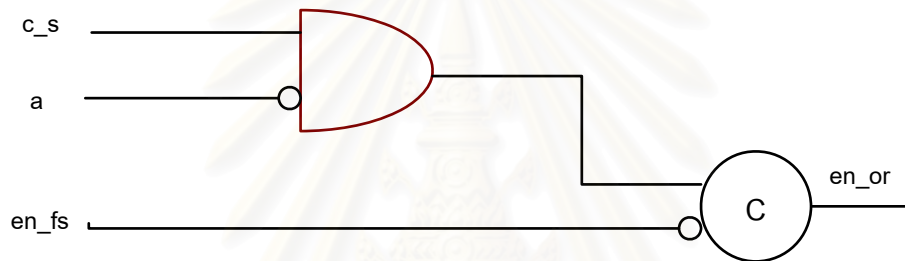
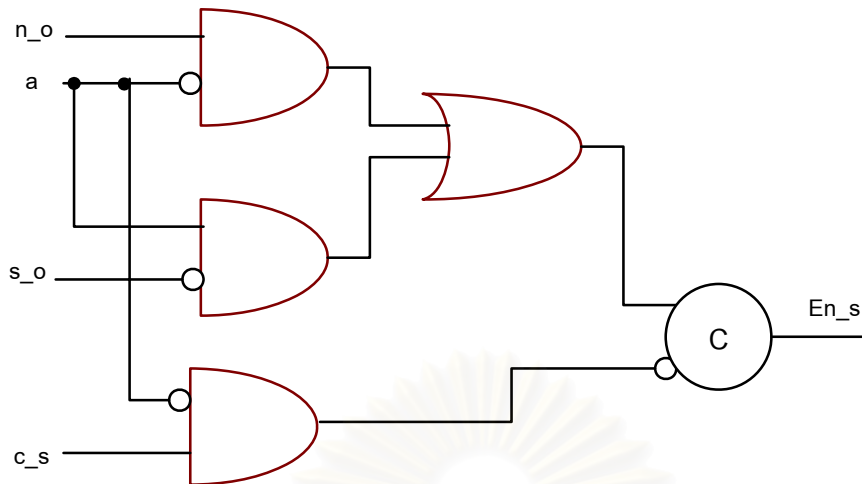
### 2.1 Control block of stage controller

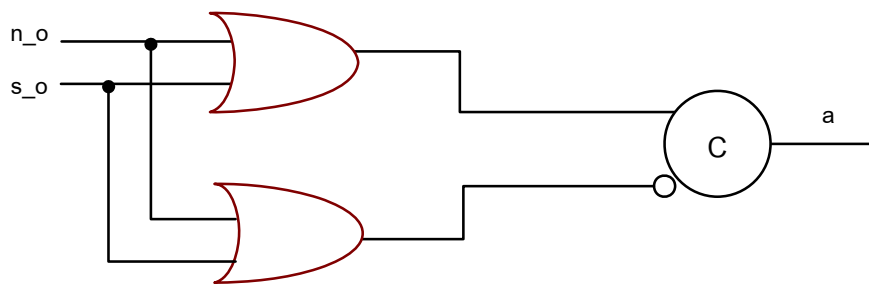












สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

Appendix B  
Published Paper



B.1 Published paper in  
The first Thailand Computer Science Conference (Thcsc 2004)

สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

## An Implementation of Asynchronous Dynamic Pipeline Controller

Benjawan Trabanpreuk  
Computer Engineering Department , Faculty of  
Engineering , Chulalongkorn University  
Email: Benjawan.Tr @student.chula.ac.th

Arthit Thongtak  
Computer Engineering Department , Faculty of  
Engineering , Chulalongkorn University  
Email : Arthit@cp.eng.chula.ac.th

### Abstract

Asynchronous pipeline used to increase speed and performance of asynchronous circuits has been introduced for years. Most studies focus on static asynchronous pipelines. In order to keep performance and avoid stage collision, the scheduling is the main technique for complex pipeline such as dynamic pipeline. This paper presents an implementation of a controller of asynchronous dynamic pipeline for multifunction circuits. The proposed controller consists of two main parts; pipeline scheduler and stage controller. The scheduling scheme based on the reservation table, the collision matrix, and the state diagram. The synthesis procedure from STG (signal transition graph) used for all control parts of circuit. In addition, VHDL was used for this circuit. The simulation result shows that pipeline can work properly, and stage collision is avoided. Furthermore, the controller can be easily applied to any complex asynchronous dynamic pipelines.

**Key words**– Dynamic pipeline, Asynchronous pipeline, Pipeline scheduling

### 1. Introduction

Since clock skew limits the speed of synchronous circuits [Nanya et.al., 1994], asynchronous circuit design has been widely restudied since the last decade. Instead of the use of global clock, the asynchronous circuits response to their signal transitions in the circuits, and output can be sensed as soon as the operation is completed. Accordingly, this becomes one of the advantage of asynchronous circuits in term of speed. Moreover, power consumption of asynchronous circuits is another advantage due to no global clock signal transition.

However, asynchronous circuits are more complicate to design and verify than the synchronous ones. In addition, there are few tools that support asynchronous circuit design. Therefore, the propose of this research is to expand the area of the asynchronous circuit study, especially for an increase of speed. Consequently, in order to increase speed of circuits, asynchronous pipelining is considered here. Conventional pipeline technique has been adopted since late 1960 [Culler D.E., 1999] [Crichow J.M., 1988] [Willkinson B. , 1999]. Asynchronous pipeline was first introduced in 1989 by Ivan E. Sutherland [I.E.Sutherland, 1989]. However, there are mainly studied on static pipeline but still few research papers considered on the asynchronous dynamic pipeline.

Hence, this research focuses on dynamic pipeline that can be reconfigured to perform multifunction [P.M. Kogge, 1981]. As a result, the number of

circuits in a system can be reduced, but control and scheduling part become much more complex [K. Hwang, 1993].

To examine control part and scheduling, and then make them simply is the main purpose of this research. This research used pipeline-scheduling scheme introduced by Kogge. Reservation table, collision matrix, and state diagram are keys. for this scheme. He also presented scheduling circuits for both static and dynamic pipeline. Although they are not suitable for asynchronous circuits, some ideas of his circuits can be used in this research.

STG (signal transition graph) is applied to the circuit because the controller is controlled by hardware, and quite complex. The controller can be controlled by a simpler scheme i.e. software control, but it is slower. After STGs of control part are completed, they must be synthesis circuits. The synthesis procedure adopted to this research is Park's [S.B.Park, 1996] because it appropriate for asynchronous circuits. Park introduced the synthesis of asynchronous VLSI circuits from STG based on SDI delay model with isochronic fork (QDI).

Section 2 and 3 gives brief explanation about asynchronous pipeline and pipeline scheduling. STG and circuit synthesis procedures from STG are in section 4. The design methodology is described in section 5. Then, section 6 and 7 show the overall of asynchronous dynamic pipeline controller circuits and results. Finally, section 8 and 9 are conclusion and acknowledgement.

**2. Asynchronous Pipeline**

Micropipeline [I.E. Sutherland, 1989] was first introduced to asynchronous pipeline in order to improve performance and speed of asynchronous circuits, as shown in figure 1.

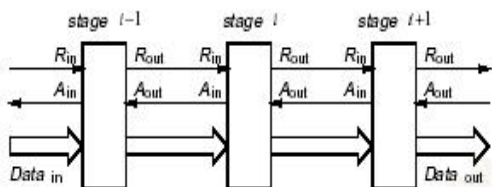


Figure 1 : An asynchronous pipeline structure [8]

It uses bundle data channel that consists of two control signals, request and acknowledge, and data. The request signal is sent to the next state when data is valid, and the next stage send back the acknowledge signal. In addition, there are combination circuit and latch in each pipe stage. Figure 2 represents asynchronous pipeline operation used in this research.

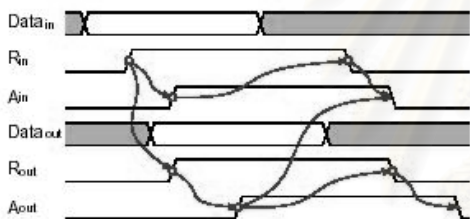


Figure 2: Asynchronous Pipeline operation [8]

**3. Pipeline Scheduling**

Pipeline scheduling is important to guarantee pipeline performance and collision avoidance. If pipeline is a purely linear, the scheduling is simply. Pipelines used in real circuits are more complex, and scheduling also becomes complicated.

The first thing should be known when studying about pipeline scheduling is types of pipeline. There are two types of pipeline.

**1. Static pipeline :** a static pipeline can be reconfigured to perform only one function (unifunction) at anytime.

**2. Dynamic pipeline :** a dynamic pipeline can be reconfigured to perform variable functions (multifunction) at different time.

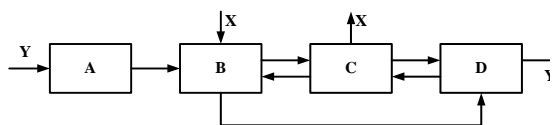


Figure 3 : A four stages pipeline

Figure 3 represent a four stages pipeline used in this research. This pipeline can perform two functions; X and Y. If the pipeline can perform only one function, X or Y, at anytime, it is static pipeline, otherwise it is the dynamic pipeline.

**3.1 Reservation table**

Reservation table represents the dataflow of pipeline. It is described in a two-dimensions tabular. Each row of the reservation table corresponds to the time usage of each stage. Each column is diagram of the internal usage of a pipeline at an instant of time. One reservation can represent the dataflow of only one function of the pipeline. The number of column is evaluation time that is the total time to complete a function. Reservations tables of three functions of the pipeline in figure 3 shown in figure 4

stage /time	0	1	2	3
A				
B	X			
C		X		X
D			X	

(a) reservation table for function X

stage /time	0	1	2	3	4
A	Y				
B		Y		Y	
C			Y		
D					Y

(b) reservation table for function Y

Figure 4 : Reservations tables of the pipeline in figure 3

**3.2 Collision Vector**

An initiation of a reservation table occurs when a function started. The number of time unit between two initiations is latency. When two or more initiation try to use a same stage in a pipeline at the same time, the collision occurs. Forbidden latency is the latency that causes the collision.

Forbidden and nonforbidden (permissible) latencies in a reservation table are used to build a collision vector. M is the evaluation time of table; collision vector = (C<sub>0</sub>, c<sub>1</sub>, ..., C<sub>m-1</sub>). C<sub>i</sub> is 0 if latency i is the permissible latency. C<sub>i</sub> is 1 if latency i is the forbidden latency. For example, collision vector for function X is 10100; it is initial collision vector.

Collision vector is sufficient for static pipeline because it performs only one function. For dynamic pipeline, collision matrix is applied. A collision matrix C is an r x t binary matrix where r is number of reservation tables in a pipeline, and t is a



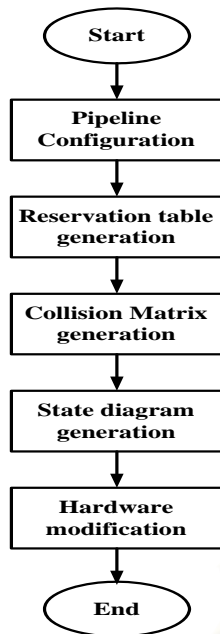


Figure 6: Steps of Design methodology

**5.3 Collision Matrix generation:** The collision matrix generated from this step will be store in registers of initial collision matrix block and is used to generate the current state of pipeline.

**5.4 State Diagram generation:** The state diagram is used to check correctness of the controller by checking the current state of pipeline generated from state generator block. The state diagram is not included in the controller circuit.

**5.5 Hardware modification:** Because of the configurations of the controllers for each pipeline are little different. Note that, scheduler control and stage control module will not be modified because they are the control circuits that can be used by any dynamic pipeline. Similarly, control block of every module of both pipeline scheduler and stage controller will not be modified. Combination circuits of modules must be modified to support each different dynamic pipeline. For pipeline scheduler, the number and size of registers in initial collision matrix, state generator, shifter & current state modules depends on the number of functions and pipeline stages. Furthermore, combination circuit in collision check module must be modified too, and can be improved to support the requirement of designers. For stage controller, function selection, and ctrl\_block must be modified to support the pipeline configuration. The number and size of registers in function\_selection modeles depends on the number of functions and pipeline stages. Similarly, the number of ctrl\_block is equal the number of pipeline functions.

Step 1 depends on the requirement of designers. Step 2 – Step 4 based on pipeline scheduling described in section 3. Step 5 is based on ideas presented in this paper.

## 6. The asynchronous dynamic pipeline controller

The asynchronous dynamic pipeline controller is shown in figure 7. The pipeline scheduler and the stage controller are interconnected with control signals. The scheduler pipeline controller handle all of pipeline scheduling and collision check, while stage controller controls function and data transfer of every stages of the dynamic pipeline. Three signals  $f$ ,  $n_f$ , and  $fi$  are sent from the outside of the controller; such as CPU etc.. In this research we assume that three signals are always valid, and will not be checked by the controller. The asynchronous dynamic pipeline controller consists of two main circuits (two blocks in a dash block ).

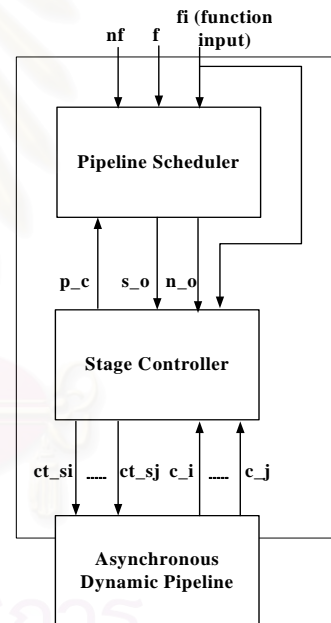


Figure 7: Asynchronous Dynamic pipeline controller

### 6.1 The pipeline scheduler

This block consists of four function blocks and a scheduler control as shown in figure 8. The design is based on pipeline scheduling scheme described in section 3.

Two initial collision matrixes,  $CM_x$ ,  $CM_y$  are stored in three registers in initial collision matrix block. All 0's are stored in the registers of shifter and recent state block at initial time. The scheduler control block is used to control another blocks and synchronized them. The operation of pipeline scheduler are described below:



First, scheduler control block receives  $p_c$  signal that represent states operation completed from stage controller, and then shift left registers in Shifter and recent state Block. Second, The scheduler control block checks whether there is function comes into pipeline. If  $nf$  is high, there is no new operation for pipeline, and scheduler control block sends back  $s_o$  (same operation) to the stage controller and ends the operation of pipeline scheduler. The shift result becomes a current state of the pipeline. If  $f$  is high, one or more new functions come into the pipeline, and then collision check block is enable. It requests current state, and uses that value and  $fi$  value to check state collision.  $C$  is high, if the pipeline can not operate those functions without collision, so functions will not be operated at this moment. The scheduler control block sets  $s_o$  high and ends the operation. If  $n_c$  (no collision) is high, the collision check block sends  $fs$  to initial collision matrix block to select initial matrixes that will be ORed together. Next, ORed result is sent to state generator block that also requests current state. This block ors them together, and send it back as a current state to Shifter and recent state Block. Finally, the scheduler control block ends the operation of pipeline scheduler.

### 6.2 The stage controller

The stage controller is composed of three main parts and a stage control block as shown in figure 9. Shift registers in function\_selection block is stored a set of functions for each pipe stages in sequence. The register size is  $2 \times 4$ ; 5 is the number of stages. Each row consists of three bits; the first two bit represent function, 00 = no function, 01 = function X, 10 = function Y, 11 is not used. A stage can be

used more than one time for a function; for example stage B of Y is used in time 1 and time 3, and data transfer for time 1 and time 3 are different. Thus, the last bit of row in the register is used to represent data transfer of stage; 0 for first data transfer, and 1 for later. For example, the register value is [000,100,000,000]. The value represent that stage B execute function Y at time 1, and another stages doesn't execute function. The next value of register is [000,000,100,000,000]

The operation of the stage controller is divided into two operations.

**6.2.1  $n_o$  is high:** The controller receives  $fi$ , and then function selection block shifts the value in rows of registers to the next stage (depends on reservation table) and changes the first row of register to be the function that will be executed. Then rows of registers are sent to Ctrl\_block. Ctrl\_block send control signal to each pipe stage. After all execute stage (define by  $ex\_st$  signal) are completed,  $s_c$  is high, and  $P_c$  goes high.

**6.2.2  $s_o$  is high:** There is no changes in the first row of registers, and follows the steps in 6.2.1.

## 7. Results

The circuit is simulated by Model Sim as shown in figure 10.  $P_c$  to  $re\_st$  are signal of pipeline scheduler, and the rest are stage controller. Let consider the left hand side of figure 10,  $or\_rs$  and  $or2\_rs$  is the result of initial collision matrix and state generator block.  $Re\_st$  is the value of recent state register which is a shift register.  $Re\_st=[11000,10000]$

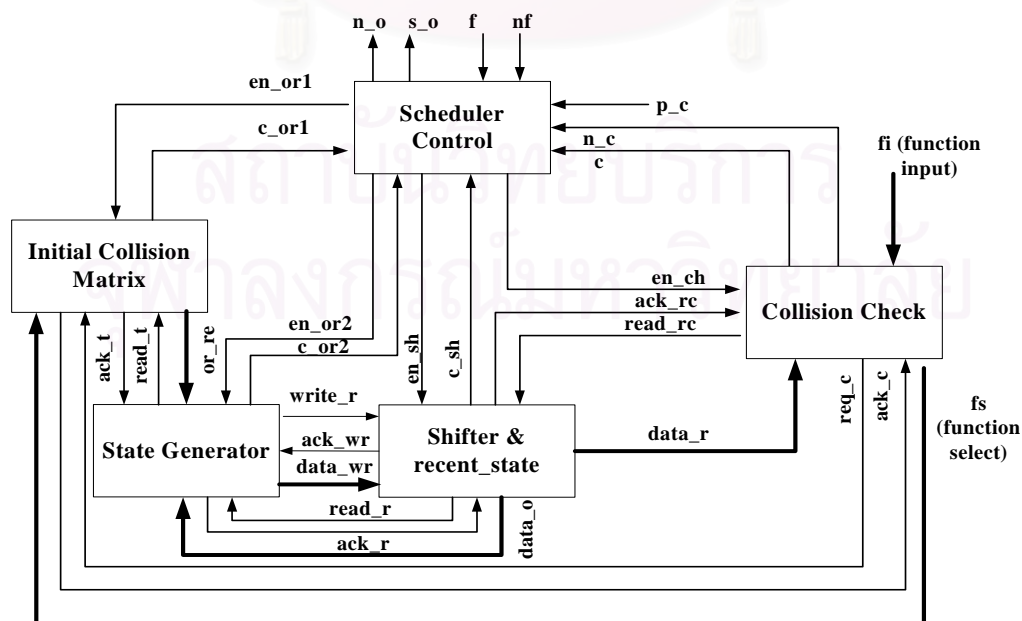


Figure 8: Block Diagram of Pipeline scheduler

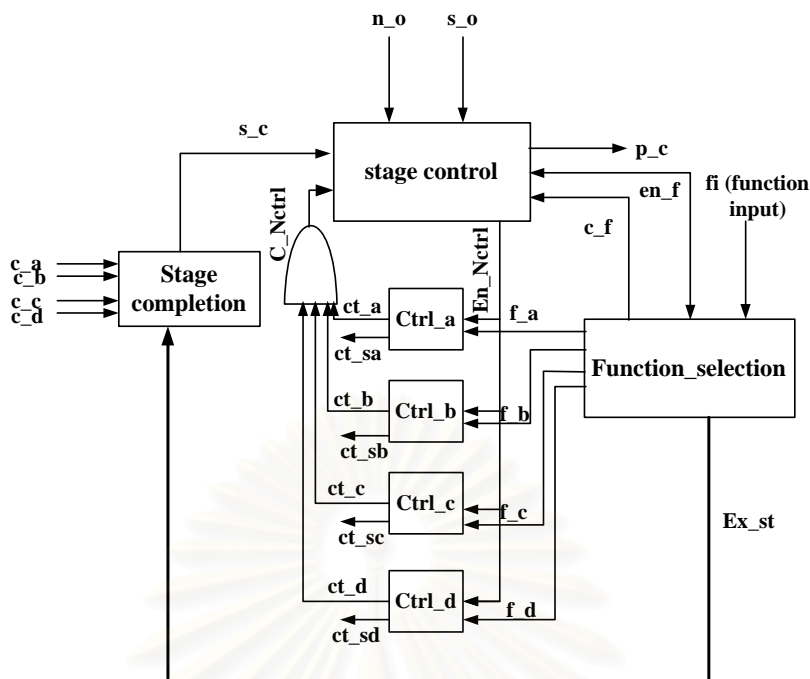


Figure 9: Block Diagram of stage controller

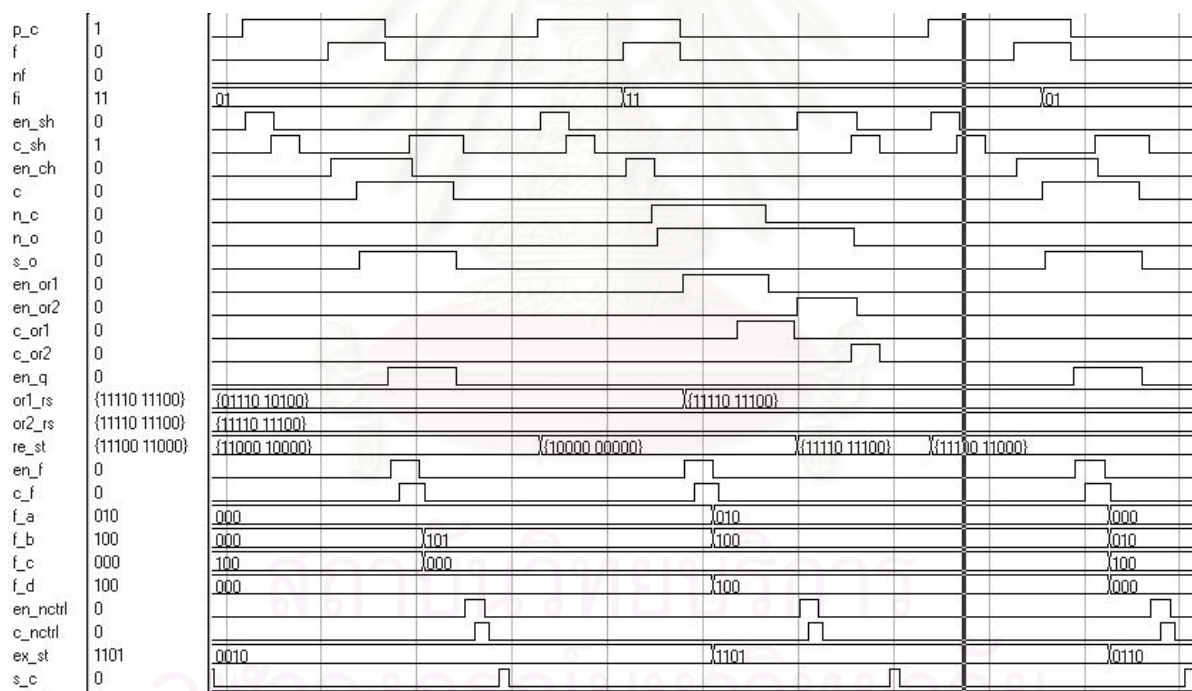


Figure 10 : Simulation result of asynchronous dynamic pipeline controller

fi represents the function initiation ; it consists of two bits ,and each bit represent each initiation of function (Y,X). In the figure 10 , fi = "01"; X function are initiated. When p\_c goes high, shift register must be shifted left. And then, the collision check block is enable by setting en\_ch.

stage controller block, the operation begin after receiving s\_o. Function X can not be executed at that time and the old function (Y) is operated from

The first bit of the first column is 1 , so function X can not be executed without collision. Check collision block checked set c signal high to tell the control block that B function can not be executed at this moment. S\_o (same operation) goes high ; then the scheduler control block end the operation. For time 2 to time 3 (reservation table in figure 4b). The value in registers become [000,101,000,000] ; its mean only stage B are enable. Then en\_nctrl is fired

,and all  $ctrl_i$  are enable. After  $ctrl_a$  to  $ctrl_d$  operation are completed  $c\_nctrl$  is fired. Then, stage B are completed and set  $s\_c$  high. Finally, stage control block end the operation of stage controller and set  $p\_c$  high.

Next, Function Y are initiated because  $fi$  is "11". After collision check, function X and Y can be executed at the same time,  $n\_c$  is high and every block in pipeline scheduler are enabled in order. In addition,  $n\_o$  is high to indicate that stage controller block must execute the new function (X and Y). The operation are similar to the previous cycle except new function, A and B, are executed [Look at time 0 of figure 4a and 4b]. At this cycle,  $re\_st = [11110, 11100]$ , and registers value in stage control is  $[010, 100, 000, 100]$

For the next cycle, right hand side of the picture, function X is initiated. Let consider  $re\_st$ , it indicates that function x can not be executed at this cycle. The operation are the same of the two previous cycle. Function X must wait until the first bit of  $re\_st$  (current state) is 0.

## 8. Conclusion

This research focused on dynamic pipeline to expand the study area of asynchronous pipeline and apply it to multifunction circuits. The effort is the use of STG to implement the pipeline controller from the collision matrix. The simulation result in section 10 showed that the circuit can operate properly without stage collision.

This circuit can be applied to any multifunction circuits using dynamic pipeline with modification as described in design methodology (section 5). The first thing is changing the initial matrix in OR1 block. Second, change a little operation in every block, except control block, to support your pipeline. Notice, no need to change anything in control part of every block.

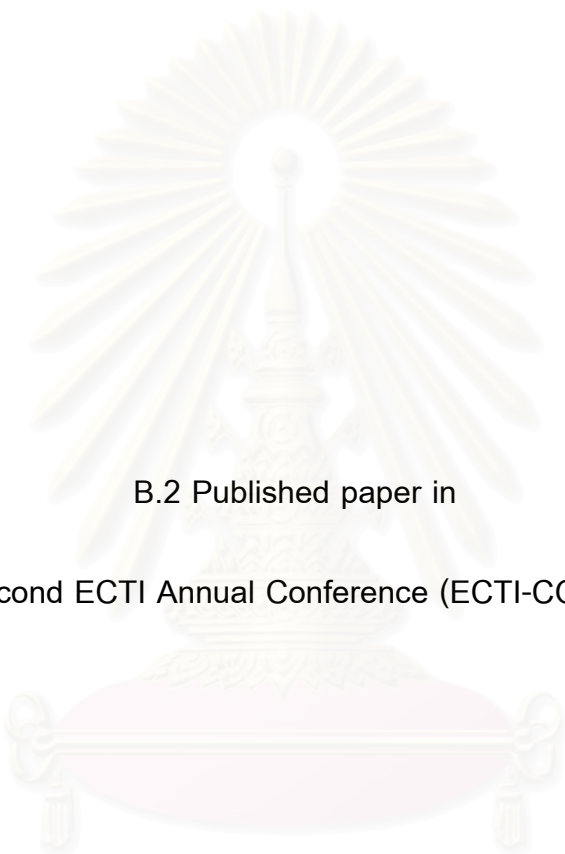
The challenge is to modify the controller to be a purely asynchronous circuit. Global clock is absent in circuit, but stages must wait until the other stage are completed. To speed up pipeline this problem should be eliminated.

## 9. Acknowledement

The research is supported by TJTTP-OECF (Thai and Japan Technology Transfer Project – Japanese Overseas Economic Cooperation Fund)

## Reference

1. Crichow, J.M. An Introduction to distributed and parallel computing, Prentice Hall International(UK) 1988
2. Culler, D.E. Parallel computer architecture: a hardware/software approach, Morgan Kaufmann Publishers, INC. 1999
3. Hauck, S. Asynchronous Design Methodologies: An Overview IEEE Transaction on Computer 83,1 (January 1995):69-93
4. I.E. Sutherland, "Micropipelines", Communications of the ACM, vol.32,#6, pp: 720-738, June 1989
5. J.L. Hennessy and D. A. Patterson, "Computer architecture a quantitative approach", Morgan Kaufman publishers, INC, Sanfrancisco, California, 1996.
6. J. Sparso, S. Furber, R.V. Leuken, A d. Graaf, R. Nouta, "Principle of A Asynchronous Circuit Design", the Netherland, Kluver Academic Publisher., 2001
7. K. Hwang, "Advanced Computer Architecture: Parallelim, Scability, Programmability" New York, New York: Mcgraw-Hill, Inc., 1993
8. K.Y. Yun, P.A. Beerel, and J. Arceo "High-Performance Asynchronous Pipeline Circuits". Proceeding of the 1996 Symposium on Advanced Research in Asynchronous Circuits and Systems, pp: 17-28, March 1996.
9. Nanya T.; Ueno Y.; Kagotoni, H.; Kuwako, H.; Takamura, A. TITAC: Design of a Quasi-Delay Insensitive Microprocessor, IEEE Design & Test of Computer Vol.11, No. 2, (1994):51-63
10. P.M. Kogge, "The Architecture of Pipelined Computer", New York, New York Hemisphere Publishing Company, 1981
11. S.B. Park, "Synthesis of Asynchronous VLSI circuits from Signal Transition Graph Specifications", Phd. Thesis, Tokyo Institute of Technology, 1996
12. S. Furber and P. Day, "Four-phase micropipeline latch control circuits," IEEE Transactions on VLSI Systems, vol. 4, pp. 247-253, June 1996
13. T.A. Chu, "Synthesis of Self-timed VLSI circuits from Graph-theoretic Specifications", Phd. Thesis, Massachusetts Institute of Technology, 1987
14. Wilkinson B., Allen, M. Parallel programming Prentice Hall, 1999



B.2 Published paper in

The second ECTI Annual Conference (ECTI-CON 2005)

สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

# A Design of Asynchronous Dynamic Pipelined Floating-point Arithmetic Unit

**Benjawan Trabanpreuk**

**Arthit Thongtak**

Department of Computer engineering, Chulalongkorn University : benjawan.tr@student.chula.ac.th  
Department of Computer engineering, Chulalongkorn University : Arthit@cp.eng.chula.ac.th

## ABSTRACT

This paper proposes the asynchronous dynamic pipeline floating-point arithmetic unit which consists of two main parts: the dynamic pipelined arithmetic unit and the controller. Dynamic pipelining is used to increase speed and reduce the size of the circuit. The arithmetic unit architecture based on a four-phase micropipeline can operate four functions: add/subtract, negate, absolute and compare. The controller is divided into two main parts: pipeline scheduler and stage controller. The format of floating point is single-precision (32 bits) IEEE 754. VHDL is used to design the circuit, and it is simulated by ModelSim. The simulation result shows that the circuit can operate correctly without stage collision, and cover the IEEE 754 including the exceptions.

**Key words:** PDF, Floating-point arithmetic unit, Dynamic pipeline, Asynchronous pipeline

## 1. Introduction

Asynchronous circuit design has been studied in wider areas because of the clock skew in synchronous circuits since the last decade [1]. The clock skew occurs when the global clock used to synchronize their operation can not be distributed to the entire system at the same frequency. It is a serious problem because the system may malfunction. It is very hard to avoid this problem when designers want to design high-speed circuits. As a result, it limits the speed of synchronous circuits.

Asynchronous circuits do not use the global clock, so the clock skew is avoided. Circuits respond to signal transitions at any time, and outputs can be sent instantly after the operation completion. Thus, another advantage of the asynchronous circuit is speed because it is not limited by the slowest part. Moreover, Low power consumption is another advantage because the signal transitions are made only when necessary.

Nevertheless, asynchronous circuits are harder to design and verify than the synchronous ones. Therefore, close study of the asynchronous circuits is neither now popular nor in wide area, although asynchronous circuits have been studied for decades. Therefore, the expansion of asynchronous circuits studying is required

To extend the area of asynchronous circuits design is the first proposal of our research. There have been many asynchronous processors presented since the last decade, such as FAM, AMULET1-2, NSR etc. [2]. However, the asynchronous floating-point arithmetic unit required for the processor to serve scientific calculation or graphic design is not proposed.

Designing the asynchronous floating-point arithmetic unit is the main purpose of our research. In addition, increasing the speed of the circuit is another purpose, so pipelining is applied to the circuit. The operation of arithmetic unit is split into pipe stages (called stage in this paper).

Pipelined floating-point circuits have been introduced such as MIPS R4000 [3], Multi-Mode pipeline [4-5], Pipeline Packet Forwarding [6-7], Floating point operator of Amphion [8], etc. However, all of them are synchronous circuits.

The floating-point arithmetic unit presented here can be operated four functions: add/subtract, negate, absolute and compare. After stages of each function are considered, we find the redundancy of stages usage. Accordingly, four functions can be combined into one circuit and reduce the circuit's size. Then, the dynamic pipelining is considered because it is a type of pipeline that can perform multifunction. Unfortunately, the pipeline scheduling becomes dramatically complex and leads to the complex controller. The controller applied to the circuit uses the pipeline-scheduling scheme introduced by Kogge [9]. Reservation table, collision matrix, and state diagram are keys for this scheme.

Section 2 and 3 give the background of the dynamic pipeline and asynchronous pipeline. Section 4 and 5 is the architecture of our design of the arithmetic unit and controller. The simulation result shown in section 6. Finally, the conclusion and acknowledgements are in section 7 and 8.

## 2. Dynamic pipeline

There are two types of pipeline:

**1. Static pipeline:** a static pipeline can be configured to perform only one function (unifunction) at anytime.

**2. Dynamic pipeline:** a dynamic pipeline can be configured to perform variable functions (multifunction) at different times.

Fig. 4 represents the seven stages pipeline used in this asynchronous floating point arithmetic unit. This

pipeline is configured to perform four functions as described above. If the pipeline can perform only one function at anytime, only add/subtract function or only negate function, it is the static pipeline; otherwise it is the dynamic pipeline.

The dynamic pipeline scheduling scheme used in the controller is based on three keys: reservation table, collision matrix and state diagram. The three keys used to describe the behavior of dynamic pipeline and define its current operation and state. In addition, it can keep the performance and avoid stage collision. The pipeline scheduling begins with the two column diagram called the reservation table that represents the dataflow of pipeline. Fig. 1-a and 1-b represent the dataflow of add/subtract and negate function shown in fig.4. Note that one reservation table is for one function. Then, the initial collision matrixes that indicate the set of possible initiation, a function or more can be initiated without stage collision are created from every reservation tables of dynamic pipeline. Finally, the state diagram is created by the initial collision matrixes that become state one. The next stage are constructed by state one, initiation set and collision matrixes consideration, and become the current state. After that, the current state and collision matrixes are used to construct the next step. Repeat that until all initiation sets are considered. The procedure of dynamic pipeline scheduling in details is in [9] and [10].

time \ stage	1	2	3	4	5	6	7	8
U	A							
C		A						
E			A					
A				A				
Ch					A	A		
N						A		
R								A

time \ stage	1	2
U	N	
C		N
E		
A		
Ch		
N		
R		

a: add/subtract function

b: negate function

Fig 1: Example of reservation tables

### 3. Asynchronous pipeline

Asynchronous pipeline was first introduced in 1989 by Ivan E. Sutherland who named his pipeline “micropipeline” [11].

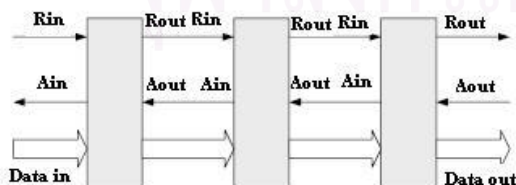


Fig. 2: An asynchronous pipeline structure

Fig. 2 represents the asynchronous pipeline structure used in our design. It uses bundle data channel that consists of two control signals, request and acknowledge, and data channel. Fig. 3 represents the

four-phase asynchronous pipeline operation presented in [12], and it is applied to be a backbone pipeline of our floating-point arithmetic unit.

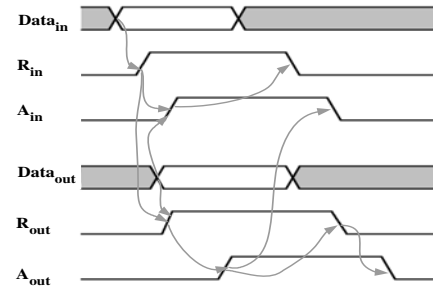


Fig. 3: A four-phase asynchronous pipeline operation

### 4. Arithmetic Unit

This unit is based on single precision (32 bits) IEEE standard 754-1985 [13-15] including standard format and exceptions. The special bit, Guard Bit, is extended to the LSB of Mantissa in standard format to increase the precision of the arithmetic result. Then, the rounding technique is used to convert the result to the 32 bits standard format. The round to nearest is the rounding technique used in our design: under half is 0 and above half is 1.

The arithmetic unit operates four functions: add/subtract, negate, absolute and compare. The operation of each function is split into stages to work as a pipelined arithmetic unit. After consideration and comparison in terms of operation and stage, the arithmetic unit can then be divided into stages as follow:

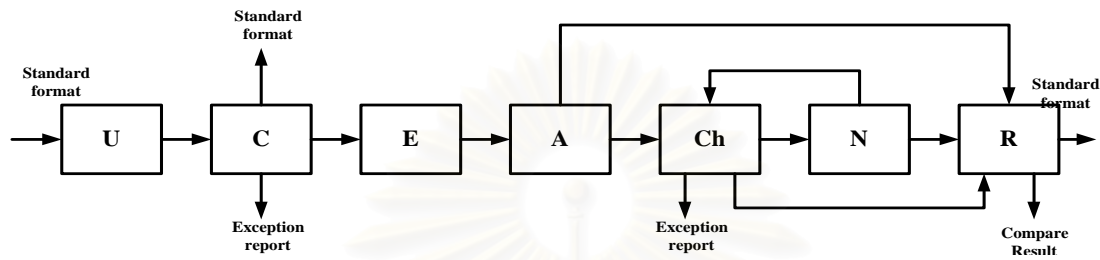
- U (Unpack): distribute input into 3 parts: Sign bit, Exponent and Mantissa.
- C (Check for zero/Change sign bit): check that input is 0 or precise exception or not, and change sign bit for negate and absolute function.
- E (Exponent Adder/Subtractor): add and subtract exponents of the input.
- A (Adder): add and subtract mantissa of the input. The complement circuit is in this stage.
- Ch (Check for Exception): check whether the result is precise exception or not.
- N (Normalize): convert the result into formatted number used in circuit, not a standard format.
- R (Rounding): round the result and convert the result to 32 bits IEEE 754 standard format.

All functions use redundancy stages as shown in Table 1. The usage and data transfer stages in the dynamic pipeline used for this arithmetic unit is shown in Fig. 4.

Stages Ch and N are considered to be a loop because some results need to be checked more than once and to format them to the standard format. In Fig. 1-a, we need to reserve Ch twice to avoid stage collision because we can not know that whether Ch will be used once or twice when inputs come into the arithmetic unit. It is time consumption that happens when the result uses Ch only once, but it is still better than stage collision.

**Table 1:** Arithmetic function and stage usage

Arithmetic Function	Stage Usage
1. Add/Subtract	U, C, E, A, Ch, N, Ch,R
2. Negate	U, C
3. Absolute	U, C
4. Compare	U, C, E, A, R

**Fig. 4:** Stage and Data transfer between stages

## 5. Control Unit

The asynchronous dynamic pipeline controller is shown in Fig. 5. The pipeline scheduler and the stage controller are interconnected with control signals. The pipeline scheduler handles all of the pipeline scheduling and collision check, while the stage controller controls functions and data transfer of every stage. Three signals— $f$ ,  $n_f$ , and  $fi$ —are sent from the outside of the controller; such as the CPU etc. In our design we assume that the three signals are always valid, and will not be checked by the controller. The asynchronous dynamic pipeline controller consists of two main circuits (two blocks in a dash block).

All of the operations of the control parts of the controller are described by STG (Signal Transition Graph); they are then synthesized to circuits. The circuit synthesis procedures from the STG based on Park's [17]. Circuits operate correctly under the input/output mode and QDI. STG is a graph-based method used to describe a behavior of a circuit. It is an interpreted Petri-Net introduced in 1987 by T.A. Chu [18-19]. It can describe concurrent and choice operation in a circuit. The control unit is our own design and has been proposed in [20].

## 6. Simulation result

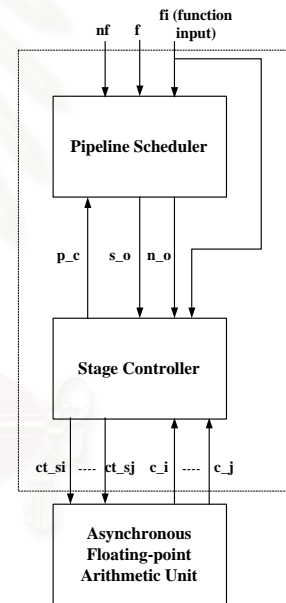
The simulation base on adding delay element in order to check the asynchronosity of the circuit. Therefore, the result is based on estimate delay, not actual delay.

The simulation shown in fig.6 presents operations of two functions: Adder/Subtractor ( $0.5_{ten} + 0.75_{ten}$ ) and compare in the last four stages.  $p_c$  is the signal that starts the cycle of operations. It is high when every enabled stages completes their operation.  $std\_result$  is the result of adder/subtrator function (fourth cycle).  $Cmp\_result$  is the result of compare function (third cycle).  $Out\_type$  is used to identify the result: 0 is adder/subtrator, 1 is compare.

normalized. The third cycle in fig.6 shows that two functions are operated at the same time and it is an advantage of dynamic pipeline. Moreover, negate function can be initiated ( $fi$  in a rectangle) in this cycle because it will not cause the stage collision (consider from stage diagram).

## 7. Conclusion

In this paper, we have presented the asynchronous floating-point arithmetic unit using the dynamic

**Fig. 5:** Asynchronous Dynamic pipeline controller

pipelining technique. It can work properly without stage collision and cover the IEEE 754 standard. Dynamic pipelining reduces size and increases the speed of the arithmetic unit. This is the main advantage of our design. Furthermore, this arithmetic unit can be applied to other asynchronous processors or systems because we use the four-phase micropipeline as the backbone of the arithmetic unit. This circuit applies the controller we designed for pipeline scheduling and stage control to maintain performance and stage collision avoidance.

The controller must know the data flow of functions before their inputs come to the pipeline. Thus, the configuration must be fixed; it is satisfactory for every function except the add/subtract function. The stage usage





## Biography

Miss Benjawan Trabanpreuk was born in December 15, 1978. I obtained my bachelor degree in Computer Engineering from Prince of Songkla University in 2000. Then, I worked as a software engineer before I furthered my master degree in Computer Engineering at Chulalongkorn University in 2002.



สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย