M-PICK FIXED-PRIORITY SELECTION AND MUXING

A Thesis

by

Mustafa Tosun

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
January 2017

M-PICK FIXED-PRIORITY SELECTION AND MUXING

Approved by:

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

Assoc. Prof. Sezer Gören Uğurdağ
Department of Computer Engineering
Yeditepe University

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

Date Approved: 12 January 2017

To My Wife...

ABSTRACT

In this thesis, we propose a class of logic architectures for multi-pick (m-pick) fixed-priority arbitration (FPA) and muxing. An m-pick FPA selects the m topmost requests out of n inputs with priority order. Arbiters usually drive multiplexers (muxes). Latency optimization of FPAs and mux trees have usually been handled separately in the literature. However, in some applications with circular data dependencies, it is the combined latency of the arbiter and muxing that needs to be optimized. Moreover, there is an ever growing need for throughput. This requires, for example, network switches that pick and mux m requests per cycle, where $m > 1$. This thesis starts with 1-pick priority based selection and muxing and then generalizes it 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 have implemented the proposed architectures through Perl programs generating Verilog netlists and synthesized them using Synopsys Design Compiler with ARM-Artisan TSMC 180 nm worst case standard-cell library. Through the results we have obtained, we demonstrated the trade-offs in the design of m-pick FPA and muxing.

ÖZETÇE

Bu tezde, çoklu(m)-seçim sabit-öncelikli iş düzenleyici (FPA) ve çoklayıcı (muxing) için bir mantık mimarisi sınıfı önermekteyiz. Bir çoklu seçim FPA, n adet talep girdisinden en yüksek m tanesini öncelik sırasına göre seçer. İş düzenleyiciler genellikle çoklayıcıları (muxes) yönlendirir. FPA'ler ve çoklayıcı ağaçları (mux trees) gecikme optimizasyonu çalışmaları literatürde genellikle ayrı olarak ele alınmıştır. Bununla birlikte, dairesel veri bağımlılıkları olan bazı uygulamalarda, optimize edilmesi gereken iş düzenleyici ve çoklayıcının gecikmesi bir araya getirilmiştir. Bunların dışında, çıktı için gittikçe artan bir ihtiyaç vardır. Bu, örneğin $m > 1$ olduğu zaman, her döngüde m adet talebi seçen ve çoklayan ağ anahtarlarını gerektirir. Bu tez, sabit önceliğe dayanan tekli seçim ve çoklama ile başlar ve sonra çoklu (m) seçim için genelleştirir. "Sınırlanmış Toplayıcı" olarak adlandırdığımız bir mantık oluşturma bloğu, 1-seçim ve 2-seçimli mimarileri basit problemler haline getirerek genellemede önemli bir rol oynamaktadır. önerilen mimarileri, verilog ağ listeleri oluşturan Perl programları vasıtasıyla uyguladık ve bunları ARM-artisan TSMC 180 nm en kötü durum standart hücre kütüphanesi ile Synopsys Design Compiler kullanarak sentezledik. Elde ettiğimiz sonuçlar vasıtasıyla, çoklu (m) seçim FPA ve çoklayıcı dizaynındaki değiş-tokuş dengesi tahlillerini gösterdik.

ACKNOWLEDGEMENTS

First and foremost, I would like to my sincere appreciation to my advisor Assoc. Prof. H. Fatih Uğurdağ for his continuous support, guidance, and motivation throughout my M.S. education and of course my thesis.

I would like to thank Assoc. Prof. Sezer Gören Uğurdağ and Assistant Prof. T. Barış Aktemur, my thesis committee members, for taking the time.

I would also like to thank my lab mate M. Akif Özkan for his involvement in some stages of this thesis and for his constructive feedback, especially on the thesis report and paper work. Without his help, it would not be possible for me to complete this thesis easily.

I would also thank to my other lab mates in Özyeğin University: Aydın Emre Güzel, Vecdi Emre Levent, Mert Kaya, and Waqas Hussain for making my graduate life easier and pleasant.

Last but not least, I would like to thank my wife and my family for their support.

I have been partly supported by TÜBİTAK (under project no. 114E343) between April and September 2014 and since January 2016. I was also fully supported by the same project between October 2014 and January 2016.

TABLE OF CONTENTS

DEDICATION . . . . **iii**

ABSTRACT . . . . **iv**

ÖZETÇE . . . . **v**

ACKNOWLEDGEMENTS . . . . **vi**

LIST OF TABLES . . . . **ix**

LIST OF FIGURES . . . . **x**

I INTRODUCTION **1**

1.1 What is Fixed Priority Selection? 1

1.2 What is Multi-Pick Fixed Priority Arbiter? 3

1.3 Previous Work 4

1.4 Contributions of the Thesis 5

1.5 Outline of the Thesis 7

II 1-PICK ARCHITECTURE **8**

2.1 1-Pick Fixed Priority Encoding Problem 8

2.2 Brute-Force Approach to 1-Pick Selection and Muxing 8

2.3 Proposed 1-Pick Selection and Muxing Architecture 10

2.4 1-Pick Selection and Muxing Versus Dimitrakopoulos's Merged Arbiter Multiplexer 11

2.5 Ladner-Fisher PPN Topology 12

2.6 Other PPN Topologies 13

2.6.1 Kogge-Stone PPN 13

2.6.2 Brent-Kung PPN 13

2.6.3 Han-Carlson PPN 14

2.7 1-pick Selection and Muxing Variants 15

III MULTI-PICK ARCHITECTURE **18**

3.1 2-pick FPE Selection And Muxing 18

3.2 m-pick FPE Selection and Muxing 20

3.3 m-pick Selection and Muxing Variants 21

IV HDL CODE GENERATORS 25

4.1 Code Generator for 1-Pick FPE Selection and Muxing 26

4.2 Code Generator for m-Pick FPE Selection and Muxing 26

V EXPERIMENTAL SETUP AND RESULTS 28

5.1 Experimental Setup 28

5.2 Synthesis Script 29

5.3 Results 29

VI CONCLUSIONS AND FUTURE WORK 36

APPENDIX — EXAMPLE CODE GENERATOR 38

REFERENCES 43

VITA 45

LIST OF TABLES

1 Truth table for the thermometer-coded adder saturated for m = 2 . . 20

2 Timing results for 1-pick FPE Selection and Muxing 31

3 Area Results (Normalized area-timing products) for 1-pick FPE Selection and Muxing 32

4 Timing results for 2-pick FPE Selection and Muxing 32

5 Area Results (Normalized area-timing products) for 2-pick FPE Selection and Muxing 33

6 Timing results for 3-pick FPE Selection and Muxing 33

7 Area Results (Normalized area-timing products) for 3-pick FPE Selection and Muxing 34

8 Timing results for 4-pick FPE Selection and Muxing 34

9 Area Results (Normalized area-timing products) for 4-pick FPE Selection and Muxing 34

10 Timing results for 5-pick FPE Selection and Muxing 35

11 Area Results (Normalized area-timing products) for 5-pick FPE Selection and Muxing 35

LIST OF FIGURES

1 (a) n2n FPE (b) n2logn FPE 2

2 n2n 2-pick FPE 3

3 Priority selector with L expressions for m requests. Source: Chiu at el. (2012) [10] 4

4 Brute-force approach to 1-pick Selection and Muxing [15] 9

5 Overlapping selection of “r”equest and muxing of the associated “d”ata lines 9

6 Muxing with a one-hot select can be done by an OR tree with masked inputs 10

7 The 1-pick topology in Figure 4 implemented with small FPEs (n=2) and ANDed OR trees (AOR) 11

8 Merged Arbiter Multiplexer 12

9 Microarchitecture of LF PPN with OR gates 13

10 Microarchitecture of KS PPN with OR gates 13

11 Microarchitecture of BK PPN with OR gates 14

12 Microarchitecture of HC PPN with OR gates 15

13 The 1-pick topology in Figure 5 with k = 4. (Figure 5 has k = 2) . . 16

14 The 2-pick version of k=4 topology in Figure 6 18

15 (a) Microarchitecture of 2-pick FPE for k = 8 and (b) PE logic 19

16 Multi-Pick FPE Selection and Muxing Architecture 23

17 (a) Microarchitecture of m-pick FPE for k = 8 and (b) PE logic (thermometer-coded saturated adder for m) 24

CHAPTER I

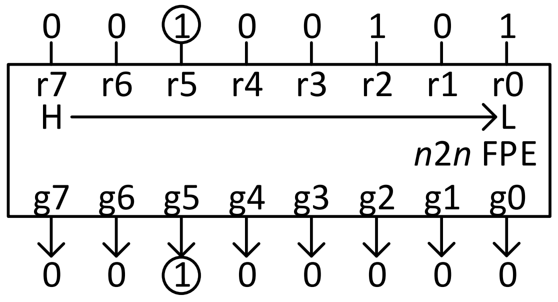
INTRODUCTION

In this chapter, the main concepts are explained to familiarize the readers with the subject of this thesis. In this direction, this chapter also provides the previous work, the summary of contributions of this study, and outline of thesis. This thesis about, a family of circuit topology where priority encoding picks one or m (2 or more picks) fixed-priority selection and muxing.

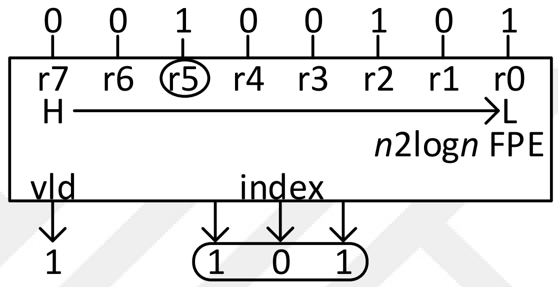
1.1 What is Fixed Priority Selection?

We use arbiters to select between incoming requests. An arbiter is used to coordinate the usage of a particular set of shared resources among multiple requester as well as in dispatch logic where the purpose is load balancing among multiple requester. The shared resources may be ports (i.e., on-chip network switches), buses, processors, memory blocks, etc., while requestors may be packet queues, processors, etc. Depending on the application, the requestors may require to be treated fairly [1], i.e., with equal prioriy, or with different priorities [2], which could be static or varying. Fixed Priority Encoders (FPE) is an arbiter where the priorities of the requester are different and static (i.e., fixed). On the other hand, an FPE is not a fair arbiter but it may be utilized to create fair arbiters, such as round-robin arbiters [1]. While a basic arbiter launches one new task on a shared resource every clock cycle, higher throughput systems require multiple tasks dispatched per cycle. That results in a need for multipack arbiters circuits [3].

In Figure 1(a) and 1(b), H denotes the position of the highest-priority input, whereas L denotes the position of the lowest-priority input. FPE starts searching from H to L, for the purpose of an $n2n$ FPE that is to find the first input port that



(a)



(b)

Figure 1: (a) n2n FPE (b) n2logn FPE

is 1 (i.e., is requesting) out of n bits where Figure 1 shows 1-pick priority encoding (i.e., priority selection) problem. This problem is the same "First 1 Finder" problem. Also this problem has variants, such as first 0 finder or with priorities going down from right to left. Moreover, the output may be one-hot with n bits in Figure 1(a) or binary with *logn* (log base 2) bits in Figure 1(b). In the binary case, the output gives the index of the first input port that is 1 when searched from the left. However, we need a "vld" (i.e., valid) pin (which becomes 0) in the case when all inputs are 0. In the one-hot case, the output is all 0s when all inputs are 0.

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

1.2 *What is Multi-Pick Fixed Priority Arbiter?*

Similar to 1-pick FPE, m-pick (multi-pick) FPE searches the indexes of m highest priority of active requests out of n inputs. The variable "m" can get the values between 1 and n. In this thesis, the terms that "multi-pick" and "m-pick" are used interchangeably. For instance, if m will take the value 2, FPE becomes 2-pick FPE according to the terminology of this thesis and searches for first 2 requests. If m equals 3, this time the problem will be a 3-pick FPE.

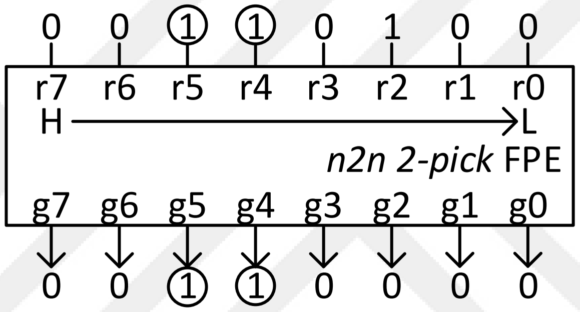


Figure 2: n2n 2-pick FPE

Figure 2 represents a 2-pick FPE example. The 2-pick FPE grants two highest-priority active requests based on the position of the requests. In Figure 2, the requests are represented at the input locations "rn" and the requests are r5, r4 and r2. According to the position of H in Figure 2, requests r5 and r4 are granted by the 2-pick FPE. For this example, if m would be equal to 3, the 3-pick FPE, requests r5, r4 and r2 would be granted by the 3-pick FPE.

This thesis attacks the 2-pick [4]-[5] and m-pick generalized version of that problem but with muxing combined. Priority encoders and arbiters usually drive multiplexers (muxes). In this thesis, we propose a family circuit topology where priority encoding picks one or m requests and takes place in parallel with muxing.

1.3 Previous Work

In the literature, despite a plethora of work (for logic optimization) on arbiters based on FPE circuits, there are only a few works that combine FPE selection with muxing. The first work that comes to mind in the area of hardware implementation of arbiters is Gupta and McKeown's work, which is on 1-pick round-robin arbitration. [6]. Logic optimization of FPEs have been studied in several works [7], [8]. Gupta and McKeown solved the Round Robin Arbiter problem by dividing the problem into small FPE problems to utilize the advantage of the FPE approach.

The work of Dimitrakopoulos' Merged Arbiter Multiplexer [9] (explained in detail in section 2.4.) proposes a new RTL soft macro that can concurrently handle arbitration and multiplexing. Hence, they try to simplify the design of low latency and high-radix switches. Their design is based on dynamic priority arbiter and multiplexer. For the dynamic priority arbitration logic, FPE has been utilized. However, we realized that the timing performance of their design can be improved by modifying how the arbitration and muxing is combined.

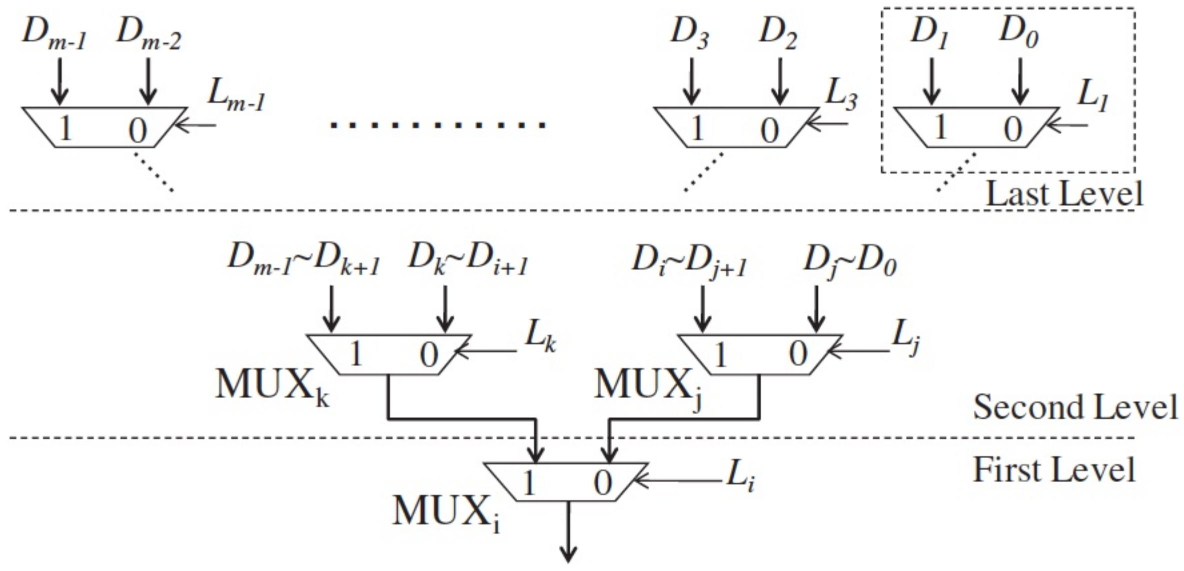


Figure 3: Priority selector with L expressions for m requests. Source: Chiu at el. (2012) [10]

Moreover, due to an unbalanced propagation path, inefficient throughput in synchronous clock systems occurs by the nature of priority policy. In order to achieve higher speed, the priority scheme of J.-C. Chiu and K.-M. Yang [10] enhances extremely unbalanced delay between the highest and lowest weight by integrating the multiplexer-based data selector with the priority encoder. This study is another work that focuses on concurrently handle arbitration and muxing problem with extra balanced propagation path solution for efficient throughput in synchronous clock systems. And also the critical path of the priority selector scheme has only $MD*((log2n)$ delay on the multiplexer path that has m requests as shown in Fig. 3.

With our architecture, we show that there is improvement in area as well as timing performance. Moreover, we extend our architecture to multi-pick selection and combine that also with muxing.

1.4 Contributions of the Thesis

Priority encoders and arbiters generally drive multiplexers. In the literature, latency optimization of priority encoders and multiplexer trees have been addressed separately. However, for some applications, which include circular data dependencies, the combined latency of the arbiter and muxing are required to be optimized. In addition to that, there is an ever growing need for the throughput. While a basic arbiter launches one new task on a shared resource every clock cycle, higher throughput systems require multiple tasks dispatched per cycle. That results in a need for multi-pick arbiters circuits that pick and multiplex more than one request per cycle. In this direction, this work focuses on these requirements and attacks the 2-pick problem [4]-[5] but with muxing combined and also with m-pick version of that. Thus, the main contribution of this work is to generate a family of circuit topology where priority encoding picks one or several (m) requests, and that takes place in parallel with muxing.

An arbiter usually multiplexes data to a wide (b-bit) bus based on its arbitration decision. Sometimes, arbitration and muxing can be pipeline. However, this requires additional area and power consumption due to the additional b flip-flops used for pipeline, which may not be desired if b is large. It is also possible that the future requests depend on the processing done on the muxed data (i.e., circular data dependency). In such scenarios, it is best to overlap arbitration (i.e., priority based selection) and muxing. Normally, we need to know which requester is picked before we can mux, because a regular mux requires a "select" value. In this work, we show multiple possibilities for parallelization and hence overlapping the two operations (selection and muxing). Rather than designing a fixed circuit, we have created a circuit generator in Perl that takes in several parameters and outputs the RTL of the design in Verilog. The parameters we take as input are the number of input ports (n), bit-width of muxing (b), a design parameter called k, which we will explain in the following parts of this thesis.

Contributions of this thesis are the following:

- We propose a data selection circuit, but different than available works in the literature, which was designed to carry out FPE Selection parallel with Muxing.
- We generalize FPE Selection and Muxing from 2-pick to m-pick, which grants up to m requests. We employed Ladner-Fisher PPN topology [11], [12], [13], [14] in our m-pick FPE Selection and Muxing implementation.
- We have constructed HDL code generators for all FPE Selection and Muxing circuits and their variants.
- We have also developed scripts for automated verification and synthesis.
- We provide a rich set of area/timing results using an iterative synthesis script.

1.5 Outline of the Thesis

Chapter II presents the 1-pick priority encoding problem, than a brute-force approach to 1-pick selection and muxing, and followed by our proposed 1-pick version of the FPE Selection and Muxing architecture, Ladner-Fisher (LF) PPN topology, Dimitrakopoulos' Merged Arbiter Multiplexer versus our 1-pick FPE Selection combined with Muxing solution and variants of 1-pick Selection and Muxing logic. Chapter III includes 2-pick FPE Selection and Muxing, m-pick version of FPE Selection and Muxing and variants of m-pick Selection and Muxing logic. Chapter IV describes our Verilog code generators for all our 1-pick to m-pick architectures and their variants. Chapter V describes our experimental setup with our HDL code generators and our iterative synthesis flow script. Also chapter V shows our timing and area synthesis results for all automatically generated RTL codes. At the end, in chapter VI, we evaluate the performances based on our experimental results and also presents possible future works.

CHAPTER II

1-PICK ARCHITECTURE

In this chapter, we first present 1-pick fixed priority encoding problem. Then, we present the brute-force approach to 1-pick selection and muxing, and finally our 1-pick selection and muxing solution and view of micro-architecture of the Dimitrakopoulos and Kalligeros' merged arbiter multiplexer architecture.

2.1 1-Pick Fixed Priority Encoding Problem

This problem is the same "First 1 Finder" problem. The purpose is to find the first input port that is 1 (ie., is requesting) out of n bits starting from highest priority input. This problem has variants, such as first 0 finder or with priorities going down from lowest priority input to highest priority input. Also, the output maybe one-hot with n bits or binary with *logn* (log base 2) bits. In the binary case, the output gives the index of the first input port that is 1 when searched from highest priority. However, we need a "vld" (ie., valid) pin (which becomes 0) in the case when all inputs are 0. In the one-hot case, the output is all 0s when all inputs are 0.

2.2 Brute-Force Approach to 1-Pick Selection and Muxing

A brute-force approach to the combined n-input selection and b-bit muxing problem uses an FPE with binary output feeding an n:1 mux in Figure 4. In this simple-minded solution, the latency (i.e., critical path) is equal to the sum of the latencies of FPE and mux.

The initial inspiration of this work is shown in Figure 5, which is a slight improvement over an idea presented in [16]. If we do not view the n:1 mux as a black box and rather expose the tree of 2:1 muxes implementing it, the internal individual 1-bit

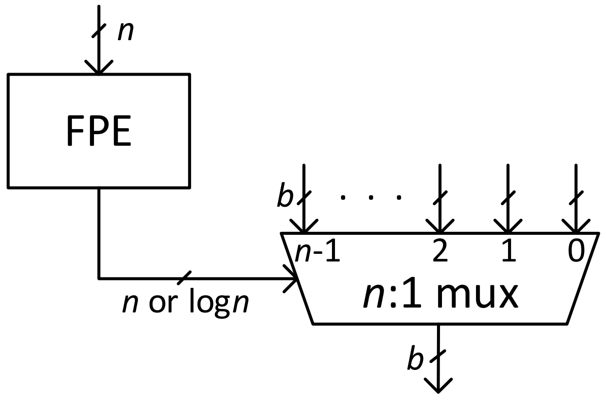


Figure 4: Brute-force approach to 1-pick Selection and Muxing [15]

select signals of these 2:1 muxes can be generated more easily. That is much better than generating a *logn* bit select signal for the bigger n:1 mux and then decoding internally in the mux and distributing it to the smaller muxes implementing it, because, in this approach, the generation of select signals can overlap in time with the muxing they control.

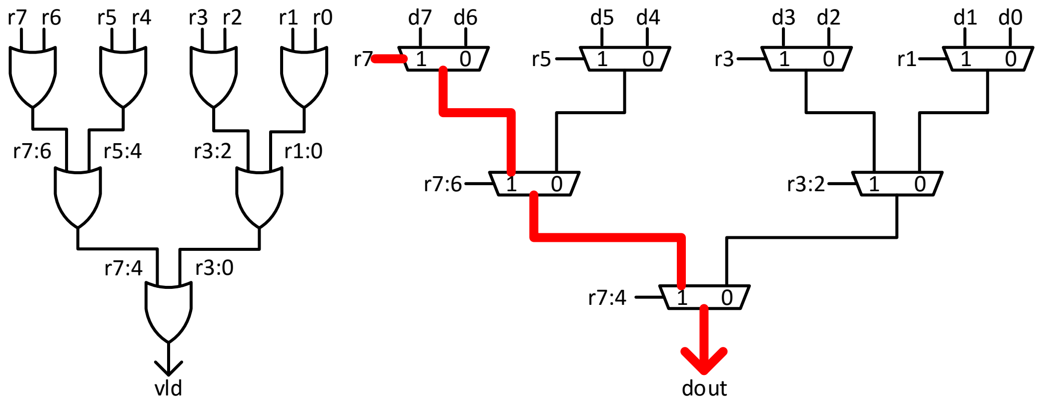


Figure 5: Overlapping selection of “r”equest and muxing of the associated “d”ata lines

2.3 Proposed 1-Pick Selection and Muxing Architecture

The fastest (i.e., lowest latency) implementation of FPE followed by mux (FPEmux) employs the *n*2*n* FPE, which is best implemented with parallel prefix networks (PPN) [1]. Detailed explanation of Ladner-Fisher PPN can be found at subsection 2.4. Although *n*2*logn* FPE offers an area efficient solution, *n*2*n* FPE with Ladner-Fisher PPN is expected to beat it in both area and latency. Using an *n*2*n* FPE has one more advantage. Since the n2n FPE has one-hot output, the mux following the FPE can be replaced by an OR tree after the inputs are masked (ANDed) by the output bits of the FPE (g0 through g7, where "g" is short for "grant") as shown in Figure 6.

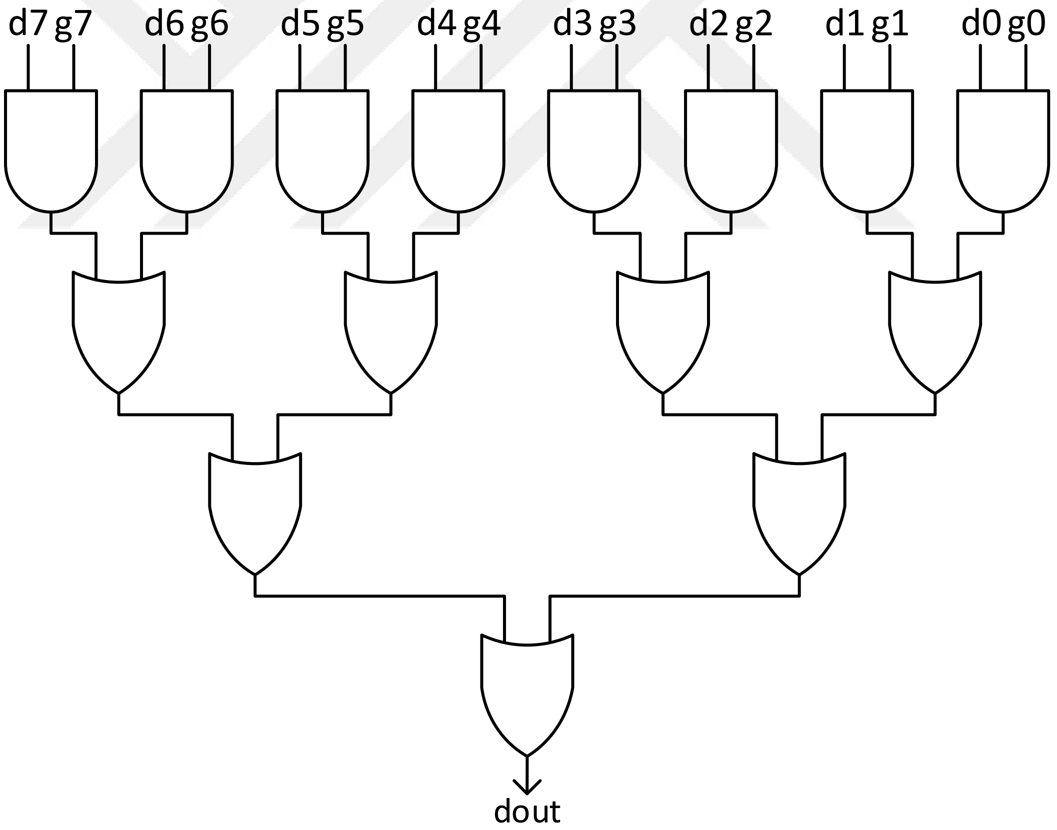


Figure 6: Muxing with a one-hot select can be done by an OR tree with masked inputs

The highlighted path in Figure 6 that a mux can be replaced by ANDs and ORs, the muxes in Figure 5 cannot be replaced by AND and OR gates. That is because, for example, (r7,r6) does not form a one-hot pair. None of the paired "r"equest lines form a one-hot pair. Another example is (r7:6, r5:4). However, if we convert the OR tree on the left in Figure 5 that generates the select signals into little (n=2) FPEs (which it almost already is), then the muxes can be optimized into AND and OR gates. Figure 7 shows what Figure 5 becomes after such conversion. The critical path in this circuit (highlighted) goes through one small FPE and then the muxes implemented through ANDed OR trees (AOR). Our results will show that this gives slightly better results than the topology in 5 as AOR is more efficient than MUX2 in timing.

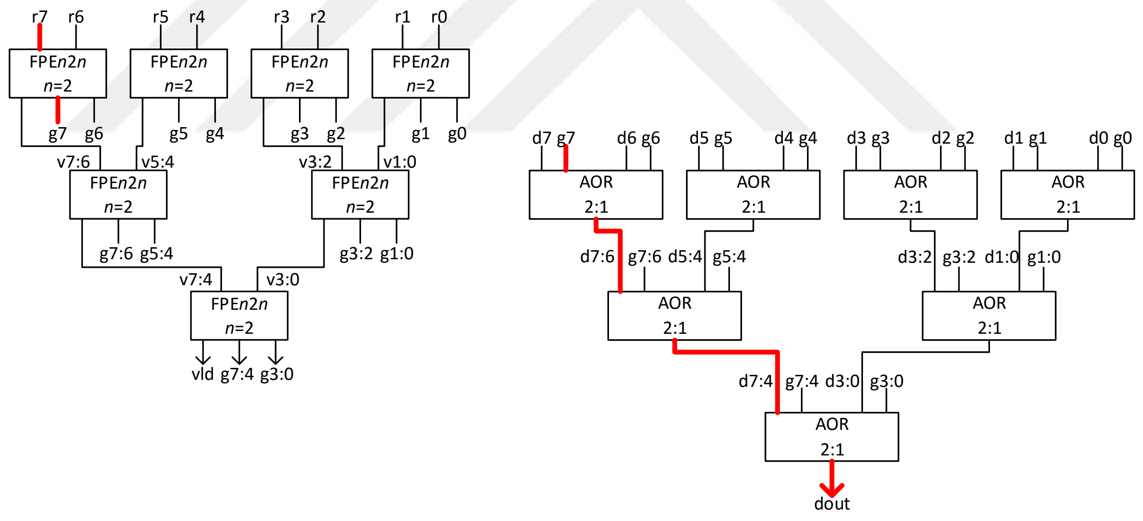


Figure 7: The 1-pick topology in Figure 4 implemented with small FPEs (n=2) and ANDed OR trees (AOR)

2.4 *1-Pick Selection and Muxing Versus Dimitrakopoulos's Merged Arbiter Multiplexer*

Our 1-pick Selection and Muxing uses a similar topology to Dimitrakopoulos's Merged Arbiter Multiplexer [9] in Figure 8. The difference is that our critical path is extremely optimized with the help of small $n2n$ FPEs combined with AOR trees instead of multiplexer trees. Furthermore, 1-pick Selection and Muxing is extendable to m-pick Selection and Muxing.

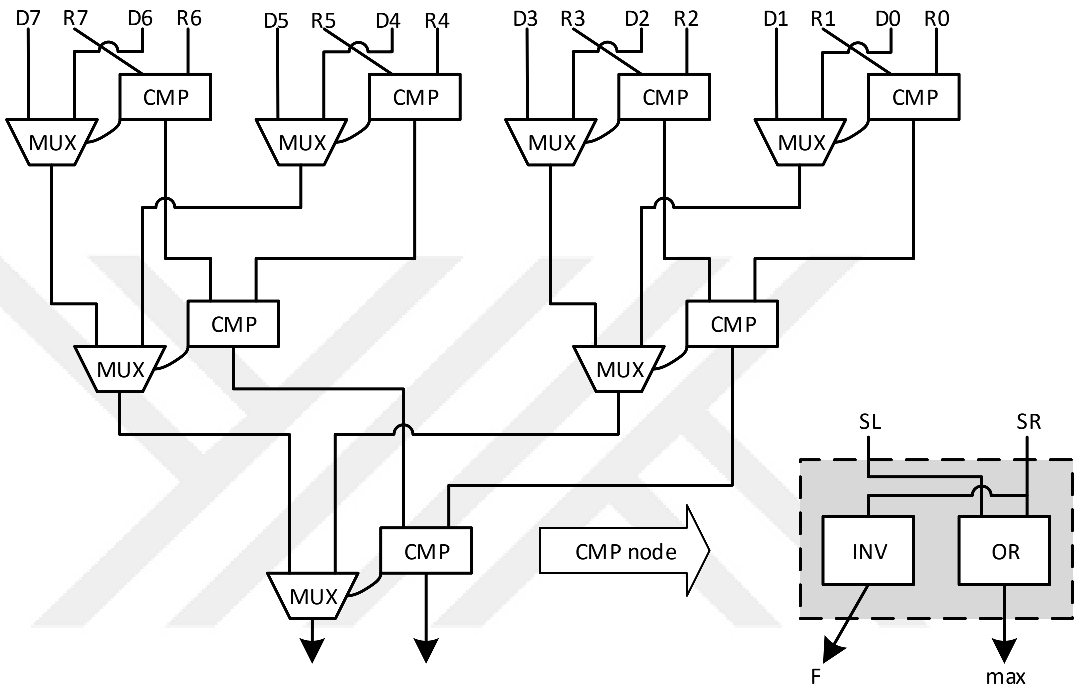


Figure 8: Merged Arbiter Multiplexer

We implement 1-pick selection's $n2n$ FPE with Ladner-Fisher (LF) PPN. These will be explained in the next section.

2.5 Ladner-Fisher PPN Topology

LF PPN for n = 8 bit input is given in Figure 9. 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 $O(logn)$ for timing as in the case of LF. However, the main disadvantage of LF is higher fan-out which is $n/2$ and this can be affects timing and area negatively. To drive multiple gates, bigger driving gates are used. Hence, these bigger driving gates increase total area and timing.

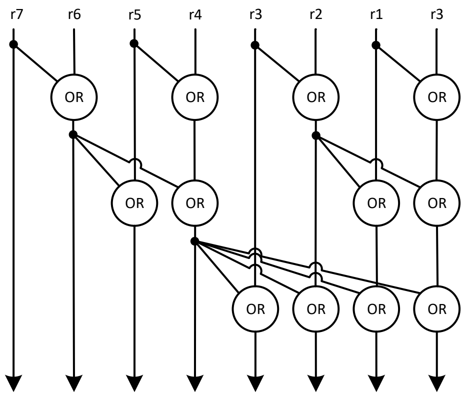


Figure 9: Microarchitecture of LF PPN with OR gates

2.6 Other PPN Topologies

2.6.1 Kogge-Stone 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. Figure 10 shows KS architecture that is drawn for n = 8 bits input.

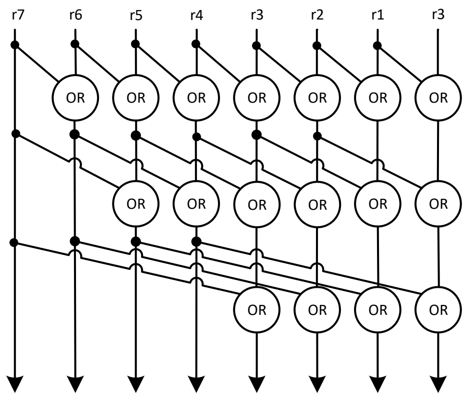


Figure 10: Microarchitecture of KS PPN with OR gates

2.6.2 Brent-Kung PPN

The complexity of BK architecture is $O(2n)$ for area and $O(2 \log n)$ for timing. 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 11 shows BK architecture that is drawn for n = 8 bits input.

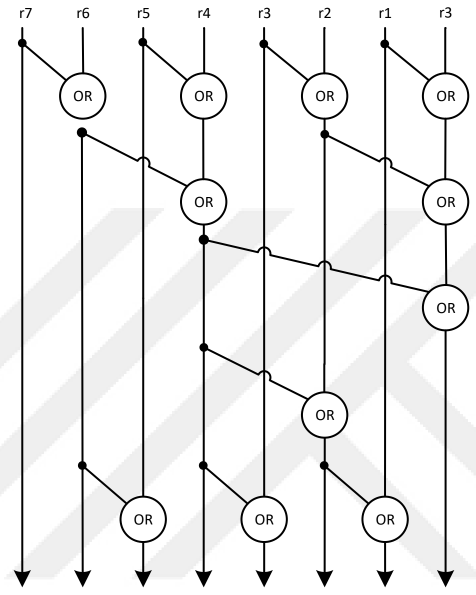


Figure 11: Microarchitecture of BK PPN with OR gates

2.6.3 Han-Carlson PPN

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 12 shows HC architecture that is drawn for n = 8 bits input.

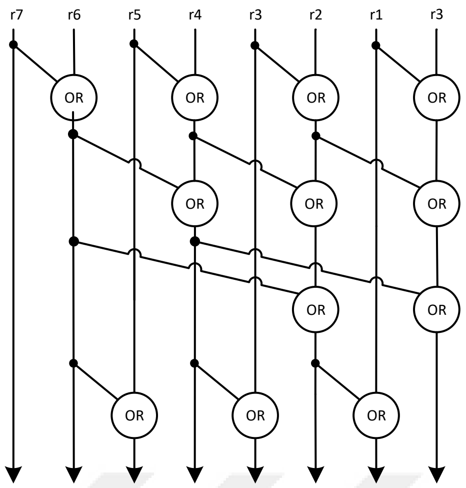


Figure 12: Microarchitecture of HC PPN with OR gates

2.7 1-pick Selection and Muxing Variants

After making a transition to the topology in Figure 7 from Figure 5, we have realized that Figure 7 is only a special instance of a family of topology, which can be characterized by an additional parameter that we call k. In Figure 7, k is equal to 2. On the other hand, k is equal to 4 in Figure 13. That means that we divide up the selection circuit on the left into small FPEs with k inputs and k outputs and implement the n-input mux on the right as a tree of k-input muxes, each of which is implemented as an OR tree (with AND gates at the inputs). The last level of the FPE tree as well as the mux tree may have a node with less than k inputs. In Figure 13, n = 8 and k = 4, and that exact situation happens. The critical path of the circuit in Figure 13 is very similar to that of Figure 7, which spans one small FPE with k inputs and the complete mux tree (realized with AORs). As a k gets larger, the part of the critical path through the small FPE becomes larger, while the part through the mux tree gets smaller.

In our experimental setup, we prepare 8 variants of 1-pick Selection And Muxing based on the combinations of different k divided small FPEs and k divided small mux

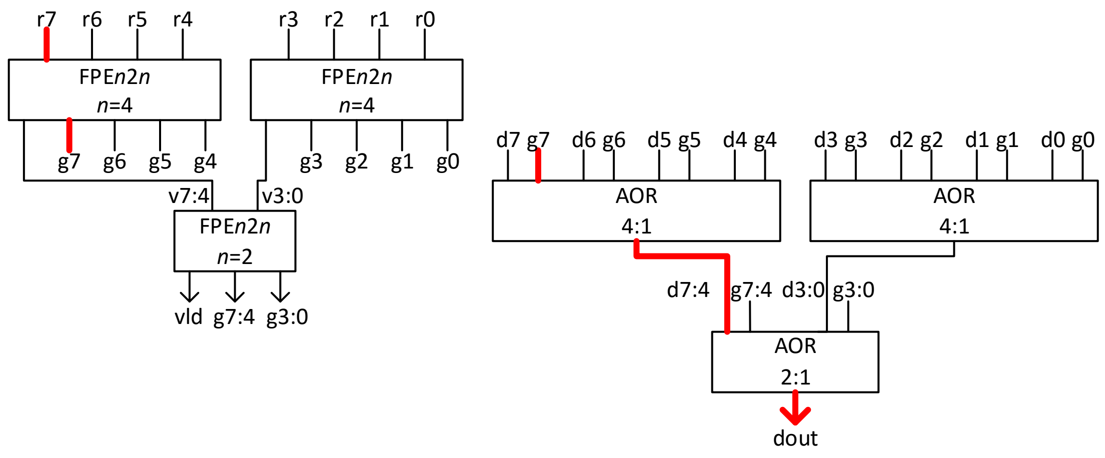


Figure 13: The 1-pick topology in Figure 5 with k = 4. (Figure 5 has k = 2)

trees. These are listed below.

- MT: Implemeted with brute-force approach selection and muxing circuit.
- k = 4: Implemented with 4 inputs and 4 outputs small FPEs and muxing circuit for data selection.
- k = 8: Implemented with 8 inputs and 8 outputs small FPEs and muxing circuit for data selection.
- k = 16: Implemented with 16 inputs and 16 outputs small FPEs and muxing circuit for data selection.
- k = 32: Implemented with 32 inputs and 32 outputs small FPEs and muxing circuit for data selection.
- k = 64: Implemented with 64 inputs and 64 outputs small FPEs and muxing circuit for data selection.
- k = 128: Implemented with 128 inputs and 128 outputs small FPEs and muxing circuit for data selection.

- k = 256: Implemented with 256 inputs and 256 outputs small FPEs and muxing circuit for data selection.
- k = 512: Implemented with 512 inputs and 512 outputs small FPEs and muxing circuit for data selection.

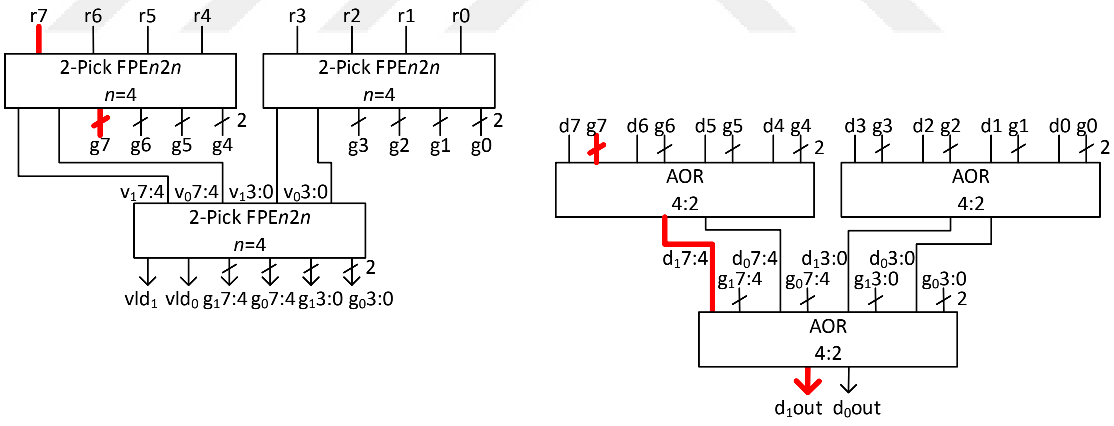
CHAPTER III

MULTI-PICK ARCHITECTURE

In this chapter, we first present 2-pick FPE Selection and muxing. Then, we generalize FPE selection and muxing for any m to construct m-pick FPE selection and muxing, and present its variants.

3.1 2-pick FPE Selection And Muxing

Once one grasps our idea of divide-and-conquer for an n-input problem with k-input FPEs and muxes, it is quite easy to also grasp how we extend it to the 2-pick version of the same problem.



r7 r6 r5 r4 r3 r2 r1 r0
2-Pick FPEn2n n=4
g7 g6 g5 g4 g3 g2 g1 g0
v1 7:4 v0 7:4 v1 3:0 v0 3:0
vld1 vld0 g1 7:4 g0 7:4 g1 3:0 g0 3:0
d7 g7 d6 g6 d5 g5 d4 g4 d3 g3 d2 g2 d1 g1 d0 g0
AOR 4:2
d1 7:4 d0 7:4 d1 3:0 d0 3:0
g1 7:4 g0 7:4 g1 3:0 g0 3:0
d1out d0out
2

Figure 14: The 2-pick version of k=4 topology in Figure 6

The smallest k for the 2-pick problem is 4. Figure 14 shows the 2-pick version of the 1-pick circuit in Figure 13 (with k=4). While the smallest 1-pick FPE is 2:1, the smallest 2-pick FPE is 4:2. The same is true for the muxes. In the bottom stage of the FPE and mux trees of the 2-pick circuit in Figure 14, we have 4:2 modules in place of the 2:1 modules in Figure 13. The 4:2 AORs (i.e., muxes) are actually two parallel muxes with the same inputs but different select signals and outputs. That is why each grant input entering an AOR 4:2 is 2 bits.

Our interpretation of 2-pick FPEn2n micro-architecture is given in Figure 15. In Figure 15(a), 2-pick FPE has a Ladner-Fisher (LF) PPN topology and it consists of priority encoder block with saturated adder (PE). The logic inside PE is also depicted in Figure 15(b). Note that Figure 15(b) presents our interpretation of PE of 2-pick FPEn2n, which consists of a thermometer-coded saturated adder for $m = 2$. That is, PE treats its left and right inputs as two 2-bit unsigned numbers and adds them to produce a 2-bit unsigned output. The PE is an saturated adder for $m = 2$, hence out = min(left_input+right_input,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 saturated adder for $m = 2$.

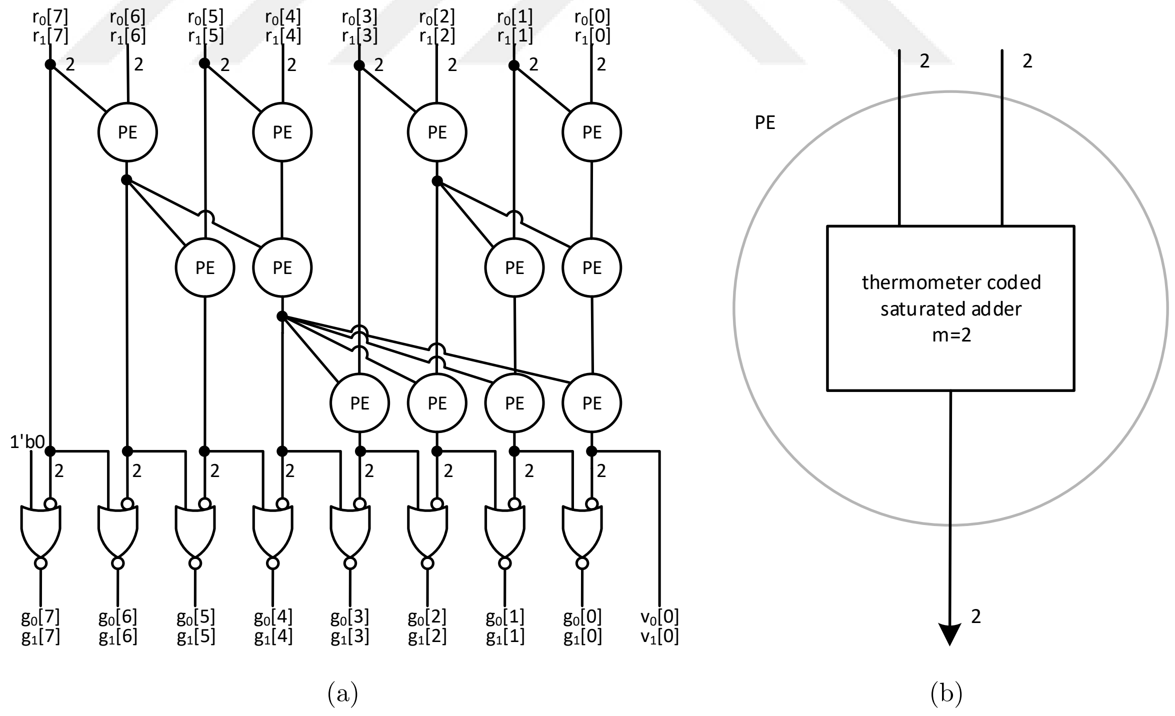


Figure 15: (a) Microarchitecture of 2-pick FPE for k = 8 and (b) PE logic

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

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

3.2 m-pick FPE Selection and Muxing

The proposed m-pick FPE Selection, which grants up to m highest priority requests, and Muxing architecture is an extension of 2-pick FPE Selection and Muxing that is explained previous section in detail. In order to generate this architecture, instead of the saturated adder at 2, we require a saturated adder at m in PE. Therefore, we modified 2-pick FPEs with m-pick FPEs for the selection process. We implement m-pick FPEs with LF PPN that is explained subsection 2.5.

Figure 16 presents the proposed m-pick FPE Selection and Muxing architecture by picturizing these two parallel processes that are selection process and muxing process. The selection part shown in left-side of Figure 16 and muxing part in right-side of Figure 16. The selection process depends on the divide and conquer methodology. According to this methodology, as a first step(level),the whole problem, which includes several requests (inputs), should be divided to n-sized group of requests and then each divided n-sized group of requests fit in the $n2n$ FPE architecture. Therefore, there are two parameters that should be defined at beginning of this level (step). These are k, division parameter, that equals to (n) the size of input and output of small $n2n$ FPE and also m, pick parameter, that is the number of requests to select(i.e., $k >= m$). The division parameter, k, can take (2^n) possible values, at the first level.

By including the first step of divide and conquer methodology, which explained above, this methodology have $\log_2 n - 1$ levels(steps). After the first level, k parameter of n2nFPEs becomes $2 * m = k$ that would be equal to n of $n2n$ FPEs. Similar to selection part, in the muxing process, the input and output of AOR would be equal to k and m of each level of selection process respectively.

Our interpretation of m-pick FPEn2n micro-architecture is given in Figure 17. In Figure 17(a), m-pick FPE has a Ladner-Fisher (LF) PPN topology and it consists of saturated block (PE). The logic inside PE is also depicted in Figure 17(b). Note that Figure 17(b) presents our interpretation of PE of m-pick FPEn2n, which consists of a thermometer-coded saturated adder for $m > 2$. That is, PE treats its left and right inputs as two $m - bit$ unsigned numbers and adds them to produce a $m - bit$ unsigned output. The PE is an saturated adder for $m > 2$, hence out = min(left_input+right_input,m). It is thermometer-coded.

3.3 m-pick Selection and Muxing Variants

In our experimental setup, we prepare 8 variants of m-pick Selection And Muxing based on the combinations of different k divided small FPEs and k divided small mux trees. These are listed below.

- k = 4: Implemented with 4 inputs and 4 outputs small FPEs and muxing circuit for data selection.
- k = 8: Implemented with 8 inputs and 8 outputs small FPEs and muxing circuit for data selection.
- k = 16: Implemented with 16 inputs and 16 outputs small FPEs and muxing circuit for data selection.
- k = 32: Implemented with 32 inputs and 32 outputs small FPEs and muxing circuit for data selection.

- k = 64: Implemented with 64 inputs and 64 outputs small FPEs and muxing circuit for data selection.
- k = 128: Implemented with 128 inputs and 128 outputs small FPEs and muxing circuit for data selection.
- k = 256: Implemented with 256 inputs and 256 outputs small FPEs and muxing circuit for data selection.
- k = 512: Implemented with 512 inputs and 512 outputs small FPEs and muxing circuit for data selection.

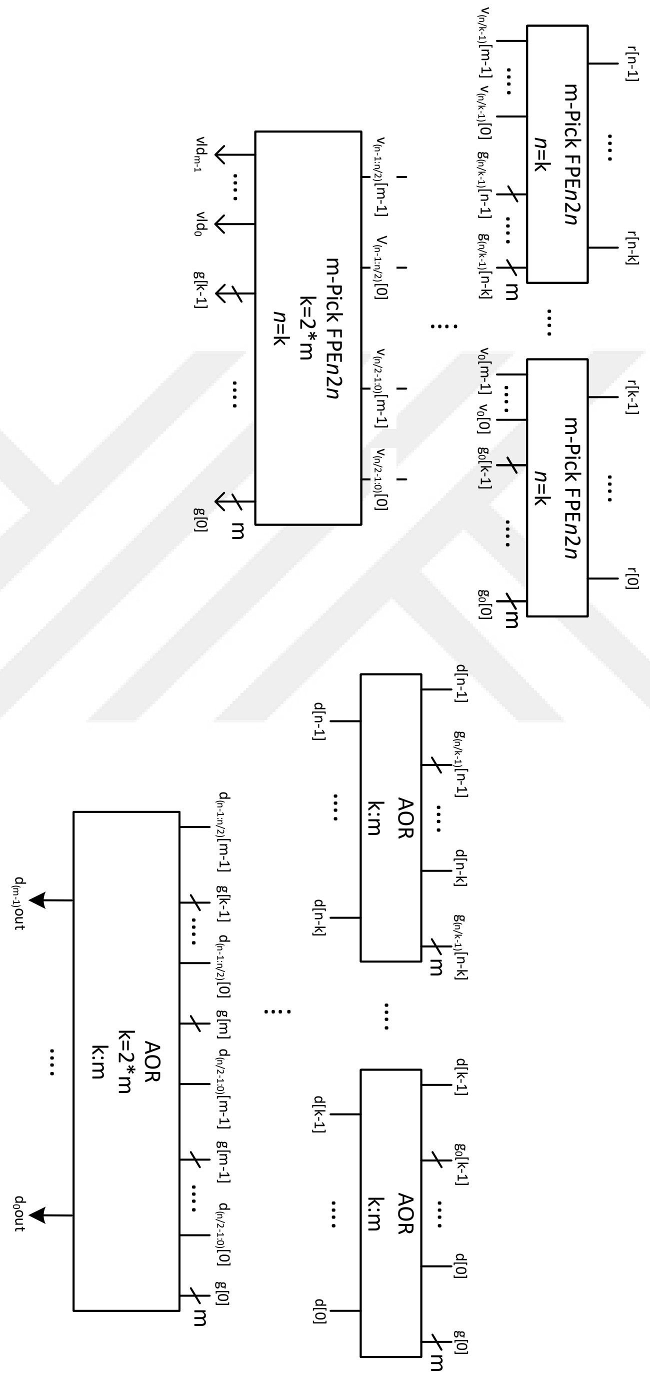
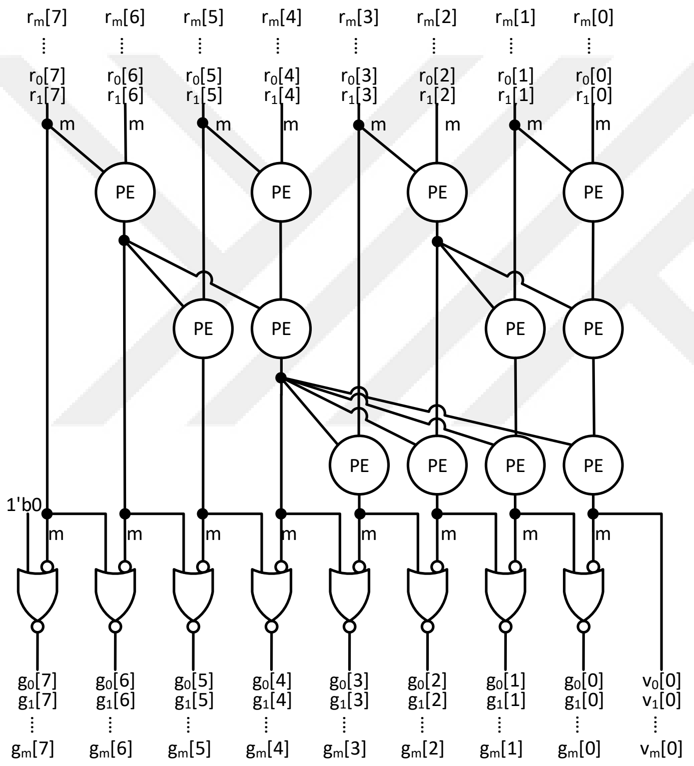
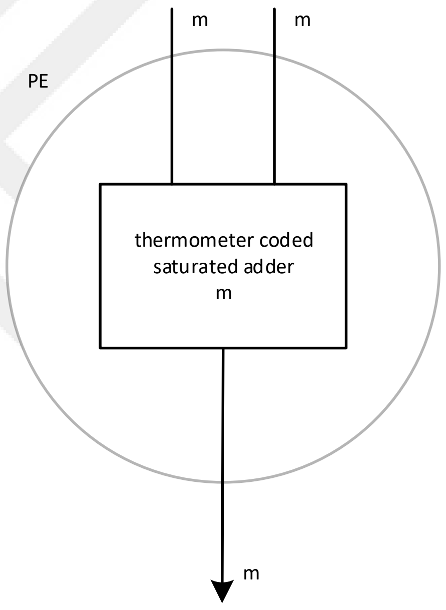


Figure 16: Multi-Pick FPE Selection and Muxing Architecture



rm[7] rm[6] rm[5] rm[4] rm[3] rm[2] rm[1] rm[0]
r0[7] r1[7] r0[6] r1[6] r0[5] r1[5] r0[4] r1[4] r0[3] r1[3] r0[2] r1[2] r0[1] r1[1] r0[0] r1[0]
m
PE
1'b0
g0[7] g1[7] gm[7] g0[6] g1[6] gm[6] g0[5] g1[5] gm[5] g0[4] g1[4] gm[4] g0[3] g1[3] gm[3] g0[2] g1[2] gm[2] g0[1] g1[1] gm[1] g0[0] g1[0] gm[0]
v0[0] v1[0] vm[0]

(a)



(b)

Figure 17: (a) Microarchitecture of m-pick FPE for k = 8 and (b) PE logic (thermometer-coded saturated adder for m)

CHAPTER IV

HDL CODE GENERATORS

During this chapter, Hardware Description Language (HDL) code generators for all proposed methods and their variants are analyzed and explained. We preferred to generate verilog code generator scripts in the Perl scripting language for all methods rather than writing a verilog code for all methods directly.

Underlying reasons for writing scripts instead of writing verilog code, is to make verilog code generation process automate, because all multi-pick FPE architectures will be compared with each other for input bit-widths varied from 16 to 512 as well as pick sizes varied from 2 to 5 in order to minimize the necessary effort for this kind of intensive verilog code writing process. We automated this process by writing the advantage of writing the code generator scripts. In addition to that, the another reason of preferring scripts is that writing veriog code for larger bit-widths and pick sizes is not trivial.

Within this scope, we designed verilog code generator scripts for mux tree based architecture, 9 different variants of 1-pick FPE Selection and Muxing and 8 different variants of m-pick FPE Selection and Muxing method. These scripts get three arguments, the input bit-width (n), the division size (k) of FPEs and the pick size of the m-pick FPE architecture. Therefore, the corresponding verilog code for the each specific method were generated by the scripts. All these generated verilog code are compatible with the verilog-1995 standard. In appendix, there is an example generator script prepared for one of the variants of 1-pick FPE Selection and Muxing.

4.1 Code Generator for 1-Pick FPE Selection and Muxing

As we mentioned above, there are 9 different variants, which are different division size (k) of FPEs, of this 1-pick FPE Selection and Muxing method. These generators get input bit-width, division size and pick size as arguments and create corresponding verilog code as output. Verilog modules generated automatically for this method and its variants are listed below.

- pencN: This module includes the micro-architecture for 1-pick FPE which is implemented with LN PPN topology.
- edge_detectorN: This module is for to find 0 to 1 transition at the selected request.
- levelMux_n_k: This module includes AOR tree with n input and k divided.
- levelPenc_n_k: This module includes FPEs with n input and k divided.
- penqDataSelection: This module is combination of the levelMux and levelPenc modules.
- wrapper: This module is used only in synthesis results. It adds flip-flops at the inputs and outputs of the design to get register to register timing.
- tb: This module is used only in simulation purposes. It verifies the correctness of the design.

4.2 Code Generator for m-Pick FPE Selection and Muxing

There are 8 different variants, which are different division size (k) of FPEs, of this m-pick FPE Selection and Muxing method. These generators get input bit-width, division size and pick size as arguments and create corresponding verilog code as output. Verilog modules generated automatically for this method and its variants are listed below.

- pencN: This module includes the micro-architecture for m-pick FPE which is implemented with LN PPN topology.
- edge_detectorN: This module is for to find 0 to 1 transition at the selected request.
- PE: This modules includes saturated adder logic.
- levelMux_n_k: This module includes AOR tree with n input and k divided.
- levelPenc_n_k: This module includes FPEs with n input and k divided.
- penqDataSelection: This module is combination of the levelMux and levelPenc modules.
- wrapper: This module is used only in synthesis results. It adds flip-flops at the inputs and outputs of the design to get register to register timing.
- tb: This module is used only in simulation purposes. It verifies the correctness of the design.

CHAPTER V

EXPERIMENTAL SETUP AND RESULTS

In this chapter, the experimental setup and results for 2 FPE architectures (with their variants) are explained and presented. The results of the experiments are provided in terms of timing and normalized area-timing products. This chapter also includes detailed discussion of our synthesis script.

5.1 Experimental Setup

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

In order to automate verification and synthesis processes, the script, which designed in Perl, were used. Pick size, division factor and bit-widths are taken as input arguments of the method by verification script. Then, the script calls the desired HDL code generator script to generate verilog codes with its test bench and contents whether the design passes the test bench or not. The working steps of the test bench as in the following:

1. It creates the random input vectors for the design
2. It controls the output of the design whether it is correct or not.
3. An error message would be occurred by the test bench and verification will be stop.

If the outputs of the design are correct, the synthesis process will start by synopsys design compiler and the results will be saved to a result file. Then a new iteration would be started by the test bench. It repeats this several times while utilizing new input vectors in each iteration.

5.2 Synthesis Script

Synthesis script, which is an iterative synthesis script, is written in the TCL scripting language for synopsys DC. Each design is synthesized by 4 times by this script. The working steps of the script are as in the following: First, the desired clock period is set to 0.1 ns (which is impossible to meet) for the first iteration. Then 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 utilized. At the last iteration (4th iteration), the last met clock * 0.8 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 recorded. Based on the our experiment, 4 iterations provides the optimum timing result. If we increase the number of iterations, we realize that there is not a timing improvement at results. 4 iterations were also utilized to save computation time.

5.3 Results

We achieved results for various input bit-widths (16, 32, 64, 128, 256, and 512), the division parameter (2, 4, 8, 16, 32, 64, 128, 256, and 512) and pick size (2, 3, 4, and 5) for all 9 different divided FPE variants.

A wrapper module around the design such that all FPE inputs are taken from flops and all outputs drive flops, are utilized for synthesis. So, our timing results are register-to-register and include register's clock-to-Q delay and setup-time (Note: do not include clock skew and jitter).

Moreover, instead of giving only area results, the normalized product of area and timing are provided. With utilizing more gates and higher drive and hence larger gates, the shorter clock period is achieved by synthesis tools.

Table 2, 4, 6, 8, and 10 present the timing results for pick sizes 1, 2, 3, 4, and 5 respectively. Table 3, 5, 7, 9, 11 present the area results which are normalized area-timing products, for pick sizes 1, 2, 3, 4, and 5 respectively. For each column, the bold numbers shows the smallest result for that input bit-width.

If we analyze the table 2 for timing results for 1-pick FPE Selection and Muxing, we can see that division parameter k when equal to 2 is the most efficient architecture for timing performance between all k variants and MT. However, the performance of MT is close to k = 2. When k selects equal to number of ports the worst timing performance would be occurred. According to table 3, for 1-pick FPE Selection and Muxing, the results of area-timing products shows the best performance when k is set to 2 except number of ports is equal to 512. For the number of ports as 512, the best result is achieved k as 4.

For 2-pick FPE selection and Muxing, table 4 represents the timing results that shows the best timing performance at set k to 8, again except the case of that the number of ports equal to 512. In spite of the dissimilar performance result at the case of 512 ports, the area-timing product results in table 5 are achieved their best performances at k is equal to 8 for all number of input-width variants.

According to table 6,8,10, when the value of pick size, m, increases, the timing performance shows the best results at the k that is close to value of the input-width size. In spite of this attitude of timing results, when for all increased m cases is investigated, at each case if the input-width size increase, the best performance of area-timing products are occurred at the value of k that is smaller than input-width size. In order to observe these analyses, tables 7, 9 and 11 can be examined. For instance, for 4-pick FPE Selection and Muxing timing results for input-width sizes of 16 and 32 provide the best timing performance, when k is equal to the exact number of input-width size. Moreover, for the remaining number of input-width size cases, k value at one-down position respectively the value of input-width size gives the best timing performance results. In order to clarify, the number of input-width size is equal to 512, the best timing performance is occurred at k = 256 and the best area-timing product result k = 32.

We observe that we can do 2-pick selection and muxing with around only 11% extra latency compared to the numbers for doing only selection [17]. Also, again in 2-pick case, we improve the latency by a little over 10% compared to the brute-force approach combined with the optimization.

Table 2: Timing results for 1-pick FPE Selection and Muxing

Methods	**Number Of Ports**					
	16	**32**	**64**	**128**	**256**	**512**
Mux Tree	1.590	1.795	1.960	2.170	2.356	2.530
k = 2	**1.580**	**1.770**	**1.930**	**2.106**	**2.310**	**2.505**
k = 4	1.715	1.940	2.075	2.300	2.430	2.660
k = 8	1.860	1.970	2.200	2.405	2.560	2.720
k = 16	1.990	2.216	2.312	2.470	2.635	2.904
k = 32	-	2.320	2.515	2.620	2.795	3.000
k = 64	-	-	2.650	2.888	3.016	3.178
k = 128	-	-	-	2.995	3.270	3.390
k = 256	-	-	-	-	3.430	3.660
k = 512	-	-	-	-	-	3.770

Table 3: Area Results (Normalized area-timing products) for 1-pick FPE Selection and Muxing

Methods	**Number Of Ports**					
	16	**32**	**64**	**128**	**256**	**512**
Mux Tree	1.241	1.406	1.709	1.177	1.301	1.352
k = 2	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	1.016
k = 4	2.766	3.121	1.936	1.420	1.483	**1.000**
k = 8	3.278	3.433	1.870	1.409	1.472	1.005
k = 16	3.604	3.769	2.407	2.056	1.999	1.540
k = 32	-	4.850	3.937	2.162	2.499	1.914
k = 64	-	-	4.346	3.303	3.496	2.590
k = 128	-	-	-	3.789	4.025	2.995
k = 256	-	-	-	-	3.880	3.263
k = 512	-	-	-	-	-	3.672

Table 4: Timing results for 2-pick FPE Selection and Muxing

Methods	**Number Of Ports**					
	16	**32**	**64**	**128**	**256**	**512**
k = 4	2.605	3.165	3.660	4.080	4.620	5.075
k = 8	**2.494**	**2.910**	**3.300**	**3.770**	**4.140**	4.622
k = 16	2.564	2.968	3.320	3.840	4.230	**4.606**
k = 32	-	3.160	3.550	3.810	4.220	4.720
k = 64	-	-	3.760	4.106	4.286	4.632
k = 128	-	-	-	4.278	4.660	4.870
k = 256	-	-	-	-	4.790	5.235
k = 512	-	-	-	-	-	5.315

Table 5: Area Results (Normalized area-timing products) for 2-pick FPE Selection and Muxing

Methods	**Number Of Ports**					
	16	**32**	**64**	**128**	**256**	**512**
k = 4	2.020	2.835	1.910	2.266	2.348	2.052
k = 8	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
k = 16	3.010	3.026	1.633	1.177	1.157	1.081
k = 32	-	4.444	2.810	2.051	1.406	1.220
k = 64	-	-	3.888	3.437	2.865	1.629
k = 128	-	-	-	3.795	3.491	2.648
k = 256	-	-	-	-	3.856	3.374
k = 512	-	-	-	-	-	3.655

Table 6: Timing results for 3-pick FPE Selection and Muxing

Methods	**Number Of Ports**					
	16	**32**	**64**	**128**	**256**	**512**
k = 4	3.530	4.450	5.445	6.415	7.345	8.392
k = 8	2.994	3.945	4.922	5.880	6.820	7.845
k = 16	**2.795**	3.458	4.440	5.370	6.385	7.330
k = 32	-	**3.455**	**4.020**	4.870	5.782	6.768
k = 64	-	-	4.120	**4.620**	5.430	6.268
k = 128	-	-	-	4.812	**5.280**	5.970
k = 256	-	-	-	-	5.440	**5.925**
k = 512	-	-	-	-	-	6.235

Table 7: Area Results (Normalized area-timing products) for 3-pick FPE Selection and Muxing

Methods	**Number Of Ports**					
	16	**32**	**64**	**128**	**256**	**512**
k = 4	2.843	2.892	3.596	3.201	2.616	2.308
k = 8	**1.000**	**1.000**	1.531	1.541	1.491	1.522
k = 16	1.726	1.128	**1.000**	**1.000**	1.034	1.041
k = 32	-	2.043	2.185	1.188	**1.000**	**1.000**
k = 64	-	-	3.631	2.688	1.285	1.077
k = 128	-	-	-	3.584	2.697	1.564
k = 256	-	-	-	-	3.448	2.886
k = 512	-	-	-	-	-	3.459

Table 8: Timing results for 4-pick FPE Selection and Muxing

Methods	**Number Of Ports**					
	16	**32**	**64**	**128**	**256**	**512**
k = 8	3.400	4.522	5.660	6.822	7.944	9.160
k = 16	**3.000**	3.965	5.115	6.233	7.410	8.572
k = 32	-	**3.840**	**4.654**	5.702	6.875	8.085
k = 64	-	-	4.62	**5.330**	6.410	7.585
k = 128	-	-	-	5.575	**6.120**	7.190
k = 256	-	-	-	-	6.305	**7.095**
k = 512	-	-	-	-	-	7.335

Table 9: Area Results (Normalized area-timing products) for 4-pick FPE Selection and Muxing

Methods	**Number Of Ports**					
	16	**32**	**64**	**128**	**256**	**512**
k = 8	2.094	3.029	2.599	2.219	1.881	1.879
k = 16	**1.000**	**1.000**	**1.000**	**1.000**	1.104	1.134
k = 32	-	2.059	1.620	1.171	**1.000**	**1.000**
k = 64	-	-	3.467	2.376	1.494	1.318
k = 128	-	-	-	3.548	2.682	2.070
k = 256	-	-	-	-	3.479	2.827
k = 512	-	-	-	-	-	3.750

Table 10: Timing results for 5-pick FPE Selection and Muxing

Methods	**Number Of Ports**					
	16	**32**	**64**	**128**	**256**	**512**
k = 8	4.220	5.855	7.750	9.696	11.516	13.410
k = 16	**3.290**	4.716	6.690	8.505	10.440	12.268
k = 32	-	**4.355**	5.435	7.400	9.140	11.120
k = 64	-	-	**5.125**	6.376	8.132	9.975
k = 128	-	-	-	**5.990**	7.112	8.830
k = 256	-	-	-	-	**6.970**	**7.966**
k = 512	-	-	-	-	-	8.460

Table 11: Area Results (Normalized area-timing products) for 5-pick FPE Selection and Muxing

Methods	**Number Of Ports**					
	16	**32**	**64**	**128**	**256**	**512**
k = 8	2.094	2.244	3.101	3.179	2.891	3.127
k = 16	**1.000**	1.024	1.297	1.568	1.651	1.571
k = 32	-	**1.000**	**1.000**	**1.000**	**1.000**	1.057
k = 64	-	-	2.197	1.435	1.030	**1.000**
k = 128	-	-	-	3.203	2.091	1.401
k = 256	-	-	-	-	3.512	2.643
k = 512	-	-	-	-	-	3.453

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

In this thesis, we propose two FPE architectures (1-pick and multi-pick FPE) combined with muxing and also their variants. A multi-pick fixed priority selection selects the m topmost requests out of n inputs with static priority order and muxing combined. We have generated a family of 1-pick and m-pick fixed-priority selection and muxing circuits. Also our proposed solution has a divide-and-conquer parameter, which we call k.

We designed automated HDL code generators for all variants of both 1-pick and multi-pick FPE. Then, all multi-pick architectures were verified and synthesized for various input bit-widths, different divide sizes, and pick sizes. We benchmark all architectures on equal grounds.

According to the synthesis results provided in the previous chapter, we show that, the divide and conquer parameter, k, lets us obtain competetive timing results as well as Area-Timing product results. There is a trend that can be observed from the result tables, namely, when pick size increases, k value also increases to achieve the best performances when timing is the optimization criterion. In addition to that, when area is the optimization criterion, smaller valued k can be chosen.

Future work for this thesis may aim to generate a better performed architecture on both two criteria by modifying our FPE based selection logic by implementing the dynamic logic of Dimitrakopoulos [9] and again by combining with our muxing architecture to utilize the advantages of the dynamic priority arbiter. In addition to that, m-pick version would be developed for the new architecture that consists of Dimitrakopoulos' dynamic priority arbiter logic and our architecture. Also, the other

PPN topology can be implemented with the FPE architecture, and the results can be compared.

APPENDIX

EXAMPLE CODE GENERATOR

```
# Saturated adder based Ladner Fisher Prefix Graph Tree Generator below
#####################################################################################
# Read LF Tree's bidwidth from command line.
$n = $ARGV[0];
$Npick = $ARGV[1];
$n_m1 = $n - 1;
#####################################################################################
# Finding stage number. If it is odd, last stage have to formed with OR gates.
$stage_number = int(log($n)/log(2) +0.1);
$last_stage = $stage_number;
#Print Module Name and input and Output Ports
print "//Ladner Fisher N2N PENC for ${n} bits.
module penc (req";
for($ii=1;$ii<=$Npick;$ii=$ii+1){
    print ", outR",$ii,"";
}
print ");\n";
print "\n\tinput [${n_m1}:0] req;\n";
for($ii=1;$ii<=$Npick;$ii=$ii+1){
    print "\toutput [${n_m1}:0] outR",$ii,";\n";
}
print "\n";
#####################################################################################
# Generating Ladner Fisher Tree
# Create a double dimensional array to keep gate names with respect to stage and
    postion.
@stg_pos_gate = ();
for ($stage=1; $stage<=$stage_number; $stage++){
    #Variables
    $inc = 1<<$stage;
    $gate_count = 1<<($stage-1);
    # Select which gate I am going to use.
    if (($stage==$last_stage) && ($stage_number%2==1)){
        $gate = "OR2";
    } elsif ($stage%2 == 1){
        $gate = "NOR2";
    } else {
        $gate = "NAND2";
    }
    #Fill the hole array with null_buffer
    for ($pos=0; $pos<$n; $pos=$pos+1) {
        $temp = "null_buffer";
        $stg_pos_gate[$stage][$pos] = $temp;
    }
    # Write the gate names into the array.
    for ($pos=0; $pos<$n; $pos=$pos+$inc) {
        for ($i=0; $i<$gate_count; $i=$i+1) {
            $temp = "${gate}";
            $stg_pos_gate[$stage][$pos] = $temp;
            $pos = $pos + 1;
        }
        $pos = $pos - $gate_count;
    }
}
# Add the inverters
for ($pos=0; $pos<$n; $pos++) {
    $tmp = "null_buffer";
    for ($stage=1; $stage<=$stage_number; $stage++){
        $internal_var = $stg_pos_gate[$stage][$pos];
        if ($stg_pos_gate[$stage][$pos] eq "NOR2"){
            if ($tmp eq "NOR2") {
                $stage_m1 = $stage - 1;
                $stg_pos_gate[$stage_m1][$pos] = "INV";
            }
            $tmp = "NOR2";
        } elsif ($stg_pos_gate[$stage][$pos] eq "null_buffer") {
            if($stage==$last_stage && $tmp eq "NOR2") {
                $stg_pos_gate[$stage][$pos] = "INV";
            }
            $null_buffer_check=1;
            for($i=$stage+1; $i<=$last_stage; $i++) {
                if ($stg_pos_gate[$i][$pos] eq "null_buffer") {
                    $null_buffer_check = 1;
                } else {
                    $null_buffer_check = 0;
                    last;
                }
                if ($i==$last_stage && $null_buffer_check==1 && $tmp eq "NOR2") {
                    $stg_pos_gate[$stage][$pos] = "INV";
                    $tmp = "null_buffer";
                }
            }
        } elsif ($stg_pos_gate[$stage][$pos] eq "NAND2") {
            if ($tmp eq "NAND2" ||  $tmp eq "null_buffer") {
                $stg_pos_gate[$stage-1][$pos] = "INV";
                $tmp = "NAND2";
            }
            $tmp = "NAND2";
        } elsif ($stg_pos_gate[$stage][$pos] eq "OR2") {
            if ($tmp eq "NOR2") {
                $stg_pos_gate[$stage-1][$pos] = "INV";
            }
        }
    }
}
# Converting Gates
for ($stage=1; $stage<=$stage_number; $stage++){
    for ($pos=0; $pos<$n; $pos=$pos+1) {
        $gate = $stg_pos_gate[$stage][$pos];
        if ($gate eq "NOR2") {
            $stg_pos_gate[$stage][$pos] = "SA";
        } elsif ($gate eq "NAND2") {
            $stg_pos_gate[$stage][$pos] = "SA";
        } elsif ($gate eq "OR2") {
            $stg_pos_gate[$stage][$pos] = "SA";
        } elsif ($gate eq "INV") {
            $stg_pos_gate[$stage][$pos] = "null_buffer";
        }
    }
}
#print ("\n--> LF tree contents after placing inverters...\n");
for ($pos=0; $pos<$n; $pos++) {
    for ($stage=1; $stage<=$stage_number; $stage++){
        $internal_var = $stg_pos_gate[$stage][$pos];
    }
}
```

```
# Print wire declerations
#print ("\n--> Writing Wire Declerations...\n\n");
for ($stage=1; $stage<=$stage_number; $stage++){
    print("\t//stage${stage}\n");
    for ($pos=0; $pos<$n; $pos++) {
        $gate = $stg_pos_gate[$stage][$pos];
        print ("\twire [",$Npick-1,":0] ${gate}_s${stage}_p${pos};\n");
    }
    print("\n");
}
# Assign requests as stage[0]position[n] for future use.
for ($pos=0; $pos<$n; $pos++){
    $stg_pos_gate[0][$pos] = "req[${pos}]";
    $internal_var = $stg_pos_gate[0][$pos];
}
# Instantiations
#print ("\n--> Writing Instantiations...\n\n");
for ($stage=1; $stage<=$stage_number; $stage++){
    #Variables
    $inc = 1<<$stage;
    $gate_count = 1<<($stage-1);
    print ("\n\t//Stage${stage} Instantiations\n");
    #NOR2 or NAND2 or OR2 instantiations (Actual LF Tree)
    for ($pos=0; $pos<$n; $pos=$pos+$inc) {
        $next_pos = $gate_count + $pos;
        for ($i=0; $i<$gate_count; $i=$i+1) {
            $prev_stage = $stage - 1;
            $prev_gate_a = $stg_pos_gate[$prev_stage][$pos];
            $prev_gate_b = $stg_pos_gate[$prev_stage][$next_pos];
            $gate = $stg_pos_gate[$stage][$pos];
            if ($stage == 1) {
                print ("\t${gate} ${gate}_s${stage}_p${pos}_ins (.A({"."",$Npick-1,"'
    b0, "."${prev_gate_a}"."}), .B({"."",$Npick-1,"'b0, "."${prev_gate_b}"."}), .Y(${
    gate}_s${stage}_p${pos}));\n");
            } else {
                print ("\t${gate} ${gate}_s${stage}_p${pos}_ins (.A(${prev_gate_a}_s$
    {prev_stage}_p${pos}), .B(${prev_gate_b}_s${prev_stage}_p${next_pos}), .Y(${gate}
    _s${stage}_p${pos}));\n");
            }
            $pos = $pos + 1;
        }
        $pos = $pos - $gate_count;
    }
    #Inverter and null_buffer(null_buffer) instantitions
    for ($pos=0; $pos<$n; $pos=$pos+1) {
        $gate = $stg_pos_gate[$stage][$pos];
        $prev_stage = $stage - 1;
        $prev_gate = $stg_pos_gate[$prev_stage][$pos];
        if ($gate eq "INV") {
            if ($stage == 1){
                print ("\t${gate} ${gate}_s${stage}_p${pos}_ins (.A(${prev_gate}), .Y
    (${gate}_s${stage}_p${pos}));\n");
            } else {
                print ("\t${gate} ${gate}_s${stage}_p${pos}_ins (.A(${prev_gate}_s${
    prev_stage}_p${pos}), .Y(${gate}_s${stage}_p${pos}));\n");
            }
        } elsif ($gate eq "null_buffer") {
            if ($stage == 1){
                print ("\tassign ${gate}_s${stage}_p${pos} = ${prev_gate};\n");
            } else {
                print ("\tassign ${gate}_s${stage}_p${pos} = ${prev_gate}_s${
    prev_stage}_p${pos};\n");
            }
        }
    }
    print ("\n");
}
# Assign the last stage gates outputs' to lf_penc's output
for($ii=1;$ii<=$Npick;$ii=$ii+1){
    for ($pos=0; $pos<$n; $pos++){
        $last_stage_gate = $stg_pos_gate[$last_stage][$pos];
        print ("\tassign outR",$ii,"[${pos}] = ${last_stage_gate}_s${last_stage}_p${
    pos}[",$ii-1,"];\n");
    }
    print ("\n");
}
# Write endmodule a the end of the lf_penc.v file
print ("\nendmodule\n");
```

REFERENCES

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

[2] B. Yuce, H. F. Ugurdag, S. Gören, and G. Dündar, “Fast and efficient circuit topologies forfinding the maximum of n k-bit numbers,” *IEEE Transactions on Computers*, vol. 63, pp. 1868–1881, 2014.

[3] H. F. Ugurdag, F. Temizkan, and S. Gören, “Generating fast logic circuits for m-select n-port round robin arbitration,” in *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 260–265, 2013.

[4] J. Ahn, D. K. Jeong, and S. Kim, “Fast three-dimensional programmable two-selector,” *Electronics Letters*, vol. 40, p. 1, 2004.

[5] M. D. Kalpande and V. Vyas, “Logic architecture for 2-select arbiter using duel pointer,” in *TENCON IEEE Region 10 Conference*, pp. 1–5, 2015.

[6] P. Gupta and N. McKeown, “Designing and implementing a fast crossbar scheduler,” *IEEE Micro*, vol. 19, pp. 20–28, 1999.

[7] F. Temizkan, “Multi-pick round robin arbiter,” Master’s thesis, Ozyegin University, 2012.

[8] G. Dimitrakopoulos, N. Chrysos, and K. Galanopoulos, “Fast arbiters for on-chip network switches,” in *2008 IEEE International Conference on Computer Design (ICCD)*, pp. 664–670, 2008.

[9] G. Dimitrakopoulos and E. Kalligeros, “Dynamic-priority arbiter and multiplexer soft macros for on-chip networks switches,” in *Conference on Design, Automation and Test in Europe (DATE)*, pp. 542–545, 2012.

[10] J.-C. Chiu and K.-M. Yang, “High-speed low-power multiplexer-based selector for priority policy,” *Computers and Electrical Engineering*, vol. 39, pp. 202 – 213, 2013.

[11] R. P. Brent and H. T. Kung, “A regular layout for parallel adders,” *IEEE Transactions on Computers*, vol. 31, pp. 260–264, 1982.

[12] D. Harris, “A taxonomy of parallel prefix networks,” in *Asilomar Conference on Signals, Systems, Computers*, pp. 2213–2217, 2003.

[13] R. E. Ladner and M. J. Fischer, “Parallel prefix computation,” *Journal of the ACM*, vol. 27, pp. 831–838, 1980.

[14] T. Han and D. A. Carlson, “Fast area-efficient vlsi adders,” in *IEEE Symposium on Computer Arithmetic (ARITH)*, pp. 49–56, 1987.

[15] M. Tosun, A. Ozkan, A. Guzel, and H. F. Ugurdag, “Fast one- and two-pick fixed-priority selection and muxing circuits,” in *EWDTS Conference*, 2016.

[16] J. Stephenson and P. Metzgen, “Logic optimization techniques for multiplexers,” *Altera Literature*, 2004.

[17] 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, 2012.

VITA

Name Surname: Mustafa Tosun

Address:

C Tech Bilişim Teknolojileri San. ve Tic. A.Ş.

Teknopark İstanbul, Teknoloji Geliştirme Bölgesi,

Sanayi Mah. Teknopark Bulvar No: 1, A Blok, Kat: 2,

Kurtköy-Pendik 34912 İSTANBUL, TURKEY

Birth Place / Year: Burdur / 1990

Languages: Turkish (native) - English

BS: Bahçeşehir University - 2013

High School: Isparta Suleyman Demirel Science High School - 2008

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

Publications:

- **M. Tosun**, M. A. Özkan, A. E. Güzel, H. F. Ugurdag, "Fast One- and Two-Pick Fixed-Priority Selection and Muxing Circuits," EastWest Design & Test Symposium (EWDTS), 2016.

- A. E. Güzel, V. E. Levent, **M. Tosun**, M. A. Özkan, T. Akgun, D. Büyükaydin, C. Erbas, H. F. Ugurdag, "Using High-Level Synthesis for Rapid Design of Video Processing Pipes," EastWest Design & Test Symposium (EWDTS), 2016.

- M. Buyukmihci, V. E. Levent, A. E. Guzel, O. Ates, **M. Tosun**, T. Akgun, C. Erbas, S. Goren, H. F. Ugurdag, "Output Domain Downscaler," Computer and Information Sciences (ISCIS), pp. 262-269, 2016.

Work Experience:

- C Tech Bilişim Teknolojileri San. ve Tic. A.Ş.

  Hardware Engineer Jan. 2016 - Ongoing

- Özyeğin University EE Engineering Department

  Teaching & Research Asst. Sept. 2013 - Jan. 2016

Honors and Awards:

- MEF Research Projects Competition Finalist (Physics Projects) * May 2008
- Full Tuition Waiver Awarded by OSYM 2008-2013
- High Honor Scholarship Awarded by Bahçeşehir University * 2011 - 2013
- 2241-B-TÜBİTAK Industry-Based Graduation Project Competition Finalist * 2013
- 2242-TÜBİTAK Undergraduate Students Software Projects Competition Finalist * 2013
- 2nd SDU Science Festival Robot Contest Award: 3rd Place * 2013
- 8th PROJISTOR Technical Project Contest Award: 2nd place * 2013
- ITURO 2013 Free Style Robot Contest Award: 4th place * 2013
- 10th International ODTU Robot Days Free Style Contest Award: 3rd place * 2013
- Ranked 1st in Mechatronics Engineering Faculty, Bahçeşehir University, Istanbul * Jun 2013
- Full tuition waiver + TAship support by Özyeğin University * 2013-2016
- Supported by TÜBİTAK 1001 RAship * 2014-2016