MULTI-PICK ROUND ROBIN ARBITER

A Thesis

by

Fatih Temizkan

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in the
Department of Electrical and Electronics Engineering

Özyeğin University
August 2012

MULTI-PICK ROUND ROBIN ARBITER

Approved by:

Asst. Prof. H. Fatih Uğurdağ, Advisor
Department of Electrical and Electronics Engineering
Özyeğin University

Professor A. Tanju Erdem
Department of Electrical and Electronics Engineering
Özyeğin University

Asst. Prof. T. Barış Aktemur
Department of Computer Science
Özyeğin University

Date Approved: 27 August 2012

To my family

ABSTRACT

In this thesis, we propose two multi(m)-pick Round Robin Arbiter (RRA) architectures. An m-pick RRA selects the m topmost requests out of n inputs with priority order indicated by an internally kept pointer (with an update policy that ensures fairness among requestors). The architectures that we propose are m-pick Thermo Coded-Parallel Prefix Arbiter (TC-PPA) and Three-Dimensional Programmable m-Selector RRA (3DPmS-RRA). Prior to this thesis, these two architectures existed in the literature as 2-pick and 1-pick arbiters, respectively. Our main contribution to the literature is the generalization of these architectures to m-pick. A logic building block that we call "Saturated Adder" plays a key role in this generalization, which makes the 1-pick and 2-pick architectures simply special cases. We developed six different variants of 3DPmS-RRA and eight different variants of m-pick TC-PPA. We wrote automated HDL code generators for all variants as well as Cascade Architecture, which is a straight-forward way of implementing a multi-pick RRA using 1-pick Programmable Priority Encoders. Then, all multi-pick architectures were verified and synthesized. Our experimental results show that 3DPmS-RRA architecture is the best choice for all pick sizes (except 2-pick) when timing is the primary design criterion. However, when area is more critical, TC-PPA architecture performs better. It is worthwhile to note that in terms of timing 3DPmS-RRA is better than TC-PPA by a mere 8% at the most based on our synthesis results. However, when we consider area, TC-PPA has significant improvements over 3DPmS-RRA, up to 53%.

ÖZETÇE

Bu tezde, iki tane çoklu(m)-seçim dönmeli iş düzenleyici (RRA) mimarisi öneriyoruz. m-seçim RRA, içinde tuttuğu işaretçi tarafından belirlenen öncelik sırasına göre n tane giriş arasından ilk m tane isteği seçer (istemciler arasında adaleti sağlayan güncelleme politikası ile). Önerdiğimiz mimariler TC-PPA ve 3DPmS-RRA'dır. Bu tezin öncesinde, bu iki mimari literatürde sadece 2-seçim ve 1-seçim (sırasıyla) iş düzenleyiciler olarak vardı. Bizim literatüre en önemli katkımız bu mimarilerin m-seçime genellenmesidir. Bu genellemede "Sınırlanmış Toplayıcı" olarak adlandırdığımız mantık yapı elemanı anahtar bir rol oynamakta ve 1-seçim ile 2-seçim mimarileri kendisinin bir özel durumuna dönüştürmektedir. 6 farklı 3DPmS-RRA ve 8 farklı m-seçim TC-PPA versiyonu geliştirdik. Çoklu-seçim iş düzenleyiciyi gerçekleştirmenin en direkt yolu olan 1-seçim PPE'lerden gerçekleştirilen Kademeli Mimari ile birlikte bütün versiyonlar için otomatik HDL kod üreteci yazdık. Sonra bütün çoklu seçim mimarileri doğrulandı ve sentezlendi. Deney sonuçlarımız, öncelikli tasarım kriteri zamanlama olduğunda 3DPmS-RRA mimarisinin, bütün seçim değerleri için en iyi seçenek olduğunu gösteriyor (2-seçim hariç). Ancak alan daha önemli olduğunda, TC-PPA mimarisinin daha iyi olduğunu görüyoruz. Şu da dikkate değer ki, sentez sonuçlarımıza göre zamanlama açısından 3DPmS-RRA mimarisinin TC-PPA'den en fazla %8 daha iyi olduğu ortaya çıkmaktadır. Ama alanı düşündüğümüzde TC-PPA, 3DPmS-RRA'ya karşı %53 gibi ciddi bir iyileştirme sağlamaktadır.

ACKNOWLEDGEMENTS

This thesis would not have been possible without the guidance and the help of several individuals, who in one way or another, contributed and extended their valuable assistance in the preparation and completion of this work.

First, I want to thank my advisor, Asst. Prof. H. Fatih Uğurdağ, for his invaluable guidance, support, and motivation to complete my M.S., and of course my thesis. He was not only my advisor, he was also sometimes my friend, sometimes a family member, and sometimes my boss. I appreciate that I had a chance to work with him. I also want to thank Asst. Prof. A. Turan Özdemir for his guidance during my B.S. and for introducing me to Dr. Uğurdağ.

I would like to thank Prof. A. Tanju Erdem and Asst. Prof. T. Barış Aktemur, my thesis committee members, for taking the time.

I would also like to thank Assoc. Prof. Sezer Gören Uğurdağ for her involvement in the later stages of this thesis and for her constructive feedback, especially on the thesis report. Without her help, it would not be possible for me to complete this thesis.

I also want to thank my colleagues that were with me throughout my graduate education especially Gökhan Güner, Abdullah Yıldız, Bilgiday Yüce, Cihan Tunç, Okan Palaz, Asım Yıldız, Ayşe Karagöz, Mustafa Kaygısız, and Erdem Ulusoy for their friendship.

In addition, I would like to thank TÜBİTAK for supporting me through BİDEB scholarship program between 2009-2011 and through ARDEB in 2011-2012.

Finally, I would like to thank my family members for their endless support throughout my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

In this chapter, main concepts are given to familiarize the readers with the subject of this thesis. This chapter also includes previous work, summary of our contributions, and outline of the thesis.

1.1 What is a Round Robin Arbiter?

Arbiters are used to multiplex the usage of shared resources among requesters as well as in dispatch logic where the purpose is load balancing among multiple resources. If any system has a shared resource and multiple requesters which want to use that shared resource, a Round-Robin Arbiter (RRA) is used to coordinate the usage of this shared resource among requesters, when fairness to all requesters is important for the system. Here shared resources may be, for example, ports, buses, ALUs, memory blocks, etc. On the other hand, requesters may be CPU cores, ingress packet queues, etc.

An $n2n$ RRA searches its n inputs for a 1, starting from the highest-priority input. It picks the first 1 and outputs its index in one-hot encoding. RRA aims to be fair to its inputs and maintains fairness by simply rotating the input priorities, i.e., the last arbitrated input becomes the lowest-priority input. A typical RRA [1] resolves the index of the highest-priority request out of n inputs, starting from the highest-priority position where a constantly updated pointer points to. RRA updates its highest-priority pointer immediately next to the last arbitrated input. Each requester has nearly equal priority to be selected by the RRA, and this is the key difference between a Fixed Priority Encoder (FPE) and an RRA. FPE does not have a constantly updated pointer, but RRA does. In RRA, the input priorities

dynamically change whenever the pointer is updated next to the last arbitrated input. As a result of this, fairness is achieved. Fig. 1 shows two consecutive cycles for an RRA scenario.

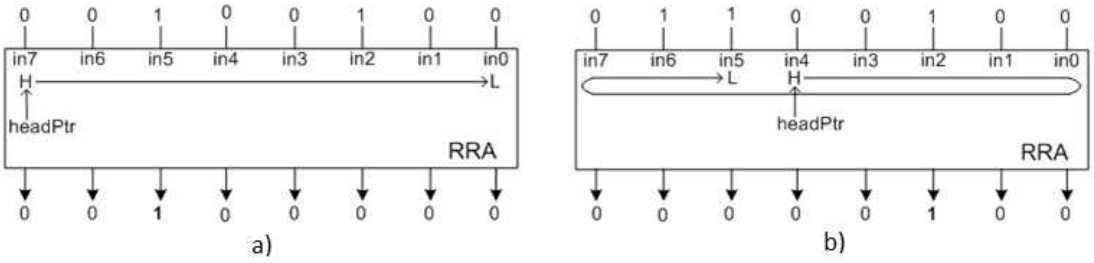

Figure 1: (a) One cycle and (b) Next cycle of the RRA

In Fig. 1a and 1b, H denotes the position of the highest-priority input, whereas L denotes the position of the lowest-priority input. RRA starts searching from H to L to find the highest-priority request at the moment. When RRA locates the highest-priority request, RRA grants it, and ignores the rest of the requests. Hence, for the input pattern shown in Fig. 1a, RRA grants request 5. For the next cycle, RRA updates H to 4, which is the location next to the granted input 5. For the next input pattern shown in Fig. 1b, RRA grants request 2, updates H to 1, and keeps working like this in the cycles to follow.

Note that, in our terminology, we call a typical RRA described in this section as a 1-pick RRA, because it grants only one request out of n requests in every cycle.

1.2 What is Multi-Pick RRA?

Similar to 1-pick RRA, m-pick (multi-pick) RRA resolves the indexes of m highest priority active requests out of n inputs with a constantly updated pointer. Here, m can be between 1 and n. In this thesis, we use the terms "multi-pick" and "m-pick" interchangeably. For example, if we take m=2, RRA is a 2-pick RRA according to our terminology and grants 2 requests in every cycle. If we take m=3, this time it will be a 3-pick RRA. Fig. 2 shows two consecutive cycles for a 2-pick RRA example.

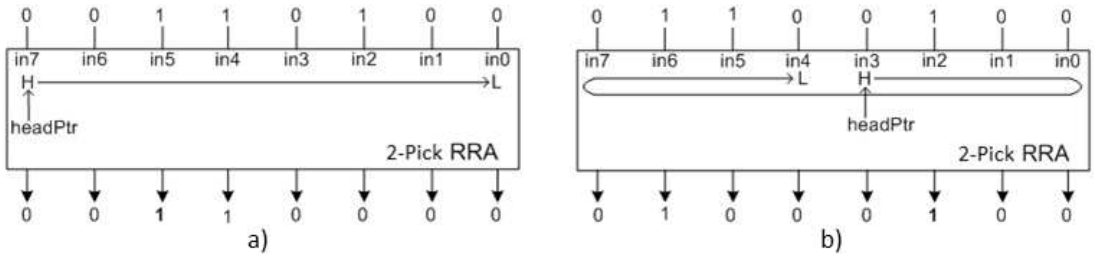

Figure 2: (a) One cycle and (b) Next cycle of the 2-Pick RRA

In Fig. 2a, requests are at input locations 5, 4, and 2. The 2-pick RRA grants two highest-priority active requests. Based on the position of H in Fig. 2a, input 5 and input 4 are granted by the 2-pick RRA. Then 2-pick RRA updates H to the location next to the location of the second granted request. For this example, H becomes 3. In the next cycle, given the input pattern shown in Fig. 2b, 2-pick RRA grants requests 2 and 6, and again updates its pointer to 5 (location next to the location of the second granted request (=5)).

1.3 Where is Multi-Pick RRA used?

While a typical 1-pick RRA grants one requester to use a shared resource every clock cycle, higher throughput systems require the use of multiple shared resources per cycle. That results in a need for multi-pick RRAs. If there are m shared resources, m-pick RRA is used to coordinate the usage of them. An example application of multi-pick RRA (2-pick RRA) is given in Fig. 3.

In Fig. 3, there are 2 shared buses and 5 requesters that want to use these shared buses. However, in any clock cycle, only 2 of the requesters can use the buses. Hence, 2-pick RRA determines which requester will use which bus. For this example shown in Fig. 3, requesters 3, 2, and 0 send request to a 2-pick RRA, and 2-pick RRA grants the two highest-priority active requests (3 and 2) to use the shared buses. For the next cycle, 2-pick RRA rotates its pointer to the location next to the second granted request in order to be fair to all requestors.

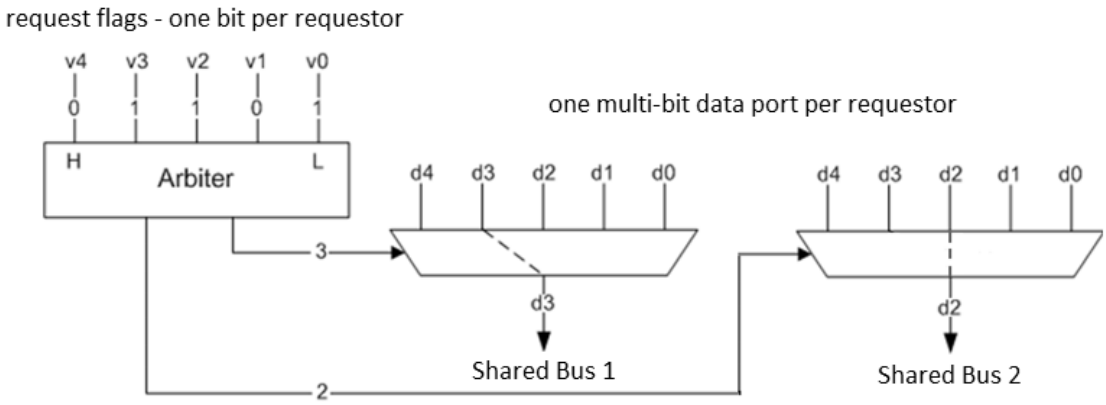

Figure 3: Example usage of a 2-Pick RRA

1.4 Previous Work

There is a plethora of work on 1-pick RRAs. The pioneering work on 1-pick RRA from a logic optimization point of view is by Gupta and McKeown [2, 3]. Gupta-McKeown's RRA consists of a Programmable Priority Encoder (PPE) and a pointer update logic block. Gupta-McKeown's initial PPE is a ripple PPE. Later, they decided that the ripple PPE has two major problems:

i. presence of combinational loop and ii. very long critical path.

Then, they proposed a divide and conquer approach by converting the RRA problem into two FPE problems by breaking up the wrap-around path. Moreover, Gupta-McKeown's second PPE has FPEs embedded in them, and as a result, high-performance FPE design becomes critical when building the RRA. An FPE is sometimes simply called PE (Priority Encoder) and can be thought of as if it has a fixed pointer. Gupta-McKeown's RRA work [2, 3] spurred interest and led to four works [4, 5, 6, 7]. Extensive review of these architectures including Gupta-McKeown's RRA can be found in [8]. Gupta et al. also discovered the existence of a similar data flow in this problem to an adder. They applied carry propagation techniques only to their ripple PPE. They did not mention using carry-lookahead (CLA) in the FPEs of their faster architecture. However, FPEs can greatly benefit from smart carry

propagation. Recently, Ugurdag and Baskirt [1] applied several Parallel Prefix Network (PPN) topologies to demostrate the benefit of smart carry propagation in FPEs with their (1-pick) TC-PPA architecture. TC-PPA stands for Thermo Coded-Parallel Prefix Arbiter. Fig. 4 depicts 1-pick TC-PPA architecture.

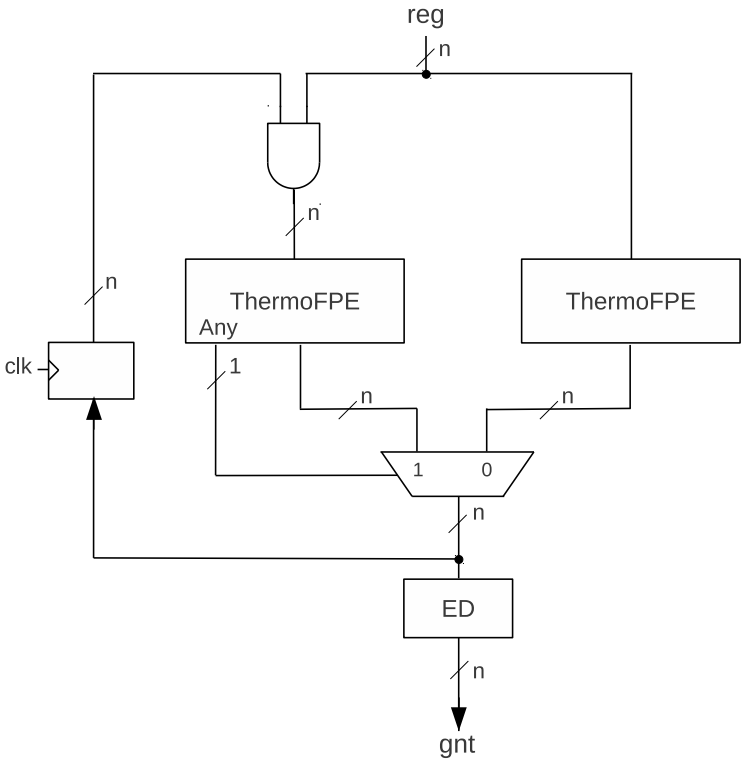

Figure 4: 1-pick TC-PPA architecture. Source: Ugurdag et al. (2012) [1].

Unlike 1-pick, the only work on multi-pick RRA was 3DP2S by Ahn et al. [9], and it is only on the 2-pick problem. 3DP2S stands for 3-Dimensional Programmable 2-Selector and is a PPE rather than an RRA. In our definition, a PPE takes the priority pointer as an external input, whereas a typical RRA manages it internally and updates it every clock cycle to achieve fairness.

In Fig. 5, top level architecture of 3DP2S (a) and unit block (UB) logic (b) are given. 3D2PS consists of two planar PPEs, which are built from Gupta and McKeown's ripple PPE. The topology used in 3DP2S is KS PPN. Upper planar PPE is responsible for selecting the first request and lower one is responsible for selecting

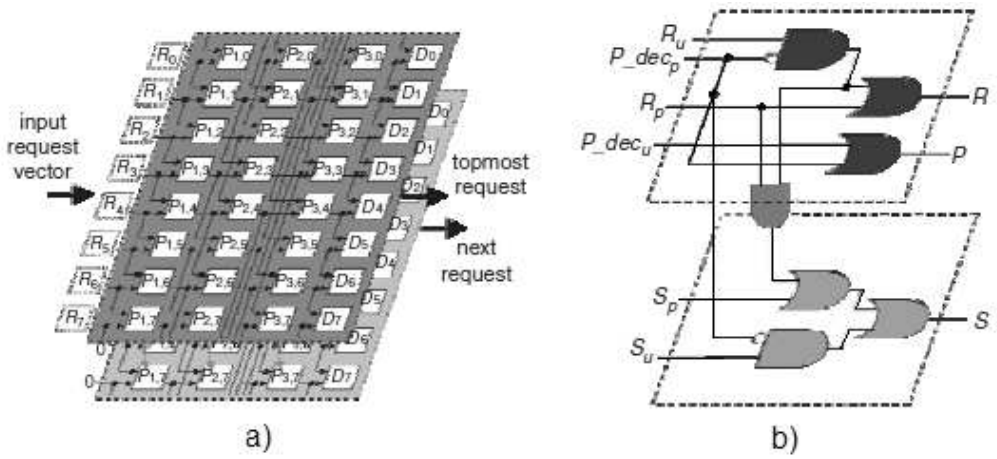

Figure 5: (a) Top level architecture of 3DP2S and (b) Logic inside Unit Block (UB) Source: Ahn et al. (2004) [9].

the second request. UB contains glue logic to transfer intermediate results from upper plane to lower plane. Ahn et al. [9] showed that 3DP2S requires only one-third more delay time to process two requests than Gupta-McKeown's ripple PPE.

1.5 Contributions of the Thesis

Fundamentally, this thesis builds upon two works in the literature, namely, 3DP2S by Ahn et al. [9] and TC-PPA by Ugurdag and Baskirt [1]. 3DP2S is a 2-pick PPE, and TC-PPA is a 1-pick RRA. 3DP2S, however, is not exactly an RRA in terms of our RRA definition, because 3DP2S takes a priority pointer as an external input instead of maintaining it internally. TC-PPA uses PPN topologies (normally used in fast adders for carry lookahead) to generate a thermometer-coded pointer, thus greatly reducing critical path. TC-PPA has a flexible architecture such that it can employ any PPN topology. Four pioneering PPNs such as Brent-Kung (BK) [10], Han-Carlson (HC) [11], Kogge-Stone (KS) [12], and Ladner-Fischer (LF) [13] are employed in TC-PPA, whereas only KS PPN is employed in 3DP2S.

Contributions of this thesis are the following:

- We propose a 3DP2S-based RRA (3DP2S-RRA), which turns 3DP2S into an RRA.
- We found and fixed a bug in 3DP2S.
- We generalize 3DP2S-RRA from 2-pick to m-pick RRA (which grants up to m requests) and call it as 3DPmS-RRA.
- We generalize TC-PPA to m-pick TC-PPA, whose output is up to m-hot, i.e., at most m of the n outputs (grants) are hot, where there are n requestors as input. We employed all four PPN topologies [10, 11, 12, 13] in our m-pick RRA implementations.
- In addition to 3DPmS-RRA and m-pick TC-PPA, we also implemented a 'cascade' method. For this, we first implement a 2-Dimensional Programmable 1-Selector (2DP1S) which is a 1-pick PPE. Using the cascade method, we implement m-pick-Cascade-RRA using 2DP1S modules.
- We developed HDL code generators for all the RRA circuits and their variants mentioned above.
- We obtained a rich set of timing/area results using a standard-cell based logic synthesis flow with a novel iterative strategy.

1.6 Outline of the Thesis

Chapter II presents 3DP2S-RRA, our 3DP2S bug fix, and 3DPmS-RRA. Chapter III presents m-pick-TC-PPA and its four different PPN variants. Chapter IV presents m-pick-Cascade-RRA using 2DP1S modules. Chapter V describes our Verilog code

generators for all proposed RRA architectures and their variants, and Chapter VI describes our experimental set-up based on HDL code generators as well as our standard-cell based logic synthesis flow with a novel iterative strategy. Chapter VI also presents timing/area synthesis results for all automatically generated RRA circuits. Finally, we evaluate our experimental results and also discuss possible future work in Chapter VII.

CHAPTER II

MULTI-PICK RRA BASED ON 3DP2S

In this chapter, we first present 3DP2S-RRA [14] and our view of microarchitecture of 3DP2S. Then, we generalize 3DP2S for any m to construct m-pick PPE, which we call 3-Dimensional Programmable m-Selector (3DPmS), and finally our 3DPmS-RRA with its six different variants.

2.1 3DP2S-RRA

As we mentioned in previous sections, 3DP2S is not an RRA, but a PPE. Hence, we convert 3DP2S to a 2-pick RRA by adding a circuit around it to update the highest-priority pointer internally, because 3DP2S takes the pointer as an external input. The 3DP2S-RRA architecture is presented in Fig. 6, where it consists of a 3DP2S module and the pointer update logic. (Note: Little cones either split up wires or combine them into buses.)

In Fig. 6, Hptr represents the highest-priority pointer. The internal signals gnt1 and gnt2 represents the first and second selected requests by 3DP2S respectively. The output of the circuit is obtained after OR2 of gnt1 and gnt2. Our interpretation of 3DP2S microarchitecture is given in Fig. 7a. In Fig. 7a, 3DP2S block has a KS PPN topology [12] and it consists of unit blocks (UB). The logic inside UB is also depicted in Fig. 7b. Note that Fig. 7b presents our interpretation of UB of 3DP2S, which consists of a thermometer-coded adder saturated at 2. That is, UB treats (S and R bits of) its left and right inputs as two 2-bit unsigned numbers and adds them to produce a 2-bit unsigned output. The UB is an adder saturated at 2, hence out = min(inLeft+inRight,2). It is thermometer-coded, i.e., 0, 1, 2 are, respectively, coded as 00, 01, 11. Table 1 shows the truth table for the thermometer-coded adder

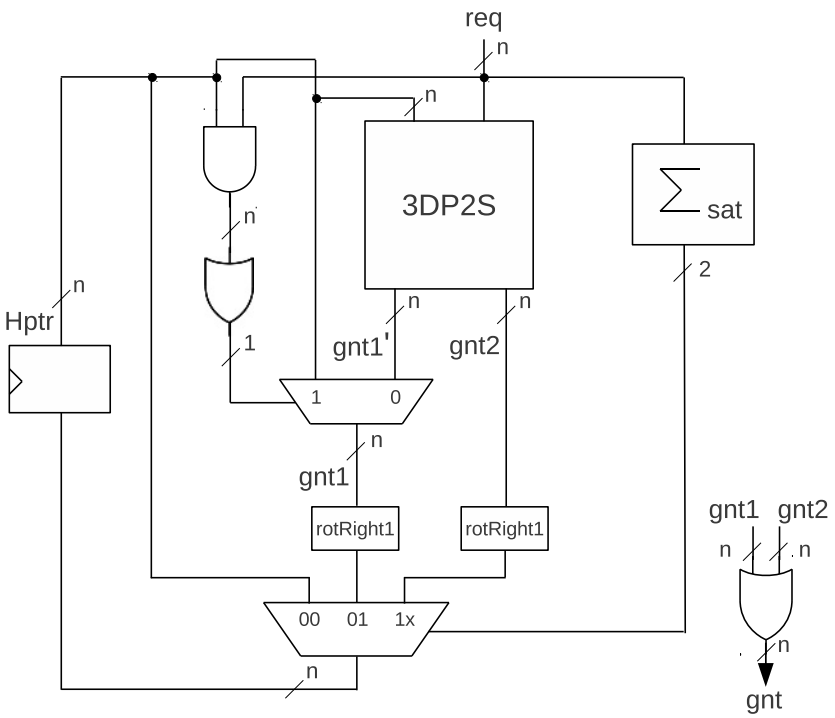

Figure 6: 3DP2S-RRA architecture.

saturated at 2. The UB also takes two pointer bits (in the middle bit position of its left and right inputs) and outputs an OR of them (in the middle bit position). The UB, which Hptr points to, ignores its right input and hence acts as the highest priority position. Our circuit around 3DP2S in Fig. 6 determines whether there are 2, 1, or 0 active requests, then updates the pointer accordingly. This is done through the subcircuit that feeds the select of MUX3. There are 3 different cases to update pointer for 2-pick RRA depend on how many requests there are at the input. These cases are the following:

- 'Two requests found' requires us to update the pointer with the position to the right (hence rotRight1) of the second requestor picked (Hptr = gnt2).
- 'Only one request found' requires rotRight1 of the first requestor found (Hptr = gnt1).

- 'No request found' requires us to recirculate the existing pointer (Hptr = Hptr).

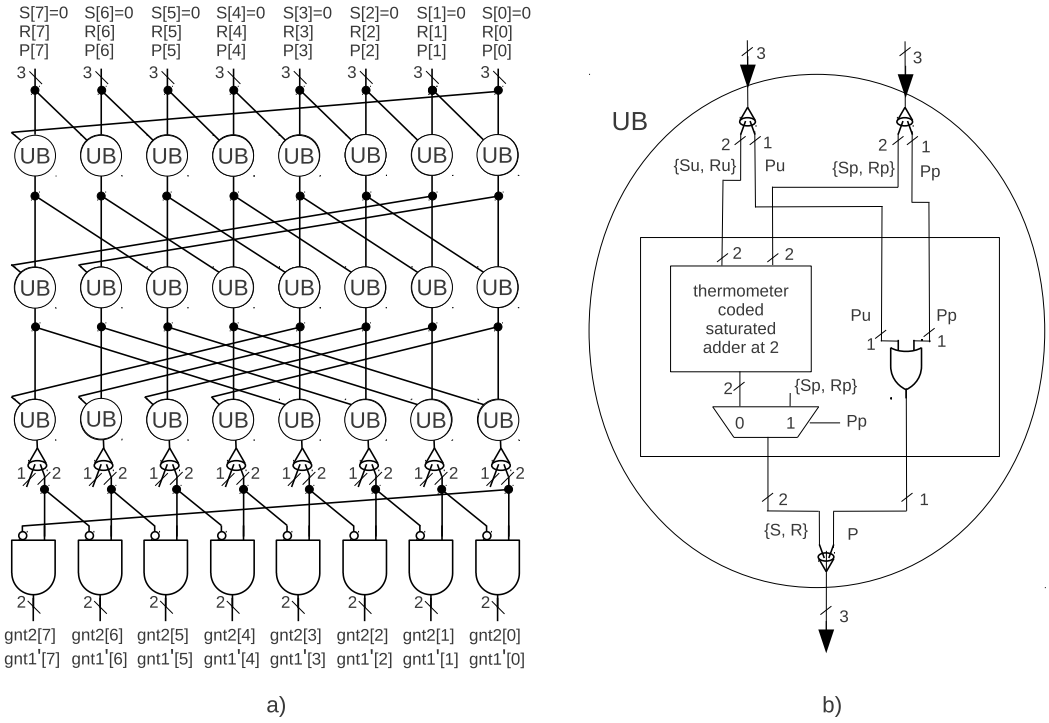

Figure 7: (a) Microarchitecture of 3DP2S and (b) UB logic

Table 1: Truth table for the thermometer-coded adder saturated at 2

IN1[1:0]	**IN2[1:0]**	**OUT[1:0]**
00	00	00
00	01	01
01	00	01
01	01	11
11	X	11
X	11	11
10	X	X
X	10	X

2.2 Bug Fix in 3DP2S

There was a bug in Ahn's 3DP2S and we fixed it. 3DP2S does not work when there is a request at the position to which Hptr points. The fix is implemented with the n-bit AND2, 1-bit ORn, and n-bit MUX2 in Fig. 6. If there is a request at the position to which Hptr points, the output of the AND2 is a one-hot vector, otherwise it is all zeros. The ORn outputs a 0 when it is all zeros and 1 otherwise. Signal gnt1 is the corrected version of gnt1′. Fig. 8 shows an example to see how internal signals of 3DP2S change, when R (requests) is equal to 8'b10101011 and P (pointer) is equal to 8'b00010000.

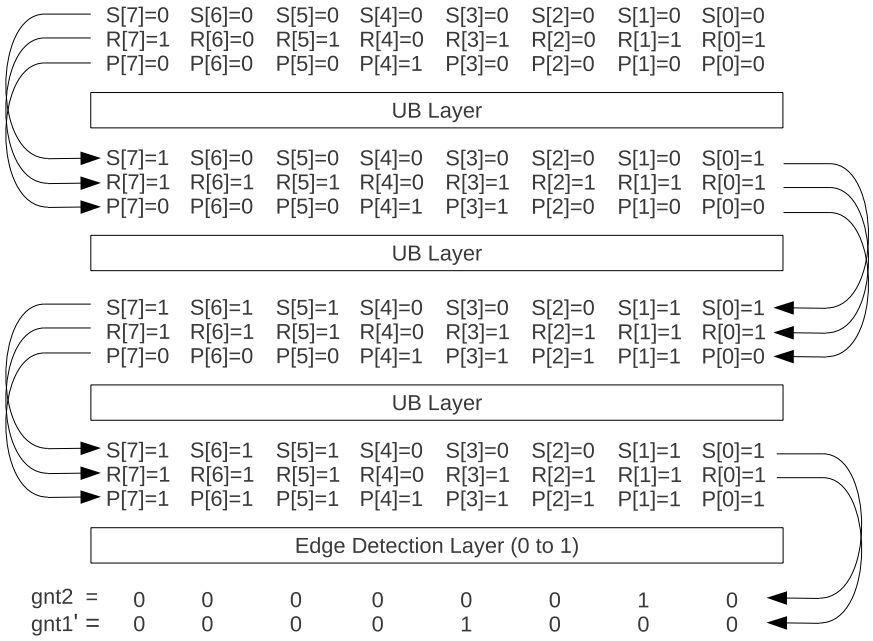

Figure 8: An example to show how 3DP2S works.

2.3 3DPmS-RRA

We extended 3DP2S to 3DPmS, which grants up to m highest priority requests. To do that, instead of the adder saturated at 2, we need an adder saturated at m in UB. Fig. 9 presents 3DPmS-RRA. In Fig. 9, gnt1, gnt2, ..., gnt$m$ are the outputs of 3DPmS. An up to m-hot coded output, gnt, is generated with n-bit ORm.

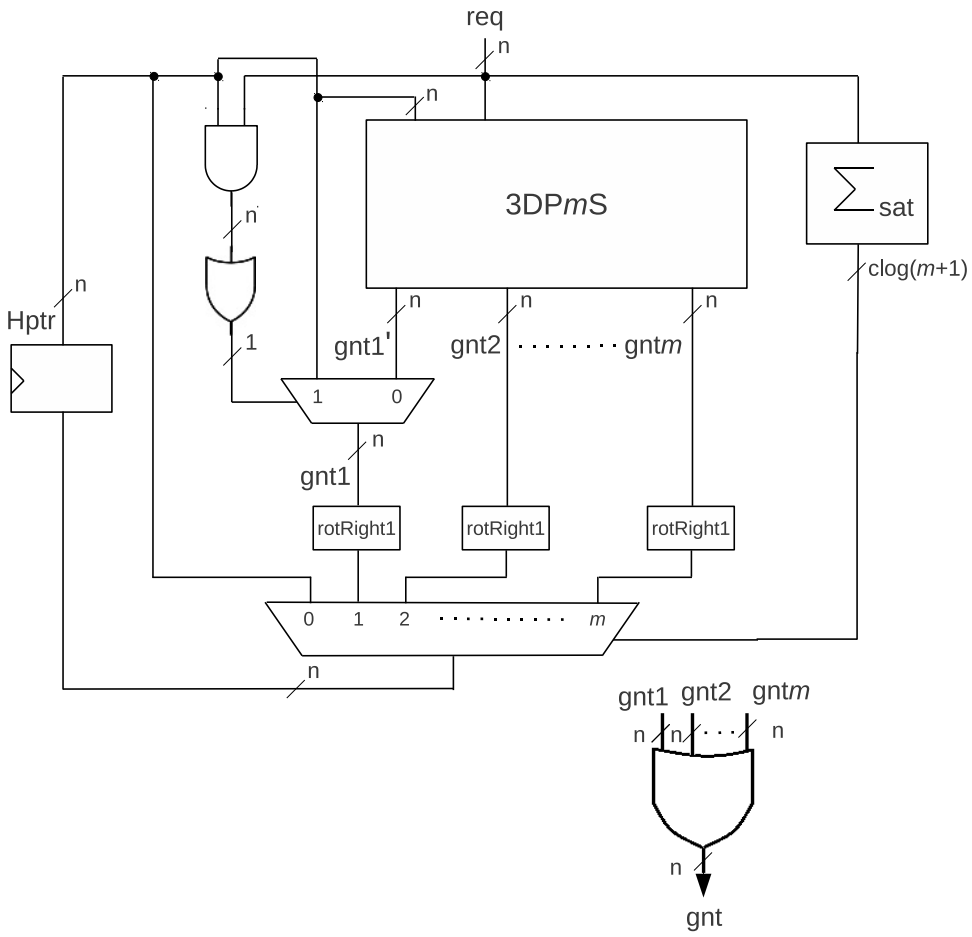

Figure 9: 3DPmS-RRA architecture.

The adder shown in Fig. 9 is the saturated adder at m and gives us the count of the requests. Output of this adder has a key role in our pointer update circuit. For 3DPmS, there are m+1 different update choices, hence we replaced MUX3 with MUX(m+1). Output of adder saturated at m feeds the select of MUX(m+1). Also,

we replaced final OR2 with ORm to get the final result. The bug fix in 3DP2S-RRA is also kept in 3DPmS-RRA.

Fig. 10a presents the microarchitecture of 3DPmS, very similar to Fig. 7a. Fig. 10b presents the logic inside UB where a thermometer-coded saturated adder at m is instantiated. Also note that R1[ ] denotes requests and R2[ ] denotes S[ ] of Fig. 7a for $m = 2$. Rm[ ], $\cdots$, R2[ ] are assigned all zeroes.

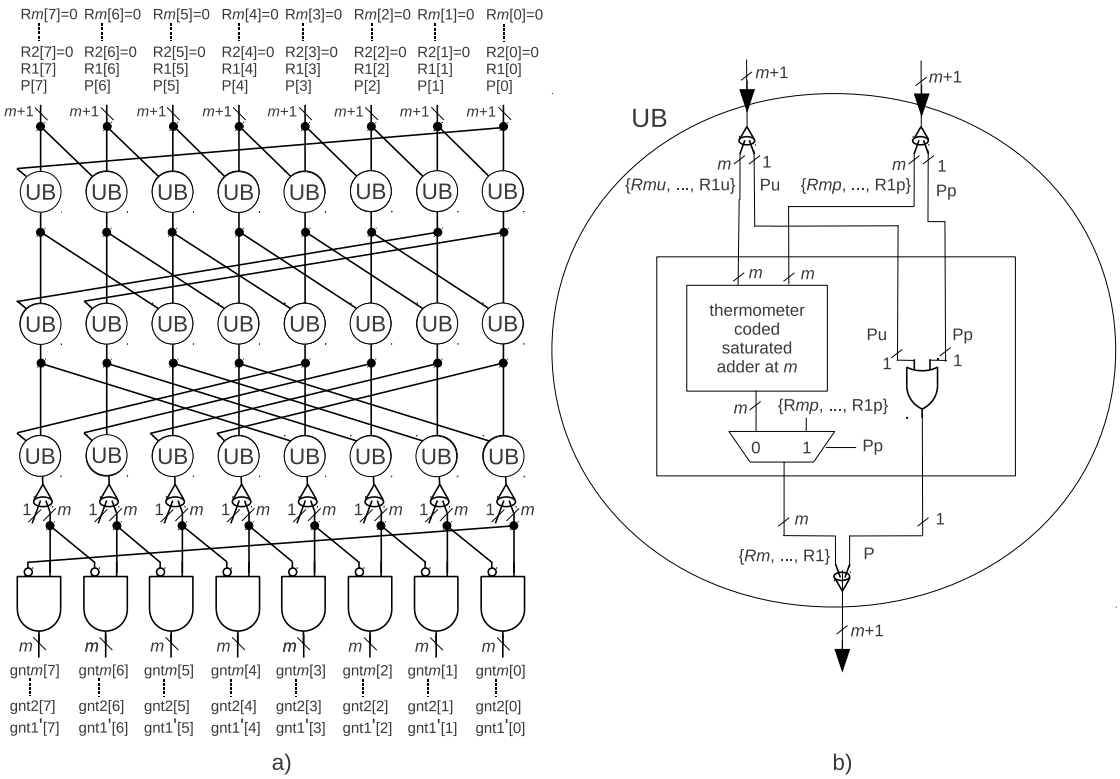

Figure 10: (a) Microarchitecture of 3DPmS, (b) UB Logic (thermometer-coded adder saturated at m).

2.4 Binary and Thermo Variants of UB

In 3DP2S-RRA, UB is implemented with a thermometer-coded adder saturated at 2. It is also possible to replace them with binary adders. Hence, we also implemented binary version of the UBs to compare their performance in synthesis. We have foreseen that RRA implemented with binary adders UBs could give better results for high

input bit-widths in terms of area and/or timing, because synthesis tool can easily map special fast adder to binary adders.

```
module unitBlock(Pu,Pp,P,Ru,Rp,Su,Sp,R,S);
input Pu,Pp;
input Ru,Rp,Su,Sp;
output P;
output reg R, S;

wire [1:0] SRu, SRp;

assign SRu = {Su,Ru};
assign SRp = {Sp,Rp};

assign P = Pu | Pp;

always @(*) begin
    if(~Pp)
        case({SRu,SRp})
            4'b00_00 : {S,R} = 2'b00;
            4'b00_01 : {S,R} = 2'b01;
            4'b01_00 : {S,R} = 2'b01;
            default : {S,R} = 2'b11;
        endcase
    else
        {S,R} = SRp;
end
```

a)

```
module unitBlock(Pu,Pp,P,Ru,Rp,Su,Sp,R,S);
input Pu,Pp;
input Ru,Rp,Su,Sp;
output P;
output reg R, S;

wire [1:0] SRu, SRp;

assign SRu = {Su,Ru};
assign SRp = {Sp,Rp};

assign P = Pu | Pp;

always @(*) begin
    if(~Pp)
        if((SRu+SRp)>2)
            {S,R} = 2;
        else
            {S,R} = SRu+SRp;
    else
        {S,R} = SRp;
end

endmodule
```

b)

Figure 11: Verilog of UB in 3DP2S-RRA (a) thermo and (b) binary.

Fig. 11 shows Verilog code for UBs, which are implemented with a) the thermometer-coded and b) the binary version of the saturated adder. Fig. 12a depicts the microarchitecture of 3DPmS-RRA that is implemented with UBs having binary-coded adders saturated at m. Binary-coded adders saturated at m take two $\lceil \log(m+1) \rceil$-bit inputs. Note that 'clog' denotes the 'ceiling log' function in Fig. 12. In Fig. 12a, an extra layer of binary to thermometer-code converters is placed after the last UB layer. The edge-detection (last) layer is as same as the one in Fig. 10a.

2.5 Speeding Up Pointer Update Circuit

According to our first synthesis results, we found out that the critical path of the circuit comes from the adder (shown as $\sum_{sat}$ in Fig. 6 and 9) in the pointer update circuit. This adder is saturated at m.

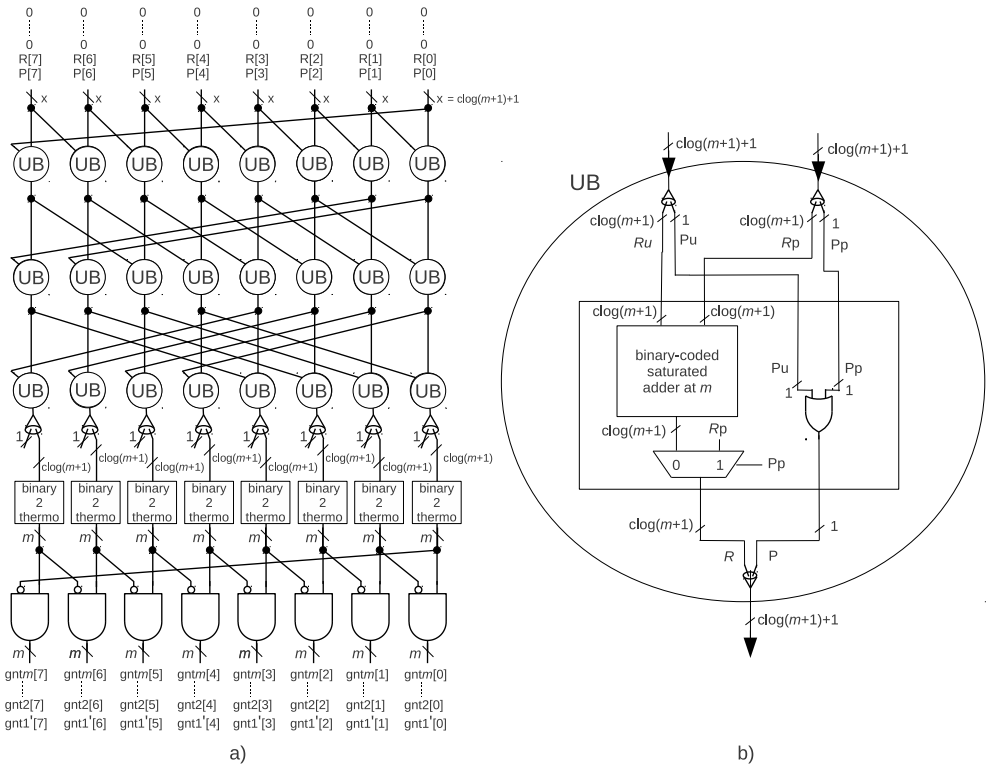

Figure 12: (a) Microarchitecture of 3DPmS, (b) Logic inside (binary adder saturated at m) UB.

Hence, to speed up, we implement the adder of the pointer update logic based on:

- Carry Save Tree (CST) [15]. CST is a compression tree and used for implementing fast adders and multipliers.
- 3DPmS modified as an FPE, i.e., we use only the PPN part of the 3DPmS to implement the saturated adder.
- RTL (high-level coded).

2.6 3DPmS Variants for Experimental Evaluation

In our experimental setup, we prepared 6 variants of 3DPmS-RRA based on the combinations of UB types: (1. thermo 2. binary) and saturated adder types in pointer update circuit (1. CST, 2. 3DPmS as an FPE instance, 3. RTL). These are

listed below.

- 3DPmS-ThermoRTL: Implemented with thermometer-coded UBs and saturated adder (from RTL) for update circuit.
- 3DPmS-BinaryRTL: Implemented with binary UBs and saturated adder (from RTL) for update circuit.
- 3DPmS-ThermoCST: Implemented with thermometer-coded UBs and CST based saturated adder for update circuit.
- 3DPmS-BinaryCST: Implemented with binary UBs and CST based saturated adder for update circuit.
- 3DPmS-ThermoInst: Implemented with thermometer-coded UBs and 3DPmS FPE based saturated adder for update circuit.
- 3DPmS-BinaryInst: Implemented with binary UBs and 3DPmS FPE based saturated adder for update circuit.

We wrote Verilog HDL generator for all these 6 different variants, and we get synthesis results for all of them. The synthesis results are given in Chapter VI.

CHAPTER III

MULTI-PICK RRA BASED ON TC-PPA

In this chapter, we first present 2-pick TC-PPA [14]. Then, we generalize TC-PPA for any m to construct m-pick TC-PPA, and present its 4 different PPN variants [10, 11, 12, 13].

3.1 2-Pick TC-PPA

As we mentioned before in previous work, TC-PPA [1, 14] is based on Gupta-McKeown's [2, 3] second PPE architecture, which consists of two FPEs. This architecture divides the RRA problem into two FPE sub-problems. TC-PPA uses Gupta-McKeown's divide-and-conquer concept, but differs in the usage of thermometer-coded pointer and PPNs inside the two FPEs. We implement FPE with a PPN unlike Gupta-McKeown. We extended 1-pick TC-PPA [1] to 2-pick TC-PPA [14]. Now, we will first look into our 2-pick TC-PPA architecture.

The first FPE sub-problem is to look for active requests in the High Priority Section (HPS), i.e., from the Hptr to the rightmost request. Second FPE sub-problem is to look for active requests from the leftmost request to the rightmost request. To solve first sub-problem, HPS is extracted from inputs and applied to one of the FPE to look for active requests. To solve second sub-problem, another FPE is used to look for active requests from the leftmost request to the rightmost request. Fig. 13 illustrates an example input segmentation.

In Fig. 13 small letters a, b, c, d, e, f, g, and h are inputs of RRA, and input f has the highest priority for this example. In TC-PPA, we use thermometer-coded lowest priority pointer (Lptr) as shown in Fig. 14 to extract HPS easily. To get HPS, just taking bit-wise AND of the inputs and the Lptr is enough. Then we apply

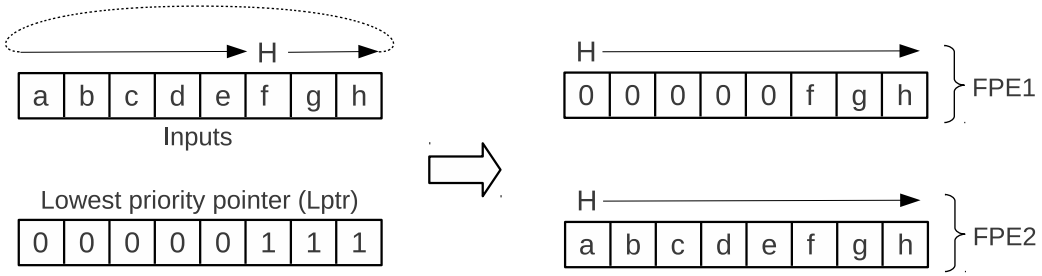

Figure 13: Example input segmentation of the proposed method

HPS to FPE1, and all requests to FPE2. Our top-level circuit architecture for 2-pick TC-PPA is shown in Fig. 14.

As can be seen in Fig. 14, we use two 2-pick FPEs (FPE1 and FPE2). FPE1 looks for 2 active requests in HPS, and similarly FPE2 looks for 2 active requests among all inputs. There are 3 possibilities for 2-pick RRA:

- HPS (FPE1) has two or more requests, and hence, both selected requests (gnt1 and gnt2) come from FPE1.
- HPS (FPE1) has one request and hence contributes gnt1, and then, FPE2 contributes gnt2 (its gnt1).
- HPS (FPE1) has zero request, and hence, FPE2 contributes both gnt1 and gnt2.

These 3 conditions are managed by the upper MUX3 in Fig. 14. It selects the proper case out of 3 cases. For each clock cycle, the pointer should be updated. Similar to 3DP2S-RRA, there are 3 possibilities to update the pointer. Hence, the lower MUX3 is used. It updates the pointer depending on how many active requests there are at the inputs. If two or more requests are found, Lptr is updated to 1 bit right rotated version of gnt2th. If only one request is found, Lptr is updated to 1 bit right rotated gnt1th, and if no request is not found, Lptr is the recirculated (existing) pointer.

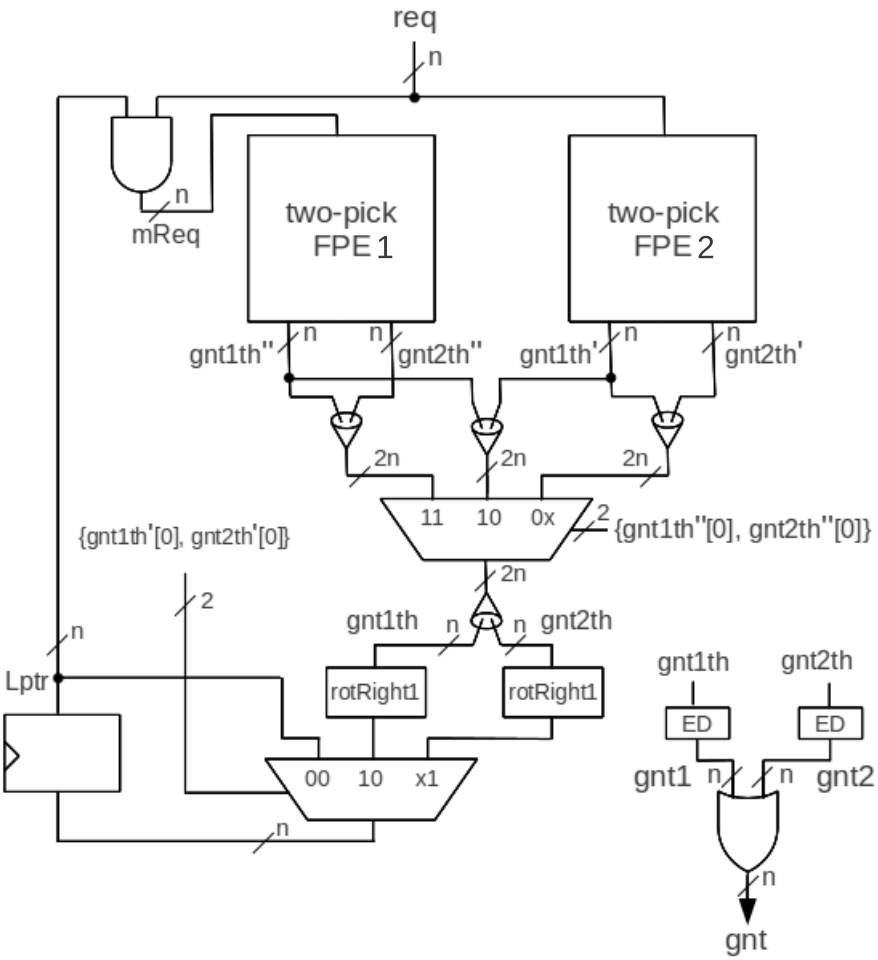

Figure 14: Top level architecture of proposed 2-pick TC-PPA.

In Fig. 14, ED stands for Edge Detector circuit. The signals, gnt1th and gnt2th are the 2 selected requests but they are thermometer-coded. The final result (gnt) is obtained after OR2 of gnt1 and gnt2.

3.2 m-Pick TC-PPA

Our top-level circuit architecture for m-pick TC-PPA is shown in Fig. 15. We replace 2-pick FPEs with m-pick FPEs. FPE1 looks at m requests at HPS and similarly FPE2 looks m at requests in all inputs. We implement m-pick FPEs with PPNs. Detailed explanation of PPNs can be found at subsection 4.4. Two m-pick FPEs can produce m+1 possible outputs. If there are m or more requests in HPS (FPE1), all selected

requests come from FPE1. If there are m-1 active requests in HPS, m-1 selected requests come from FPE1, and the last one comes from FPE2. If there are m-2 active requests in HPS, m-2 selected requests come from FPE1 and the last two come from FPE2. These possibilities continue this way until there is no request in HPS, this time all selected requests come from FPE2. To manage these m+1 possibilities, we replaced upper MUX3 with a MUX(m+1).

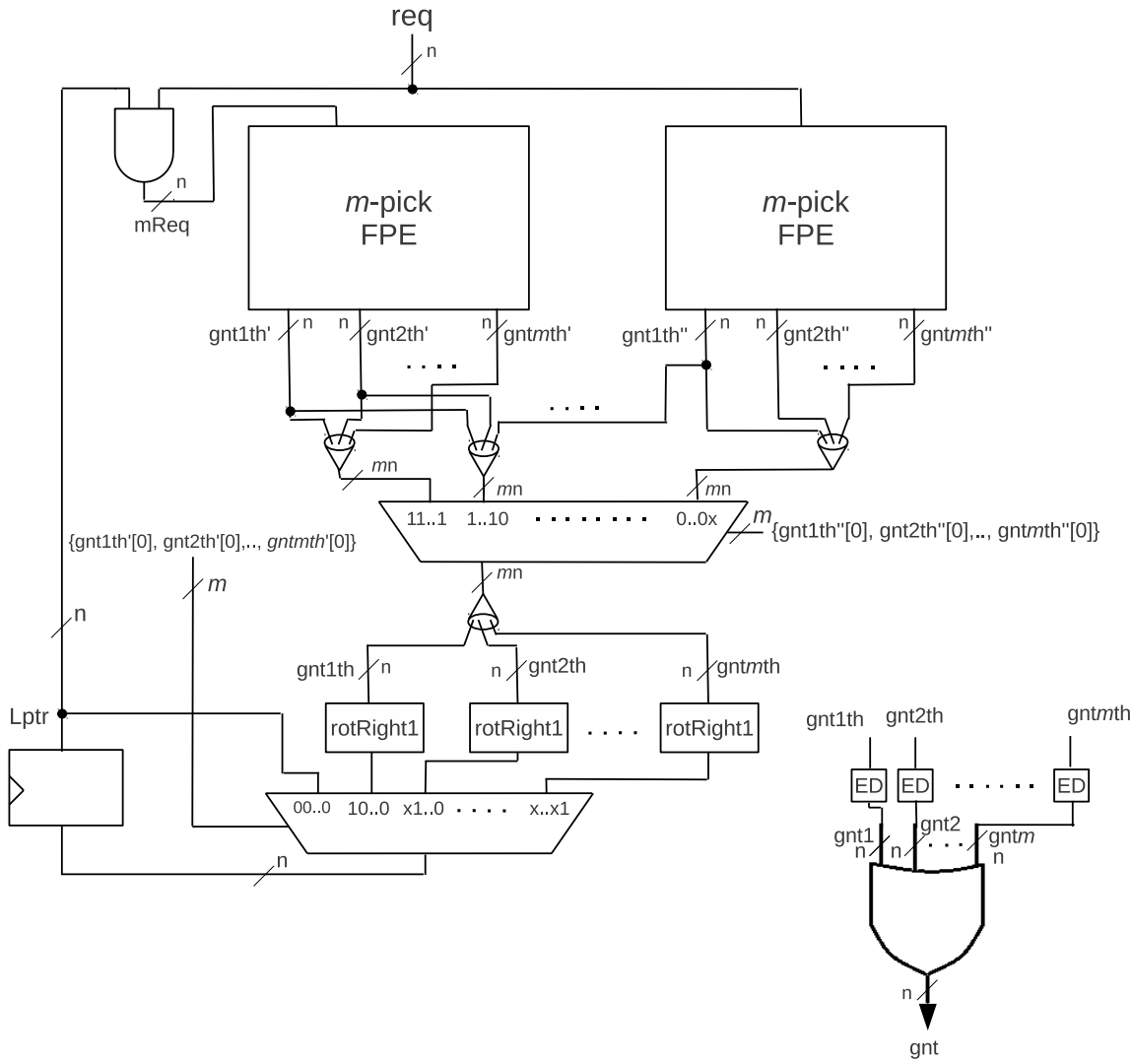

Figure 15: Top level architecture of proposed m-pick TC-PPA.

There are m+1 update choices for pointer, hence we replaced lower MUX3 (in

2-pick TC-PPA) with MUX(m+1) similar to 3DPmS-RRA. As you can see in Fig. 15, we need m times rotRight1 circuits and m times ED circuits for m-pick TC-PPA. Also, to get final result (gnt) ORm is used instead of OR2.

3.3 TC-PPA Versus Gupta-McKeown's 1-Pick RRA

TC-PPA uses a similar topology to Gupta's 1-pick RRA. The difference is that our critical path is extremely optimized with the help of a thermometer-coded pointer (Lptr: Lowest priority pointer instead of Hptr) and PPNs inside the FPEs. Furthermore, TC-PPA is extendable to m-pick RRA. This requires m-pick FPEs.

We implement FPE with a PPN unlike Gupta-McKeown. We can use any PPN in [16] not just KS PPN [12] as in 3DP2S. The most well-known PPNs are BK [10], HC [11], KS [12], and LF [13]. These will be explained in the next section.

3.4 FPEs with Different PPN Topologies

PPNs offer carry lookahead over the complete range of input bits and share logic extensively among different lookaheads. We used PPNs to implement multi-pick FPEs. Our method can use any PPNs, and their mutations are also possible. We used LF, KS, HC, and BK PPNs inside our multi-pick FPEs. This way, we derived several variants of our m-pick TC-PPA.

3.4.1 LF PPN

LF PPN for $n = 8$ bit input is given in Fig. 16. The complexity of LF PPN is $O((3n\log n)/4)$ for area and $O(\log n)$ for timing. Meanwhile, the minimum complexity of PPNs is $O(\log n)$ for timing as in the case of LF. However, the main disadvantage of LF PPN is higher fan-out as shown in Fig. 16, and this affects timing and area negatively. To drive multiple gates, bigger driving gates are used. Hence, these bigger driving gates increase total area and timing.

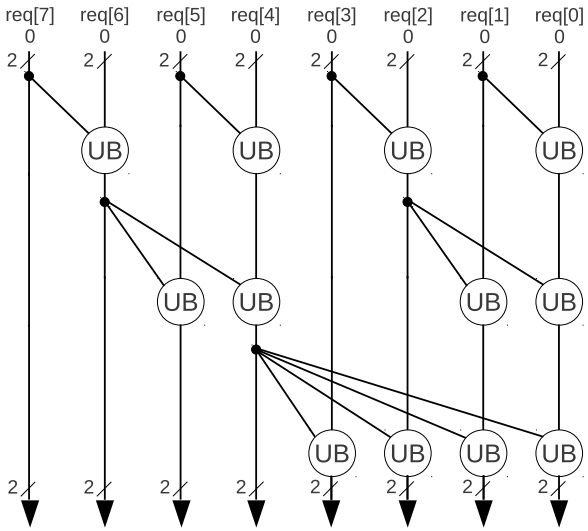

Figure 16: Microarchitecture of FPE with LF PPN.

3.4.2 KS PPN

KS architecture has complexity $O(n \log n)$ for area and $O(\log n)$ for timing similar to LF. Also, it has low fan-out compared to LF. Its main disadvantage is wiring tracks. Fig. 17 shows KS architecture that is drawn for $n = 8$ bits input.

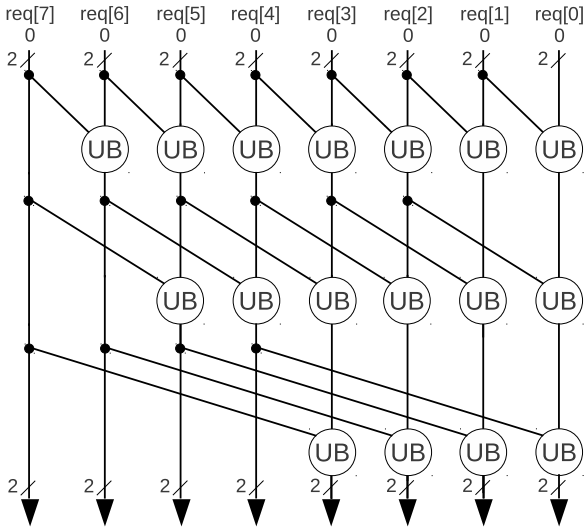

Figure 17: Microarchitecture of FPE with KS PPN.

3.4.3 BK PPN

The complexity of BK architecture is $O(2n)$ for area and $O(2\log n)$ for timing. Fig. 18 shows BK architecture that is drawn for $n = 8$ bits input. It has the minimum area complexity compared to the other PPN architecture. Hence, to implement area efficient design, it is the best choice for implementing FPEs. On the other hand, it has maximum timing complexity against to the other PPNs. Thus, it is the slowest architecture among the four PPN architectures, and it is not a good solution for timing efficient or fast designs.

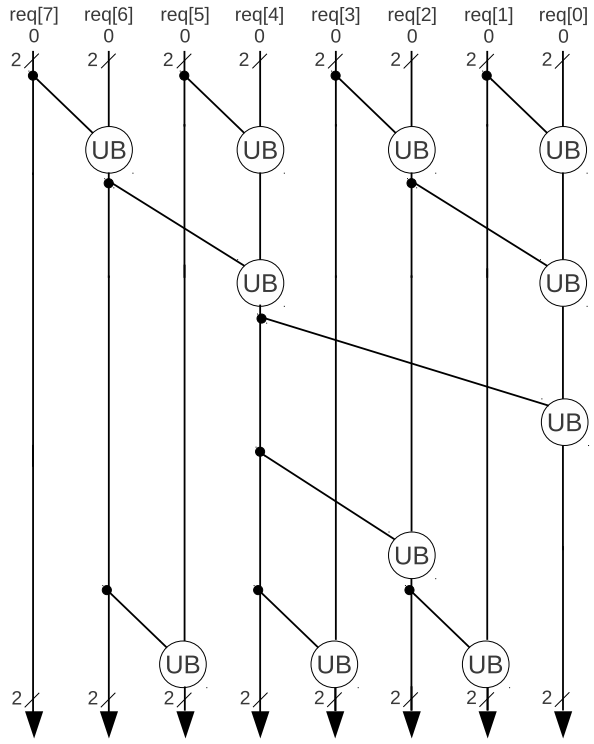

Figure 18: Microarchitecture of FPE with BK PPN.

3.4.4 HC PPN

In Fig. 19, HC architecture for $n = 8$ bit input is shown. For area, HC has $O(n\log n/2)$ complexity and for timing it has $O(\log n)$ complexity as similar as LF and KS. HC architecture is a hybrid architecture of KS and BK. When we compare

the wiring structure of HC and KS, HC has simple wiring structure, and its wiring cost is not much as KS. This is the one advantage of HC over KS in terms of area and timing.

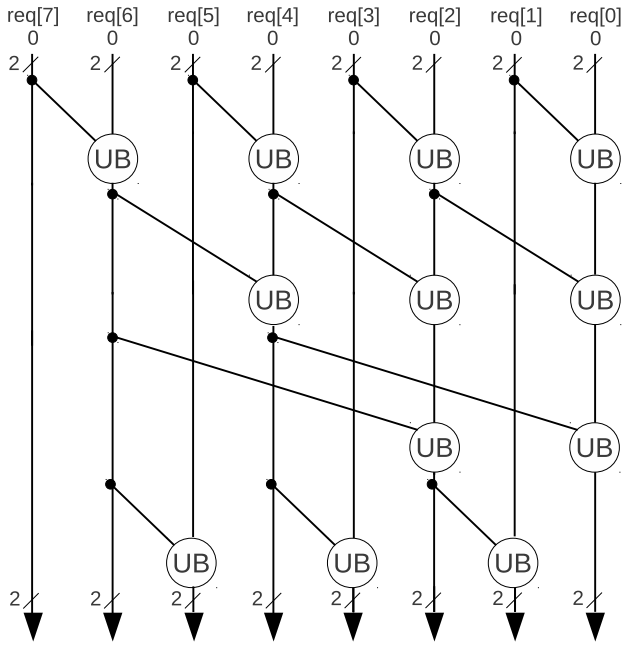

Figure 19: Microarchitecture of FPE with HC PPN.

Table 2 summarizes area and critical path complexities as well as max fanout of each PPN topology.

Table 2: Area, critical path and max fanout of each PPN topology

	Area	**Critical Path**	**Max Fanout**
LF	$(3n \log n)/4$	$\log n$	$n/2$
KS	$n \log n$	$\log n$	2
BK	$2n$	$2 \log n$	2
HC	$n \log n/2$	$\log n$	2

3.5 Binary and Thermo Variants of UB

UB, which we used in for all PPN architectures to implement multi-pick FPEs, can be either thermometer-coded or binary adder saturated at m for m-pick TC-PPA. For

example, if we want to implement 2-pick TC-PPA, UB should be either thermometer-coded or binary adder saturated at 2. Fig. 20 shows Verilog RTL for both variants of adder saturated at 2.

By the way, UB for proposed method does not have any pointer input or output compared to UB, which is for 3DPmS-RRA. Similar to 3DPmS-RRA, UB has two 2-bit unsigned inputs and one 2-bit unsigned output.

When taking into account thermometer-coded and binary versions of UB with four different types of PPN architectures (LF, KS, BK, and HC), we have 8 different variants of proposed m-pick TC-PPA architecture.

```
module unitBlock (A,B,Y);
input [2-1:0] A,B;
output reg [2-1:0] Y;

always @(*) begin
        case({A,B})
            4'b00_00 : Y = 2'b00;
            4'b00_01 : Y = 2'b01;
            4'b01_00 : Y = 2'b01;
            default : Y = 2'b11;
        endcase
end

endmodule
```

a)

```
module unitBlock (A,B,Y);
input [2-1:0] A,B;
output reg [2-1:0] Y;

always @(*) begin
        if((A+B)>2)
            Y = 2;
        else
            Y = A+B;
end

endmodule
```

b)

Figure 20: Verilog of UB in 2-pick TC-PPA (a) thermo and (b) binary.

These 8 different variants of m-pick TC-PPA are listed below.

- TC-PPA-BinaryLF: Implemented with thermometer-coded UB and LF PPN architecture for FPE.
- TC-PPA-ThermoLF: Implemented with binary UB and LF PPN architecture for FPE.
- TC-PPA-BinaryKS: Implemented with thermometer-coded UB and KS PPN architecture for FPE.

- TC-PPA-ThermoKS: Implemented with binary UB and KS PPN architecture for FPE.
- TC-PPA-BinaryBK: Implemented with thermometer-coded UB and BK PPN architecture for FPE.
- TC-PPA-ThermoBK: Implemented with binary UB and BK PPN architecture for FPE.
- TC-PPA-BinaryHC: Implemented with thermometer-coded UB and HC PPN architecture for FPE.
- TC-PPA-ThermoHC: Implemented with binary UB and HC PPN architecture for FPE.

We wrote Verilog HDL generators for all these 8 different variants of our method. In addition, we got synthesis results for all variants. Our experimental results are presented in Chapter VI.

CHAPTER IV

CASCADE METHOD

In this chapter, we first present a 2-pick RRA implemented with cascade method. Then, we generalize it for any m to construct m-pick RRA with cascade method.

4.1 2-Pick Cascade Method

Cascade method is the obvious way to implement an m-pick RRA. The method is based on selecting request one by one with cascaded 1-pick PPEs. Also, we believe that the cascade method is the method used in the chip design industry frequently to implement m-pick RRA.

In a 2-pick cascade architecture, we use a simplified 1-pick version of 3DP2S and call it 2DP1S (Two-Dimensional Programmable One-Selector). To turn 3DP2S into 2DP1S, only the first layer of the 3DP2S is implemented. Fig. 21 shows 2-pick RRA which is implemented with cascade method.

In Fig. 21, first 2DP1S selects the highest priority request (gnt1), then one bit right rotated gnt1 is fed to the second 2DP1S as Hptr. Hence, the second highest priority request will be selected by the second 2DP1S (gnt2). The output of the circuit is obtained by OR of these two requests (gnt1 and gnt2) as similar as the other methods. To update Hptr, a MUX2 is used based on the number of requests at input. If there is no request, previous Hptr will be recirculated. If there are two or more requests, one bit right rotated gnt2 will be the next Hptr. If there is only one request, again one bit right rotated gnt2 will be the next Hptr. Because, when there is one request, gnt1 and gnt2 will be the same for 2-pick cascade method.

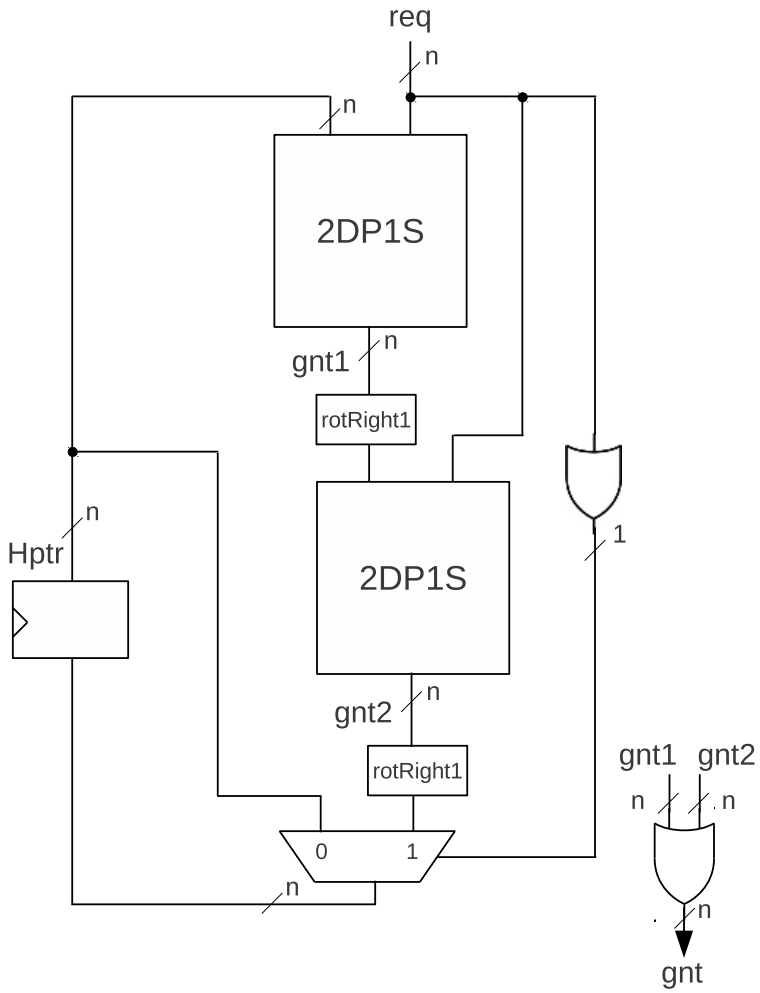

Figure 21: Cascade method for 2-pick RRA

4.2 m-Pick Cascade Method

Fig. 22 shows cascade method for m-pick RRA, which is the m-pick version of Fig. 21. Here, m 2DP1S and an adder saturated at m are used. In addition, there are m+1 pointer update choices, hence, a MUXm is used. Saturated adder at m determines the number of the requests. According to the result of saturated adder at m, Hptr will be updated. Also, ORm is used instead of OR2 to get final result (gnt) as similar as the other m-pick RRA methods.

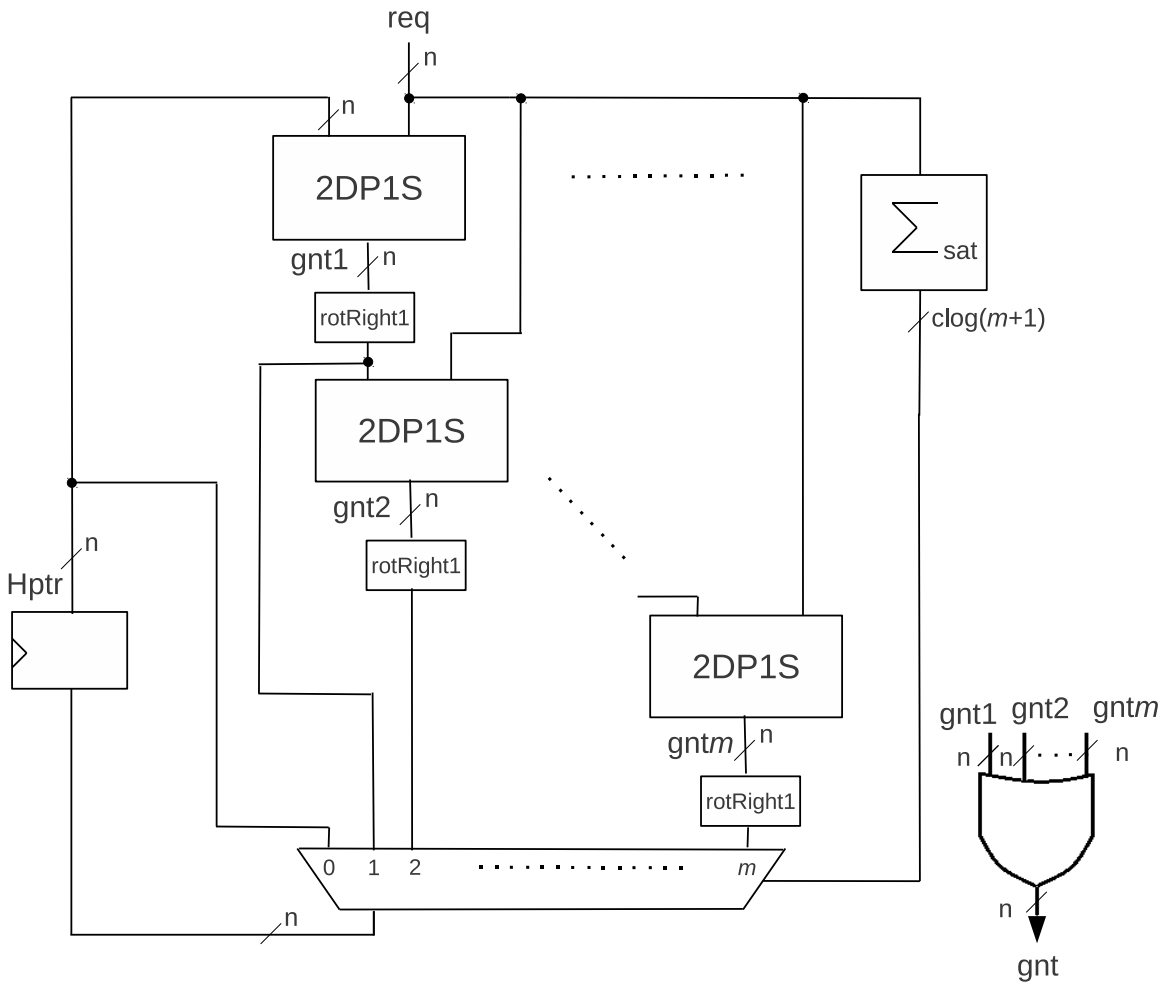

Figure 22: Cascade method for m-pick RRA

We wrote a Verilog HDL generator for the cascade method. Our experimental results are presented in Chapter VI.

4.3 Time-Division Multiplexing

It is possible to implement m-pick RRA using Time-Division Multiplexing (TDM) concept. It is based on the idea of selecting only 1 request every clock cycle. After m cycles, you will have selected m requests. However, TDM has some critical drawbacks such as the following: You need to run the circuit at m times faster clock frequency, and that may not be possible because of the longest path delay. Also, you need to

add some extra logic to solve clock domain crossing problems in m-pick RRA based on TDM. While sending a signal from one clock domain to another, you need to use synchronizer circuits. If multiple signals need to be sent from one clock domain to another, you need to use asynchronous FIFO. When considering all these drawbacks, m-pick RRA based on TDM is not a good solution to implement an m-pick RRA. Hence, in this thesis, we did not implement m-pick RRA based on TDM.

CHAPTER V

HDL CODE GENERATORS

In this chapter, HDL (Hardware Description Language) code generators for all proposed methods and their variants are examined. Instead of writing Verilog code for all methods directly, we wrote Verilog code generator scripts in the Perl scripting language for all methods. The idea behind writing scripts is to make Verilog code generation automatic, because all multi-pick RRA architectures will be compared against each other for input bit-widths varied from 8 to 128 as well as pick sizes varied from 2 to 5. In order to get rid of this intensive Verilog code writing process, we automated it by writing scripts. Another reason of writing scripts is that writing Verilog code for larger bit-widths and pick sizes is not trivial.

By the way, we wrote Verilog code generator scripts for (i) 6 different variants of 3DPmS-RRA, (ii) 8 different variants of m TC-PPA and (iii) cascade method. These scripts take two arguments, the input bit-width and pick size of the m-pick RRA architecture. Then, scripts generate corresponding Verilog code for the specific method. All generated Verilog code are compatible with the Verilog-1995 standard. An example generator script written for one of the variants of 3DPmS-RRA can be found in Appendix.

5.1 Code Generation for 3DPmS-RRA

There are 6 different variants of this method as we mentioned. Hence, 6 different generators are written for this method. These generators take input bit-width and pick size as arguments and return corresponding Verilog code as output. Verilog modules generated automatically for this method and its variants are listed below.

- pickN: This module is the top level module for the method and its 6 different variants. It includes Verilog code for the circuit which is shown in Fig. 9. and performs multi-pick RRA operation.

- unitBlock: This module is for the UB of 3DPmS. Its Verilog code (thermometer and binary versions) are given in Fig 11.

- binary2thermo: This module is used in only binary versions of the method. It takes binary number as input and returns the thermometer-coded version of the number as output.

- cstAdder: This module is used in 3DPmS-ThermoCST and 3DPmS-BinaryCST versions of the method. It contains Verilog code for the circuit of CST adder saturated at m.

- testBench: This module is just for simulation purposes. It contains the test-bench code written in Verilog that verifies the correctness of the design while instantiating the top module of the design.

- wrapper: This module is just used in synthesis results. It instantiates the top module of the design and adds flip-flops at the inputs and outputs of the design to get flop-to-flop timing.

5.2 Code Generation for m-Pick TC-PPA

Eight different generators are written for 8 variants of the proposed method. These variants of the method are mentioned at the end of Chapter IV. Similar to the generators written for 3DPmS-RRA architecture, these generators take two arguments (input bit-width and pick size) and generate Verilog code. Verilog modules generated automatically for this method are listed below.

- pickN: This module is the top level module of the architecture. It includes Verilog code for the circuit which is shown in Fig. 15.
- mPickFPE: This module includes the circuit for multi-pick FPE which is implemented with any of the PPN architectures (LF, BK, KS, HC).
- unitBlock: This module includes the UB circuitry for multi-pick FPEs. Its Verilog code is given in Fig 20.
- edgeDetector: This module is for the block shown in Fig. 15 as ED. It finds 0 to 1 transition at the selected requests.
- wrapper, testBench, binary2thermo: These modules are similar to the modules that are explained in the previous sections.

5.3 Code Generation for Cascade Architecture

Only one generator (i.e., no variants) is written for cascade method. It works similar to the generators which are mentioned in the previous sections. It generates automatically pickN, 2DP1S, wrapper, and testBench modules based on input bit-width and pick size. Module pickN is the top module of this architecture. It includes Verilog implementation of the circuit shown in Fig. 22. Module wrapper and module testBech are similar to the modules explained at the previous sections. Module 2DP1S is the simplified version of 3DP2S.

CHAPTER VI

EXPERIMENTAL SETUP AND RESULTS

In this chapter, the experimental setup and results for all 3 RRA architectures (with their variants) are presented. Results are given in terms of timing and area. This chapter also includes detailed discussion of our binary search based synthesis script.

6.1 Experimental Setup

In our experiments, we used Synopsys Design Compiler (DC) with ARM-Artisan TSMC 180nm worst-case (slow) standard-cell library.

To automate verification and synthesis processes, we used scripts written in Perl. Verification script takes as input arguments the method, pick size, and bit-widths and calls the desired HDL code generator script to create Verilog codes with its testbench, and checks whether the design passes the testbench or not. The testbench works as in the following: First, it generates random input vectors for the design, and then it checks the output of the design whether it is correct or not. If the output of the design is not correct, the testbench gives an error message and stops verification. If the outputs of the design are correct, the testbench starts a new iteration. It repeats this several times while using new input vectors in each iteration.

Our synthesis script takes the file location of Verilog codes as argument, calls Synopsys DC, and applies our binary search based synthesis script in Synopsys DC to obtain the smallest clock period.

6.2 Binary Search Based Synthesis Script

Binary search based synthesis script is an iterative synthesis script written in the TCL scripting language for Synopsys DC. It synthesizes each design 4 times and it

works as in the following: First, it sets the desired clock period to 0.1ns (which is impossible to meet) for the first iteration. For the next iterations, the average of the largest period that was not achieved (lower bound) and the achieved clock of the previous iteration is used. For the last iteration (4th iteration), the last met clock * 0.9 is fed to Synopsys to check if it can achieve to meet this period. After all of the iterations, the best achieved clock period, area, and netlist file are saved. According to our experiments, 4 iterations are usually enough to get the optimum timing result. When we increase the number of iterations, we see that there is not much timing improvement. 4 iteratations was also used to save computation time.

6.3 Experimental Results

We obtained results for various input bit-widths (8, 16, 32, 64, and 128) and pick size (2, 3, 4, and 5) for all 15 RRA variants.

For synthesis, we used a wrapper module around the design such that all RRA inputs come from flops and all outputs drive flops. Therefore, our timing results are flop-to-flop and hence include flop's clock-to-Q delay and set-up time but do not include clock skew and jitter.

On the other hand, we give normalized product of area and timing instead of giving only area results. Since synthesis tools achieve shorter clock period by using more gates and/or higher drive and hence larger gates. As a result, area and timing are inversely proportional, and you should consider timing when looking at area results. That is why we give normalized area and timing product results.

Table 3, 5, 7, and 9 present the timing results for pick sizes 2, 3, 4, and 5, respectively. Table 4, 6, 8, and 10 present the area results (normalized area-timing products) for pick sizes 2, 3, 4, and 5, respectively. For each column, the bold number shows the smallest result for that input bit-width.

When we look at Table 3 for timing results for 2-pick RRA, we can see that TC-PPA is the most timing efficient architecture. For 128 bit input, TC-PPA-ThermoLF achieves 16.5% and 65% timing improvements over 3DPmS-RRA and cascade architectures, respectively. According to Table 4 for 128 bit 2-pick RRA area results, TC-PPA achieves 199% and 214% area improvements over 3DPmS-RRA and cascade architectures, respectively, when the TC-PPA result is taken as reference.

Although TC-PPA gives better timing results than the others for 2-pick RRA, for the bigger pick sizes (3, 4, and 5) 3DPmS-RRA gives better results in terms of timing. According to timing results in Table 5 for 3-pick RRA, for 128 bit input, one of the variants of the 3DPmS-RRA, 3DPMSThermoInst, has 4.9% and 38.9% timing improvements over TC-PPA and cascade architectures, respectively. However, according to area results for 128 bit 3-pick RRA as shown in Table 6, TC-PPA is 91% and 141% more area efficient than the 3DPmS-RRA and cascade architectures, respectively, where TC-PPA is considered 100%100.

When we look at the Table 7 and 8 for 128 bit timing and area results for 4-pick RRA, we can see that 3DPmS-RRA is 5.9% and 53% more timing efficient than TC-PPA and cascade architectures, respectively. However, TC-PPA has 117% and 85% area improvements over 3DPmS-RRA and cascade architecture, respectively.

Regarding the 5-pick 128 bit RRA timing and area results shown in Table 9 and 10, 3DPmS-RRA has 8.5% and 74% timing improvements over TC-PPA and cascade architectures respectively, but TC-PPA yields 112% and 124% better area than 3DPmS-RRA and cascade architecture, respectively.

In summary, 3DPmS-RRA architecture is the best choice for all pick sizes (except the pick size 2 - TC-PPA is the best for that pick size), when timing is the optmization criterion. However, when area is the criterion, TC-PPA architecture should be chosen. Note that there are small timing improvements between 3DPmS-RRA and TC-PPA like at most 8% 3DPmS-RRA is better, but when we consider area, TC-PPA has

important improvements over 3DPmS-RRA like at least 91%.

Table 3: Timing results for 2-pick RRA

Methods	**Number of Ports**				
	8	**16**	**32**	**64**	**128**
3DPmS-ThermoRTL	2.14	2.95	3.95	4.68	5.45
3DPmS-BinaryRTL	2.40	3.04	3.99	4.78	5.54
3DPmS-ThermoCST	2.43	2.86	3.54	3.97	4.43
3DPmS-BinaryCST	2.60	3.20	3.74	4.33	4.87
3DPmS-ThermoInst	2.16	2.56	2.96	3.37	3.87
3DPmS-BinaryInst	2.51	2.95	3.50	3.91	4.47
TC-PPA-ThermoBK	**2.05**	**2.37**	2.75	3.04	3.39
TC-PPA-BinaryBK	2.13	2.53	2.98	3.33	3.60
TC-PPA-ThermoHC	2.08	2.39	2.72	3.05	3.34
TC-PPA-BinaryHC	2.25	2.63	2.90	3.27	3.56
TC-PPA-ThermoKS	2.08	2.39	2.71	3.02	3.33
TC-PPA-BinaryKS	2.19	2.68	2.90	3.24	3.56
TC-PPA-ThermoLF	2.06	2.40	**2.70**	**2.95**	**3.32**
TC-PPA-BinaryLF	2.14	2.57	2.91	3.26	3.52
Cascade	2.79	3.27	3.97	4.67	5.49

Table 4: Area results (Normalized area-timing products) for 2-pick RRA

Methods	Number of Ports				
	8	**16**	**32**	**64**	**128**
3DPmS-ThermoRTL	1.79	1.74	2.36	2.68	3.21
3DPmS-BinaryRTL	2.15	2.54	3.02	3.52	4.27
3DPmS-ThermoCST	1.90	2.17	2.35	2.55	2.99
3DPmS-BinaryCST	2.41	2.67	3.31	3.51	4.06
3DPmS-ThermoInst	1.85	2.21	2.46	2.87	3.08
3DPmS-BinaryInst	2.32	2.60	3.22	3.48	4.04
TC-PPA-ThermoBK	1.04	**1.00**	**1.00**	**1.00**	**1.00**
TC-PPA-BinaryBK	1.16	1.19	1.24	1.13	1.18
TC-PPA-ThermoHC	**1.00**	1.05	1.24	1.22	1.36
TC-PPA-BinaryHC	1.39	1.40	1.57	1.51	1.59
TC-PPA-ThermoKS	1.11	1.33	1.51	1.63	1.83
TC-PPA-BinaryKS	1.28	1.58	1.69	1.78	1.99
TC-PPA-ThermoLF	1.07	1.13	1.16	1.14	1.15
TC-PPA-BinaryLF	1.13	1.39	1.47	1.32	1.33
Cascade	2.08	2.49	2.61	2.76	3.14

Table 5: Timing results for 3-pick RRA

Methods	Number of Ports				
	8	**16**	**32**	**64**	**128**
3DPmS-ThermoRTL	**2.47**	3.17	4.16	4.85	5.63
3DPmS-BinaryRTL	2.71	3.16	4.14	4.83	5.65
3DPmS-ThermoCST	2.69	3.23	3.82	4.42	5.07
3DPmS-BinaryCST	2.76	3.25	3.90	4.40	4.96
3DPmS-ThermoInst	2.48	**2.96**	**3.57**	**4.15**	**4.70**
3DPmS-BinaryInst	2.73	3.18	3.71	4.28	4.74
TC-PPA-ThermoBK	2.62	3.33	4.15	5.04	5.86
TC-PPA-BinaryBK	2.89	3.51	4.29	4.98	5.92
TC-PPA-ThermoHC	2.65	3.20	3.83	4.41	5.05
TC-PPA-BinaryHC	2.86	3.44	3.89	4.47	4.99
TC-PPA-ThermoKS	2.63	3.22	3.88	4.42	5.01
TC-PPA-BinaryKS	2.78	3.31	3.90	4.48	5.02
TC-PPA-ThermoLF	2.60	3.20	3.78	4.37	4.94
TC-PPA-BinaryLF	2.84	3.42	3.96	4.43	4.93
Cascade	3.81	4.48	5.15	5.82	6.53

Table 6: Area results (Normalized area-timing products) for 3-pick RRA

Methods	Number of Ports				
	8	**16**	**32**	**64**	**128**
3DPmS-ThermoRTL	1.52	1.71	1.90	2.16	2.75
3DPmS-BinaryRTL	1.52	1.45	1.60	1.82	1.91
3DPmS-ThermoCST	1.46	1.72	2.07	2.39	2.81
3DPmS-BinaryCST	1.55	1.69	1.80	2.00	2.34
3DPmS-ThermoInst	1.61	1.82	2.18	2.61	3.14
3DPmS-BinaryInst	1.44	1.50	1.59	1.81	2.08
TC-PPA-ThermoBK	1.03	**1.00**	1.09	1.07	1.14
TC-PPA-BinaryBK	1.17	1.07	**1.00**	**1.00**	**1.00**
TC-PPA-ThermoHC	1.12	1.28	1.43	1.54	1.85
TC-PPA-BinaryHC	1.19	1.22	1.28	1.33	1.42
TC-PPA-ThermoKS	1.28	1.48	1.73	2.11	2.54
TC-PPA-BinaryKS	1.42	1.40	1.60	1.70	2.00
TC-PPA-ThermoLF	**1.00**	1.13	1.15	1.27	1.43
TC-PPA-BinaryLF	1.22	1.11	1.17	1.20	1.28
Cascade	1.93	2.03	2.15	2.24	2.41

Table 7: Timing results for 4-pick RRA

Methods	Number of Ports				
	8	**16**	**32**	**64**	**128**
3DPmS-ThermoRTL	2.71	**3.33**	4.24	5.00	5.80
3DPmS-BinaryRTL	2.98	3.70	4.47	5.25	5.94
3DPmS-ThermoCST	2.84	3.53	4.23	4.98	5.69
3DPmS-BinaryCST	3.14	3.82	4.55	5.39	6.04
3DPmS-ThermoInst	**2.66**	3.34	**3.98**	**4.73**	**5.45**
3DPmS-BinaryInst	3.15	3.78	4.48	5.25	5.90
TC-PPA-ThermoBK	2.87	3.83	4.82	5.89	6.92
TC-PPA-BinaryBK	3.48	4.39	5.47	6.60	7.57
TC-PPA-ThermoHC	3.02	3.74	4.49	5.17	5.85
TC-PPA-BinaryHC	3.52	4.22	5.00	5.65	6.39
TC-PPA-ThermoKS	2.98	3.74	4.45	5.18	5.87
TC-PPA-BinaryKS	3.52	4.26	4.91	5.61	6.32
TC-PPA-ThermoLF	2.87	3.67	4.37	5.04	5.77
TC-PPA-BinaryLF	3.48	4.13	4.90	5.60	6.38
Cascade	4.85	5.73	6.60	7.48	8.36

Table 8: Area results (Normalized area-timing products) for 4-pick RRA

Methods	Number of Ports				
	8	**16**	**32**	**64**	**128**
3DPmS-ThermoRTL	1.35	1.67	1.88	2.10	2.37
3DPmS-BinaryRTL	1.46	1.70	1.95	2.09	2.28
3DPmS-ThermoCST	1.26	1.66	1.87	2.28	2.49
3DPmS-BinaryCST	1.55	1.65	1.85	1.93	2.17
3DPmS-ThermoInst	1.44	1.76	2.11	2.44	2.72
3DPmS-BinaryInst	1.54	1.77	1.95	2.10	2.35
TC-PPA-ThermoBK	1.01	**1.00**	**1.00**	**1.00**	**1.00**
TC-PPA-BinaryBK	1.19	1.12	1.08	1.01	**1.00**
TC-PPA-ThermoHC	1.04	1.29	1.49	1.72	1.91
TC-PPA-BinaryHC	1.23	1.30	1.35	1.42	1.45
TC-PPA-ThermoKS	1.13	1.45	1.79	2.18	2.60
TC-PPA-BinaryKS	1.26	1.40	1.62	1.76	1.95
TC-PPA-ThermoLF	**1.00**	1.09	1.14	1.22	1.28
TC-PPA-BinaryLF	1.13	1.17	1.10	1.10	1.09
Cascade	1.91	1.91	1.96	1.87	1.85

Table 9: Timing results for 5-pick RRA

Methods	Number of Ports				
	8	**16**	**32**	**64**	**128**
3DPmS-ThermoRTL	2.91	**3.51**	4.40	5.25	6.08
3DPmS-BinaryRTL	3.29	4.09	5.07	5.89	6.78
3DPmS-ThermoCST	3.04	3.66	4.51	5.36	6.15
3DPmS-BinaryCST	3.32	4.15	5.01	5.93	6.83
3DPmS-ThermoInst	**2.80**	3.61	**4.35**	**5.16**	**5.90**
3DPmS-BinaryInst	3.43	4.18	5.16	6.05	6.91
TC-PPA-ThermoBK	3.18	4.18	5.38	6.48	7.79
TC-PPA-BinaryBK	3.87	4.86	6.11	7.50	8.83
TC-PPA-ThermoHC	3.19	4.02	4.86	5.58	6.40
TC-PPA-BinaryHC	3.94	4.78	5.54	6.42	7.34
TC-PPA-ThermoKS	3.21	3.99	4.86	5.69	6.46
TC-PPA-BinaryKS	3.81	4.73	5.55	6.37	7.28
TC-PPA-ThermoLF	3.10	3.91	4.71	5.57	6.42
TC-PPA-BinaryLF	3.62	4.53	5.41	6.31	7.17
Cascade	5.88	7.05	8.11	9.20	10.28

Table 10: Area results (Normalized area-timing products) for 5-pick RRA

Methods	**Number of Ports**				
	8	**16**	**32**	**64**	**128**
3DPmS-ThermoRTL	1.32	1.60	1.94	2.30	3.23
3DPmS-BinaryRTL	1.30	1.38	1.53	1.71	2.12
3DPmS-ThermoCST	1.28	1.56	1.96	2.33	3.04
3DPmS-BinaryCST	1.26	1.30	1.52	1.67	2.15
3DPmS-ThermoInst	1.48	1.70	2.13	2.58	3.48
3DPmS-BinaryInst	1.32	1.40	1.52	1.71	2.13
TC-PPA-ThermoBK	1.02	**1.00**	1.06	1.10	1.18
TC-PPA-BinaryBK	1.22	1.02	**1.00**	**1.00**	**1.00**
TC-PPA-ThermoHC	1.06	1.24	1.62	1.91	2.48
TC-PPA-BinaryHC	1.24	1.23	1.29	1.40	1.73
TC-PPA-ThermoKS	1.16	1.40	1.89	2.43	3.36
TC-PPA-BinaryKS	1.34	1.46	1.69	1.98	2.42
TC-PPA-ThermoLF	**1.00**	1.08	1.20	1.29	1.55
TC-PPA-BinaryLF	1.22	1.06	1.13	1.18	1.36
Cascade	2.30	1.80	1.90	1.91	2.24

CHAPTER VII

CONCLUSIONS AND FUTURE WORK

In this thesis, we propose three multi-pick RRA architectures (3DPmS-RRA, m-pick TC-PPA, and cascade). 6 different variants of 3DPmS-RRA and 8 different variants of m-pick TC-PPA are also introduced. A multi-pick RRA selects the m topmost requests out of n inputs with priority order indicated by an internally kept pointer (with an update policy that ensures fairness among requestors).

We wrote automated HDL code generators for all variants of both 3DPmS-RRA and m-pick TC-PPA as well as the cascade architecture (which is the obvious way to implement a multi-pick RRA). Then all multi-pick architectures were verified and synthesized for various input bit-widths and pick sizes for the purposes of benchmarking of all architectures on equal grounds.

According to the synthesis results given in previous chapter, we show that our proposed m-pick TC-PPA yields better area than the other two architectures (3DPmS-RRA and cascade architecture) for all cases, and yields better timing for pick size 2. On the other hand, 3DPmS-RRA yields better timing than the other two architectures (m-pick TC-PPA and cascade architecture) for pick sizes 3, 4, and 5.

For all cases, cascade architecture gives worse results in terms of both area and timing, and we believe that this method is frequently used in the chip design industry to implement multi-pick RRA. Hence, this thesis shows that there are more efficient ways to implement multi-pick RRA in terms of timing and area.

Future work for this thesis may include obtaining more results for different input bit-widths and bigger pick sizes, as well as getting synthesis results on FPGA (Field Programmable Gate Array), implementing a more efficient testbench that covers all

corner cases, and further optimization of both multi-pick TC-PPA and 3DPmS-RRA in terms of area and timing.

APPENDIX – EXAMPLE GENERATOR SCRIPT

This Verilog code generator script was written for 3DPmS-ThermoRTL.

```
##############################################################################################
# Generator for 3DPmS-ThermoRTL
##############################################################################################

use diagnostics;
use POSIX qw(ceil floor);
##############################################################################################
# Read bidwidth and pick size from command line.
##############################################################################################
$bitwidth = $ARGV[0];
$Npick = $ARGV[1];

##############################################################################################
# Finding depth number and internal signals bitwidth for 3DPmS.
##############################################################################################
$depth = ceil(log($bitwidth)/log(2)) ;
$logCeil = ceil(log($bitwidth+1)/log(2)) ;

#Array definition for unit block instantiations
for($ii=0;$ii<$bitwidth;$ii++)
{
    $array[$ii]=$ii;
}

#Print Module Name and input and Output Ports
print "module pickN(clk, rst, req, gnt);

input clk, rst;
input [$bitwidth-1:0] req;
output [$bitwidth-1:0] gnt;

wire [$bitwidth-1:0] R = req;

wire [$bitwidth-1:0] ";

#Internal signals definitions for 3DPmS
for($ii=1;$ii<=$depth;$ii++)
{
    if($ii!=1)
    {
        print ",";
    }
    print "uBP",$ii,"";
}

print ";
wire [$bitwidth-1:0] ";
for($ii=1;$ii<=$Npick;$ii++)
{
    if($ii!=1)
    {
        print ",";
    }
    print "uBS",$ii,"";
}
print ";
wire [$bitwidth-1:0] ";
for($ii=1;$ii<=$Npick;$ii++)
{
```

```
    if($ii!=1)
    {
        print ",";
    }
    print "uBS",$ii,"$depth";
}
print ";
";
for($ii=1;$ii<=$depth;$ii++)
{
    print "wire [$Npick-1:0] ";
    for($jj=$bitwidth-1;$jj>=0;$jj--)
    {
        if($jj==0)
        {
            print "uBR",$ii,$jj,";\n";
        }
        else
        {
            print "uBR",$ii,$jj,",";
        }
    }
}
print "wire [$bitwidth-1:0] ";
for($ii=1;$ii<=$Npick;$ii++)
{
    if($ii!=1)
    {
        print ",";
    }
    print "S",$ii,"Edge";
}

#RTL based saturated adder for update logic
print "";
print ";
wire [$logCeil-1:0] RbitCount;
reg [$bitwidth-1:0] P;

assign RbitCount = R[$bitwidth-1]";

for($ii=$bitwidth-2;$ii>=0;$ii--)
{
    print "+R[",$ii,"]";
}
print ";\n";

#Pointer update logic

print "
always @(posedge clk) begin
    if(rst)
        P <= 1 << ($bitwidth -1);
    else
        case(RbitCount)\n";

            for($ii=0;$ii<=$Npick;$ii++)
            {
                if($ii==0)
                {
                    print "\t\t$ii : P <= P;\n";
                }
                elsif($ii==$Npick)
                {
                    print "\t\tdefault : P <= {uBS",$ii,"[0], uBS",$ii,
```

```
                                                "[$bitwidth-1:1]};\n";
                }
                else
                {
                    print "\t\t$ii : P <= {uBS",$ii,"[0], uBS",$ii,
                                                "[$bitwidth-1:1]};\n";
                }
            }

print "\tendcase
end

";

#Unit Block instantiations
shiftArray(1);
for($ii=$bitwidth-1;$ii>=0;$ii--)
{
    print "unitBlock inst1",$ii,"(.Pu(P[",$array[$ii],"]),.Pp(P[",$ii,"]),
           .P(uBP1[",$ii,"]),.Ru({",$Npick-1,"'b0,R[",$array[$ii],"]}),
           .Rp({",$Npick-1,"'b0,R[",$ii,"]}),.R(uBR1",$ii,"));\n";
}

for($jj=1;$jj<$depth;$jj++)
{
    for($nn=2**($jj-1);$nn>0;$nn--)
    {
        shiftArray(1);
    }
    print "\n";
    for($ii=$bitwidth-1;$ii>=0;$ii--)
    {
        print "unitBlock inst",$jj+1,"",$ii,"(.Pu(uBP",$jj,"[",$array[$ii],"]),
              .Pp(uBP",$jj,"[",$ii,"]),.P(uBP",$jj+1,"[",$ii,"]),.Ru(uBR",$jj,"",
              $array[$ii],"),.Rp(uBR",$jj,"",$ii,"),.R(uBR",$jj+1,"",$ii,"));\n";
    }
}

#Edge detector implementation for 3DPmS
print "\n";
for($ii=1;$ii<=$Npick;$ii++)
{
    print "\nassign uBS",$ii,"",$depth," = {";
    for($jj=$bitwidth-1;$jj>=0;$jj--)
    {
        if($jj==0)
        {
            print "uBR",$depth,"",$jj,"[",$ii-1,"]};";
        }
        else
        {
            print "uBR",$depth,"",$jj,"[",$ii-1,"], ";
        }
    }
}

print "\n";
    for($jj=1;$jj<=$Npick;$jj++)
    {
        if($jj==1)
        {
            print "\n";
            print "assign S",$jj,"Edge = ~{uBS",$jj,"",$depth,"[0],uBS",$jj,"",
                   $depth,"[$bitwidth-1:1]} & uBS",$jj,"",$depth,";\n";
            print "assign uBS",$jj," = (|(R&P)) ? P : S",$jj,"Edge;\n";
```

```
        }
        else
        {
            print "assign S",$jj,"Edge = ~{uBS",$jj,"",$depth,"[0],uBS",$jj,"",
                   $depth,"[$bitwidth-1:1]} & uBS",$jj,"",$depth,";\n";
            print "assign uBS",$jj," = S",$jj,"Edge;\n";
        }
        print "\n";
    }

#Final results gnt is obtained
print "assign gnt = ";
    for($jj=1;$jj<=$Npick;$jj++)
    {
        if($jj==$Npick)
        {
            print "uBS",$jj,";\n";
        }
        else
        {
            print "uBS",$jj," | ";
        }
    }

#Module definition for thermometer coded Unit Block
print "
endmodule

//////////////////////////////////////////////////////////////////////

module unitBlock(Pu,Pp,P,Ru,Rp,R);
input Pu,Pp;
input [$Npick-1:0] Ru,Rp;
output P;
output reg [$Npick-1:0] R;

assign P = Pu | Pp;

always @(*) begin
    if(~Pp)
        case({Ru,Rp})";
        for($ii=0;$ii<$Npick;$ii++)
        {
            for($jj=0;$jj<$Npick;$jj++)
            {
                if(($ii+$jj)<$Npick)
                {
                    print "\n\t\t\t",2*$Npick,"'b";
                    for($zz=0;$zz<$Npick-$ii;$zz++)
                    {
                        print "0";
                    }
                    for($zz=0;$zz<$ii;$zz++)
                    {
                        print "1";
                    }
                    print "_";
                    for($kk=0;$kk<$Npick-$jj;$kk++)
                    {
                        print "0";
                    }
                    for($kk=0;$kk<$jj;$kk++)
                    {
                        print "1";
                    }
```

```
                    print " : R = ",$Npick,"'b";
                    for($zz=0;$zz<$Npick-($ii+$jj);$zz++)
                    {
                        print "0";
                    }
                    for($zz=0;$zz<($ii+$jj);$zz++)
                    {
                        print "1";
                    }
                    print ";";
                }
            }

        }
        print "\n\t\t\tdefault : R = ",$Npick,"'b";
        for($jj=0;$jj<$Npick;$jj++)
        {
            print "1";
        }
        print ";\n";
        print "\t\tendcase";
print "
    else
            R = Rp;
end

endmodule
";


#Module definition for testbench
print "

\`define PORT $bitwidth
\`define pickN $Npick

module rr_arbiter_tb ();

  //Datatype Declerations
    reg clk, rst;
    reg [\`PORT-1:0] req;
    reg [\`PORT-1:0] tb_out;
    wire [\`PORT-1:0] design_out;
    integer       ptr, ptrNxt, ii, kk, arbitrations, flag;
    reg [3:0] counter;

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

  //Declaration of Initial Values
    initial begin
        \$dumpvars;
        clk = 0;
        \#5 rst = 1;
        \#15 rst = 0;
    end

  //Toggle clock every 10 nanoseconds
    always
        \#10 clk = \~clk;

  //Test Vectors (Inputs) Declerations
    always \@(posedge clk) begin
    if (rst) begin
        ptr <= \#1 0;
```

```
        req <= \#1 0;
        tb_out <= \#1 0;
        counter = \$random;
        arbitrations = 0;
    end else begin
        if(counter == 0) begin
            req <= \#1 0;
            counter = \$random;
        end else begin
            req <= \#1 {\$random, \$random, \$random, \$random,
                        \$random, \$random, \$random, \$random,
                        \$random, \$random, \$random, \$random,
                        \$random, \$random, \$random, \$random};
            counter = counter -1;
        end

        if(((\`PORT < 32) && (arbitrations >= (1<<\`PORT)))
            || (arbitrations >= 300))
            \$finish;
            arbitrations = arbitrations +1;
        end
    end

  //Control and Monitoring Part
    always \@(posedge clk) begin
        \#2 //Wait 2ns for correct comparison
        tb_out = 0;
        ptrNxt = ptr;
        if (req !== 0) begin
            flag = 0;
            for (kk=1; kk<=\`PORT; kk=kk+1) begin
                ii = ptr -kk;
                ii = (ii < 0) ? (ii +\`PORT) : ii;
                if (req[ii]) begin
                    tb_out[ii] = 1;
                    ptrNxt = ii;
                    flag = flag +1;
                    if (flag == \`pickN)
                        kk = \`PORT;
                end

            end
            ptr = ptrNxt;
        end
        if (design_out === tb_out) begin
            \$display (\"time = \%d req = \%b dut_out = \%b model_out = \%b --> CORRECT\",
                       \$time, req, design_out, tb_out);
        end else begin
            \$display (\"time = \%d req = \%b dut_out = \%b model_out = \%b --> ERROR\",
                       \$time, req, design_out, tb_out);
  // \$finish;
    end
end

endmodule // rr_arbiter_tb
";

#Module definition for wrapper
print "


\`define N $bitwidth

module wrapper (clk, rst, in, gnt);
```

```
    input clk, rst;
    input [\`N-1:0] in;
    output [\`N-1:0] gnt;

    wire [\`N-1:0] my_gnt;

    reg [\`N-1:0] my_in;
    reg [\`N-1:0] gnt;
    reg my_rst;

    pick pickN (.clk(clk), .rst(my_rst), .req(my_in), .gnt(my_gnt));

    always \@ (posedge clk) begin
       my_in <= in;
       gnt <= my_gnt;
       my_rst <= rst;
    end

endmodule
";

#Sub-module used to rotate array for unit block instatiations
sub shiftArray
{
        my ($sNum) = @_;
        for($tt=0;$tt<$sNum;$tt++)
        {
            my $ind0 = $array[0];
            for($kk=1;$kk<=$#array;$kk++)
            {
                $array[$kk-1]=$array[$kk];
            }
            $array[$kk-1] = $ind0;
        }
}
```

Bibliography

[1] H. F. Ugurdag and O. Baskirt, "Fast parallel prefix logic circuits for n2n round-robin arbitration," *Microelectronics Journal*, vol. 43, pp. 573–581, Aug. 2012.

[2] P. Gupta and N. McKeown, "Designing and implementing a fast crossbar schedular," *Hot Interconnects*, 1998.

[3] P. Gupta and N. McKeown, "Designing and implementing a fast crossbar schedular," *IEEE Micro*, vol. 19, pp. 20–28, Jan. 1999.

[4] S. Q. Zheng and M. Yang, "Algorithm-hardware codesign of fast parallel round-robin arbiters," *IEEE Trans. Parallel and Distributed Systems*, vol. 18, pp. 84–95, Jan. 2007.

[5] G. Dimitrakopoulos, N. Chrysos, and K. Galanopoulos, "Fast arbiters for on-chip network switches," *Proc. Int. Conf. Computer Design (ICCD)*, vol. 4, pp. 333–336, 2004.

[6] C. Savin, T. McSmythurs, and J. Czilli, "Binary tree search architecture for efficient implementation of round robin arbiters," *Proc. Int. Conf. Acoustics Speech and Signal Processing (ICASSP)*, pp. 664–670, 2008.

[7] J.-M. Jou and Y.-L. Lee, "An optimal round-robin arbiter design," *J. Inf. Sci. Eng.*, vol. 26, pp. 2047–2058, 2010.

[8] H. F. Ugurdag and O. Baskirt, "An in-depth look at prior art int fast round-robin arbiter circuits," *Technical Report (Ozyegin University)*, vol. OZU-EF-2011-0001 http://eresearch.ozyegin.edu.tr/xmlui/handle/10679/159, 2011.

[9] J.-S. Ahn, D.-K. Jeong, and M.Yang, "Fast three-dimensional programmable two-selector," *Electronics Letters*, vol. 40, pp. 1098–1100, Sept. 2004.

[10] R. P. Brent and H. T. Kung, "A regular layout for parallel adders," *IEEE Trans. Computers*, vol. C-31, pp. 260–264, Mar. 1982.

[11] T. Han and D. A. Carlson, "Fast area-efficient vlsi adders," *Proc. Symp. Computer Arithmetic*, pp. 49–56, 1987.

[12] P. M. Kogge and H. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence relations," *IEEE Trans. Computer*, vol. C-22, pp. 786–793, Aug. 1973.

[13] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *J. ACM*, vol. 27, pp. 831–838, Oct. 1980.

[14] H. F. Ugurdag, F. Temizkan, O. Baskirt, and B. Yuce, "Fast two-pick n2n round-robin arbiter circuit," *Electronics Letters*, vol. 48, pp. 759 –760, June 2012.

[15] O. Keskin, *PCST: Plowing based Carry Save Tree generation*. Master's Thesis, Bahçeşehir University, Turkey, Sept. 2010.

[16] D. Harris, "A taxonomy of parallel prefix networks," *Proc. Asilomar Conf. Signals Syst. Computers*, vol. 2, pp. 2213–2217, Nov. 2003.

[17] O. Baskirt, *New Logic Architecture for Round Robin Arbitration and Their Automatic RTL Generation*. Master's Thesis, Bahçeşehir University, Turkey, June 2008.

VITA

Name Surname : Fatih Temizkan

Address :

Ericam Vision & Defence Technologies Co.

Kemal Nehrozoğlu Cd. GOSB Tecnopark Hi-Tech Binası

Kat:1 A6 Gebze 41480 Kocaeli-Turkey

Birth Place / Year : Rize / 1986

Languages : Turkish (native) - English

High School : Rize Anatolian High School - 2004

BS : Erciyes University - 2009

MS : Özyeğin University - 2012

Name of Program : M.Sc. in Electrical and Electronics Eng.

Publications

- B. Yuce, S. Korkmaz, V. B. Esen, **F. Temizkan**, C. Tunc, G. Guner, I. F. Baskaya, I. Agi, G. Dundar, H. F. Ugurdag, "Synthesis of clock trees for sampled-data analog IC blocks," *East-West Design & Test Symposium*, Kharkov, 2012.
- **F. Temizkan**, H. Sahin, G. Guner, H. F. Ugurdag, S. Gören, "Look-up table based polynomial approximation for exponential functions on ASICs," *Computer Science Student Workshop*, İstanbul, 2012.
- H. F. Ugurdag, **F. Temizkan**, O. Baskirt, and B. Yuce, "Fast two-pick n2n round-robin arbiter circuit," *Electronics Letters*, vol. 48, pp. 759–760, June 2012.

- F. Ugurdag, O. Keskin, C. Tunc, **F. Temizkan**, G. Fici, S. Dedeoglu, "RoCoCo: Row and column compression for high-performance multiplication on FPGAs," *East-West Design & Test Symposium*, Sevastopol, 2011. **(outstanding paper award)**

Work Experience

- Ericam Vision & Defence Technologies Co.
  ASIC/FPGA Design Engineer
  July 2012 - Ongoing
- Özyeğin University
  EE Engineering Department - Teaching & Research Asst.
  Sept. 2010 - July 2012
- Bahçeşehir University
  EE Engineering Department - Teaching & Research Asst.
  Sept. 2009 - Aug. 2010

Honors and Awards

- TUBİTAK-ARDEB scholarship for graduate education (Sep. 2011 - Jun. 2012)
- TUBİTAK-BİDEB scholarship for graduate education (Sep. 2009 - Aug. 2011)
- Ranked 1st in all of Engineering Faculty, Erciyes University (Jun. 2009)
- Erciyes University Engineering Faculty Outstanding Student Honor (Jun. 2009)