T.C.
BAHÇEŞEHİR ÜNİVERSİTESİ

NEW LOGIC ARCHITECTURES FOR ROUND ROBIN ARBITRATION AND THEIR AUTOMATIC RTL GENERATION

M.S. Thesis

ONUR BAŞKİRT

İSTANBUL, 2008

T.C.

BAHÇEŞEHİR ÜNİVERSİTESİ

INSTITUTE OF SCIENCE

ELECTRICAL & ELECTRONICS ENGINEERING

NEW LOGIC ARCHITECTURES FOR ROUND ROBIN ARBITRATION AND THEIR AUTOMATIC RTL GENERATION

M.S. Thesis

ONUR BAŞKİRT

Advisor: Dr. H. FATİH UĞURDAĞ

İSTANBUL, 2008

T.C

BAHÇEŞEHİR ÜNİVERSİTESİ

INSTITUTE OF SCIENCE

ELECTRICAL & ELECTRONICS ENGINEERING

Name of the thesis: New Logic Architectures
for Round Robin Arbitration
and Their Automatic RTL Generation
Name/Last Name of the Student: Onur BAŞKİRT
Date of Thesis Defense: June 6, 2008

The thesis has been approved by the Institute of Science.

Prof. Erol SEZER
Director

_____

I certify that this thesis meets all the requirements as a thesis for the degree of Master of Science.

Asst. Prof. Bülent BİLİR
Program Coordinator

_____

This is to certify that we have read this thesis and that we find it fully adequate in scope, quality, and content as a thesis for the degree of Master of Science.

Examining Committee Members	Signature
Asst. Prof. H. Fatih UĞURDAĞ	_____
Asst. Prof. Sezer GÖREN UĞURDAĞ	_____
Prof. Nizamettin AYDIN	_____
Asst. Prof. Bülent BİLİR	_____

ACKNOWLEDGMENTS

It has been an adventurous journey. This journey is called “MS Thesis”. When I was a senior at Bahçeşehir University, I saw a course name in the elective course list in Spring semester. That course was “EENG4050 VLSI Design”. Then, I picked that course and my journey started at point. Thanks to EENG4050, I met Dr. H. Fatih Uğurdağ and I entered the world of digital design.

I would like to express enormous appreciation to my advisor, Dr. H. Fatih Uğurdağ, for his invaluable guidance throughout my long and adventurous MS journey. His many years of digital circuit design experience in Silicon Valley has allowed him to focus on critical and interesting issues of digital design problems. It has been a matchless privilege to be his assistant, and without doubt the credibility and prestige of his name will continue to open doors for me in the future. ACTreS project with Intersil in Silicon Valley, FPGA based GPF card Project with Boğaziçi University and MIT, and my STMicroelectronics employment is an example of his prestige, credibility, and network. I also want to express my gratitude for Dr. Sezer Gören Uğurdağ. It was her connections that allow us to go to University of California at Santa Cruz as a group (BUVLSI) for a three month research appointment in the summer of 2006 for HDLtk project.

I would like to thank the members of my thesis committee for their valuable feedback. I would like to thank Dr. Bülent Bilir for his insightful comments. Also, I would like to thank Dr. Nizamettin Aydın, a member of BUVLSI.

For their help and invaluable friendship, my special thanks go to Aylin Saydağ and Sait Can Saydağ. I will never forget my Orange County visit, their kindness and warmth. Additionally, I am extremely grateful to Aylin and Sait for their help with proofreading of my thesis.

I would like to thank Erinç Topdemir, who was always there for me as a fellow graduate student, as a colleague (teaching – research assistant), and more importantly as one of my best friends. By the way I would like to thank Bahçeşehir University supporting me as a graduate assistant from 2005 to 2007. I also thank all of my ex-colleagues at Bahçeşehir University.

I sincerely appreciate everybody's kindness and warmth at ST. They have truly created a wonderful environment, and I am very pleased to be a member of this team. First, I would like to thank my manager Fatma Özdemir for her understanding and endless support. I want to thank Levent Çetrez for his technical guidance and patience. I have learnt many things from him. I thank my team leader Görkem Canverdi and my teammate Levent Ergün. Whenever I got in trouble, they always helped me. I would like to express my graditute to Ömer Yetik for proofreading and his amazing impersonate talent. I am very thankful to Burak Akoğuz for technical suggestions and our enjoyable table tennis matches. I thank Erdal Öztürk for discussions on technical and more importantly non-technical issues. I also thank Bariş Güven, Cağdaş Sengel, Sinan Topçu, Orkun Sağlamdemir, and the funniest colleague of all Sırma Altay. There are so many people that deserve my appreciation for their friendship, support, and help. If I had to explain why I have to thank each person, these acknowledgments would have to be a chapter in this thesis.

Finally, I would like to thank my wonderful parents. Words cannot convey my appreciation of their love. This thesis would not have been possible without their help. For their dedication to my education, I dedicate this thesis to my parents.

ABSTRACT

NEW LOGIC ARCHITECTURES
FOR ROUND ROBIN ARBITRATION
AND THEIR AUTOMATIC RTL GENERATION

BAŞKİRT, Onur

Electrical & Electronics Engineering

Advisor: Asst. Prof. H. Fatih UĞURDAĞ

June 2008, 72 pages

Resource arbitration is a major problem in communications and computer systems. One of the most prevalent usage areas of arbitration is in computer networks. In gigabit and terabit routers, the challenge is to design ultra high speed, cost effective, and fair arbitration hardware to speed up packet forwarding. This issue is highly important for supporting high quality multimedia services in next generation networks.

This thesis is focused on architectures for fast and area efficient round robin arbiters (RRA) and their Register Transfer Level (RTL) design generation. One of the most notable works in this area is the work of Pankaj Gupta and Nick McKeown at Stanford University — which we call Stanford Round Robin Arbiter (STA_RRA). Although there have been further enhancements on top of STA_RRA, we have seen that there is still room for improvement in both speed and area departments.

This thesis work proposes two new RRA logic architectures with better speed or area metrics than STA_RRA and its variants. One of the proposed RRA designs is focused on achieving minimum area results, and the other one is designed for speed. The novelty of these designs is in their use of parallel prefix tree (PPT) algorithms for thermometer encoding and priority encoding operations. Synthesis of proposed arbiters and their rivals were carried out from 8 bits to 256 bits. Benchmarks of 256 bits arbiters show that our proposed architectures perform better than their rivals by a factor of 42% in speed and 22% in area.

Keywords: Round Robin Arbiters, RTL Generation, Parallel Prefix Tree Algorithms

ÖZET

DEĞİŞMEZ ZAMAN PAYLAŞIMLI İŞ-DÜZENLEME İÇİN YENİ MİMARİLER VE BU MİMARİLERİN OTOMATİK YTS ÜRETEÇLERİ

BAŞKİRT, Onur

Elektrik - Elektronik Mühendisliği

Tez Danışmanı: Yrd. Doç. Dr. H. Fatih UĞURDAĞ

Haziran 2008, 72 sayfa

İşlem isteklerinin sıraya konulması (iş-düzenleme), bilgisayar ve iletişim sistemlerinin önemli problemlerinden birisidir. İş-düzenleme işleminin en çok kullanıldığı alanlardan birisi bilgisayar ağlarıdır. Gigabit ve terabit yönlendirici tasarımının önemli uğraşlarından biri, hızlı, maliyeti düşük ve adil iş-düzenleyici donanımları tasarlayarak paket yönlendirme işlemini hızlandırmaktır. Bu konu, yüksek kaliteli, gelecek nesil, çoklu-ortam servislerinin desteklenmesi için son derece kritiktir.

Bu tezde, hızlı ve alan açısından verimli iş-düzenleyici mimarilerine ve bunların Yazmaç Transfer Seviyesi (YTS) tasarım üreteçlerine odaklanılmıştır. Bu alanda en çok dikkate değer çalışma Stanford Üniversitesi'nden Pankaj Gupta ve Nick McKeown'un çalışmasıdır. Biz bu çalışmaya STA_RRA adını verdik. Daha sonraları STA_RRA üzerinde iyileştirme çalışmaları yapılmasına rağmen, hala hız ve alan açısından ilerleme kaydedilebileceğini gördük.

Bu tezde, STA_RRA ve değişik türevlerinden hız ve alan açışından daha iyi iki yeni iş-düzenleyici mimarisi önerilmektedir. Önerilen iş-düzenleyici tasarımlarından birisi minimum alan sonuçlarına odaklanırken, diğeri hız için tasarlanmıştır. Bu tasarımlardaki yenilik, termometre kodlamasında ve öncelik kodlamasında Paralel Prefiks Ağaç yordamlarının kullanılmasıdır. Önerilen ve rakip iş-düzenleyiciler, 8 bit'ten 256 bit'e kadar sentezlenmiştir. Yapılan karşılaştırma çalışmalarında, bizim iş-düzenleyicilerimizin rakip iş-düzenleyicilere göre hız açısından 42% ve alan açısından 22% oranla daha iyi sonuç verdiği görülmüştür.

Anahtar Kelimeler: Değişmez Zaman Paylaşımlı İş-düzenleyiciler, YTS Üretimi, Paralel Prefiks Ağaç Yordamları

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Round Robin Arbiter	:	RRA
Stanford Round Robin Arbiter	:	STA_RRA
Chinese Round Robin Arbiter	:	CHN_RRA
Programmable Priority Encoder	:	PPE
Binary Tree Search	:	BTS
Parallel Round Robin Arbiter	:	PRRA
Improved Parallel Round Robin Arbiter	:	IPRRA
Switch Arbiter	:	SA
Ping Pong Arbiter	:	PPA
Parallel Prefix Tree Round Robin Arbiter	:	PPT_RRA
Parallel Prefix Tree Round Robin Arbiter Resource Sharing	:	PPT_RRA_RS
Parallel Prefix Tree Round Robin Arbiter Best Timing	:	PPT_RRA_BT
Resource Sharing	:	RS
Round Robin	:	RR
Grant Unit	:	GUNIT
Virtual Output Queue	:	VOQ
OR Binary Tree	:	OR_BT
Simple Priority Encoder	:	Smple_PE
Thermometer Encoder	:	tothermo
Han Carlson	:	HC
Brent Kung	:	BK
Kogge Stone	:	KS
Ladner Fisher	:	LF
Complementary Metal Oxide Semiconductor	:	CMOS
Set Max Area	:	SMA
Compile Incremental	:	CI
Map Effort High	:	MEH
No Constraint	:	NC

1. INTRODUCTION

As chip manufacturing technology shrinks toward sub-nanometers, a chip die will comprise more and more of processing blocks. Interconnection, communication, and utilization of shared resources of these blocks are getting more complicated for System-on-Chip (SoC). This complex structure introduces an important challenge to the designer: fast and fault-free on-chip communication. When processing blocks access a shared resource simultaneously, arbitration of these clients has to be ensured. Priority encoders (PE) and arbiters are widely used to allow only one block to access a shared resource. Priority encoding scheme always selects the highest precedence as defined by a priority sequence. As a result of this static scheme, unfairness is revealed. This unfairness is also called starvation. Conversely, programmable priority encoder (PPE) provides a non-static scheme to alter the priority sequence during an operation. These are the core functional blocks of round robin arbiters (RRA). If an arbiter is designed in round-robin fashion, the priority of the system is altered in every cycle, and round-robin arbiter selects the highest priority starting from the last selected request. This technique almost always guarantees fairness.

RRAs are widely used in network switches and routers. Network switches and routers consist of crossbar switches as the internal switching fabric, as shown in Figure 1.1. A crossbar switch comprises of three macro blocks: Input FIFO buffer, arbiter/scheduler, and a crossbar fabric core.

Switch scheduling algorithms are important aspects to implement high speed network switches. These algorithms are implemented in schedulers/arbiters. A scheduling algorithm selects input packets and generates proper control signals for crossbar fabric to set up conflict free paths between input ports and output ports. Then, the crossbar core transfers the requests or packets according to granted control signals. Hence, in order to ensure high speed and fairness, a crossbar switch requires an intelligent, centralized, and conflict free scheduler/arbiter algorithm.

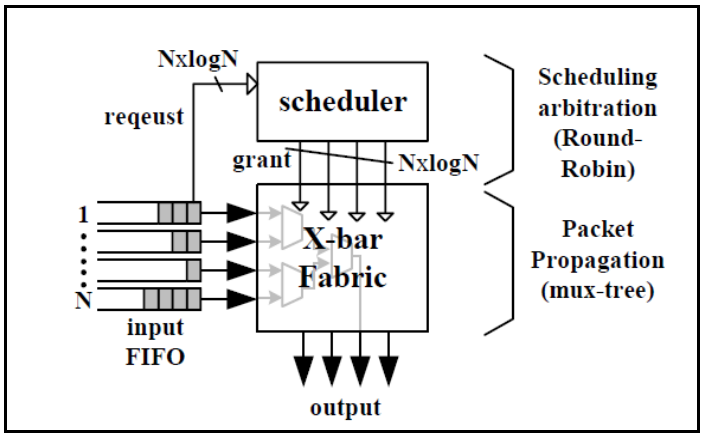

Figure 1.1: Block diagram of a conventional crossbar switch

Most crossbar schedulers are implemented in round robin fashion to prevent starvation of input ports. For example, a well-known design is applied in Stanford University's Tiny Tera prototype. The scheduler combines *i*SLIP unicast scheduling algorithm and mRRM multicast scheduling algorithm. The resulting algorithm is almost identical to the ESLIP algorithm. A brief overview of this algorithm is explained in the following paragraph and a high level block diagram of the scheduler is shown in Figure 1.2. This scheduling algorithm consists of three steps; Request, Grant, and Accept (RGA).

1. *Request:* Each unmatched input sends a request to the destination output as pointed out by the queued cell. In this step requests or packets are just transferred to grant arbiters.

2. *Grant:* Each unmatched output acknowledges one of the requests is received. Round-robin schedule starting from the highest priority element.

3. *Accept:* Each input accepts one of the received grants to establish the connection. Round-robin schedule starting from the highest priority element.

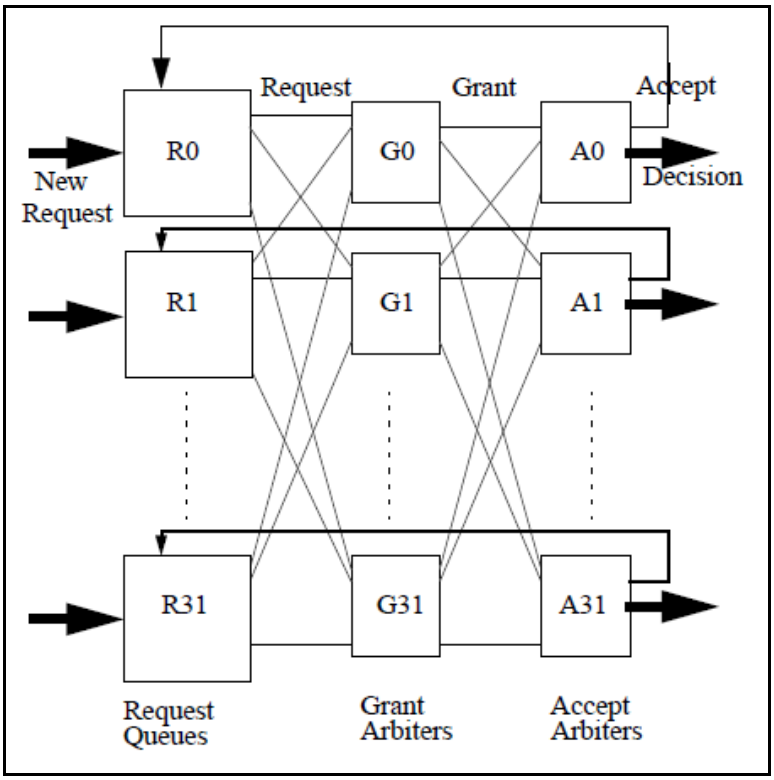

Figure 1.2: High level block diagram of the scheduler

In order to assuage the Head-of-Line (HOL) blocking problem Virtual Output Queues (VOQs) are employed as request queues. When a single FIFO input queue is used for each input port, HOL blocking problem occurs. A request/packet at the end of the queue is blocked from being destined to its corresponding output port because of port connection, and thus the entire FIFO is blocked. HOL problem is eliminated by using separate input queues for each input-output port pair.

HOL blocking problem and its solution is shown in Figure 1.3. “For this example, assume that input port 1 is granted when output port contentions occur. Each numbered rectangle in Figure 1.3 corresponds to a packet with the destination specified by the number. Thus, the packet numbered ‘1’ indicates that this packet is destined to output port 1. Without VOQs case, packet 1 in the queue at input port 0 is blocked by packet 0 located at the head of the queue, even though output port 1 is available at this point. Therefore, only packet 0 is sent to output port 1 in the current cycle. To remove HOL blocking, multiple VOQs are placed at input ports. In the VOQs case, packet 1 at VOQ (0, 1) is forwarded to output port 1 simultaneously as packet 0 at VOQ (1, 0) is delivered to output port 0. Consequently, multiple packets

can be delivered to the appropriate unique destinations by employing VOQs." (Shin 2003, pp.23)

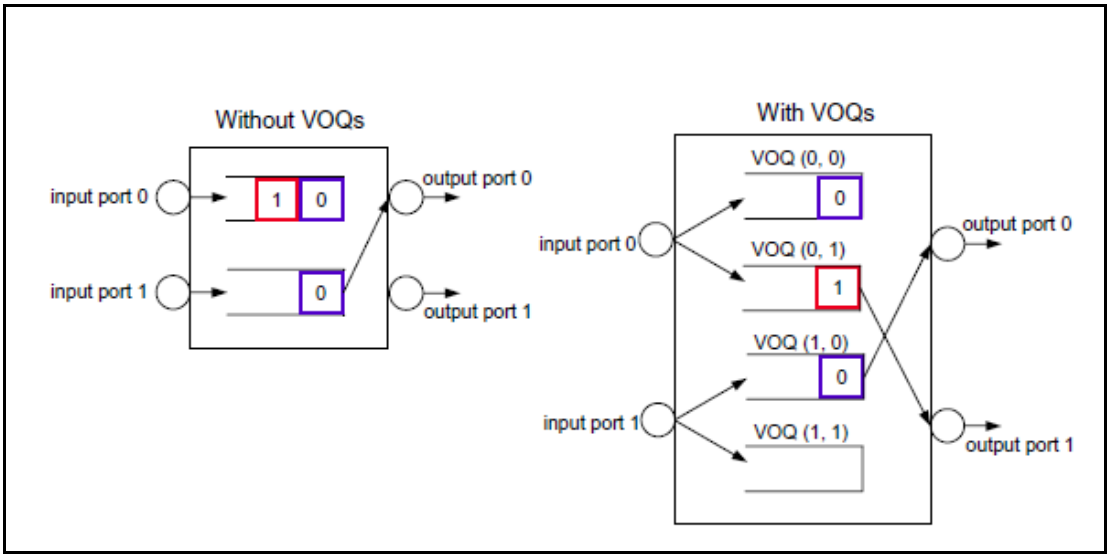

Figure 1.3: HOL blocking problem without VOQs and with VOQs

Detailed block diagram of scheduler based on RGA maximal size matching algorithm is shown in Figure 1.4 and 32X32 network switch example is shown in Figure 1.5. These figures show the scheduler blocks and how it works in a network switch/crossbar switch.

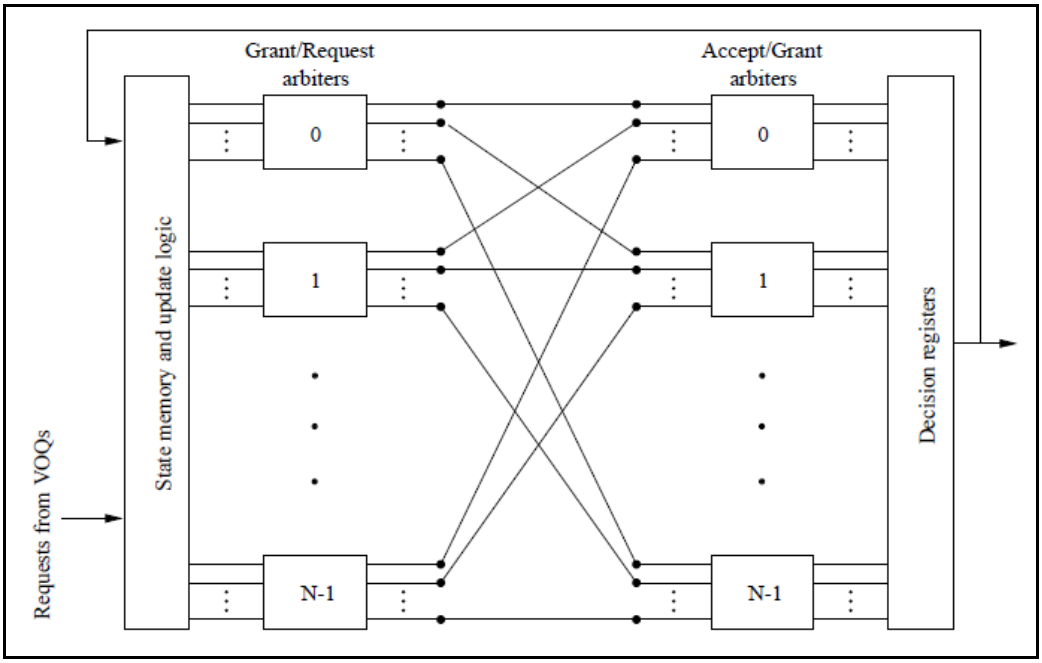

Figure 1.4: Block diagram of a scheduler/arbiter based on RGA/RG maximal size matching algorithm

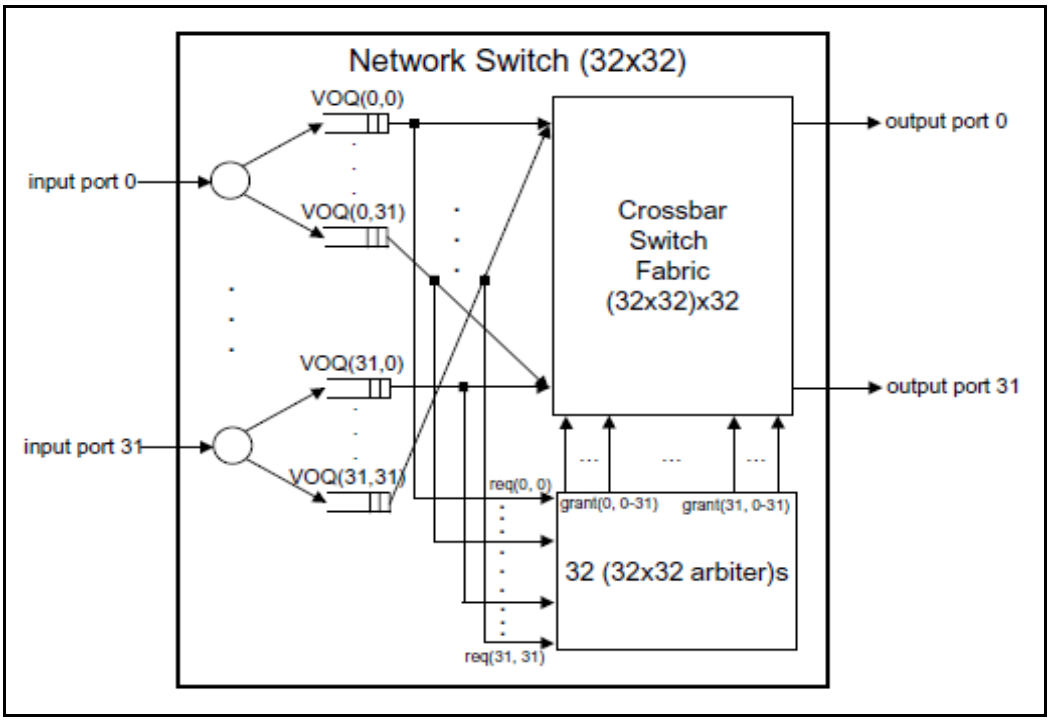

Figure 1.5: 32X32 Network switch architecture

Gupta's and McKeown's work combines the request and grant steps of the scheduler. The decision feedback information is provided by accept arbiters. This feedback information performs as a pointer. The scheduler uses this information to mask off requests from already matched inputs and outputs in successive iterations, and starts round robin schedule from the highest priority element.

A comprehensive explanation of pipelined implementation of the scheduler is explained in Gupta's and McKeown's paper. This thesis is not focused on scheduling algorithms so details of algorithms are omitted.

In this thesis, two new round-robin arbiter logic architectures are proposed by taking parallel prefix tree algorithms into consideration. Register Transfer Level (RTL) generators, Application Specific Integrated Circuit (ASIC) synthesis, and benchmarks of all RRA architectures are represented.

A brief overview of the previous work is provided in Chapter 2. Conventional RRA design and its macro blocks, most recent architectures based on conventional design, and other round robin arbitration schemes are represented. Chapter 3 introduces

hardware implementation and breakthrough of related work. In this chapter, a new macro block is embedded into the rival architectures to strengthen them. Verilog hardware description language (HDL) register transfer level (RTL) code generators of all architectures are explained in chapter 4. In Chapter 5, verification and synthesis techniques are represented in detail. Synthesis results are illustrated in Chapter 6. Chapter 7 presents some concluding remarks.

2. PREVIOUS WORK

A well-known fast crossbar scheduler design is implemented by Pankaj Gupta and Nick McKeown from Stanford University. They published their arbiter architecture in 1999. This design is called conventional round robin arbiter or conventional round robin scheduler in many research papers. In this thesis, this architecture is called Stanford Round Robin Arbiter (STA_RRA). Seven years later, Gao Xiaopeng, Zhang Zhe, and Long Xiang tried to enhance the conventional round robin scheduler design. They proposed two different architectures: PPE_Conflict and PPE_NonConflict. These two architecture names are modified as CHN_RRA_PPE_Conflict and CHN_RRA_PPE_NonConflict. This work is explained in Sections 2.2 and 2.3 in detail. A brief overview of other RRA architectures in literature is represented in Section 2.4.

2.1. STA_RRA ARCHITECTURE

Top level block diagram of STA_RRA architecture is shown in Figure 2.1. It is comprised of the following macro blocks: Simple Priority Encoder (Smpl_PE), Simple Priority Thermo Encoder (Smpl_PE_thermo), thermometer encoder (tothermo), and n to $\log_2 N$ (N2LOGN) encoder for update path. A simplified multiplexer is also used to select the appropriate Priority Encoder's (PE) grant.

Data flow of the architecture is explained as follows. When a new request is asserted from request queues, and the feedback information of priority pointer is forwarded from thermometer encoder, these two signals go into an AND gate to mask off new requests with respect to priority pointer or accepted request in the previous iteration. This new/masked request is connected to Smpl_PE_thermo block. This block is identical to Smpl_PE. Its function is to perform fixed priority encoding.

In PEs, each of the requester has a fixed priority and PE gives its grant to the active requester which has the highest priority. This is a static and unfair scheme. It always gives the shared resource's usage authority to the most powerful requester or in other words, the highest priority requester. The highest priority requester depends on

priority direction and your preference. You can change priority direction by flipping your architecture scheme horizontally. In some research work, Least Significant Bit (LSB) is selected as highest priority requester. For example, Gupta and McKeown selected LSB as the highest priority requester. However, in this thesis, Most Significant Bit (MSB) is selected as the highest priority requester for all architectures. For instance, if the request vector is declared as req[N-1:0], req[N-1] is defined as the highest priority. If req[N-1] is active, grant[N-1] gets the grant. Else, if req[N-2] is active, grant[N-2] is asserted, and so on.

Smple_PE_thermo block has two outputs which are grant (Gnt_smpl_PE_thermo) and any (anyGnt_smpl_PE_thermo). Gnt_smpl_PE_thermo is the output of priority encoding operation. anyGnt_smpl_PE_thermo bit is the OR of Smple_PE_thermo block's all inputs. This bit gives us information on whether there is any unmatched request in at least one bit position or not. On the other hand, unmasked request moves to Smpl_PE. This block performs priority encoding operation and outputs its grant (Gnt_smpl_PE) as well. anyGnt_smpl_PE_thermo is inverted and ANDed with Gnt_smpl_PE. Therefore, if an unmatched request exists from the previous iteration, anyGnt_smpl_PE_thermo disables Gnt_smpl_PE. As a matter of fact anyGnt_smpl_PE_thermo is used as a multiplexer select; Gnt_smpl_PE and Gnt_smpl_PE_thermo are this multiplexer's inputs. In some papers this operation is shown as carried on by a normal multiplexer rather than a simplified multiplexer. Pankaj and McKeown simplified this block in order to reduce the loading on the select signal by half.

Final output is obtained by OR operation of Gnt_smpl_PE_thermo and Gnt_smpl_PE_masked. Final output of this iteration should be forwarded to tothermo block for next iteration. The next iteration uses the previous iteration's final output as a priority pointer. This pointer is generated by tothermo block. This block executes thermometer encoding operation. Thermometer encoding masks all accepted requesters of previous iterations. Then, tothermo's output and a new request is ANDed to mask off the new request's accepted bit positions. All iterations are performed in this way.

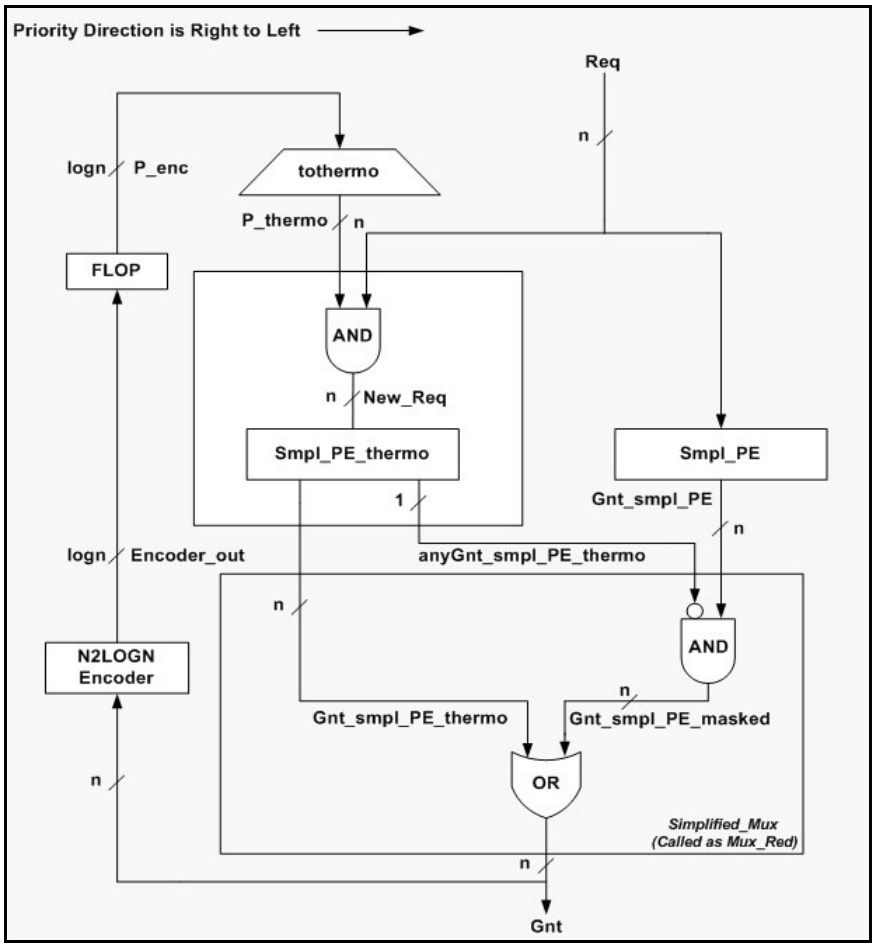

Figure 2.1: STA_RRA_N2N architecture
(Stanford Round Robin Arbiter N to N Smpl_PEs)

According to top level block diagram and signal flow description, a sample test case is shown for 8 bits STA_RRA in Table 2.1.

Table 2.1: STA_RRA_N2N data flow

Initial Iteration					
P_thermo	→	00000000	Gnt_smpl_PE	→	00100000
Request	→	00101001	Gnt_smpl_PE_masked	→	00100000
New_Req	→	00000000	Gnt	→	00100000
Gnt_smpl_PE_thermo	→	00000000	Encoder_out	→	101
anyGnt_smpl_PE_thermo	→	0	P_enc	→	101
Second Iteration					
P_thermo	→	00011111	Gnt_smpl_PE	→	10000000
Request	→	10100101	Gnt_smpl_PE_masked	→	00000000
New_Req	→	00000101	Gnt	→	00000100
Gnt_smpl_PE_thermo	→	00000100	Encoder_out	→	010
anyGnt_smpl_PE_thermo	→	1	P_enc	→	010

2.1.1. Simple Priority Encoder (Smpl_PE and Smpl_PE_thermo)

Detailed definition of simple priority encoding is described in the previous section. This encoding is a fixed and static encoding that always grants the highest priority requester. Its truth table is shown in Table 2.2.

Table 2.2: 8-bit Smple_PE truth table

in[7]	in[6]	in[5]	in[4]	in[3]	in[2]	in[1]	in[0]	out[7]	out[6]	out[5]	out[4]	out[3]	out[2]	out[1]	out[0]
1	x	x	x	x	x	x	x	1	0	0	0	0	0	0	0
0	1	x	x	x	x	x	x	0	1	0	0	0	0	0	0
0	0	1	x	x	x	x	x	0	0	1	0	0	0	0	0
0	0	0	1	x	x	x	x	0	0	0	1	0	0	0	0
0	0	0	0	1	x	x	x	0	0	0	0	1	0	0	0
0	0	0	0	0	1	x	x	0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	x	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

From table 2.2, following equation list for 8-bit Smpl_PE can be derived.

```
out[0] = ~in[7]&~in[6]&~in[5]&~in[4]&~in[3]&~in[2]&~in[1]&in[0];
out[1] = ~in[7]&~in[6]&~in[5]&~in[4]&~in[3]&~in[2]&in[1];
out[2] = ~in[7]&~in[6]&~in[5]&~in[4]&~in[3]&in[2];
out[3] = ~in[7]&~in[6]&~in[5]&~in[4]&in[3];
out[4] = ~in[7]&~in[6]&~in[5]&in[4];
out[5] = ~in[7]&~in[6]&in[5];
out[6] = ~in[7]&in[6];
out[7] = in[7];
```

This implementation has some drawbacks. When we feed this code into a synthesis tool, a ripple carry chain is inferred. This scheme produces significant timing problems and it is not area efficient. Hence, Smpl_PE should be implemented in a better way.

In order to eliminate the aforementioned drawbacks, we applied two techniques and optimized Smpl_PE. This optimization also strengthened our rivals' ASIC synthesis results. In these optimizations, binary tree algorithm/technique is used to reduce the logic level of the architecture. Also, a smart pre-computation/factoring method reduced the area dramatically. These two modifications can be explained as follows.

Firstly, all HDL files are coded with respect to binary tree fashion. This implementation model lowers the logic depth from n to $\log_2 N$. The sample HDL code for this implementation is shown below.

```
wire temp_0_7_6 = ~req[7] & ~req[6];
wire temp_0_5_4 = ~req[5] & ~req[4];
wire temp_0_7_4 = temp_0_7_6  & temp_0_5_4;

wire temp_0_3_2 = ~req[3] & ~req[2];
wire temp_0_1_0 = ~req[1] & req[0];
wire temp_0_3_0 = temp_0_3_2  & temp_0_1_0;

wire temp_0_7_0 = temp_0_7_4  & temp_0_3_0;

assign out[0] = temp_0_7_0;
```

The code snippet above is written for first grant bit's computation. This bit's computed values such as temp_0_7_6, temp_0_5_4, temp_0_7_4, etc… are used for computation of other grant bit positions. These pre-computed values are used with binary tree methodology. Eventually, design's area cost is alleviated and timing is improved. This implementation technique's HDL code snippet is shown below.

```
//FIND PRECOMPUTED VALUES and USE THEM FOR INDEX 1
wire temp_1_1_0 = temp_0_7_4 & temp_0_3_2;
wire temp_1_1_1 = req[1];

wire temp_1_2_0 = temp_1_1_0 & temp_1_1_1;

assign out[1] = temp_1_2_0;


//FIND PRECOMPUTED VALUES and USE THEM FOR INDEX 2
wire pre_temp_2_3_2 = ~req[3] & req[2];
wire temp_2_1_0 = temp_0_7_4 & pre_temp_2_3_2;

assign out[2] = temp_2_1_0;


//FIND PRECOMPUTED VALUES and USE THEM FOR INDEX 3
wire temp_3_1_0 = temp_0_7_4 & req[3];

assign out[3] = temp_3_1_0;


//FIND PRECOMPUTED VALUES and USE THEM FOR INDEX 4
wire pre_temp_4_5_4 = ~req[5] & req[4];
wire temp_4_1_0 = temp_0_7_6 & pre_temp_4_5_4;

assign out[4] = temp_4_1_0;
```

```
//FIND PRECOMPUTED VALUES and USE THEM FOR INDEX 5
wire temp_5_1_0 = temp_0_7_6 & req[5];

assign out[5] = temp_5_1_0;


//FIND PRECOMPUTED VALUES and USE THEM FOR INDEX 6
wire pre_temp_6_7_6 = ~req[7] & req[6];
wire temp_6_1_0 = pre_temp_6_7_6;

assign out[6] = temp_6_1_0;


→ FIND PRECOMPUTED VALUES and USE THEM FOR INDEX 7
wire temp_7_1_0 = req[7];

assign out[7] = temp_7_1_0;
```

2.1.2. Thermometer Encoder (tothermo)

Thermometer encoding performs $\log_2$N-bit to n-bit transformation. Its equation can be defined in this way:

$$out[index] = 1 \text{ if and only if } index < value(in) \text{ for all } 0 \le i < n \qquad (2.1)$$

This transformation works similarly to a normal thermometer operation. It takes $\log_2$N-bit input and increases thermometer level according to this input value. For example, 8-bit thermometer encoder takes 3-bit wide input vector and outputs an 8-bit wide vector. In a way, this output vector designates the level number of a thermometer. Its indicator level starts from 0 and ends up at 7. If thermometer encoder gets 101 as an input vector, it produces five piece of logic 1. In short, it transforms input 101 to 00011111 as an output. This operation is clearly shown in Figure 2.2.

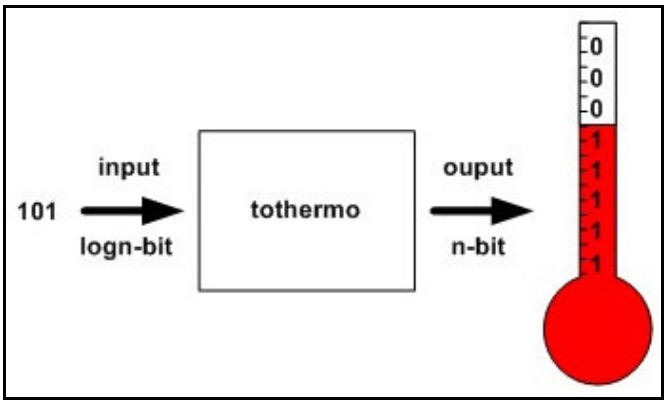

Figure 2.2: Thermometer encoding

Truth table for 3-bit to 8-bit thermometer encoding block is shown in Table 2.3.

Table 2.3: Truth table for tothermo block

in [2:0]	out[7:0]
000	00000000
001	00000001
010	00000011
011	00000111
100	00001111
101	00011111
110	00111111
111	01111111

This transformation's equations are shown below.

```
out7 = 0
out6 = in2.in1.in0
out5 = in2.in1
out4 = in2.(in1+in0)
out3 = in2
out2 = in2+in1.in0
out1 = in2+in1
out0 = in2+in1+in0
```

This algorithm is used to generate thermometer encoder code for any number of *n*. We designed STA_RRA arbiter's thermometer encoder with respect to this algorithm. Algorithm's code snippet shows that each output bit is either ORed or ANDed not greater than $\log_2 N$ bit width. This is also stated in Gupta's and McKeown's work. This algorithm's parameterized Verilog HDL code snippet is as follows.

```
parameter log_n = 3;
parameter n=(1<<log_n);

always @(thermo_in) begin
  pow2 = {(log_n){1'b0}};
  thermo_out = {(n){1'b0}};
  tmp = 0;
  pow2[0] = 1'b1;
  thermo_out[0] = 1'b0;
  for(i = 0; i < log_n; i = i+1) begin
    for(j = 0; j < pow2; j = j+1) begin
      tmp = thermo_out[j];
      thermo_out[j] = tmp | thermo_in[i];
      thermo_out[j+pow2] = tmp & thermo_in[i];
    end
    pow2 = pow2 + pow2;
  end
end
```

The purpose of thermometer encoding is to mask off new request bit positions which are previously accepted. This is also explained is Section 2.1 in detail.

2.1.3. N to LOGN Encoder (N2LOGN Encoder)

STA_RRA architecture outputs n-bit grant output. This grant output is also used to generate priority pointer for the next iteration. Typically, STA_RRA's grant vector is one-hot –only one bit position is high for each arbitration iteration. The active bit position shows us the accepted requester. Therefore, in the next iteration, priority precedence must start from this bit position.

In previous sections we mentioned that priority pointer generation is accomplished by thermometer encoder macro block. However, this block takes $\log_2$N-bit input, then executes thermometer encoding algorithm in order to output n-bit priority pointer. For this reason, there must be a logic block between the thermometer encoder block and the grant output of STA_RAA. This block's core function is to convert n bit grant vector to $\log_2$N bit vector.

Simply put, the N2LOGN encoder block outputs the active bit's index value of its n-bit one-hot input vector. In other words, the N2LOGN encoder states the accepted bit-position – or accepted requester's index to thermometer encoder. Its $\log_2$N bit output is taken by thermometer encoder. N2LOGN block's operation is shown in Figure 2.3.

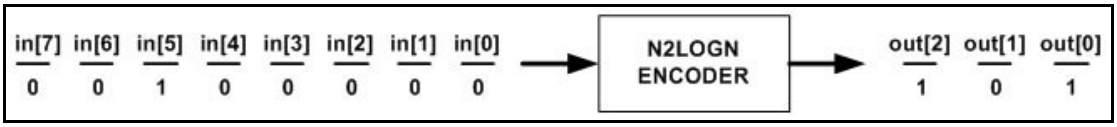

Figure 2.3: N2LOGN encoder operation

Gupta's and McKeown's work did not consist of priority pointer feedback information part. Furthermore, we did not find any explanation about this macro block's implementation from other research work. In some papers this block is stated as "encode" or "binary_enc". Therefore, we implemented this block by taking into

consideration the binary tree structure. This is the best way to reduce logic depth and to speed up timing.

Truth table of N2LOGN encoder is shown in Table 2.4.

Table 2.4: Truth table for N2LOGN Encoder

out[2]	out[1]	out[0]	input bit positions
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

According to Table 2.4 following equations are derived:

```
out[0] = in[1] | in[3] | in[5] | in[7]
out[1] = in[2] | in[3] | in[6] | in[7]
out[2] = in[4] | in[5] | in[6] | in[7]
```

In order to reduce the logic depth and timing bottleneck, these equations are coded in binary tree style. Verilog HDL code snippet below is an example of N2LOGN encoder implementation.

```
wire out_0_0 = in[1]|in[3];
wire out_0_1 = in[5]|in[7];

wire out_1_0 = in[2]|in[3];
wire out_1_1 = in[6]|in[7];

wire out_2_0 = in[4]|in[5];
wire out_2_1 = in[6]|in[7];

//output[0] assignments
wire out_0_0to1 = out_0_0 | out_0_1;

//output[1] assignments
wire out_1_0to1 = out_1_0 | out_1_1;

//output[2] assignments
wire out_2_0to1 = out_2_0 | out_2_1;

assign out[0] =  out_0_0to1;
assign out[1] =  out_1_0to1;
assign out[2] =  out_2_0to1;
```

2.2. CHN_RRA_PPE_CONFLICT

This round robin arbiter architecture is generated by three Chinese researchers: Gao Xiaopeng, Zhang Zhe, and Long Xiang. Therefore, we used CHN prefix for this architecture. They proposed two different arbitration schemes. These architectures are similar to STA_RRA architecture. One of the proposed round robin arbitration architectures is PPE_CONFLICT. Consequently, this design is called CHN_RRA_PPE_CONFLICT, as shown in Figure 2.4. In this design, the round robin arbiter architecture is divided into two paths. These are grant path and update path. Grant path is shown from request to grant. On the other hand, update path is a feedback path that forwards the priority pointer for next iteration.

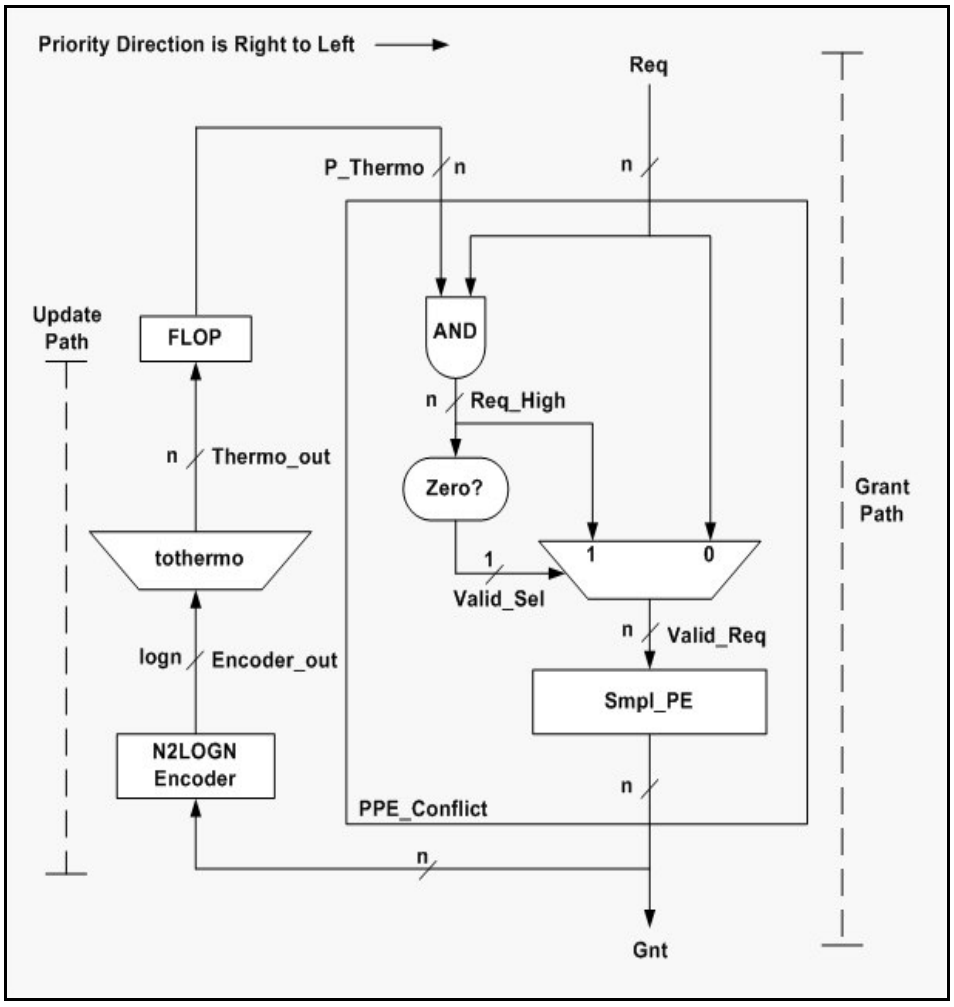

Figure 2.4: CHN_RRA_PPE_CONFLICT architecture (Chinese Round Robin Programmable Priority Encoder Conflict)

In conflicting virtual channel routers, arbiters work in an iterative way. Arbiters located at inputs and outputs perform Grant and Accept operations in a single clock cycle alternatively (in introduction section Request-Grant-Accept (RGA) pipeline is explained). Results of these two operations are dependent on each other, as shown in Figure 2.5. Grant/Update paths should be executed in two clock cycles. Therefore, the critical path in this scheme is the grant path; as the update path can be executed in parallel with respect to grant path. For this reason tothermo block is moved from grant path to update path.

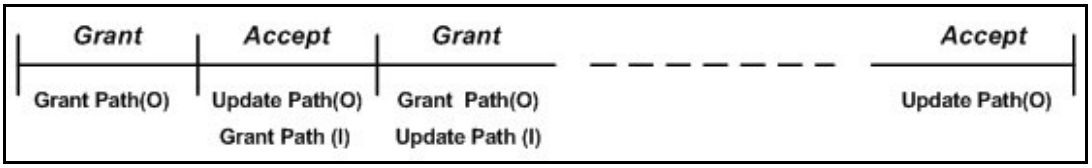

Figure 2.5: Iterative scheduling

CHN_RRA_PPE_CONFLICT architecture is divided into two timing paths as grant path and update path. Timing Path 1 is equals grant path; which is critical path for Conflicting VCR. Combo Logic 2's delay is equal to update path's delay. Timing Path 2 is comprised of both grant and update paths. In order to minimize the clock cycle, timing balance of grant and update paths is very crucial for Conflicting VCRs.

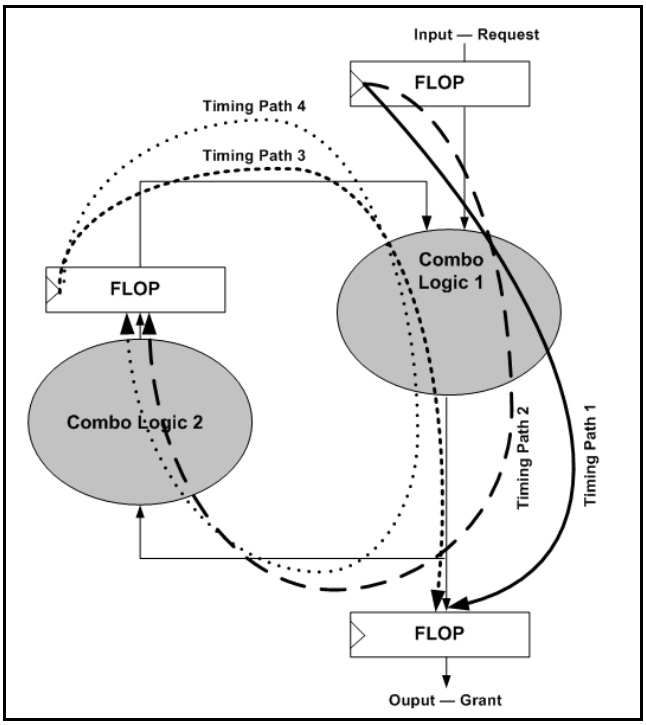

Figure 2.6: CHN_RRA_PPE_CONFLICT architecture's timing paths

One of the distinct points is the location of flip-flops. Flops are located after thermometer encoder. If their bit width is broadened, this may cause a negative effect on area. Also, flops can be located between N2LOGN encoder and thermometer encoder. Thus, sequential logic area can be lowered. On the other hand, this modification destabilizes timing balance between grant path and update path, and also maximizes the iteration cycle of arbitration.

In CHN_RRA_PPE_CONFLICT architecture Resource Sharing (RS) technique is implemented to reduce the combinational area cost. Only one Smpl_PE is used rather than two Smple_PEs. When masking operation is done from AND gate, the resulting output (Req_High) and request go to multiplexer's inputs. Then, the valid request is chosen by multiplexer select (Valid_Req). Valid_Req is just the bitwise OR of Req_High's all bits. The selected output is forwarded to Smpl_PE for final grant operation.

2.3. CHN_RRA_PPE_NONCONFLICT

This architecture is designed for non-conflicting VCRs. We called this design: CHN_RRA_PPE_NONCONFLICT, as shown in Figure 2.7. In non-conflicting VCRs, scheduling operation takes one cycle, and it is carried out by arbiters which are located at outputs. In a pipelined router, PPEs at outputs execute the arbitration cycle by cycle. Hence, priority pointer has to be ensured before next iteration start-up. PPE must succeed grant and update operations in a single clock cycle. Thus, critical path of PPE comprises both the grant and update paths.

In this architecture, grant and update operations are accomplished in parallel. Grant/Update path is also a timing path for synthesis tool. When we implement this architecture's ASIC synthesis, design compiler (DC) accepts grant/update path as a timing path. This design architecture is more realistic with respect to digital design timing concepts. Therewithal, its timing results should be better than CHN_RRA_PPE_CONFLICT architecture. On the other hand, RS technique is applied to increase the design's area efficiency. This design consists of only one Smple_PE rather than two Smpl_PEs. RS methodology is described in the previous

section in detail. The drawback of RS technique is to increase the critical path of the grant/update path. “Zero?” block is just a binary OR tree and it adds extra $\log_2 N$ stages and decelerates timing of PPE.

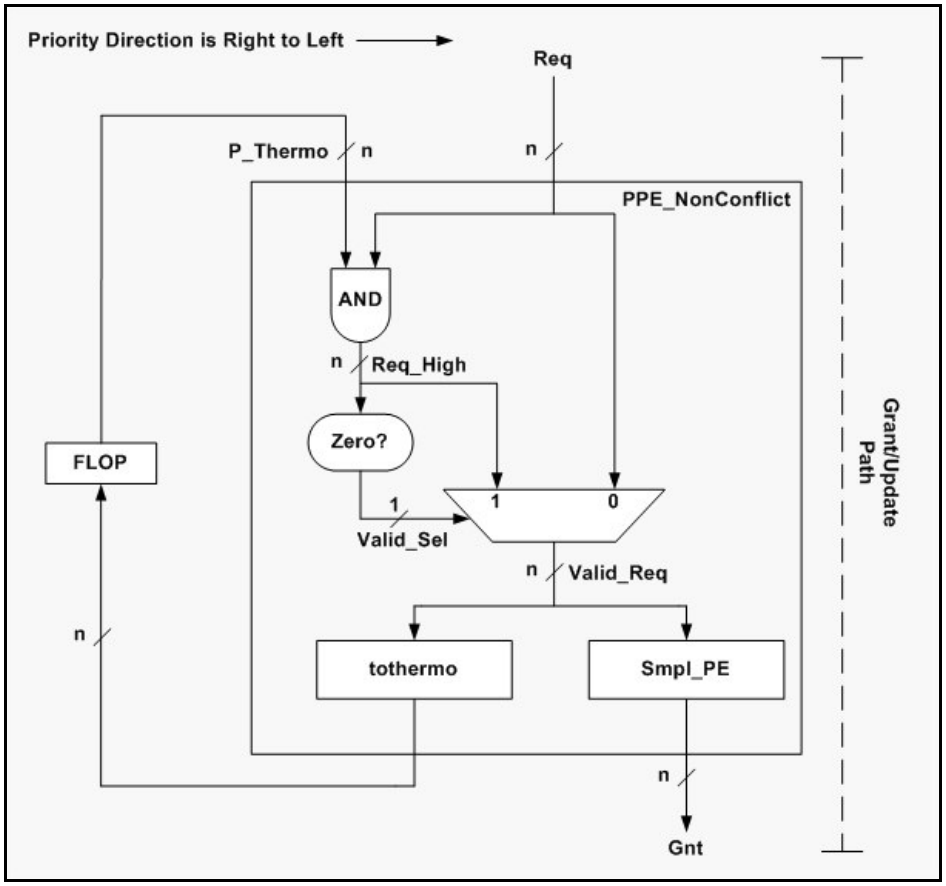

Figure 2.7: CHN_RRA_PPE_NONCONFLICT architecture (Chinese Round Robin Programmable Priority Encoder NonConflict)

Maybe the most conspicuous point in this architecture is the tothermo block because this thermometer encoder performs n-to-n conversion. In their implementation, this tothermo block combines Smple_PE, N2LONG encoder, and the tothermo block of PPE design. Actually, this implementation decreases the speed of the architecture. Hence, we implemented this block in a different way. We used one of the Parallel Prefix Tree (PPT) algorithms — Han Carlson (HC) tree. We used OR gates in HC tree’s nodes, then finally we shifted tree’s output to the right by one. One bit right shifting is just a wiring modification; it does not contain any logic element. In essence , this method is one of the core novelties for our proposed designs.

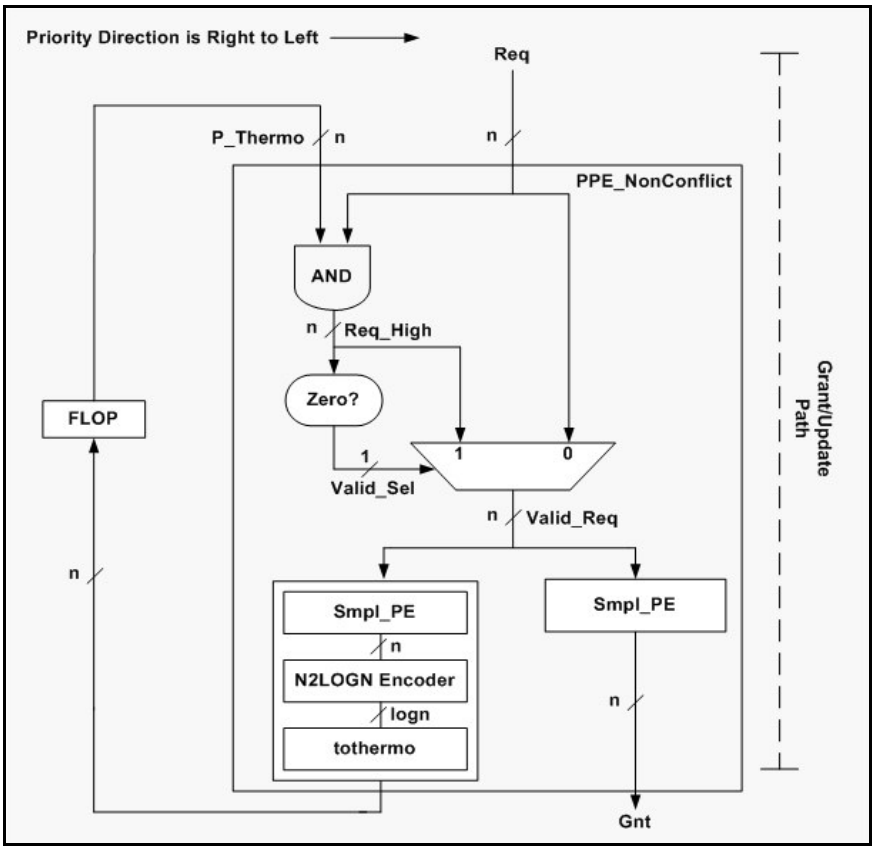

Figure 2.8: Proposed CHN_RRA_PPE_NONCONFLICT architecture

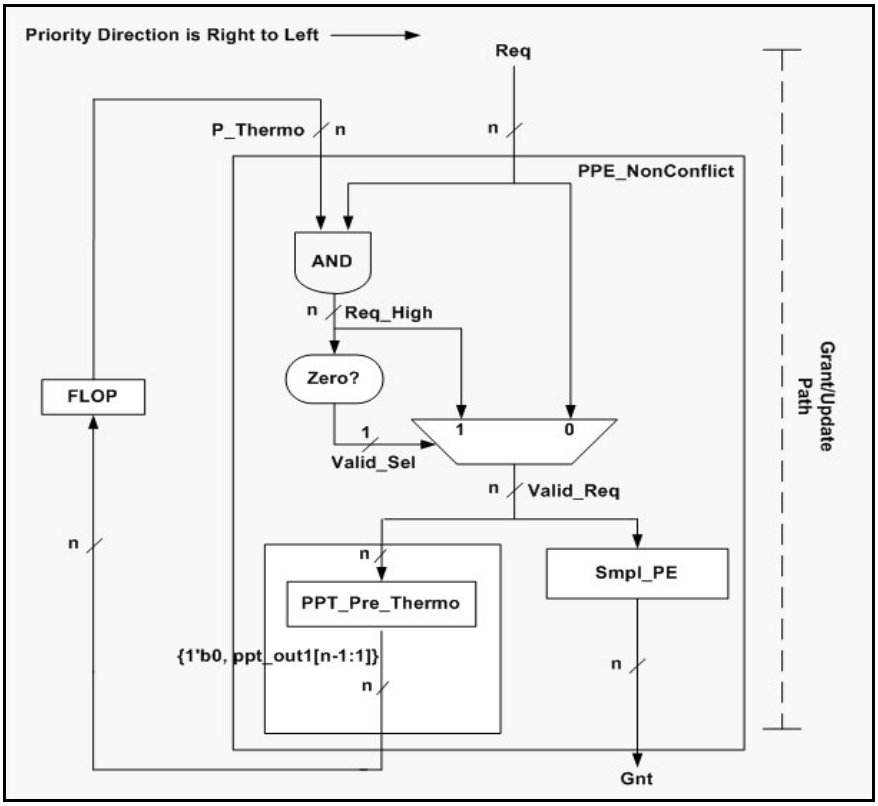

Figure 2.9: Optimized CHN_RRA_PPE_NONCONFLICT architecture

We applied this technique in this architecture to make our rival stronger. This technique is area efficient and a fast technique, which is much better than the combination of Smpl_PE — N2LOGN Encoder — tothermo blocks. This obvious improvement can be understood when we compare Figure 2.8 and Figure 2.9. You can easily see that the combination of Smpl_PE — N2LOGN Encoder — tothermo blocks significantly increases the area and slows down the speed of the RRA. This methodology is not proposing a novelty. Moreover, combinational path from request to flops' input is longer than CHN_RRA_PPE_CONFLICT architecture. However, if our PPT technique is applied rather than the Smpl_PE — N2LOGN Encoder — tothermo combination, area and timing results can be improved dramatically. Hence, in this thesis we applied our technique for this architecture to make our rival stronger. This ensures a fair benchmark. In Figure 2.9 PPT_Pre_Thermo block contains a HC tree and its nodes are comprise OR gates. Details of our novel approach and architecture will be described in later chapters.

2.4. LITERATURE SUMMARY

In literature there are many round robin arbitration algorithms and several design work are proposed. On the algorithm side, PIM, *i*SLIP, DDRM, FIRM, SSR, and PPA are the examples of the most practical scheduling algorithms. They are all iterative algorithms that approximate a maximum matching by finding a maximal size matching. They comprise of three or two steps/iterations, which were mentioned in previous sections. These steps are Request—Grant—Accept (RGA) or Request—Grant (RG). This algorithms' main goal is to ensure fairness. Also, round robin arbiter designs are carried out with respect to this criterion.

The most notable round robin architecture — STA_RAA is proposed by Gupta and McKeown which we explained in previous sections.

Some researchers have proposed modified version of STA_RRA. One of them is published by Savin C.E., McSmythurs T., and Czilli J. in 2004. They used Binary Tree Search (BTS) technique to minimize the area and maximize the speed. Their

proposed architecture has ($\log_2 N + 4$) logic level, ($n\log_2 N + 7n - 6$) combinational gate count and n-bit flop.

Other STA_RRA modifications are proposed by Gao X., Zhang Z., and Long X. in 2006 which we called CHN_RRA. Their proposed RRA architectures are depicted in former sections. Detailed synthesis results of their architecture's are given in Chapter 6.

On top of STA_RRA and its variants, different design architectures are proposed. The most recognized ones are Ping Pong Arbiter (PPA), Switch Arbiter (SA), Parallel Round Robin Arbiter (PRRA) and Improved Parallel Round Robin Arbiter (IPRRA).

PPA architecture is proposed by Chao H.J., Lam C.H., and Guo X. in 1999. PPA has $O(\log_2 N)$ level tree structure and $O(\log_2 N)$ gate delay. This round robin arbiter architecture performs the round robin arbitration rule if and only if all requests are available in each cell slot. If there are less than N request available, at that time unfairness occurs. We can examine this situation according the following example. This example is also mentioned in PPRA—IPPRA paper. Assume that N/2 + 1 input ports repeatedly serve requests in a pattern. Under this condition, one input port's request is captured by one-half of the tree. At the same time, the other half-of the tree captures the remaining input ports' requests. As a result of this situation, this round robin arbiter grants the one input port more than N/2 times more than each of the remaining N/2 input ports. This example demonstrates the unfairness of PPA design. Also, we can claim that PPA's scheduling algorithm performance is worse than iSLIP and mRRM, which are the algorithms of STA_RAA.

Other round robin arbiter design — Switch Arbiter (SA) is proposed by Shin E.S., Mooney V.J. III, and Riley G.F. in 2002. This architecture is designed with the same concept of PPA. This architecture is formed by a tree structure composed of 4x4 SA nodes. These nodes comprise of a flop, 4 PE, a 4-bit ring counter, five 4-input OR gates, and four 2-input AND gates. Some of the research work such as PPRA—IPPRA paper states and benchmarks that SA architecture is faster than the other

architectures. On the other side, architectural complexity and unfairness for non-uniformly distributed requests are its drawbacks. For example, in a 64X64 SA: if request signals (req[0] to req[31]) are asserted from the top half of the tree and only one request — req [32] is asserted from the bottom half of the tree, then req [32] is granted thirty-two times while each of 32 request signals (req[0]-req[31]) are granted only once in sixty-four consecutive cycles. This unfairness is as the same as PPA architecture's unfairness.

The most recent and remarkable round robin arbiter architectures are PRRA and IPRRA. These architectures are presented by Zheng S. Q. and Yang M. in January 2007. The proposed architectures are constructed by a recursive binary tree structure. The hardware implementation of these architectures is based on simple binary tree search algorithm. They claimed that IPRRA achieved 30.8% timing improvement and 66.9% area improvement over PPE design.

3. PROPOSED ARCHITECTURES

In this chapter, three different RRA architectures are represented. One of the RRA architectures has a similar appearance as a BOW-TIE, and it is entirely different than STA_RRA architecture. It was inspired by a Silicon Valley engineer who decided to stay anonymous. This architecture is still in its development phase; it should be enhanced in the future. The other two proposed RRA architectures are based on STA_RRA architecture. However, in these architectures we constructed a new block which executes pre-thermometer encoding and pre-priority encoding operations. Hence, critical path is shortened by using this new logic block. In the following sections details of proposed RRA architectures are explained.

3.1. PPT_RRA_RS ARCHITECTURE

The full name of this architecture is Parallel Prefix Tree Round Robin Arbiter Resource Sharing and we simply called it PPT_RRA_RS. This architecture is applied for eliminating the area cost of the RRA. In order to diminish RRA's area, we used Resource Sharing (RS) method. This architecture has one PPT_Pre_Thermo block rather than two Smple_PE's, as shown Figure 3.1. This block is constructed by Parallel Prefix Tree (PPT) topologies. In this work, we used four different PPT topologies. These are Ladner Fisher (LF), Kogge Stone (KS), Han Carlson (HC), and Brent Kung (BK) topologies. These tree structures have internal nodes for each stage and any operations which have associative property could be used in these nodes. In order to accomplish pre-thermometer encoding operation, OR gate is placed in these nodes. The resulting structure performs pre-thermometer encoding operation. If this block's output is shifted right by one unit, actual thermometer encoder output can be obtained. 1-bit right shifting is just a wiring operation and it does not contain any logic elements. At this level, we have obtained "shifted pre_thermo_out" and "pre_thermo_out" signals. These two signals are routed to Edge Detector (ED) block to generate grant output of the RRA architecture. ED block has one level logic depth; this decreases the critical path of the architecture. ED's function is to output the final grant output by using "pre_thermo_out" and "shifted pre_thermo_out" signals. In fact the shifted signal is used as a priority

pointer for next iterations. This architecture is used as the same scheduling algorithm as STA_RAA. Therefore, fairness criterion is better with respect to that of PPA and SA architectures which are proposed in literature formerly. This structure is better than our rivals with respect to the area criterion. Especially, BK topology for PPT_Pre_Thermo block provides significant area reduction. Also, this architecture has a competent speed performance. Its ASIC synthesis results and comparison against its rivals are represented in next chapters. Building blocks of PPT_RRA_RS such as OR Binary Tree (OR_BT), PPT_Pre_Thermo, and ED are explained in next sections.

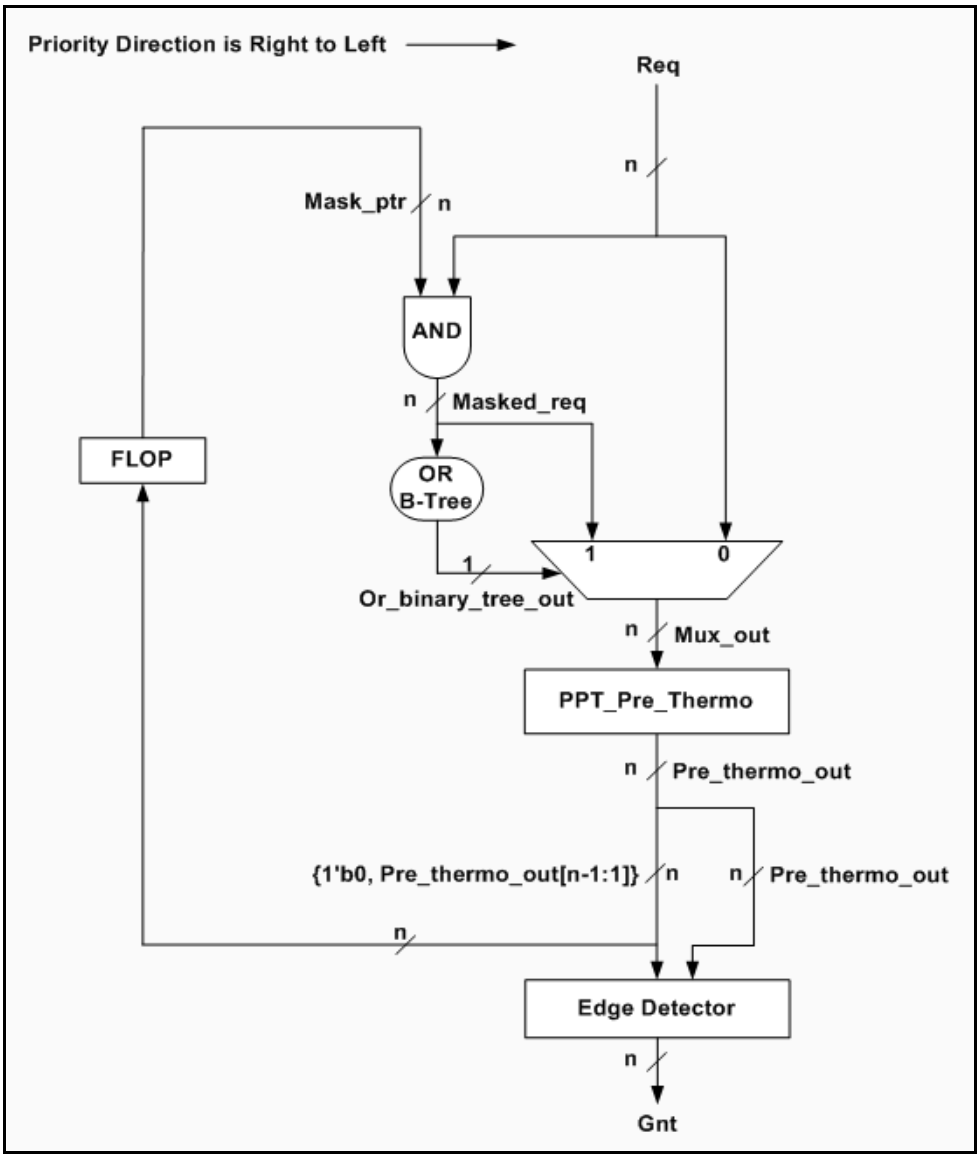

Figure 3.1: PPT_RRA_RS architecture

3.1.1. OR Binary Tree (OR_BT)

OR_BT block performs OR operation. It gets masked request (Masked_req) as an input and it outputs a select signal for multiplexer to select appropriate signal for PPT_Pre_Thermo block.

One bit right shifted version of PPT_Pre_Thermo signal is just a priority pointer (Mask_ptr) and it masks the matched requester in an ongoing iteration. In the next iteration, priority pointer is ANDed with a new request to mask matched bit positions for new request signal. This masking operation's output is called as Masked_req. It contains information on whether or not there is any unmatched requester available with respect to previous iterations or not. If all bit positions of Masked_req are zero, OR_BT block's output will be '0'. This comprises of two different meanings:

1- *New request signal does not want to grant an unmatched requester. This means that unmatched bit positions with respect to previous iterations are zero. However some matched bit positions are active/one.*

2- *New request signal does not want to grant any of the requester. This means that all bit positions of new request are zero.*

On the other hand, if a Masked_req signal has one or more active bits, at this time we can conclude that the new request signal wants to grant unmatched requesters, and OR_BT block outputs '1'.

OR_BT block is playing a key role to simplify RRA architecture's area. It controls the multiplexer and routes Masked_req or new request signals to PPT_Pre_Thermo block. For this reason, this architecture uses only one block rather that two Smple_PEs. This technique is called Resource Sharing (RS). Thus, we put RS prefix for this architecture's name.

OR_BT block is implemented by using NOR, NAND, and OR gates. This transformation is represented in Figure 3.2. For N bit input, stage number is calculated by $\log_2 N$. If stage number is odd, last stage must contain OR gate. On the other hand, if stage number is even last stage must contain NAND gate. It is shown in Figure 3.3.

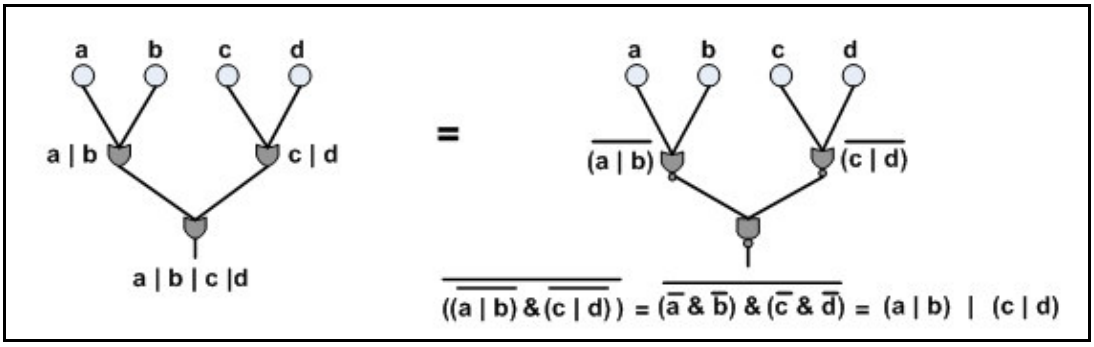

Figure 3.2: OR_BT with OR gates and OR_BT with NOR-NAND gates

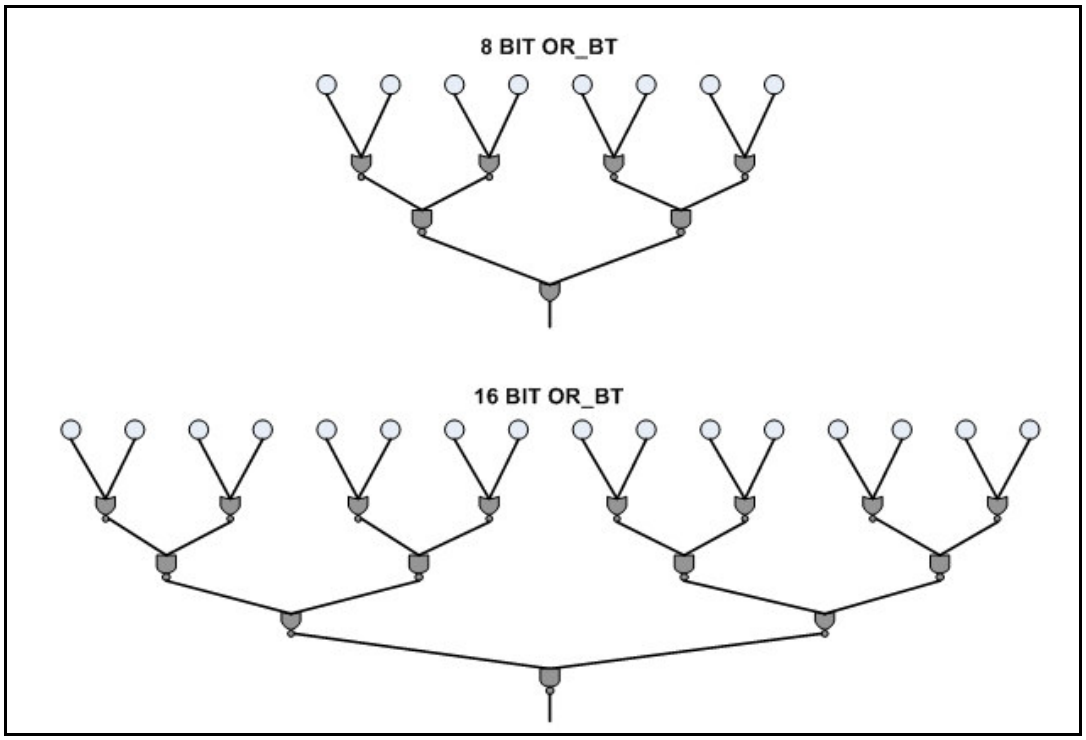

Figure 3.3: 8-bit and 16-bit NOR-NAND OR_BT

The reason for using these gates is to minimize the area cost. In Complementary Metal Oxide Semiconductor (CMOS) technology NOR and NAND gates are more area efficient than OR and AND gates. OR and AND gates have extra inverter part

that effects area efficiency negatively. This negative effect is represented in Figure 3.4 and Figure 3.5.

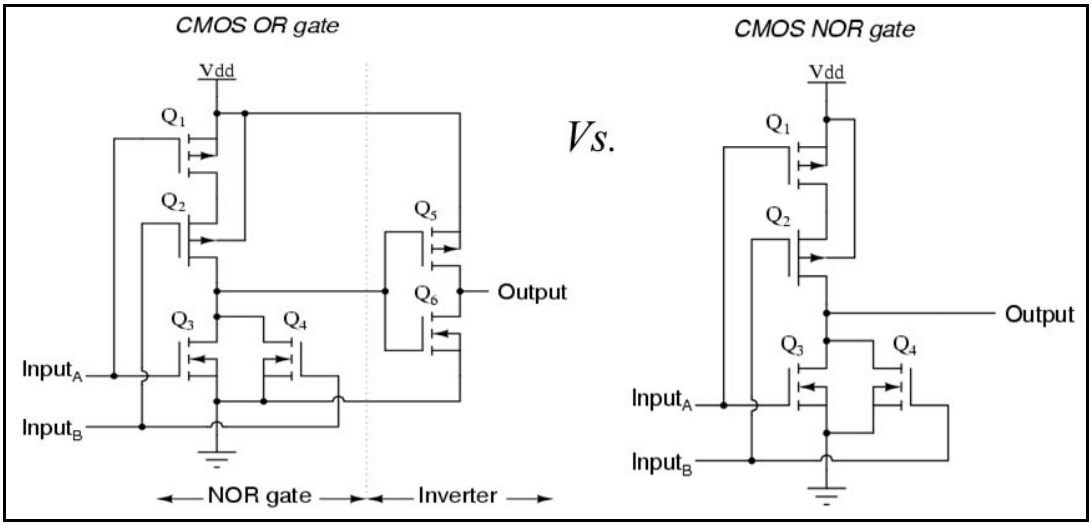

Figure 3.4: CMOS OR gate and CMOS NOR gate

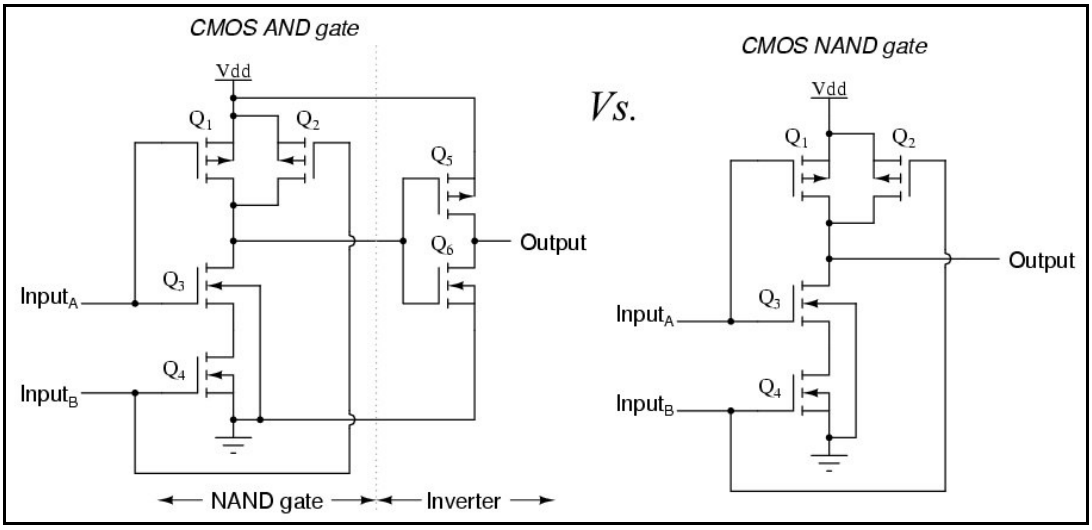

Figure 3.5: CMOS AND gate and CMOS NAND gate

3.1.2. Parallel Prefix Tree Pre Thermo Encoder (PPT_Pre_Thermo) Overview

Our proposed architecture's novelty is based on this logic block. PPT_Pre_Thermo block is constructed by PPT topologies. There are four well-known PPT topologies in literature. These are LF, KS, HC, and BK topologies. All of these topologies are implemented via our RTL generator in this work. Also, any of the PPT topology could be applied in this block. PPT topologies' taxonomy and their drawbacks among each other are stated by David L. Harris in 2003, as shown in Figure 3.6. In

this work, highest priority requester is selected as MSB bit rather than LSB bit. Hence, all topologies are flipped horizontally. Priority direction depends on our preference.

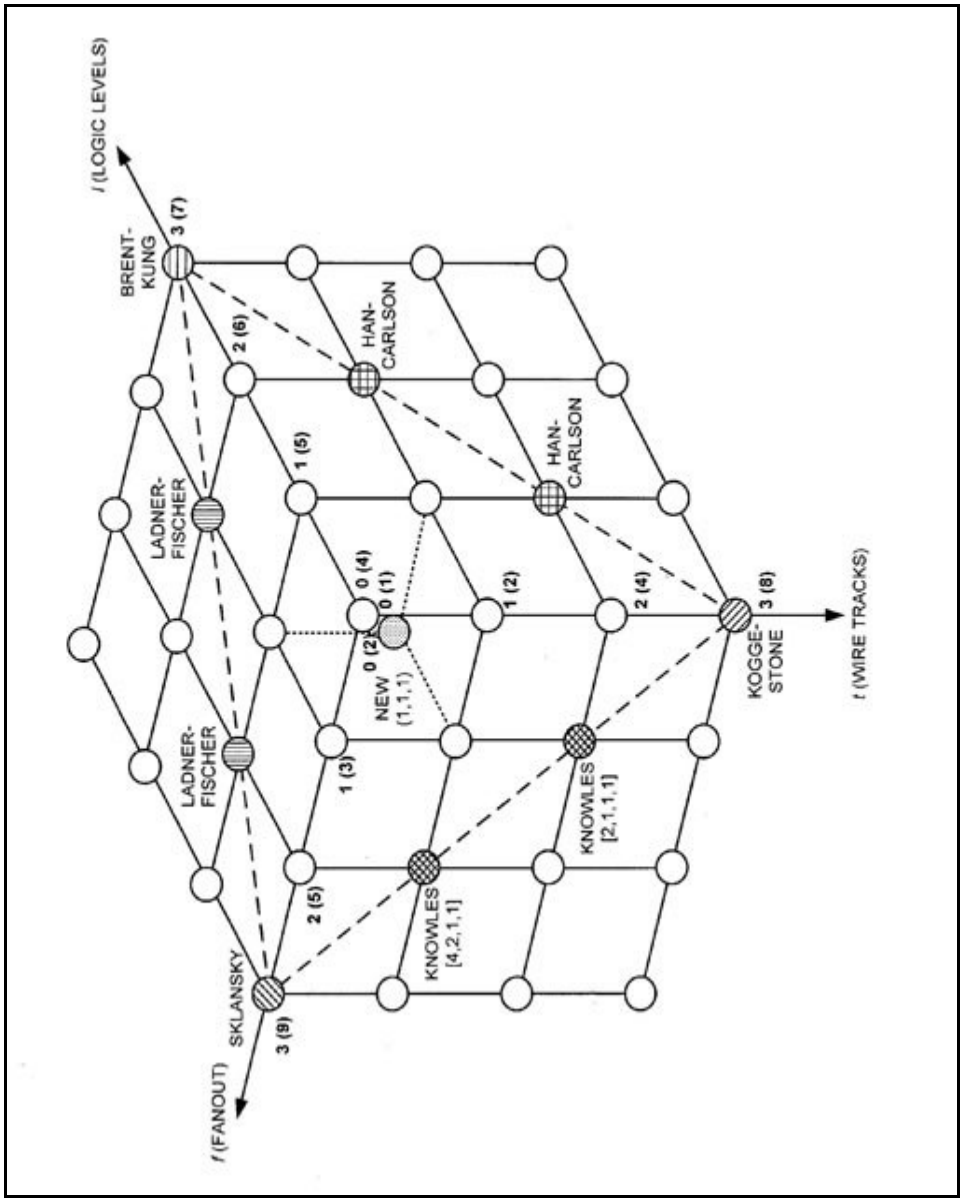

Figure 3.6: Taxonomy of PPT topologies

The function of this block is to prepare an output for thermometer encoding and priority encoding. If its output is 1-bit shifted to the right, this operation gives us thermometer encoder output (priority pointer). This shifting operation does not consist of any logic elements. It only modifies wiring connections. Moreover, if PPT_Pre_Thermo output is forwarded to ED block, ED block performs an edge detection operation to output RRA's grant. This edge detection operation has only

one level logic depth. Essentially, combination of PPT_Pre_Thermo block and ED block forms the Smpl_PE block. Also, one of the superior advantages of PPT_Pre_Thermo is any-bit computation. PPT_Pre_Thermo block's output LSB bit is equal to the OR of all its inputs. Thus, any-bit equals to this block's output's LSB. Operations of PPT_Pre_Thermo are divided into three parts: *pre-tothermo* computation, *pre-Smpl_PE* computation, and *any-bit* computation. So, it can be seen that this new macro block performs multiple operation at the same time. This makes our architecture exceptional with respect to our rivals.

In order to show this blocks functionality, a test scenario is shown in Figure 3.7. In that figure inputs and outputs of the 16-bit PPT_Pre_Thermo are illustrated.

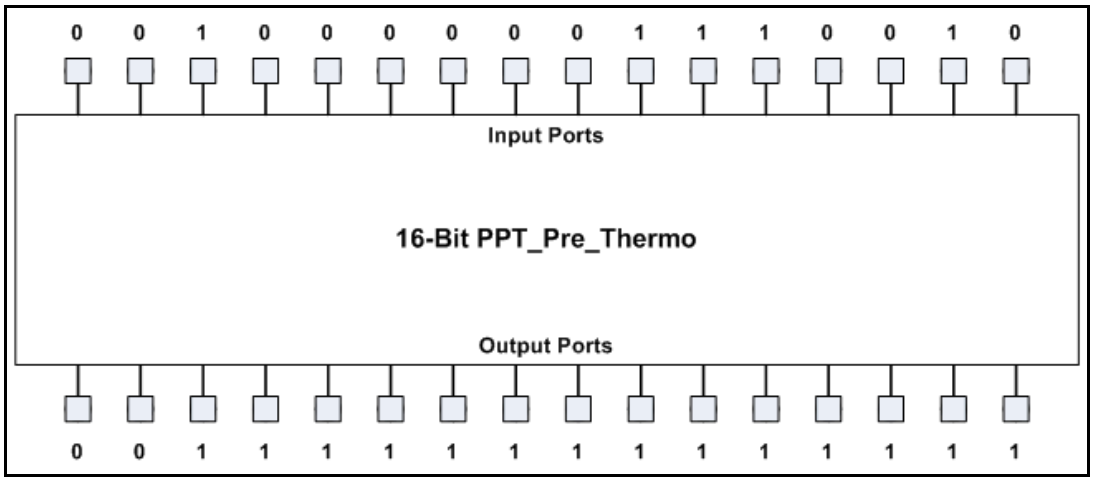

Figure 3.7: PPT_Pre_Thermo example

All PPT topologies which are implemented in this new block are explained in the following sections.

3.1.3. Brent Kung PPT_Pre_Thermo (BK_PPT_Pre_Thermo) Architecture

BK_PPT_Pre_Thermo topology is shown in Figure 3.8. In this topology's nodes, mostly NOR—NAND—INV gates are used rather than OR gate, as explained in OR_BT section. OR gates are used only in the last stages. For N-bit input, BK_PPT_Pre_Thermo architecture's logic level is equal to (2* $\log_2$N -1). On the other hand its gate count is fewer than other PPT topologies. BK_PPT_Pre_Thermo topology is a more useful tree for implementing area efficient designs.

Implementation of NOR—NAND—INV—OR method for different N-bit input brings a variation for inverter placements. Therefore, the designer has to locate inverters attentively. This topology is coded via our RTL generator, and that generator considers these kinds of variations.

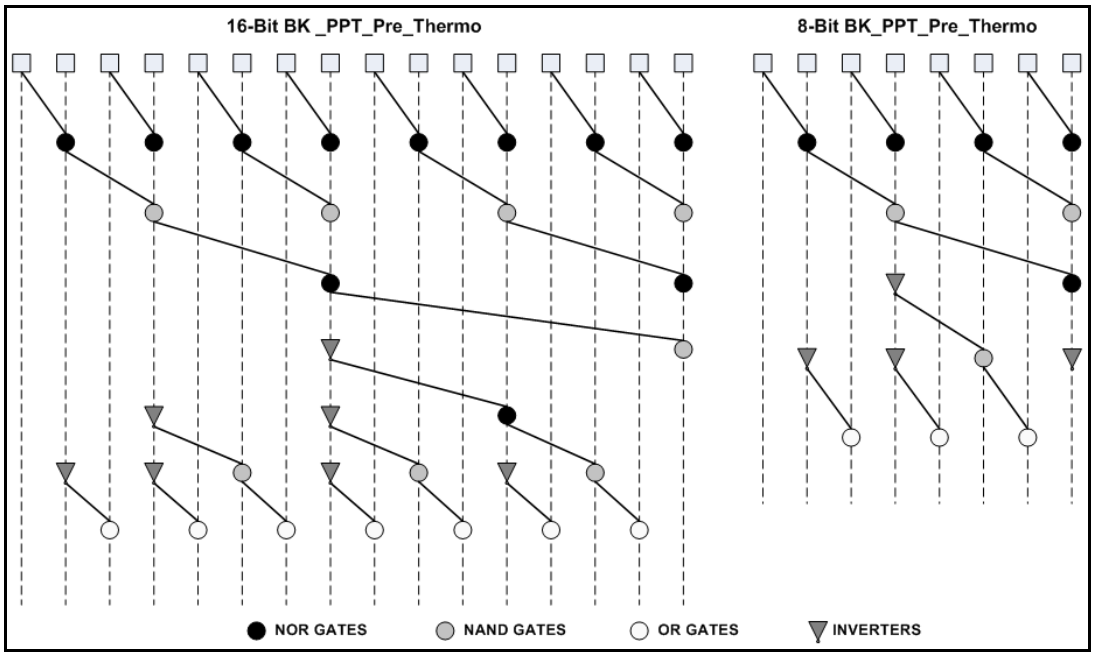

Figure 3.8: BK_PPT_Pre_Thermo structure

3.1.4. Ladner Fisher PPT_Pre_Thermo (LF_PPT_Pre_Thermo) Architecture

The number of stages is reduced by modifying the structure of the PPT graph. The minimum number of stages for PPT is $\log_2 N$. For $n=32$, the stage number is equal to $log_2 N=5$. On the other hand, higher fan-out is the major drawback for this tree, as shown in Figure 3.9. For each stage, fan-out is equal to 2^{stage_number}, and this can cause a negative effect on timing and area. In order to drive multiple cells, bigger driving cell are used. Unfortunately, those bigger cells can have increased area.

LF_PPT_Pre_Thermo is implemented with NOR—NAND—INV—OR technique too. RTL code for this tree is generated by our RTL generator. Inverter placement varies with respect to the total stage number. For example, when stage number is odd, at last stage no inverter is used. On the other hand, when stage number is even, we have to put inverters at the last stage to construct a functional PPT_Pre_Thermo block. Also, in Figure 3.10, equivalence of LF_PPT_Pre_Thermo is constructed by

OR gates, and LF_PPT_Pre_Thermo is constructed by NOR—NAND—INV—OR gates is shown.

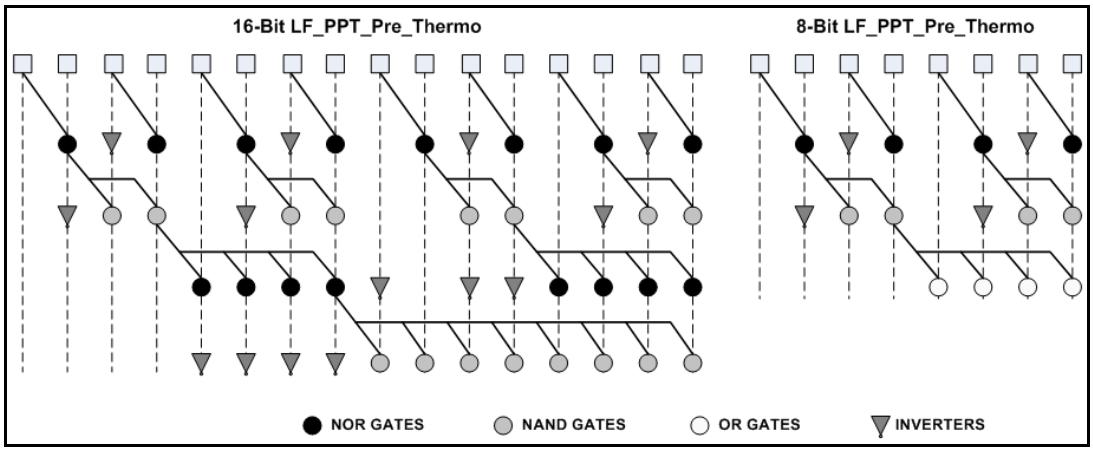

Figure 3.9: LF_PPT_Pre_Thermo structure

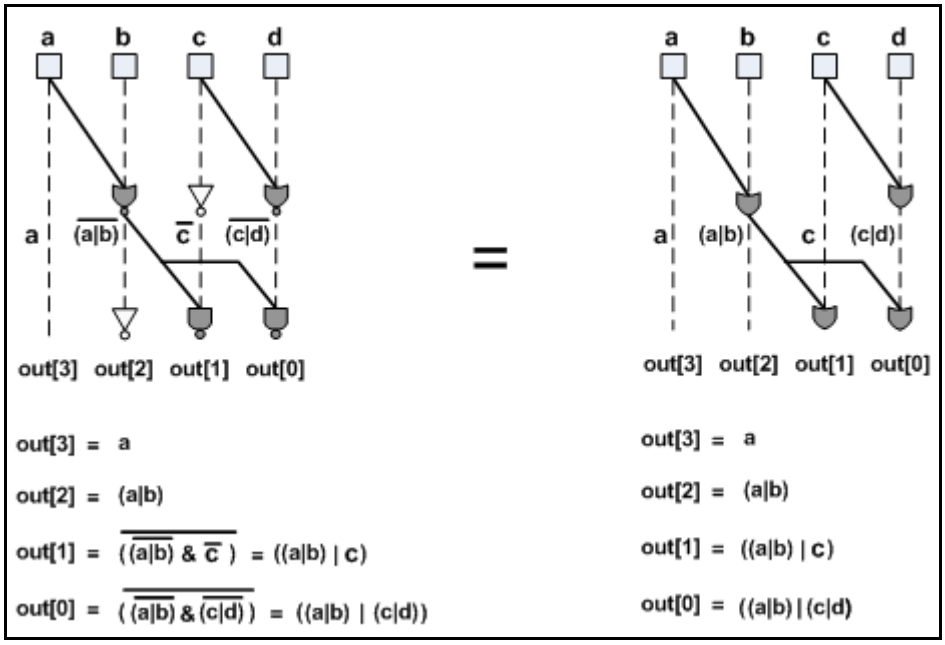

Figure 3.10: Equivalence of NOR—NAND—INV PPT and OR PPT

In our architectures, all of the topologies are constructed using NOR—NAND—INV—OR gates rather than only OR gates. This transformation ensures smaller area results. For example, in Figure 3.10 OR PPT structure has 4 OR gates and each OR gate has extra inverter part in CMOS technology. On the other hand, NOR—NAND—INV tree has 2 NOR, 2 NAND gates, and these gates do not contain any inverters. However, extra 2 inverters are added to ensure the functionality of PPT_Pre_Thermo. Even then, with this technique we save 2 inverters for 4-bit PPT

structure. For large input bit width, this cost reduction effects the RRA's area significantly.

3.1.5. Kogge Stone PPT_Pre_Thermo (KS_PPT_Pre_Thermo) Architecture

KS_PPT_Pre_Thermo uses $\log_2 N$ as similar as LF_PPT_Pre_Thermo. Also, it has low fan-in and fan-out requirement. However, this structure's main drawback is wiring tracks. It has a higher number of lateral wires with longer span which may need extra buffering, thus bringing extra delay. In order to accomplish further improvements on timing and area, it is constructed by NOR—NAND—INV—OR gates too. Its structure is shown in Figure 3.11. Its HDL code is generated via our RTL generator.

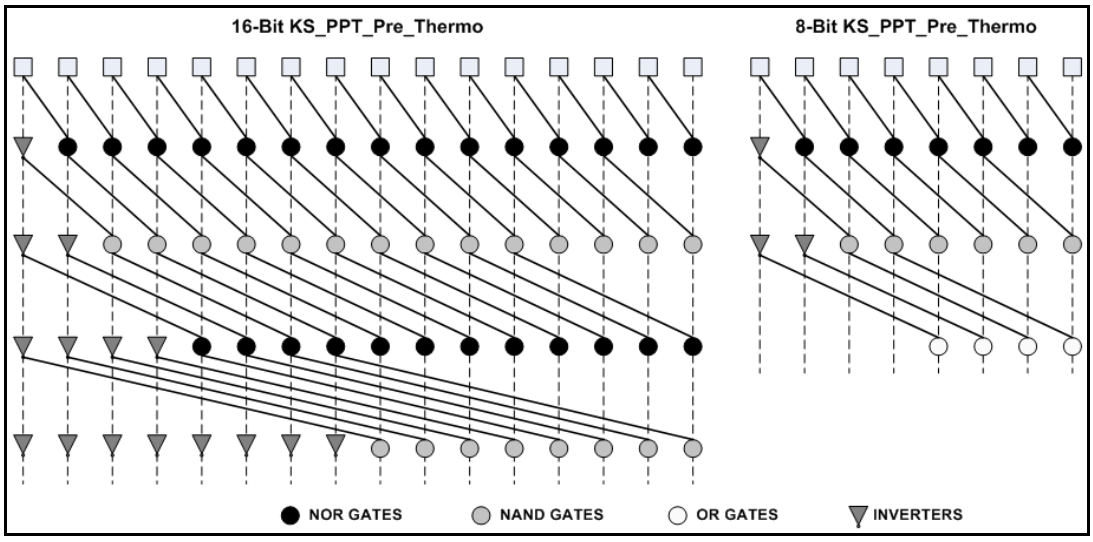

Figure 3.11: KS_PPT_Pre_Thermo structure

3.1.6. Han Carlson PPT_Pre_Thermo (HC_PPT_Pre_Thermo) Architecture

HC_PPT_Pre_Thermo structure is a hybrid structure which combines stages from KS and BK structures, as shown in Figure 3.12. For n-bit input, its stage number is equal to $(\log_2 N+1)$. Its wires have shorter span than KS, this is an advantage against KS structure. This structure is implemented by NOR—NAND—INV—OR gates and its HDL code is generated by our RTL generator. Inverter locations vary with respect to odd or even stage numbers too.

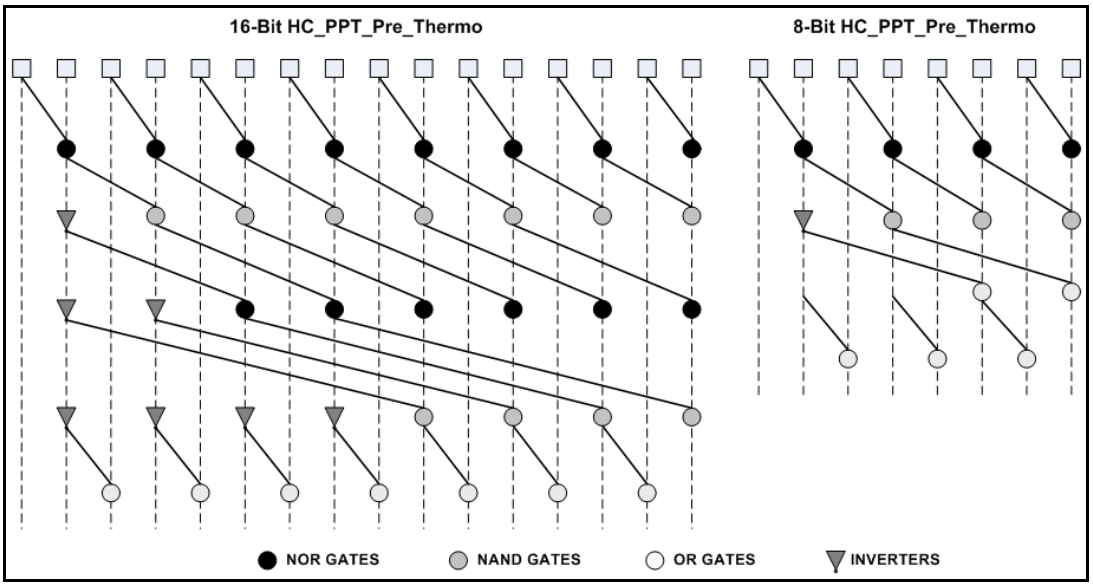

Figure 3.12: HC_PPT_Pre_Thermo structure

3.1.7. Edge Detector (ED)

Edge detector is used to compute RRA's grant, as shown in Figure 3.13. It has two inputs; PPT_Pre_thermo's output and its 1-bit right shifted version. It has one output which is grant of RRA. ED block is the last block before RRA's output and it is bounded with PPT_Pre_Thermo block. It has one level logic depth and this extremely decreases the critical path of the RRA. Combination of PPT_Pre_Thermo and ED performs simple priority encoding operation. Smple_PE operation's logic level almost depends on PPT_Pre_Thermo block. ED has a negligible effect on this operation's logic level.

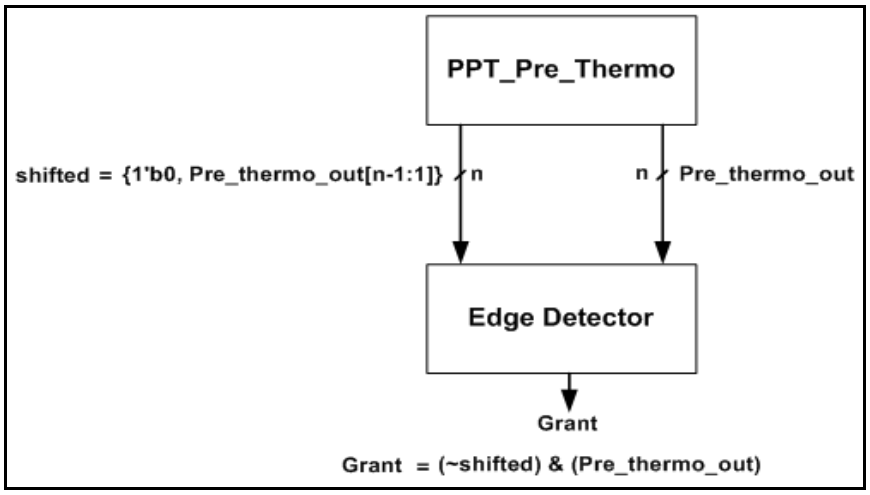

Figure 3.13: Edge detector's top level block diagram

Its main function is capturing 0 → 1 transition. Examples of this functionality are shown in Figure 3.14.

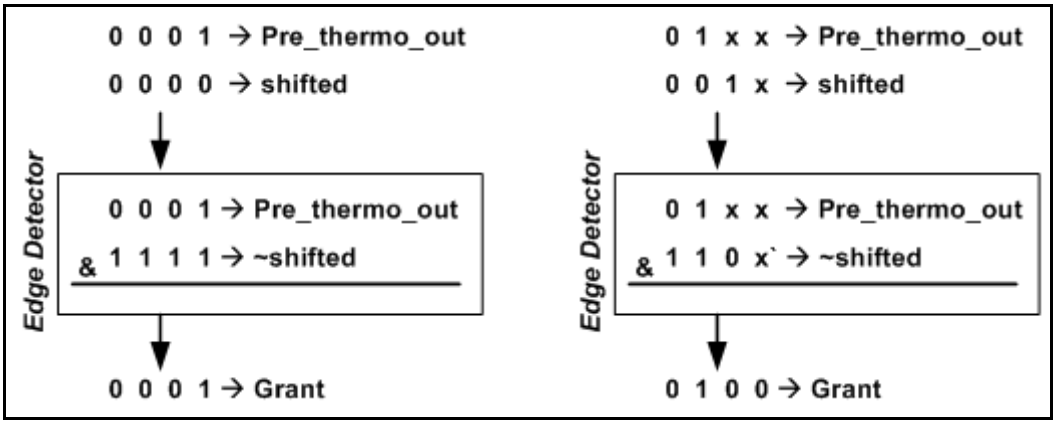

Figure 3.14: Edge detection examples

First, this architecture is implemented with AND—INV gates, as shown in Figure 3.15.

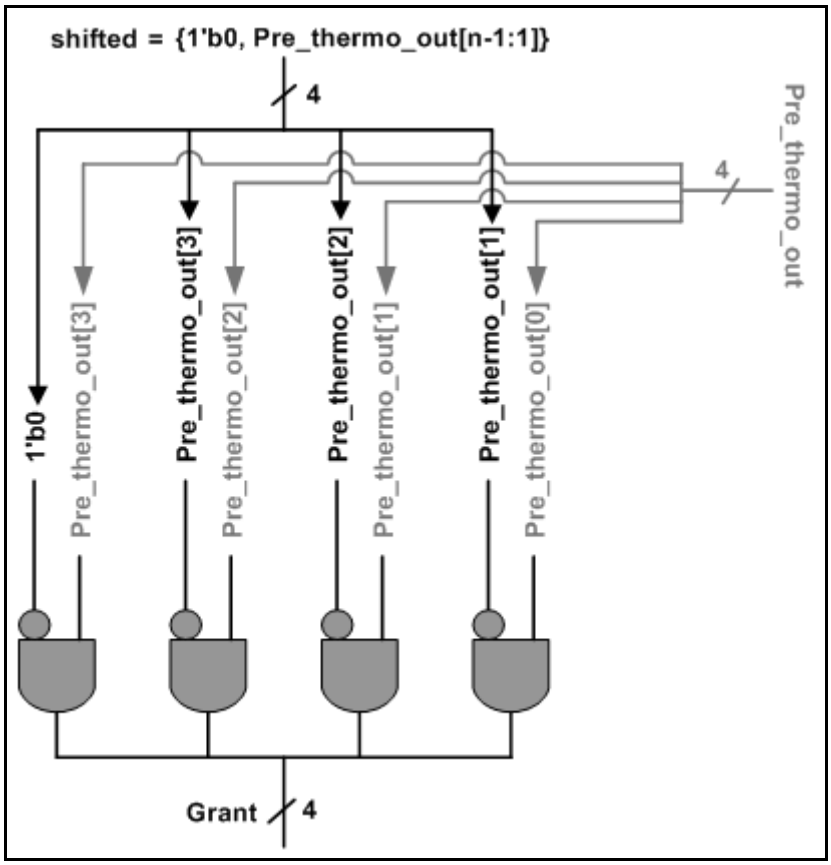

Figure 3.15: Edge detection architecture is constructed by AND—INV gates

Then, we optimized this structure with NOR—INV gates. Thus, this block's area efficiency is improved. This optimization and the new architecture are shown in Figures 3.16 and 3.17, respectively.

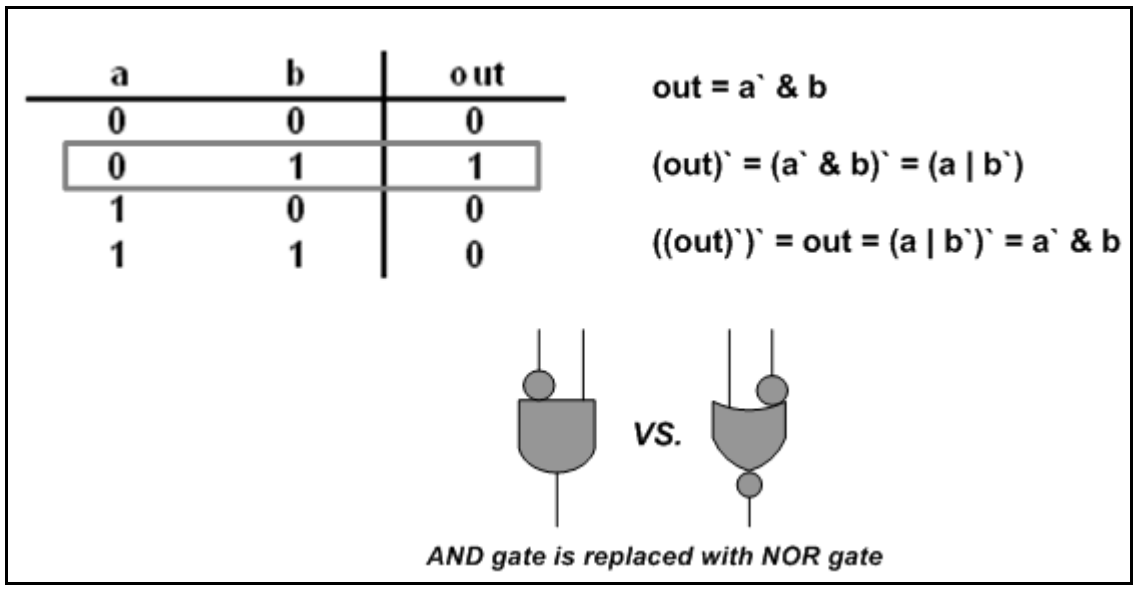

Figure 3.16: Edge detector optimization

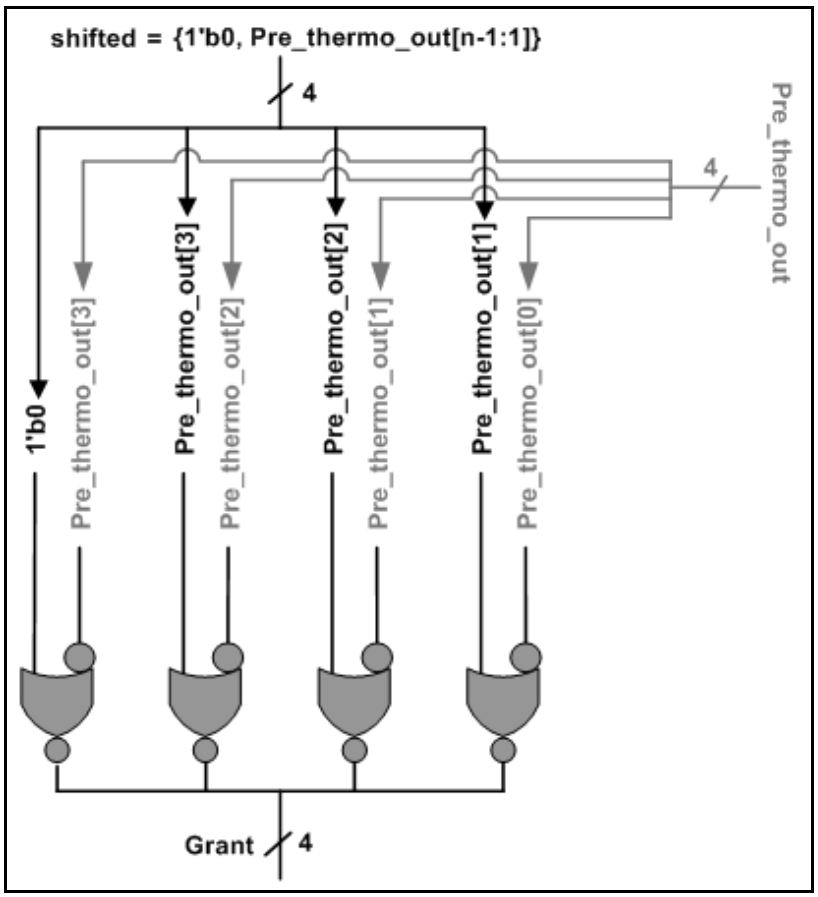

Figure 3.17: Optimized edge detector

3.2. PPT_RRA_BT ARCHITECTURE

This architecture is proposed to achieve maximum speed. Thus, we called this architecture Parallel Prefix Tree Round Robin Arbiter Best Timing (PPT_RRA_BT), as shown in Figure 3.18. When we compare PPT_RRA_BT architecture with PPT_RRA_RS architecture, we see that PPT_RRA_BT architecture uses two PPT_Pre_Thermo blocks. However, it does not consist of OR_BT block. In PPT_RRA_RS architecture, OR_BT is in critical path so this block adds extra delay to input—output path. This problem is eliminated by using two PPT_Pre_Thermo blocks. Therefore, logic level is decreased by factor in log2N. On the other hand, OR_BT gate count is smaller than most of the PPT_Pre_Thermo architecture, so this effects area negatively in some cases. This architecture is very similar to STA_RRA but its PPT_Pre_Thermo block performs pre-thermometer encoding and priority encoding operations to remove negative effects of Smpl_PE, tothermo, and N2LOGN encoder blocks.

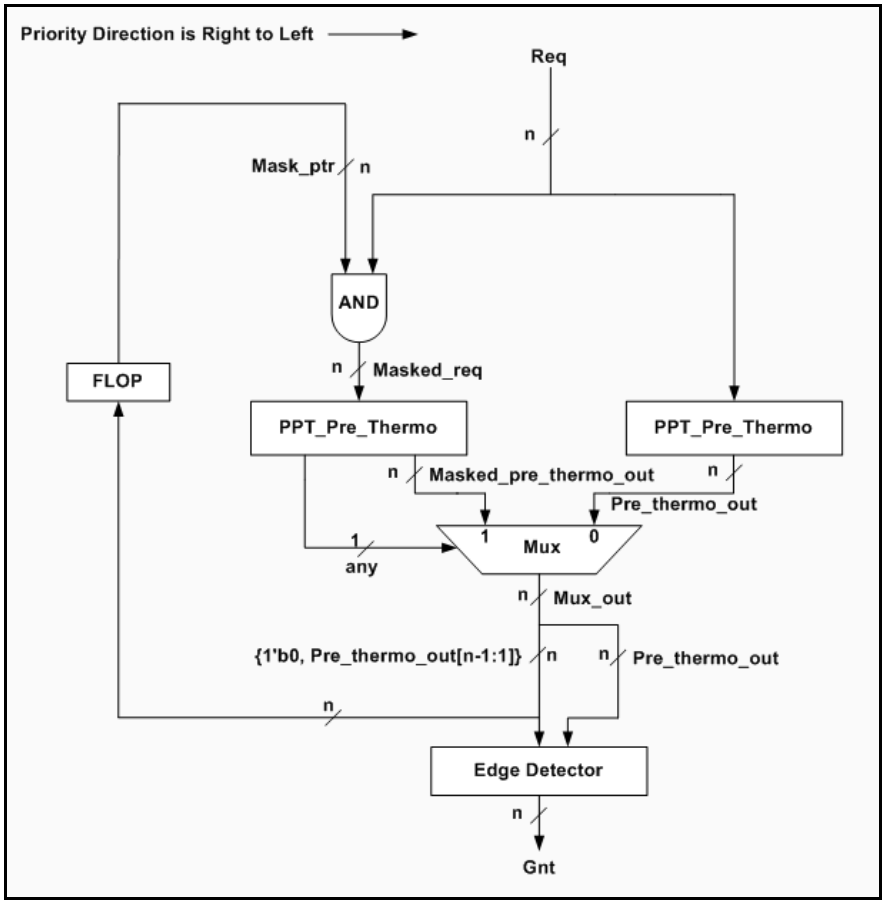

Figure 3.18: PPT_RRA_BT architecture

In Figure 3.19, we replaced the multiplexer with a simplified multiplexer which is used in STA_RRA too. This modification lessens the area cost. This architecture's logic level is shown as follows.

AND Gate → 1 level

PPT_Pre_Thermo → BK:(2* $\log_2 N$ -1) — KS:$\log_2 N$ — LF:$\log_2 N$ — HC:($\log_2 N$ +1)

Simplified Mux → 2 levels

ED → 1 level

Total Logic Level → 4 + Logic level of PPT_Pre_Thermo Block

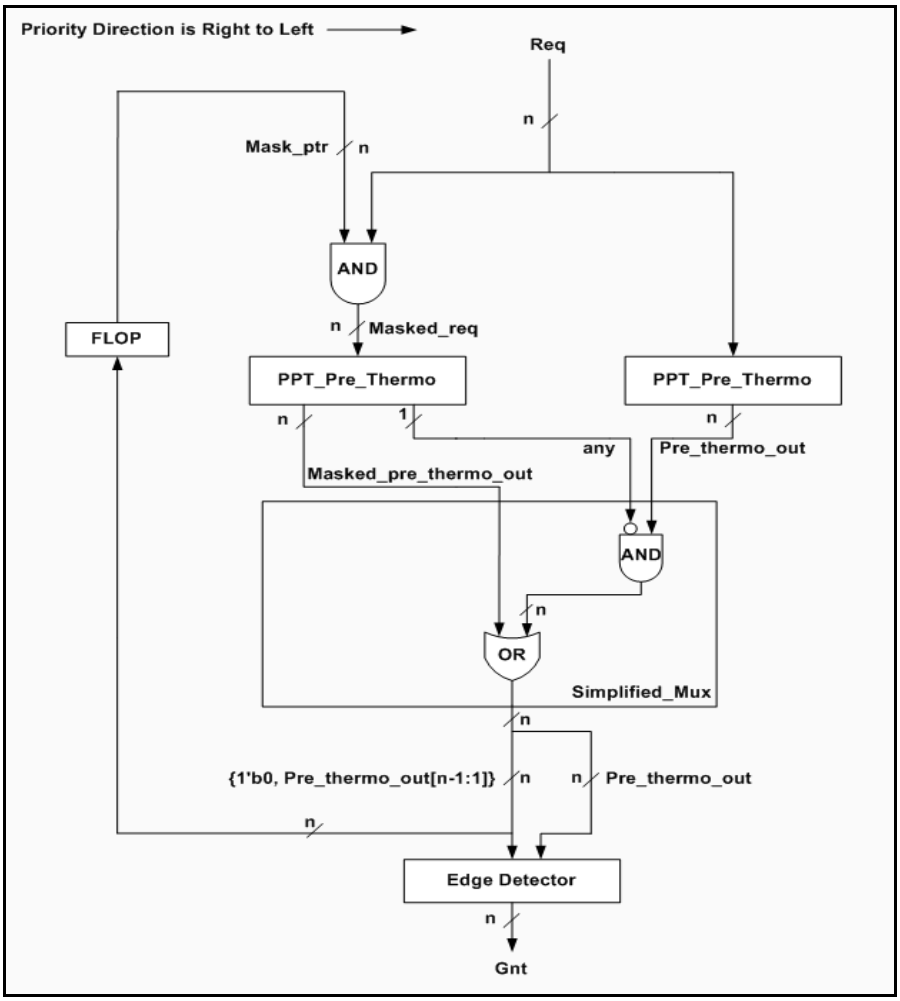

Figure 3.19: PPT_RRA_BT architecture with simplified multiplexer

Details of the sub blocks such as PPT_PRE_Thermo and ED are described in Section 3.1.

3.3. BOW-TIE ARCHITECTURE

This architecture is generated with respect to a Silicon Valley engineer's idea who wants to stay anonymous. As you can see in Figure 3.20, this is comprised of two macro blocks. These are Round Robin Arbiter Macro Block (RR BLOCK) and Grant Unit Macro Block (GUNIT BLOCK).

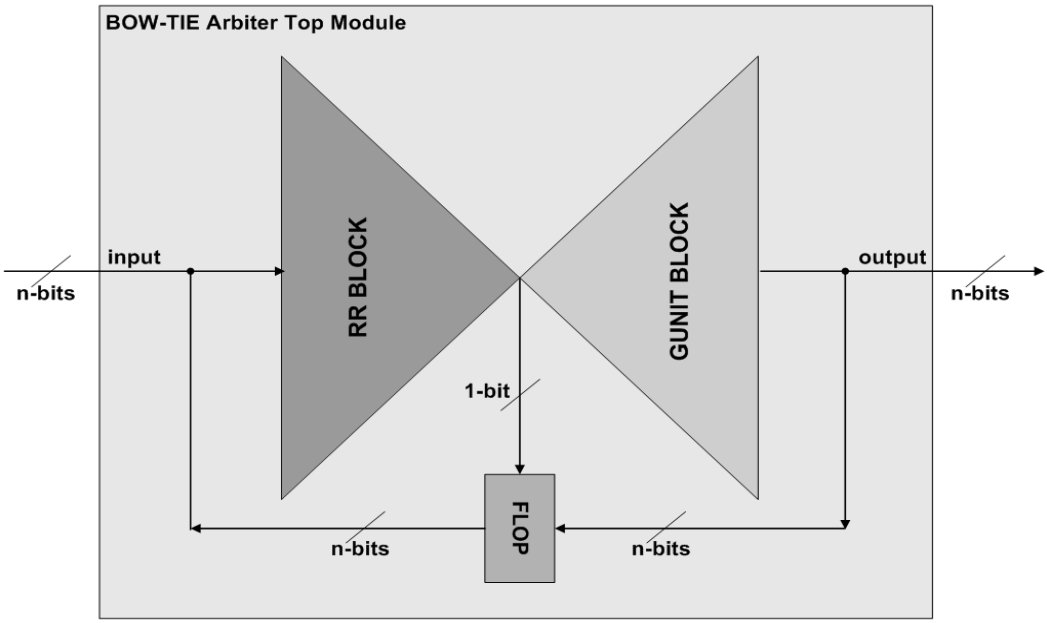

Figure 3.20: BOW-TIE_RRA high level architecture

RR blocks perform two bit round robin arbitration operation. It has six inputs and five outputs. Its inputs are request (req1, req0), pointer (ptr1, ptr0), and high-low (HL1, HL0). New incoming packets/requests are represented as request. Previous iteration's grant is called a pointer, and high-low represents the location of the pointer. If HL is one, it means that the pointer is located at the top of the request. Otherwise, its value is equal to zero and the pointer is located under the request.

RR blocks' outputs are divided into two categories; first category is previous grants (pgnt1, pgnt 0) and second category is request—pointer—high-low (req-ptr-HL). Previous grants are forwarded directly to GUNIT blocks for final output/grant generation. On the other side, req-ptr-HL outputs are routed to next level RR blocks and these RR blocks are used them as input. These connections are shown in Figure 3.21.

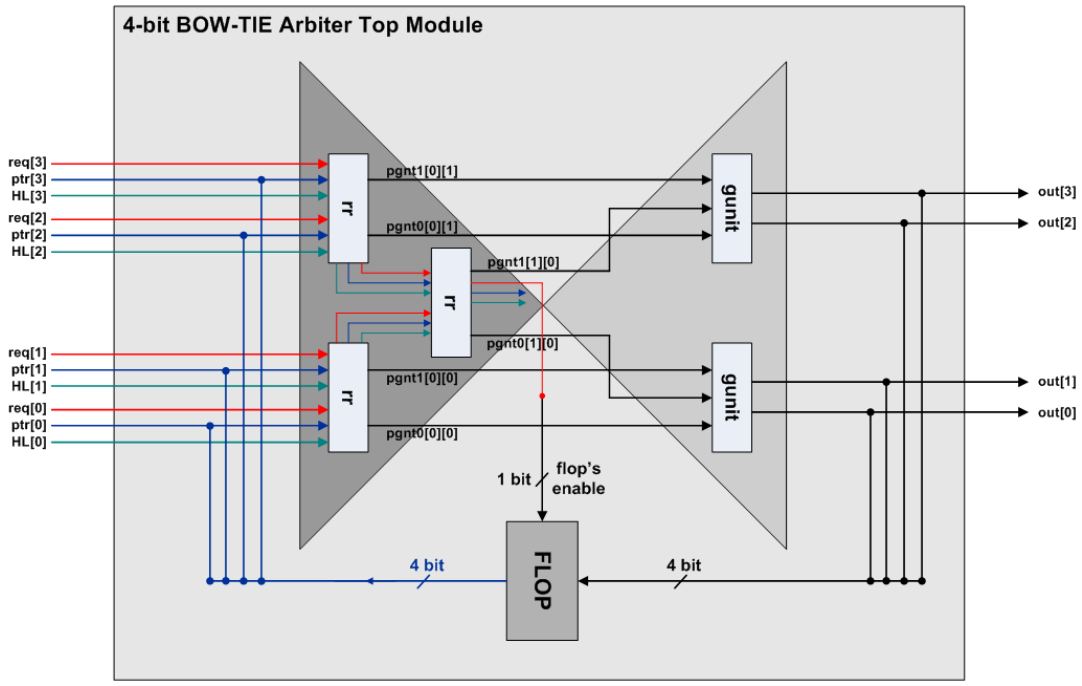

Figure 3.21: BOW-TIE_RRA's building blocks

Last level's RR block's "output req" signal is as the same as any of the STA_RRA. It enables or disables the flop with respect to new incoming request. If there is no request asserted for a new iteration, then last level's RR block's "output req" signal is equal to zero. Thus, the flop will be disabled and next iteration priority pointer will be equal to zero. This prevents the conflicts when no request is asserted.

Table 3.1: RR block's truth table

	in1			in0			out_int		ptr/req		
	left input			right input			pgnts		next HL/ptr/req		
	req1	ptr1	HL1	req0	ptr0	HL0	pgnt1	pgnt0	HL	ptr	req
1	0	x	x	0	x	x	0	0	x	x	0
2	0	0	x	1	0	x	0	1	x	0	1
3	0	0	x	1	1	0/1	0	1	0/1	1	1
4	0	1	x	1	x	x	0	1	1	1	1
5	1	0	x	0	0	x	1	0	x	0	1
6	1	0	x	0	1	x	1	0	0	1	1
7	1	0	x	1	0	x	1	0	x	0	1
8	1	0	x	1	1	0	1	0	0	1	1
9	1	0	x	1	1	1	0	1	1	1	1
10	1	1	0	0	x	x	1	0	0	1	1
11	1	1	0	1	x	x	0	1	1	1	1
12	1	1	1	x	x	x	1	0	1	1	1

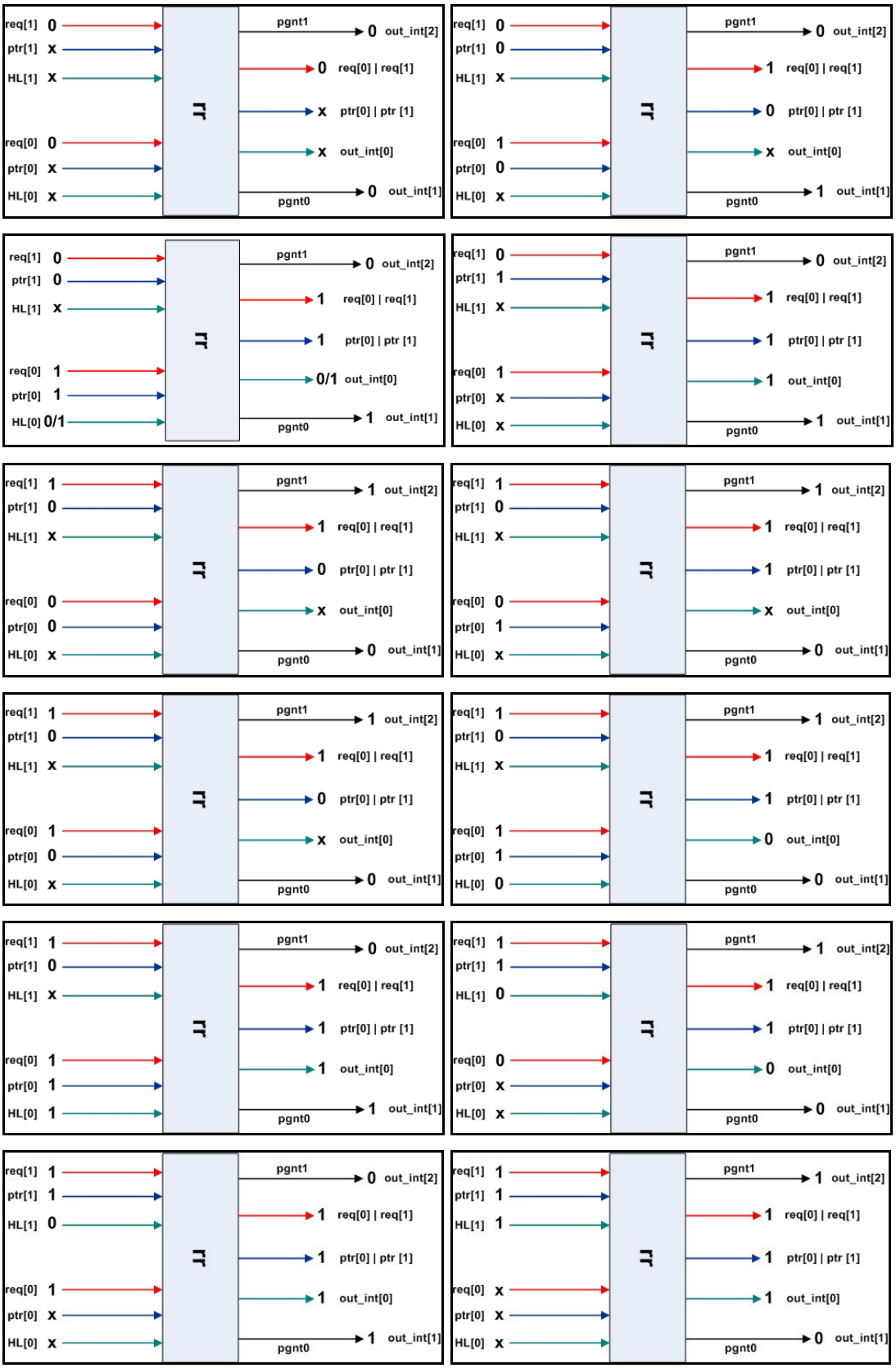

Figure 3.22: RR block's all possible states

The core blocks of this architecture are the RR blocks. Actually, they are working as small RRAs because they perform two-bit round robin arbitration and they produce any, pointer, and high-low outputs for next level RRA blocks. Their truth table is shown in Table 3.1. Also, all possible input—output combinations of RR blocks are shown in Figure 3.22. Essentially, RR blocks are slightly complex blocks, this effects this architecture's timing and especially area performance in a negative manner. If these blocks are designed in a simple way, this architecture's performance could be increased immensely.

GUNIT blocks are very simple blocks, as shown in Figure 3.23. It gets three inputs and produces the final outputs of the BOW-TIE_RRA. All RR blocks' pgnt signals are forwarded to GUNIT blocks and they are ANDed to output the final grant output. This routing is clearly seen in Figure 3.21.

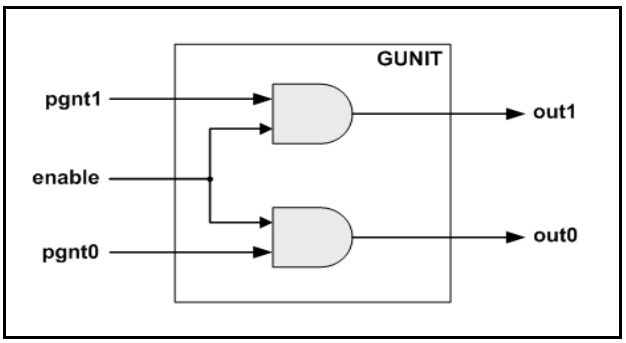

Figure 3.23: GUNIT block

3.4. ENHANCEMENTS ON PREVIOUS WORK

In order to enhance STA_RRA and CHN_RRA architectures we replaced Optimized Smple_PE blocks with Han Carlson PPT_Pre_Thermo (HC_PPT_Pre_Thermo)–ED combination. In this manner, we can see the effects of HC_PPT_Pre_thermo–ED on RRAs performance.

Furthermore, we tried to use N2LOGN Smple_PE encoder to eliminate the N2LOGN encoder block for STA_RRA, as shown Figure 3.24. This new architecture is called STA_RAA_N2LOGN. All of these architectures are described in next sections.

3.4.1. STA_RRA_N2LOGN Architecture

This architecture's Smple_PE and Smpl_PE_Thermo blocks take n-bit inputs and output $\log_2 N$ bit outputs as shown in Figure 3.24. Therefore, there is no need to use N2LOGN Encoder block before tothermo block. Thus, area efficiency and timing performance can be increased with respect to well-known STA_RRA_N2N architecture.

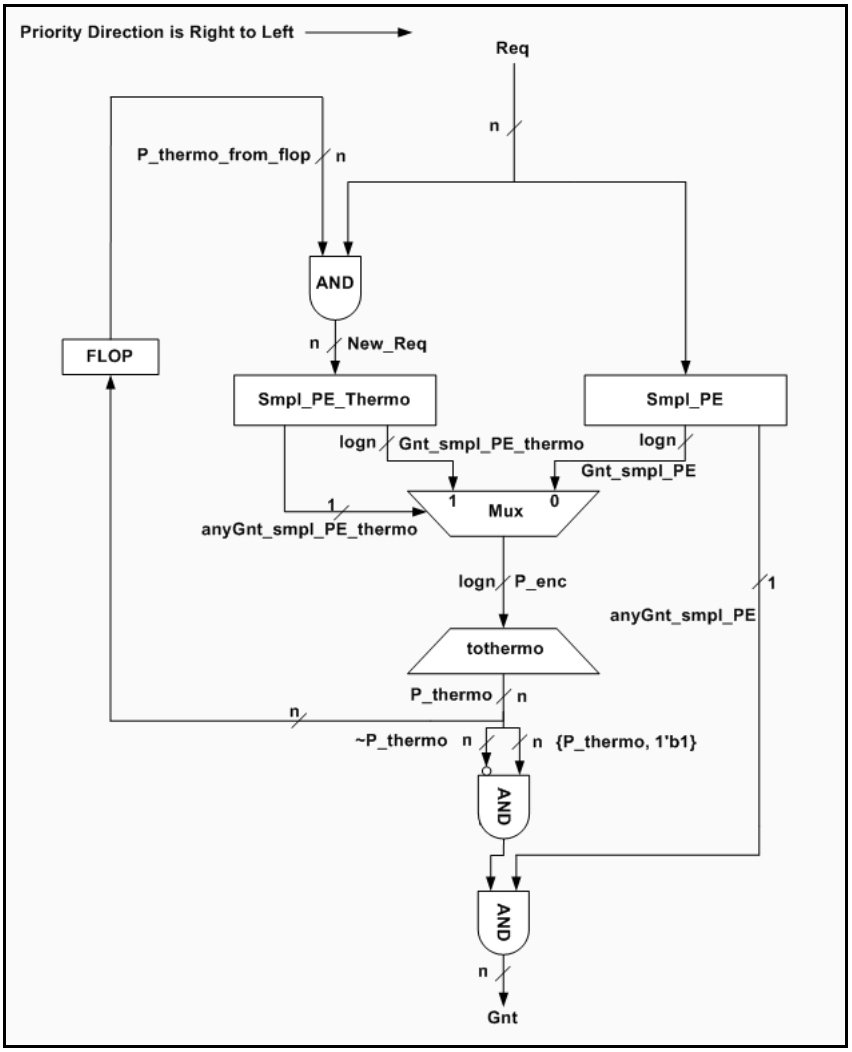

Figure 3.24: STA_RRA_N2LOGN architecture

After the tothermo block an edge detection operation is accomplished by AND gate. Also, final output is ANDed with Smpl_PE's any bit to eliminate the conflicts when zero requests are asserted. For example, a request which has 1'b0 value for each bit position is asserted for new iteration. In this iteration, RRA's output's all bit positions to be 1'b0. If we do not use final AND gate, RRA's output's LSB will be

1’b1. This example is illustrated in Figure 3.25. In order to eliminate this problem, that AND gate is placed at the end of the critical path.

Without using final AND gate		With using final AND gate	
Request	→ 0 0 0 0 0 0 0 0 0	Request	→ 0 0 0 0 0 0 0 0 0
~P_thermo	→ 1 1 1 1 1 1 1 1	~P_thermo	→ 1 1 1 1 1 1 1 1
{P_thermo,1'b1}	→ 0 0 0 0 0 0 0 1	{P_thermo,1'b1}	→ 0 0 0 0 0 0 0 1
~P_thermo & {P_thermo,1'b1}	→ 0 0 0 0 0 0 0 1	~P_thermo & {P_thermo,1'b1}	→ 0 0 0 0 0 0 0 1
Gnt = P_thermo & {P_thermo,1'b1}	→ 0 0 0 0 0 0 0 1	Gnt = (8'b0 & (P_thermo & {P_thermo,1'b1}))	→ 0 0 0 0 0 0 0 0

Figure 3.25: Zero request example

N2LOGN Smpl_PE’s are implemented recursively with binary tree technique. These blocks have order of $\log_2 N$ logic level and this is better than ripple carry Smpl_PE.

3.4.2. STA_PPT_RRA Architecture

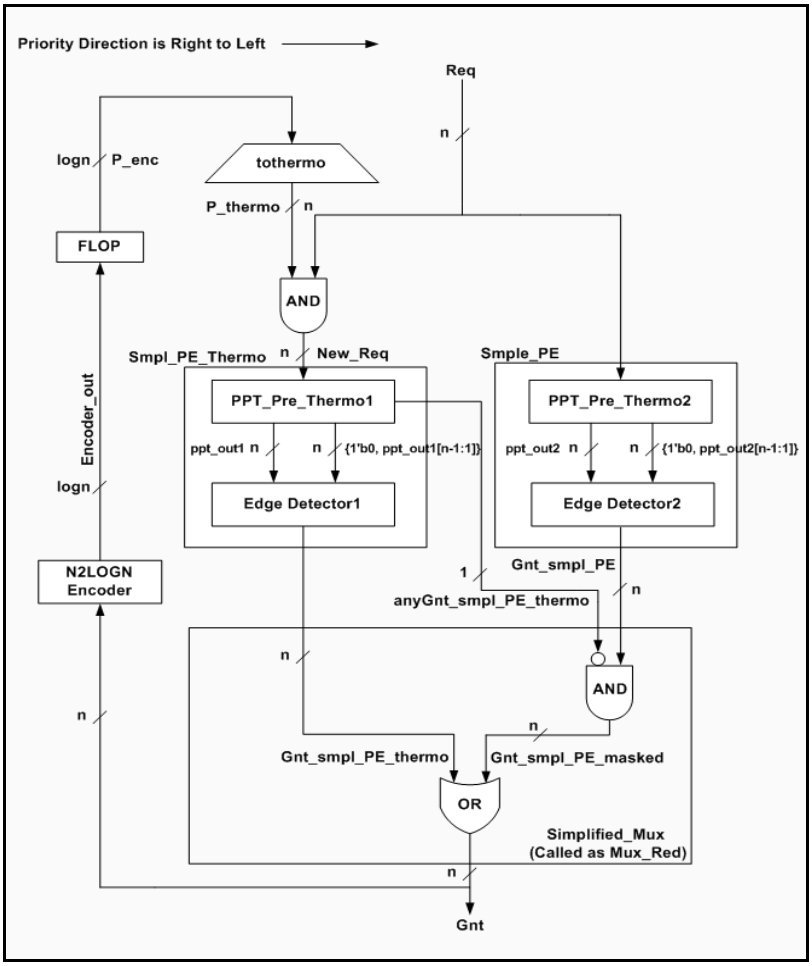

Figure 3.26: STA_PPT_RRA architecture

This architecture is almost as the same as STA_RRA_N2N architecture. Only difference is in Smpl_PE blocks. In this architecture HC_PPT_Pre_Thermo— ED blocks combination is used rather than Optimized Smpl_PE blocks. This architecture is shown in Figure 3.26.

3.4.3. CHN_PPT_RRA_PPE_Conflict Architecture

In this architecture HC_PPT_Pre_Thermo— ED blocks combination is used rather than Optimized Smpl_PE block. This architecture is shown in Figure 3.27. This is the only difference from the original CHN_RRA_PPE_Conflict architecture.

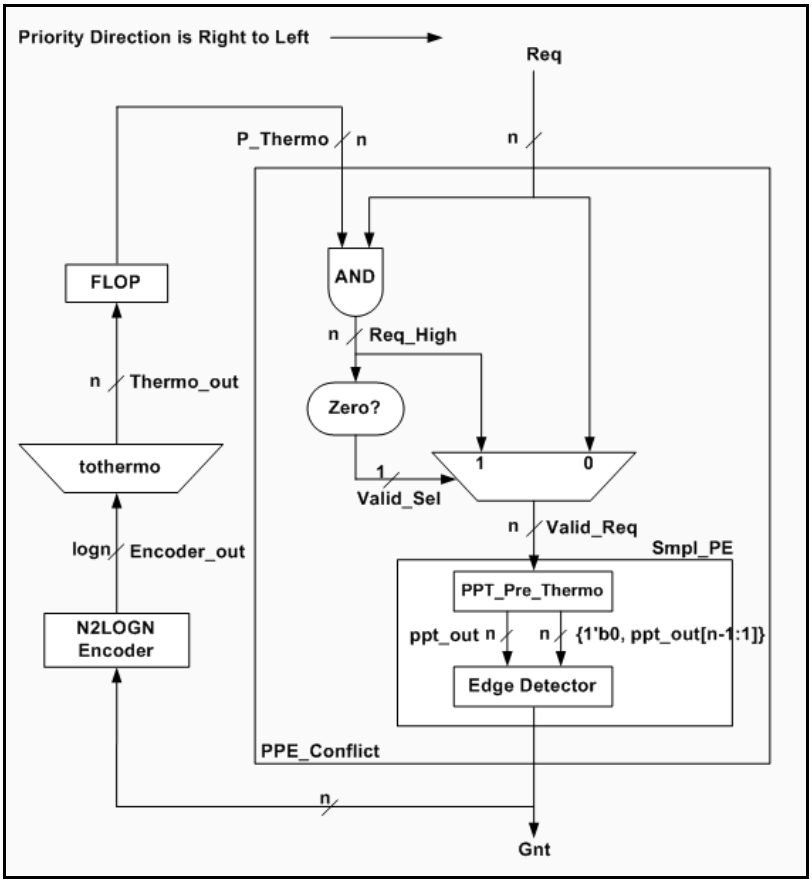

Figure 3.27: CHN_PPT_RRA_PPE_Conflict architecture

3.4.4. CHN_PPT_RRA_PPE_NonConflict Architecture

In this architecture HC_PPT_Pre_Thermo— ED blocks combination is used rather than Optimized Smpl_PE block too. This architecture is shown in Figure 3.28. This is the only difference from the original CHN_RRA_PPE_NonConflict architecture.

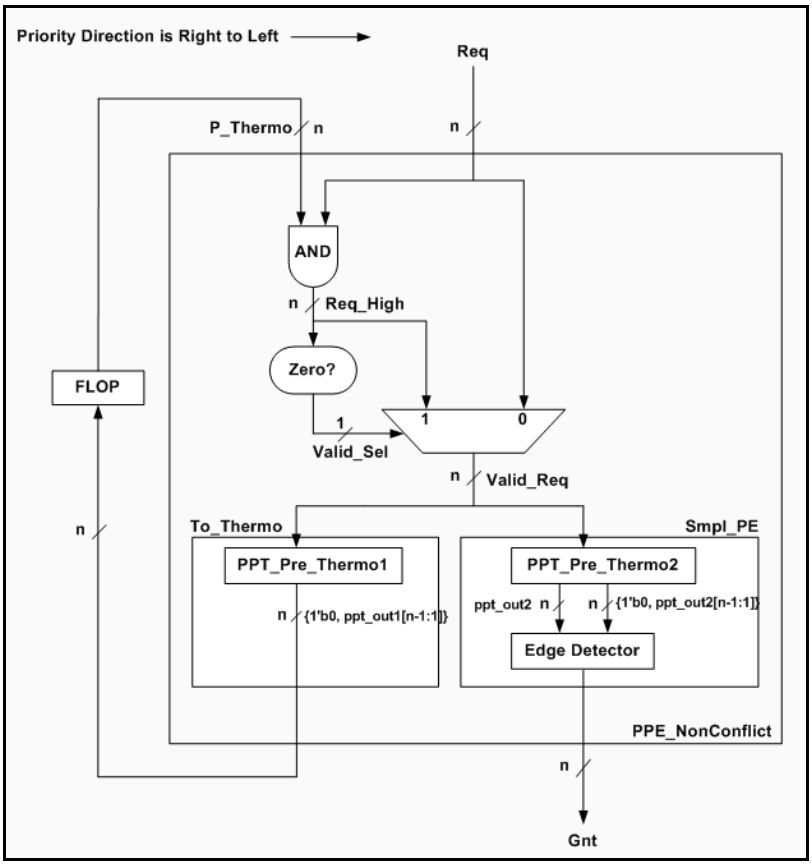

Figure 3.28: CHN_PPT_RRA_PPE_NonConflict architecture

4. AUTOMATIC RTL GENERATION

Register Transfer Level (RTL) coding is an HDL coding technique in which the behavior of a design is defined in terms of transfer of data between hardware registers, and logical operations performed on those signals.

Automatic RTL code generation has a very important role in this work. All RRA architectures have to be compared against each other from 8-bit input to 256-bit input variations. It is easy to write 8 Bit RRA's HDL code. On the other hand, if we try to write 256-bit RRA code, we would get into huge trouble. We probably lose consistency and make a lot of syntax and instantiation errors during writing up 128-bit or 256-bit RRA's Verilog HDL code. It is very cumbersome and tough to write 256-bit RRA code. In addition, it takes too much time. In order to get rid of these problems we tried to automate this process. We used PHP language and wrote scripts that automatically generate Verilog HDL code of RRA architectures with respect to their input bit widths. Those scripts take only one argument, which is the input bit width of the RRA. Then, scripts automatically generate all necessary Verilog HDL files for a specific RRA. Automatic RTL code generation task is the one of the most coercive tasks for this thesis and details of all generators are described in the next sections.

4.1. STA_RRA GENERATION

There are three different types of STA_RRA generators are coded. These are STA_RRA_N2N generator which is shown in figure 4.1., STA_RRA_N2LOGN generator, and STA_RRA_PPT generator. All generators perform the same task. They take an argument which is the input bit width of architecture, then generate Verilog HDL files, and finally put all generated files to a specific folder.

Generated Verilog HDL files of STA_RRA_N2N generator are shown in Figure 4.1.

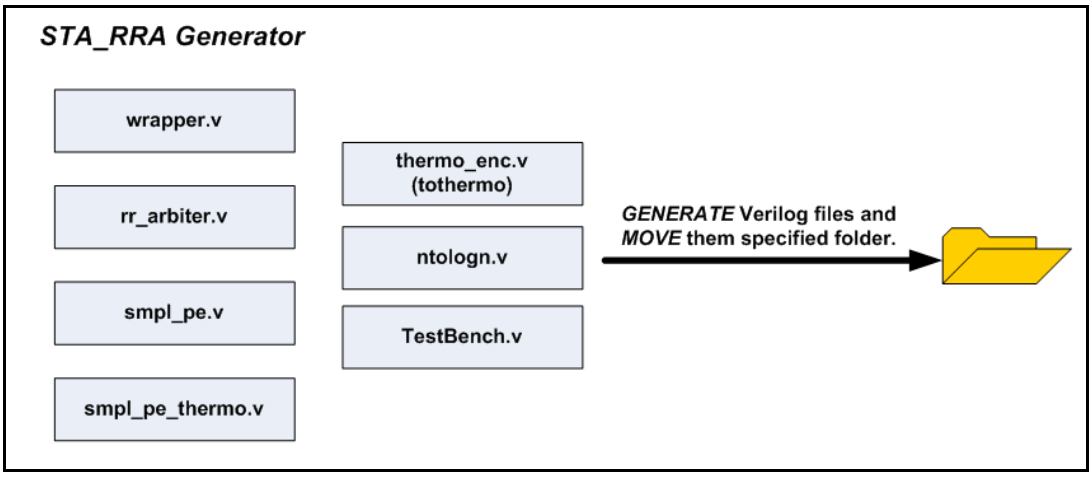

Figure 4.1: STA_RRA generation

wrapper.v is used for iterative synthesis. It is just a wrapper for top level design. It has two n-bit flops at its input and output ports. These flops are necessary to create timing path from request to grant of RRA.

rr_arbiter.v is top level block of RRA. It instantiates other blocks and performs RRA operation. Simplified multiplexer code is in this Verilog HDL file.

smpl_pe.v and *smpl_pe_thermo.v* is optimized simple priority encoder code. This priority encoder's Verilog code is generated with regard to binary tree and pre-computation/factoring techniques which are explained in Section 2.1.1. "Any bit" of smpl_pe_themro.v is generated with binary tree technique.

thermo_enc.v executes thermometer encoding and its algorithm is represented in Pankaj's and McKeown's work. Hence, we did not try to optimize this logic. Also, its Verilog HDL code is written in a parameterized fashion, so generation of this block became easier than the other blocks.

ntologn.v is generated with respect to binary tree technique as described in Section 2.1.3

testbecnh.v is the verification code of RRA which is used for RTL and gate level verifications.

In STA_RRA_N2LOGN architecture normal multiplexer is used rather than simplified multiplexer and N2LOGN encoder block generation is removed. Its architecture, which is shown in Figure 3.24 is taken into consideration when writing its RTL generator script.

Also, STA_RRA_PPT architecture generated with respect to Figure 3.26.

4.2. CHN_RRA GENERATION

There are four different generators written for CHN_RRA architectures. One of the generators is written for CHN_RAA_PPE_Conflict which is shown in Figure 4.2. It generates necessary Verilog HDL files and moves them into a specific folder. Smpl_PE, N2LOGN, tothermo, wrapper, and testbench modules are the same for STA_RRA and this architecture. The other generator is written for CHN_PPT_RAA_PPE_Conflict architecture with respect to Figure 3.27. In that architecture the difference is the HC_PPT_Pre_Thermo block plus ED block generation.

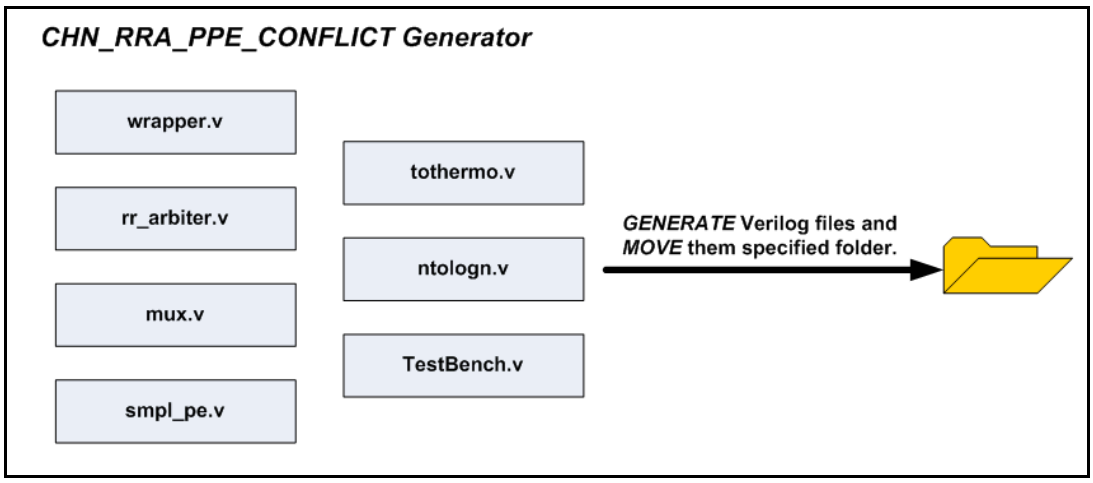

Figure 4.2: CHN_RRA_PPE_Conflict generation

CHN_RRA_PPE_NonConflict generation is shown is Figure 4.3 and its variant CHN_PPT_RRA_PPE_NonConflict architecture is generated according to Figure 3.28. In the original version of these architectures, combinations of "Smpl_PE + N2LOGN Encoder + tothermo" blocks are used to construct the N2N thermometer encoder block. This is not an efficient way to construct N2N thermometer encoder.

Thus, we used HC_PPT_Pre_thermo block to optimize N2N thermometer encoder block.

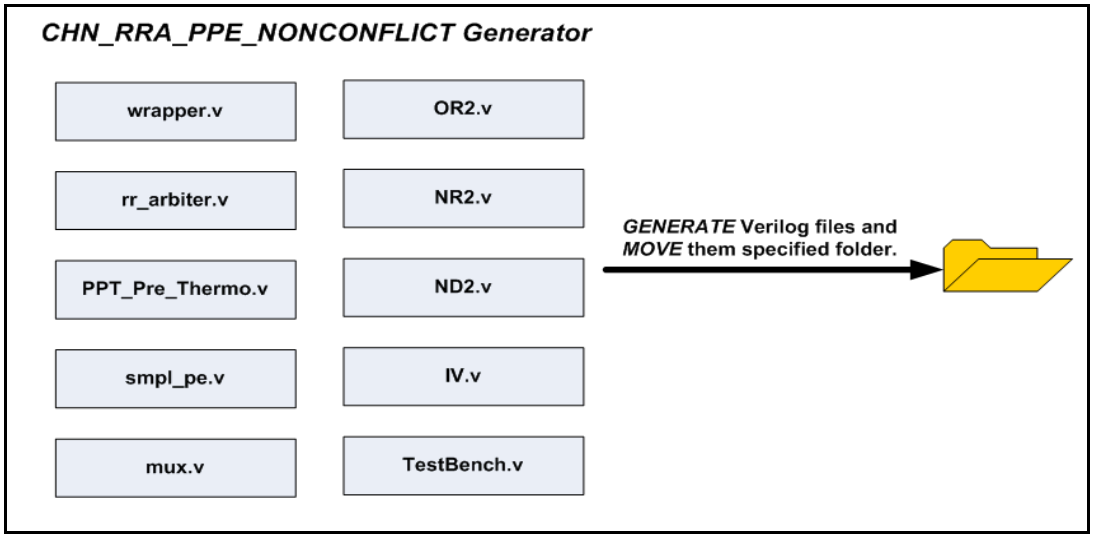

Figure 4.3: CHN_RRA_PPE_NonConflict (*OPTIMIZED) generation

4.3. PPT_RRA GENERATION

Nine different generators are written for PPT_RRA_RS and PPT_RRA_BT architectures, which are shown in Figures 4.4 and 4.5, respectively. All PPT algorithms' generators are written for PPT_Pre_Thermo Block. This is one of the most difficult tasks of this thesis. LF, BK, HC, and KS PPT_Pre_Thermo blocks are generated with respect to figures which are shown in chapter 3. In top-level module all of the sub-modules are instantiated and connected. Testbench code is same for all generators because they are performing the same RRA algorithm. Also, wrapper is the same for all architectures as well.

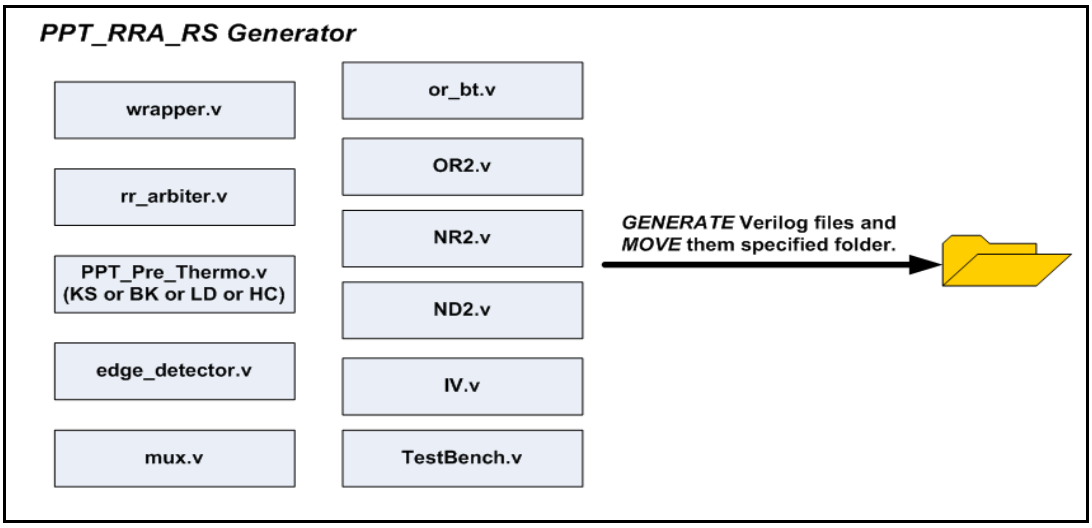

Figure 4.4: PPT_RRA_RS generation

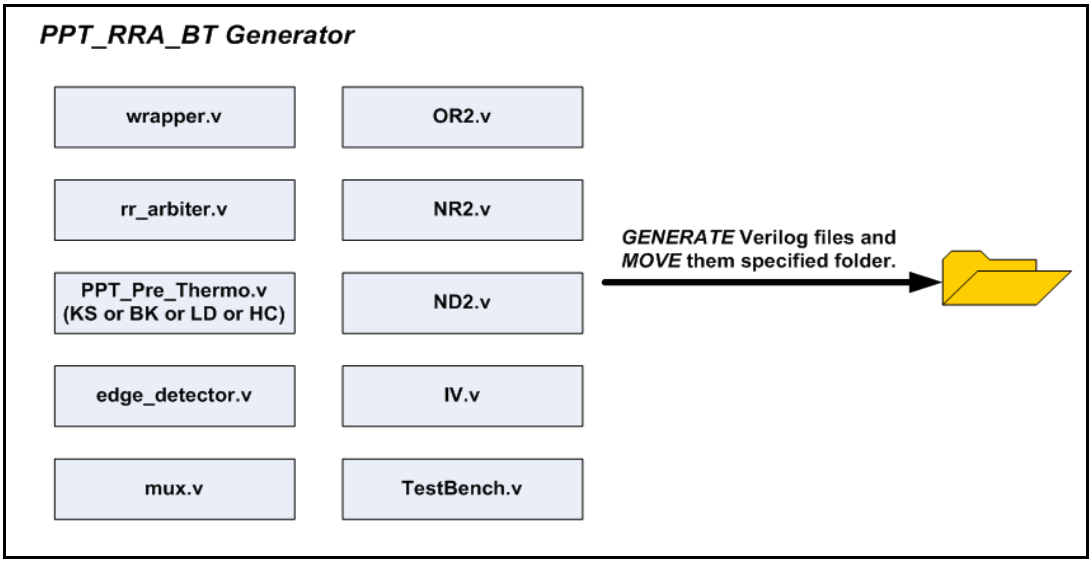

Figure 4.5: PPT_RRA_BT generation

4.4. BOW-TIE_RRA GENERATION

BOW-TIE_RRA architecture generation is accomplished with respect to all figures in Section 3.3. RR blocks' functionality is captured from Table 3.1 and its generator is written by using those equations. Also, GUNIT module is a very small module that it is generated easily. GUNIT modules are instantiated in gunit_n.v and RR modules are instantiated in rr_n.v. Eventually, GUNIT macro block and RR macro block are instantiated in bow-tie.v top level module. This architecture's generator

outputs the same wrapper.v and testbecnh.v files as the other generators. This generator is shown in Figure 4.6.

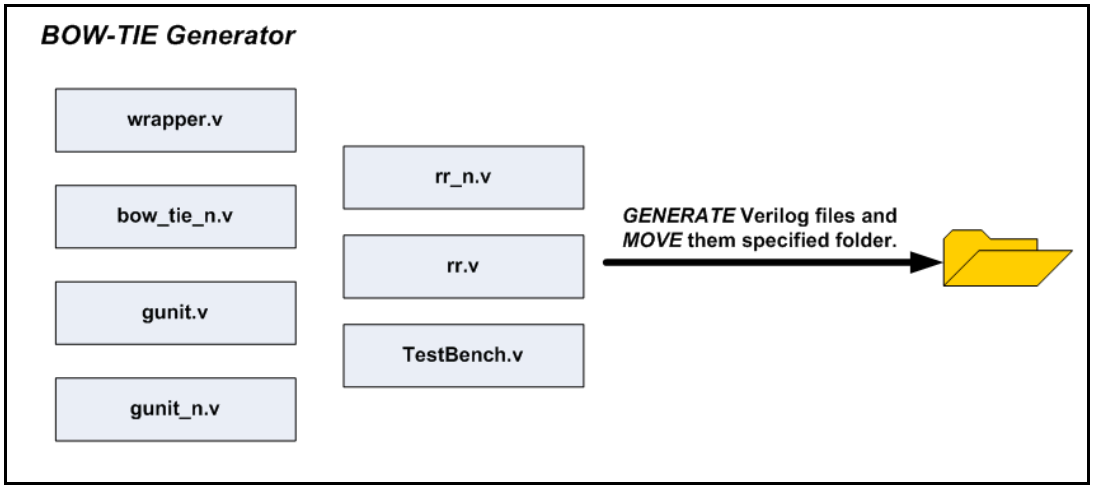

Figure 4.6: BOW-TIE generation

5. VERIFICATION AND SYNTHESIS METHODOLOGY

All RRA architectures' Verilog HDL codes are generated via their RTL generators. After this generation operation, all RTL codes are verified by RTL simulation. Then, verified RTL code is synthesized to get gate level netlist. There are many options in synthesis work. Different constraints could be applied to our design to get best synthesis results. All of these issues are covered in the following sections.

5.1. VERIFICATION

In verification task, we have to verify our RTL code's functionality first. Our design's RTL code is tested with behavioral test model. Behavioral test model performs our design functionality so RTL simulation is also called functional verification. The test model is written with a very high level of abstraction to fulfill the functionality of the RRA algorithm. This behavioral model, or in other words functional test model is in a module which is called testbench. We instantiate our RTL design into that testbench and apply the same test vectors to both our RTL design and behavioral test model. Eventually, outputs of these two blocks are compared for verification. In this way we check our RRA RTL design's functionality. If RTL verification phase is passed without any problem, then that RTL code could be synthesized by DC. After synthesis, we get gate level netlist regarding to our technology library. We have to verify this gate level netlist' functionality too, so gate level verification process is started. Verilog HDL code of technology library, gate level netlist, and test model are used to accomplish the gate level functional verification. Testbench instantiates gate level netlist to compare its functionality with that of the behavioral test model. After these phases our RTL design and gate level design are verified. Verification process is shown in Figure 5.1. This figure is valid for both RTL and gate level verification tasks.

In this thesis, the test model is written with respect to STA_RRA arbitration algorithm, which is the combination of *i*SLIP and mRRR algorithms, and it is almost identical to ESLIP algorithm, which is described in the introduction section. This

behavioral test model is coded with a high level of abstraction and it is used for verification of all RRA architectures.

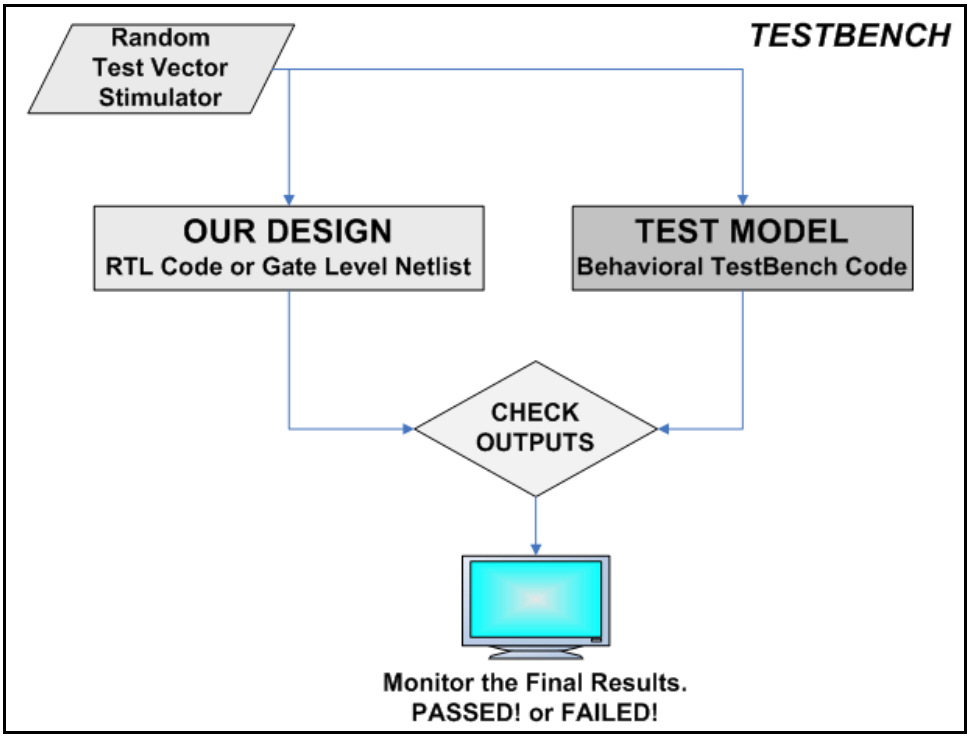

Figure 5.1: Verification strategy

Testbench code for 8 bit RRA is shown below.

```
module rr_arbiter_tb ();

    //Datatype Declerations
    reg clk, rst;
    reg arbitrate;
    reg [7:0] req;
    reg [7:0] tb_out;
    reg [3:0] ptr;
    reg flag;
    wire [7:0] design_out;
    integer i, k, m, down, up;


    //Instatiations
    rr_arbiter rr_arbiter_ins (.clk(clk), .rst(rst), .req(req),
                               .gnt(design_out));

    //Decleration of Initial Values
    initial begin
        clk = 0;
        #5 rst = 1;
        #15 rst = 0;
    end
```

```
//Toggle clock every 10 nanoseconds
    always begin
        #10 clk = ~clk;
    end


//Test Vectors (Inputs) Declerations
    always @(posedge clk) begin
        if (rst) begin
            ptr <= 4'd8;
            req <= 8'b0;
            tb_out <= 8'b0;
            arbitrate <= 1'b0;
        end
        else begin
           req <= $random;
           arbitrate <= ~arbitrate;
        end
    end


//Test Model
    always @(arbitrate) begin:RR_ARBITER
       if (req == 8'b0) begin
          tb_out = 8'b0;
          ptr = 4'd8;
       end
       else begin
          flag = 0;
          if (ptr != 3'b0) begin
             for (i=(ptr-1); i>=0; i=i-1) begin
                 if (req[i]==1'b1) begin
                     tb_out[i] = 1'b1;
                     ptr = i;
                     for (down=(i-1); down>=0; down=down-1) begin
                         tb_out[down] = 1'b0;
                     end
                     for (up=(i+1); up<8; up=up+1) begin
                         tb_out[up]=1'b0;
                     end
                     flag = 1;
                     disable RR_ARBITER;
                 end
             end
          end
          if (flag == 1'b0) begin
              for (k=7; k>=ptr; k=k-1) begin
                  if(req[k]==1'b1) begin
                      tb_out[k] = 1'b1;
                      ptr = k;
                      for (m=(k-1); m>=0; m=m-1) begin
                          tb_out[m] = 1'b0;
                      end
                      disable RR_ARBITER;
                  end
              end
          end
       end
    end
```

```
//Control and Monitoring Part
    always @(posedge clk) begin
        #2 //Wait 2ns for correct comparison
        if (design_out == tb_out) begin
            $display ("Time = %d\t req = %b\t design_output = %b\t
                      testmodel_output = %b\t --> CORRECT", $time, req,
                      design_out, tb_out);
        end
        else begin
            $display ("Time = %d\t req = %b\t design_output = %b\t
                      testmodel_output = %b\t --> ERROR", $time, req,
                      design_out, tb_out);
            $finish;
        end
    end

endmodule
```

5.2. SYNTHESIS METHODOLOGY

The synthesis task is done by DC. It calculates timing results according to timing paths. Timing path is a path that starts from a flop's clock trigger and ends at another flop's input. The timing paths are taken into account when writing timing constraints. In order to write timing constraints for RRA design, we have to put that design into a wrapper. The wrapper has flops at its input and output ports. RRA design also comprises of a flop to keep priority pointer. Therefore, we can create two timing paths from flop1's clk input to flop2's input and flop1's clk input to flop3's input. These two timing paths and other two timing paths are shown in Figure 5.2. Critical path of the design depends on combinational logic blocks which are located in these timing paths. If a timing path contains more combinational logic blocks, its delay will be bigger than the other timing paths. This path is called the critical path. Any design's speed is calculated with its critical path's timing.

A wrapper structure is ideal to create timing constraints for all RRA designs. We created a clock period in our synthesis scripts iteratively and tried to find in which clock period a positive slack occurs. We started with an over-constrained clock period value, but for each synthesis iteration we increased the clock period by one nanosecond to reach positive slack results. Positive slack means that our design meets the timing constraint. There are no timing violations when we see positive slack at the end of synthesis.

Area result is reported for RRA design. If we report area for wrapper, area result contains input and output flops in wrapper block. Wrapper block is not our actual design so area result must be reported with respect to RRA design.

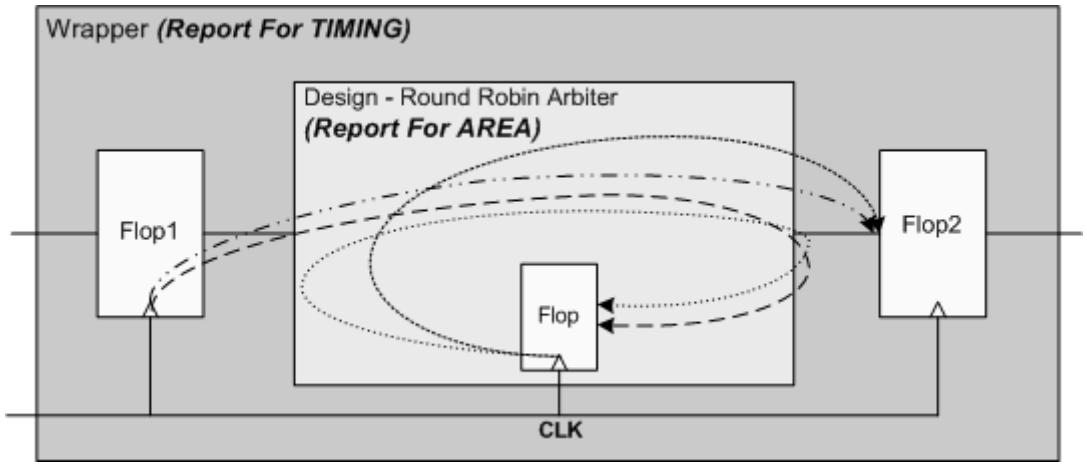

Figure 5.2: Synthesis methodology

We run several synthesis tasks to find the best timing and area constraint. Synthesis tasks are done with no constraint (NC), set_max_Area 0 (SMA0), and SMA0 + compile incremental –map effort high (CI_MEH) constraints. After all synthesis results we compare synthesis constraints to use the best one. Eventually we use the constraints below in our synthesis scripts.

```
# Timing constraint
create_clock –name clk –period 6 clk

# Area constraint
set_max_area 0

# Compile the design incrementally with high effort
compile –incremental_mapping –map_effort high

# Report area results
current_design rr_arbiter
report_area

# Report timing results
current_design wrapper
report_timing
```

In this work many RRA designs are synthesized. This task is notably time consuming. In order to shorten the synthesis task time, we automated synthesis with PERL scripts. Two PERL scripts are written for synthesis. One of them does the

synthesis task and the other one uses this script and accomplishes all RRA designs' synthesis tasks. These scripts are called core synthesis script and regression script. They are explained as follows.

Core Synthesis Script *does the following tasks:*

- Creates folders for synthesis scripts and results.
- Reads sample synthesis scripts and create new ones for iterative synthesis. In this part clock period is increased by one nanosecond from start point to end point. These points could be defined in the script. If you want to do synthesis from 6 to 20 you can easily set these points in the script.
- Invokes DC and runs all scripts and keeps their logs.
- Moves synthesis scripts and synthesis results into related folders.
- Modifies synthesis results, wraps and writes them into a related folder.
- Captures slack and total cell area values from modified result file.
- Writes total cell area and slack values to excel file.

Regression Script synthesizes all RRA designs by using specific *synthesis script* for each RRA designs. This script automatically synthesizes all synthesis tasks. Sample code snippet for this script is shown below.

```
$path1="C:/workspace/CHN_RRA/PPE_Conflict/8Bits";
$path2="C:/workspace/PPT_RRA/LF_RRA_BT/8Bits";
$path3="C:/workspace/STA_RRA/STA_RRA_N2N/8Bits";

print ("\n\n$path1 --> This path's synthesis is started!!!\n\n");
chdir($path1) || die "Can't chdir: $!";
system ("perl RRA_Syn_CHN.pl");
print ("\n\n$path1 --> This path's synthesis is finished!!!\n\n");

print ("\n\n$path2 --> This path's synthesis is started!!!\n\n");
chdir($path2) || die "Can't chdir: $!";
system ("perl RRA_Syn_LF_PPT.pl");
print ("\n\n$path2 --> This path's synthesis is finished!!!\n\n");

print ("\n\n$path3 --> This path's synthesis is started!!!\n\n");
chdir($path3) || die "Can't chdir: $!";
system ("perl RRA_Syn_STA_N2N.pl");
print ("\n\n$path3 --> This path's synthesis is finished!!!\n\n");
```

6. SYNTHESIS RESULTS

All of the synthesis is performed with Synopsys DC with lsi_10k technology library. Detailed iterative synthesis results from 8-bits to 256-bits and synthesis results at first positive or zero slack are shown in this chaper.

Round Robin Arbiters *TIMING* Comparison for *8Bit Input*

Time (ns)	STA N2N	STA PPT	STA N2LOGN	CHN PPE Conflict	CHN PPE Conflict PPT	CHN PPE Non Conflict	CHN PPE Non Conflict PPT	BK_BT	BK_RS	HC_BT	HC_BT Simple Mux	HC_RS	KS_BT	KS_RS	LF_BT	LF_RS	BOWTIE
6	-2,51	-2,88	-2,65	-4,3	-5,12	-2,08	-2,58	-1,7	-2,83	-1,38	-1,24	-3,17	-1,35	-3,16	-1,27	-2,95	-2,76
7	-1,51	-1,88	-1,65	-3,3	-4,22	-1,08	-1,58	-0,45	-1,85	-0,39	-0,26	-2,17	-0,62	-2,06	-0,27	-1,95	-1,87
8	-0,49	-0,91	-0,65	-2,22	-3,14	-0,2	-0,61	0,01	-0,89	0	0,07	-1,17	0,02	-1,22	0	-0,96	-0,73
9	0	-0,15	0,02	-1,32	-1,99	0,01	0,01	0,39	-0,01	0,39	0,08	-0,11	0,02	0	0,48	-0,01	0,02
10	0,11	0,02	0,03	-0,5	-1,28	0,16	0,04	0,46	0	0,2	1,23	0,08	0,13	0	0,46	0,01	0,1
11	0,95	0,18	0,11	0,02	-0,42	0,15	0,13	1,46	0,17	1,2	2,23	0,02	1,13	0,1	1,46	0,03	0,06
12	1,75	0,06	0,8	0,04	0,01	1,15	0,49	2,46	0,11	2,2	3,23	0,38	2,13	0,61	2,46	0,41	0,08
13	2,75	0,74	1,82	0,24	0,01	2,15	1,49	3,46	1,11	3,2	4,23	1,38	3,13	1,61	3,46	0,86	0,86
14	3,75	1,74	2,82	0,21	0,01	3,15	2,49	4,46	1,34	4,2	5,23	2,38	4,13	2,19	4,46	1,86	1,86
15	4,75	2,74	3,82	1,61	0,13	4,15	3,49	5,46	2,34	5,2	6,23	3,38	5,13	3,19	5,46	2,86	2,86
16	5,75	3,74	4,82	2,55	0,74	5,15	4,49	6,46	3,34	6,2	7,23	4,38	6,13	4,19	6,46	3,86	3,86
17	6,75	4,74	5,82	3,55	1,69	6,15	5,49	7,46	4,34	7,2	8,23	5,38	7,13	5,19	7,46	4,86	4,86
18	7,75	5,74	6,82	4,55	2,69	7,15	6,49	8,46	5,34	8,2	9,23	6,38	8,13	6,19	8,46	5,86	5,86
19	8,75	6,74	7,82	5,55	3,69	8,15	7,49	9,46	6,34	9,2	10,23	7,38	9,13	7,19	9,46	6,86	6,86
20	9,75	7,74	8,82	6,55	4,69	9,15	8,49	10,46	7,34	10,2	11,23	8,38	10,13	8,19	10,46	7,86	7,86

Note: Green RRA's are our architectures, orange and blue ones are rival architectures and their modifications.

Round Robin Arbiter *AREA* Comparison for *8Bit Input*

Time (ns)	STA N2N	STA PPT	STA N2LOGN	CHN PPE Conflict	CHN PPE Conflict PPT	CHN PPE Non Conflict	CHN PPE Non Conflict PPT	BK_BT	BK_RS	HC_BT	HC_BT Simple Mux	HC_RS	KS_BT	KS_RS	LF_BT	LF_RS	BOWTIE
6	180	188	225	246	262	236	244	186	213	199	190	207	230	209	222	212	437
7	179	188	231	246	249	237	244	204	213	200	183	207	206	211	223	212	440
8	170	187	232	241	251	238	247	165	215	174	159	206	177	179	169	202	466
9	167	189	196	241	256	173	195	154	211	154	148	196	161	187	156	206	404
10	138	153	182	225	254	161	182	153	155	153	145	160	160	156	155	157	387
11	132	147	178	186	237	152	166	153	149	153	145	148	160	150	155	147	378
12	133	146	173	166	211	152	161	153	144	153	145	144	160	148	155	145	364
13	133	146	173	160	177	152	161	153	144	153	145	144	160	148	155	145	362
14	133	146	173	158	171	152	161	153	144	153	145	144	160	148	155	145	362
15	133	146	173	156	166	152	161	153	144	153	145	144	160	148	155	145	362
16	133	146	173	156	166	152	161	153	144	153	145	144	160	148	155	145	362
17	133	146	173	156	166	152	161	153	144	153	145	144	160	148	155	145	362
18	133	146	173	156	166	152	161	153	144	153	145	144	160	148	155	145	362
19	133	146	173	156	166	152	161	153	144	153	145	144	160	148	155	145	362
20	133	146	173	156	166	152	161	153	144	153	145	144	160	148	155	145	362

Note: Green RRA's are our architectures, orange and blue ones are rival architectures and their modifications.

Figure 6.1: 8 Bit RRAs synthesis results

	Round Robin Arbiters *TIMING* Comparison for *16Bit Input*																	
Time (ns)	STA N2N	STA PPT	STA N2LOG N	CHN PPE Conflict	CHN PPE Conflict PPT	CHN PPE Non Conflict	CHN PPE Non Conflict PPT	BK_BT	BK_RS	HC_BT	HC_BT Simple Mux	HC_RS	KS_BT	KS_RS	LF_BT	LF_RS	BOWTIE
6	-4,63	-5,11	-3,88	-7,43	-8,3	-4,02	-4,47	-2,39	-4,89	-2,39	-2,1	-4,8	-2,27	-4,48	-2,33	-4,39	-4,91
7	-3,63	-4,11	-2,76	-6,43	-7,3	-3,02	-3,47	-1,32	-3,77	-1,25	-1,1	-3,8	-1,17	-3,48	-1,27	-3,39	-4,2
8	-2,63	-3,08	-1,76	-5,46	-6,3	-2,04	-3,13	-0,38	-2,77	-0,24	-0,12	-2,8	-0,27	-2,48	-0,27	-2,39	-3,32
9	-1,79	-2,08	-0,87	-4,46	-5,3	-1,05	-1,98	0,01	-1,77	0	0	-1,9	0,02	-1,48	0,01	-1,41	-2,03
10	-0,62	-1,05	0	-3,22	-4,34	-0,18	-0,99	0,2	-0,93	0,11	0,02	-0,92	0,04	-0,46	0,02	-0,47	-1,34
11	0,02	0,01	0,04	-2,4	-3,01	0,01	0	0,11	0	0,36	0,11	0	0,36	0	0,51	0	-0,5
12	0,01	0,06	0,03	-1,59	-2,51	0	0,02	0,03	0,06	0,03	0,24	0	0,07	0	0,55	0,02	0
13	0,94	0,17	0,13	-0,62	-1,02	0,03	0,01	0,37	0,02	0,97	2,63	0,02	0,75	0,04	0,6	0,07	0,06
14	0,95	0,1	0,05	0,06	-0,02	0,32	0,13	0,06	0,01	0,66	3,63	0,3	0,59	0,04	1,6	0,23	0,08
15	0,89	0,06	0,62	0,12	0	0,94	1,13	1,06	0,16	1,66	4,63	0,93	1,59	0,47	0,54	0,19	0,03
16	2,14	0,33	0,25	0,15	0,03	0,27	0,46	2,06	0,13	2,66	5,63	0,66	2,59	1,37	1,54	1,04	0,49
17	3,14	1,33	0,12	0,13	0,02	1,27	1,46	3,06	0,49	3,66	6,63	1,26	3,59	0,76	2,54	0,98	1,49
18	4,14	2,33	1,12	0,24	0,13	0,65	0,85	4,06	0,46	4,66	7,63	0,65	4,59	1,76	3,39	1,37	2,49
19	5,14	3,33	2,12	0,84	0,27	1,65	1,85	5,06	1,46	5,66	8,63	1,65	5,59	2,76	4,39	0,75	3,49
20	6,14	4,33	3,12	0,79	0,87	4,27	2,85	6,06	0,85	6,66	9,63	2,65	6,59	3,76	5,39	1,75	4,49
Note: Green RRA's are our architectures, orange and blue ones are rival architectures and their modifications.																		

	Round Robin Arbiter *AREA* Comparison for *16Bit Input*																	
Time (ns)	STA N2N	STA PPT	STA N2LOG N	CHN PPE Conflict	CHN PPE Conflict PPT	CHN PPE Non Conflict	CHN PPE Non Conflict PPT	BK_BT	BK_RS	HC_BT	HC_BT Simple Mux	HC_RS	KS_BT	KS_RS	LF_BT	LF_RS	BOWTIE
6	412	409	515	516	519	510	544	393	405	403	425	411	496	409	415	425	920
7	411	409	498	516	570	508	544	414	428	421	429	411	523	409	440	425	906
8	410	390	501	510	570	493	481	400	428	421	405	411	501	403	435	430	868
9	393	390	507	512	569	448	482	361	418	349	322	391	374	403	352	393	874
10	372	411	441	542	535	425	483	320	379	326	311	382	352	364	327	385	877
11	378	409	422	487	547	346	417	316	377	326	311	363	352	329	326	332	879
12	307	329	414	533	479	327	357	315	321	325	310	319	351	315	325	304	800
13	289	318	390	512	600	317	342	315	307	325	309	300	351	310	324	299	791
14	290	313	383	400	564	321	339	315	295	325	309	297	351	309	324	297	756
15	285	309	379	379	420	321	339	315	292	325	309	297	351	309	323	296	751
16	293	315	378	353	385	320	338	315	292	325	309	296	351	309	323	295	746
17	293	315	377	342	375	320	338	315	291	325	309	296	351	308	323	294	746
18	293	315	377	339	359	319	337	315	291	325	309	295	351	308	323	294	746
19	293	315	377	339	356	319	337	315	291	325	309	295	351	308	323	293	746
20	293	315	377	337	356	320	337	315	290	325	309	295	351	308	323	293	746
Note: Green RRA's are our architectures, orange and blue ones are rival architectures and their modifications.																		

Figure 6.2: 16 Bit RRAs synthesis results

	Round Robin Arbiters *TIMING* Comparison for *32Bit Input*																	
Time (ns)	STA N2N	STA PPT	STA N2LOGN	CHN PPE Conflict	CHN PPE Conflict PPT	CHN PPE Non Conflict	CHN PPE Non Conflict PPT	BK_BT	BK_RS	HC_BT	HC_BT Simple Mux	HC_RS	KS_BT	KS_RS	LF_BT	LF_RS	BOWTIE
6	-6,55	-6,67	-5,26	-9,75	-10,33	-5,59	-5,79	-3,76	-6,14	-3,12	-2,92	-6,03	-3,21	-6,18	-2,97	-6,2	-8,03
7	-5,81	-5,82	-4,26	-8,81	-9,33	-4,64	-4,79	-2,76	-5,14	-2,13	-1,84	-5,03	-2,14	-5,18	-2	-5,2	-6,8
8	-4,73	-4,81	-3,26	-7,84	-8,33	-3,59	-3,82	-1,76	-4,05	-1	-0,84	-3,87	-1,2	-4,18	-1	-4,2	-5,46
9	-3,58	-3,81	-2	-6,88	-7,34	-2,62	-2,78	-0,63	-3,3	-0,24	0	-3,01	-0,05	-2,83	0	-2,86	-4,52
10	-2,8	-2,74	-0,9	-5,89	-6,34	-1,29	-1,83	0,02	-2,25	0,02	0,01	-2,22	0,01	-1,85	0,01	-2,24	-3,71
11	-1,87	-1,86	-0,13	-4,88	-5,3	-0,34	-0,83	0,02	-0,89	0,19	0,06	-1,08	0,19	-0,85	0,01	-1,1	-2,58
12	-0,55	-0,8	0,05	-3,8	-5,01	0,01	0	0,1	-0,59	0,43	0,35	0	0,43	0,01	0,1	0	-1,93
13	0	0	0,04	-3,13	-3,61	0,01	0	0,22	0	0,5	0,35	0	0,32	0,03	0,37	0	-1,3
14	0,03	0,01	0,01	-2,06	-2,54	0,17	0,04	0,52	0,01	1,15	1,29	0,02	1,22	0,31	0,37	0	0
15	0,02	0,04	0,01	-1,32	-1,41	0,11	0,29	1,35	0	2,15	1,02	0,14	2,22	0,6	0,53	0,18	0
16	0	0,08	0	0	-0,96	0,07	0,38	2,35	0,03	3,15	1,22	0,69	3,22	0,64	0,67	0,01	0,02
17	0,17	0,15	0,12	0,11	0	0,04	0,33	3,35	0,14	4,15	1,96	0,35	4,22	0,94	2,16	0,1	0,05
18	0,24	0,02	0,2	0,02	0,01	0,38	0,82	0,17	0,49	3,08	5,29	0,83	3,14	1,6	3,05	0,77	0,01
19	1,14	0,72	0,48	0,05	0,01	0,51	1,82	0,3	0,43	4,08	6,29	1,62	4,14	2,6	3,67	0,29	0
20	0,65	0,61	0,35	0,16	0,04	0,52	2,82	1,3	0,63	0,85	7,29	2,62	0,83	3,6	4,67	0,63	1
Note: Green RRA's are our architectures, orange and blue ones are rival architectures and their modifications.																		

	Round Robin Arbiter *AREA* Comparison for *32Bit Input*																	
Time (ns)	STA N2N	STA PPT	STA N2LOGN	CHN PPE Conflict	CHN PPE Conflict PPT	CHN PPE Non Conflict	CHN PPE Non Conflict PPT	BK_BT	BK_RS	HC_BT	HC_BT Simple Mux	HC_RS	KS_BT	KS_RS	LF_BT	LF_RS	BOWTIE
6	952	1004	1024	1169	1249	1077	1054	756	937	866	878	924	980	826	966	835	1743
7	841	981	1024	1185	1249	1052	1054	756	937	862	951	924	992	828	980	835	1794
8	890	964	1024	1139	1248	1086	1065	751	993	973	951	940	988	828	952	842	1801
9	902	923	1059	1190	1251	1046	1079	758	971	882	862	909	1038	964	875	920	1774
10	884	945	1058	1116	1251	1056	1039	721	990	701	699	916	785	880	776	860	1780
11	870	880	1019	1135	1245	1027	1046	653	989	686	653	850	769	868	697	814	1801
12	832	930	899	1135	1172	893	942	644	868	682	655	920	764	808	681	878	1751
13	785	825	853	1064	1179	756	795	643	762	680	652	686	762	687	677	687	1762
14	680	727	810	1072	1112	723	741	643	628	680	647	636	763	650	682	651	1663
15	639	679	810	1112	1157	687	715	643	609	680	651	617	763	651	676	630	1624
16	639	666	804	1096	1140	673	710	643	595	680	648	610	763	649	670	627	1573
17	623	664	773	887	1018	670	711	643	590	680	647	612	763	650	674	626	1553
18	626	660	766	808	879	669	707	643	589	680	647	608	763	649	672	624	1522
19	617	656	758	749	800	668	707	643	589	680	647	607	763	649	671	621	1514
20	619	658	751	733	787	673	707	643	589	679	647	607	762	649	671	607	1514
Note: Green RRA's are our architectures, orange and blue ones are rival architectures and their modifications.																		

Figure 6.3: 32 Bit RRAs synthesis results

	Round Robin Arbiters *TIMING* Comparison for *64Bit Input*																
Time (ns)	STA N2N	STA PPT	STA N2LOGN	CHN PPE Conflict	CHN PPE Conflict PPT	CHN PPE Non Conflict	CHN PPE Non Conflict PPT	BK_BT	BK_RS	HC_BT	HC_BT Simple Mux	HC_RS	KS_BT	KS_RS	LF_BT	LF_RS	BOWTIE
6	-8,87	-8,91	-6,69	-13,37	-13,33	-7,22	-7,47	-4,39	-8,32	-3,85	-3,66	-7,67	-3,76	-7,1	-3,81	-7,64	-10,21
7	-7,79	-7,88	-6,1	-12,37	-12,68	-6,18	-6,47	-3,37	-7,9	-2,97	-2,62	-6,65	-2,76	-6,1	-2,77	-6,6	-9,99
8	-6,46	-6,88	-4,96	-11,07	-11,62	-5,38	-5,62	-2,36	-6,52	-1,97	-1,76	-5,65	-1,75	-5,1	-1,79	-5,54	-9,11
9	-5,64	-6,03	-3,96	-10,64	-10,7	-4,44	-4,62	-1,29	-5,6	-0,97	-0,59	-4,67	-0,79	-4,12	-0,87	-4,64	-7,91
10	-5,07	-4,81	-2,81	-8,83	-9,66	-3,25	-3,61	-0,37	-4,43	0	0	-3,67	0	-3,18	0	-3,67	-6,05
11	-3,53	-4,06	-1,39	-8,2	-8,51	-2,09	-2,59	0	-3,55	0,02	0,01	-2,7	0,09	-2,18	0,01	-2,52	-5,58
12	-2,59	-2,82	-0,67	-7,09	-7,69	-1,33	-1,52	0,01	-1,86	0,04	0,1	-1,7	0	-1,15	0,01	-1,9	-4,19
13	-1,39	-1,89	0	-5,45	-6,61	-0,44	-0,72	0	-1,25	0,37	0	-0,7	0,21	-0,18	0,01	-0,97	-3,07
14	-0,63	-0,84	0	-5,13	-5,56	0	0,01	0,01	-0,45	0,01	0,02	0	0,01	0,01	0,03	0	-2,27
15	0,01	0	0,01	-4,34	-4,66	0,01	0	0,01	0	0,46	0,04	0,01	0,03	0,03	0,09	0	-1,31
16	0,01	0,01	0,01	-3,23	-3,68	0	0,04	0,02	0	0,29	0,3	0	0,07	0,07	0,25	0,03	-0,31
17	0,02	0,04	0,06	-2,04	-2,73	0,01	0,07	0,03	0,01	0,02	1,53	0,06	0,08	0,08	0,6	0,02	0
18	0,38	0,01	0,26	-1,06	-1,58	0,03	0,11	0,09	0,07	0,61	2,75	0,08	0,45	0,36	0,66	0,03	0
19	0,06	0	0,01	-0,17	-0,67	0,02	0,05	0,05	0,04	0,91	3,75	0,41	0,89	0,13	0,11	0,2	0,02
20	0,03	0,01	0,26	0	0	0,08	0,04	0,15	0,19	1,91	0,36	0,03	1,89	0,03	1,02	0,16	0,03

Note: Green RRA's are our architectures, orange and blue ones are rival architectures and their modifications.

	Round Robin Arbiter *AREA* Comparison for *64Bit Input*																
Time (ns)	STA N2N	STA PPT	STA N2LOGN	CHN PPE Conflict	CHN PPE Conflict PPT	CHN PPE Non Conflict	CHN PPE Non Conflict PPT	BK_BT	BK_RS	HC_BT	HC_BT Simple Mux	HC_RS	KS_BT	KS_RS	LF_BT	LF_RS	BOWTIE
6	1977	1872	2128	2283	2198	2019	1951	1603	1608	1814	1722	1676	2067	1764	1821	1712	3484
7	1971	1903	2099	2282	2157	2130	1949	1607	1571	1713	1751	1670	2068	1763	1840	1708	3450
8	1946	1886	2100	2203	2152	1986	1912	1598	1579	1712	1821	1670	2048	1752	1784	1723	3440
9	1848	1886	2100	2209	2141	1973	1906	1635	1580	1716	1870	1691	2071	1752	1738	1689	3481
10	1874	1978	2154	2217	2141	1983	1840	1628	1636	1680	1591	1605	1835	1621	1631	1568	3510
11	1896	1912	2125	2214	2239	2017	1816	1500	1609	1446	1414	1552	1746	1621	1536	1590	3474
12	1956	1881	2206	2262	2224	1957	1792	1384	1579	1436	1375	1544	1656	1637	1464	1608	3505
13	1867	1874	2072	2289	2236	1917	1850	1300	1576	1435	1401	1561	1655	1616	1443	1556	3524
14	1871	1890	1899	2227	2204	1719	1660	1298	1519	1435	1372	1443	1654	1416	1482	1485	3513
15	1631	1801	1748	2164	2161	1619	1551	1298	1395	1434	1371	1317	1717	1417	1408	1334	3476
16	1479	1532	1618	2311	2203	1460	1515	1297	1281	1433	1371	1274	1653	1369	1408	1284	3401
17	1401	1420	1578	2241	2124	1438	1491	1296	1219	1432	1368	1262	1652	1400	1407	1265	3276
18	1368	1396	1588	2218	2098	1427	1491	1296	1203	1432	1367	1256	1652	1365	1403	1269	3241
19	1364	1394	1607	2125	2049	1435	1492	1296	1201	1432	1367	1256	1652	1370	1404	1249	3188
20	1341	1449	1589	1842	1854	1418	1494	1296	1189	1432	1369	1257	1652	1375	1402	1246	3092

Note: Green RRA's are our architectures, orange and blue ones are rival architectures and their modifications.

Figure 6.4: 64 Bit RRAs synthesis results

	Round Robin Arbiters *TIMING* Comparison for *128Bit Input*																
Time (ns)	STA N2N	STA PPT	STA N2LOGN	CHN PPE Conflict	CHN PPE Conflict PPT	CHN PPE Non Conflict	CHN PPE Non Conflict PPT	BK_BT	BK_RS	HC_BT	HC_BT Simple Mux	HC_RS	KS_BT	KS_RS	LF_BT	LF_RS	BOWTIE
6	-10,48	-10,9	-8,12	-16,06	-15,95	-8,78	-8,92	-5,61	-10,73	-4,71	-4,71	-8,85	-4,98	-9,02	-4,67	-9,19	-13,2
7	-9,75	-9,69	-7,15	-15,53	-14,71	-7,59	-7,95	-4,55	-9,63	-3,74	-3,8	-7,92	-3,98	-8,02	-3,77	-8,3	-11,87
8	-8,46	-8,24	-6,29	-13,89	-14,15	-6,49	-6,9	-3,43	-8,36	-2,89	-2,71	-6,93	-2,94	-6,97	-2,75	-7,36	-10,94
9	-7,83	-7,69	-5,27	-12,73	-13,06	-5,75	-5,89	-2,58	-7,17	-1,89	-1,68	-5,96	-1,94	-5,79	-1,74	-6,1	-9,94
10	-6,23	-6,32	-4,04	-12,31	-12,03	-4,59	-5,12	-1,64	-6,4	-0,8	-0,77	-4,86	-0,84	-4,79	-0,59	-5,37	-8,94
11	-5,41	-5,91	-3,19	-10,45	-10,73	-3,44	-3,91	-0,39	-5,5	0	0	-4,01	0	-3,81	0	-4,32	-7,99
12	-4,46	-4,37	-2,16	-9,49	-10,25	-2,51	-3	0	-4,44	0,01	0,01	-2,93	0,01	-2,81	0	-3,29	-8,37
13	-3,68	-3,69	-1,28	-8,73	-9,34	-1,53	-1,91	0,01	-3,74	0,06	0,02	-1,96	0,02	-1,81	0,05	-2,19	-5,73
14	-2,35	-2,52	-0,08	-7,37	-8,35	-0,67	-0,83	0	-2,14	0,08	0,17	-1	0,1	-0,81	0,02	-1,1	-5,32
15	-1,57	-1,73	0	-6,82	-6,89	0	-0,02	0,02	-1,31	0,05	0,07	0	0,14	0	0,01	-0,31	-3,93
16	-0,66	-0,89	0	-5,94	-5,69	0,05	0	0,01	-0,94	0,1	0,22	0	0,1	0,02	0,05	0	-3,28
17	0	0	0	-4,44	-5,12	0,01	0,03	0,05	0	0,15	0	0,01	0,1	0,03	0	0	-1,69
18	0	0	0	-3,39	-4,32	0	0,02	0,02	0	0,37	0,11	0,09	0,41	0,01	0,13	0,06	-1,17
19	0	0,01	0,07	-2,47	-2,85	0,07	0,04	0,02	0	0,55	0,04	0,06	0,37	0	0,15	0,03	-0,18
20	0,01	0,01	0,04	-1,74	-2,37	0,02	0,11	0,05	0,09	1	0,08	0,06	0,66	0,05	0,13	0,02	0
Note: Green RRA's are our architectures, orange and blue ones are rival architectures and their modifications.																	

	Round Robin Arbiter *AREA* Comparison for *128Bit Input*																
Time (ns)	STA N2N	STA PPT	STA N2LOGN	CHN PPE Conflict	CHN PPE Conflict PPT	CHN PPE Non Conflict	CHN PPE Non Conflict PPT	BK_BT	BK_RS	HC_BT	HC_BT Simple Mux	HC_RS	KS_BT	KS_RS	LF_BT	LF_RS	BOWTIE
6	3883	3780	4089	4225	4673	4305	4469	3128	3193	3875	3837	3626	4220	3626	3741	3805	6946
7	3960	3871	4357	4088	5063	4690	4427	3096	3192	3914	3896	3624	4221	3625	3657	3539	6872
8	3872	3974	4053	4398	4774	4630	4457	3236	3174	3632	3813	3722	4273	3708	3785	3486	6862
9	3717	3860	4130	4525	4600	4379	4455	3110	3318	3627	3834	3728	4287	3883	3724	3690	6844
10	4033	3980	4183	4348	4525	4438	4257	3060	3287	3818	3710	3907	4307	3854	3751	3506	6889
11	3907	3810	4240	4546	5184	4660	4441	3192	3182	3679	3182	3505	3960	3610	3420	3605	6849
12	3928	3929	4209	4985	4802	4689	4393	2862	3204	3113	2939	3355	3734	3557	3343	3490	6873
13	3675	3894	4201	4442	4530	4327	4458	2656	3024	3009	2891	3366	3557	3552	3224	3500	6904
14	3852	3924	4002	5061	4492	4356	4572	2616	3118	3000	2876	3419	3556	3572	3166	3920	6896
15	3612	3820	3641	4384	4973	3786	4361	2612	3003	2993	2871	3285	3553	3364	3027	3428	6906
16	3912	3741	3587	4353	5082	3526	3326	2609	2947	2989	2865	2876	3548	3005	3001	2915	6924
17	3480	3494	3265	4459	4840	3321	3205	2609	2916	2990	2868	2697	3549	2961	2988	2753	6965
18	3215	3177	3229	4851	4699	3050	3136	2609	2568	2989	2865	2581	3548	2869	2959	2672	6913
19	2937	3083	3164	4272	4822	3001	3136	2608	2446	2987	2863	2596	3545	2886	2987	2640	6784
20	2996	3005	3156	4631	4449	2972	3132	2608	2399	2988	2864	2595	3546	2871	2947	2640	6475
Note: Green RRA's are our architectures, orange and blue ones are rival architectures and their modifications.																	

Figure 6.5: 128 Bit RRAs synthesis results

	Round Robin Arbiters *TIMING* Comparison for *256Bit Input*																
Time (ns)	STA N2N	STA PPT	STA N2LOG N	CHN PPE Conflict	CHN PPE Conflict PPT	CHN PPE Non Conflict	CHN PPE Non Conflict PPT	BK_BT	BK_RS	HC_BT	HC_BT Simple Mux	HC_RS	KS_BT	KS_RS	LF_BT	LF_RS	BOWTIE
10	-8,66	-8,12	-5,97	-14,76	-14,57	-6,98	-6,45	-2,61	-8,53	-0,88	-1,35	-6,76	-1,62	-6,07	-1,78	-5,88	-12,26
11	-7,38	-7,14	-5,15	-13,92	-13,68	-5,88	-5,53	-1,32	-7,25	0	-0,42	-5,76	-0,7	-5,14	-0,69	-4,99	-11,15
12	-6,64	-6,21	-3,97	-12,84	-12,68	-4,61	-4,49	-0,27	-6,24	0,13	0	-4,8	0	-4,14	0	-3,98	-9,85
13	-5,61	-5,09	-2,74	-11,94	-11,62	-3,48	-3,55	0	-5,72	0,05	0	-3,81	0	-3,14	0	-3,39	-9,11
14	-4,45	-4,13	-1,96	-10,66	-10,55	-2,7	-2,64	0	-4,41	0,01	0,01	-2,77	0,04	-2,11	0	-2,49	-8,25
15	-3,51	-3,5	-1,7	-10,03	-9,55	-1,65	-1,64	0,01	-3,77	0,01	0,08	-1,7	0,01	-1,11	0,02	-1,33	-8,26
16	-2,76	-2,39	0	-8,86	-8,57	-0,4	-0,59	0	-2,79	0,15	0,14	-0,8	0,2	-0,1	0	-0,62	-7,3
17	-1,4	-1,33	0	-8,48	-7,49	0	0	0,02	-1,37	0,89	0,03	0	0,07	0	0	0	-4,71
18	-0,74	-0,44	0	-6,54	-6,6	0	0	0	-0,48	1,89	0,02	0,01	0,22	0,01	0,04	0	-3,68
19	0	0	0	-5,7	-5,42	0	0,01	0,01	0	2,89	0,08	0,01	0,02	0,04	0	0	-2,6
20	0	0	0	-5,01	-4,7	0	0,06	0	0	0,61	0,35	0,01	0,07	0,01	0,01	0,01	-1,38
21	0	0	0	-3,98	-3,57	0	0,01	0,02	0	1,61	0,58	0	0,17	0	0,01	0,01	-0,56
22	0	0,01	0,01	-2,7	-2,39	0,03	0,05	0,02	0	2,61	0,57	0,03	0,05	0,02	0,31	0,13	0
23	0,03	0	0,02	-1,75	-1,79	0,01	0,02	0	0	3,61	0,55	0,03	0,11	0,07	0	0	0,01
24	0	0,04	0,03	-0,67	-0,81	0,03	0,02	0,16	0,1	4,61	0,65	0,21	0,56	0	0,04	0,05	0
25	0,01	0,08	0,21	0	0	0,01	0,04	1,16	0,01	5,61	1,91	0,11	1,56	0,07	0,16	0,03	0,1
26	0,02	0,04	0,18	0	0	0,01	0,02	2,16	0,04	6,61	2,71	0,08	2,56	0,02	0,29	0,02	0,08
27	0,08	0,25	0,01	0,02	0	0,04	0,03	3,16	0,01	7,61	0,39	0,11	3,56	0,05	0,2	0,06	0,08
28	0,01	0,02	0,2	0,01	0	0,03	0,04	0,08	0	8,61	0,02	0,13	0	0,12	0,32	0,01	0,07
29	0	0,02	0,14	0	0,02	0,01	0,07	0,37	0,05	9,61	0,03	0,12	0,04	0,21	0,16	0	0,02
30	0,08	0,23	0	0,08	0	0,08	0	0,49	0,04	10,61	0,04	0,12	0,34	0,13	0,07	0,03	0,05

Note: Green RRA's are our architectures, orange and blue ones are rival architectures and their modifications.

	Round Robin Arbiter *AREA* Comparison for *256Bit Input*																
Time (ns)	STA N2N	STA PPT	STA N2LOG N	CHN PPE Conflict	CHN PPE Conflict PPT	CHN PPE Non Conflict	CHN PPE Non Conflict PPT	BK_BT	BK_RS	HC_BT	HC_BT Simple Mux	HC_RS	KS_BT	KS_RS	LF_BT	LF_RS	BOWTIE
10	8129	8145	8289	8877	8999	8412	8140	6203	6235	6002	7293	6829	9322	7234	7669	6675	13693
11	8396	8285	8172	8938	9067	8382	8348	6369	6506	5689	7155	6644	9092	6689	7575	6505	13676
12	8238	8289	8421	8823	8915	8510	8136	6406	6595	5196	6477	6167	8402	6680	6923	6381	13673
13	8282	8199	8657	8791	9093	8791	8013	5703	6237	5046	6086	6158	7913	6986	6853	6889	13729
14	8077	8130	8364	8767	9072	8375	7937	5350	6107	5041	6003	6299	7642	6784	6801	6504	13724
15	7919	8074	7901	8738	9066	8523	7936	5259	5959	5041	5992	6738	7635	6785	6668	6386	13686
16	7897	8127	8180	8595	9338	8403	7872	5244	6144	5041	6039	6156	7632	6733	6402	6486	13665
17	8192	8111	7398	8544	9317	7353	7096	5239	6108	5041	5989	6031	7629	6215	6467	5967	13769
18	7783	8069	7236	9120	9123	7315	6794	5236	5899	5041	6203	5538	7628	6178	6397	5592	13862
19	7648	7003	6668	8899	9077	6574	6567	5233	5597	5041	5995	5356	7626	5976	6199	5495	13842
20	6877	6528	6450	8748	9077	6448	6485	5487	5054	5040	5988	5302	7880	6050	6130	5437	13820
21	6522	6358	6510	8840	9146	6205	6533	5232	4914	5040	5990	5328	7625	5970	6132	5419	13578
22	6216	6290	6357	8885	9771	6179	6520	5231	4882	5040	5987	5367	7624	5966	6141	5382	12908
23	6196	6221	6281	8704	9039	6149	6476	5231	4831	5040	5989	5283	7623	5964	6126	5356	12871
24	6069	6188	6264	8722	8569	6136	6471	5231	4774	5040	5984	5271	7623	6029	6110	5345	12820
25	6152	6194	6268	7977	8553	6095	6495	5231	4795	5040	5984	5320	7623	5962	6080	5342	12618
26	5899	6179	6259	7701	7660	6105	6477	5231	4789	5040	5984	5271	7623	5978	6063	5336	12376
27	6173	6298	6298	7404	7080	6137	6482	5231	4775	5040	5985	5266	7623	5972	6069	5340	12288
28	5948	6201	6294	7187	6996	6155	6483	5230	4769	5040	5986	5284	7623	5970	6055	5323	12307
29	6176	6214	6287	6810	6935	6144	6472	5230	4770	5040	5987	5281	7622	5961	6058	5327	12273
30	6143	6227	6296	6817	6886	6170	6476	5230	4762	5040	5988	5279	7622	5965	6056	5315	12267

Note: Green RRA's are our architectures, orange and blue ones are rival architectures and their modifications.

Figure 6.6: 256 Bit RRAs synthesis results

		Rivals			*Our Other Archs.*		*Proposed Arbiters*							
Bitwidth	**Criterion**	**STA N2N**	**CHN_PPE Conflict**	**CHN_PPE NonConflict**	**BOWTIE**	**STA LOGN2N**	**BK_BT**	**BK_RS**	**HC_BT**	**HC_RS**	**KS_BT**	**KS_RS**	**LF_BT**	**LF_RS**
8	Area	176	186	173	387	196	165	155	174	160	177	187	169	157
	Timing	9	11	9	10	9	8	10	8	10	8	9	8	10
16	Area	452	400	346	800	441	361	321	349	363	374	329	352	332
	Timing	12	14	11	12	10	9	12	9	11	9	11	9	11
32	Area	1409	1096	893	1663	899	721	628	701	920	785	808	875	878
	Timing	14	16	12	14	12	10	14	10	12	10	12	9	12
64	Area	1631	1842	1719	3276	2072	1500	1395	1680	1443	1835	1416	1631	1485
	Timing	15	20	14	17	13	11	15	10	14	10	14	10	14
128	Area	3480	4631	3786	6475	3641	2862	2916	3679	3285	3960	3364	3420	2915
	Timing	17	>20 ns	15	20	15	12	17	11	15	11	15	11	16
256	Area	7648	7977	7353	12908	8180	5703	5597	5689	6031	8402	6215	6923	5967
	Timing	19	25	17	22	16	13	19	11	17	12	17	12	17

Figure 6.7: Synthesis results at first positive or zero slack

7. CONCLUSION AND FUTURE WORK

In this thesis, we proposed two RRA architectures PPT_RRA_RS and PPT_RRA_BT. We also described another new architecture – BOW-TIE_RRA which is still under development. We optimized and implemented our rival architectures. Various modifications and optimizations are applied on these architectures to strengthen them. Then, all RRA architectures are verified and synthesized. According to synthesis results we proved that our proposed designs are area efficient and faster than the rival architectures. The fastest architecture is HC_PPT_RRA_BT and the most area efficient one is BK_PPT_RRA_RS with respect to 256-bit synthesis results. These results are simplified as follows.

According to 256-bit synthesis results, our new architectures achieve 22% area improvement and 42% timing improvement over optimized STA_RRA_N2N which is the well-known RRA design in literature. We optimized Smpl_PE block which is described in Chapter 2.

Two CHN_RRA designs were published in 2006. The authors tried to enhance the STA_RRA architecture. However, their architectures had some drawbacks. We applied some optimizations on their work. These optimizations are explained in Chapter 2. When we compared our proposed architectures' synthesis results against CHN_RRAs' results for 256-bit, we saw that we outperform them by factor in 35% for speed and 23% for area.

Savin C.E., McSmythurs T., and Czilli J. published BTS_RRA in 2004. In their paper, they represented that their architecture has ($\log_2 N + 4$) logic level, ($n\log_2 N + 7n - 6$) combinational gate count and it has n-bit flop. Our proposed architecture for best timing (with Ladner Fisher Pre_Thermo block — LF_PPT_RRA_BT) has ($\log_2 N + 4$) logic level too. However, its gate count is equal to ($n\log_2 N + 4n$) + n-bit flop. Thus, our proposed architecture's gate count is better than BTS_RRA.

The most recent work was carried out by Zheng S. Q. and Yang M. in January 2007. They proposed two architectures: PRRA and IPRRA. They indicated that IPRRA

achieved 30.8% timing improvement and 66.9% area improvement over PPE design which is the core block of STA_RRA. Our proposed design for best timing achieved 42% timing improvement over STA_RRA, which is better than both IPPRA and PPRA.

Other two different architectures SA and PPA use different algorithms and they cannot ensure the fairness for non-uniformly distributed requests. Thus, we did not implement these architectures for comparison.

In summary, PPT_RRA_BT and PPT_RRA_RS architectures both achieve important improvements in area and speed departments compared with former RRA architectures. Due to their performance results, these architectures will play important roles for high-speed switches, routers, and the systems where arbitration is required for various purposes.

It is possible to enhance the area and timing results of BOW-TIE_RRA architecture. It consists of complex RR Blocks. So, in the future we will work on to reduce the RR Blocks' complexity of that architecture. Also, unimplemented RRA architectures will be implemented for area and timing comparison.

REFERENCES

Books

Koren, I., 2001. *Computer Arithmetic Algorithms*, 2nd edition, A K Peters Ltd.

Synopsys, Inc.,2007. *Design Compiler 1 Student Guide*, v2007.03

Synopsys, Inc.,2008. *Prime Time 1 Student Guide*, v2006.06

Periodical Publications

Gupta, P. & McKeown, N., 1999. Designing and implementing a fast crossbar scheduler, *Micro IEEE,* **19** (1), pp. 20 – 28.

Chao, H.J., Lam, C.H. & Guo, X., 1999. A fast arbitration scheme for terabit packet switches, *Global Telecommunications Conference, 1999. GLOBECOM '99, pp.* 1236 – 1243.

Shin, E.S., Mooney, V.J. III & Riley, G.F., 2002. Round-robin Arbiter Design and Generation, *15th International Symposium on System Synthesis,* pp. 243 – 248.

Lee, K., Lee, S.-J. & Yoo, H.-J., 2003. A high-speed and lightweight on-chip crossbar switch scheduler for on-chip interconnection networks, *ESSCIRC '03. Proceedings of the 29th European Solid-State Circuits Conference,* pp. 453 – 456

Yoghigoe, K., Christensen, K.J. & Roginsky, A., 2003. Design of a high-speed overlapped round robin (ORR) arbiter, *28th Annual IEEE International Conference on Local Computer Networks,* pp. 638 – 639.

Savin, C.E., McSmythurs, T. & Czilli, J., 2004. Binary tree search architecture for efficient implementation of round robin arbiters, *(ICASSP '04). IEEE International Conference on Acoustics, Speech, and Signal Processing,* **6** (5), pp. V – 333.

Gao, X., Zhang Z. & Long X., 2006. Round Robin Arbiters for Virtual Channel Router, *IMACS Multiconference on Computational Engineering in Systems Applications,* pp. 1610 – 1614.

Si Q. Z. & Mei Y., 2007. Algorithm-Hardware Codesign of Fast Parallel Round Robin Arbiters, *IEEE Transactions on Parallel and Distributed Systems*, **18** (1), pp. 84 – 95.

Harris, D. 2003. A taxonomy of parallel prefix Networks, *Conference Record of the Thirty-Seventh Asilomar Conference on Signals, Systems and Computers,* pp. 2213 – 2217.

Other Publications

Shin, E.S., 2003. Automated Generation of Round-Robin Arbitration and Crossbar Switch Logic. *PhD Thesis*. Georgia Institute of Technology, School of Electrical and Computer Engineering.

Weber, M. 2001. Arbiters: Design Ideas and Coding Styles, *SNUG Boston*, Silicon Logic Engineering, Inc.

Kuphaldt T. R., CMOS Gate Circuitry [online], http://www.allaboutcircuits.com/vol_4/chpt_3/7.html, [citied 2 May 2008]

VITA

Name Surname : Onur BAŞKİRT

Address : STMicroelectronics Elektronik Araştırma ve Geliştirme A.Ş.
Büyükdere Cad. İTÜ Ayazağa Kampüsü
Koru Yolu ARI2 B Blok Ofis No: 3–1 34469
Maslak, Sarıyer – İSTANBUL

Birth Place / Year : Kars / 1982

Languages : Turkish (native) – English

Elementary School : Ahmet Paşa Elementary School – 1994

High School : N. M. Baldöktü Anatolian High School – 2000

BS : Bahçeşehir University – 2005

MS : Bahçeşehir University – 2008

Name of Institute : Institute of Science

Name of Program : Electrical & Electronics Engineering

Publications : H.F. Uğurdağ, Y. Şahin, **O. Başkirt**, S. Dedeoğlu, S. Gören, Y.S. Koçak, "Population-based FPGA Solution to Mastermind Game," Proceedings of the IEEE/NASA Adaptive Hardware and Systems Conference, Istanbul, June 2006.

L. Eren, **O. Başkirt**, and M. J. Devaney, "Rule Based Motor Fault Detection," Proceedings of the IEEE Instrumentation and Measurement Technology Conference, Ottawa, May 2005.

Work Experience : STMicroelectronics
Connectivity Division – Digital IC Design Engineer
September 2007 – Ongoing

University of California, Santa Cruz
School of Engineering – Visiting Researcher
June 2006 – August 2006

Bahçeşehir University
EE Engineering Department – Teaching & Research Asst.
September 2005 – August 2007