



**NEAR EAST UNIVERSITY**

**INSTITUTE OF GRADUATE STUDIES**

**DEPARTMENT OF COMPUTER ENGINEERING**

**………………………………………………………………**

**HIGH-PERFORMANCE TOPOLOGY, SCALABLE NETWORK ON-CHIP FOR MANY-CORE SYSTEM**

**PhD Thesis**

**ALLAM R. M. ABUMWAIS**

**Nicosia**

**March, 2022**

**NEAR EAST UNIVERSITY**

**INSTITUTE OF GRADUATE STUDIES**

**DEPARTMENT OF COMPUTER ENGINEERING**

**………………………………………………………………**

**HIGH-PERFORMANCE TOPOLOGY, SCALABLE NETWORK ON-CHIP FOR MANY-CORE SYSTEM**

**PhD THESIS**

**ALLAM R. M. ABUMWAIS**

**Supervisor**

**Prof. Dr. ADİL AMİRJANOV**

**Assist. Prof. Dr. Kaan Uyar**

**Nicosia**

**March, 2022**

# Approval

We certify that we have read the thesis submitted by Allam R. M. Abumwais titled **“High Performance Topology, Scalable Network On-Chip For Many-Core System**” and that in our combined opinion it is fully adequate, in scope and in quality, as a thesis for the degree of PhD in Computer Engineering.

Examining Committee Name-Surname Signature

Head of the Committee: Assoc. Prof. Eser GEMİKONAKLI ……………………… University of Kyrenia

Committee Member1: Assist. Prof. Dr. Umar ÖZGÜNALP ……………………..

Cyprus International University

Committee Member2: Assist. Prof. Dr. Ümit İLHAN ………………………..

Near East University

Committee Member3: Assist. Prof. Dr. Ali SERENER ………………………..

Near East University

Supervisor: Prof. Dr. Adil Amirjanov ………………………..

Assist.Prof.Dr.Kaan Uyar ………………………..

Near East University

Approved by the Head of the Department

10/03/2022

………………………..

Title, Name-Surname

Head of Department

Approved by the Institute of Graduate Studies

10/03/2022

………………………..

Prof. Dr. Kemal Hüsnü Can Başer

Head of the Institute

# Declaration

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Allam R. M. Abumwais

******

10/03/2022

# Acknowledgments

I thank Allah for everything.

“Say: "Surely, my prayer and my service of sacrifice, my life and my death, are (all) for Allah, the Lord of the Worlds"

Foremost, this thesis was supervised by Prof. Dr. Adil Amirjanov and Assist. Prof. Dr. Kaan Uyar. Special thanks go out to them. Their high levels of commitment, their comments, and the opportunity they provided for interactive discussion were of great value to this thesis. Furthermore, their advice was invaluable to me throughout my research and thesis writing.

I would like to offer my heartfelt gratitude to my family, particularly my parents and wife, for the unending care, wisdom, and support they've shown me. I would like to thank my friend Dr. Mujahed Eleyat for his encouragement and assistance whilst conducting the FPGA implementation.

I would like to extend my sincere thanks to all people who provided support for this thesis.

Thanks are also due to all friends who supported me both financially and emotionally during the difficult times, I faced during this thesis.

**Allam R. M. Abumwais**

**To my parents...**

# Abstract

**High-performance Topology, Scalable Network On-chip For Many-core System**

**Abumwais, Allam**

**PhD, Department of Computer Engineering**

**March 2022, 199 pages**

The advent of the multi/many-core systems has represented a great hope for massively parallel computation. In shared memory multi/many-core systems, a significant improvement is still needed in some issues. The shared cache memory architecture and the interconnection network are the two major issues of prime concern in this thesis.

In shared cache memory, there is a problem when more than one core try to simultaneously access the same shared module. Developments to cache architecture and cache replacement's algorithms are two options that have been followed in this direction. This thesis presents a new design of a dedicated pipeline cache memory for multicore processors called dual-port content addressable memory (DPCAM). In addition, it proposes a new hardware-based replacement algorithm called near-far access replacement algorithm (NFRA) in order to reduce the cost overhead of the cache controller and improve the cache access latency. The experimental results indicated that the latency for write and read operations is significantly less than that of a set-associative cache memory. Moreover, it was shown that the latency of a write operation is nearly constant regardless of the size of DPCAM. However, an estimation of the power dissipation showed that DPCAM consumes about 7% greater than a set-associative cache memory of the same size.

The interconnection network connects the cores of a multicore system to a shared memory through which they exchange information where the core is responsible to make a request when it wishes to access the shared memory. The arbitration logic, which is a router in the modern network-on-chip (NoC), and the contention of shared resources, and hence the latency, grow as more cores are added to the system. An interconnection scheme is introduced in this thesis that is based on a router-less interconnection scheme that connects a number of multi-core systems, each called a cluster, to construct a many-core system. Within a cluster, a multi-port content addressable memory (MPCAM) organization is used for efficient communication. MPCAM allows all cores to access shared memory simultaneously without the need for queuing requests and using a router. ​In inter-clusters communication, N clusters, each has N cores, can be connected using a scalable N-conjugate shuffle cluster network (NCSC) without the need for redesigning or reprogramming the system which also eliminates the need for router devices and their complexity. A comparison study is then made between the proposed scheme and other network topologies that are normally used in the modern multi/many-core systems. It revealed a remarkable improvement in terms of static performance metrics. MPCAM and NCSC have been implemented using the Intel Field Programmable Gate Array (FPGA) Cyclone device family. The latency of read and write by multiple cores within the cluster and between clusters have been assessed. Moreover, all cores can access the shared data in the MPCAM of another cluster with an average access time of 1.92738±0.139588 ns. These access times are fixed regardless of the value of N. NCSC supports non-blocking write access between cores in the same cluster with an average latency of 1.14785±0.04532 ns which is nearly equal to the latency of write operation in MPCAM while the average read latency within the same cluster is 1.26226±0.090591 ns.

***Key Words***: many-core; shared memory; interconnection network; content addressable memory; power dissipation.

# ÖZET

**Yüksek performanslı Topoloji, Ölçeklenebilir Ağ çip üzerinde Çok çekirdekli Sistem için**

**Abumwais, Allam**

**PhD, Bilgisayar Mühendisliği Bölümü**

**Mart 2022, 199 sayfa**

Çoklu veya çok çekirdekli sistemlerin ortaya çıkışı güçlü paralel hesaplama için büyük bir umut doğrumuştur. Paylaşımlı bellekli çoklu veya çok çekirdekli sistemlerde, bazı konularda hala ciddi iyileştirmelere ihtiyaç duyulmaktadır. Bu tezin başlıca iki konusu paylaşımlı önbellek mimarisi ve ara bağlantı ağıdır.

Paylaşımlı önbellekte kullanıldığında, birden fazla çekirdek paylaşılan bir modüle aynı anda erişmeye çalıştığında sorun oluşur. Önbellek mimarisindeki gelişmeler ve önbellek değiştirme algoritmaları bu yönde izlenen iki seçenektir. Bu tez, çoklu çekirdekli işlemcilere yönelik özel veri hattı önbellekleri için yeni bir tasarım sunar. Bu tip belleklere çifte-girişli içeriği adreslenebilir bellek (DPCAM) denir. Ek olarak, bu tez, önbellek denetleyicisinin ek işletim yükünü azaltmak ve önbellek erişim gecikmesini iyileştirmek için yakın-uzak erişim değiştirme algoritması (NFRA) adı verilen yeni bir donanım tabanlı değiştirme algoritması önerir. Deneysel sonuçlar, yazma ve okuma işlemlerinin gecikmesinin, küme ilişkili önbellek belleğinin performansından önemli ölçüde daha az olduğunu gösterdi. Ayrıca, okuma işleminin gecikmesinin, DPCAM büyüklüğünden bağımsız olarak neredeyse sabit olduğu gösterilmiştir. Bununla birlikte, güç tüketimi tahmini göstermiştik ki DPCAM'nin aynı büyüklükteki bir set-ilişkisel önbelleğe kıyasla yaklaşık %7 daha fazla güç tüketmektedir.

Ara bağlantı ağı, çok çekirdekli bir sistemin çekirdeklerini, paylaşılan belleğe erişmek istediğinde çekirdeğin bir talepte bulunmaktan sorumlu olduğu bilgi alışverişinde bulundukları paylaşılan bir belleğe bağlar. Çekişme çözümleme mantığı, ki modern çip-üzerindeki-ağda (NoC) bir yönlendiricidir, paylaşılan kaynakların arasındaki çekişme, ve dolayısı ile gecikme sisteme daha fazla çekirdek eklendikçe artar. Bu tezde, çok çekirdekli bir sistem oluşturmak için her biri bir küme olarak adlandırılan bir dizi çok çekirdekli sistemi birbirine bağlayan yönlendiricisiz bir ara bağlantı yaklaşımı tanıtılmaktadır. Bir küme içinde, verimli iletişim için çok bağlantı noktalı içerik adreslenebilir bellek (MPCAM) organizasyonu kullanılır. MPCAM, istekleri sıraya koymaya ve bir yönlendirici kullanmaya gerek kalmadan tüm çekirdeklerin paylaşılan belleğe aynı anda erişmesine olanak tanır. Kümeler arası iletişimde, her biri N sayıda çekirdeğe sahip N küme, sistemi yeniden tasarlamaya veya yeniden programlamaya gerek kalmadan ölçeklenebilir bir N-konjugat karma küme ağı (NCSC) kullanılarak bağlanabilir. Bu yaklaşım da yönlendirici cihazlara olan ihtiyacı ve bunların karmaşıklığını ortadan kaldırır. Buna bağlı olarak, önerilen yaklaşım ile modern çoklu veya çok çekirdekli sistemlerde normal olarak kullanılan diğer ağ topolojileri arasında bir karşılaştırma çalışması yapılır. Bu karşılaştırma, statik performans ölçütleri açısından dikkate değer bir gelişmenin sağlandığını ortaya koydu. MPCAM ve NCSC, Intel Alanda Programlanabilir Kapı Dizisi (FPGA) Cyclone cihaz ailesi kullanılarak uygulanmıştır. Küme içindeki ve kümeler arasındaki birden çok çekirdek tarafından okuma ve yazma gecikmesi değerlendirilmiştir. Ayrıca, tüm çekirdekler, ortalama 1.92738±0.139588 ns sürede başka bir kümenin MPCAM'ındaki paylaşılan verilere erişebilir. Bu erişim süreleri, N değerinden bağımsız olarak sabittir. NCSC, aynı kümedeki çekirdekler arasında tıkanmasız yazma erişimini destekler ve ortalama 1,14785±0,04532 ns gecikme süresi, MPCAM'daki yazma işleminin gecikme süresine neredeyse eşittir. Aynı küme içindeki ortalama okuma gecikmesi ise 1.26226±0.090591 ns'dir.

***Anahtar Kelimeler*:** çok çekirdekli; paylaşılan bellek; ara bağlantı ağı; içerik adreslenebilir bellek; güç dağılımı.

# Table of Contents

[**Approval 2**](#_30j0zll)

[**Declaration 3**](#_1fob9te)

[**Acknowledgments 4**](#_2et92p0)

[**Abstract 6**](#_3dy6vkm)

[**ÖZET 8**](#_4d34og8)

[**Table of Contents 10**](#_17dp8vu)

[**List of Figures 13**](#_3rdcrjn)

[**List of Tables 16**](#_26in1rg)

[**List of Abbreviations 16**](#_lnxbz9)

[**CHAPTER I 20**](#_35nkun2)

[**Introduction 20**](#_1ksv4uv)

[1.1 Parallel Processing 20](#_44sinio)

[1.2 Shifted to Multi/many-core Systems 21](#_2jxsxqh)

[1.3 Thesis Problem and its Motivation 22](#_z337ya)

[1.4 Aims and Objectives 24](#_3j2qqm3)

[1.5 Contribution “Significance of the Study” 25](#_1y810tw)

[1.6 Structure of the Thesis 26](#_4i7ojhp)

[**CHAPTER II 28**](#_2xcytpi)

[**Background and Related Work 28**](#_1ci93xb)

[2.1 Parallel Architecture 28](#_3whwml4)

[*2.1.1 Pipeline Processor Architecture* 28](#_2bn6wsx)

[*2.1.2 Array Processor Architecture* 29](#_qsh70q)

[*2.1.3 Multiprocessor system* 30](#_3as4poj)

[2.2 Multicore System Architecture 32](#_1pxezwc)

[*2.2.1 SSM Architecture* 33](#_49x2ik5)

[*2.2.2 DSM Architecture* 34](#_2p2csry)

[2.3 Many-core System Architecture 35](#_147n2zr)

[2.4 Major Problems in Multi/many-core System Architecture 38](#_3o7alnk)

[*2.4.1 Communication Problem* 38](#_23ckvvd)

[*2.4.2 Coherence Problem* 39](#_ihv636)

[*2.4.3 Scheduling Problems* 40](#_32hioqz)

[*2.4.4 Power and Area Problems* 41](#_1hmsyys)

[2.5 SoC Interconnection Networks 43](#_41mghml)

[*2.5.1 The Buses Interconnection era* 44](#_2grqrue)

[*2.5.2 The Crossbars Networks era.* 45](#_vx1227)

[*2.5.3 The NoCs based router era.* 47](#_3fwokq0)

[*2.5.4 The Routerless NoCs era.* 49](#_1v1yuxt)

[2.6 Related Work 50](#_4f1mdlm)

[*2.6.1 Related Shared cache architecture* 50](#_2u6wntf)

[*2.6.2 Related Routerless NoC* 53](#_19c6y18)

[**CHAPTER III 57**](#_3tbugp1)

[**The DPCAM Architecture 57**](#_28h4qwu)

[3.1 Introduction 57](#_nmf14n)

[3.2 Types of Shared Cache Memory 58](#_37m2jsg)

[*3.2.1 Fully Associative Memory* 58](#_1mrcu09)

[*3.2.2 Direct Mapped Memory* 59](#_46r0co2)

[*3.2.3 Set-associative Memory* 60](#_2lwamvv)

[*3.2.4 Content-addressable Memory (CAM)* 61](#_111kx3o)

[3.3 Proposed DPCAM 63](#_3l18frh)

[3.4 Implementation of the DPCAM and Performance Analysis 67](#_206ipza)

[*3.4.1 Functional Simulation* 67](#_4k668n3)

[*3.4.2 Latency Assessments* 70](#_2zbgiuw)

[*3.4.3 Estimation of a Power Dissipation* 75](#_1egqt2p)

[**CHAPTER IV 79**](#_3ygebqi)

[**Shared Cache Based on DPCAM in a Multicore Architecture 79**](#_2dlolyb)

[4.1. Introduction 79](#_sqyw64)

[4.2 DPCAM Inside Multicore System Architecture 80](#_3cqmetx)

[*4.2.1 DPCAM Architecture* 80](#_1rvwp1q)

[*4.2.2 Multicore Using DPCAM Architecture* 81](#_4bvk7pj)

[4.3 A Modified Replacement Policy for DPCAM 83](#_2r0uhxc)

[4.4 Implementation of the DPCAM within the Multicore System 85](#_1664s55)

[*4.4.1 Experimental Setup* 85](#_3q5sasy)

[*4.4.2 Benchmarking Program* 87](#_25b2l0r)

[4.5. Simulation Results 89](#_kgcv8k)

[**CHAPTER V 92**](#_34g0dwd)

[**A New Design of Multicore IN Based on DPCAM 92**](#_1jlao46)

[5.1 Introduction 92](#_43ky6rz)

[5.2 Embedded the shared cache within INs 94](#_2iq8gzs)

[5.3 Proposed INfor Multicore System 95](#_xvir7l)

[*5.3.1 Multi-Port CAM (MPCAM) Organization* 95](#_3hv69ve)

[*5.3.2 MPCAM Organization Analysis* 96](#_1x0gk37)

[*5.3.3 The MPCAM-Based Multi-core System* 97](#_4h042r0)

[5.4 Performance Analysis of MPCAM Organization 98](#_2w5ecyt)

[*5.4.1 Functional Simulation* 99](#_1baon6m)

[*5.4.2 Latency Assessments* 100](#_3vac5uf)

[5.5 The MPCAM Status 105](#_2afmg28)

[**CHAPTER VI 107**](#_pkwqa1)

[**A Scalable Interconnection Scheme In Many-core Systems 107**](#_39kk8xu)

[6.1 Introduction 107](#_1opuj5n)

[6.2 The Component of the Proposed Scheme 108](#_48pi1tg)

[*6.2.1 The N-Conjugate Shuffle* 109](#_2nusc19)

[*6.2.2 The MPCAM Organization* 113](#_1302m92)

[6.3 The NCSC Interconnection Scheme 114](#_3mzq4wv)

[6.4 Performance Analysis of NCSC 118](#_2250f4o)

[*6.4.1 Static Performance Analysis* 119](#_haapch)

[*6.4.2 Dynamic Performance Analysis* 126](#_319y80a)

[**CHAPTER VII 136**](#_1gf8i83)

[**Conclusion 136**](#_40ew0vw)

[7.1 Discussion Results 137](#_2fk6b3p)

[7.2 Potential Applications 138](#_upglbi)

[7.3 Future Work 139](#_3ep43zb)

[**References 140**](#_1tuee74)

[**Appendices 148**](#_4du1wux)

[**Appendix A 148**](#_2szc72q)

[**Cache Coherence Approach 148**](#_184mhaj)

[**Appendix B 154**](#_3s49zyc)

[**DPCAM Implementation 154**](#_279ka65)

[**Appendix C 166**](#_meukdy)

[**DPCAM With GEM5 Implementation 166**](#_36ei31r)

[**Appendix D 170**](#_1ljsd9k)

[**MPCAM Implementation 170**](#_45jfvxd)

[**Appendix E 182**](#_2koq656)

[**A Scalable Interconnection Implementation 182**](#_zu0gcz)

[**Appendix X 197**](#_3jtnz0s)

[**Turnitin Similarity Report 197**](#_1yyy98l)

[**Curriculum Vitae 198**](#_4iylrwe)

# List of Figures

**Figure 1.1:** *Parallel Processor Architecture Classifications..........................................* 24

**Figure 2.1:** *Array processor architecture with local memory*......................................... 31

**Figure 2.2:** *An Array Processor Architecture With Global Memory*..............................31

**Figure 2.3-A:** *SSM Architecture.....................................................................................* 33

**Figure 2.3-B:** *Core Unit*.................................................................................................. 34

**Figure 2.4:** *DSM Architecture*......................................................................................... 35

**Figure 2.5:** *Many-core System Based Multi-cluster Architecture*................................... 37

**Figure 2.6:** *Many-core System Based On Grid Network Architecture*............................ 38

**Figure 2.7:** *Shared Bus Network*..................................................................................... 46

**Figure 2.8:** *Multiple Bus Network*................................................................................... 46

**Figure 2.9:** *Crossbar Switch Network*............................................................................. 46

**Figure 2.10:** *MINs 2X2 switch element*........................................................................... 47

**Figure 2.11:** *2-D Mesh Interconnection*…………...............................…....................... 49

**Figure 2.12:** *Ring Interconnection*………………………………..……….................... 49

**Figure 3.1:** *Fully Associative Architecture*……………................................................. 59

**Figure 3.2:** *Direct Mapped Cache Memory*.................................................................... 60

**Figure 3.3:** *Set Associative Cache Memory*.................................................................... 61

**Figure 3.4:** *CAM Architecture*…………………………................................................ 63

**Figure 3.5:** *DPCAM Design*........................................................................................... 64

**Figure 3.6:** *Control Unit (CU)*....................................................................................... 65

**Figure 3.7:** *Near-access, Far-access DPCAM modules*................................................ 67

**Figure 3.8:** *Test-bench Program*.................................................................................... 69

**Figure 3.9:** *Functional Simulation*................................................................................. 69

**Figure 3.10:** *Latency Assessments*.................................................................................. 71 **Figure 3.11:** *Latency Of A Write Operation (ns)*............................................................ 73 **Figure 3.12:** *Power Dissipation With Variation In Memory Size*…………….............. 78

**Figure 3.13:** *Total Power Dissipation With Variation In Size*........................................ 78

**Figure 4.1:** *DPCAM Architecture Within A Multicore System*....................................... 81 **Figure 4.2:** *Multicore Architecture Using DPCAM Shared Cache*................................ 82

**Figure 4.3:** *A Modified Replacement Policy For DPCAM*…………………………..... 84

**Figure 4.4:** *Ruby Memory System Simulator*........................................………………... 86

**Figure 4.5:** *System Performance In (IPC) Comparison*……………………….…......... 90

**Figure 4.6:** *System Miss Rate Comparison*…...................................…………….......... 91

**Figure 5.1:** *The Organization Of The MPCAM*............................………….……......... 95

**Figure 5.2:** *The Proposed Multicore System With MPCAM*.......………..……….......... 98

**Figure 5.3:** *MPCAM Organization Functional Simulation*..................…………......... 101

**Figure 5.4:** *MPCAM Organization Timing Simulation 1*............………….……......... 102

**Figure 5.5:** *MPCAM Organization Timing Simulation 2*............………….…............. 103

**Figure 6.1:** *A 3-shuffle Of 12 Objects*...........................................………………......... 111

**Figure 6.2-A:** *The 4-conjugate Shuffle*....………………………………….…............. 111

**Figure 6.2-B:** *The Equivalent Bidirectional Connection 4-conjugate Shuffle*.............. 112

**Figure 6.3:** *MPCAM Organization Inside Each Cluster*........…………...….…........... 114

**Figure 6.4:** *Connecting Two Cores Of Two Different Clusters Via The Conjugate Shuffle*……………………………………………………………………….……........ 115

**Figure 6.5:** *NCSC Cores Connection*............................……..…………...................... 118

**Figure 6.6:** *Diameter Of The Networks For Different Sizes*………..…………........... 123

**Figure 6.7:** *Degree Of The Networks For Different Sizes*...............……...…….......... 124

**Figure 6.8:** *Connectivity Of The Networks For Different Sizes*...………...….…......... 124

**Figure 6.9-A:** *Cost Of The Networks For Different Sizes*............………………......... 125

**Figure 6.9-B:** *Cost Of The Networks For Different Sizes*............………...…….......... 125

**Figure 6.10:** *Bisection-width Of The Networks For Different Sizes*....…….…............ 125

**Figure 6.11:** Test-benchmark Functions……..…………...…………………….......... 126

**Figure 6.12:** *Function Simulation For Read Operation Scenario*...………….…........ 129

**Figure 6.13:** *Function Simulation For Read Operation Scenario 2*...…….……......... 129

**Figure 6.14:** *Function Simulation For Write Operation*....…………….……….......... 130

**Figure 6.15:** *Timing Simulation For Read Operation Scenario 1*…….…….…........... 132

**Figure 6.16:** *Timing Simulation For Read Operation Scenario 2*...………….…......... 133

**Figure 6.17:** *Timing Simulation For Write Operation*...…….………………….......... 133

**Figure A-1:** Write-Once transaction diagram................................................................. 150

**Figure B-1:** DPCAM Location Architecture................................................................. 161

**Figure B-2:** Control Unit Architecture …….................................................................. 162

**Figure D-1:** MPCAM Organization ……….................................................................. 177

**Figure E-1:** NCSC Architecture……………................................................................ 195

# List of Tables

**Table 3.1:** Latency Of A Write Operation (ns)……………........................................... 72

**Table 3.2:** Descriptive Statistics For Write Latency In DPCAM................................... 74

**Table 3.3:** Descriptive Statistics For Read Latency In DPCAM.................................... 74

**Table 3.4:** Descriptive Statistics For The Simultaneous Read And Write Operations Into The Same Memory Location Latency In DPCAM.......................................................... 74

**Table 3.5:** Descriptive Statistics For The Simultaneous Read And Write Operations Into The Same Memory Location Latency In DPCAM.......................................................... 75

**Table 3.6:** Descriptive Statistics For Write Latency In Set-associative.......................... 75

**Table 3.7:** Descriptive Statistics For Read Latency In Set-associative........................... 75

**Table 3.8:** DPCAM Power Dissipation…………………………………....................... 77

**Table 3.9:** SA Memory Power Dissipation…………………..…………....................... 77

**Table 4.1:** The Configurations Of The Proposed Multicore Architecture...................... 86

**Table 5.1:** Descriptive Statistics For Write/read Latencies........................................... 104

**Table 5.2:** T-test For Write Comparisons...................................................................... 104

**Table 5.3:** T-test For Read Comparisons....................................................................... 104

**Table 6.1:** Properties Of Some INs................................................................................ 123

**Table 6.2:** Descriptive Statistics For Read Latencies.................................................... 134

**Table 6.3:** T-test For Read Comparisons....................................................................... 134

**Table 6.4:** Levens’s Test................................................................................................ 135

**Table B.1:** Descriptive Statistics For Write Latency Comparison................................163

**Table B.2:** Levens’s Test Independent Samples Test..................................................... 163

**Table B.3:** T-test For Write Comparisons..................................................................... 163

**Table B.4:** T-test For Read Comparisons...................................................................... 164

**Table B.5:** Group Statistic For Read latencies............................................................... 164

**Table B.6:** Levens’s Test For Read................................................................................. 164

**Table B.7:** T-test For Write Comparision........................................................................ 165

**Table C.1A:** Benchmark Programs Reference count..................................................... 166

**Table C.1B:** Benchmark Programs Reference count..................................................... 166

**Table C.2A:** Benchmark Programs Miss count............................................................. 166

**Table C.2B:** Benchmark Programs Miss count............................................................. 167

**Table C.2A:** Benchmark Programs Miss Rate.............................................................. 167

**Table C.2A:** Benchmark Programs Miss Rate.............................................................. 167

**Table D.1:** Descriptive Statistics For Write Latency Comparison................................ 179

**Table D.2:** Levens’s Test Independent Samples Test …...…….................................... 179

**Table D.3:** T-test For Write Comparisons ………………………................................ 179

**Table D.4:** Descriptive Statistics For Read Latency Comparison................................. 180

**Table D.5:** Levens’s Test Independent Samples Test …...…….................................... 180

**Table D.6:** T-test For Read Comparisons ………………………................................. 180

# List of Abbreviations

**SoC:** System on Chip

**NoC:** Networks-on-chip

**PE:** Processor Element

**PEs:** Processor Elements

**IN:** Interconnection Network

**INs:**  Interconnection Networks

**BW:** Bandwidth

**SISD:** Single Instruction Stream Single Data Stream

**MISD:** Multiple Instruction Stream Single Data Stream

**SIMD:** Single Instruction Stream Multiple Data Stream

**MIMD:** Multiple Instruction Stream Multiple Data Stream computers

**CAM:** Content Addressable Memory

**DPCAM:** Dual Port Content Addressable Memory

**CAP:** Community Acquired Pneumonia

**NCSC:** N-conjugate Shuffle Cluster Network

**NFRA:** Near-Far access Replacement Algorithm

**FPGA:** Field Programmable Gate Array

**ILP:**  Instruction level parallelism

**GPU:** Graphics Processing Unit

**CPU:** Central Processing Unit

**CU:** Control Unit

**CP:**  Control Unit Processor

**CM:** Control Unit Memory

**SSM:** Symmetric Shared Memory

**DSM:** Distributed Shared Memory

**SMM:** Shared Memory Module

**PM:** Private Memory

**MIT:** Massachusetts Institute of Technology

**L1 cache:** Level 1 cache

**L2 cache:** Level 2 cache

**ARM:** Advanced RISC Machines

**AMBA:** Advanced Microcontroller Bus Architecture

**IBM:** International Business Machines Corporation

**MINs:** Multistage Interconnection Networks

**2D-mesh:** Two Dimensional mesh

**AM:** Associative Memory

**ReCAM:** Resistive Conﬁgurable Associative Memory

**LRU:** Least Recently Used

**LER:** Least Error Rate

**BiCAM:** Binary Content Addressable Memory

**LRU:** Graphical Processing Unit

**RFFE:** Random First Flash Enlargement

**CMIT:** Cluster Mesh Inter-layer Topology

**NRO:** Nesting Ring NoC

**IMR:** Isolated Multi-Ring

**IC:** Integrated Circuit

**SRAM:** Static Random Access Memory

**DRAM:** Dynamic Random Access Memory

**COM:** Content Operated Memory

**CMOS:** Complementary Metal Oxide Semiconductor

**DM:** Direct Mapped

**SA:**  Set Associative

**FA:** Fully Associative

**FIFO:** First In First Out

**TM:** Transaction Memory

**SB:** Store Back

**OF:** Operand Fetch

**IF:** Instruction Fetch

**ID:** Instruction Decoder

**Ex:** Execute

**L:** Cache Line

**CMP:** Comparator

**D-FF:** D Flip Flop

**LE:**  Latch Enable

**RD:** Read Signal

**OE:** Output Enable

**KiB:** Kilobyte

**VHDL:** Verilog Hardware Description Language

**VWF:** Vector Wav File

**ns:** nanosecond

**LWR:** Latency for Write

**LRD:** Latency for Read

**Tcl:** Cycle Time

**LSDL:** Latency for Simultaneous write/read to the Different Locations

**LSSL:** Latency for Simultaneous write/read to the Same Locations

**I/O:** Input/Output

**mW:** Milliwatts

**MESI:** Modified-Exclusion-Shared-Invalid

**MMU:** Memory Management Unit

**L[k]:** Current Location

**SLICC:** Specification Language Including Cache Coherence

**NUCA:** Non-Uniform Cache Access

**SPEC:** Standard Performance Evaluation Corporation

**ISA:** Instruction Set Architecture

**IPC:** Instruction Per Second

**COTS:** Component Off the Shelf

**PCB:** Printed Circuit Board

**GSN:** Generalized Shuffle Network

**MPCAM i:** Multi-port CAM organization in cluster i

**MPCAM j:** Multi-port CAM organization in cluster j

**SW:** Switch

**SE:** Switch Element

**OFij:** Operand Fetch core i in cluster j

**SBij:** Store Back core i cluster j

# CHAPTER I

# Introduction

Multi/many-core systems architecture are implemented as System-On-Chip (SoC). Each core is a simple pipeline processor, cores are connected each other using interconnection network (IN). A Networks-on-chip (NoC) based router architecture is extensively used in modern systems. According to current scalability trends, Many-core systems could have hundreds and thousands of cores in a single chip in the next few years. However, expanding multi/many-core architecture is not without problems and challenges including contention over IN, memory consistency, power consumption, scheduling policy, memory bottlenecks and scalability issues. The introduction describes the limitations of current system architecture, the context work of this thesis and presents its motivation and contributions.

## 1.1 Parallel Processing

For many years now, the idea of using multiprocessors to increase computer processing power has been stirred researchers' interest. Several architectures were proposed and some of them were chosen by the manufactures to be produced commercially. Ideally, in such systems, the computing power would be directly proportional to the number of processor elements (PEs) used in the system. Because the number of processors that can be used in the system is theoretically unlimited, multiprocessor systems are considered as a hop for limitless computing power. However, the problems associated with splitting the program up into parts and running these parts in parallel on a number of processors have proven to be considerable. This issue is called the scheduling policy ([Chen et al., 2011](#_3oy7u29); [Patterson & Hennessy, 2020](#_243i4a2)). Most problems cannot be easily split into a number of parts or sub-parts equal to the number of processors and totally independent of each other. More often, values calculated by one part of the program are required by other parts. The problem of transferring data between program parts that are executed by different processors has proven more difficult than might be expected. This issue is called the communication policy ([Abumwais & Ayyad, 2018](#_j8sehv); [Bohnenstiehl et al., 2016](#_338fx5o)). Furthermore, because the shared data is accessible with all processors that require them. So that each processor access the proper version of shared data, access by number of processors should be synchronized ([Martin et al., 2012](#_1idq7dh); [Patterson & Hennessy, 2020](#_243i4a2)). There will be a requirement for hardware and software synchronization. This issue is called the cache coherence policy.

The elements of data which need to be transferred between parts that are executed by different processors are considered shared data. In almost all existing architecture, the shared data is held in a common shared memory (The words "shared memory" and "shared cache" are interchangeable hereafter) which is accessible by all processors of the system. When the processor element (PE) wishes to access shared data, it generates a request to shared memory. Such an organization, in practice, can generate considerable delays due to the complexity of the IN linking all PEs to the shared memory, and due to the contention between PEs attempting to access the shared memory simultaneously.

The idea which led to this work undertaken for this thesis was that the access will be more efficient if a non-blocking IN and a new design of shared memory that allows simultaneous access are presented. If the architecture is constructed in this way, all PEs requiring the shared data can access it simultaneously. This improves system performance not only by speeding up the transfer data, but also by improving the organization of the system so that the shared data are ready in demand, and possibly in parallel, which means that shared data is to hand when required.

## Shifted to Multi/many-core Systems

This section is a result of the author’s readings ([Hruska, 2018](#_2hio093); [Oveis-Gharan & Khan, 2018](#_wnyagw); [Patterson & Hennessy, 2020](#_243i4a2); [Singh et al., 2013](#_3gnlt4p); [Stallings, 2013](#_1vsw3ci); [Thomadakis, 2011](#_4fsjm0b); [Udipi et al., 2010](#_2uxtw84)), and the “Advanced Computer Architecture” course notes.

The power of multiprocessors can theoretically be expanded up to any size we want. Simply by adding more PEs to a multiprocessor system, the system's processing power can be raised. However, expanding a multiprocessor system is not as simple as it appears. Normally, many major issues must be addressed; the communication between the PEs and access the shared memory, the scheduling policy of the system and the coherency or consistency in memory values. With the remarkable advancement of semiconductor technology over the preceding two decades, it became possible to build a number of PEs with multi-level caches and IN on a single chip. This is known as a multicore processor system. The multicore architecture is similar to the discrete multiprocessor architectures. They are both plagued by the same architectural issues. The additional issues are related to the amount of space available on the chip; which restricts the number of cores that may be integrated as well as the chip's power consumption. All these problems will be discussed in chapter 2.

With the sophistication of multicore systems applications, the need appeared for a large number of cores. If we assumed that cores number is usually not more than eight cores. The 8-cores in a single-chip called multicore system, but, when the number of cores is expanded significantly over this number the marginal profit performance starts to fall, because, the Bandwidth (BW) of the IN becomes a bottleneck and the consistency values between memories more complex for accommodating a large number of cores. Recently, the many-core system which contains tens or hundreds of cores is a substitution for multicore in various parallel systems. As the number of cores in a many-core system increased, it becomes more difficult for traditional INs to supply the necessary communication BW to serve all cores. On the other side, it will be an uphill work to maintain a consistent view across tens or more of cores; even if we are able to do this, it is difficult to justify the cost related to this design ([Chen et al., 2011](#_3oy7u29); [Martin et al., 2012](#_1idq7dh)). This study is aiming for improving the performance of these systems by providing flexible and scalable IN architecture based on designing a set of clusters and connect them over an effective network.

## 1.3 Thesis Problem and its Motivation

At the beginning of this thesis, multicore systems were identified as a major research area. This is due to the large potential of parallelism that lies with this type of system. Due to the nature of most applications, the program comprises a number of interactive processes. The process communicates through shared data. Interactive processes that are executed by different cores must have access to this data. Hence, this data must be placed in a global area of the memory which is accessible by all cores. The system that can meet the need of such architecture is called shared memory multicore system with each of its cores having its own local memory and all can access the shared memory modules with a single space address ([Mohammed & Abandah, 2016](#_2981zbj); [Patterson & Hennessy, 2020](#_243i4a2)). Therefore, the research area of this thesis was narrowed to this type of multicore system, and work was begun to identify problems associated with this type of multicore. This thesis will concentrate on shared cache architecture and INproblems on shared memory multicore architecture. Figure 1.1 illustrates the taxonomy of parallel processor architecture according to the Flynn's classification ([Flynn, 1966](#_odc9jc)) and the track of this thesis.

The main problem the author identified in such a system was the communication bottleneck problem that lies in the INand accessing the shared memory. This bottleneck problem stems from the fact that more than one core needs to use the same shared resources in the INand the shared memory area at the same time. These shared resources can be links between IN components or a shared memory module. So far, this problem is solved by arbitrating (which is ranges from the decentralized simple control unit to router in most cases) among the cores which are competing for the same resource. The arbitration process creates a sequentially accessing the shared resources in the system. This degrades the parallelism expected from the system. Furthermore, the arbitration components like routers and buffers on NoC pay a heavy penalty in area and power consumption due to the complex structure. So, the arbitrator structure puts additional latency due to the increase in hardware structures that negatively affect the performance. Synchronization (The expression "cache coherence" and "cache consistency" are interchangeable hereafter) is another major problem in a multicore system. It is necessary to guarantee the correctness of the order in which various cores access shared data. This represents a non-compute task to the cores of the system and adds to the communication overhead. A cache coherence protocol is typically used to solve this problem in modern multicore systems. In this study, the shared data synchronization is handled in the shared cache. Synchronization and cache coherence protocols are explained in appendix A.

Figure 1.1.

*Parallel Processor Architecture Classifications*

**Parallel computer architecture**

**MIMD**

**Tightly coupled (multi-processors)**

Multi-processor

**Multicore**

SSM

**DSM**

**Interconnection network**

**routerless Network**

**Shared cache architecture**

**CAM memory**

Loosely coupled (parallel computing)

Distributed systems

SIMD

Vector processor

Array processor

MISD

In some control system

SISD

Uniprocessor

Key: SISD=Single Instruction Stream Single Data Stream computers MISD=Multiple Instruction Stream Single Data Stream computers SIMD=Single Instruction Stream Multiple Data Stream computers MIMD=Multiple Instruction Stream Multiple Data Stream computers (MIMD) SSM= Symmetric Shared Memory DSM= Distributed Shared Memory

## 1.4 Aims and Objectives

To solve the above problems a scalable IN scheme was sought which was capable of:

1. Eliminating the need for arbitration (The expression "eliminating arbitration" and "routerless" are interchangeable hereafter) and providing simultaneous access between cores.
2. Providing the possibility of reading/writing to the shared data simultaneously by more than one core of the system.

Taking the above two points into consideration, the authors proposed a new shared memory and a new IN scheme which completely fulfilled the two points in a multicore system. The shared memory was called DPCAM and the IN scheme was called MPCAM organization. In the later months of the thesis, a new design of a scalable IN scheme based on multicore multi-cluster architecture was proposed. This new version of the scalable scheme was called NCSC. The NCSC is a many-core INthat fulfilled the previous two points also.

## 1.5 Contribution “Significance of the Study”

In this thesis, the INtopology of the large scalable many-core system is thoroughly investigated. The scalable many-core systems architectures are not much different from the conventional multicore system with respect to the main problems that have been referred in the previous sections, but the complexity of these problems will increase as the cores number increase dramatically.

In current multi/many-core systems([Chrysos, 2012](#_11si5id); [Cutress, 2017](#_3ls5o66); [Mulnix, 2017](#_20xfydz); [Ruaro et al., 2019](#_4kx3h1s); [Singh et al., 2013](#_3gnlt4p); [Thomadakis, 2011](#_4fsjm0b); [Torres, 2010](#_302dr9l); [Vajda, 2011](#_1f7o1he)), the communication among cores and the shared memory modules suffer a bottleneck problem that degrades the system performance and increase the power consumption. The main bottleneck and power consumption come from two points. The first is data movement through the INand the second is the bottleneck from shared memory itself. Therefore, redesigning the shared cache memory and moving its location within the INare the first contributions of this work.

The second contribution is the routerless multicore IN. In this scheme, a new INbased on DPCAM modules that are relocated at the cross points of the INis presented. It eliminates the needs of the router device and provides a non-blocking communication between cores and the shared memory modules. It also represents a snooping cache coherence protocol by itself. The MPCAM organization was established by combining the INand the DPCAM shared memory.

Expanding MPCAM based on multi-clusters represents the third contribution. A scalable IN in a many-core system is proposed which is called NCSC. It has the following parts:

1. Effective IN between cores within a cluster (intra-cluster connections) and between clusters (inter-clusters).
2. Low cost, simple structure and easy programming connection between clusters. It provides a routerless IN that eliminates the need for a complex router device.
3. Eliminating the synchronization overhead, that governs the access variables between cores within a cluster and in different clusters that are related to cache coherence protocols.

It should be noted that by addressing the aforementioned problems, the proposed architecture will significantly contribute to the major goal of enhancing the performance of the large-scale many-core system architecture, which will open new windows for manufacturers of parallel-processors computing.

## 1.6 Structure of the Thesis

The thesis is organized into Introduction, five main chapters, and a conclusion. The Introduction introduces the context of this work and explains its motivation and contributions.

Chapter 2 presents a background in multi/many-core architectures. It is based on extensive literature and develops a detailed analysis of all common types of INs known so far. It also includes the related works used to develop the proposed architecture. The conclusion of this chapter leads to the IN scheme that was proposed and developed in this research program i.e the MPCAM and NCSC scheme.

Chapter 3 includes a description and analysis of the proposed shared cache architecture, i.e. DPCAM module. The analysis is given in this chapter is an implementation of a standalone DPCAM using FPGA Intel Cyclone V family and a comparative. It compares the access latency and power consumption of this architecture with the Set-Associative (SA) cache that is used on most multi/many-core architecture.

In chapter 4, the DPCAM is embedded into a multicore system, which it uses as an L2 shared cache, and it is analyzed. The analysis is given in this chapter is a comparative study. It compares various parameters of this architecture with traditional multi-core that use SA cache in the shared level of cache.

Chapter 5 describes the MPCAM organization. FPGA Intel Cyclone V family was used to analyze the features and simulate the results. The analysis is given in this chapter is a functional and timing simulation of the proposed MPCAM. The conclusions reached in chapter 5 have helped in the implementation of the large scale INscheme which is described in chapter 6.

Chapter 6 presents the main contribution of this thesis, which is the NCSC IN. The proposed scheme is described and implemented using the Cyclone IV-E FPGA device family. Furthermore, the proposed NCSC is compared with the state-of-the-art INin terms of static and dynamic performance metrics.

Chapter 7 contains the conclusion of the thesis. In this chapter, the discussion results, the potential application of the proposed architecture, and future works that are related to it are discussed.

# CHAPTER II

# Background and Related Work

Throughout this chapter, the background and related work of the thesis will be explained and the expected problems will be discussed.

In the following sections, the discussion of parallel and multicore systems is given in sections 2.1 and 2.2. In section 2.3, the types of shared memory in computer architecture and major problems of multi/many-core systems are discussed. In section 2.4, the System on Chip (SoC) INs are described and classified into four eras. Finally, the related work is clarified in section 2.5.

## 2.1 Parallel Architecture

There had been an argument on whether a single powerful processor would be fast and more cost effective than a system with multiple slower but less expensive processors. Instruction level parallelism (ILP), memory wall, memory consistency, and area and power overhead are just some of the limitations that come with developing a single powerful processor. These limitations are related to each other. Consequently, architects would not be able to produce a large powerful processor due to the complexity of these limitations. Therefore, the solution is multiple slower processors vs. single fast processors. The advent of modern nanometer technology enables architects to implements these systems.

According to Flynn's classification ([Flynn, 1966](#_odc9jc)), parallel computers architecture can be classified into four classes. Single Instruction Stream Single Data Stream computers (SISD), Multiple Instruction Stream Single Data Stream computers (MISD), Single Instruction Stream Multiple Data Stream computers (SIMD), and Multiple Instruction Stream Multiple Data Stream computers (MIMD). The first two classes can be found in single-processor computers and pipelined processors whereas the third and fourth class is an architecture with several processors. Figure 1.1 describes the Flynn's classification.

In this section a brief presentation of three parallel architectures, their advantages and problems is given.

### *2.1.1 Pipeline Processor Architecture*

The processor that is used in this architecture comprises several stages which are serially linked to each other. Each stage is capable of performing a computational operation. Each processor instruction comprises a number of such operations. e.g, fetch, decode and execute. Several instructions can be pipelined in the processor in such a way that their execution can be overlapped. Ideally, the execution time of one instruction will be equal to the time needed by the stage of the processor to execute an operation. Due to the memory conflicts, data dependency, branches, and interrupts, this ideal speed-up may not be achieved ([Patterson & Hennessy, 2020](#_243i4a2); [Stallings, 2013](#_1vsw3ci)).

Parallelism in a pipelined processor can be degraded due to memory conflict, data dependency, branching, and interruptions. The number of stages in a pipelined processor is limited by the possible number of operations that can comprise a pipeline task. Hence, pipeline parallelism is limited. The manufactures of pipelined computers realized this point and developed multiprocessor and multicore versions of their system e.g, the 4 processor Cray 2 system ([Agarwal et al., 2010](#_16x20ju))and modern intel pipeline processor ([Bohnenstiehl et al., 2016](#_338fx5o); [Chrysos, 2012](#_11si5id); [Mulnix, 2017](#_20xfydz)) .

### *2.1.2 Array Processor Architecture*

In this type, a central control unit (which is effectively a processor) masters an array of PEs. The central control unit fetches the program instruction from its local memory. If the instruction is to be executed by the processing elements, the central control unit broadcasts the instruction to all PEs simultaneously. The PEs use the instruction to act upon several data items simultaneously and return the result to the central unit. Hence, the PE array behaves like a big co-processor that is mastered by the central control unit, i.e. the relationship is a master-slave one. Computers that belong to this one architecture are classified by Flynn as SIMD computers. Many manufactured products are related to this type e.g, vectors and Graphics Processing Unit (GPU) parallel computer ([Agulleiro & Fernandez, 2015](#_3qwpj7n); [Cebrian et al., 2020](#_261ztfg); [NVIDIA, 2017](#_l7a3n9)).

The data memory may be local to each PE or it may be a global memory connected to each PEs through an IN. These two possible schemes are shown in figure 2.1-2.2.

In such computers, parallelism is limited by the following factors:

* + 1. The number of instructions in the program which can be executed by the processing element array.
    2. The number of processing elements which it practical to include in the array.
    3. The communication overhead resulting from the communication between processing elements and data memory or PEs via IN.

The only hope so far for unlimited parallelism is the MIMD computer in which large number of processors can be employed. However, MIMD computers (Known as multicore systems) have their own problems, which degrade their parallelism.

### *2.1.3 Multiprocessor system*

Computers of this type utilize several equal processors. Depending on the degree of interaction among the processors of the system, multiprocessor systems are MIMD. In MIMD each processor fetches its instructions and operates on its data. MIMD multiprocessors are classified into two categories, loosely coupled and tightly coupled ([Flynn, 1966](#_odc9jc); [Patterson & Hennessy, 2020](#_243i4a2)).

In loosely coupled systems, the processes that are executed by different processors are logically autonomous, i.e. one process does not exercise direct control over another. This means that there is no direct sharing of processor address space and hence no sharing of primary memory is required. At the physical level, this means that processors do not communicate through a shared memory. Rather, they communicate by message passing. This class is popular for clusters computing which often use standard network technology ([Hennessy & Patterson, 2019](#_1kc7wiv)).

In a tightly coupled multiprocessor system, in contrast to a loosely coupled one, the system mostly executes a single program on several processors and it has overlapping primary address spaces, i.e. they share these spaces. Hence communicating through shared memory is required ([Mohammed & Abandah, 2016](#_2981zbj); [Patterson & Hennessy, 2020](#_243i4a2)). This class is popular for desktop or server computers with multicore and many core systems. It is a pool of homogeneous processors running independently where each processor executes its own instructions ([Kaplan, 1987](#_44bvf6o)). In some cases like the LDF 100 system, the processors are logically tightly coupled but implemented in a physically loosely coupled system i.e. with no shared memory. The interconnection medium, in this case, must have a bandwidth that can meet the need of a highly interacting process.

In tightly coupled systems, the need for effective INwith a large bandwidth is quite obvious. This INserves the processor-processor and/or processor-shared memory communication process. Tightly coupled multiprocessor systems are classified into two models; Symmetric Shared Memory (SSM) models and Distributed Shared Memory (DSM) models ([Patterson & Hennessy, 2020](#_243i4a2)).

Since shared memory multiprocessor systems are the best candidates known so far for massively parallel processing, the work presented in this thesis concentrates on the problems associated with this type of system.

Figure 2.1

*Array processor architecture with local memory*

PE1

M1

PE2

M2

PEn

Mm

Inter-PE connection network

Key: CP: Control unit processor, CM: Control unit memory, PE: Processor element, M: Memory

CP

CM

Central control unit

Control

Data Bus

Instruction Bus

Figure 3.2

*An Array Processor Architecture With Global Memory*

PE1

PE2

PEn

Inter-PE connection network

Key: CP: Control unit processor, CM: Control unit memory, PE: Processor element, M: Memory

CP

CM

Central control unit

Control

Data Bus

Instruction Bus

M1

M2

Mm

## 2.2 Multicore System Architecture

As it was explained, a multiprocessor on a chip is an SoC which includes multiple PEs, known as a multicore system. This includes a number of pipelined cores and their components (i.e. caches and INs) built on a chip. Multicore systems are considered as MIMD and tightly coupled. Most mainframe computers with a large number of processors are MIMD and tightly coupled ([Patterson & Hennessy, 2020](#_243i4a2)).

A multicore system usually consists of several cores which cooperate to execute a program. Multicore systems architecture can be classified into three classes:

1. Distributed Shared Memory (DSM) systems: In this class, there is no specific shared memory module.
2. Symmetric Shared Memory (SSM) systems: In this class, the system has local and specific shared memory.
3. Multicores with global memory: In this class, the system has global shared memory only.

In the first and second classes, the program is divided into streams that can be executed in parallel. Each stream resides in the local memory of a core. During the execution of a program, the processes of the program need to pass information to each other. If the communication process resides in the same core, a communication passing is straightforward, but if they are in different cores, a communication facility is needed. This communication facility is referred to as the IN. Most of the real multicore systems are adopted these two classes ([Patterson & Hennessy, 2020](#_243i4a2); [Stallings, 2013](#_1vsw3ci)). Therefore, the following subsections will address these two classes. In the third class, the core of the system accesses the global shared memory via the IN in all access cases. This class is not found in a tightly coupled multicores system but it is implemented in a loosely coupled class ([Feeley et al., 1995](#_ymfzma)).

### *2.2.1 SSM Architecture*

In SSM architecture, the cores and shared memory modules are on opposing sides of the IN. Each core comprises a pipelined core and private level/s of cache. Figure (2.3-a) shows SSM architecture, and figure (2.3-b) shows a core unit.

The following points can be noticed while looking at this architecture:

1. Through the IN, every core can access any shared memory module.
2. When multiple core try to access the same destination at the same time, even with the best interconnection NoC, A bottleneck still exists.
3. appropriate for small number of cores
4. The cache coherence protocol adds to the network's overhead and creates a bottleneck. At any one time, only one core can broadcast the updated shared data. Hence, the INprovides all necessary communication between cores and shared memory.

Figure 2.3-A

*SSM Architecture*

PE = Processing element

SMM = Shares Memory Modules

Shared Memory Modules

PE1+private caches

IN

SMM1

PE2+private caches

PEn + private caches

SMM2

SMMm

Key: SMM= shared memory module PE= Processing element

Figure 2.3-B

*Core Unit*

LM = Local Memory

PE

PM

Key: PE= processor elements PM= Private memory

LM: Local Memory

### *2.2.2 DSM Architecture*

As illustrated in figure 2.4, each core in the DSM multicore architecture has its own shared cache plus its private cache memory. [Protic et al. (1996)](#_2wwbldi) presented a survey of DSM architecture. INconnects between all cores in this architecture. The process mostly communicates with multiple cores during task execution.

The following points can be noticed while looking at this architecture:

1. Through the IN, any core can access any shared memory that belongs to other cores.
2. DSM systems also have communication bottleneck (even for the best IN)
3. The cache coherence protocols add a heavy overhead to the network and this overhead will increase as the system is expanded.

A hybrid DSM architecture was organized in ([Chen et al., 2011](#_3oy7u29); [Chen et al., 2015](#_3w19e94)). The main goals of hybrid DSM are to provide fast access to private data while keeping shared data in global memory.

From the above list of SSM and DSM connections, it is clear that the purpose of the network is to provide connections between the cores and the shared resources. Since the network connects the cores to shared resources, the network is considered as a shared resource itself.

Figure 2.4.

*DSM Architecture*

SMM

SMM

PEn+

Caches

PE2+

Caches

PE5+

Caches

PE4+

Caches

SMM

SMM

SMM

SMM

PE1r+

Caches

Interconnection Network

PE3+

Caches

Key: SMM= shared memory module PE= Processing element

## 2.3 Many-core System Architecture

As mentioned in chapter 1, the many-core system is a special type of multicore specifically built for huge parallel processing. Many-core systems usually contain tens to hundreds of cores or more. They are often employed in embedded systems and high-speed processing ([Pereira et al., 2021](#_2b6jogx)). Many-core processors differ from ones in multicore in that they are designed from the beginning to have a higher level of explicit parallelism and reduced power consumption ([Hennessy & Patterson, 2019](#_1kc7wiv)).

The many-core system architecture is not much different from the multi-core architecture. They both share the same architectural issues. Cache coherency, a problem that limits multicore system scalability, has been addressed in many-core processors using techniques such as message passing. Read only non-coherent caches and a partitioned global address ([Olofsson et al., 2014](#_qbtyoq)). The INs of multi/many-core systems, particularly NoC, are used by a many-core system ([Hennessy & Patterson, 2019](#_1kc7wiv)).

In many-core architecture, whether a system is divided into clusters or all cores are allowed to access any destination in the communication medium, limited numbers of options are available concerning INused topologies. The first option, based on cluster many-core systems. In this architecture, usually a server (manager) core is provided for each cluster and the cores compete for accessing the server through local network topology and the servers of the clusters compete for the global network topology ([Hamid et al., 2015](#_3abhhcj); [Ruaro et al., 2019](#_4kx3h1s); [Udipi et al., 2010](#_2uxtw84)). Figure 2.5 depicts based clusters many-core system architecture. The second option, many-core systems based on grid architecture. In this architecture, the cores of the whole system can compete for accessing the global grid network topology, most of the conventional topologies use this option like the tree, ring, mesh, torus and the mesh of tree, and hypercube ([Abdullah et al., 2011](#_1pgrrkc); [Alam & Varshney, 2015](#_49gfa85); [Awal et al., 2015](#_2olpkfy); [Li et al., 2017](#_13qzunr); [Li et al., 2019](#_3nqndbk); [Ou et al., 2020](#_22vxnjd)). Each core consists of PE, private level of cache, shared cache and router to access the destination through the network topology. Figure 2.6 depicts many-core based grid network architecture.

In both options, the final destination is the shared memory of other cores. It is obvious that both options complicate the communication of the system and its programming. The same argument applies for the quick path, if any exists, among the clusters. Therefore, it is vital to provide high-performance communication between cores through the IN. In this chapter, the INs in the multi/many-core systems is introduced.

Several manufacturers of many-core systems have recently released real many-core systems. The previous two architecture options are used in these real systems. The Tile architecture was developed at Massachusetts Institute of Technology (MIT) ([Bell et al., 2008](#_i17xr6); [Vangal et al., 2008](#_320vgez)) the system is scaled up to 64 and 100 cores. It uses a grid architecture with interconnect network called iMesh.

Rigel architecture with 64,128, 1000, and 1024 processor cores is another example of the many-core system ([Johnson et al., 2011](#_1h65qms)). In this architecture, every eight cores share a large cache and constitute a Rigel cluster. Cores in one cluster communicate through their shared cache. Every 16 clusters are arranged into a tile using a bidirectional tree-topology IN. The GPU was first introduced in NVIDIA ([Kilgariff et al., 2018](#_415t9al); [NVIDIA, 2017](#_l7a3n9)). This system is distributed into 16 multi-core clusters. The pico-Chip system is another example of many-core system. A pico-Chip system consists of more than 256 processing elements with different specifications; it usually uses a mesh INthat called the pico-Array. It is mostly similar to the iMesh network used in Tilera’s.

Intel introduced the first commercial many-core system called Knights-Corner that was manufactured at a 22 nm process technology, with more than 50 cores ([Duran & Klemm, 2012](#_2gb3jie)). The second generation of Intel many-core system was introduced at 2013 code name Knights Landing. Intel's produced this architecture using 14 nm technology. It contains more than 72 cores ([Anthony, 2013](#_vgdtq7)). The third-generation was called Knights Hill produced using 10 nm technology ([Gardner, 2014](#_3fg1ce0)). In 2017, Intel decided that Knights Hill architecture had been canceled to be substituted by another powerful architecture built from the ground up to enable powerful processing in the future. This new architecture is expected to be in commercial between 2021–2022 the codename for this future work is Exascale supercomputers ([Damkroger, 2017](#_1ulbmlt)). Finally, Knights Mill is the last Intel version for many-core processors used in deep learning, the initial release display at the end of 2017 ([Cutress, 2017](#_3ls5o66)). In the end, it can be noticed that there are few architecture details available publicly about the Intel devices.

Figure 2.5

*Many-core System Based Multi-cluster Architecture*

Global Interconnection

Core11

Core1n

Local interconnection Cluster 1

Coren1

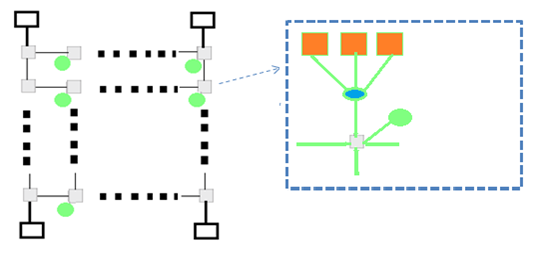
Corenn

Local interconnection Cluster n

Key: core11= core 1 in cluster 1 core1n= core n in cluster 1

Figure 2.6.

*Many-core System Based on Grid Network Architecture.*



Key:

Core+ private caches

Wires

Local switch

L3- shared cache memory

Router

Memory controller

## 2.4 Major Problems in Multi/many-core System Architecture

The key problems with multi/many-core systems are briefly summarized in this section.

### *2.4.1 Communication Problem*

This problem is related to INs architecture. Figures 2.2-A and 2.4 show multicore architectures for both SSM and DSM, as well as figures 2.5 and 2.6 depict many-core architecture based on multi-cluster architecture and many-core systems based on grid network architecture. The INrepresents the heart of these architectures. The communication problem indicates that the INmust provide effective communication between cores as well as between cores and shared memory.

In general, several INof multi/many-core were proposed, implemented, and simulated. Historically, two types of networks were developed that connects between cores in the system, the first is traditional INs (buses based interconnection) common bus, multiple bus, crossbar switch, multistage INs (Omega, Baseline, Butterfly…), and ring are examples ([Com et al., 2007](#_18vjpp8)). When more than one PE accesses the shared bus or requests the same shared memory, all of these topologies face a bottleneck. The second one is called NoCs that use routers and buffers to create multiple paths between cores for improving throughput, there are two schemes in this type buffers and buffer-less. Tree, mesh, mesh of tree, hypercube and torus are examples ([Abdullah et al., 2011](#_1pgrrkc); [Alam & Varshney, 2015](#_49gfa85); [Awal et al., 2015](#_2olpkfy); [Li et al., 2017](#_13qzunr); [Li et al., 2019](#_3nqndbk); [Ou et al., 2020](#_22vxnjd); [Sun et al., 2007](#_3sv78d1)). Most research groups have been working to improve the performance of this scheme recently, either by developing a hybrid NoC that combines buffer and buffer-less ([Fallin et al., 2012](#_280hiku)) or by proposing a new scheme termed routerless NoC that eliminates the need for a router ([Alazemi et al., 2018](#_n5rssn); [Awal et al., 2015](#_2olpkfy); [Liu et al., 2015](#_375fbgg); [Udipi et al., 2010](#_2uxtw84); [Xiao et al., 2019](#_1maplo9)).

In addition, two communication techniques among the system cores were used; communication using a shared variable, and message passing techniques ([Patterson & Hennessy, 2020](#_243i4a2); [Stallings, 2013](#_1vsw3ci)). The communication through shared variables must be synchronized to ensure that the cores access the shared data at the correct time and in the correct version to obtain optimal performance. This problem leads to the cache coherence issue ([Martin et al., 2012](#_1idq7dh); [Patterson & Hennessy, 2020](#_243i4a2)). In message passing technique the cores communicate among themselves through packet-switched communication, this overcomes the coherence problem, but it raises new issues that associate with the concepts of communication networks such as routing algorithms, selection methods, switching techniques, flow and congestion control, etc ([Patterson & Hennessy, 2020](#_243i4a2); [Stallings, 2013](#_1vsw3ci)). As mentioned above this thesis concentrates on shared memory architecture.

The following section 2.5 will cover the INs in more detail. They are divided into four eras, with some performance analysis of each era.

### *2.4.2 Coherence Problem*

Coherence problem happens in all shared memory multi/many-core systems that used shared links communication medium. Each core is provided with Level 1 (L1)/ Level 2 (L2 )private caches, Level 3 (L3) shared cache and a cache controller ([Martin et al., 2012](#_1idq7dh); [Schauer, 2008](#_2lfnejv)), here the cache controller observes the bus transactions of all other cores and takes suitable action to preserve the consistency of the copy of the shared values in the cache. Snooping Cache Controller is a cache controller that monitors bus traffic for consistency purposes. Since each core has its own caches and multiple caches of the system are allowed to have copies of memory location or a shared memory, these copies of the data might not always be the most recent. It is possible to have a number of copies; one copy can be stored in shared memory, while other copies can be stored in individual cache memory. A technique must exist in order to ensure that all copies remain consistent when the contents of that location (or that block) are modified ([Patterson & Hennessy, 2020](#_243i4a2)). A piece of software called cache coherence protocol is used to help the cache controller in maintaining the consistency of the multiple copies. Generally, in multi-core architecture, the two most common cache coherence protocols are a snooping protocol and a directory-based protocol. ([Patterson & Hennessy, 2020](#_243i4a2); [Stallings, 2013](#_1vsw3ci)).

In the snooping protocol, any change in the state of the shared data is detected by the snooping cache controller and reported to the cache. All controllers can receive the update simultaneously and take an action on that basis. The core polls the state of the block or the variable locally which reduces the traffic on the network. If the copy is the most recent version of the block the processor reads it locally, otherwise it presents a request to access the global locations through the network. [Archibald and Baer (1986)](#_10kxoro) have reported these protocols. As a detailed discussion of these protocols is out of the scope of this thesis, only one of these protocols is presented in order to demonstrate how they work and their effect on the synchronization and hence the communication overhead. This protocol is the Goodman's Write-Once Cache Coherence method ([Archibald & Baer, 1986](#_10kxoro)). On some multi/many-core systems, the directory-based coherence protocol is employed; it is more scalable to a large number of cores than the snooping protocol. In this approach, a directory is utilized to store information that was previously shared across numerous cached memory locations ([Martin et al., 2012](#_1idq7dh); [Mittal, 2014](#_3kkl7fh)). A detailed description of this problem is shown in appendix A.

The cache coherence protocol puts an additional overhead on the INs. At a time, one core can transmit the updated version of the shared data. Adding additional cores in multicore systems has a greater effect on the time required to validate the protocol ([Martin et al., 2012](#_1idq7dh); [Zhang et al., 2010](#_4jpj0b3)). As a result, it creates a scaling issue.

### *2.4.3 Scheduling Problems*

Scheduling policy in multi/many-core systems decides how the program is partitioned into processes, how these processes are scheduled on a set of cores, and how these processes communicate has a great effect on the performance of the system. A good scheduling policy is one that partitions and schedules the program in such a way that the following points are achieved.

1. Maximum possible number of the cores of the system are employed in executing the program.
2. A good load balancing among the cores of the system.
3. Minimum possible number of shared data items are transferred between the cores; this means fewer dependencies between cores.
4. Minimum non-compute activities for the processors of the system: Such as program management activities, which core performs during the run-time and synchronization instructions.

Two types of scheduling policies have been used, static and dynamic policies. In static policy, the processes are assigned to the processors at the compile-time ([Xu & Ding, 2010](#_3xzr3ei)). The set of processes assigned to a core regards the program stream dependency. Dynamic scheduling, on the other hand, assigned the process to the cores at runtime. It is also widely employed when the loads are adjusted on a regular basis. Several static and dynamic scheduling policies have been constructed in recent years, however, research on this topic has been overkilled, and this is out of the scope of this thesis.

### *2.4.4 Power and Area Problems*

With the advanced fabrication of SoC, another tremendous issue comes is the power dissipation issue. Researches have shown that the dissipation increases as the SoC area shrink more and more and as the processor frequency increase, this increases both static and dynamic power consumptions to large values ([Roy et al., 2008](#_2d51dmb)). The power had increased about 32 times from the 80386 release through 2005 ([Danowitz et al., 2012](#_sabnu4)). Power will be affected by both frequency and size area, the frequency cannot be scaled further. Therefore, we couldn't have more intelligent devices inside cores. In SoCs, there are two types of power dissipation: dynamic and static.

If the transistor switches to change the voltage of a specific node, dynamic power consumption will occur. This happens when the processor fetches and stores instructions (read or write). So, dynamic power occurs during the unit's activity and signal toggling. On the other hand, static power is dissipated when the device is inactive. So, it mostly depends on the number of transistors of any design. Both static and dynamic power in the cache can be modeled mathematically see equations 1 and 2. This helps to gain insight into how power conservation techniques operate. More information can be found in ([Com et al., 2007](#_18vjpp8); [Zhang et al., 2010](#_4jpj0b3)).

(1)

(2)

represents the activity factor, is the voltage, represent the leakage current, is the effective capacitance, refer to the frequency, refer to the amount of transistors in a system , and refer to the design dependent parameter.

From Eq. (1) the dynamic power can be saved by changing the operating voltage and frequency or by minimizing the factor. From Eq. (2), in the same way, saving static power can be achieved by redesigning the structure to use low power cells, decreasing the number of transistors or redesigning certain parts of the system in a low leakage mode. According to the principles of the two equations, several cache techniques have been designed to save total power consumption. These techniques can be divided into three types: first, dynamic energy saving. Second, static power savings, and third, both dynamic and static saving techniques.

In fact, cache memory and routers are the two main components that consume the maximum power inside the NoC. Researches have shown that cache memory consumes between 20-40 % of chip power depending on the type of chip ([Fischer et al., 2016](#_3c9z6hx); [Li et al., 2009](#_1rf9gpq); [Qodirov et al., 2019](#_4bewzdj)). Current researches have already verified that NoC consumes about 40% of the chip's power without the cache power consumption ([Hoskote et al., 2007](#_2qk79lc)) and routers cause about half of this power. Therefore, a lot of research work has been done to introduce power-saving techniques in NoC architectures for both cache hierarchy and router.

At the beginning of this thesis, a survey for NoC power-saving techniques was introduced. This is due to the large importance of power in any system architecture. Therefore, the decision was taken that the shared cache memory and router must be redesigned to improve the power dissipation in the proposed IN.

## 2.5 SoC Interconnection Networks

The efficiency of multi/many-core systems is affected by the number of cores waiting to access shared resources in the same span of time. The longer the cores have to wait, the more the system parallelism and efficiency are degraded. The fact that the network and other shared resources, mainly shared memory, accept a limited number of the requests made by the cores at any time makes the rest of the requesting cores wait for their requests to be honored. The more connections provided by the network at a given time the greater is the efficiency of the system. The BW or connectivity is the main factor in IN, it is described by ([Al-Ali & Jesshope, 2000](#_3pp52gy)) as the average number of simultaneous requests accepted by the IN. Complexity is another factor which must be taken into consideration in designing an IN, which is usually indicated by the number of routers and the associated control and arbitration used in the network. The complexity can affect the design in two ways, the first is the practicality, and the second is the cost. Scalability and modularity are other two important factors in IN. Scalability is the possibility of adding more cores to the network. Modularity refers to the effect of this expansion on the performance of the network. According to Stenstrom ([Stenstrom, 1988](#_24ufcor)), for multicores to be modular (meaning that bandwidth must be proportional to the number of cores) the network BW must be increased to meet the growing number of requests made by the added cores.

The above parameter mainly decides the performance and the cost of the network. Yet more decisions have to be made to decide the shape and function of the network. Control strategy and network topology are the primary parameters. Control strategy refers to the number of links and switches or routers elements of the IN. These elements can be controlled centralized or decentralized. Network topology refers to the arrangement of the (cores, routers, and links) elements of an IN ([Patterson & Hennessy, 2020](#_243i4a2)). Network topology can be static or dynamic. In static networks, links between cores are passive and dedicated buses cannot be reconfigured for direct connection to other cores e.g. near-neighbor mesh network ([Com et al., 2007](#_18vjpp8); [Stallings, 2013](#_1vsw3ci)). On the other hand, links in the dynamic network can be configured by setting network switching elements, e.g., the crossbar switch network.

The on-chip interconnects can be classified into four eras. Buses era, crossbar era, NoC based router era, and the Routerless NoCs era. These INs with a brief explanation of the problems and the performance are presented in the following subsections.

### *2.5.1 The Buses Interconnection era*

The first generation of SoC INs consisted of the traditional common bus and some additional cross-point structures. The bus mostly consists of sets of wires, use to connect between cores and memory units in some architectures, usually add an arbitrator that manages the access to the shared buses. The common bus and Multiple Buses are obvious examples of this type.

#### 2.5.1.1 The Shared Bus INs. The shared bus is the first, relatively inexpensive, easiest to configure and simplest interconnection topology. All processors of the system are connected to a shared path figure 2.7.

The shared bus is a bottleneck in and of itself. It serves only core access at a time, so to reduce bottleneck an arbitrator is needed and any access to the network must take permission from an arbitrator ([Edwards, 1986](#_33zd5kd)).

The disadvantages of the shared bus are twofold:

1. A system failure occurs in the case of a malfunction in any of the bus interface circuits.
2. There is a limit on expansion: by adding more cores the shared bus contention increases and this degrades the system performance and increases the complexity of arbitration. In this case, the shared bus is not modular since it has a fixed BW.

The disadvantages of the shared bus led to introduce a multi-layer bus interconnection that improves the bandwidth.

#### 2.5.1.2 The Multiple Bus Networks. Several studies have suggested that connecting N cores to M shared memory modules using a set of buses S equal to half the number of cores, pluses one, this design is called the multiple bus interconnection (see figure 2.8). A small number of requests, fewer than or equal to the number of buses, can be served by this interconnection. Only one request can be served in case of two or more requests are sent to the same destination ([Mudge et al., 1986](#_1j4nfs6)). Examples of the bus and multiple bus architectures are the Advanced Microcontroller Bus Architecture (AMBA) from Advanced RISC Machines )ARM(, Inc, and the CoreConnect™ from International Business Machines Corporation (IBM).

### *2.5.2 The Crossbars Networks era.*

The advent multicore on SoCs that contains tens of cores created a significant challenge on buses architecture. So, a new solution of crossbar switch plus the buses was suggested. Simply these topologies consist of busses and crossbars switches. Both crossbar and multi-stage switch networks are obvious examples of this type.

#### 2.5.2.1 The Crossbar INs. Crossbar connectivity is achieved by increasing the number of buses in horizontal and vertical connection for each core and shared memory so that buses can access at cross points. As shown in figure 2.9 crossbar switch can be viewed as a number of vertical and horizontal links interconnected by a switch at each intersection.

However, the blocking happens if more than one cores access simultaneously the same shared memory. Only one request will be fulfilled ([Com et al., 2007](#_18vjpp8)) . To solve this problem, an arbitrator must be inserted on every bus going to shared memory units. The arbitrator adds extra complexity and cost to the crossbar design.

#### 2.5.2.2 Multistage INs. Multistage Interconnection Networks (MINs) are another example for connecting cores and to shared memory units ([Com et al., 2007](#_18vjpp8)). The crossbar network establishes a complete link between the cores and the memory units. However, as previously stated, the cost and complexity of the switches increase rapidly as scale the system. When creating a crossbar system with 2 x 2 switching elements, the total number of switching elements required is (N/2)2. Various interconnections of MINs were proposed such as Delta networks, Omega networks, and Butterfly networks ([Francalanci & Giacomazzi, 2005](#_2i9l8ns); [Patterson & Hennessy, 2020](#_243i4a2); [Yunus & Othman, 2011](#_xevivl)). Figure 2.10 shows that the bottleneck of MINs occurs if multiple requests use the same link or access the same shared memory module.

Figure 2.7

*Shared Bus Network*

Core 1

Core 2

Core n

SMM 1

SMM 2

SMM m

Common bus

Sn

S2

S1

Key: S = Switch

Figure 2.8

*Multiple Bus Network*

SMM1

SMMm

SMM2

core1

coren

core2

S Buses

Figure 2.9

*Crossbar Switch Networks.*

SMM2

SMM3

SMMm

core1

core2

core3

Core n

SMM1

Switch

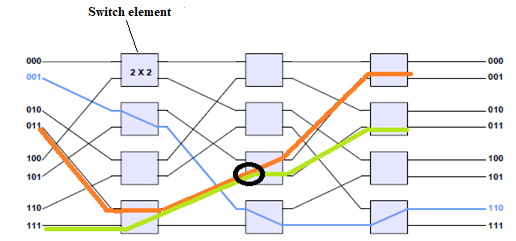
Arbitrator

Requests

Switching signals

Figure 2.10

*MINs 2X2 switch element (*[*Com et al., 2007*](#_18vjpp8)*)*



### *2.5.3 The NoCs based router era.*

The advantages of the previous two types of INs are a simple topology, easy to implements, and low cost. On the other hand, they have several challenges which will reduce its use in modern multi/many-core systems, such as scalability, latency in the large system, and power issues. With advances in SoCs, systems can have billions of transistors; NoCs become the optimal solution to overcome these challenges due to many features such as power efficiency, scalability, effective bandwidth, and others. In NoC architecture, the direct paradigm each node contains one core with its cache and router, each node sends data to another through the packet switched communication with deterministic best path. Whereas, in the indirect paradigm, not every router is connected to a core such as the direct. Some routers are used only to transmit the packets through the path on the network. NoCs architecture has many topics to be addressed. They can be summarized as routing algorithm inside the router, flow control and error control between packets, control units or arbitration, buffering and buffer-less the packets and flits, and the interconnection topology ([Kamal & Yadav, 2012](#_1wjtbr7)). There are many networks topology like tree, butterfly multi-stage in in-direct paradigm and hypercube, mesh, torus, and ring in direct paradigm ([Cota et al., 2011](#_4gjguf0)). Because mesh and ring interconnection topologies are commonly used in multi/many-core systems, they are addressed in the following sections.

#### 2.5.3.1 Mesh IN. The 2-dimensional (2D) mesh IN is a famous NoC used in multi/many-core systems. It can be implemented by an m × n grid, where m and n refer to the numbers of rows and columns respectively. Figure 2.11 shows the 4X4 mesh. Meshes can be implemented in any dimension ([Patterson & Hennessy, 2020](#_243i4a2)), but, the common manufacturing systems are mostly based on Two Dimensional (2D) mesh. Most Intel and Tile products use 2D-mesh such as Teraflop, Knights Hill, Xeon, and Skylake-X Server ([Alam & Varshney, 2015](#_49gfa85); [Chrysos, 2012](#_11si5id); [Damkroger, 2017](#_1ulbmlt); [Ou et al., 2020](#_22vxnjd); [Vangal et al., 2008](#_320vgez)). Moreover, several NoC simulators and studies chose 2D-mesh as referenced architecture, because of its reliability, low cost, and low complexity ([Murali et al., 2007](#_2vor4mt)).

#### 2.5.4.2 Ring IN. Ring interconnection topology consists of a set of cores, where each one is connected to the former and posterior core. Figure 2.12 depicts A ring topology with N = 8, where N is the number of nodes. This network is a special type of torus IN. Recently, the ring topology was used in many industrial projects. The 16 modules interconnect bus in SONY and IBM was designed with dual rings that connect more than ten cores ([Kistler et al., 2006](#_1au1eum)). Intel also uses dual-ring topology in Intel I9 Xeon and other series ([Mulnix, 2017](#_20xfydz)).

Figure 2.11

*2-D Mesh Interconnection*

Router

Router

Router

Router

Router

Router

Node

Figure 2.12

*Ring Interconnection*

Router

Node

Node 1

Node 2

Node 3

Node 4

Node 5

Node 6

Node 7

Node 8

### *2.5.4 The Routerless NoCs era.*

The conventional networks in the first two eras are simple, low cost, and easy to implement, but with less ability of scalability. While NoCs based router have the best ability of scalability, but it incurs significant power consumption and area space because of the complex router architecture. Due to the complicated router structure, NoCs pay a high price in terms of area and power consumption. It also adds additional access latency due to increased hardware structures from both routers and topological links, which negatively affect the performance ([Hoskote et al., 2007](#_2qk79lc)). Researchers start with the strenuous pursuit to produce NoCs without a router, which is called routerless NoC. ([Alazemi et al., 2018](#_n5rssn); [Liu et al., 2015](#_375fbgg); [Udipi et al., 2010](#_2uxtw84); [Xiao et al., 2019](#_1maplo9)) are examples of these works.

The need becomes urgent in the future when the NoCs serve a large number of cores to eliminate the router complexity, which imposes overheads in power, latency, area, and complexity. Therefore, the new class of NoCs that refer to Routerless NoCs, without the need for routers is the state of the art in this area. This opens a wide area to search in routerless NoC because it represents a great hope for increasing the scalability of SoC to hundreds of cores without distracting effort on the area and power exploited on running routers and buffers.

## 2.6 Related Work

As already mentioned, the shared cache memory and router are the key components that increase latency and bottlenecks over INs in multi/many-core systems. Further, Current researches have shown that these two components consume more than half of the chip's power consumption ([Hoskote et al., 2007](#_2qk79lc)). This section brieﬂy lists the literature review related to multi/many-core INs concentrates with shared cache and routerless NoC architecture. Therefore, these works were divided into two classes. Architecture that focused on the shared cache architecture ([Chauan et al., 2015](#_p49hy1); [Huang et al., 2016](#_393x0lu); [Imani et al., 2016](#_1o97atn); [Irfan, Ullah, et al., 2019](#_488uthg); [Monazzah et al., 2016](#_2ne53p9); [Naderializadeh et al., 2017](#_12jfdx2); [Nguyen et al., 2018](#_3mj2wkv); [Syed & Ullah, 2017](#_21od6so); [Syed et al., 2017](#_gtnh0h)) and Architecture that focused on the proposed new routerless IN ([Alazemi et al., 2018](#_n5rssn); [Awal et al., 2015](#_2olpkfy); [Li et al., 2017](#_13qzunr); [Li et al., 2019](#_3nqndbk); [Lin et al., 2020](#_30tazoa); [Liu et al., 2015](#_375fbgg); [Udipi et al., 2010](#_2uxtw84)).

### *2.6.1 Related Shared cache architecture*

In multicore processors, there are multi-level of caches and most of them are set-associative ([Patterson & Hennessy, 2020](#_243i4a2); [Stallings, 2013](#_1vsw3ci)). Shared level is usually shared across the cores and is placed on the system chip. Various types of Associative Memory (AM) and CAM have been designed and implemented on the FPGA to be use on special purpose applications ([Ahmed et al., 2016](#_3zy8sjw); [Irfan & Ullah, 2017](#_2f3j2rp); [Nguyen et al., 2018](#_3mj2wkv); [Prasanna Venkatesan](#_u8tczi); [Ullah, 2017](#_3e8gvnb)). However, some of these works suffer from memory efﬁciency due to the limited size, update latency, power cost, low density etc. Thus, they could not be used as shared level of cache in modern multi-core system. There are several works on two approaches to improve memory efficiency by an architectural design ([Chauan et al., 2015](#_p49hy1); [Imani et al., 2016](#_1o97atn); [Irfan, Ullah, et al., 2019](#_488uthg)) and by efficient cache replacement algorithms ([Huang et al., 2016](#_393x0lu); [Monazzah et al., 2016](#_2ne53p9); [Naderializadeh et al., 2017](#_12jfdx2); [Syed & Ullah, 2017](#_21od6so); [Syed et al., 2017](#_gtnh0h)).

AM architecture using Virtex-6 FPGA series inside the cache controller was presented that was designed to work as a look-up table inside the cache controller with size of 1KiB and a block size of 16 byte ([Chauan et al., 2015](#_p49hy1)). The simulation results show that the cache controller's setup latency is 1.66 ns and total power consumed is 5.53 mW. The main disadvantage of this design is that difficult to scale it to a bigger size. Therefore, it cannot be used in shared memory.

Resistive Conﬁgurable Associative Memory (ReCAM) is used to solve some of the issues. ReCAM improves the access latency by limiting the load/store fatigue at the beginning of executions ([Imani et al., 2016](#_1o97atn)). ReCAM uses hamming distance algorithm which searching the nearest cell for reading and writing. It exploits this feature to design a memory with better efficiency in both performance and power consumption. In ReCAM architecture, the processing element has two kinds of execution units: the first is composed of ReCAM arrays where memories are connected using a crossbar. The second is a traditional core. The main disadvantage of this design, it was implemented on single processor system and not suit for multi-core systems. Whereas the DPCAM purpose to be used in multi/many-core processors.

The design called Gate-based Area Efficient Ternary Content Addressable Memory (G-AETCAM) which uses flip-flops as FPGA storage elements and it can be configured as binary and ternary CAM where gate levels reduce the resources on FPGA ([Irfan, Cheung, et al., 2019](#_1tdr5v4)). The design has been implemented in different sizes for the Virtex-5,-6 and-7 FPGA series. The performance is increased by 28 percent compared to the other FPGA-based Ternary content-addressable memory (TCAM). It also facilitates better scalability than other TCAMs due to less complexity of the architecture. In ([Irfan, Ullah, et al., 2019](#_488uthg)), the authors presented an efficient FPGA resources and power consumption called Zi-CAM. It has less complexity and power consumption than traditional RAM-based CAM designs on FPGAs. The internal structure consists of two main units: RAM unit and lookup tables unit. Each unit is activated according to the sequence of data. The design has been implemented on Virtex-6 FPGA. The presented results showed that Zi-CAM improved FPGA resources cost, power consumption and update latency when compared to common FPGA-based CAMs. The main benefit is update latency of Zi-CAM is nearly constant with different sizes. These two designs have attractive features for networking applications with limited size especially in the routing table. However, it is difficult to implement as shared memory in multi/many-core systems.

In ([Ullah, 2017](#_3e8gvnb)), the author presented a logic-based high-performance Binary CAM (BiCAM) architecture (LH-CAM) with Xilinx FPGA. Multiple data may be written simultaneously if enough I/Os are available on the FPGA device; therefore, improving writing latency. It also provides faster updating algorithms but the complexity linearly increases with the CAM depth and hence access latency will increase linearly.

The second approach to improve memory efficiency is to use efficient cache replacement algorithms. Least Recently Used (LRU), random, Round Robin, and modified LRU are commonly used. Many other advanced strategies have been proposed, most of which are based on LRU to solve the miss rate and access latency issues and are designed for general purpose applications rather than multi-core processors ([Huang et al., 2016](#_393x0lu)). On the other hand, few of researches were touched to evaluate the multi-core system performance associated with these types of replacement algorithms. In ([Monazzah et al., 2016](#_2ne53p9)), the authors presented a Least Error Rate (LER) replacement algorithm in shared cache L2 with minimum error rate in writing. LER modifies the algorithm used to store incoming data in a cache line with minimum write error rate. To accomplish this, LER compares the incoming data with the contents of the set lines simultaneously. The experiment’s results of these algorithms were compared to LRU and show 1.4% improvements on miss rate and 3.6% less overhead. A new Random First Flash Enlargement (RFFE) had been proposed ([Naderializadeh et al., 2017](#_12jfdx2)). It enhances the overwriting on L2 shared caches when a cache line should be replaced depending on a random Gaussian-coding scheme. This replacement algorithm increases the complexity of the cache controller.

In ([Syed et al., 2017](#_gtnh0h)) , a new update algorithm was proposed; it focused on designing a high-speed intelligent update algorithm for a RAM-based TCAM because it is the main factor that affects power, performance and scalability in TCAM. The design was successfully implemented on Virtex-6 FPGA series. The results show the functional simulation of the design where the authors prove that their design consumed less latency for updating the blocks. Authors in ([Syed & Ullah, 2017](#_21od6so)) have successfully designed another updating technique based on RAM-based TCAM, which automatically adds incoming data and delete old one whenever the TCAM become full. The main disadvantage of these two algorithms is the complexity especially when increase the memory size and hence consumes large power.

Moreover, it should be emphasized that all of the previous replacement algorithms add a new overhead of accessing data and increase the non-computational times to update the location because they do not utilize cache hardware architecture.

The main function of the cache replacement algorithm is determining the effective response of the cache. Although the replacement algorithms goal is to erase the block that will not be accessed in the near future, some of the erased blocks will be accessed in the far future while executing instructions. The data written on the shared level of cache can be divided into two versions. Near-access data may be used by near coming instruction and far-accessing that will be accessed, relatively, long time after being written. The far-accessing data stored in a line could be overwritten before being used. In most cases, the controller can produce far-access operations such that the interested core/s read them from lower level and write them in their private caches ([Chauan et al., 2015](#_p49hy1); [Patterson & Hennessy, 2020](#_243i4a2)).

The goal of this work is to improve the cache access latency by designing a standalone memory that can be used in multi-core systems as a shared pipeline cache. For the pipeline processors usually dual port is used to avoid stalling during simultaneous access to the memory ([Loi & Benini, 2014](#_4ddeoix); [Seo et al., 2016](#_2sioyqq)). The proposed design is based on Dual Port CAM (DPCAM), which provides simultaneous write and search operations within the CAM memory if more than one core tries to access the memory. In addition, Near-Far Access Replacement Algorithm (NFRA) is proposed and implemented as a hardware component inside DPCAM to reduce the cost overhead and the complexity of the cache controller.

### *2.6.2 Related Routerless NoC*

NoCs pay a heavy penalty in area overhead and power consumptions due to the complex routers structure, researches revealed that routers structure in mesh or tour topology consumes about 28% of the total power and 17% of the total area in the Intel chip ([Hoskote et al., 2007](#_2qk79lc)), also, it puts additional latency in connection due to increase hardware structures that negatively affect the performance.

Because of the NoC based router problems, researchers start with the strenuous pursuit to reduce the number of routers and then produce a routerless NoC. [Daneshtalab et al. (2010)](#_3rnmrmc) proposed a new INthat connects between clusters called Cluster Mesh Inter-layer Topology (CMIT). CMIT uses 3D mesh NoC, it still uses routers for intra-layer interconnects but eliminates the need for routers between clusters. This paper illustrates the key concepts of routerless interconnection, which is utilized to reduce chip area overhead and power consumption.

[Awal et al. (2015)](#_2olpkfy) proposed a combination of 2-D mesh on multilayers NoC. It reduces the number of routers needed in the network based on multilayers chip. This network has several attractive metrics, such as diameter, network cost, and constant degree. It has moderate bisection width, fault tolerant structure and links count relative to other INs. Whilst this INhas several attractive metrics, it has a drawback of the difficulty to scale the design because this requires additional chip layers which makes it not feasible in current technology. Moreover, this architecture does not take into account the connection of the cores to shared memory.

[Li et al. (2017)](#_13qzunr) proposed Nesting Ring NoC (NRO). This topology consists of a set of clusters each with a fixed size of four cores. NRO archives attractive features in terms of performance and latency but it has an obvious problem in terms of scalability specifically as the number of cores increases to hundreds or more. This will increase the network diameter causing higher delays and increased traffic along inter-cluster paths. In ([Li et al., 2019](#_3nqndbk)) authors try to address these problems using a large-scale NRO INthat modifies NRO in two ways. First, by introducing new links between cores and clusters in order to reduce the network diameter. Second, by exploiting the advantages of multilevel chips to combine large cores on each cluster. On the other hand, this work doesn't discuss the shared memory issue between intra and inter clusters.

[Udipi et al. (2010)](#_2uxtw84) designed a new interconnection structure that eliminates the routers between segments. This architecture is based on a shorted bus and a segmented bus. The main idea is to divide the system chip into various segments of cores, with a shorted buses interconnection on each segment. Each sub-segment-bus is connected to a central bus that is directly connected with the manager core. A simple control unit, called "Filter", was implemented on each central bus to allow the data to transmit between segments. This design has a scalability problem because large numbers of cores need large links and complex control unit especially to preserve consistency for shared cache memory.

A ring topology that had discussed in section 2.5.3 is commonly used in some industry products but it still suffers from scalability problems. [Liu et al. (2015)](#_375fbgg) proposed a new architecture called Isolated Multi-Ring (IMR) that tries to connect up to 1024 cores with Multi-Ring topology. In IMR, any two cores can be connected via one or more isolated ring so that each packet can reach the destination directly by the same ring structure. This totally eliminates the need for complex routers and so improves the performance and reduces hardware cost. Authors present a set of guidelines and features to design the IMR. Then propose an algorithm to address these guidelines. Any source and destination cores should be connected using one or more rings and the data transmitted from the source to the destination is decided at the source core. IMR enhanced the throughput, latency but still has many issues like the need for a large number of rings and a large number of buffers at interfaces.

[Alazemi et al. (2018)](#_n5rssn) proposed a novel routerless architecture that exploits the buses resources perfectly to achieve the shortest path and solve the scalability problem. As the new technologies scale the chips to smaller dimensions, it supports a higher level of metal layers for integration. With this new trend, the increasing number of layers will be exploited in routing metal layers. For example, Intel Xeon Phi series are designed in 13 metal layers ([Bohnenstiehl et al., 2016](#_338fx5o)). The simulation results show that this architecture achieves a significant advantage in latency, performance, and power consumption. Whilst this idea is so promising, the speciﬁc architecture has several issues and deals with ideal many-core architecture without taking into consideration issues related to shared caches.

[Xiao et al. (2019)](#_1maplo9) proposed multi rings that provide full connectivity between each pair of cores and minimize the distance between source and destination over the system. The authors propose a flexible algorithm called the first optimal integer linear programming to design the multi rings for chip size under any given loop restrictions.

Recently, Routerless NoCs have become promising to be applied for various areas such as deep reinforcement framework. [Lin et al. (2020)](#_30tazoa) proposed a novel deep reinforcement framework, exploiting loop placement for routerless NoC as an assessment case study. The method learns and chooses the best loop placement for routerless NoCs that connect pairs of cores with different design restrictions.

# CHAPTER III

# The DPCAM Architecture

This chapter presents a special purpose shared memory architecture based on content addressable memory and a replacement algorithm. The main purpose of this design is to allow simultaneous access to the cache memory by multicore processors that achieve more efficient access latency with various CAM cache sizes compared to the SA cache.

In the following sections a detailed description of DPCAM architecture, design analysis and simulation results, is given. In section 3.2, the types of shared memory in computer architecture are discussed. In section 3.3, the architecture of the proposed DPCAM and the Near-Far Access NFRA is presented. ​In section 3.4, the implementation of the DPCAM using FPGA, the functional and timing simulation and the power estimation analysis are shown.

## 3.1 Introduction

The microprocessor that contains multiple cores (processors) in a single integrated circuit (IC) is named as multicore. Many-core systems are multicore systems designed for a huge degree of parallel processing containing many cores. In multi/many core systems, the shared memory has the key role in providing efficient communication between the cores. When multiple cores try to access the same shared memory at the same time, it may cause a hazard. There are many studies in the literature to improve the performance of shared memory. Reducing access latency and power consumption are the main directions to improve the efficiency of the shared memory. Improvements in cache architecture and cache replacement algorithms are two options to pursue in this direction.

Most multi/many core systems use AM cache as a shared memory ([Patterson & Hennessy, 2020](#_243i4a2); [Stallings, 2013](#_1vsw3ci)). Enhanced cache architecture aims to empower parallel search and rapid access ([Karam et al., 2015](#_1l354xk)). On the other hand, replacement algorithms are used to help the cache controller choose which data to discard to make room for the new ones ([Olanrewaju et al., 2016](#_452snld); [Priya et al., 2019](#_2k82xt6)). In addition, an efficient replacement algorithm will improve the cache access latency. CAM is a type of AM that refers to memory whose locations are reached by comparing tags (part of contents) instead of producing their addresses, and has some features to be used as a shared memory ([Abumwais & Ayyad, 2018](#_j8sehv); [Irfan, Ullah, et al., 2019](#_488uthg); [Karam et al., 2015](#_1l354xk)).

## 3.2 Types of Shared Cache Memory

Traditional memory architectures, both Static Random Access Memory (SRAM) and Dynamic RAM (DRAM) have a unique address for each location that is used to retrieve/store the data. On the other side, content-operated memory (COM) is another kind of memory which is built to access data in a different way. It can access the stored data by part of its contents instead of addresses ([Patterson & Hennessy, 2020](#_243i4a2)). Like any other memory, COM has two essential processes: write operation (store data) and read operation (access the correct match data). COM is used in various applications in digital computer systems, simply as a part of branch prediction techniques, and in very-high speed parallel systems to concurrently search/store data where the address is not available ([Karam et al., 2015](#_1l354xk)). Moreover, packet switching routing and classification on network systems is the major application of COMs ([Cheriton, 2015](#_1yib0wl)). It is expected for COM memory to take its turn in emerging applications for non-Complementary Metal Oxide Semiconductor (CMOS) next-generation electronics devices([Karam et al., 2015](#_1l354xk)). COM memory architectures can be classified into two main types, AM and CAM and they perform the same functions but in different ways.

There are three kinds of AM memories that are all used to cache data in the computer system, but they differ from each other by the restrictions on where the data should be written. Three forms namely, direct-mapped (DM), SA, and fully associative (FA), differ from each other by the restrictions on data storage locations and the used replacement algorithms. With direct-mapped, only one location is possible for the particular data. FA data can be mapped to any location and SA allows a set of possible locations for data to be stored. In the following subsections a brief description of these three main types of cache memory.

### *3.2.1 Fully Associative Memory*

In FA cache memory, the address and data are stored in the cache location. The incoming address is compared with all stored addresses in each caches location. Figure 3.1 depicts the fully associative cache architecture.

The main advantage of this type of cache is its high performance compared to its size. The design complexity is the main drawback of FA cache. In addition, it requires a replacement algorithm to determine where data should be stored. Random, First in First out (FIFO), and the Least Recently Used (LRU) algorithms were used to tackle this problem ([Stallings, 2013](#_1vsw3ci)).

However, FA caches are rarely used in multicore processors because it has a lower cache hit rate. Each time a new memory is referenced to the same cache location, the cache line will be replaced, which increases the miss rate ([Patterson & Hennessy, 2020](#_243i4a2); [Stallings, 2013](#_1vsw3ci)).

Figure 3.1

*Fully Associative Architecture*

Generate Address from core

Address Data

Cache

Main Memory

Main Memory accessed if address not in cache

Access miss

Compare with all addresses simultaneously

Access each location by comparing address

Access hit

### *3.2.2 Direct Mapped Memory*

In this type, the main memory is split into blocks and the cache is split into a set of lines. Therefore, in each cache line, one block of the main memory can be hold. In this architecture instead of storing total address information in the address field, only a part of address bits is stored along with data field ([Patterson & Hennessy, 2020](#_243i4a2); [Stallings, 2013](#_1vsw3ci)).The main advantages of direct mapped are that it is simple and inexpensive to implement, although performance will suffer if access to different locations with the same index are done.

Figure 3.2

*Direct Mapped Cache Memory*

From core Address Index

Address Data

Cache

Main Memory

To main memory in case of not match address

Access miss

Access location

Address and index

Different

Same

Compare

### *3.2.3 Set-associative Memory*

Set associative is a hybrid cache that combines full associative and direct mapped caches. The Cache is split into a set of lines. One block of main memory can be stored in n potential sets of cache. It is not quite as complicated as a fully associative cache. Set-associative improves the performance because more than one address can be stored under the same index. The main disadvantage of set-associative cache is very expensive as the set size increases and the access latency increase as it needs to compare each address in all sets after generating its index. Anyway, most commercial multi-core systems are used set-associative caches because of better performance ([Patterson & Hennessy, 2020](#_243i4a2); [Stallings, 2013](#_1vsw3ci)). Figure 3.3 represents set associative cache memory.

In fully associative mapping and set-associative, there is multiple choice for selecting locations for data storage so replacement algorithms are needed.

Figure 3.3

*Set Associative Cache Memory*

Address Index

Address Data

Cache

Address from PEs

Address Data

Address Data

Address match

To main memory if miss occurs

Main Memory

Address and index

### *3.2.4 Content-addressable Memory (CAM)*

CAM refers to memory whose locations are reached by comparing tags (part of contents) instead of supplying their addresses. In some aspects, CAM is identical to DM on AM form; both permit instant retrieval of an output specified by the input. Nevertheless, both DM and CAM use different techniques to empower the parallel search and rapid storage ([Karam et al., 2015](#_1l354xk); [Stallings, 2013](#_1vsw3ci)). DM restricts storing particular data in only one location; however, CAM has no limitations on where to write data.

On the other hand, both CAM and FA are identical not only in terms of no restriction on where to store data. Nevertheless, they also use similar update and replacement policies such as random, FIFO and LRU in order to overwrite data when memory becomes full or the data become un-useful. These algorithms should choose a line which is not likely to be wanted in the near future, from all the lines on memory ([Priya et al., 2019](#_2k82xt6)).

Another new emerging memory associated with CAM is called Transactional Memory (TM). It does not much differ from CAM, but it is used to allow sharing data between processors in a distributed system. This type is actually used as the autonomous storage memory with various hardware components ([Nakaike et al., 2015](#_3ws6mnt); [Papagiannopoulou et al., 2018](#_2bxgwvm)).

Generally, the main function of the CAM memory is to search through the memory contents. This is done by simultaneously searching the content of data rather than its address; which is completely different from traditional SRAM. Due to the power in parallel and simultaneous search of CAMs, various kinds of CAMs are used in a wide range of applications such as image processing, signal processing, pattern recognition, a part of switching network techniques, and in some parallel processing systems ([Bhattacharya et al., 2014](#_r2r73f); [Imani et al., 2019](#_3b2epr8); [Karam et al., 2015](#_1l354xk); [Martyshkin et al., 2018](#_1q7ozz1); [Prasanna Venkatesan](#_u8tczi)).

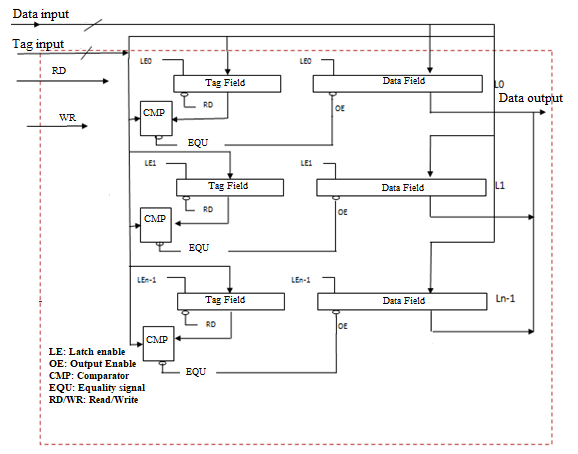
Each data is assigned with a unique tag in CAM. To read the data, the read signal and tag are applied to all locations at the same time. The applied tag is simultaneously compared with all tags that were previously stored. If the match occurs, the data in the matched location is selected, and output on the data bus in order to be read by the core. Figure 3.4 shows the CAM architecture with a single port.

Universal CAMs that have been designed for a particular hardware are made using specific designed cells, but CAM was not possible to be designed as a standalone memory in any system because very large number of pins were needed. Recently, the tremendous development in semiconductor technology and in FPGAs had attracted researchers for implementing CAM in FPGAs ([Irfan, Ullah, et al., 2019](#_488uthg); [Ullah et al., 2018](#_4a7cimu)). These types of memory improve the search rate and reduce the processing latency and sometimes the power consumption.

The SA cache is the most popular architecture type used as shared memory in multi-core systems ([Karam et al., 2015](#_1l354xk); [Patterson & Hennessy, 2020](#_243i4a2)). It still suffers from many problems such as increasing access latency and contention if more than one core tries to access the same shared memory simultaneously. These problems is solved using the proposed DPCAM architecture.

Figure 3.4

*CAM Architecture*



## 3.3 Proposed DPCAM

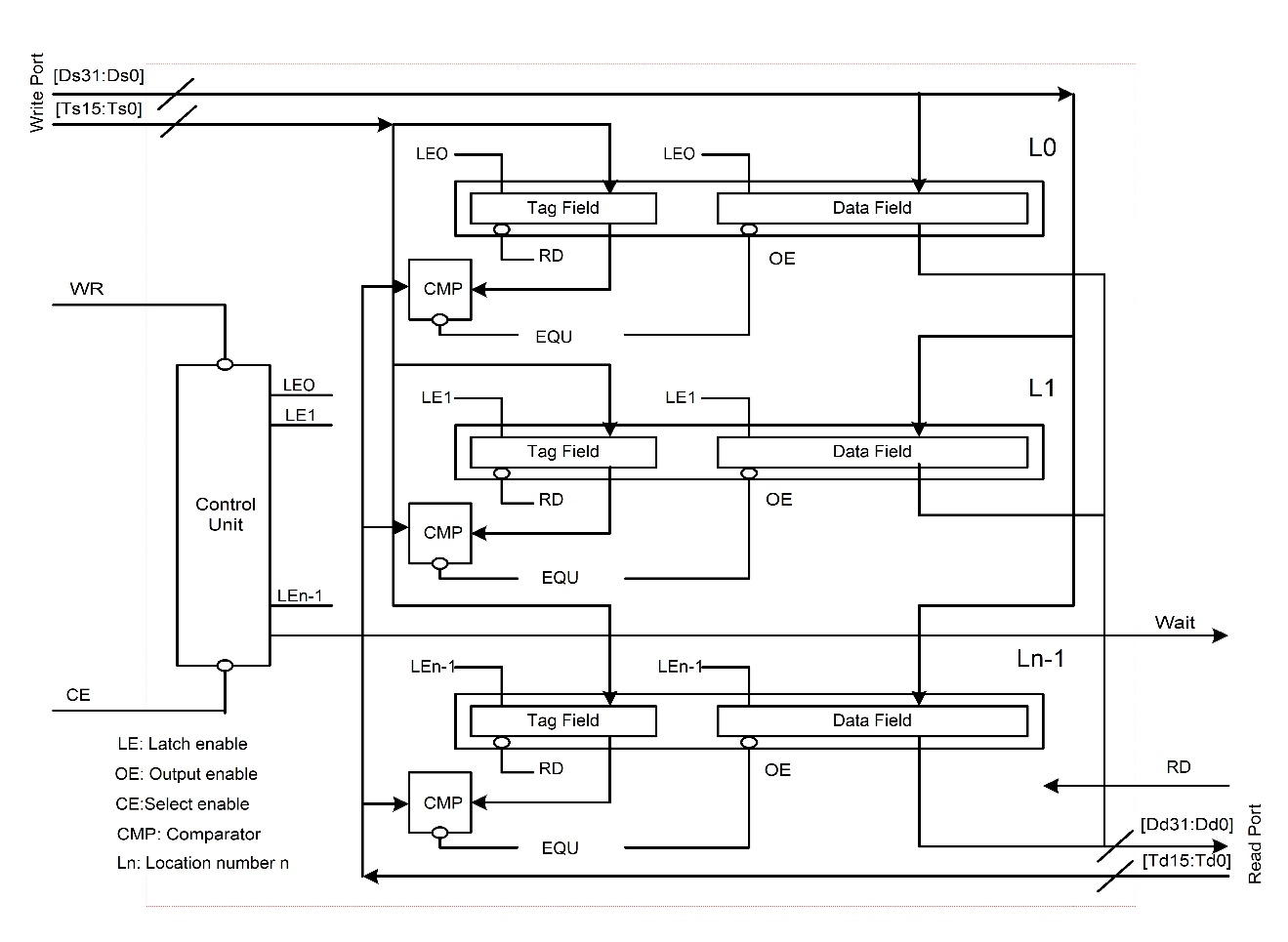
The proposed DPCAM works as a standalone pipelined shared cache where one port is used for writing and the other one is used for reading. The DPCAM includes dual port: DS31-DS0 for writing and Dd31-Dd0 for reading, Tag Field, Data Field, Control Unit (CU), and comparator (CMP) as shown in Fig 3.5.

In the stage of Store Back unit (SB) the core provides the Data source [Ds31-Ds0] and the Tag source [Ts15-Ts0] to be written on a selected cache line. In the stage of Operand Fetch unit (OF) the core provides the Tag destination [Td15-Td0] to be compared with all cache lines simultaneously and the stored data will be read to the destination data bus [Dd31-Dd0].These two ports work concurrently.

Each cache line (L) is consists of two fields: the Data Field and the Tag Field and is associated with a simple 2X1 comparator (CMP). Data Field contains the shared data to be stored while the Tag Field contains a unique tag (a part of data plus version number) for each Data Field. Each field’s length is determined by the architecture in which the CAM is used. The tag field can be diverse to fit the size of shared unique data versions, e.g., 24-bit tag can accommodate up to 16 Mega versions of shared data. The CMP is needed in reading operations. It compares the tag coming from the OF stage [Td15-Td0] with tags stored [Ts15-Ts0] in cache lines.

Figure 3.5

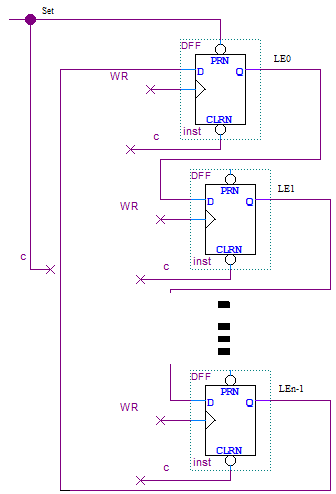
*DPCAM Design*

****

Another main component of the DPCAM architecture is the (CU). It has two functions, controlling the write operation and implementing the replacement algorithm. The CU has a pointer to produce an active signal for each cache line on a rotating basis. Fig. 3.6 depicts the architecture of CU. The control circuit can be employed to select the location where data will be stored where locations are selected sequentially for writing with simple overwriting techniques to update the contents and erase the old one. A set of D Flip Flop (D-FF) is used that is equal to the number of DPCAM lines, the output of each D-FF points to the corresponding DPCAM location. When the system is reset, this pointer points to the LE0 first line, so that the first writing operation will take place on L0. After writing to the current line, the pointer points to the next line, and so on until line n-1 (Ln-1) is reached.

Figure 3.6

*Control Unit (CU)*



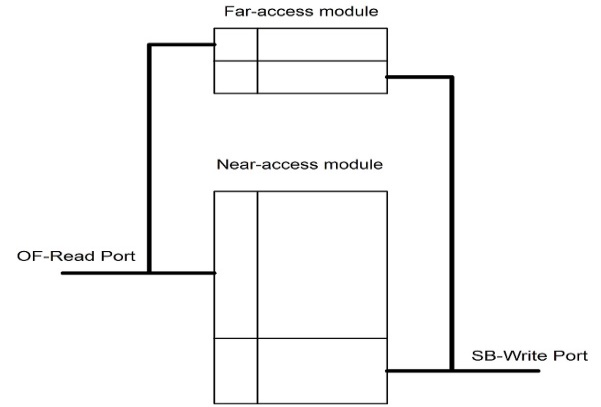
The writing operation in DPCAM is managed by the CU and the Write (WR) signal on the first port (write port). The stage SB unit provides the source data [Ds31-Ds0], the source tag [Ts15-Ts0] and the active low WR signal. On the negative edge of WR signal (the end of WR), in preparation for the next writing, which will be to line 1, the control circuit shifts the LE to LE1. The read operation occurs when the OF unit of the reading core applies the tag destination [Td15-Td0] and an active high read (RD) signal to all Tag Fields simultaneously. The RD signal outputs the stored source tags to the CMP of each memory line in order to be compared with the applied tag simultaneously. If a match occurs, the equality signal of the comparator is used as an output enable (OE) signal which outputs the stored data from the Data Field to the destination data bus [Dd31-Dd0] where it can be read by the OF unit of the reading core. In case of reading and writing to the same memory location, the CU will give priority for writing and will give a WAIT signal to the reading operation. In case of reading and writing to the different memory locations simultaneously, both read and write ports operate concurrently, which significantly reduces a cache access latency. The SB unit of the writing core provides the data [Ds31-Ds0] and the tag [Ts15-Ts0] which are written to the location determined by the CU. On the other hand, the OF unit of the reading core provides the destination tag [Td15-Td0] and RD signal to all tag fields simultaneously and the stored data can be read to the destination data bus [Dd31-Dd0].

In the DPCAM, a replacement algorithm called NFRA based on simple hardware units is implemented. In the proposed architecture, as an alternative solution to reading the data from lower level memory by the cache controller which increases the access latency, a new small DPCAM is implemented. The main DPCAM contains the near-access data, while the new module is used for the far-access data, as shown in Fig. 3.7. Certainly, because the modules of the far-accessing are less frequently used, it will be smaller than those of the near-access. For example, with four cores and 64 Kilobyte (KiB) shared DPCAM, each core can write 2k operand, each includes eight bytes of data and tag, to the DPCAM before it needs to be overwritten.

Referring to the CU and writing through the pointer which is the main part of the NFRA, if the processor writes to location Lx, the next instruction will write its operand on location Lx+1. This procedure can be continued until location Ln-1 is accessed, after then, it returns to LE0, where it starts overwriting the old data and tags. This technique is applied in both near-access and far-access modules. It can be noted, that the NFRA can be implemented on the hardware level with less cost and less access overhead in compiler computation. The cost overhead is mostly related to the cache controller complexity and its latency. As a new solution, the proposed NFRA is implemented using simple hardware inside DPCAM control unit instead of a complex algorithm installed inside the cache controller, thus NFRA improves the cache access latency. Using an algorithm at compile time allows separating the near-access and far-access to be stored/loaded in/from different DPCAM modules ([Patterson & Hennessy, 2020](#_243i4a2)). The far-access module works on demand. In other words, it gets activated if any core needs to access it for storing/loading data. Other than that blocks are transferred to a new inactive mode. This is a well-known concept in caches called the migration principle to save the power consumption ([Luo et al., 2019](#_14hx32g); [Ofori-Attah et al., 2017](#_3ohklq9)).

Figure 3.7

*Near-access, Far-access DPCAM modules*



## 3.4 Implementation of the DPCAM and Performance Analysis

DPCAM has been implemented, compiled, simulated, and verified using Quartus prime 19.1 including ModelSim package for design and simulation supported by Intel . DPCAM has been built and evaluated as a standalone memory using Intel FPGA family Cyclone V with 28 nm technology ([Intel, 2018](#_is565v)). This is the first step to demonstrate that DPCAM can replace the shared cache in the memory hierarchy of a multi-core processor. For testing DPCAM two cores were used to assess the latency of read and write operations. DPCAM has been implemented by both block schematic files and Verilog Hardware Description Language (VHDL) code. Files have then been verified and debugged using ModelSim and Vector Wav File (VWF) in both functional and timing simulation. A special tests-bench was written to simulate and observe the latency of reading and writing operations of DPCAM. In addition, the Power Analyzer Tool has been used to estimate the static and dynamic power consumption of DPCAM. The performance of DPCAM was compared to the SA cache which is the most popular architecture type used as shared memory in multi-core systems ([Patterson & Hennessy, 2020](#_243i4a2); [Stallings, 2013](#_1vsw3ci)). All the schematic, Verilog code, test-bench programs, functional simulation, latency assessments and power estimations are presented in appendix B.

### *3.4.1 Functional Simulation*

During the functional simulation the following tests to assess the performance of DPCAM were accomplished: write and read operations, the simultaneous read and write operations into the different memory location and the simultaneous read and write operations into the same memory location.

The test-bench program starts by resetting the CU. It then generates random 16-bit tags and 32-bit data and puts them on the input pins to perform the write operation. After that, it keeps generating repeated read/write signals until the end of simulation time. It finally puts the 16-bit tagd to the input pins in order to compare with stored tags and output the data to the output pins to perform the read operation. The test-bench is used for functional simulation in DPCAM. Moreover, it is used for DPCAM which uses NFRA and SA cache which uses LRU replacement algorithm in order to compare them in terms of latency and power dissipation. The diagram in Fig. 3.8 illustrates the use of the test-bench program, which is a special benchmark program that was written for analyzing the latency and power dissipation assessments for all read and write cases on both DPCAM and SA architecture.

Running the simulator several times it was noticed that the main functions for DPCAM architecture in terms of reading, writing, simultaneous read-write, CU and replacement algorithm are accomplished.

Fig. 3.9 shows a thumbnail image of several clock cycles (clock period equal 10 nano-second (ns)) for reading and writing to the 64 KiB DPCAM. In the first interval (0 to 10 ns), the CU was set to start pointing on the first location. Instantly when the WR signal goes down, because it operates on negative edge, the written data (outI) appear clearly on the DPCAM locations. This means that the input values have been stored in the targeted DPCAM locations. In the second interval (10 to 20 ns), the processor retrieves the data that is stored in DPCAM location, it loads the corresponding tag (tagd) of the data written before (outI) and applies a RD signal. Instantly when the RD signal becomes high, the stored data appears on the output buses of the processor (outE). Interval 4 (30 to 40 ns) displays the read and write operations simultaneously in different DPCAM locations, where new data with tags ([0]13) is written to the target location and data that already have been written with tagd ([0]12) is read correctly. While interval 5 (40 to 50 ns) shows the simultaneous read and write operations into the same location with the higher priority of a write operation and delayed read to the next cycle.

Figure 3.8

*Test-bench Program*

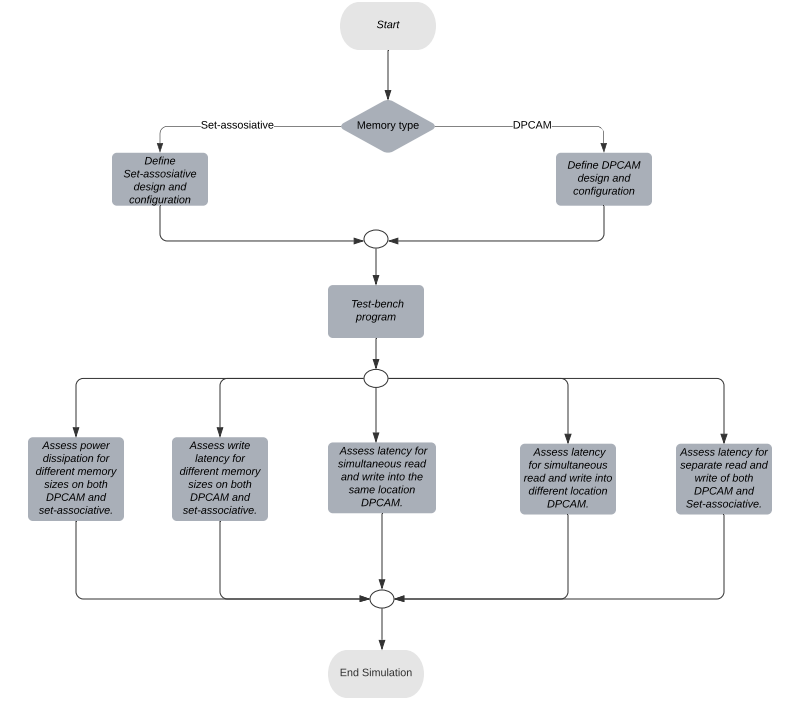
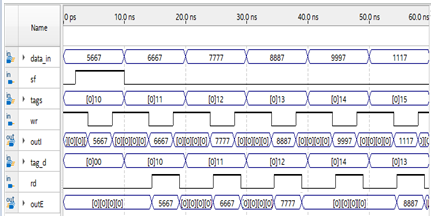


Figure 3.9

*Functional Simulation*



### *3.4.2 Latency Assessments*

In this part, the Timing Analyzer tool in Intel Quartus prim is used to evaluate DPCAM performance in terms of timing assessments. The Intel® Quartus® Prime Timing Analyzer uses industry-standard constraint and analysis methodology to report on all data required times, data arrival times, and clock arrival times for all components in your design ([Intel, 2021](#_41wqhpa)). The Timing Analyzer was used to assess the access latency for reads and writes operations in DPCAM architecture and confirms actual signal arrival times against the constraints that specify by the design.

Fig. 3.10 illustrates the timing simulation of both reads and writes operations of the 64 KiB with the near and far DPCAM module using Cyclone V FPGA from Intel.

In the first interval (0 to 10 ns), the CU was set to input data and their tags into the first location. Instantly when the WR signal goes down, the written data (pine outI) appear clearly on the DPCAM locations after a time delay. Running the simulator 100 times it was noticed that the average delay time of writing on DPCAM is about 0.9529±0.03393 ns. The second (10 to 20 ns) and third (20 to 30 ns) intervals show the simulation to assess a latency of a read operation. To read data that is already stored in any DPCAM locations, the tagd ([0]10) in the second interval is compared simultaneously to the tags in all locations with RD signal. The equality occurs when comparing with the tag associated with data in the first location and the data appears on output buses (outE) after delay time. Taking the average of around one hundred intervals of test-benches, it was noticed that the delay for read operation is around 1.1782±0.08830ns. The fourth (30 to 40 ns) and fifth (40 to 50 ns) intervals were used for an assessment of latencies for the simultaneous read and write operations to the memory locations.

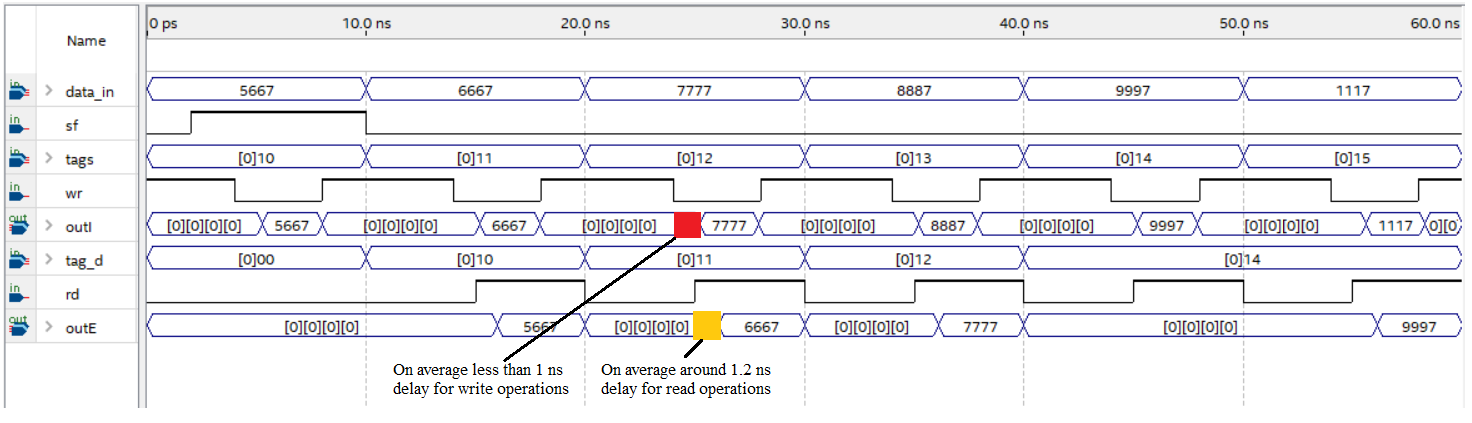
Memory latency is the time between initiating a request for data and the actual data transfer. For simultaneous write and read operations of DPCAM the memory latency is the time between two requests for simultaneous write and read operations. To assess a latency for these modes the following expressions can be used:

* for simultaneous write and read operations to the different memory locations
* for simultaneous write and read operations to the same memory locations

Where is a latency of a write operation, is a latency of a read operation and is a cycle time.

Figure 3.10

*Latency Assessments*



The simulations for both modes were performed 100 times as well. For the mode with simultaneous write and read operations to the different memory locations ns which indicates that the latency is equal to the latency of the read operation measured before in the second interval of tests. It was proved by using T-test with 95% of a confidence interval.

For the mode with simultaneous write and read operations to the same memory locations the data is written to the target location with a latency 0.9828±0.0412 ns, whereas the read operation waits until the next cycle then targeted data is read from the same location with a latency 1.2226±0.09446, that is common latency for the mode ns.

The same value of test-bench is used with the 64 KiB four-way SA cache to assess the latency of different operations for different cases of memory access. The experiments that were done 100 times showed that the average latency of a write operation is 1.9434±0.0382 ns while for a read operation it is 2.1584±0.1056 ns. Simultaneous read and write operations were not tested because it is not allowed in SA caches.

The read latency of the tested DPCAM is less than that of the tested SA cache because SA caches require an index to be determined to access a location that has a tag to be compared with tag part of the target address which increases the latency. Whereas in DPCAM, the incoming tag is directly compared with the stored tag. Usually, the cache memory based on AM has around 2 ns read latency with the 64 KiB ([Cargnini et al., 2014](#_2h20rx3)), 1.66 ns in AM with 1KiB, and 1.69 ns in 4-way set associative with 2 KiB which is used in cache controller ([Chauan et al., 2015](#_p49hy1)). But write latency for the cache memory based on AM usually exceeds 2 ns for 64KiB ([Cargnini et al., 2014](#_2h20rx3)).

*Table 3.1.*

*Latency Of A Write Operation (ns)*

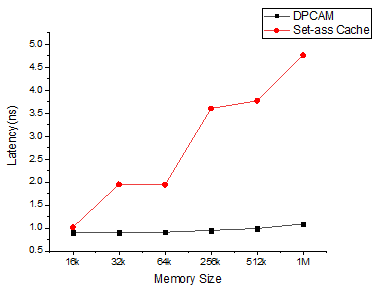
| Size | DPCAM | Cache | Size | DPCAM | Cache |
| --- | --- | --- | --- | --- | --- |
| 16k | 0.90 | 1.02 | 256k | 0.95 | 3.60 |
| 32k | 0.90 | 1.95 | 512k | 0.99 | 3.77 |
| 64k | 0.91 | 1.94 | 1M | 1.09 | 4.76 |

Latency of a write operation is a critical issue and it can jeopardize the adoption of any design in the multi-core memory hierarchy. DPCAM design with near-far access modules and various memory sizes was simulated and compared with the traditional four-way SAcache of equivalent sizes with the same FPGA technology. To find the average latency of a write operation using various memory sizes, the memory size was modified and the test-bench was accomplished with a new time interval for each size. Fig. 3.11 and Tab. 3.1, show that DPCAM has a small writes latency that is nearly constant for different sizes; this is because the CU directly points to the memory location. In this case, there is no need for generating the address to the next write location which is used in AM cache memory and makes it faster to select the appropriate location for a write operation. Fig. 3.11 illustrates that the gap of write latency between DPCAM and the SA cache is increasing as the size of memory increased.

According to the latency estimation of write and read operations, the NFRA replacement algorithm used by DPCAM was compared with LRU algorithm used by the set associative cache memory. For size 64 KiB DPCAM achieves less access latency. For DPCAM the write operation is about 0.9529±0.03393 ns and the read operation is 1.1782±0.08830ns, whereas, for the SA cache the latency of the write operation is 1.9434±0.0382 ns while the read operation is 2.1584±0.1056 ns.

Figure 3.11

*Latency Of A Write Operation (ns)*



#### 3.4.2.1 Descriptive Statistics. The simulator was run around one hundred times with different values of test-bench. Each time the write and read latency, the simultaneous read and write operations into the different memory location latency and the simultaneous read and write operations into the same memory location latency were recorded. In order to analyse these latency values which are recorded in different cases, SPSS and T-test tools were used.

SPSS was used to find the minimum, maximum, mean and statistic error for DPCAM and SA architecture. Table 3.2 shows descriptive statistics for write latency in DPCAM. Table 3.3 shows descriptive statistics for read latency in DPCAM. Table 3.4 shows descriptive statistics for the simultaneous read and write operations into the different memory location latency in DPCAM and Table 3.5 shows descriptive statistics for the simultaneous read and write operations into the same memory location latency in DPCAM. Table 3.6 shows descriptive statistics for write latency in set-associative. Table 3.7 shows descriptive statistics for read latency in set-associative.

For comparisons of the write operation and the read operation for DPCAM and a SA memory the T-test was used which showed that there is an evidence that the write and the read latencies of DPCAM less than the same latencies of SA with 95% of the confidence interval. Other t-test statistics are shown in appendix B.

*Table 3.2.*

*Descriptive Statistics For Write Latency In DPCAM*

|  | N | Minimum | Maximum | Mean | Std. | |
| --- | --- | --- | --- | --- | --- | --- |
| Statistic | Statistic | Statistic | Statistic | Std. Error | Statistic |
| Write DPCAM | 100  100 | .89 | 1.10 | .9529 | .00339 | .03393 |

*Table 3.3.*

*Descriptive Statistics For Read Latency In DPCAM*

|  | N | Minimum | Maximum | Mean | Std. | |
| --- | --- | --- | --- | --- | --- | --- |
| Statistic | Statistic | Statistic | Statistic | Std. Error | Statistic |
| Read DPCAM | 100  100 | 1 | 1.35 | 1.1782 | .00883 | .0883 |

*Table 3.4.*

*Descriptive Statistics For The Simultaneous Read And Write Operations Into The Same Memory Location Latency In DPCAM*

|  | N | Minimum | Maximum | Mean | Std. | |
| --- | --- | --- | --- | --- | --- | --- |
| Statistic | Statistic | Statistic | Statistic | Std. Error | Statistic |
| Write/read  Different location DPCAM | 100  100 | 1 | 1.40 | 1.2262 | .00945 | .0945 |

*Table 3.5.*

*Descriptive Statistics For The Simultaneous Read And Write Operations Into The Same Memory Location Latency In DPCAM*

|  | N | Minimum | Maximum | Mean | Std. | |
| --- | --- | --- | --- | --- | --- | --- |
| Statistic | Statistic | Statistic | Statistic | Std. Error | Statistic |
| Write/read  same location DPCAM | 100  100 | .91 | 1.10 | .9828 | .00412 | .0412 |

*Table 3.6.*

*Descriptive Statistics For Write Latency In Set-associative.*

|  | N | Minimum | Maximum | Mean | Std. | |
| --- | --- | --- | --- | --- | --- | --- |
| Statistic | Statistic | Statistic | Statistic | Std. Error | Statistic |
| Write Set-associative | 100  100 | 1.88 | 2.10 | 1.9434 | .00382 | .0382 |

*Table 3.7.*

*Descriptive Statistics For Read Latency In Set-associative.*

|  | N | Minimum | Maximum | Mean | Std. | |
| --- | --- | --- | --- | --- | --- | --- |
| Statistic | Statistic | Statistic | Statistic | Std. Error | Statistic |
| Read Set-associative | 100  100 | 1.95 | 2.35 | 2.1584 | .01056 | .1056 |

### *3.4.3 Estimation of a Power Dissipation*

Despite the recent trends towards smaller and faster memories, power management has become increasingly important. As the chip technology size shrinks, the overall size, performance and cost will improve, but the power density will increase. Hence, power dissipation estimation is essential to guide architects to define the components that consume main power and try to modify and improve the design. For estimation of power dissipation the Power Analyzer Tool of a Quartus simulator is used that provides an average accuracy of ±10% ([Quartus, 2015](#_3g6yksp)). The script with DPCAM design provided by ModelSim was run out to generate a file to be used in Power Analyzer Tool. Static, dynamic, (Input/Output) I/O and total power are calculated in accordance with waveform file generated by the Power Analyzer during the gate level simulation.

In this section, the power dissipation is compared between DPCAM and four-way SA cache with different memory sizes. For an assessment of the power dissipation, the DPCAM includes near-far access modules.

Static power is the thermal power consumed on a chip. Except for the I/O port, static power always includes the leakage power of the functional unit on the FPGA. Whereas, the dynamic power is the additional power consumption of the device due to the unit's activity and signal toggling. On the other hand, I/O power is generated by the pins. Pins always drive components off-chip or on-chip, that effect on dynamic power ([Quartus, 2015](#_3g6yksp)).

Table 3.8 and Table 3.9 illustrate the static, dynamic, I/O and total power dissipation of DPCAMs and SA cache respectively for different sizes. Figure 3.12 allows comparing the static, dynamic and I/O power dissipation between DPCAM and SA cache. Figure 3.12a shows the comparison of the static power dissipation. It is observed that the main drawback of DPCAM is the static power dissipation especially when the size increases; this is because the complexity of hardware produced by the CU, which is a part of DPCAM architecture and the internal wires cover more area and increase the power dissipation. Figure 3.12b shows the comparison of the dynamic power dissipation. It is observed that the dynamic power of DPCAM is close to the SA when the size is less than 512K, but dramatically increased after 256K; this is due to a large number of locations is active at the same time, especially in the read operations. Figure 3.12c illustrates the comparison of the I/O power dissipation. It is observed that the I/O power of DPCAM is very close to the SA with different sizes; this is because the off-chip pins are constant regardless of the internal memory size.

Figure 3.13 shows that the total power consumed by DPCAM is a little greater than the SA cache, it was estimated at about 7% of total power. Since the increase in total power is small and acceptable, it will not affect the adoption of DPCAM in multi-core systems. This slight increase in power dissipation is acceptable and can be improved using different algorithms of power-saving techniques ([Adegbija & Gordon-Ross, 2017](#_1vc8v0i); [Luo et al., 2019](#_14hx32g); [Ofori-Attah et al., 2017](#_3ohklq9); [Park et al., 2019](#_4fbwdob); [Rossi et al., 2017](#_2uh6nw4)) that are mainly used to improve static and dynamic power dissipation. Therefore, this problem will not affect the adoption of DPCAM in multi-core systems.

*Table3.8.*

*DPCAM Power Dissipation*

|  | Power in Milliwatts (mW) | | | |
| --- | --- | --- | --- | --- |
| Size | Static | Dynamic | I/O | Total |
| 16k | 32.214 | 1.13 | 11 | 44.344 |
| 32k | 64.33 | 2.14 | 11.21 | 77.68 |
| 64k | 107.57 | 2.99 | 11.10 | 121.66 |
| 256k | 349.5 | 7.98 | 11.88 | 369.28 |
| 512k | 796.2 | 22.90 | 12.07 | 831.17 |
| 1M | 1411.10 | 39.26 | 13.21 | 1463.57 |

*Table 3.9.*

|  | Power in (mW) | | | |
| --- | --- | --- | --- | --- |
| Size | Static | Dynamic | I/O | Total |
| 16k | 28.9166 | 1.12 | 10.1 | 40.1366 |
| 32k | 57.33 | 1.62 | 10.1 | 69.05 |
| 64k | 99.41 | 2.79 | 10.6 | 112.8 |
| 256k | 334.750 | 5.48 | 11 | 351.23 |
| 512k | 696.261 | 10.021 | 11.025 | 771.307 |
| 1M | 1325.310 | 19.28 | 11.737 | 1356.326 |

*SA Memory Power Dissipation*

Figure 3.12

*Power Dissipation With Variation In Memory Size: In a) It Is Compared The Static Power Dissipation Between DPCAM And Set-associative. In b) It Is Compared Dynamic Power Dissipation. In c) It Is Compared I/O Power Dissipation*

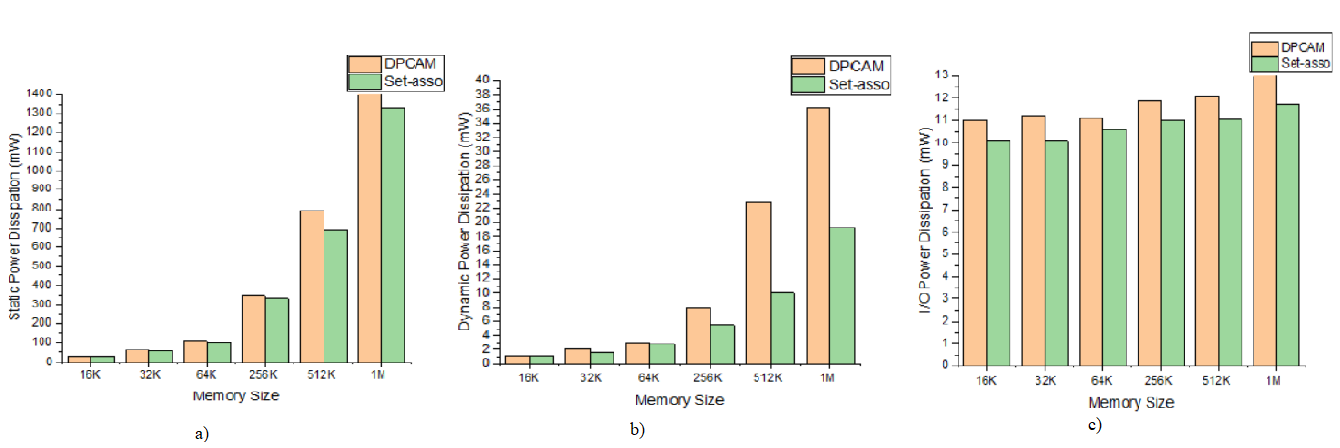
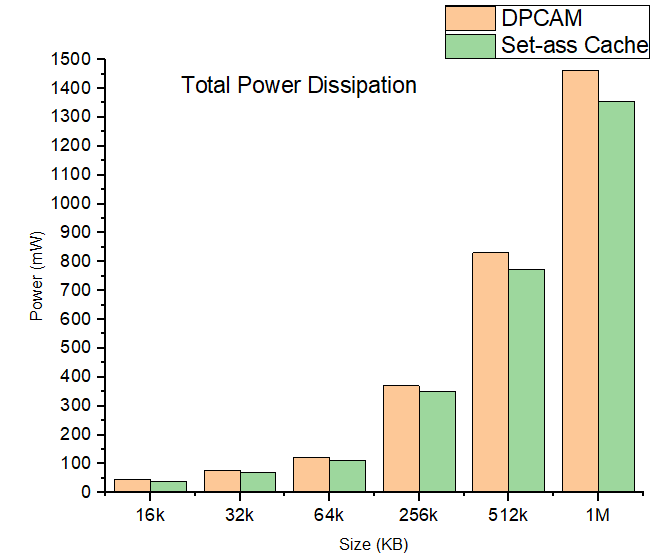


Figure 3.13

*Total Power Dissipation With Variation In Size*

****

# CHAPTER IV

# Shared Cache Based on DPCAM in a Multicore Architecture

In this chapter, the DPCAM is embedded inside the multicore system as an L2 shared cache and the multi-core performance is tracked in terms of throughput and access miss/hit rate. All results are compared to traditional multicore systems that use SA in the L2 cache. In the following section, a detailed description of DPCAM within a multi-core architecture and the simulation results analysis are presented. In section 4.2, the architecture of DPCAM inside multicore system is briefly discussed. In section 4.3, a modified replacement policy for DPCAM inside multicore system is given. ​In section 4.4, the implementation of the DPCAM within a multicore system and the performance analysis are presented.

## 4.1. Introduction

Nowadays, multicore architecture is widely used in processor design, and the cache hierarchy is moved from one level cache to multi-level cache in the modern multi-core system ([Pan & Pai, 2015](#_nwp17c); [Patterson & Hennessy, 2020](#_243i4a2); [Stallings, 2013](#_1vsw3ci)). Cache L1 is usually private for each core while other levels are either private or shared. Within a multi-core system, the INs connect cores to the shared level of caches (level 2 or level 3 in most systems). Therefore, shared cache architecture affects overall system performance. This work adopted two levels of the cache hierarchy, private L1 cache and shared L2 cache, to improve cores communication ([Mars et al., 2016](#_37wcjv5)).

As the cores count increases, the contention on the shared cache between cores becomes more intense because only one core can access the shared cache simultaneously. In addition, there are also some other problems that need to be improved. First, a quick replacement of data that is never accessed can occur due to the limited shared cache space architecture and competition between cores to access cache lines. This causes shared data access failure and requires, in most cases, reading data from lower level memory ([Patterson & Hennessy, 2020](#_243i4a2)). As a result, the system performance will be decreased ([Das & Kapoor, 2016](#_1n1mu2y)). Second, In addition to cache architecture, the cache replacement algorithm also has the main function of determining the effective response of the cache.

In chapter three, a special purpose shared memory architecture based on CAM called DPCAM was presented. In addition, an efficient replacement algorithm, called NFRA, which is based on hardware rather than software executed by the cache controller was also presented. The main purpose of the DPCAM and NFRA design is to allow simultaneous access and achieve less access latency to the shared memory. The DPCAM design was implemented on cyclone V FPGA family as a standalone memory, and then the performance related to power consumption and access latency was estimated. However, evaluation of the DPCAM as a standalone memory could not reflect its performance within a multi-core system. So, the performance of DPCAM inside multicore system should be evaluated.

## 4.2 DPCAM Inside Multicore System Architecture

### *4.2.1 DPCAM Architecture*

CAM is a memory that can be accessed by comparing content of data instead of supplying their addresses ([Karam et al., 2015](#_1l354xk)). However, previous FPGA technology did not allow large CAMs to be implemented on a chip. Small memories (up to 32 locations CAMs) were available. Therefore, architects resorted to utilize RAMs based lookup tables in order to simulate the CAMs. With the advent of FPGAs technology, large CAMs are available now (an order of hundred Kbytes CAMs).

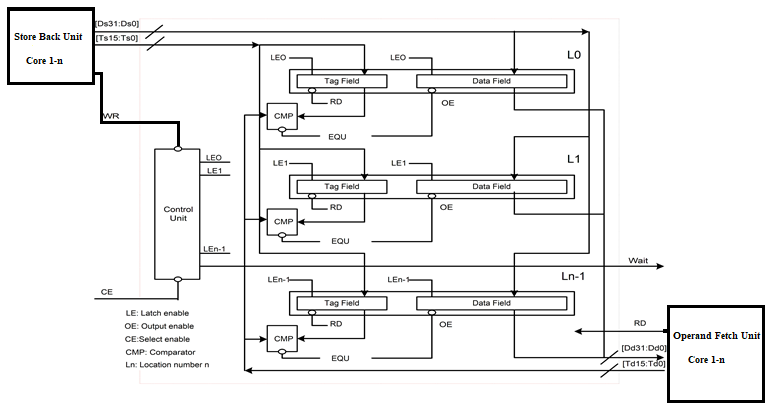
To the best of our knowledge, DPCAM that was proposed in ([Abumwais et al., 2022](#_11bux6d)) is the first work that addresses using the content addressable memory as a shared cache and taking advantage of its features. Figure 4.1 shows DPCAM within a multicore system architecture that has been already discussed, implemented and simulated on cyclone V FPGA family as a standalone memory in chapter 3.

DPCAM was redesigned for a special purpose to be employed as a shared cache in the multicore system. This architecture allows simultaneous access to the same shared module based on dual port architecture. It reduces the access latency due to the simple architecture and simple NFRA technique. The DPCAM shared cache is mainly divided into two techniques; dual port and NFRA. The main points are as follows:

1. The SB unit of all cores in the multicore system can use the first port (write port) of DPCAM to write the shared data where each new write is controlled by the CU.
2. The OF unit of all cores in the multicore system can use the second port (read port) of DPCAM to read the shared data, which have been stored by other cores.
3. For the writing mechanism, the DPCAM memory architecture is built to store the shared data in the first available empty location or to the oldest location written in case of no empty location. Therefore, there is no unused DPCAM location.
4. As shown in figure 3.5 in chapter 3. The shared data in any location will not be overwritten before at least n times writing elapses, where n is the number of DPCAM locations or lines. All principles of DPCAM in figure 3.5 in chapter 3 are applied in this work using the Gem5 simulator.

Figure 4.1

*DPCAM Architecture Within A Multicore System*



### *4.2.2 Multicore Using DPCAM Architecture*

The main parts of the multicore system simulator are the cache levels, shared cache architecture, cache coherence, INand the main memory. Figure 4.2 shows the main parts of the proposed multicore architecture. The proposed architecture is built based on two-level cache, INusing crossbar switch, Modified-Exclusion-Shared-Invalid (MESI) cache coherence protocol and main memory access.

Accessing the shared cache memory still has the problem of contention between the cores. This leads to an increase in the access latency and hence decreases the system throughput. Both cache levels design and INs among cores are the main factors that affect the performance in multicore systems ([Patterson & Hennessy, 2020](#_243i4a2)). In the proposed architecture, the DPCAM is employed as a global L2 (shared cache) and the local data cache is graded as a SA L1 cache.

When the Memory Management Unit (MMU) loads the threads to the L1 instruction cache, it loads the local data of the thread to the local data cache, and the shared variables to the DPCAM. A shared variable, of course, is seen and can be accessed by all cores in multicore system ([Patterson & Hennessy, 2020](#_243i4a2); [Stallings, 2013](#_1vsw3ci)). During the thread execution, the OF unit can equally access the local and the shared cache to fetch the required operands. The SB unit writes the resulting shared variables to be accessed by other cores. The tag of the variable includes its address, its version number and its valid bit. In the memory hierarchy access, cores first check L1 cache and L2 cache respectively. A miss is returned if no cache hits are found and the request for data is forwarded to the main memory. On the other hand, shared data should be checked whether another core has changed them or not in order to preserve the consistency between cores. This action has been achieved using MESI cache coherence protocol.

The proposed multicore architecture has been compared with the traditional multicore architecture using Gem5 simulator. Both architectures have the same parts of level of caches, cache coherence, IN and main memory. The shared cache memory is the sole difference, as the proposed design uses DPCAM in shared cache and the traditional design uses a four-way SA shared cache. The comparison results for both architectures will be shown in section 4.4.

Figure 4.2

*Multicore Architecture Using DPCAM Shared Cache*

Private Cache L1=64KB

Private Cache L1=64KB

Private Cache L1=64KB

DPCAM - L2 Shared cache

DPCAM - L2 Shared cache

DPCAM - L2 Shared cache

Crossbar Switch (interconnection networks between cores and shared cache)

MMU

To main memory

Size =1 MB

SB unit

OF unit

Key: MMU= Memory Management Unit OF= Operand Fetch SB= Store Back

## 4.3 A Modified Replacement Policy for DPCAM

In the proposed DPCAM, a NFRA replacement algorithm and a new writing mechanism were presented in section 3.3. The write operation is based on a pointer is implemented in the CU. If the core writes shared data in location Lx, the next write operation (from the same of other cores) will store shared data into location Lx+1. This mechanism can be repeated until location Ln-1, then returns to L0 to overwrite the oldest shared data. This technique is practical for implementation in a single-core or a limited number of cores inside a multicore system. The main problem of this mechanism is that an empty DPCAM location that stores a shared data related to a process recently terminated can be found with any DPCAM location. e.g., if the pointer points to location L10 to write a shared data, then the next coming data from any core will be stored in L11 automatically, whether it is empty or overwriting if it is full. This happens although there may be several empty locations in DPCAM because the CU automatically writes to the next location. Of course, this problem will decrease the system throughput because the shared data may be replaced before its use.

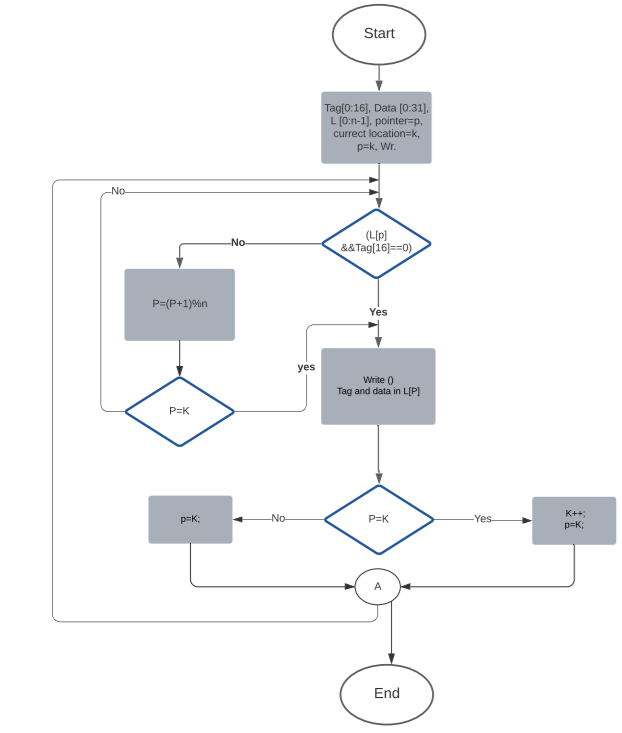
As a solution to this problem, a modified mechanism of the write operation and a replacement policy are proposed. For this purpose, a new bit is added to the tag field in each DPCAM location. Whether a location is valid to be written (if it is empty) or invalid to be written (if it is full) is indicated using the most significant bit in the tag field tag [16]. If tag [16] equals zero, it means that the location is empty and can be written, otherwise, the location is full and it cannot be written except if it is pointed to by the CU and no other empty location in DPCAM exists.

Figure 4.3 shows the flowchart of the proposed mechanism, in the beginning, the CU was set to initially point to the first location and test the tag [16]. Because the DPCAM is empty, the tag [16] always equals zero in all locations. Therefore, the first write operation will occur on the first line of the DPCAM, and so on until location n-1 (Ln-1). This means n write operations can happen before there is a need for overwriting any location. When the DPCAM becomes full, the pointer is pointed to the current location) L[k]( and the tag [16] is tested. If tag [16] equals zero then the shared data is directly written to the current pointer location and the pointer is incremented. Otherwise, if tag [16] equals one the tag[16] for another location is tested until finding an empty location. This means that writing the first empty memory location and then return to the current location (L[K]) for the next write operation. In case that no empty location is found, the pointer returns to its current location and overwrites the new shared data then it is incremented to point to location )L[K++](.

Note that tag [16] values are tuned by the scheduler in the compiler which must ensure that data is used during allowed time. Furthermore, if any core requires the read version of shared data for longer than the allowed time, it can store it in its private L1 cache.

Figure 4.3

*A Modified Replacement Policy For DPCAM*



## 4.4 Implementation of the DPCAM within the Multicore System

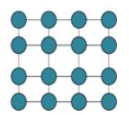
### *4.4.1 Experimental Setup*

The Gem5 simulator is chosen to study the performance of multi-core system based DPCAM as the second level of cache (shared cache). In Gem5, we use a full-system functional and timing simulator of multicore memory system called Ruby. Ruby is a part of Gem5 project. It provides full specifications and flexible cache memory such as cache architecture, cache coherence protocols, cache replacement algorithm and various INmodels ([Binkert et al., 2011](#_3yqobt7)). Figure 4.4 shows the main components in Ruby that can be tested and developed.

Ruby model is made up of multiple Specification Language Including Cache Coherence )SLICC) files. These files are compiled with a special compiler called SLICC compiler, which is written in Python and part of the gem5 source. Most of Ruby files were written in Python and some of them were in C++ files for inputs and outputs files ([Binkert et al., 2011](#_3yqobt7)).

Figure 4.4

*Ruby Memory System Simulator*



Interconnection network

Cache and Memory architecture

Cache Coherence Protocol

*Table 4.1.*

*The Configurations Of The Proposed Multicore Architecture*

| System Component | values |
| --- | --- |
| No. of processors  Processor specifications  Cache Levels  L1 cache specifications  L2 cache specifications  Cache coherence protocol  Cache model  Memory | 4.  Out-of-order, 2,6 GHz.  2.  2-way set-associative, size = 64KB and block size = 64 B.  Size= 1MB and size of each Bank=256 KB.  MESI directory-based protocol  Non-Uniform cache Access (NUCA).  1GB |

Ruby uses a first level of cache as private L1 data and instruction, and a second level of cache as shared L2. In our proposed study, same size and cache coherence protocol type of L1 and new architecture of DPCAM with same size of traditional SA L2 cache are used. Table 4.1 shows the detailed configuration of the multi-core architecture (number of processors, cache hierarchy, cache coherence protocol and main memory) used in our proposed system.

### *4.4.2 Benchmarking Program*

To test the functionality of the proposed system, simple and single-threaded test programs are used. Both traditional and proposed systems should produce similar results for these test programs. They also indicate that the proposed system works without any problems. Two test programs (who) and (df) are used; (who) returns the users currently logged in the system and (df) displays the amount of used and available space for disk mounted devices. On the other hand, multithreaded benchmarks programs are used to evaluate the proposed system performance and to provide comparative measures between proposed and traditional multicore systems. Various types of Standard Performance Evaluation Corporation (SPEC) CPU 2006 benchmarks are selected which are Bzip2, astar, gammes, hummer,gcc, mcf and sphinx3 ([Henning, 2006](#_3d0wewm)).

Bzip2 is usually used in compression and decompression files. The compression and decompression happen entirely in the different memory hierarchies. Bzip2 contains a very large number of instructions. Astar is usually used in computer games, used for 2D Path finding. Astar is memory-intensive and includes a large number of iterations during inputs and executions. Gammess is usually used in quantum chemical computations, it is Weak in parallel execution. Hmmer is usually used in the search sequence database. It uses Hidden Markov Models. Gcc is an optimizing compiler with a large number of input parameters. Mcf is usually used in combinatorial optimization and it consumes a large size of main memory during executions. Sphinex is usually used in speech recognition and it has a large number of instructions and random operands. These benchmark programs were chosen because they provide the best evaluation for the system's performance by having memory-intensive access with a large number of iterations and instructions.

#### 4.4.2.1 Single thread (who) and (df) benchmark programs. To test whether the proposed architecture works without hitches (who) and (df) programs are used, these programs were executed in both traditional and proposed architecture as the following:

1. #scons build/ALPHA/gem5.opt RUBY:True PROTOCOL=MESI\_Two\_Level -j n , n=number of cores+1, depend on the number of cores in your computer.
2. #./build/ALPHA/gem5.opt configs/example/se.py \

> --RUBY --num -cpu=4 --L1i\_size=32KB --lL1d=32KB --L2\_size=512KB \

> --bench= who, 4: number of cores in simulated architecture

In point one, the cache memory hierarchy architecture on gem5 simulator is built. This architecture is based on RUBY model. It also uses an ALPHA instruction set architecture (ISA) and MESI cache coherence protocol. This should be applied in both two traditional and proposed architecture. In point two, the (who or df) benchmark on multicore system architecture is executed. The entire system configuration (number of cores and cache level size) is stored in the file configs/example/se.py that is the emulation mode of the gem5. Finally, the name of the benchmark program (who or df) is written.

Analyses of the traditional and proposed multicore architecture can be found in appendix C. In addition, it can be noticed that the results in both traditional and proposed systems are the same. Therefore, these results in (who) and (df) benchmark programs prove that the proposed DPCAM architecture works in a multicore system without problems.

#### 4.4.2.2 Multithreading benchmark programs. This section describes the performance using the tested Multithreading benchmark programs. Seven benchmark programs; Bzip2, astar, gammes, hummer,gcc, mcf and sphinx3 are used in this part. These programs were executed in both traditional and proposed architecture as the following:

1. #scons build/ALPHA/gem5.opt RUBY:True PROTOCOL=MESI\_Two\_Level -j n , n=number of cores+1, depend on the number of cores in your computer.
2. #./build/ALPHA/gem5.opt configs/example/se.py \

> --RUBY --num -cpu=4 --L1i\_size=32KB --lL1d=32KB --L2\_size=512KB \

> --bench= X --maxinsts=1000000 , X: is the name of benchmark program

1. # cat /gem5/m5out/stats.txt

It can be noticed that running these benchmark programs is similar to the single thread benchmark the maximum number of instructions is selected depending on the number of instructions for each benchmark program. In this case, if the maximum number of instructions is less than the instruction number of any benchmarking program then the Gem5 displays a message after reaching the maximum number of instructions. The message is "tick 186333000 because a thread reached the max instruction count". The results of both traditional and proposed multicore architecture, which contains instruction count hit and miss rate and Instruction Per Cycle (IPC) for any benchmarking program, are stored in the path in point 3. All these results and configurations are shown in appendix C.

## 4.5. Simulation Results

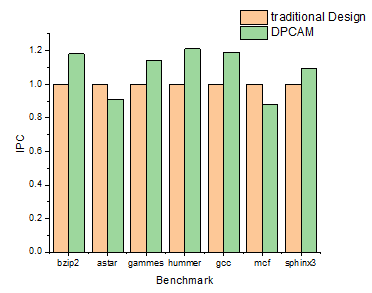
According to the above configuration, the proposed architecture has been simulated with various test benchmarks. Figures 4.5-4.6 show a comparative study of system performance in terms of IPC and miss rate between a multicore systems that use L2 based SA shared cache and the proposed multicore system which uses L2 based DPCAM shared cache. The proposed architecture exploits the features for DPCAM and NFRA replacement algorithm to improve its performance. All of these findings are displayed in Figures 4.5-4.6 and are normalized to the traditional architecture result for each benchmark.

Figure 4.5 shows the performance of the multicore system of the traditional and the proposed DPCAM design in terms of Instruction per Cycle (IPC) with various benchmark programs. It can be observed that the DPCAM architecture improves performance by about 8.7% from the average of various benchmark programs. In addition, it can be seen that among all test programs, mcf and astar do not produce any improvement. On the contrary, the system performance declines in terms of IPC. This happens because some benchmark programs are memory intensive and they may not benefit from using the shared cache. This is the case for the mcf and astar programs ([Prakash & Peng, 2008](#_4c5u7s8)).

Figure 5.5 presents the miss rate for the proposed DPCAM with NFRA instead of the LRU replacement policy in traditional design (Appendix C displays the count number of this miss). In the proposed architecture, the DPCAM miss happens if the data has not yet been created due to a scheduling issue or if it has been overwritten due to a size limitation. Otherwise, if the shared data is produced it becomes available to all OFs units.

Figure 4.5

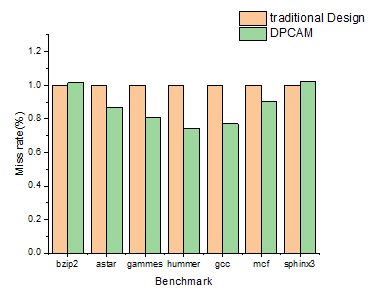
*System Performance in (IPC) Comparison*



Generally, it can be observed that the miss rate of the proposed system is clearly reduced with an average of about 13% compared with traditional design. However, the miss rate is reduced for all test programs except for test programs sphinx3 and bzip2. Sphinx3 has produced around a 1.9% increase in miss rate. On the other hand, bzip2 has produced a 1.2% increase in miss rate. This is because of the large number of reference instructions and iterations for these two types that have about three billion load instructions. Finally, it can be observed that in some benchmarks, the shared cache miss rate is not enough to judge the system performance, because it does not take into account the cause of miss, penalty of misses, and the latency of access in hit. This is obvious in bzip2 and sphinx3 where they do not improve the miss rate but the performance is clearly increased.

Figure 4.6

*System Miss Rate Comparison*



# CHAPTER V

# A New Design of Multicore IN Based on DPCAM

This chapter discusses a new IN organization called MPCAM organization. It is a new IN in the multicore system that utilizes the DPCAM as a part of the interconnection structure. In the following two sections, an overview of the DPCAM principles is given. The detailed architecture of the MPCAM organization is presented in section 5.3. The results of the testing are summarized in section 5.4. Finally, section 5.5 presents an overview of MPCAM applications.

## 5.1 Introduction

The need for multicore architecture has been arisen from the demand for larger and faster computers that can meet the growth of computational tasks. Again, within multicore architecture, the purpose affects the architecture. The purpose can be special on the application level e.g. scientific simulation and evaluation of list elements. This leads to a special purpose multicore system as in ([Bohm et al., 1989](#_kqmvb9); [Emani et al., 2021](#_34qadz2)). The purpose can be on a higher level than application e.g. to meet the need of dataflow computations or architecture dataflow ([DOU et al., 2020](#_1jvko6v); [Levchenko et al., 2018](#_43v86uo)).

In the case of dataflow computation, the data flow principle is a new model of looking at instruction execution in programs. In a dataflow model, an instruction is ready to execute when its operands have been completed, and there is no need for a program location counter ([Patterson & Hennessy, 2020](#_243i4a2); [Stallings, 2013](#_1vsw3ci)). The dataflow computation is represented by a program representation called "dataflow program graphs". These graphs serve as a formally specified interface between system architecture on one hand and the user programming language on the other. The early dataflow machines were designed to meet what is called now fine grain dataflow. Fine grain exploits parallelism on the machine instruction set level. This type of parallelism involves a large exchange of parameters among processors, this resulting in a large communication overhead ([Patterson & Hennessy, 2020](#_243i4a2)). By raising the level to that of multiple instructions, a "large-grain" dataflow or "macro-dataflow" is obtained. In macro-dataflow, the node of the dataflow graph corresponds to several instructions which can be executed in sequence. In this way, both control and dataflow are combined, resulting in much reduced operand passing overheads.

Several architectures have been created to suit this model of computation ([DOU et al., 2020](#_1jvko6v); [Levchenko et al., 2018](#_43v86uo); [Nowatzki et al., 2017](#_2j0ih2h); [Teifel & Manohar, 2004](#_y5sraa)). Since the shortcomings of the architecture of dataflow cannot be completely separated from the computational, the system must be redesigned to meet the needs of the computational model it uses. Based on the findings of chapters 3-4 and with the attractive features of the DPCAM to simultaneous access borne in mind, the MPCAM interconnection model was suggested. In this model, the main goal is to allow all cores to access any shared data over the INwithout contention.

In a multicore system, an INscheme to provide communication between the cores and shared memory is needed. Several interconnection schemes have been utilized in different architecture. A detailed description and analysis of these schemes were given in chapter 2. All these schemes are based on the idea of arbitrating or controlling simultaneous requests made by several cores to the same interconnection resources or shared memory. Both interconnection structure and shared memory architecture represents the main problems when more than one core tries to access the same destination. A new shared memory called DPCAM was proposed to solve the shared memory problems ([Abumwais et al., 2022](#_11bux6d)), and developed later to suit multicore architecture. The later version of DPCAM was embedded and simulated inside multicore architecture in chapter 4. In this organization, each DPCAM module can be accessed by two cores simultaneously, on the other hand, throughput and the hit rate were improved to good values compare to traditional SA shared cache organization over the system. DPCAM can service two requests, however, when more than two cores try to access the shared memory, the multicore system suffers from latency due to competition between cores to access the shared data. A full description, implementation and analysis of the DPCAM were given in chapter 3 and chapter 4 respectively.

The DPCAM is designed to support both the computational model and the architecture model. The goal of this architecture is to test the following:

1. The validity of the MPCAM IN.
2. The performance of the MPCAM IN.
3. To find possible applications for this architecture in many-core systems.

## 5.2 Embedded the shared cache within INs

The architecture of the multicore system described in chapter 2 either DSM or SSM has a long-standing issue in the IN. During our research in this work, we noticed that all of the issues stem from the fact that, while transferring from multiprocessor to multicore architecture, the architects are still stuck with the same Component off the Shelf (COTS) they were used on the discrete architecture. There is no reason to believe that the shared cache, for example, needs to be a separate component and that the busses cannot be used within the cache architecture. On the other hand, the arbitrating unit, which was shifted from a small control unit in based bus interconnection to a router in NoCs to allow multiple cores to access the same resources at the same time, is still another fundamental challenge in multicore architecture.

In fact, in any type of IN, shared cache modules can be embedded within the buses. These embedded modules will be exploited in two ways: they will be restructured to service multiple cores at the same time, and they will compensate for the complexity of arbitrating unit, reducing the router's burden and its overhead. This means small shared cache modules can be relocated at the cross points of the IN, as in “router or switch can be embedded within the IN.” As a result, the DPCAM that has been implemented and verified can be moved at the cross point’s buses of any INtopology.

Small FIFO buffers are embedded in the crossbar IN([Ayyad et al., 1996](#_1xaqk5w)). At Hebrew University, this architecture was designed on a printed circuit board (PCB) to connect quad Intel 486 processor. The number of gates and input/output pins of this architecture could not be accommodated by semiconductor technology at the time. 16-bit data width buses and a message-passing were used in communication between source and destination. This architecture suffers from communication overhead, as the message passing techniques needs buffering, packetizing, and de-packetizing the shared data in the right destination memory address.

Small RAM-based DPCAM is embedded in the crossbar IN([Abumwais & Ayyad, 2018](#_j8sehv)). The design was implemented using the Stratix-3 FPGA family from ALTERA to connect three SSM cores. Due to the ALTERA FPGA device family's transistors and input/output pins limitations, it is difficult to expand the multicore architecture. ​In this organization, all the first ports of RAM-based DPCAMs are connected to the horizontal buses for writing the shared data, and all the second ports of RAM-based DPCAMs are connected to the vertical busses (output buses) of the crossbar. This organization represents the base of the proposed design "A scalable interconnection scheme in many-core systems" which is described in chapter 6. However, it has been demonstrated that the suggested DPCAM ([Abumwais et al., 2022](#_11bux6d)) is extremely beneficial and may be used as shared memory in multicore architecture. Following that, we used this as the foundation for developing MPCAM as a key component of a multicore IN.

## 5.3 Proposed INfor Multicore System

### *5.3.1 Multi-Port CAM (MPCAM) Organization*

Figure 5.1 shows the MPCAM organization. We have redesigned the DPCAM to become an array of two-dimensional DPCAM elements. These elements are distributed and embedded on the cross points of the multi-core interconnection. The DPCAM module that is used has two ports. They allow concurrent read/write operations from the two ports as long as they do not access the same DPCAM location. The MPCAM rows and columns are equal and are equal to the number of cores in the proposed multicore system. Through the input port, the SB unit of the pipeline core writes the data and its tag to the first available location. Through the output port, the OF unit of the pipeline core applies the tag of the required data so that the DPCAM searches for the data in all DPCAM location simultaneously and reads it if found. In the MPCAM organization, DPCAM modules are connected to the horizontal buses for SB cores, each in its row, and are connected to the vertical buses for OF cores, each in its column. This allows the shared cache to be accessed by multiple cores simultaneously without blocking. Furthermore, because there is no access blocking, this design totally eliminates the need for shared memory access arbitration.

Figure 5.1

*The Organization Of The MPCAM*

Array of CAM element

SB0

SB1

SBn-1

OF0

OF1

OFn-1

**Port 1**

**Port 2**

Key: SB1= The output of the store back of core 1

OF1= The output of the operand fetch of core 1

### *5.3.2 MPCAM Organization Analysis*

The MPCAM organization is depicted in Figure 5.1, and the following are the essential points that summarize the organization's function.

1. SB units from all pipeline cores can simultaneously write shared data to the DPCAM modules in their row. There is no waiting in line, no request for arbitration and so no contention between cores. This is totally different from DPCAMs located on the crossbar's edge, as is customary, where the broadcast induces significant latency and contention on the shared resources.
2. OF units from all cores can simultaneously read the shared data to the DPCAM modules in their column. In addition, there is no waiting in line, and there is no request for arbitration.
3. During the writing operation, the DPCAM is implemented to write the shared data in the first empty location or pointed to the least recently written in case there are no empty locations as is described in chapter 4.
4. The stored shared data will not be overwritten until at least n clock cycles have passed, n refers to the number of DPCAM lines. Furthermore, if the core requires a shared data for longer than the time limit, it can store it in its private L1 cache.
5. As explained in chapter 3, two DPCAM modules have been implemented one for near access, and the second for far access which is less frequently used.
6. As a solution for point 4, these two DPCAM modules (near access and far access) are implemented at each cross point; the cores will reach the far access module less frequently, allowing more time to pass before they need to replace a shared data.
7. By supplying the tag to all DPCAM modules in its column, the OF unit for any core can search for the needed shared data. It will be observed that each column has a copy of the shared data because it was broadcast to all DPCAM modules in the row. As a result, all cores can concurrently search for the same or different shared data in their MPCAM columns.
8. It is observed that there is no contention between cores to access the shared MPCAM. Furthermore, because all shared data is updated versions for all cores in the shared MPCAM, there is no need to use cache coherence protocols or core-to-core communication to broadcast the updated version of the shared data. Therefore, MPCAM is snooping cache coherence protocol by itself. Full description of cache coherence protocol is given in appendix A.
9. Finally. In the MPCAM organization, there is no access blocking, no need for router devices, no need for cache coherence protocol and hence no additional latency.

### *5.3.3 The MPCAM-Based Multi-core System*

In the MPCAM-based multi-core system, if n cores are used, then the MPCAM must have n horizontal buses, n vertical buses, and n2 DPCAM modules; n modules in each row and n modules in each column. Eight cores can be built design using this organization. The SB units of the core pipelines are connected to the row busses and the OF units are connected to the vertical buses. Through the horizontal bus, the SB unit can write the data and its tag to all DPCAMs in the row. This means that each column will have a copy of the data. So, the OF unit of the core can search for and read the data through the vertical bus of its column. Figure 3 shows an MPCAM-based multi-core organization. This organization achieves simultaneous access to the shared cache and eliminated the need for router devices between cores inside the same multi-core system.

Each core has two level-1 cache modules: a 32 KB instruction cache and a 32 KB data cache. Without exceeding the silicone restrictions, level two of cache can range from 2 MB to 8 MB of the data cache. The Memory Management Unit (MMU) loads the shared data to a specified raw in the MPCAM while executing a program. All of the system's cores have equal and simultaneous access to MPCAM architecture.

Figure 5.2

*The Proposed Multicore System With MPCAM*

Core1

Coren-1

Core2

Embedded MPCAM Interconnection Network

Global Memory Management Unit (MMU)

To main memory

L1 Data cache

L1 Ins. cache

L1 Data cache

L1 Ins. cache

L1 Data cache

L1 Ins. cache

Key: MMU= Memory Management Unit L1 Ins cache: Level 1 instruction cache

## 5.4 Performance Analysis of MPCAM Organization

MPCAM has been implemented, compiled, and verified in a multicore system using Quartus Prime 20.1, which includes the Intel-supported ModelSim package and Nios II EDS for design and simulation ([Intel, 2021](#_41wqhpa)). MPCAM was designed using Cyclone V FPGA family, which uses 28 nm technology ([Intel, 2018](#_is565v)). Four cores were used to assess the latency of read and write operations of DPCAM organization. Both schematic files and Verilog HDL code have been used to implement the  MPCAM organization. VWF was used to verify and debug the files in both functional and timing simulation. To simulate and observe the latency of MPCAM reading and writing operations, a special test bench was used. Cyclone V FPGA family which has suitable specifications such as high-integration capabilities, number of pins, area of the chip and power consumption was used to implement DPCAM organization. The main advantage of a Cyclone V FPGA is that consumes about 40% lower power than all previous generation FPGA families, increased memory capacity and high-integration capabilities ([Intel, 2018](#_is565v)). More other characteristics can be found in intel-site ([Skhiri et al., 2019](#_39uu90j)).

### *5.4.1 Functional Simulation*

Using functional simulation the functionality of the MPCAM organization was achieved. Separate write and read operations, as well as simultaneous read and write operations into the different memory locations, and simultaneous read and write operations into the same memory location were accomplished.

The same test-bench which was used in DPCAM evaluation is used to analyze functional and timing MPCAM. The test-bench is detailed described in chapter 3, figure 3.8. The main functions for MPCAM organization in terms of reading, writing, broadcasting shared data and simultaneous read-write have been accomplished after multiple runs of test-bench on the simulator.

Figure 5.3 shows a thumbnail image of several clock cycles in the functional simulation of MPCAM organization, the clock period equals 10 ns for reading and writing to the 64 KiB in each crosspoint module. In the first interval (0 to 10 ns), all cores can write their shared data simultaneously. As it was explained in chapter 3, in the write process core adds data and tags (D-core and Tags-core) when the write (WR) signal goes down. The written data (DI-core) appear clearly on the DPCAM locations. This indicates that the input shared data have been stored in the DPCAM locations that were specified. In the second interval (10 to 20 ns), all cores can read shared data simultaneously. In case of reading, the core generates the corresponding tag (Tagd) with a read (RD) signal to retrieves the stored data. The stored data appears on the core's output buses (Dout -core) as soon as the RD signal becomes high. In the third Interval (20 to 30 ns), various cores can read and write simultaneously into different locations, where core1 and core 2 writes shared data with Tags (1F) and (2F) respectively. On the other hand, core 3 and core 4 read the shared data that already have been written with Tagd (2A) and (3A). In this case, all shared data are read and written correctly. In the fourth interval (30 to 40 ns), each core can read and write simultaneously with other cores and into the same or different locations. Where core 2 writes shared data with tags (8F) and simultaneously reads data from the different locations with Tagd (2A). On the other hand, core 3 writes a shared data with Tags (9F) and simultaneously reads data from the same location with tagd (9F), here the priority is given to the write operation and delayed read to the next cycle. Finally, in the fifth interval (40 to 50 ns) core 1, core 2 and core 4 simultaneously read the same data previously produced by the third core. The results appear on the output pins in the functional simulation without taking into account the delay from design components.

### *5.4.2 Latency Assessments*

Again, Timing Analyzer tool from Intel is used to evaluate the MPCAM organization access latency. The Timing Analyzer is used to assess the access latency for reads and writes in all of the access scenarios covered in functional simulation. The MPCAM organization timing simulation for both read and write operations of the size 64 KiB on each cross point module using the Cyclone V FPGA device is shown in Figures 5.4 and 5.5.

Figure 5.4 shows the same scenarios that have been discussed in functional simulation. In the first interval (0 to 10 ns), all cores simultaneously broadcast their shared data with their special Tags. Instantly when the WR signal goes down, the written data (pine DI-core) appear clearly on all modules in its row after a time delay. After running the test-bench on the simulator, one hundred times it was noticed that the average access latency of writing on MPCAM organization is about 1.084115±0.03384 ns. The second (10 to 20 ns) interval shows the latency assessment for read operation. To read data that is already written in all MPCAM modules, the Tagd (1A,2A,3A and 4A) which is provided by core1,core2,core3 and core4 respectively are simultaneously compared to the tags in all MPCAM column modules. The results appear on output buses (Dout-core) after delay time. The delay for read operation was calculated using an average of roughly one hundred intervals of test-benches, it was noticed that the delay for a read operation is around 1.27804±0.086823 ns. The third (20 to 30 ns) and fourth (40 to 50 ns) show the latencies assessment for simultaneous read and write operations to the memory locations. Again, the simulator for both two cases was run around 100 times. For the case with simultaneous write and read operations to the different memory locations the latency equals 1.1909±0.02363 ns which is nearly equal to the latency of the read operation. In the case of simultaneous write and read operations to the same memory locations the data is written to the target location with a latency 1.3105±0.091955 ns, while the read operation waits until the next interval then the shared data is read with a total latency 1.2780±0.086823 plus the time of the interval. All these measurements are already compatible with the equations were described in section 3.4.2. In addition, these measurements were proved by using SPSS and T-test with 95%of a confidence interval.

Figure 5.5 shows two scenarios. In the first interval (0 to 10 ns), core 1 broadcast shared data to all DPCAM models in its row with Tags (0078). Instantly when the WR signal goes down, the written data (DI-core1) appear after a time delay, which is equal to the latency of the write operation, is about 1.084115±0.03384 ns. In the second (10 to 20 ns) all cores simultaneously read the shared data which was produced from core 1 in the first interval. The results appear on Dout-core pines with a time delay which is equal to the latency of the read operation 1.27804±0.086823 ns. From the timing simulation, we can see that the write and read operations to the shared cache wouldn’t take more than 1.3 ns compared to 5.3 ns at L2 cache and 19.5 ns in L3 cache in Nehalem Intel i7 and 12 ns at L2 cache and 21 ns in L3 cache in AMD’s Bulldozer family ([Hruska, 2018](#_2hio093)).

As a result of this simulation, it was demonstrated that all cores in the MPCAM organization can access shared data simultaneously without contention, blocking, and arbitration issues.

Figure 5.3

*MPCAM Organization Functional Simulation*

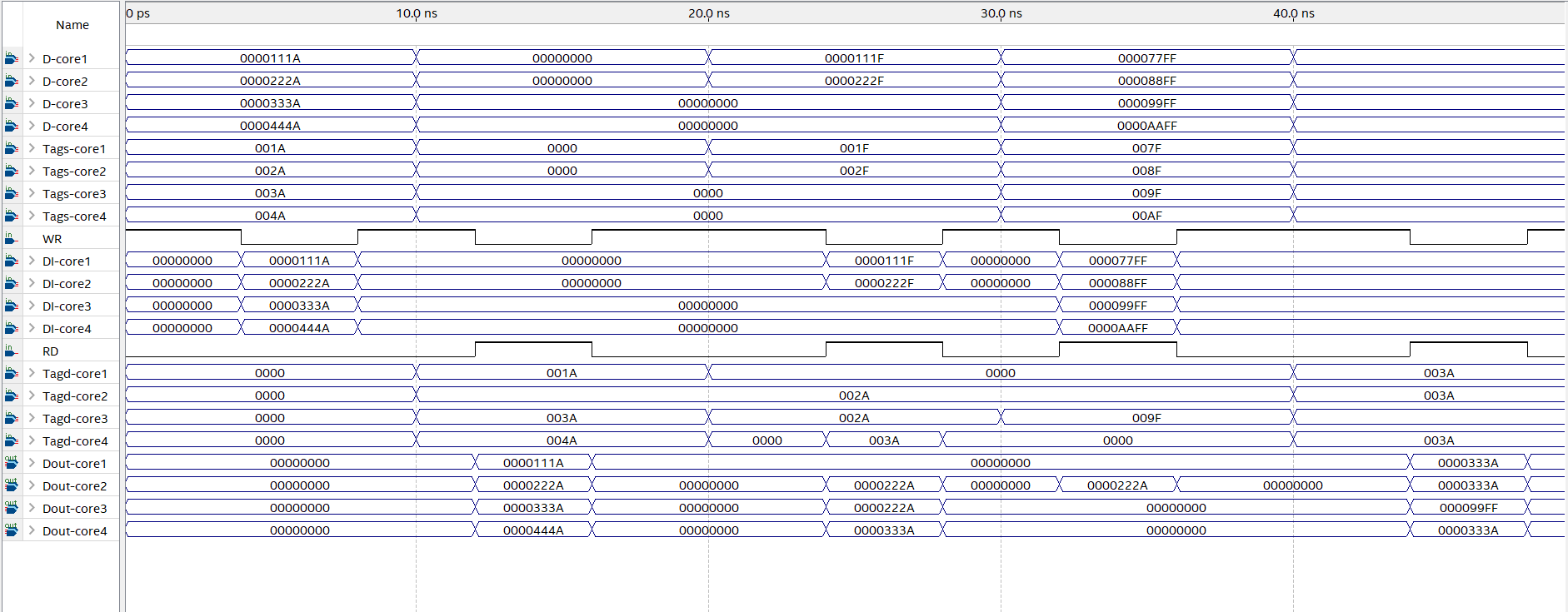


Figure 5.4

*MPCAM Organization Timing Simulation 1*

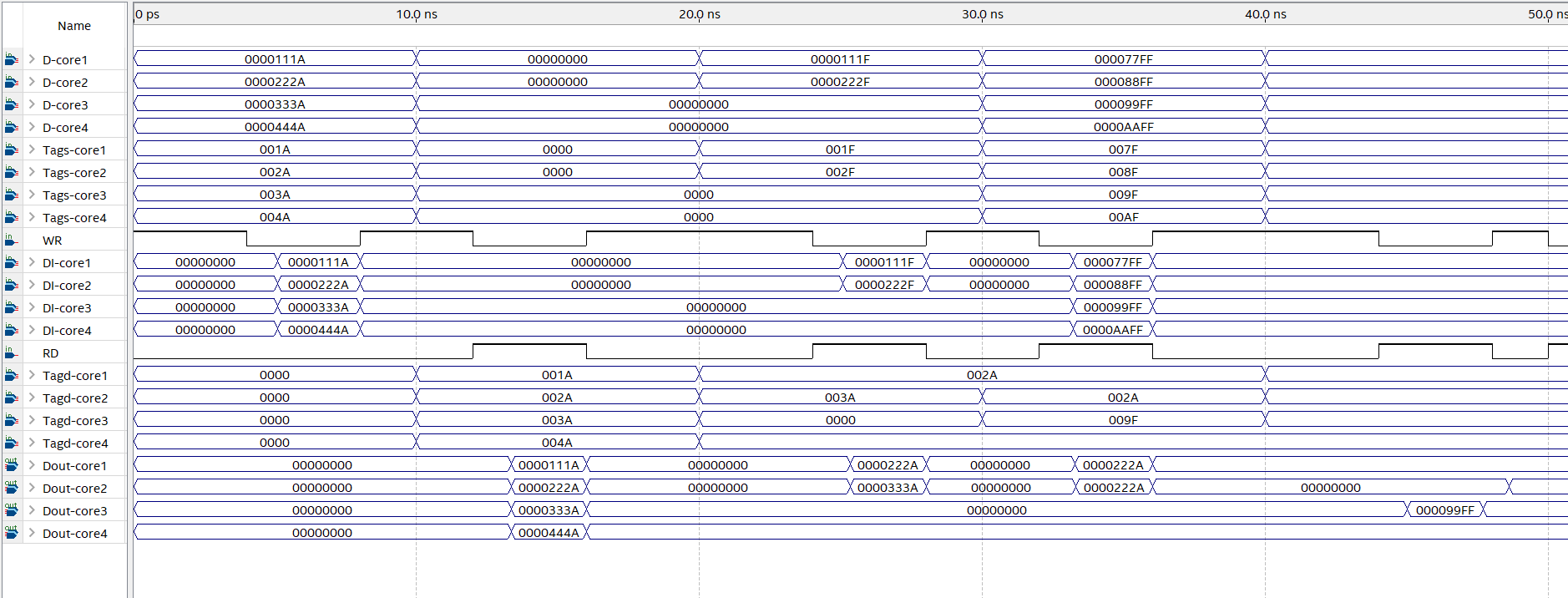
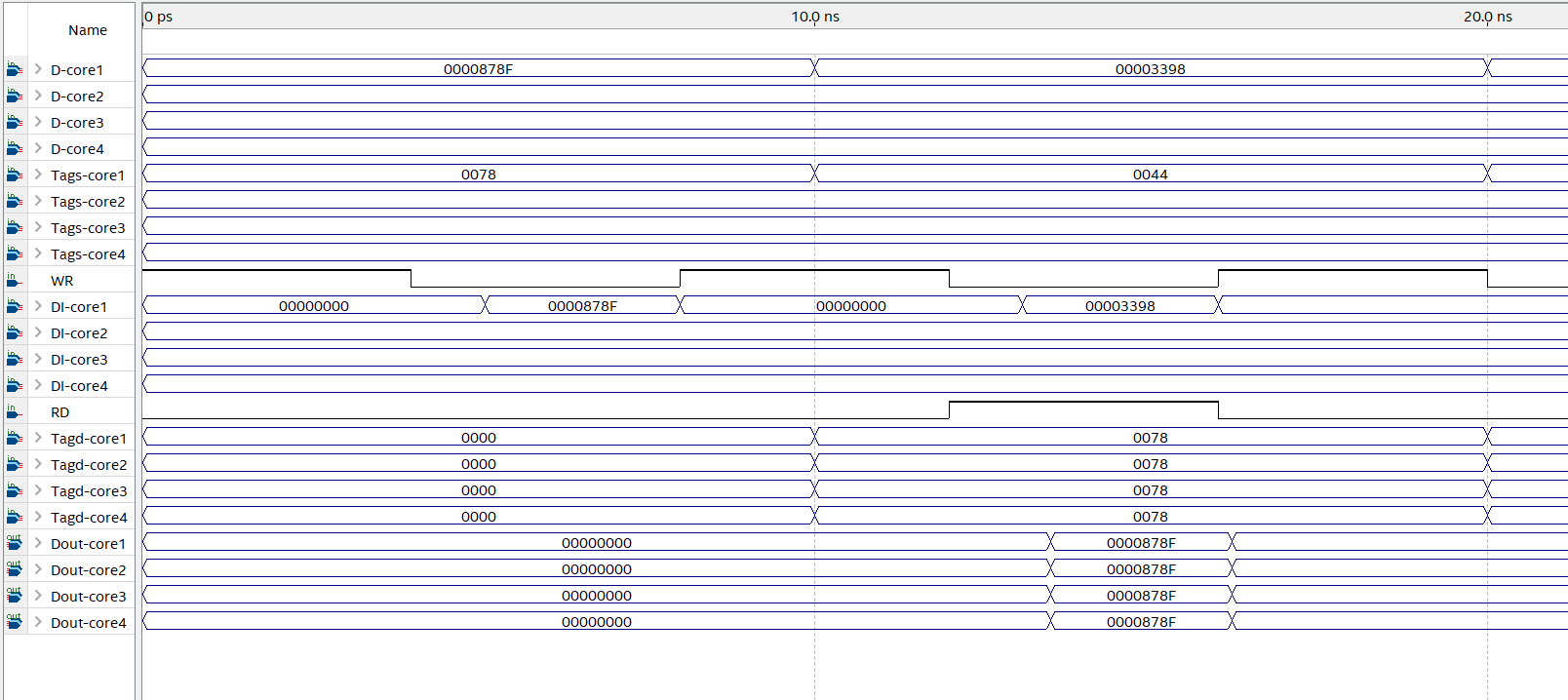


Figure 5.5

*MPCAM Organization Timing Simulation 2*



#### ***5.4.2.1 MPCAM Comparison Statistics.*** SPSS was used to find the minimum, maximum, mean and statistic error for all scenarios. Table 5.1 shows descriptive statistics for write/read latencies in MPCAM organization.

The T-test showed that there is evidence that write latency of Write is less than write latency of write\_simultaneous with 95% of confidence interval (significance is less than 0.05). It also showed that there is evidence that read latency of Read is less than read latency of read\_simultaneous with 95% of confidence interval (significance is less than 0.05).

The Levene’s Test showed that there is evidence that variances of Write and Write\_simultaneous are not equal to each other (Significance is less than 0.05). On the other hand, the Levene’s Test for Read and read\_simultaneous showed that the variances for both of them are equal (Significance is greater than 0.05). Table 5.2 and 5.3 show the t-test for write comparisons and read comparisons respectively. All t-test and the Levene’s test statistics are shown in appendix D.

*Table 5.1*

*Descriptive Statistics For Write/read Latencies*

|  | N | Minimum | Maximum | Mean |
| --- | --- | --- | --- | --- |
| Statistic | Statistic | Statistic | Statistic | Std. Deviation |
| Write  Read  Write-simultaneous  Read-simultaneous | 100  100  100  100 | 1.025  1.0820  1.0840  1.0200 | 1.2250  1.4790  1.4920  1.9850 | 1.084115  1.278040  1.310480  1.160854 | .0338499  .0868235  .0919551  .2363034 |

*Table 5.2*

*T-test For Write Comparisons*

|  | | t-test for Equality of Means | | |
| --- | --- | --- | --- | --- |
| Std. Error Difference | 95% Confidence Interval of the Differences | |
| Lower | Upper |
| Write\_Comparisons | Equal variances assumed  Equal variances not assumed | .0238716  .0238716 | .1238141  .0238716 | .0296639  .0293957 |

*Table 5.3*

*T-test For Read Comparisons*

|  | | t-test for Equality of Means | | |
| --- | --- | --- | --- | --- |
| Std. Error Difference | 95% Confidence Interval of the Differences | |
| Lower | Upper |
| Read\_Comparisons | Equal variances assumed  Equal variances not assumed | .0126468  .0126468 | .0573796  .0573801 | .0075004  .0074999 |

## 5.5 The MPCAM Status

As mentioned earlier, this organization is designed so that it can accommodate eight cores. Four of these cores were built. The Cyclone V capabilities and the time available in the work allowed the testing of this size successfully. The design of the organization required sequential logic, FPGA chip fitting compatibility and the initial design had to be modified due to timing problems. As a result, only four-core MPCAM organization was fully tested as described in section 5.3. During tests, the data parameters were successfully transferred between the cores and shared memory organization in both directions. A sample of the results is shown in figures 5.3-5.5. This sample was recorded using ModelSim, VWF and Timing Analyzer tools available in the INTEL Quartus Prime. An explanation of these analyses is given in appendix D. Currently, the author is testing the other large size 8X8 MPCAM organization in order to facilitate further projects by any interested researcher to scale this organization or develop another version of the organization with more powerful processing. Tests on this organization have shown that cores can open channels to all at the same time, and perform simultaneous store of shared data to all shared memory modules. The fact that the DPCAM architecture was able to write and read to the same module simultaneously shows that the MPCAM organization should be able to write and search shared data simultaneously from any core.

An unprecedented shared memory organization was presented. By adding a number of input and output ports equal to the number of the cores. This organization mostly removes the main issues of communication between cores and shared memory. The MPCAM organization acts as a scratch book, allowing the core to broadcast all versions of shared data without fear of overwriting until at least (n-1) memory lines. n is refer to memory lines in each DPCAM module. So, as all the versions of each shared data is available, each has its unique tag, the core can access the version it needs at any time and in any order, as far as the version is available. A 32-bit tag provides a unique value for four Giga versions of variables. Therefore, the need for cache coherence is eliminated a thing which saves the non-compute time spent in executing the cache coherence protocols.

As all tags of the DPCAM modules in the column of the MPCAM are compared with the applied tag simultaneously, the memory access time will be the same regardless of the module size and the number of modules in the rows and the columns of the MPCAM. This means that regardless of the N value, the access time of an (NXN) MPCAM, where N is the number of MPCAM in each row or column, will be the same as far as the FPGA semiconductor technology can accommodate it.

The MPCAM organization presented in this chapter proved to be used in a scalable organization to the many-core systems based on a multi-clusters architecture, with constant access time within a cluster. This organization will be discussed in chapter 6.

# CHAPTER VI

# A Scalable Interconnection Scheme In Many-core Systems

In this chapter, we are going to design an effective network topology that reduces the penalties that may be produced in a large-scale system. This chapter presents a routerless high effective INtopology based on a multi-cluster architecture where each cluster has a group of cores. The proposed INis capable of improving static performance metrics like scalability, cost, size, bandwidth, complexity, network diameter and latency. After implementation using FPGA design, some dynamic performance metrics such as shared memory access latency are also analyzed.

Throughout this chapter, the main components and the structure of the proposed INwill be explained in sections 6.2 and 6.3. The static performance metric of the proposed design and a comparative study with other topologies is conducted then the results are discussed in section 6.4.1. The implementation and testing using FPGA will be discussed in section 6.4.2.

## 6.1 Introduction

In multicore systems, an area in which considerable differences are found is that of INlinking core to shared resources such as shared cache, global memory banks and input/output devices. In this thesis, emphasis has been placed on the INto shared memory.

Multicore architectures with different interconnection schemes have been proposed. These different schemes include the based bus, based crossbar, NoCs based router and routerless NoCs. Most of the architectures use some variant of these schemes, with modification to improve the performance of the scheme used. Some architecture use a combination of two schemes. The goal of using different schemes is to improve some metrics like reliability, size, speed, diameter, connectivity and cost. In most architecture, speed (represented by bandwidth and latency) of the INis the most important factor, provided that the implementation is relatively simple and economical.

The nature, advantages and problems of known INs with various topologies and arbitration schemes were discussed thoroughly in chapter 2. In all interconnection schemes discussed in chapter 2, the following features can be recognized:

1. The performance varies widely from that of the common bus to that of the router or routerless NoCs.
2. An arbitration scheme is needed to solve the bottleneck problem wherever it is located in the interconnection scheme; this problem is costly resolved in routerless NoCs.
3. The request from the core to the shared memory is treated individually regardless of whether it is a read or write request and whether or not the same data has been accessed before.
4. The INtopology that includes arbitration units and the shared cache represents the main problems in any scheme i.e. cost, latency and bandwidth.
5. When the number of cores is increased significantly, all of these schemes face additional penalties. The arbitration time and hence the latency time of the interconnection increases as the number of cores connected to the network increases.

In Chapter 5, the authors present a MPCAM organization that eliminates these issues from the multicore system. The results we have got during these works are encouraging and this motivates us to continue working in this field. One main question is: whether we can scale system further while maintaining its advantages that achieved in the proposed multicore system at the same time? The main goal is to scale this system up with minimum penalty.

During the past decade, the processor designers started working on expanding the multicore system to many-core system that contains a large number, tens to hundreds, of cores ([Patterson & Hennessy, 2020](#_243i4a2); [Singh et al., 2013](#_3gnlt4p); [Udipi et al., 2010](#_2uxtw84)). A number of many core systems were produced over the last few years ([Anthony, 2013](#_vgdtq7); [Bohnenstiehl et al., 2016](#_338fx5o); [Cebrian et al., 2020](#_261ztfg); [Damkroger, 2017](#_1ulbmlt); [Mulnix, 2017](#_20xfydz); [NVIDIA, 2017](#_l7a3n9); [Vangal et al., 2008](#_320vgez)).

## 6.2 The Component of the Proposed Scheme

During the early stages of the proposed NCSC design, a decision was taken that using routers is not a choice because it adds burden and complexity to the network architecture. NCSC uses N-shuffle stages connected by N crossbar network based clusters; each consists of N processors. Conjugate shuffle stages have been used before in many INs ([Ayyad, 2005](#_40p60yl); [Francalanci & Giacomazzi, 2005](#_2i9l8ns); [Khanna et al., 2017](#_2fugb6e); [Yunus & Othman, 2011](#_xevivl)). This scheme has been modified to N-conjugate shuffle in order to meet our needs of multi-cluster connection where the conjugate core (manager core) of each cluster is made responsible for inter-cluster communication. Combining N-conjugate shuffle with the MPCAM organization as a shared cache for each cluster, a many-core system is obtained and it has the following features:

1. Within a single cluster (the multi-core system), the communication between cores is accomplished using the shared cache (the MPCAM) with an access time that equals that of the local cache.
2. In the inter-cluster communication, the core can read shared data of any other cluster with an access time equal to the time of one or two local cache accesses. The length of the access time (one or two clock cycles) depends on whether there exists a request from a local core to that shared cache or not.
3. Core i in cluster j can write a shared data to other cores by store its shared data in MPCAMj. Cores within the same cluster can read the shared data by OF of MPCAMj. On the other hand, cores from another cluster can read the shared data produced from core i in cluster j simply by access MPCAMj through the NCSC IN.
4. In the whole system, in both inter-cluster and intra-cluster communication, there is no need for arbitration units and their complexity which reduces the power and area cost. It also permits simultaneous access from every source to any destination on the proposed system, eliminating contention and lowering access latency.
5. In the whole system, each shared data, in whichever shared cache it exists, has a unique tag. This tag can be equally used by any core in any cluster of the system to access its data. The tag includes the variable identity (can be address + version number) in addition to two bits. The first bit decides whether the data is local or shared, and the second decides whether the shared data exists in the cluster shared memory (MPCAM) or in the shared memory of another cluster. This means that the shared cache address space is homogenous to all cores of the system.

### *6.2.1 The N-Conjugate Shuffle*

The Generalized Shuffle Network (GSN) is a model of MINs proposed by Bhuyan and Agrawal ([Patterson & Hennessy, 2020](#_243i4a2)). The Delta network, the (2 x 2) SE based MINs, the crossbar switch and the common bus can be derived from this model by substituting the correct values for the parameters used in this model. In this model, N inputs are connected to M outputs, for any arbitrary values of N and M. the size of crossbar module is uniform for each stage but may be different from the size used in other stages. The number of crossbar modules per stage may differ from one stage to another.

To systematize the link pattern between stages of the GSN, the shuffle concept must be defined. A q-shuffle of M playing cards, where (M=qc) can be viewed as follows: divide the deck of qc cards into q piles of c cards each, top c cards <0,1, ........, (c-1)> in the first pile, next c cards <c,c+1, ........., (2c-1)> in the 2nd pile, and so on. Now pick the cards one at a time from the top of each pile, the first card from the top of pile one, the 2nd from the top of pile two, and so on in a circular fashion until all the piles of cards are exhausted. This new order of cards represents an S q\*c permutation (q-shuffle) of the previous order. Figure 6-1 shows a 3-shuffle of 12 objects.

Recently, various improved shuffle interconnection has been proposed and implemented to provide effective inter-processor communication ([Francalanci & Giacomazzi, 2005](#_2i9l8ns); [Lahdhiri et al., 2020](#_3eze420); [Yunus & Othman, 2011](#_xevivl)). In multi-cluster communication, each cluster contains a core processor that is responsible for communicating with other clusters. So, a new INbetween clusters or between core processors in each cluster is needed.

N-conjugate shuffle is the best combination in this situation, it connects each conjugate core to another in a multi-cluster with minimum links cost and simple way. In order to understand this scheme, it is necessary to explain the structure and the function of the N-conjugate shuffle. The N-conjugate shuffle is a passive N-shuffle (no silicone devices are involved) which connects the element to its conjugate, e.g., it connects the element Eij to the element Eji and vice versa. Figure 2-a shows a 4-conjugate shuffle. Note: each connection is a bus connection.

It can be noted that the buses coming out from the same cluster do not cross each other. By aligning the opposite bus with these buses it can be obtained a group of buses that do not cross each other. As can be seen from figure 2-b, 12 bidirectional buses can be put in two groups, i.e., they can be accommodated in two layers of the chip.

Figure 6.1

*A 3-shuffle Of 12 Objects*

0

1

2

3

4

5

6

7

8

9

10

11

**M**

**S 4\*3 (M)**

Key: M= Number of objects

S= Shuffle

0

1

2

3

4

5

6

7

8

9

10

11

Figure 6.2-A

*The 4-conjugate Shuffle*

00

10

20

30

01

11

21

31

02

12

22

32

03

13

23

33

00

10

20

30

01

11

21

31

02

12

22

32

03

13

23

33

**Cluster 0**

**Cluster 1**

**Cluster 2**

**Cluster 3**

Figure 6.2-B

*The Equivalent Bidirectional Connection 4-conjugate Shuffle.*

00

10

20

30

21

31

32

00

10

20

30

01

11

21

31

02

12

22

32

03

13

23

33

**Cluster 0**

**Cluster 1**

**Cluster 2**

**Cluster 3**

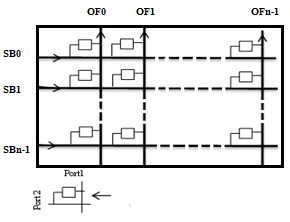
### *6.2.2 The MPCAM Organization*

The MPCAM organization and the MPCAM-based multi-core system were presented in the previous chapter. Figure 6.3 shows the MPCAM organization inside each cluster. On each cross point of the MPCAM, a DPCAM module is embedded. DPCAM architecture was described in chapters 3 and 4. In the MPCAM organization, the first port of DPCAM modules are connected to the horizontal buses for SB cores, each in its row, and are connected the second port to the vertical buses for OF cores, each in its column. This organization allows any cores in a multicore system to access the shared cache simultaneously without blocking.

MPCAM organization is intended to be used in intra-cluster communication and other clusters using N-Conjugate shuffle can access it. Within-cluster, cores can access (read/write) the shared data simultaneously with a constant access time equal cores local memory. In addition, using the proposed N-conjugate Shuffle IN, the core can access the shared data belong to another cluster in the MPCAM with a maximum access time of the core’s local memory plus the latency from N-conjugate shuffle. A full description and analysis of the MPCAM organization and MPCAM-Based multicore system were given in chapter 5 in section 5.3.

Figure 6.3

*MPCAM Organization Inside Each Cluster*



Key: SB= Store Back OF= Operand Fetch

## 6.3 The NCSC Interconnection Scheme

An efficient many-core system can be created if an efficient and simple interconnection scheme is provided. We found that the NCSC scheme suits the MPCAM based clusters very much. The NCSC uses an N-conjugate shuffle combination to connect between cores of the N system clusters. It is a simple connection method which removes most of the system contention on the inter-cluster level and it is easy to program.

Figure 6.4 shows how a bidirectional link (two buses) of the conjugate shuffle connects the OF units of two cores in two different clusters of the many-core system. The switches are shown in the figure guarantee that only one core can access a column of the shared cache of the cluster, regardless to which cluster this core belongs to.

Three bits of the address or the tag would be good enough to control the design switches. The only competition occurs when the OF unit of core i in cluster j (OFij) tries to access column j in cluster i while the OFji is trying to access the same column of the MPCAMi in its cluster. The same occurs if OFji tries to fetch a data from column i in MPCAMj of cluster j while OFij is trying to access the same column in its cluster (cluster j). In this case, the request coming from within the cluster is given the higher priority, where the request coming from another cluster has to wait for an extra clock cycle. This allows extra time for the writing core (the core producing the shared data) to write (broadcast) the variable to all modules in its row of the MPCAM.

Again, it must be noted that as the producing core of the shared data writes it to all modules in the Row, each column of the MPCAM is going to have a copy of this variable. Therefore, any core and its conjugate can access it regardless of which core has produced it.

Figure 6.4

*Connecting Two Cores Of Two Different Clusters Via The Conjugate Shuffle*

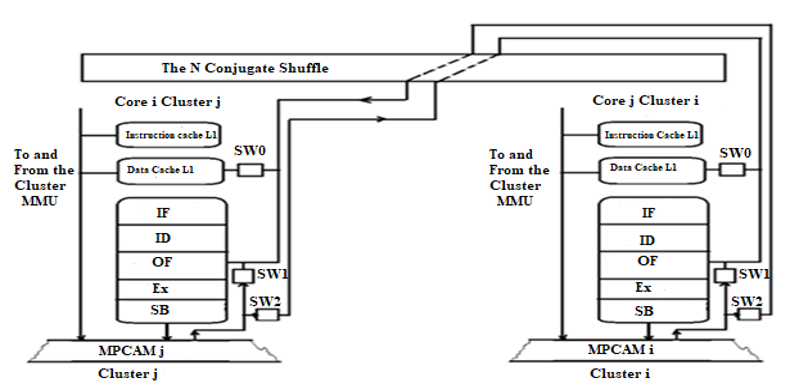
Key: IF: Instruction Fetch SW: Switch

ID: Instruction Decoder MPCAM i: Multi-port CAM organization in cluster i

OF: Operand Fetch MPCAM j: Multi-port CAM organization in cluster j

Ex: Execute MMU: Memory Management Unit

SB: Store Back



Let us consider various scenarios in which a path is to be constructed from source core to destination as shown in Figure 6.5. In scenario 1, assume that core OF00 in cluster 0 wants to access core OF30 in the same cluster and core OF32 in cluster 2 access/read core OF13 in cluster 3 simultaneously. Because the source and destination cores are in the same cluster, the communication process is as follows: OF00 searches the local MPCAM0 then access core 30 data if it was produced formerly. Whereas, OF32 searches the local MPCAM2 and if not found, by a simple mechanism. OF32 recognizes that the data does not belong to this cluster and communicates directly with the OF23, which is the conjugate of OF32 through the NCSC to access internal MPCAM3. In the next cycle, SB23 sends the data to be stored directly to OF32 on second cluster.

Scenario 2, similar to scenario 1 but two sources accessed two destinations simultaneously where all sources and destinations are located in different clusters. Core OF30 in cluster 0 wants to access core OF03 in cluster 3 and core OF32 in cluster 2 access/read core OF01 in cluster 1 simultaneously. Because the source and destination cores are in a different cluster, OF30 searches the local MPCAM0 and if not found, by a simple mechanism. OF30 recognizes that the data does belong to cluster 3 and communicates directly with the OF03, which is the conjugate of OF30 through the NCSC to access internal MPCAM3. In the next cycle, SB23 sends the data to be stored directly to OF30 on cluster 0. Whereas, OF32 searches the local MPCAM2 and if not found, by a simple mechanism. OF32 recognizes that the data does not belong to this cluster and communicates directly with the OF21, which is the conjugate of OF12 through the NCSC to access internal MPCAM1. In the next cycle, SB21 sends the data to be stored directly to OF12 on cluster 2.

In scenario 3, two sources accessed the same destinations simultaneously where the two sources are located in different clusters. Core OF33 in cluster 3 and core OF12 in cluster 2 want to access the same destination core 31, core 3 in cluster 1. This means that OF33 and OF12 wants to read from the same destination simultaneously. Since the source and destination are located in different clusters, OF33 uses OF13 which is the conjugate of OF31. By simple mechanism, OF13 recognizes that the data does not belong to this cluster and communicates directly with OF31 through the NCSC to internally search the data on MPCAM1. At the same time, OF12 searches the local MPCAM2 and if not found, it will search in the destination cluster. Because OF21 is the conjugate of OF12, they are directly connected through the NCSC to access internal MPCAM1. In the next cycle, SB21 sends the data to be stored directly to OF12 on cluster 2. Therefore, both OF31 and OF21 simultaneously search in MPCAM1, each on its column. In this case, both two sources can read from the same destination without any delay or deflecting the path.

Scenario 4 is another example for communication where the source and destination cores are on different clusters. Core OF30 in cluster 0 wants to access shared data from OF11 in cluster 1. Because the source and destination are located in different clusters, OF30 uses the OF10 which is the conjugate of OF01. The shared data are read from the column of MPCAM1. So, any core in cluster 1 can read the shared data produced by any core simply by providing the tag of these data on the OFn. In the next cycle, the shared data is read by OF30 on cluster 0.

Finally, it can be observed that NCSC meets our needs in terms of removing the complexity of router devices and resolving the problem of simultaneous access to the same destination. Furthermore, the latency (the access time) of the shared data will be improved. It's equivalent to that of two cores sharing a bus. The mathematical model of the shared bus performance, i.e. the bandwidth, the probability of acceptance, and the latency, is very well known ([Ayyad, 2005](#_40p60yl); [Patterson & Hennessy, 2020](#_243i4a2)).

The bandwidth (BW) is the number of requests accepted and fulfilled by the bus during one bus cycle (here for the pipeline, the bus cycle equals one clock cycle). It is formulated as shown in equation 1, where n is the number of cores connected to the bus (here two cores are connected), and r is the probability that the connected core is making a request to use the bus.

………… (1)

The probability of acceptance (Pa) is the bandwidth divided by the number of requests presented to the bus during the cycle (nr).

………… (2)

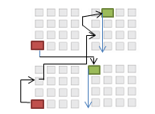
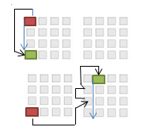
The average latency (the average access time tacc) is given by 1/Pa.

tacc= ………… (3)

In the worst case, the latency occurs when the core and its conjugate present their requests to corresponding column at the same time (r = 1). In this case, one of them is going to have a one clock cycle access time (tacc = 1 clock cycle) while the other will have a two clock cycle access time, i.e., if r = 1, the Pa = 0.5, and tacc = 2 clock cycles. In the case of r = 0.5, the Pa = 0.75, and tacc = 1.33 clock cycles.

Figure 6.5

*NCSC Cores Connection*



**C0**

**C1**

**C2**

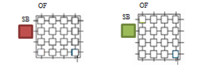
**C3**

Senario1: Core0 read core3 in the cluster 0 and Core 3 in cluster 2 read core 1 in cluster 3 simultaneously

Senario3: Core3 in cluster 3 and Core1 in cluster 2 access the same destination (core 3 in cluster 1) simultaneously

Senario2: Core3 in cluster 0 access Core1 in cluster 3, and Core3 in cluster 2 access Core0 in cluster 1, simultaneously

Senario4: Core3 in cluster 0 access Core1 in cluster 1.

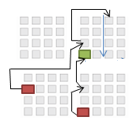
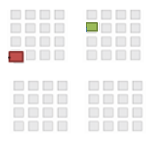


Read/Write MPCAM Direction

Core source

Core destination

Key:



## 6.4 Performance Analysis of NCSC

In order to analyze and assess the behavior of the proposed topology, two methods are used. The first method is based on using static network performance metrics (size, diameter, degree, connectivity, cost and bisection width) by derivation of a mathematical equation for each metric. Most famous network topology such as mesh, ring, tree, mesh of tree, hypercube and any other modified topology should have a known static network performance metrics, in order to compare it to any proposed topology. This is very important because these metrics usually reflect the dynamic metric of any INbefore implementation. For example, a small diameter means better latency and power consumption to the network. Hence, a smaller diameter is suitable. In the last few years, many researchers made a comparison study to evaluate the static performance of various topologies ([Abd-El-Barr & Al-Somani, 2011](#_18ewhd8); [Abdullah et al., 2011](#_1pgrrkc); [Alam & Varshney, 2015](#_49gfa85); [Awal et al., 2015](#_2olpkfy); [Ou et al., 2020](#_22vxnjd); [Sharma & Khilji, 2017](#_3sek011)).

The second method is called dynamic communication performance evaluation which is based on parameters such as latency, throughput, power consumption, etc. FPGA using INTEL Quartus prime simulator is needed to test the functionality of each part in NCSC and to evaluate the dynamic performance metrics. The time available in this thesis allowed the testing of the latency performance metric successfully.

### *6.4.1 Static Performance Analysis*

Some of the static performance metrics of the NCSC topology have been studied and compared with other most common topologies used in many-core systems including mesh, ring, tree, mesh of tree and hypercube. Several metrics are used to study the static performance of any network topology in a parallel system. The following metrics to study the NCSC (N, n) *IN* are used. Assume n=N for all metrics analysis.

1. Size: is the total number of nodes (cores) in the network, the size can increase the scalability of the network. The size of the NCSC is shown in the following equation:

………… (4)

In simple words, the size of this topology equals number of cores (n) in each cluster multiplied by number of clusters (N) in a system.

1. Diameter: is a main parameter that affects the network, it refers to the maximum distance between any two sources and destination cores through the network. Small and constant diameter means better latency and power consumption to the network ([Awal et al., 2015](#_2olpkfy)). Hence, a smaller diameter is desirable. The diameter of the NCSC is shown in the following equation:

………… (5)

Regardless of the number of cores and clusters in a system. The NCSC diameter is obtained by adding the diameter of MPCAM within the cluster to the inter-cluster diameter. In MPCAM, each core is directly connected to any other core by a broadcast bus which means that the diameter equals 1. Moreover, the diameter between the clusters equals one because the conjugate shuffle interconnection is connected by a bidirectional bus. Therefore, the diameter equals 2.

1. Degree: this parameter calculates the maximum number of links that are directly connected to any core in a topology. For INs, a constant node degree is preferred. It is easy to scale a network with a constant degree. Degree is formulated as shown in the following equation:

………… (6)

Each core is connected to other cores within the same cluster using two links on OF and SB; on the other hand, the conjugate core is connected to other clusters using two links. So, the maximum number of links connected to any core is 4 for conjugate cores.

1. Connectivity: is a standard for measuring number of paths between source and destination cores. Connectivity is formulated for NCSC as shown in the following equation:

*…….. (7)*

There are two paths between any two cores within the same cluster or across different clusters.

1. Cost: Though hardware devices determine the total cost of the INthat contains the number of routers and links. For comparative purposes, cost mostly refers to the total number of links needed to build the network. The cost of NCSC is shown in the following equation:

*………… (8)*

The cost depends on the number of cores in each cluster (n) and the number of clusters in the system (N). The number of links in each cluster and so, the number of total links within all clusters.

Number of total links between different clusters depends on n and N and is equal to the conjugate shuffle interconnection. Assuming that n=N, series analysis for n= {2, 3,) and referring to figure 6.2-a connection, inter-cluster links is equal to. So, the total links in the scheme are equal to.

1. Bisection width: this parameter refers to the minimum number of links that needed to be moved to divide the overall network into two equal parts. Low bandwidth between two portions is produced by small bisection width. A very large bisection width, on the other hand, requires a large number of wires for designing. As a result, large bisection width is preferred for all INs. The bisection width of the proposed system is shown by the following formula:

………… (9)

Dividing this scheme into two equal parts means having an equal number of clusters in each part; on the other hand, we should maintain the connection links in each part to be an independent part. It can be noticed that the bisection width can be calculated as the total number of links between different clusters is (), calculated in point 5, subtracted from them the number of links of the two new parts which is equal). For example, in a system with 10 clusters, the result after dividing the system into two parts is five clusters in each part. Therefore, by logic the bisection width is equal to the number of links in N=10 minus the links of two new parts each with N=5. So, Bisection width of = .

#### 6.4.1.1 Comparison and discussion of static performance metrics. In this section, a comparison study between the proposed scheme NCSC (N, n) and the other familiar topologies used in the parallel systems for evaluating the performance of the topological properties of NCSC is conducted. In addition, a study is made to see if it achieves the planned improvements.

We chose the following topologies: a 2-D mesh M (n, n), a tree T(h) with high h, a mesh of trees Mot(n, h) of n by n mesh and a tree of height h, and a hypercube Q(d) of dimension d. These topologies have been chosen because of their good properties and because they are used in most of many-core and parallel systems. Table 1 displays the topological properties of these networks. In order to best evaluate the NCSC (N, n), all of the mentioned properties of the proposed topology need to be computed with respect to different systems sizes and then compared with others topologies. This allows determining NCSC location relative to other INs. After that, we can judge if NCSC has some improvements over other Networks. Figures 6.6 to 6.10 show the comparative results.

Figure 6.6 depicts the diameter of the five studied topologies as their sizes increase. It is clear that NCSC has a constant diameter that is independent of its size and so it scores best while hypercube topology has the second rank and mesh topology is the worst in this regard. Therefore, the NCSC has the highest speed due to its small diameter. Because the latency and power consumption of an INis influenced by a number of factors, among them diameter. As a result, a network with a reduced diameter has the properties of a power-efficient network. Figure 6.7 depicts a comparison of the degree parameter between the five topologies. NCSC, mesh, tree and Mot topologies have a contiguous and constant degree. This means that they have the best capability for scaling the system to any number of cores without changing the old cores. In addition, there is no need in NCSC to rebuild the clusters or change the algorithms used by the compiler to schedule the tasks on the system. On the other hand, the hypercube topology has the worst degree which increases the cost and complexity when scaling up the system.

Figure 6.8 shows a comparison of the connectivity parameter between several INs. It shows that the NCSC has the same connectivity as mesh and Mot networks with constant connectivity of 2 between any two cores. Whereas, the hypercube has the highest connectivity which increases the cost as will be explained later. On the other hand, as the connectivity increases, the robustness of the network increases also. Both figure 6.9-a and figure 6.9-b show the cost of the five studied topologies; however, figure 6.9-a shows the cost when using a smaller number of cores. The hypercube has the highest cost while the tree topology has the cheapest cost relative to other topologies. In addition, the NCSC can be considered to have a mild cost that is often close to that of mesh. However, this parameter does not reflect the total cost in many-core systems because it depends on cost related only to the number of nodes and links without considering other factors such as power and area costs, which are very important issues to be considered. Anyway, nodes and links can be useful in another parallel system where the main goal is to produce high-performance systems bargained with cost. Figure 6.10 displays the bisection widths of the above topologies. Both the NCSC and hypercube topologies have the highest bisection width. On the other hand, Tree has the lowest bisection width that equals to one in all situations. In addition, it can be noted that Mesh and Mot have low values of bisection widths. The bisection width is quite important in INs because the reliability of a topology increases with it. Therefore, NCSC and hypercube have the best reliability.

As a result, it is obvious that NCSC has encouraging characteristics when compared with other well-known topologies, except for the links cost, which has a moderate cost, as we have mentioned earlier. Moreover, the cost of NoCs is mostly dependent on other parameters like the area and power consumption, which are related to the complex structure of network and usage of routers ([Hoskote et al., 2007](#_2qk79lc)). In contrast, the NCSC has no routers and arbiters needed in other topologies with free blocking or contention. So, it can be expected that the cost will be reduced if these parameters are taken into account.

Finally, in addition to the features presented in section 6.4.1, we can claim that this system is scalable and can be expanded to have N clusters without the need to redesign the cluster or change the connectivity program and the compiler. All that is needed to change the value of the “number of clusters” in the program. For example, in the case of N=n= 32 cores, a system of 2, 3, 4, …., N clusters can be produced as far as there is a silicone space on the chip. Furthermore, system latencies from 2 to N clusters, in both intra-cluster and inter-cluster, are constant and independent of the number of clusters because each cluster is connected only to its conjugate. This applies to any feasible value of N. However, the results can be extended to any homogeneous multi-core multi-cluster architecture.

*Table 6.1.*

*Properties Of Some INs.*

|  | | | Parameter/Properties | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Topology | | M  (n,n) | | T(h) | | Mot  (n,n) | | Q(d) | | NCSC(N,n) | |
|  | Size | | n2 | | 2h+1-1 | | 3n2-2n | | 2d | | n2 | |
|  | Diameter | | 2n-2 | | 2h | | 4 | | D | | 2 | |
|  | Degree | | 4 | | 3 | | 3 | | D | | 4 | |
|  | Connectivity | | 2 | | 1 | | 2 | | d | | 4 | |
|  | Cost | | 2n2-2n | | 2h+1-2 | | 4n2-4n | | d2d-1 | | 3n2-n | |
| Bisection  Width | | n | | 1 | | n | | 2d-1 | |  | |

Figure 6.6

*Diameter Of The Networks For Different Sizes.*

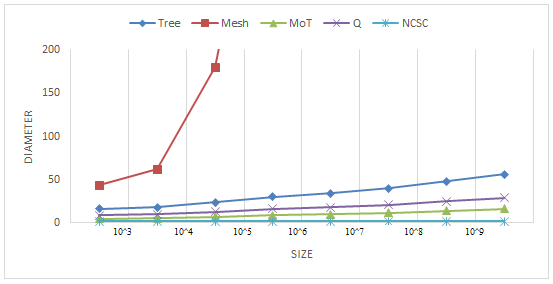


Figure 6.7

*Degree Of The Networks For Different Sizes.*

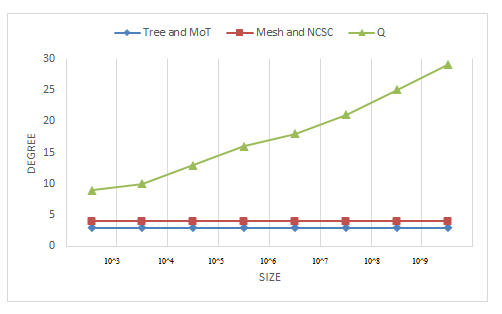


Figure 6.8

*Connectivity Of The Networks For Different Sizes*

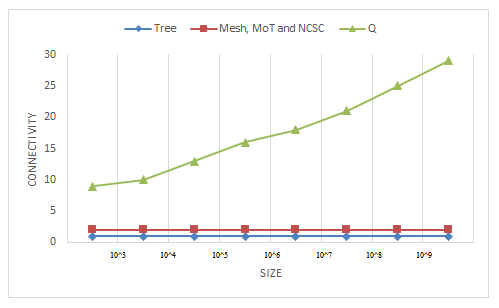
****

Figure 6.9-A

*Cost of The Networks For Different Sizes.*

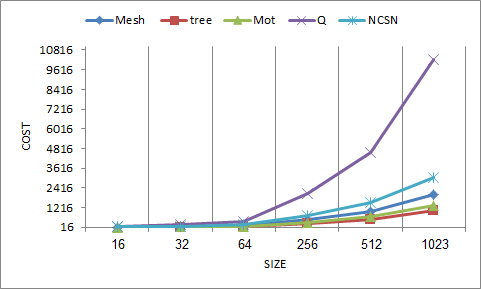


Figure 6.9-B

*Cost of The Networks For Different Sizes.*

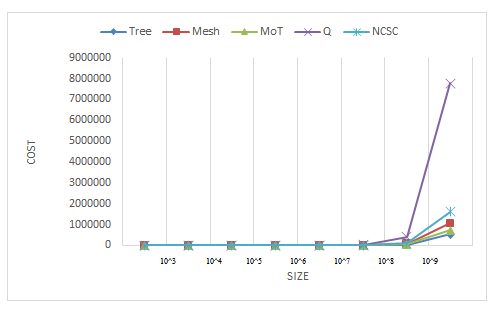
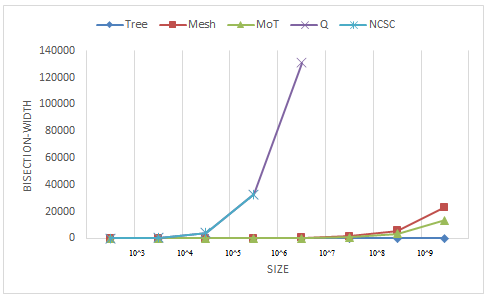
****

Figure 6.10

*Bisection-width Of The Networks For Different Sizes.*

****

### *6.4.2 Dynamic Performance Analysis*

As most of the research in this field, experimental method simulation tools are used. NCSC has been implemented, compiled, and simulated in a many-core system using Quartus Prime 20.1 which includes the Intel-supported ModelSim package and Nios II EDS for design and simulation ([Intel, 2021](#_41wqhpa)). NCSC was designed using the Cyclone IV-E FPGA device family, which has new attractive features especially in the number of inputs/outputs pins and power consumption ([Marouf et al., 2017](#_36os34g)). Sixteen cores were used to assess the latency of read and write operations with four cores in each cluster. Both schematic files and Verilog HDL code have been used to implement the NCSC in a multi-cluster system. ModelSim and VWF were used to verify and debug the files in both functional and timing simulations. A specific test bench was written to simulate and observe the latency of reading and writing operations.

**6.4.2.1 Functional Simulation.** The functionality of the NCSC INbased on multi-cluster architecture was achieved through the functional simulation. Multiple write and read operations by different cores where source and destination are in the same cluster, as well as simultaneous read and write operations by different cores where source and destination are in different clusters, and simultaneous read and write operations by different cores where sources are in different clusters but access the same core, were accomplished. The test-bench was written to evaluate all scenarios of NCSC multi-cluster architecture. It is used to analyze functional and timing simulations. The test-bench is detail described in figure 6.11.

Figure 6.11

*Test-benchmark Functions*

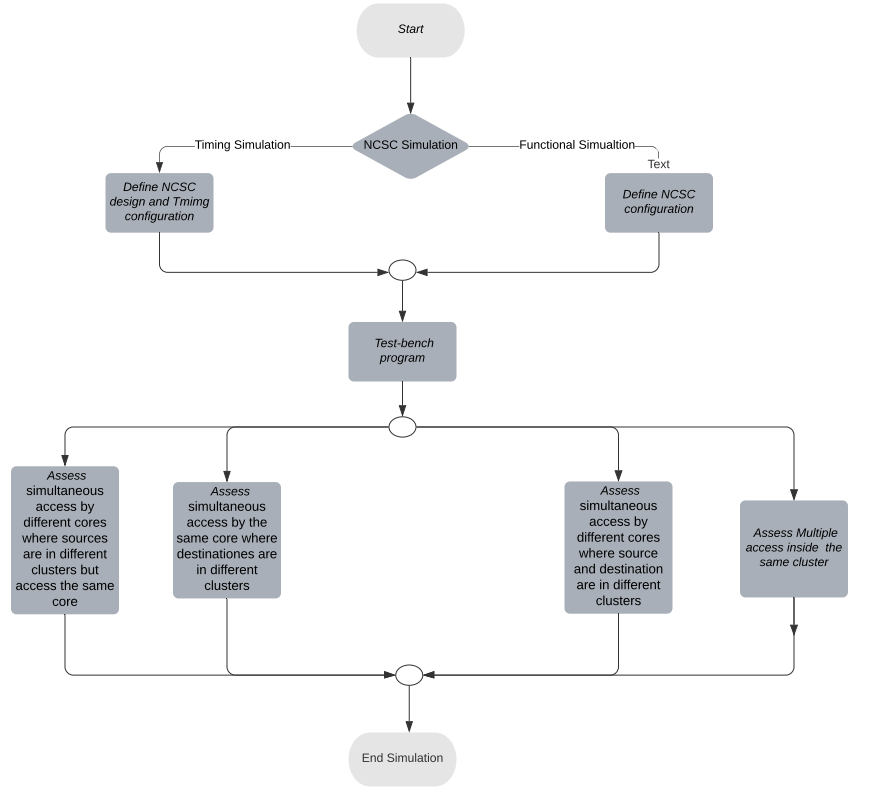


Figure 6.12 shows an image of several intervals in the functional simulation of NCSC, the clock period equals 10 ns for reading and writing. In the first interval (0 to 10 ns), both core 11 in the first cluster and core 22 in the second cluster broadcast their shared data with tags (A1) and (10C1) respectively. In the second interval (10 to 20 ns), core OF31 in the first cluster read shared data produced by core11 in the same cluster, and core OF44 in the fourth cluster read shared data produced by core22 from another cluster (in the second cluster) simultaneously. In the OF31, because the source and destination cores are in the same cluster, the communication process is followed using the MPCAM1 organization within the local cluster. Whereas OF44 does not belong to the same cluster of core22, the wanted shared data will not be found in the local cluster. So, by comparing the tags. OF44 communicates directly using the OF24 which is the conjugate of the second cluster through the NCSC to read internal MPCAM2. In this interval, the wanted shared data appeared on the Dout-core31 and Dout-core44 correctly. In the third Interval (20 to 30 ns), various cores from different clusters can read and write simultaneously, where core11 in the first cluster write its shared data with tags (A7), core OF23 in the third cluster read shared data with tags (A1) produced by another cluster (in the first cluster), and core OF44 in the fourth cluster read shared data with tags (A1) produced from the different cluster (in the first cluster) simultaneously. In this scenario, core11 broadcast its shared data to MPCAM1 in cluster one, and OF23 and OF 44 use OF13 and OF14 respectively which is the conjugate of these cores to cluster one. The output result appeared on the Dout-core23 and Dout-core44 correctly. Finally, in the fourth interval (30 to 40 ns), core OF44 in the fourth cluster read shared data with tags (A7) produced by another cluster (in the first cluster). Here, OF44 uses OF14 which is the conjugate of cluster one in the NCSC IN. The output result appeared on the Dout-core44 correctly.

Figure 6.13 shows other scenarios of NCSC. In the first interval (0 to 10 ns), both core 11 in the first cluster and core 42 in the second cluster broadcast their shared data with tags (A6) and (1071) respectively. In the second interval (10 to 20 ns), three cores from different clusters read simultaneously. Core OF11 read shared data previously produced by itself, core OF42 in the second cluster read shared data produced by core22, which means access from the same cluster, and core OF24 in the fourth cluster read shared data produced by core22, which means access from different clusters. In this scenario, core11 read shared data locally from the MPCAM1 organization. Whereas, competition will occur when the OF24 unit of core 2 in cluster 4 access column 4 in cluster 2 (OF42) while the OF42 is trying to read the shared data produced locally through the same column of the MPCAM2 in its cluster. As mentioned earlier, the access coming from the same cluster is given the higher priority, where the request coming from another cluster has to wait until the next clock cycle. In other words, the core and its conjugate could not access the same column in MPCAM simultaneously. The output result was successfully produced on the Dout-core11 and Dout-core42 in the current interval. The Dout-core24, on the other hand, gives the result on the third available interval (20 to 30 ns).

Figure 6.14 shows an image of several intervals for writing operation over NCSC, a new DI-core pin was added to monitor the writing operation. In the first interval (0 to 10 ns), core 21 in the first cluster, core 42 in the second cluster and core 24 in the fourth cluster write their shared data with tags (0211), (1422) and (3074) respectively, each to its MPCAM. It can be observed that the written data are stored correctly and displayed in DI-core21, DI-core42 and DI-core24. In the second interval (10 to 20 ns), core21, core42 and core 24 write the shared data to these MPCAM simultaneously, each with its special tags. The written data have appeared on the DI-core21, Dout-core42, and Dout-core42 correctly.

In the functional simulation, it can be noticed that written or read data appear on the DI-core and Dout-core pins without taking into account the delay produced by design components.

Figure 6.12

*Function Simulation For Read Operation Scenario 1*

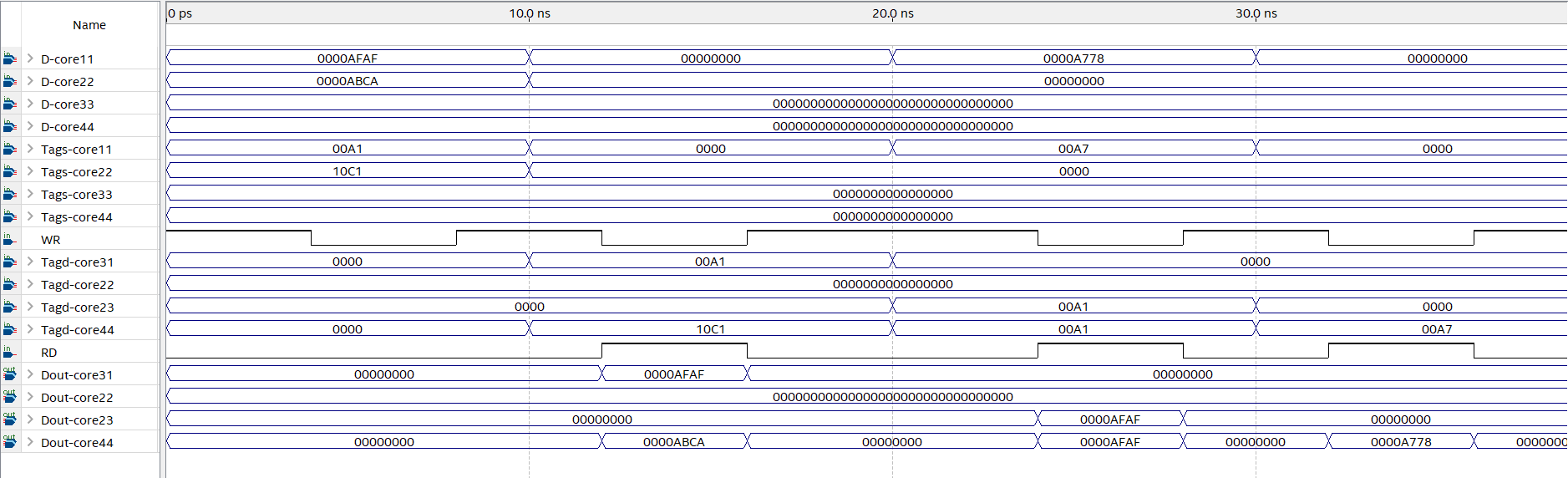


Figure 6.13

*Function Simulation For Read Operation Scenario 2*

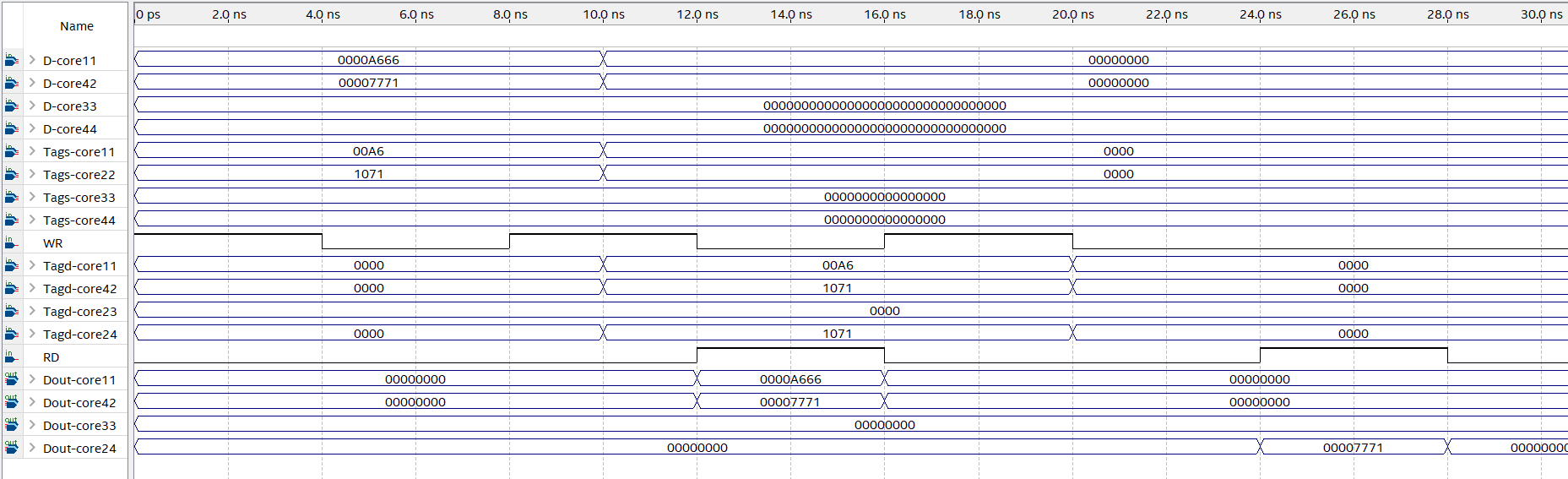
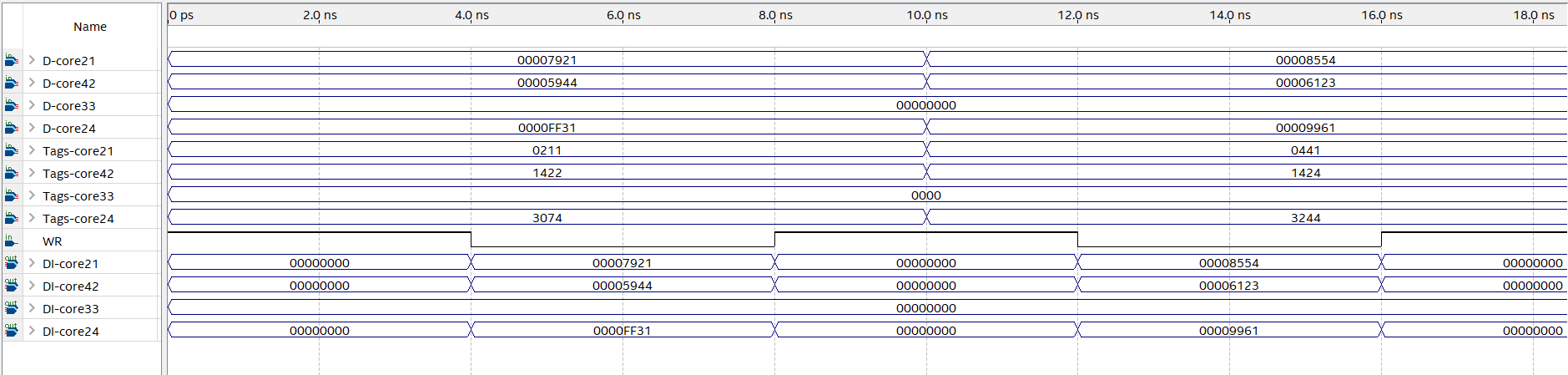


Figure 6.14

*Function Simulation For Write Operation*



#### 6.4.2.2 Latency Assessments. The Timing Analyzer tool is used to

#### evaluate the read/write latency for the proposed system. In this section, all scenarios that were presented in the functional simulation will be assessed in timing simulation. The timing simulation for both read and write operations of the proposed system is shown in Figures 6.15, 6.16 and 6.17.

Figure 6.15 shows the timing simulation of the proposed NCSC with several intervals. In the first interval (0 to 10 ns), both core 11 and core 22 write their shared data with tags (00A1) and (10C1) respectively. In the second interval (10 to 20 ns), core OF31 read shared data with tags (00A1) from the same cluster, and core OF44 read shared data with tags (10C1) from another cluster simultaneously. In this interval, the read latency differs between two cases. To read data that is already written in the same cluster, the tagd (00A1) which is provided by core31are simultaneously compared to the tags in MPCAM column within cluster one. The results appear on output buses (Dout-core 31) after delay time. The delay for read operation was assessed using an average of roughly one hundred intervals of test-benches, it was noticed that the delay for read operation within the same cluster is around 1.26226±0.090591 ns. On the other hand, to read data that is already written in another cluster, core44 wants to read data produced by core22. OF44 communicates directly using the OF24 which is the conjugate of the second cluster through the NCSC to read internal MPCAM2. The results appear on output buses (Dout-core 44) after delay time. Using an average of one hundred intervals of test-benches, it was noticed that the delay for read access between cores from different clusters is around 1.92738±0.139588. In the third Interval (20 to 30 ns), core11 in the first cluster write its shared data with tags (00A7), core 23 in the third cluster read shared data with tags (00A1) produced by core11, and core OF44 read shared data with tags (00A1) simultaneously. In this interval, core11 broadcast its shared data to MPCAM1 in cluster one, and OF23 and OF44 connect to core 11 in the first cluster using OF13 and OF14 respectively which is the conjugate of these cores to cluster one through the NCSC IN. The results appear on output buses (Dout-core 23) after delay time. It was noticed that the delay for read access between cores from different clusters is around 1.92738±0.139588 ns which is nearly equal to the latency of read operation in (Dout-core 44). In the third Interval (30 to 40 ns), the same scenario of interval (20 to 30 ns).

Figure 6.16 shows two scenarios. In the first interval (0 to 10 ns), core 11 and core 42 broadcasts a shared data to MPCAM1 and MPCAM2 respectively, each with its tags. In the second (10 to 20 ns) three cores (OF11, OF42 and OF24) simultaneously read the shared data. In the first scenario, OF11 and OF42, read the data that was written in its MPCAM with delay for read operation around 1.26226±0.090591 ns which indicates that the latency is equal to the latency of the read operation within the same cluster. On the other hand, As mentioned in functional simulation, competition occurs when the OF24 unit reads column 4 in cluster 2 (OF42) while the OF42 is trying to read the shared data produced locally in MPCAM2. In this case, the read operation waits until the next cycle to read from the MPCAM2 with a latency 11.92738±0.139588, that is common latency for the read data from another cluster plus the wait time.

Figure 6.17 shows an image of two intervals to assess the writing latency over NCSC. In the first interval (0 to 10 ns), core 21, core 42 and core 24 write their shared data with their tags (0211), (1422) and (3074) respectively, each to its MPCAM. It can be observed that the written data are stored in DI-core21 DI-core42 and DI-core24 pins after a delay time. It was noticed that the average delay for write operation between cores in NCSC is around 1.14785±0.04532 which is nearly equal to the latency of write operation in MPCAM organization. In the second interval (10 to 20 ns), core21, core42 and core 24 write the shared data to their MPCAM simultaneously with average delay for a write operation is around 1.15235±0.06132 which is identical to the latency in the previous interval.

Figure 6.15

*Timing Simulation For Read Operation Scenario 1*

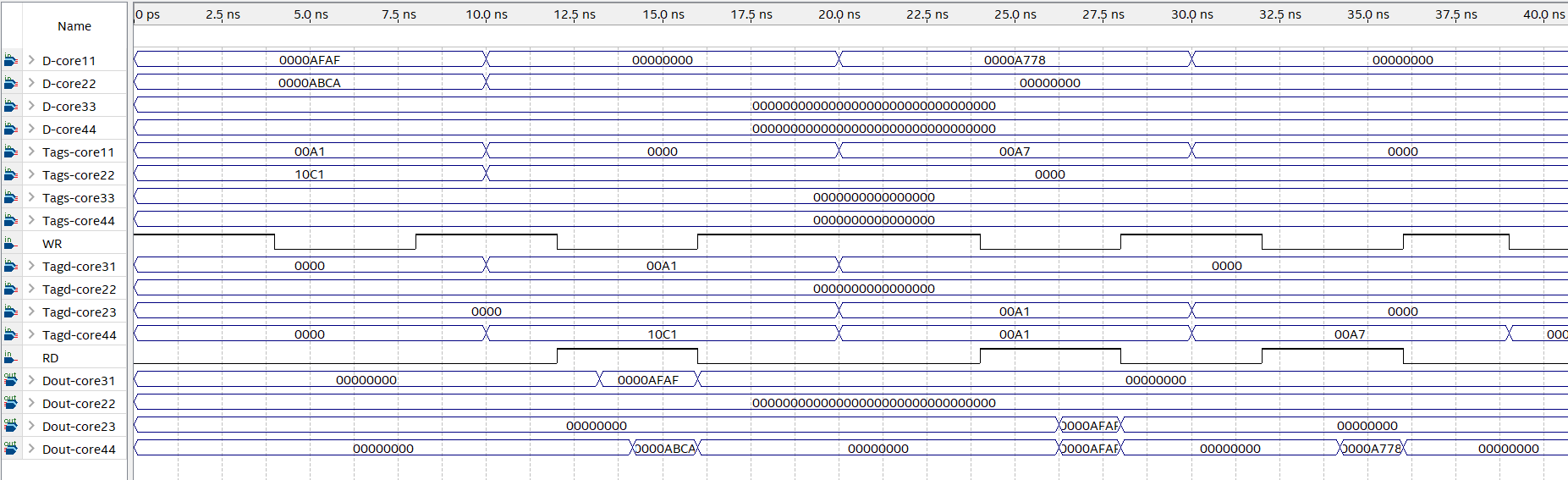


Figure 6.16

*Timing Simulation For Read Operation Scenario 2*

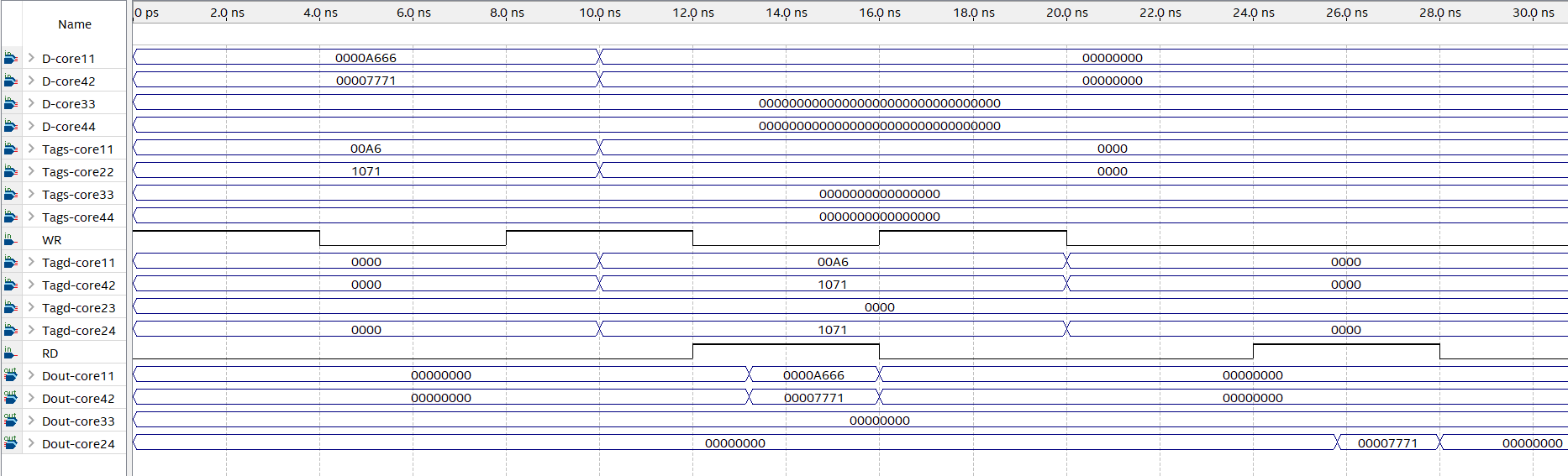
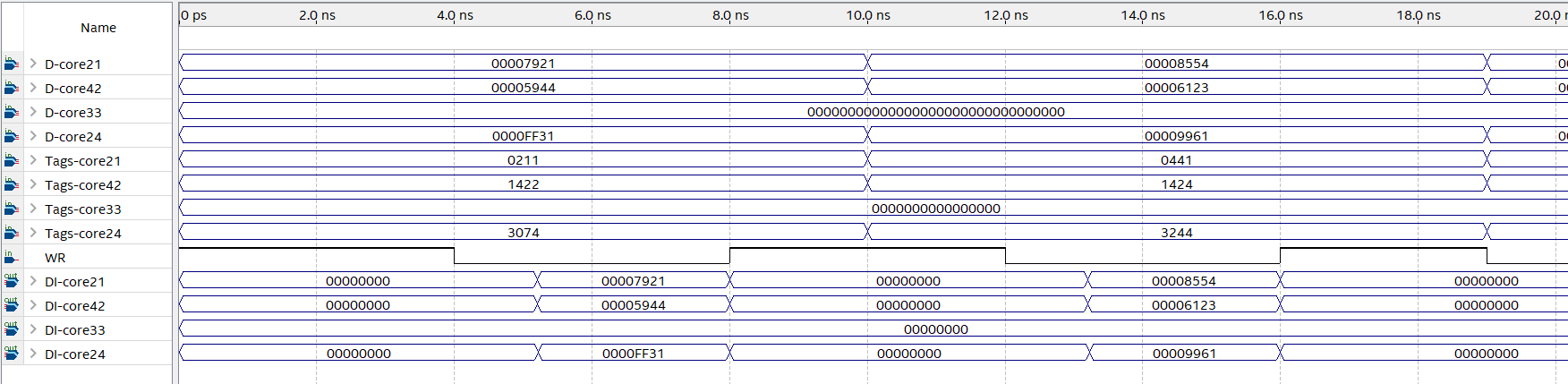


Figure 6.17

*Timing Simulation For Write Operation*



#### 6.4.2.3 NCSC Comparisons Statistics. SPSS and T-test were used to analyze and compare the latencies of the simulation scenarios. The comparisons were done by using T-test between read operations (Read\_within\_same\_cluster and Read\_between\_different\_clusters). Tab. 6.2 shows descriptive statistics for read latencies in NCSC.

The T-test showed that there is evidence that the read latency of Read\_within\_same\_cluster is less than the read latency of Read\_between-different\_clusters with 95% of confidence interval (Significance is less than 0.05). The Levene’s Test showed that there is evidence that variances of Read\_ within\_same\_cluster and Read\_between\_different\_clusters are not equal to each other with 95% of confidence interval (Significance is less than 0.05).

Tables 6.3 and 6.4 illustrate the t-test and Levene’s test comparisons for read latency within and between clusters. All descriptive statistics, t-test and the Levene’s test statistics are shown in appendix E.

*Table 6.2.*

*Descriptive Statistics For Read Latencies*

| Group | N | Mean | Std. Deviation | Std. error mean |
| --- | --- | --- | --- | --- |
| Read 1  Latency 2 | 100  100 | 1.26226  1.92738 | .090591  1.395884 | .009059  .139588 |

*Table 6.3.*

*T-test For Read Comparisons*

|  | | t-test for Equality of Means | | |
| --- | --- | --- | --- | --- |
| Std. Error Difference | 95% Confidence Interval of the Differences | |
| Lower | Upper |
| Read\_Comparisons | Equal variances assumed  Equal variances not assumed | . 139882  . 139882 | -.940965  -.942643 | -.389265  -.387587 |

Table 6.4.

Levens’s Test

|  | | Levene's Test for Equality of Variances | |
| --- | --- | --- | --- |
| F. | Sig. |
| Read\_Comparisons | Equal variances assumed  Equal variances not assumed | 5.964 | .015 |

# CHAPTER VII

# Conclusion

In this thesis, the author has tackled the two major issues that most affect the design of INin multi/many-core systems. The major elements in any INs are contention in shared cache architecture and arbitrator complexity, which ranges from a tiny control unit to a complicated router in modern NoCs. The main function of shared memory is for providing efficient communication between cores. There is a problem when multiple cores simultaneously access the same shared module. In multi/many-core systems, there are overheads due to inter-core or core to shared memory data transfer. Some of these overheads, like time consumed in arbitrating for multiple simultaneous requests, are subtle and often overlooked. On the other hand, the power consumption always increases as the complexity of the arbitrator increase. However, these overheads are important because they become large if a large number of cores are used in the system. In today's multi/many-core systems, the chip contains tens to hundreds of cores. The complexity of shared cache and routers increases as the number of cores in a many-core system grows, and the value of the INgrows as well to provide effective communication between these cores.

NCSC is the proposed scalable interconnection scheme that is based on a multi-cluster architecture. It offers great possibilities of avoiding INissues, this is because the NCSC uses a DPCAM architecture that allows two simultaneous access to each module and MPCAM organization that allows simultaneous access between all cores within a cluster without needs an arbitration as opposed to traditional interconnection scheme which waits for request and permission before accessing the inter-cores or cores to shared memory. Besides its MPCAM organization, NCSC is capable of accessing inter-cluster simultaneously using a simple conjugate shuffle interconnection. In both intra and inter-cluster communication, there is no need for arbitration and router devices.

As a part of this thesis, a new shared cache called DPCAM was built and successfully tested on Cyclone V FPGA. Implementing using a FPGA design was important to prove that the shared cache works correctly, to find out what practical problems exist with such architecture, and to work out possible improvements needed in building the prototype of the multi core system. In the following section, discussion the result of each part of the proposed scheme are presented. In section 7.2 potential applications of the system are identified. In section 7.3 future work is suggested.

## 7.1 Discussion Results

A DPCAM shared cache has been presented in chapter 3. This work has demonstrated that DPCAM can replace the shared cache in the memory hierarchy of a multi-core processor. This conclusion was drawn based on evaluating the design using the Cyclone V Intel FPGA. The latency of both read and write accesses and power characteristics of DPCAM have been investigated. The DPCAM achieves average 1.2±0.09138 ns latency for reading and 0.9679±0.0642 for writing operations which are clearly better than in other types of AM. Moreover, for the DPCAM the latency of a write operation was nearly constant for different memory sizes. On the other hand, DPCAM consumes about 7% more power than SA memory which can be reduced by some power-saving techniques.

In chapter 4, The DPCAM is embedded inside a multi-core system as shared L2 cache memory. Other necessary parameters were evaluated to study the DPCAM's behavior within the multi-core system. In addition, a comparative study of system performance in terms of IPC and miss rate between multi-core uses L2 associative shared cache and multi-core uses L2 based DPCAM shared cache has been implemented. The experimental results have shown that the performance of the proposed architecture in terms of IPC is improved by 8.7% for various types of benchmarks and the miss rate is decreased by about 13% except in sphinx3 and bzip2 benchmarks.

The MPCAM organization has been presented in chapter 5. This organization totally removes the contention access between shared memory and cores. There is no multicore INcontention and there is no shared memory interference. Because each core of the MPCAM can broadcast the shared data to all modules in its row, a copy of these data will be available in each OF unit, where it can be independently and simultaneously searched by any core. This conclusion was drawn based on evaluating the design using the Cyclone V Intel FPGA. The latency of read, write and simultaneous read/ write accesses have been tested. The MPCAM achieves average 1.27804±0.086823 ns latency for reading, 1.084115±0.03384 ns for writing operations, for simultaneous write and read operations to the different memory locations the latency equals 1.1909±0.02363 ns and 1.3105±0.091955 ns for simultaneous write and read operations to the same memory locations. The MPCAM organization demonstrated to be scalable to a many-core system without obstacles in communication between cores and shared memory.

A scalable topology of the many-core processor systems based multi-cluster architecture called NCSC was presented in chapter 6. It has desirable static metrics in terms of diameter, degree, connectivity, cost, latency, scalability and reliability. The main additional features of using this topology can be summarized as follows. First, it has high scalability with no need to redesign the cluster or modify compiler techniques to schedule the tasks. Second, it has a fixed latency on the intra-cluster and inter-cluster levels. Third, eliminates the need for routers and arbiters and solves the problem of simultaneous access to the shared cache. NCSC has been implemented using the Cyclone IV-E FPGA device family. The latency of read and write by multiple cores within the cluster and between clusters have been assessed. NCSC provides non-blocking access between cores with average latency for write access within the same cluster is around 1.14785±0.04532 ns which is nearly equal to the latency of write operation in MPCAM, the average read latency within the same cluster is 1.26226±0.090591 ns and the delay for read access between cores from different clusters is around 1.92738±0.139588 ns.

## 7.2 Potential Applications

In addition to MIMD architecture, the proposed NCSC-based many-core system seems to be most advantageous in MISD computation ([Batabyal & Sarkar, 2019](#_3k3xz3h)). Pipeline architectures belong to the MISD system, in spite, the datum in the pipeline is constantly changing after each phase of processing. In MISD computation, a single datum is required by more than one program stream to complete its computation. If a primary shared datum is required by more than one program stream, it is recommended to provide each of these streams with a copy of this datum. ​If such a datum can only be produced at the run time by program stream, the MPCAM organization represents the fastest method to provide other program streams with a copy of this datum simultaneously, no matter how many streams need that datum.

Another field in which MPCAM and NCSC-based many-core systems would be of great advantage is controlling systems with a large number of fast changing parameters, like aeronautical systems. Usually, in the control system, a process which controls a certain parameter needs to know what effects have been created in other functions due to the changes in other parameters in order to take the correct decision. If time is critical in passing the results of these effects, a fast communication medium which can transfer each of these results to all destinations simultaneously, like the MPCAM organization, would be an ideal solution.

## 7.3 Future Work

NCSC-based many-core system, as shown throughout the thesis, is a promising development for use in massively parallel processing systems. This system is modular and expandable without the risk of large overhead, especially in access latency.

As future work, more research can be conducted on the NCSC topology. Other crucial dynamic performance metrics like throughput, latency overhead, area analysis, and power consumption should be evaluated. In addition, all components of this topology, including core interfaces, must be built to ensure the authenticity of NCSC within a many-core system.

The potential application of this system can be thought of as projects only after the NCSC-based many-core system is well established.

# References

Abd-El-Barr, M., & Al-Somani, T. F. (2011). Topological properties of hierarchical interconnection networks: a review and comparison. *Journal of Electrical and Computer Engineering*, *2011*.

Abdullah, M., Abuelrub, E., & Mahafzah, B. (2011). The chained-cubic tree interconnection network. *Int. Arab J. Inf. Technol.*, *8*(3), 334-343.

Abumwais, A., Amirjanov, A., Uyar, K., & Eleyat, M. (2022). Dual-Port Content Addressable Memory for Cache Memory Applications. *Computers, Materials \& Continua*, *70*(3), 4583--4597. <http://www.techscience.com/cmc/v70n3/44969>

Abumwais, A., & Ayyad, A. (2018). The MPCAM Based Multi-core Processor Architecture: A Contention Free Architecture.

Adegbija, T., & Gordon-Ross, A. (2017). PhLock: A cache energy saving technique using phase-based cache locking. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *26*(1), 110-121.

Agarwal, V., Petrini, F., Pasetto, D., & Bader, D. A. (2010). Scalable graph exploration on multicore processors. SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis,

Agulleiro, J.-I., & Fernandez, J.-J. (2015). Tomo3D 2.0–exploitation of advanced vector extensions (AVX) for 3D reconstruction. *Journal of structural biology*, *189*(2), 147-152.

Ahmed, A., Park, K., & Baeg, S. (2016). Resource-efficient SRAM-based ternary content addressable memory. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *25*(4), 1583-1587.

Al-Ali, F., & Jesshope, C. (2000). Survey of High-Latency Tolerance in Future Microprocessor Architectures. Proceedings of the 4th New Zealand Computer Science Research Students' Conference (NZCSRSC2001),

Alam, M., & Varshney, A. K. (2015). A comparative study of interconnection network. *International Journal of Computer Applications*, *127*(4), 37-43.

Alazemi, F., Azizimazreah, A., Bose, B., & Chen, L. (2018). Routerless network-on-chip. 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA),

Anthony, S. (2013). Intel unveils 72-core x86 Knights Landing CPU for exascale supercomputing. In: ExtremeTech.

Archibald, J., & Baer, J.-L. (1986). Cache coherence protocols: Evaluation using a multiprocessor simulation model. *ACM Transactions on Computer Systems (TOCS)*, *4*(4), 273-298.

Awal, M. R., Rahman, M. H., Mohd Nor, R., Bin Tengku Sembok, T. M., & Akhand, M. (2015). Architecture and network-on-chip implementation of a new hierarchical interconnection network. *Journal of Circuits, Systems and Computers*, *24*(02), 1540006.

Ayyad, A. (2005). The Penta-S: A Scalable Crossbar Network for Distributed Shared Memory Multiprocessor Systems.

Ayyad, A., Exman, I., Land, M., & Rudolph, L. (1996). An Experimental Cross-Bar Switch For Support Of Collective Communications In Parallel Processing. Proceedings of the Nineteenth Convention of IEEE in Israel,

Batabyal, S., & Sarkar, K. (2019). Realizing parallelism in quantum MISD architecture. Proceedings of the 16th ACM International Conference on Computing Frontiers,

Bell, S., Edwards, B., Amann, J., Conlin, R., Joyce, K., Leung, V., . . . Brown, J. (2008). Tile64-processor: A 64-core soc with mesh interconnect. 2008 IEEE International Solid-State Circuits Conference-Digest of Technical Papers,

Bhattacharya, D., Bhoj, A. N., & Jha, N. K. (2014). Design of efficient content addressable memories in high-performance FinFET technology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *23*(5), 963-967.

Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., . . . Sardashti, S. (2011). The gem5 simulator. *ACM SIGARCH Computer Architecture News*, *39*(2), 1-7.

Bohm, A., Gurd, J. R., & Teo, Y. M. (1989). The effect of iterative instructions in dataflow computers. 1989 International Conference on Parallel Processing, University Park, PA,

Bohnenstiehl, B., Stillmaker, A., Pimentel, J., Andreas, T., Liu, B., Tran, A., . . . Baas, B. (2016). A 5.8 pj/op 115 billion ops/sec, to 1.78 trillion ops/sec 32nm 1000-processor array. 2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits),

Cargnini, L. V., Torres, L., Brum, R. M., Senni, S., & Sassatelli, G. (2014). Embedded memory hierarchy exploration based on magnetic random access memory. *Journal of Low Power Electronics and Applications*, *4*(3), 214-230.

Cebrian, J. M., Natvig, L., & Jahre, M. (2020). Scalability analysis of AVX-512 extensions. *The journal of supercomputing*, *76*(3), 2082-2097.

Chauan, P., Singh, G., & Singh, G. (2015). Cache Controller for 4-way Set-Associative Cache Memory. *International Journal of Computer Applications*, *975*, 8887.

Chen, X., Chen, S., Lu, Z., & Jantsch, A. (2011). Hybrid distributed shared memory space in multi-core processors. *Journal of Software*, *6*(12 SPEC. ISSUE), 2369-2378.

Chen, X., Lu, Z., Jantsch, A., Chen, S., Guo, Y., Chen, S., & Chen, H. (2015). Performance Analysis of Homogeneous On-Chip Large-Scale Parallel Computing Architectures for Data-Parallel Applications. *Journal of Electrical and Computer Engineering*, *2015*, 902591. https://doi.org/10.1155/2015/902591

Cheriton, D. R. (2015). Hierarchical associative memory-based classification system. In: Google Patents.

Chrysos, G. (2012). Intel® xeon phi coprocessor (codename knights corner). 2012 IEEE Hot Chips 24 Symposium (HCS),

Com, T., Quadri, I.-R., Boulet, P., & Dekeyser, j.-l. (2007). Modeling of Topologies of Interconnection Networks based on Multidimensional Multiplicity.

Cota, É., de Morais Amory, A., & Lubaszewski, M. S. (2011). *Reliability, Availability and Serviceability of Networks-on-chip*. Springer Science & Business Media.

Cutress, I. (2017). Intel Lists Knights Mill Xeon Phi on ARK: Up to 72 cores at 320W with QFMA and VNNI. In.

Damkroger, T. (2017). Unleashing high-performance computing today and tomorrow. *Intel IT Peer Network*.

Daneshtalab, M., Ebrahimi, M., Liljeberg, P., Plosila, J., & Tenhunen, H. (2010). CMIT—A novel cluster-based topology for 3D stacked architectures. 2010 IEEE International 3D Systems Integration Conference (3DIC),

Danowitz, A., Kelley, K., Mao, J., Stevenson, J. P., & Horowitz, M. (2012). CPU DB: recording microprocessor history. *Communications of the ACM*, *55*(4), 55-63.

Das, S., & Kapoor, H. K. (2016). Towards a better cache utilization by selective data storage for CMP last level caches. 2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID),

DOU, Y., WANG, J., SU, H., XU, C., GONG, X., YANG, W., . . . YU, G. (2020). The impact of data flow computing thinking on the development of computer architecture. *SCIENTIA SINICA Informationis*, *50*(11), 1697.

Duran, A., & Klemm, M. (2012). The Intel® many integrated core architecture. 2012 International Conference on High Performance Computing & Simulation (HPCS),

Edwards, R. (1986). Futurebus—the independent standard for 32-bit systems. *Microprocessors and Microsystems*, *10*(2), 65-68.

Emani, M., Vishwanath, V., Adams, C., Papka, M. E., Stevens, R., Florescu, L., . . . Sujeeth, A. (2021). Accelerating scientific applications with sambaNova reconfigurable dataflow architecture. *Computing in Science & Engineering*, *23*(2), 114-119.

Fallin, C., Nazario, G., Yu, X., Chang, K., Ausavarungnirun, R., & Mutlu, O. (2012). MinBD: Minimally-buffered deflection routing for energy-efficient interconnect. 2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip,

Feeley, M. J., Morgan, W. E., Pighin, E. P., Karlin, A. R., Levy, H. M., & Thekkath, C. A. (1995). Implementing global memory management in a workstation cluster. *SIGOPS Oper. Syst. Rev.*, *29*(5), 201–212. https://doi.org/10.1145/224057.224072

Fischer, K., Chang, H., Ingerly, D., Jin, I., Kilambi, H., Longun, J., . . . Plekhanov, P. (2016). Performance enhancement for 14nm high volume manufacturing microprocessor and system on a chip processes. 2016 IEEE International Interconnect Technology Conference/Advanced Metallization Conference (IITC/AMC),

Flynn, M. J. (1966). Very high-speed computing systems. *Proceedings of the IEEE*, *54*(12), 1901-1909.

Francalanci, C., & Giacomazzi, P. (2005). High-performance self-routing algorithm for multiprocessor systems with shuffle interconnections. *IEEE Transactions on Parallel and Distributed Systems*, *17*(1), 38-50.

Gardner, E. (2014). What public disclosures has Intel made about Knights Landing? *Intel Corporation*.

Hamid, N., Walters, R. J., & Wills, G. B. (2015). An architecture for measuring network performance in multi-core multi-cluster architecture (MCMCA). *International Journal of Computer Theory and Engineering*, *7*(1), 57-61.

Hennessy, J. L., & Patterson, D. A. (2019). A new golden age for computer architecture. *Communications of the ACM*, *62*(2), 48-60.

Henning, J. L. (2006). SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, *34*(4), 1-17.

Hoskote, Y., Vangal, S., Singh, A., Borkar, N., & Borkar, S. (2007). A 5-GHz mesh interconnect for a teraflops processor. *IEEE Micro*, *27*(5), 51-61.

Hruska, J. (2018). How L1 and L2 CPU Caches Work, and Why They’re an Essential Part of Modern Chips. *Extremetech, August*, *30*.

Huang, S., Wei, Q., Feng, D., Chen, J., & Chen, C. (2016). Improving flash-based disk cache with lazy adaptive replacement. *ACM Transactions on Storage (TOS)*, *12*(2), 1-24.

Imani, M., Gupta, S., Kim, Y., Zhou, M., & Rosing, T. (2019). Digitalpim: Digital-based processing in-memory for big data acceleration. Proceedings of the 2019 on Great Lakes Symposium on VLSI,

Imani, M., Peroni, D., Rahimi, A., & Rosing, T. S. (2016). Resistive cam acceleration for tunable approximate computing. *IEEE Transactions on Emerging Topics in Computing*, *7*(2), 271-280.

Intel. (2018). *Cyclone V device overview*. I. Corporation.

Intel. (2021). *Intel® Quartus® Prime Pro Edition User Guide, Timing Analyzer tool*. I. Corporation.

Irfan, M., Cheung, R. C., & Ullah, Z. (2019). High-Throughput Re-configurable content-addressable memory on FPGAs. Proceedings of the 2019 International Conference on Information Technology and Computer Communications,

Irfan, M., & Ullah, Z. (2017). G-AETCAM: Gate-based area-efficient ternary content-addressable memory on FPGA. *IEEE Access*, *5*, 20785-20790.

Irfan, M., Ullah, Z., & CC Cheung, R. (2019). Zi-CAM: a power and resource efficient binary content-addressable memory on FPGAs. *Electronics*, *8*(5), 584.

Johnson, D., Johnson, M., Kelm, J., Tuohy, W., Lumetta, S., & Patel, S. (2011). Rigel: A 1,024-core single-chip accelerator architecture. *IEEE Micro*, *31*(4), 30-41.

Kamal, R., & Yadav, N. (2012). NoC and Bus architecture: A comparison. *International Journal of Engineering Science and Technology*, *4*(4), 1438-1442.

Kaplan, I. (1987). The LDF 100: a large grain dataflow parallel processor. *SIGARCH Comput. Archit. News*, *15*(3), 5–12. https://doi.org/10.1145/25059.25061

Karam, R., Puri, R., Ghosh, S., & Bhunia, S. (2015). Emerging trends in design and applications of memory-based computing and content-addressable memories. *Proceedings of the IEEE*, *103*(8), 1311-1330.

Khanna, G., Mishra, R., & Chaturvedi, S. K. (2017). Design of Fault Tolerant Shuffle Exchange Gamma Interconnection Network Layouts. *Journal of Interconnection Networks*, *17*(02), 1750005.

Kilgariff, E., Moreton, H., Stam, N., & Bell, B. (2018). NVIDIA Turing Architecture In-Depth. *URL: https://devblogs. nvidia. com/nvidia-turing-architecture-indepth/(visited on 2020-05-11)*.

Kistler, M., Perrone, M., & Petrini, F. (2006). Cell multiprocessor communication network: Built for speed. *IEEE Micro*, *26*(3), 10-23.

Lahdhiri, H., Lorandel, J., Monteleone, S., Bourdel, E., & Palesi, M. (2020). Framework for Design Exploration and Performance Analysis of RF-NoC Manycore Architecture. *Journal of Low Power Electronics and Applications*, *10*(4), 37.

Levchenko, N., Okunev, A., & Zmejev, D. (2018). Solutions to Problem of CAM Overflow in the Parallel Dataflow Computing System “Buran”. 2018 IEEE East-West Design & Test Symposium (EWDTS),

Li, S., Ahn, J. H., Strong, R. D., Brockman, J. B., Tullsen, D. M., & Jouppi, N. P. (2009). McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture,

Li, W., Guo, B., Li, X., Yin, S., Zhou, Y., & Huang, S. (2017). Nesting ring architecture of multichip optical network on chip for many-core processor systems. *Optical Engineering*, *56*(3), 035106.

Li, W., Guo, B., Li, X., Zhou, Y., Huang, S., & Rouskas, G. N. (2019). A large-scale nesting ring multi-chip architecture for manycore processor systems. *Optical Switching and Networking*, *31*, 183-192.

Lin, T.-R., Penney, D., Pedram, M., & Chen, L. (2020). A deep reinforcement learning framework for architectural exploration: A routerless noc case study. 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA),

Liu, S., Chen, T., Li, L., Feng, X., Xu, Z., Chen, H., . . . Chen, Y. (2015). IMR: High-performance low-cost multi-ring NoCs. *IEEE Transactions on Parallel and Distributed Systems*, *27*(6), 1700-1712.

Loi, I., & Benini, L. (2014). A multi banked—Multi ported—Non blocking shared L2 cache for MPSoC platforms. 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE),

Luo, J.-Y., Cheng, H.-Y., Lin, C., & Chang, D.-W. (2019). TAP: Reducing the energy of asymmetric hybrid last-level cache via thrashing aware placement and migration. *IEEE Transactions on Computers*, *68*(12), 1704-1719.

Marouf, I., Asad, M. M., Bakhuraibah, A., & Al-Haija, Q. A. (2017). Cost analysis study of variable parallel prefix adders using altera cyclone IV FPGA kit. 2017 International Conference on Electrical and Computing Technologies and Applications (ICECTA),

Mars, J., Hundt, R., & Vachharajani, N. A. (2016). Cache contention management on a multicore processor based on the degree of contention exceeding a threshold. In: Google Patents.

Martin, M. M., Hill, M. D., & Sorin, D. J. (2012). Why on-chip cache coherence is here to stay. *Communications of the ACM*, *55*(7), 78-89.

Martyshkin, A., Salnikov, I., Pashchenko, D., & Trokoz, D. (2018). Associative Co-processor on the Basis of Programmable Logical Integrated Circuits for Special Purpose Computer Systems. 2018 Global Smart Industry Conference (GloSIC),

Mittal, S. (2014). A New Approach to Directory Based Solution for Cache Coherence Problem. 2014 3rd International Conference on Eco-friendly Computing and Communication Systems,

Mohammed, M. S., & Abandah, G. A. (2016). CHARACTERIZATION OF SHARED-MEMORY MULTI-CORE APPLICATIONS. *Jordanian Journal of Computers and Information Technology (JJCIT)*, *2*(1).

Monazzah, A. M. H., Farbeh, H., & Miremadi, S. G. (2016). LER: Least-error-rate replacement algorithm for emerging STT-RAM caches. *IEEE Transactions on Device and Materials Reliability*, *16*(2), 220-226.

Mudge, T. N., Hayes, J. P., Buzzard, G. D., & Winsor, D. C. (1986). Analysis of multiple-bus interconnection networks. *Journal of parallel and distributed computing*, *3*(3), 328-343.

Mulnix, D. (2017). Intel Xeon processor scalable family technical overview. *Intel Corp*.

Murali, S., Atienza, D., Meloni, P., Carta, S., Benini, L., De Micheli, G., & Raffo, L. (2007). Synthesis of predictable networks-on-chip-based interconnect architectures for chip multiprocessors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *15*(8), 869-880.

Naderializadeh, N., Maddah-Ali, M. A., & Avestimehr, A. S. (2017). Fundamental limits of cache-aided interference management. *IEEE Transactions on Information Theory*, *63*(5), 3092-3107.

Nakaike, T., Odaira, R., Gaudet, M., Michael, M. M., & Tomari, H. (2015). Quantitative comparison of hardware transactional memory for Blue Gene/Q, zEnterprise EC12, Intel Core, and POWER8. *ACM SIGARCH Computer Architecture News*, *43*(3S), 144-157.

Nguyen, X.-T., Hoang, T.-T., Nguyen, H.-T., Inoue, K., & Pham, C.-K. (2018). An efficient I/O architecture for RAM-based content-addressable memory on FPGA. *IEEE Transactions on Circuits and Systems II: Express Briefs*, *66*(3), 472-476.

Nowatzki, T., Gangadhar, V., Ardalani, N., & Sankaralingam, K. (2017). Stream-dataflow acceleration. 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA),

NVIDIA, T. (2017). V100 GPU architecture. *The world's most advanced data center GPU. Version WP-08608-001\_v1*, *1*.

Ofori-Attah, E., Bhebhe, W., & Agyeman, M. O. (2017). Architectural techniques for improving the power consumption of noc-based cmps: A case study of cache and network layer. *Journal of Low Power Electronics and Applications*, *7*(2), 14.

Olanrewaju, R. F., Baba, A., Khan, B. U. I., Yaacob, M., Azman, A. W., & Mir, M. S. (2016). A study on performance evaluation of conventional cache replacement algorithms: A review. 2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC),

Olofsson, A., Nordström, T., & Ul-Abdin, Z. (2014, 2-5 Nov. 2014). Kickstarting high-performance energy-efficient manycore architectures with Epiphany. 2014 48th Asilomar Conference on Signals, Systems and Computers,

Ou, Y., Agwa, S., & Batten, C. (2020). Implementing low-diameter on-chip networks for manycore processors using a tiled physical design methodology. 2020 14th IEEE/ACM International Symposium on Networks-on-Chip (NOCS),

Oveis-Gharan, M., & Khan, G. N. (2018). Flexible Reconfigurable On-chip Networks for Multi-core SoCs. Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies,

Pan, A., & Pai, V. S. (2015). Runtime-driven shared last-level cache management for task-parallel programs. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis,

Papagiannopoulou, D., Marongiu, A., Moreshet, T., Benini, L., Herlihy, M., & Bahar, R. I. (2018). Hardware transactional memory exploration in coherence-free many-core architectures. *International Journal of Parallel Programming*, *46*(6), 1304-1328.

Park, J., Lee, M., Kim, S., Ju, M., & Hong, J. (2019). MH cache: A multi-retention STT-RAM-based low-power last-level cache for mobile hardware rendering systems. *ACM Transactions on Architecture and Code Optimization (TACO)*, *16*(3), 1-26.

Patterson, D. A., & Hennessy, J. L. (2020). *Computer Organization and Design The Hardware Software Interface* (Second Edition ed.). Morgan kaufmann.

Pereira, R. V. M., Seman, L. O., Berejuck, M. D., Melo, D. R. D., Morales, A. S., & Bezerra, E. A. (2021). MPI hardware framework for many-core based embedded systems. *International Journal of Sensor Networks*, *35*(1), 42-56.

Prakash, T. K., & Peng, L. (2008). Performance characterization of spec cpu2006 benchmarks on intel core 2 duo processor. *ISAST Trans. Comput. Softw. Eng*, *2*(1), 36-41.

Prasanna Venkatesan, G. Content Addressable Memory with Efficient Power Consumption and Throughput. *International journal of Emerging Trends in Science and Technology, Volume01, Issue03*, 399-404.

Priya, B. K., Kumar, S., Begum, B. S., & Ramasubramanian, N. (2019). Cache lifetime enhancement technique using hybrid cache-replacement-policy. *Microelectronics Reliability*, *97*, 1-15.

Protic, J., Tomasevic, M., & Milutinovic, V. (1996). Distributed Shared Memory: Concepts and Systems. *IEEE Parallel Distrib. Technol.*, *4*(2), 63–79. https://doi.org/10.1109/88.494605

Qodirov, F., Mansurova, Z., Jurayev, S., & Sayfullayev, B. (2019). FEATURES OF INTEL CORE i9 X-SERIES PROCESSORS AND ITS ADVANTAGE FROM OTHER PROCESSORS. ПУТИ ПОВЫШЕНИЯ РЕЗУЛЬТАТИВНОСТИ СОВРЕМЕННЫХ НАУЧНЫХ ИССЛЕДОВАНИЙ,

Quartus, I. (2015). handbook volume 3: Verification. *Source:〈 https://www. altera. com/content/dam/alterawww/global/en\_US/pdfs/literature/hb/qts/qts\_qii5v3. pdf*.

Rossi, D., Tenentes, V., Reddy, S. M., Al-Hashimi, B. M., & Brown, A. (2017). Exploiting aging benefits for the design of reliable drowsy cache memories. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *37*(7), 1345-1357.

Roy, A., Xu, J., & Chowdhury, M. H. (2008). Multi-core processors: A new way forward and challenges. 2008 International Conference on Microelectronics,

Ruaro, M., Velloso, N., Jantsch, A., & Moraes, F. G. (2019). Distributed SDN architecture for NoC-based many-core SoCs. Proceedings of the 13th IEEE/ACM International Symposium on Networks-on-Chip,

Schauer, B. (2008). Multicore processors–a necessity. *ProQuest discovery guides*, 1-14.

Seo, Y., Kwon, K.-W., Fong, X., & Roy, K. (2016). High performance and energy-efficient on-chip cache using dual port (1R/1W) spin-orbit torque MRAM. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, *6*(3), 293-304.

Sharma, A., & Khilji, N. (2017). Comparative study of interconnect network on chip topology in multicore processors. 2017 International Conference on Information, Communication, Instrumentation and Control (ICICIC),

Singh, A. K., Shafique, M., Kumar, A., & Henkel, J. (2013). Mapping on multi/many-core systems: Survey of current and emerging trends. 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC),

Skhiri, R., Fresse, V., Jamont, J. P., Suffran, B., & Malek, J. (2019). From FPGA to support cloud to cloud of FPGA: State of the art. *International Journal of Reconfigurable Computing*, *2019*.

Stallings, W. (2013). Computer Organization and Architecture. Designing for Performance 9th Edition Ed. In: Pearson Education International, Inc. Wu, H.-K., SW-Y. Lee, H.-Y. Chang & J ….

Stenstrom, P. (1988). Reducing contention in shared-memory multiprocessors. *Computer*, *21*(11), 26-37.

Sun, C.-M., Hung, C.-N., Huang, H.-M., Hsu, L.-H., & Jou, Y.-D. (2007). Hamiltonian laceability of faulty hypercubes. *Journal of Interconnection Networks*, *8*(02), 133-145.

Syed, F., & Ullah, Z. (2017). Updating algorithm for SRAM-based TCAM and its implementation on FPGA. *International Journal of Computer Science and Information Security*, *15*(1), 116.

Syed, F., Ullah, Z., & Jaiswal, M. K. (2017). Fast content updating algorithm for an SRAM-based TCAM on FPGA. *IEEE Embedded Systems Letters*, *10*(3), 73-76.

Teifel, J., & Manohar, R. (2004). An asynchronous dataflow FPGA architecture. *IEEE Transactions on Computers*, *53*(11), 1376-1392.

Thomadakis, M. E. (2011). The architecture of the Nehalem processor and Nehalem-EP SMP platforms. *Resource*, *3*(2), 30-32.

Torres, G. (2010). Inside the AMD Bulldozer Architecture. *hardwaresecrets. com, retrieved December*, *16*.

Udipi, A. N., Muralimanohar, N., & Balasubramonian, R. (2010). Towards scalable, energy-efficient, bus-based on-chip networks. HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture,

Ullah, I., Ullah, Z., & Lee, J.-A. (2018). Ee-tcam: An energy-efficient sram-based tcam on fpga. *Electronics*, *7*(9), 186.

Ullah, Z. (2017). LH-CAM: Logic-based higher performance binary CAM architecture on FPGA. *IEEE Embedded Systems Letters*, *9*(2), 29-32.

Vajda, A. (2011). Multi-core and many-core processor architectures. In *Programming Many-Core Chips* (pp. 9-43). Springer.

Vangal, S. R., Howard, J., Ruhl, G., Dighe, S., Wilson, H., Tschanz, J., . . . Jain, S. (2008). An 80-tile sub-100-w teraflops processor in 65-nm cmos. *IEEE Journal of solid-state circuits*, *43*(1), 29-41.

Xiao, J., Yeung, K. L., & Jamin, S. (2019). ILP formulation for designing rings in routerless network-on-chip. ICC 2019-2019 IEEE International Conference on Communications (ICC),

Xu, C., & Ding, Y. (2010). An improved static-priority scheduling algorithm for multi-processor real-time systems.

Yunus, N. A. M., & Othman, M. (2011). Shuffle Exchange Network in Multistage InterconnectionNetwork: A Review and Challenges. *International Journal of Computer and Electrical Engineering*, *3*(5), 724.

Zhang, M., Lebeck, A. R., & Sorin, D. J. (2010). Fractal coherence: Scalably verifiable cache coherence. 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture,

# Appendices

# Appendix A

# Cache Coherence Approach

This approach is applied in all shared memory multicore system that used shared links of interconnection network. Each core is provided with a private cache and a cache controller, here the cache controller observes the bus transactions of all other cores, and takes a suitable action to preserve consistency of the version of the shared values in the cache. Snooping cache controllers is a protocol which monitors bus transaction for coherence purposes. Since multiple caches of the system are allowed to have same values of memory or a shared block of the global memory, a technique must exist in order to ensure that all versions stay consistent when the contents of that location (or that block) are changed. A piece of software called Cache Coherence protocol is used to help the cache controller in maintaining the consistency of the multiple versions. Any change in the state of the shared block is detected by the snooping cache protocol and reported to the cache. All controllers can receive the update simultaneously and take an action on that basis. The core polls the state of the block or the variable locally which reduces the traffic on the network. If the version is the most recent version of the block the core reads it locally, otherwise it presents a request to access the global locations through the network. [Archibald and Baer (1986)](#_10kxoro) have reported these protocols. As a detailed discussion of these protocols is out of the scope of this thesis, only one of these protocols is presented in order to demonstrate how they work and their effect on the synchronization and hence the communication overhead. This protocol is the Goodman's Write-Once Cache Coherence method [Archibald and Baer (1986)](#_10kxoro).

In the write-once method, the core local cache memory keeps a record of the state of the global blocks which have versions in it. The block can be in one of four states: DIRTY, VALID, RESERVED, and INVALID, see figure I-1 In order to make the definition of these states clear and to show how they affect the cache operations, it is important to distinguish between three things: the global version of the block, the cache version of the block, and the state of the block. The global version resides in the global memory, whereas the cache version and the state of the block reside in the local cache.

The DIRTY state refers to the most recent version of the block. It is a unique version and it resides in the local cache of the core which created it. If other core tries to read the old global version of the same block from the global memory, the snooping cache protocol of the cache holding the DIRTY version Prevents the global memory from provisioning the block and provides the block itself, in addition writing back this block to the global memory. This is called a READ-MISS operation. All caches which have a version of this block set their states to VALID when their controllers detect the READ-MISS operation. Hence the state VALID refers to the most recent version of the block residing in the global memory, and so the core must access the global memory via the common bus in order to read it or to modify it. The core which modifies its cache version of the block while the state of the block is VALID in other caches, sets its local state to RESERVED, and writes one word to the global version ( i.e. to the global memory via the common bus). As soon as the controllers of the caches with a version of that block detect this one word write, they change the VALID state in their caches with a version of that block detect this one word write, they change the VALID state in their caches to INVALID. This operation is one of the WRITE-HIT operations described below. Hence, the RESERVED state refers to a unique version in a local cache which does not need to be copied to the global memory, and it is not needed by other caches.

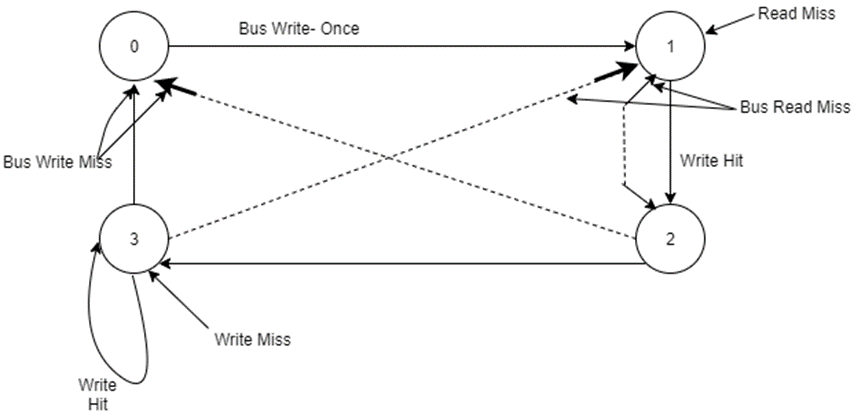
Other than READ-MISS operation two other cache coherence operations can be identified:

1. WRITE-HIT: if DIRTY state for a block, writing can do locally. In case of the RESERVED state, writing can also do immediately, and the state will be modified to DIRTY. If the state is VALID, directly writing the word to global memory then set the state to RESERVED. Whereas, the other caches in the different cores that have a version of that block monitor the Bus write transaction and change the state to be INVALID.

WRITE-MISS: similar to READ-MISS, the blocks are loaded from memory, or if it is a DIRTY block, a version that invalidates the version. When the writing is missing, all versions will be invalidated in all caches, and when the block is written, directly the state becomes DIRTY.

Figure A-1.

*Write-Once transaction diagram (*[*Martin et al., 2012*](#_1idq7dh)*)*



Key:

State 0: INVALID

State 1: VALID (CLEAN POTENTIALLY SHARED)

State 2: RESERVED

State 3: DIRTY (MODIFIED IN ONE CACHE ONLY)

From the above example of cache of coherence protocol we can deduce the following:

1. It is only the state of the block which is transmitted (updated) simultaneously in all caches. In most cases cores need to read the block itself from the global memory based on its state as presented in the cache.
2. In the case of cores whose their caches have an INVALID state of the block, they need to spin-wait on the global memory until a DIRTY version becomes available. This adds to the traffic on the common bus. Nothing in the protocol suggests otherwise.
3. Cache coherence protocols are treated as a separate issue from the synchronization process. In fact cache coherence protocols on their own (with no linkage to the synchronization process) represents a brute-force approach to ensure coherence since their is no semantic knowledge associated with the data being cached. In reality, memory coherence and process synchronization are closely interrelated. The ability of a process to read or write shared data is invariably acquired through some synchronization method. Since no synchronization information is associated with cached data, these algorithms are considered to be an overkill owing to their generality ([Stallings, 2013](#_1vsw3ci))..
4. Assuming that synchronization information (like the lock) is included in the state of the block in the cache, and the core is spin-waiting on the cache version of the lock rather than on version of global memory, the following factors cause the performance of spin-waiting on cache version to be worse than expected ([Martin et al., 2012](#_1idq7dh))..
5. There is an appreciable time interval between the release of the lock and its re-acquiring by another core. This time interval allows several cores to realize the lock has been released, allowing the test to pass (test for locked/un locked), and proceed to try the test-and-set.
6. When the lock is released, all cache versions will all be changed to invalid state; READ-misses will then be applied to each core. The first core to get the new value will then execute test-and-set, which must compete for the bus with any remaining read misses. When it does acquire the bus, it will invalidate all existing versions, and acquire the lock. Any core which completed its READ-MISS, however, will see the lock as free, and do a test-and-set itself.
7. If locks are updated simultaneously by the cache coherence protocol, i.e. all reads of the lock are done locally, and then each test-and-set requires a bus transaction.

A solution to reduce the problems mentioned in 4 & 5 by inserting delays into the spin-waiting loops ([Martin et al., 2012](#_1idq7dh)). He states as well that the cache controllers must be developed to an extent such that if it has a pending read-miss and observes the bus grant a read on the same location to some other cores, it can simply wait and take the data returning from that request. This idea is already developed and implemented in the FMOA research vehicle at Brighton Polytechnic.

1. The cache coherence organization is easy to apply only to a shared bus system. This is because its application to a multiple-bus system or a crossbar system requires that each core has to have a separate cache controller to monitor each global memory bus, which causes a large, extra organizational overhead on the core. Applying it to a MIN system is unthinkable.
2. We believe that with the current organization of the Cache Coherence protocol, unless a conditional critical section variable is associated with the protocol states and updated simultaneously in all cache after each change in the state of the variable, there will be a possibility of losing some versions of the global data This leads to incorrectness in the program execution, e.g. the case when a DIRTY cache version is updated locally before it has the chance of being used by other cores. When the time come for these cores to use the block the wrong (the new) DIRTY version will be supplied in a READ-MISS operation.
3. To execute a process for each synchronization and cache coherence operation is time consuming. Embedding the synchronization and coherence information in the hardware of the system will free the core or the cache controller from a time consuming process and increase the performance of the system.

It can be concluded that synchronization process is necessary in executing a parallel program but that it represents a big overhead if it is not carefully handled. In order to reduce this overhead the following must be considered:

1. Wherever it is possible, the synchronization process must be done in a single atomic instruction.
2. The program must be partitioned and scheduled in such a way that minimum communication is needed among program units that are scheduled on different cores.
3. Failure synchronization operations must be reduced to a minimum by including a full synchronization variable in the cache coherence protocol. This stops the core from accessing the global memory unless it has an equal chance of success with other cores.

However, the minimum possible synchronization still represents an extra communication overhead that contributes to the network delay ([Martin et al., 2012](#_1idq7dh)). Overlapping shared cache with the network communication process as in the MPCAM organization (chapter 5) improves the performance considerably.

# Appendix B

# DPCAM Implementation

**DPCAM Verilog Code**

// This code was written to simulate the DPCAM

// This code represent the top design file of the DCAM design

//Allam Abumwais 1.5.2020

module DPCAM (

sf , // SET Fipfope needed to operate the CU

rst , //reset

tags , // Input tag from pin comes from fetching process

data\_in , // data\_inpts

wr , // Write Enable

outI , // data after wrting

rd , // Read Enable

tag\_d , // tag of data bi-directional

outE , // data output to processors

);

parameter Data\_WIDTH = 32 ; // this parameter periodically change in simulation

parameter Tag\_WIDTH = 32 ; // this parameter periodically change in simulation

parameter Cam\_DEPTH = 4096; // this parameter periodically change in simulation

//--------------Input Ports Declaration-----------------------

input sf ;

//input clk;

input rst ;

input [Data\_WIDTH-1:0] data\_in ;

input [Tag\_WIDTH-1:0] tags ;

input wr ;

input rd ;

input [Tag\_WIDTH-1:0] tag\_d ;

//--------------output Ports Declaration-----------------------

output [Data\_WIDTH-1:0] outI ;

output [Data\_WIDTH-1:0] outE ;

//output full ;

//--------------in-output Ports Declaration-----------------------

//input [Tag\_WIDTH-1:0] tag\_d ;

//--------------Internal variables----------------

reg [Data\_WIDTH-1:0] outI ;

reg [Data\_WIDTH-1:0] outE ;

//reg [Tag\_WIDTH-1:0] tag\_d ;

reg [Data\_WIDTH-1:0] dcam [0:Cam\_DEPTH-1];

reg [Tag\_WIDTH-1:0] tcam [0:Cam\_DEPTH-1];

// reg [data\_0\_WIDTH-1:0] data\_1\_out ;

reg [data\_0\_WIDTH-1:0] mem [0:RAM\_DEPTH-1];

//type tag\_array is array (0 to Cam\_DEPTH-1) (Tag\_WIDTH-1 downto 0);

//type data\_array is array (0 to dpcamDepth-1) of std\_ulogic\_vector(dpcamWidthOut-1 downto 0);

//type ram\_type is array (0 to dpcamDepth-1) of std\_ulogic\_vector(dpcamWidthOut-1 downto 0);

reg tag\_array [Cam\_DEPTH-1:0][Tag\_WIDTH-1:0] ;

reg data\_array [Cam\_DEPTH-1:0][Data\_WIDTH-1:0] ;

integer counter\_wr=0 ;

integer counter\_rd=0 ;

integer full=0 ;

integer i ;

integer j ;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//--------------DPCAM (read and write) operations code starts here------------------

initial begin: DPCAM\_START

if (rst) begin

counter\_wr <= 0;

for (i=0; i< Cam\_DEPTH; i = i+1)

begin

dcam [i] <= 32'b0000000000000000 ;

tcam [i] <= 32'b0000000000000000 ;

//tag\_array [i] <= 20'b0000000000000000 ;

//data\_array [i] <= 32'b0000000000000000 ;

end

end

end

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Memory Write Block--- the write port

// Write Operation : When wr = 1, le = 1

always @ (negedge wr) // the first port for is writing port at negative edge…..

begin : MEM\_WRITE

if ( wr ) begin

if ( counter\_wr < Cam\_DEPTH-1)

begin

dcam [counter\_wr] <= data\_in;

outI[counter\_wr]<= data\_in;

tcam [counter\_wr] <= tags ;

counter\_wr = counter\_wr + 1 ;

end

else if ( counter\_wr > Cam\_DEPTH-1)

begin

full<= 1;

counter\_wr = 0 ;

dcam [counter\_wr] <= data\_in;

outI[counter\_wr]<= data\_in;

tcam [counter\_wr] <= tags ;

counter\_wr = counter\_wr + 1 ;

end

end

end

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

always @ (posedge rd) // the second port for reading port at positive edge…..

begin : MEM\_READ

//reg [9:0] counter\_rd; // to enable loop inside the always block

counter\_rd=0;

//if ( rd )

begin

for (counter\_rd=0; counter\_rd< Cam\_DEPTH; counter\_rd = counter\_rd+1)

begin

if (tag\_d == tcam [counter\_rd])

begin

outE<=outI;

outE<= dcam [counter\_rd];

//counter-rd = counter-rd+1

end

end

outE<= 0;

end

end

endmodule

**Test\_bench Program Code**

// This testbench was written to assess the performance of DPCAM Design

//Allam Abumwais 1.5.2020

module testbench ;

reg [31:0] data\_in ;

wire [31:0] outE ;

wire [31:0] outI ;

reg wr ;

reg rst ;

reg sf ;

reg rd ;

reg [15:0] tag\_d ;

reg [15:0] tags ;

DPCAM

DUT (

.data\_in (data\_in ) ,

.outE (outE ) ,

.outI (outI ) ,

.wr (wr ) ,

.rst (rst ) ,

.sf (sf ) ,

.rd (rd ) ,

.tag\_d (tag\_d ) ,

.tags (tags ) );

// "Constant Pattern"

// Start Time = 0 ns, End Time = 1 us, Period = 0 ns

initial

begin // rest the control unit to start from the fist location

sf = 1'b1 ;

# 4 sf = 1'b0 ;

# 996 ;

// dumped values till 1 us

end

// "Constant Pattern"

// Start Time = 0 ns, End Time = 1 us, Period = 0 ns

initial

begin

rst = 1'b1 ;

# 4 rst = 1'b0 ;

# 996 ;

// dumped values till 1 us

end

// "Constant Pattern"

// Start Time = 10 ns, End Time = 1 us, Period = 0 ns

initial

begin

1 = 16'b00000000000000000000000000000000 ;

# 10 tags = 16'b11111111111111111111111111111111 ;

# 10 tags = 16'b00000000000000000000000000000000 ;

# 980 ;

// dumped values till 1 us

end

// "Constant Pattern"

// Start Time = 10 ns, End Time = 1 us, Period = 0 ns

initial

begin

data\_in = 32'b00000000000000000000000000000000 ;

# 10 data\_in = 32'b11111111111111111111111111111111 ;

# 10 data\_in = 32'b00000000000000000000000000000000 ;

# 980 ;

// dumped values till 1 us

end

// "Clock Pattern" : dutyCycle = 40

// Start Time = 0 ns, End Time = 1 us, Period = 10 ns

initial

begin

repeat(100)

begin

wr = 1'b1 ;

#4 wr = 1'b0 ;

#6 ;

// 1 us, repeat pattern in loop.

end

end

// "Constant Pattern"

// Start Time = 10 ns, End Time = 1 us, Period = 0 ns

//you can change the start and end time every simulation

initial

begin

if (outI != (32'b00000000000000000000000000000000 )) $display($time, " test case failed");

# 10 if (outI != (32'b11111111111111111111111111111111 )) $display($time, " test case failed");

# 10 if (outI != (32'b00000000000000000000000000000000 )) $display($time, " test case failed");

# 980 ;

// dumped values till 1 us

end

// "Clock Pattern" : dutyCycle = 30

// Start Time = 0 ns, End Time = 1 us, Period = 10 ns

initial

begin

rd = 1'b0 ;

# 7 ;

repeat(999)

begin

rd = 1'b1 ;

#3 rd = 1'b0 ;

#7 ;

// 999X10 ns, repeat pattern in loop.

end

rd = 1'b1 ;

# 3 ;

// dumped values till 1 us

end

// "Constant Pattern"

// Start Time = 10 ns, End Time = 1 us, Period = 0 ns

initial

begin

tag\_d = 16'b00000000000000000000000000000000 ;

# 10 tag\_d = 16'b11111111111111111111111111111111 ;

# 20 tag\_d = 16'b00000000000000000000000000000000 ;

# 970 ;

// dumped values till 1 us

end

// "Constant Pattern"

// Start Time = 0 ns, End Time = 1 us, Period = 0 ns

initial

begin

if (outE != (32'b00000000000000000000000000000000 )) $display($time, " test case failed");

# 1000 ;

// dumped values till 1 us

end

initial

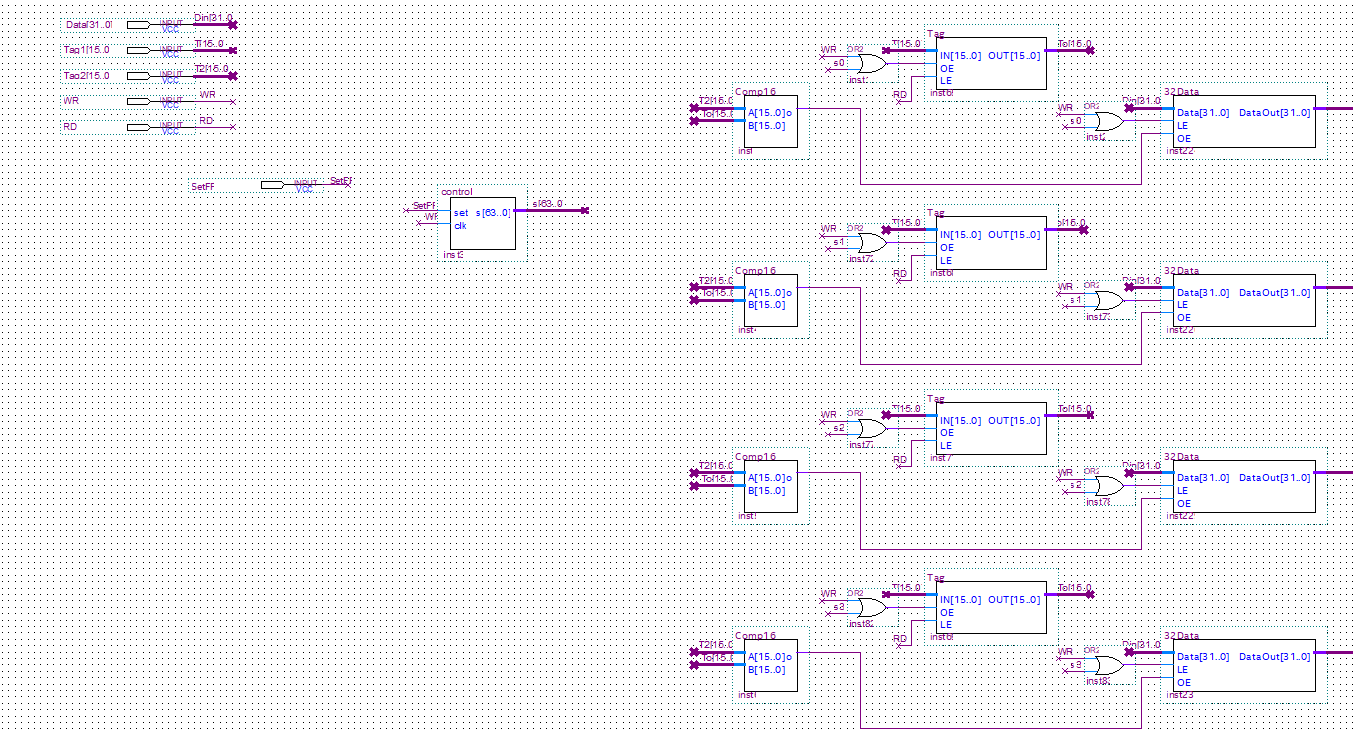
#2000 $stop;

endmodule

**DPCAM Location Design**

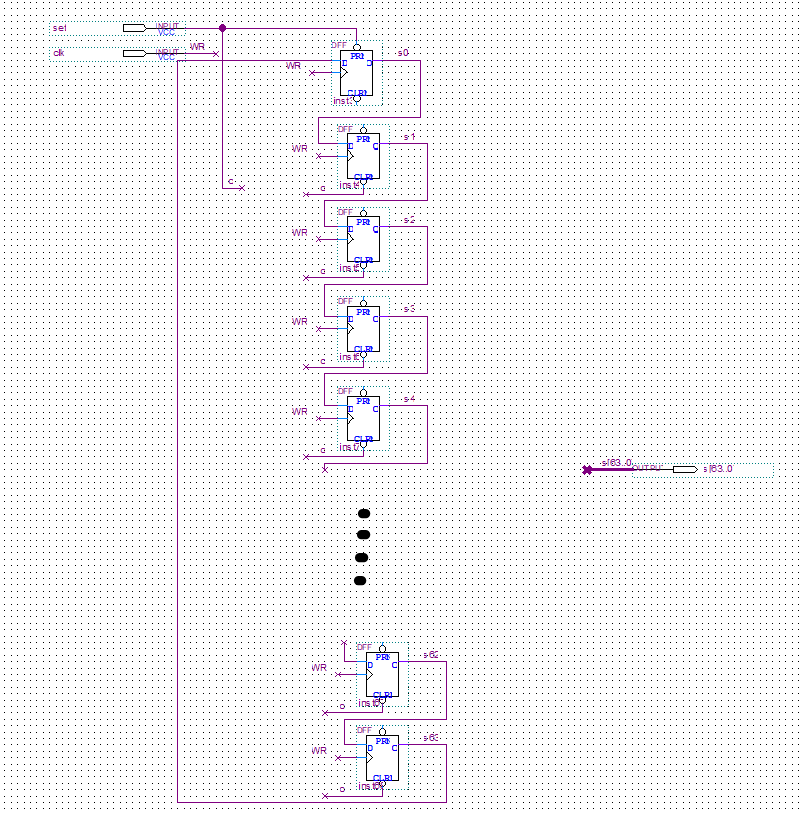
**Figure B-1.**

*DPCAM Location Architecture*



**Figure B-1.**

*Control Unit Architecture*



**Latency measurements T-Test and Levene's Test**

*Table B.1.*

*Descriptive Statistics For Write Latency Comparison.*

|  | WR-Group | N | Mean | Std. Devi | Std. error means | |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 100 | 0.9592 | 0.03393 | 0.0339 |
| Wr-Comparison | 2 | 100 | 1.9434 | 0.03822 | .00382 |  |

*Table B.2.*

Levens’s Test Independent Samples Test

|  | | Levene's Test for Equality of Variances | |
| --- | --- | --- | --- |
| F. | Sig. |
| WR\_Comparisons | Equal variances assumed  Equal variances not assumed | 1.582 | .210 |

*Table B.3*

*T-test For Write Comparisons*

|  | | t-test for Equality of Means | | |  |
| --- | --- | --- | --- | --- | --- |
| T | df | Sig. | Mean Diff |
| WR\_Comparisons | Equal variances assumed  Equal variances not assumed | -193.809  -193.809 | 198  195.252 | .000  .000 | -.9905  -.9905 |

*Table B.4*

*T-test For Read Comparisons*

|  | | t-test for Equality of Means | | |
| --- | --- | --- | --- | --- |
| Std. Error Difference | 95% Confidence Interval of the Differences | |
| Lower | Upper |
| Read\_Comparisons | Equal variances assumed  Equal variances not assumed | .00511  .00511 | -1.00063  -1.00063 | -.98047  -.98047 |

*Table B.5.*

*Group Statistic For Read Latencies*

| Group | N | Mean | Std. Deviation | Std. error mean |
| --- | --- | --- | --- | --- |
| Read 1  Comparison 2 | 100  100 | 1.9434  2.1584 | .03822  .10559 | .00382  .1056 |

Table B.6.

Levens’s Test For Read

|  | | Levene's Test for Equality of Variances | |
| --- | --- | --- | --- |
| F. | Sig. |
| Read\_Comparisons | Equal variances assumed  Equal variances not assumed | 87.654 | .000 |

*Table B.7.*

*T-test For Write Comparisons*

|  | | t-test for Equality of Means | | |  |
| --- | --- | --- | --- | --- | --- |
| T | df | Sig. | Mean Diff |
| WR\_Comparisons | Equal variances assumed  Equal variances not assumed | -19.146  -19.146 | 198  124.510 | .000  .000 | -.2150  -.2150 |

# Appendix C

# DPCAM With GEM5 Implementation

*Table C.1.A.*

*Benchmark Programs Reference Count*

| Traditional | DPCAM | Traditional | DPCAM | Traditional | DPCAM | Traditional | | DPCAM |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BZip2 | | astar | | gammes | | hummer | | |
| 30491718 | 30491718 | 5234678 | 5234678 | 8971498 | 8971498 | 10967781 | 10967781 | |

*Table C.1.B.*

*Benchmark Programs Reference Count*

| Traditional | DPCAM | Traditional | DPCAM | Traditional | DPCAM |
| --- | --- | --- | --- | --- | --- |
| gcc | | mcf | | sphinx3 | |
| 9873018 | 9873018 | 5994666 | 5994666 | 34491313 | 34491313 |

*Table C.2.A.*

*Benchmark Programs misses Count*

| Traditional | DPCAM | Traditional | DPCAM | Traditional | DPCAM | Traditional | | DPCAM |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BZip2 | | astar | | gammes | | hummer | | |
| 838316 | 982213 | 183213 | 20155 | 376802 | 18840 | 570324 | 20400 | |

*Table C.2.B.*

*Benchmark Programs misses Count*

| Traditional | DPCAM | Traditional | DPCAM | Traditional | DPCAM |
| --- | --- | --- | --- | --- | --- |
| gcc | | mcf | | Sphinx3 | |
| 473904 | 18265 | 194826 | 20981 | 896716 | 1751917 |

*Table C.3.A.*

*Benchmark Programs miss rate*

| Traditional | DPCAM | Traditional | DPCAM | Traditional | DPCAM | Traditional | | DPCAM |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BZip2 | | astar | | gammes | | hummer | | |
| 2.7% | 3.22% | 3.5% | 0.38% | 4.2% | 0.21% | 5.2% | 0.186% | |

*Table C.3.B.*

*Benchmark Programs miss rate*

| Traditional | DPCAM | Traditional | DPCAM | Traditional | DPCAM |
| --- | --- | --- | --- | --- | --- |
| gcc | | mcf | | Sphinx3 | |
| 4.8% | 0.185% | 3.25% | 0.35% | 2.6% | 5.1% |

**(df) Benchmarking Program**

// the following results represent the single thread benchmark simulation running on both traditional and DPCAM architecture (df) benchmark program as example.

Traditional-multicore=4

==2596== Number of simulated cores: 4

==2596== I refs: 398,496

==2596== I1 misses: 1,630

==2596== I1 miss rate: 0.40%

==2596==

==2596== D refs: 193,321 (133,782 rd + 59,539 wr)

==2596== D1 misses: 2,213 ( 1,837 rd + 376 wr)

==2596== L2d misses: 1,691 ( 1,370 rd + 321 wr)

==2596== D1 miss rate: 1.1% ( 1.3% + 0.6% )

==2596== L2d miss rate: 0.8% ( 1.0% + 0.5% )

==2596==

==2596== L2 refs: 3,843 ( 3,467 rd + 376 wr)

==2596== L2 misses: 3,169 ( 2,848 rd + 321 wr)

==2596== L2 miss rate: 0.5% ( 0.5% + 0.5% )

==2596==

==2596== Gem5 Ruby Cache Simulator

==2596== Thread Stats: Instructions

==2596== Thread 0:398496

==2596== Thread 1:0

==2596== Thread 2:0

==2596== Thread 3:0

==2596== Bus Transactions:

==2596== Thread 0: Reads:2861; Writes: 59600; Shared Reads: 0; Invalidation: 0

==2596== Thread 1: Reads:0; Writes: 0; Shared Reads: 0; Invalidation: 0

==2596== Thread 2: Reads:0; Writes: 0; Shared Reads: 0; Invalidation: 0

==2596== Thread 3: Reads:0; Writes: 0; Shared Reads: 0; Invalidation: 0

==2596== Summary: Reads: 2861; Writes: 59600; Shared Reads:0; Invalidation: 0

==2596==

==2596== Number of invalidations per write access

==2596== 1;2;<5;>4

==2596== Thread 0: 0,0,0,0

==2596== Thread 1: 0,0,0,0

==2596== Thread 2: 0,0,0,0

==2596== Thread 3: 0,0,0,0

==================

DPCAM-multicore=4

==2594== Number of simulated cores: 4

==2594== I refs: 398,496

==2594== I1 misses: 1,630

==2594== I1 miss rate: 0.40%

==2594==

==2594== D refs: 193,321 (133,782 rd + 59,539 wr)

==2594== D1 misses: 2,213 ( 1,837 rd + 376 wr)

==2594== L2d misses: 1,691 ( 1,370 rd + 321 wr)

==2594== D1 miss rate: 1.1% ( 1.3% + 0.6% )

==2594== L2d miss rate: 0.8% ( 1.0% + 0.5% )

==2594==

==2594== L2 refs: 3,843 ( 3,467 rd + 376 wr)

==2594== L2 misses: 3,169 ( 2,848 rd + 321 wr)

==2594== L2 miss rate: 0.5% ( 0.5% + 0.5% )

==2594==

==2594== Gem5 Ruby Cache Simulator

==2594== Thread Stats: Instructions

==2594== Thread 0:398496

==2594== Thread 1:0

==2594== Thread 2:0

==2594== Thread 3:0

==2594== Bus Transactions:

==2594== Thread 0: Reads:2861; Writes: 59600; Shared Reads: 0; Invalidation: 0

==2594== Thread 1: Reads:0; Writes: 0; Shared Reads: 0; Invalidation: 0

==2594== Thread 2: Reads:0; Writes: 0; Shared Reads: 0; Invalidation: 0

==2594== Thread 3: Reads:0; Writes: 0; Shared Reads: 0; Invalidation: 0

==2594== Summary: Reads: 2861; Writes: 59600; Shared Reads:0; Invalidation: 0

==2594==

==2594== Number of invalidations per write access

==2594== 1;2;<5;>4

==2594== Thread 0: 0,0,0,0

==2594== Thread 1: 0,0,0,0

==2594== Thread 2: 0,0,0,0

==2594== Thread 3: 0,0,0,0

# Appendix D

# MPCAM Implementation

**MPCAM Verilog code**

// PROGRAM "Intel Quartus Prime - Version 20.1.0 Lite Edition" "Intel Corporation's design tools"

// CREATED by Allam Abumwais 5.6.2021

//DOWNLOAD https://fpgasoftware.intel.com/eula. "Full components"

//design file for four cores interconnection using MPCAM organization.

module Design( // the following lines show all inputs and outputs ports...

WR, //Write input port

RD, //Read input port

clk, //clk input port

D-core1, // Data input for core1

D-core2, // Data input for core2

D-core3, // Data input for core3

D-core4, // Data input for core4

Tagd-core1, // Tag destination input for core1

Tagd-core2, // Tag destination input for core2

Tagd-core3, // Tag destination input for core3

Tagd-core4, // Tag destination input for core4

Tags-core1, // Tag source input for core1

Tags-core2, // Tag destination input for core2

Tags-core3, // Tag destination input for core3

Tags-core4, // Tag destination input for core4

Dout-core1, // Data output for core1

Dout-core2, // Data output for core2

Dout-core3, // Data output for core3

Dout-core4 // Data output for core4

);

input wire WR;

input wire RD;

input wire clk;

input wire [31:0] D-core1; //32-bit data input

input wire [31:0] D-core2;

input wire [31:0] D-core3;

input wire [31:0] D-core4;

input wire [15:0] Tagd-core1; // 16-bit tagd input

input wire [15:0] Tagd-core2;

input wire [15:0] Tagd-core3;

input wire [15:0] Tagd-core4;

input wire [15:0] Tags-core1; // 16-bit tags input

input wire [15:0] Tags-core2;

input wire [15:0] Tags-core3;

input wire [15:0] Tags-core4;

output wire [31:0] Dout-core1; // 32-bit data output

output wire [31:0] Dout-core2;

output wire [31:0] Dout-core3;

output wire [31:0] Dout-core4;

//Then assign each wire to pines SIGNAL...........

//example assign Dd31 = GDFX\_TEMP\_SIGNAL\_11[31];

DPCAM b2v\_inst0( // access DPCAM instance 0

.Wr(WR),

.Rd(RD),

.clk(clk),

.Ds(GDFX\_TEMP\_SIGNAL\_0),

.Td(GDFX\_TEMP\_SIGNAL\_1),

.Ts(GDFX\_TEMP\_SIGNAL\_2),

.Dd(GDFX\_TEMP\_SIGNAL\_3));

DPCAM b2v\_inst1( // access DPCAM instance 1

.Wr(WR),

.Rd(RD),

.clk(clk),

.Ds(GDFX\_TEMP\_SIGNAL\_4),

.Td(GDFX\_TEMP\_SIGNAL\_5),

.Ts(GDFX\_TEMP\_SIGNAL\_6),

.Dd(GDFX\_TEMP\_SIGNAL\_7));

DPCAM b2v\_inst2( // access DPCAM instance 2

.Wr(WR),

.Rd(RD),

.clk(clk),

.Ds(GDFX\_TEMP\_SIGNAL\_8),

.Td(GDFX\_TEMP\_SIGNAL\_9),

.Ts(GDFX\_TEMP\_SIGNAL\_10),

.Dd(GDFX\_TEMP\_SIGNAL\_11));

DPCAM b2v\_inst3( // access DPCAM instance 3

.Wr(WR),

.Rd(RD),

.clk(clk),

.Ds(GDFX\_TEMP\_SIGNAL\_12),

.Td(GDFX\_TEMP\_SIGNAL\_13),

.Ts(GDFX\_TEMP\_SIGNAL\_14),

.Dd(SYNTHESIZED\_WIRE\_1));

DPCAM b2v\_inst4( // access DPCAM instance 4

.Wr(WR),

.Rd(RD),

.clk(clk),

.Ds(GDFX\_TEMP\_SIGNAL\_15),

.Td(GDFX\_TEMP\_SIGNAL\_16),

.Ts(GDFX\_TEMP\_SIGNAL\_17),

.Dd(GDFX\_TEMP\_SIGNAL\_18));

DPCAM b2v\_inst5( // access DPCAM instance 5

.Wr(WR),

.Rd(RD),

.clk(clk),

.Ds(GDFX\_TEMP\_SIGNAL\_19),

.Ts(GDFX\_TEMP\_SIGNAL\_20),

.Dd(GDFX\_TEMP\_SIGNAL\_21));

DPCAM b2v\_inst6( // access DPCAM instance 6

.Wr(WR),

.Rd(RD),

.clk(clk),

.Ds(GDFX\_TEMP\_SIGNAL\_22),

.Td(GDFX\_TEMP\_SIGNAL\_23),

.Ts(GDFX\_TEMP\_SIGNAL\_24),

.Dd(GDFX\_TEMP\_SIGNAL\_25));

DPCAM b2v\_inst7( // access DPCAM instance 7

.Wr(WR),

.Rd(RD),

.clk(clk),

.Ds(GDFX\_TEMP\_SIGNAL\_26),

.Td(GDFX\_TEMP\_SIGNAL\_27),

.Ts(GDFX\_TEMP\_SIGNAL\_28),

.Dd(SYNTHESIZED\_WIRE\_4));

DPCAM b2v\_inst8( // access DPCAM instance 7

.Wr(WR),

.Rd(RD),

.clk(clk),

.Ds(GDFX\_TEMP\_SIGNAL\_29),

.Td(GDFX\_TEMP\_SIGNAL\_30),

.Ts(GDFX\_TEMP\_SIGNAL\_31),

.Dd(GDFX\_TEMP\_SIGNAL\_32));

DPCAM b2v\_inst9( // access DPCAM instance 9

.Wr(WR),

.Rd(RD),

.clk(clk),

.Ds(GDFX\_TEMP\_SIGNAL\_37),

.Td(GDFX\_TEMP\_SIGNAL\_38),

.Ts(GDFX\_TEMP\_SIGNAL\_39),

.Dd(SYNTHESIZED\_WIRE\_3));

DPCAM b2v\_inst10( // access DPCAM instance 10

.Wr(WR),

.Rd(RD),

.clk(clk),

.Ds(GDFX\_TEMP\_SIGNAL\_47),

.Td(GDFX\_TEMP\_SIGNAL\_48),

.Ts(GDFX\_TEMP\_SIGNAL\_49),

.Dd(GDFX\_TEMP\_SIGNAL\_50));

DPCAM b2v\_inst11( // access DPCAM instance 11

.Wr(WR),

.Rd(RD),

.clk(clk),

.Ds(GDFX\_TEMP\_SIGNAL\_51),

.Ts(GDFX\_TEMP\_SIGNAL\_52),

.Dd(GDFX\_TEMP\_SIGNAL\_53));

DPCAM b2v\_inst12( // access DPCAM instance 12

.Wr(WR),

.Rd(RD),

.clk(clk),

.Ds(GDFX\_TEMP\_SIGNAL\_54),

.Td(GDFX\_TEMP\_SIGNAL\_55),

.Dd(SYNTHESIZED\_WIRE\_0));

DPCAM b2v\_inst13( // access DPCAM instance 13

.Wr(WR),

.Rd(RD),

.clk(clk),

.Ds(GDFX\_TEMP\_SIGNAL\_56),

.Td(GDFX\_TEMP\_SIGNAL\_57),

.Ts(GDFX\_TEMP\_SIGNAL\_58),

.Dd(SYNTHESIZED\_WIRE\_2));

DPCAM b2v\_inst14( // access DPCAM instance 14

.Wr(WR),

.Rd(RD),

.clk(clk),

.Ds(GDFX\_TEMP\_SIGNAL\_59),

.Td(GDFX\_TEMP\_SIGNAL\_60),

.Ts(GDFX\_TEMP\_SIGNAL\_61),

.Dd(GDFX\_TEMP\_SIGNAL\_62));

DPCAM b2v\_inst15( // access DPCAM instance 15

.Wr(WR),

.Rd(RD),

.clk(clk),

.Ds(GDFX\_TEMP\_SIGNAL\_63),

.Ts(GDFX\_TEMP\_SIGNAL\_64),

.Dd(GDFX\_TEMP\_SIGNAL\_65));

//always block for OF and SB units

Always @ (Rd, posedge clk)

begin

assign Dout in each core with all column in MPCAM necessary for read operation

assign Dout-core1 = GDFX\_TEMP\_SIGNAL\_33 | GDFX\_TEMP\_SIGNAL\_34 | GDFX\_TEMP\_SIGNAL\_35 | GDFX\_TEMP\_SIGNAL\_36;

assign Dout-core2 = GDFX\_TEMP\_SIGNAL\_40 | SYNTHESIZED\_WIRE\_0 | GDFX\_TEMP\_SIGNAL\_41 | GDFX\_TEMP\_SIGNAL\_42;

assign Dout-core3 = SYNTHESIZED\_WIRE\_1 | SYNTHESIZED\_WIRE\_2 | SYNTHESIZED\_WIRE\_3 | SYNTHESIZED\_WIRE\_4;

assign Dout-core4 = GDFX\_TEMP\_SIGNAL\_43 | GDFX\_TEMP\_SIGNAL\_44 | GDFX\_TEMP\_SIGNAL\_45 | GDFX\_TEMP\_SIGNAL\_46;

assign D-core and tags in each core with all row in MPCAM necessary for write operation

// example assign Dout-core1 = GDFX\_TEMP\_SIGNAL\_33 | GDFX\_TEMP\_SIGNAL\_34 | GDFX\_TEMP\_SIGNAL\_35 | GDFX\_TEMP\_SIGNAL\_36;

// assign D-core1= SIG\_33 & D-core1= SIG\_40 & D-core1= SIG\_1 & D-core1= SIG\_43..........

end

endmodule

**MPCAM Organization Ports Verilog code**

// PROGRAM "Intel Quartus Prime - Version 20.1.0 Lite Edition" "Intel Corporation's design tools"

// CREATED by Allam Abumwais 1.7.2021

//DOWNLOAD https://fpgasoftware.intel.com/eula. "Full components"

//Top design file for four cores system using MPCAM organization main ports .

module MPCAM (

WR,

RD,

D-core1,

D-core2,

D-core3,

D-core4,

DI-core1,

DI-core2,

DI-core3,

DI-core4,

Tagd-core1,

Tagd-core2,

Tagd-core3,

Tagd-core4,

Tags-core1,

Tags-core2,

Tags-core3,

Tags-core4,

Dout-core1,

Dout-core2,

Dout-core3,

Dout-core4

);

input wire WR;

input wire RD;

input wire [31:0] D-core1;

input wire [31:0] D-core2;

input wire [31:0] D-core3;

input wire [31:0] D-core4;

input wire [31:0] DI-core1;

input wire [31:0] DI-core2;

input wire [31:0] DI-core3;

input wire [31:0] DI-core4;

input wire [15:0] Tagd-core1;

input wire [15:0] Tagd-core2;

input wire [15:0] Tagd-core3;

input wire [15:0] Tagd-core4;

input wire [15:0] Tags-core1;

input wire [15:0] Tags-core2;

input wire [15:0] Tags-core3;

input wire [15:0] Tags-core4;

output wire [31:0] Dout-core1;

output wire [31:0] Dout-core2;

output wire [31:0] Dout-core3;

output wire [31:0] Dout-core4;

Design b2v\_inst(

.WR(WR),

.RD(RD),

.D-core1(D-core1),

.D-core2(D-core2),

.D-core3(D-core3),

.D-core4(D-core4),

.DI-core1(DI-core1),

.DI-core2(DI-core2),

.DI-core3(DI-core3),

.DI-core4(DI-core4),

.Tagd-core1(Tagd-core1),

.Tagd-core2(Tagd-core2),

.Tagd-core3(Tagd-core3),

.Tagd-core4(Tagd-core4),

.Tags-core1(Tags-core1),

.Tags-core2(Tags-core2),

.Tags-core3(Tags-core3),

.Tags-core4(Tags-core4),

.Dout-core1(Dout-core1),

.Dout-core2(Dout-core2),

.Dout-core3(Dout-core3),

.Dout-core4(Dout-core4));

endmodule

//Top design file for each instance in MPCAM organization

Design Design\_inst

(

.D-core1(D-core1\_sig) , // input [31:0] D-core1\_sig

.D-core2(D-core2\_sig) , // input [31:0] D-core2\_sig

.D-core3(D-core3\_sig) , // input [31:0] D-core3\_sig

.D-core4(D-core4\_sig) , // input [31:0] D-core4\_sig

.Tags-core1(Tags-core1\_sig) , // input [15:0] Tags-core1\_sig

.Tags-core2(Tags-core2\_sig) , // input [15:0] Tags-core2\_sig

.Tags-core3(Tags-core3\_sig) , // input [15:0] Tags-core3\_sig

.Tags-core4(Tags-core4\_sig) , // input [15:0] Tags-core4\_sig

.WR(WR\_sig) , // input WR\_sig

.RD(RD\_sig) , // input RD\_sig

.Tagd-core2(Tagd-core2\_sig) , // input [15:0] Tagd-core2\_sig

.Tagd-core3(Tagd-core3\_sig) , // input [15:0] Tagd-core3\_sig

.Tagd-core4(Tagd-core4\_sig) , // input [15:0] Tagd-core4\_sig

.Tagd-core1(Tagd-core1\_sig) , // input [15:0] Tagd-core1\_sig

.clk(clk\_sig) , // input clk\_sig

.DI-core1(DI-core1\_sig) , // input [31:0] DI-core1\_sig

.DI-core2(DI-core2\_sig) , // input [31:0] DI-core2\_sig

.DI-core3(DI-core3\_sig) , // input [31:0] DI-core3\_sig

.DI-core4(DI-core4\_sig) , // input [31:0] DI-core4\_sig

.Dout-core1(Dout-core1\_sig) , // output [31:0] Dout-core1\_sig

.Dout-core2(Dout-core2\_sig) , // output [31:0] Dout-core2\_sig

.Dout-core3(Dout-core3\_sig) , // output [31:0] Dout-core3\_sig

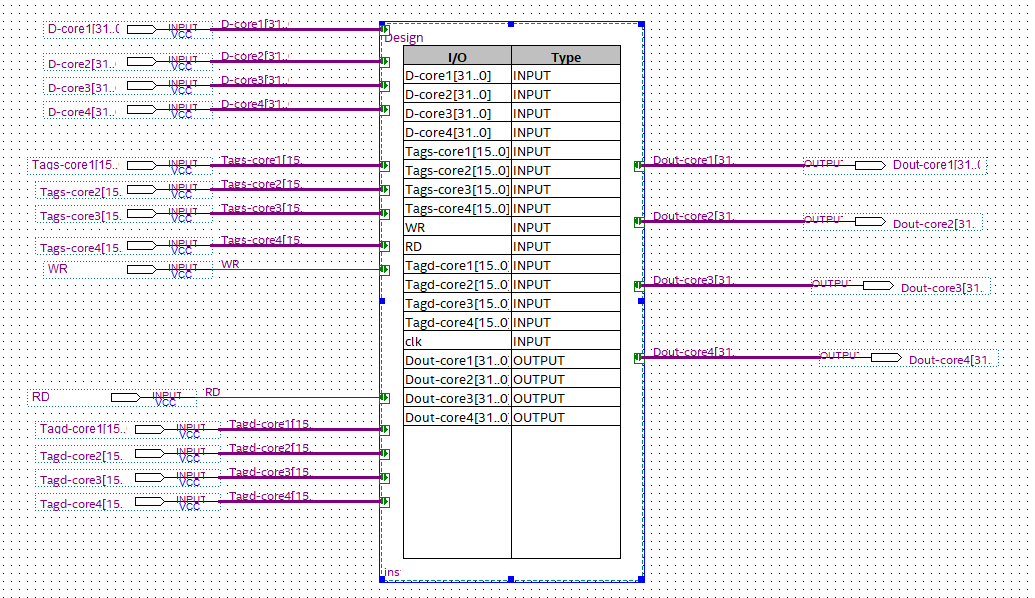
.Dout-core4(Dout-core4\_sig) // output [31:0] Dout-core4\_sig

);

**MPCAM Organization Design**

**Figure E-1.**

*MPCAM Organization*

****

****

**Latency measurements T-Test and Levene's Test**

*Table D.1.*

*Descriptive Statistics For Write Latency Comparison.*

|  | WR-Group | N | Mean | Std. Devi | Std. error means | |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 100 | 1.084115 | 0.0338499 | 0.0033850 |
| Wr-Comparison | 2 | 100 | 1.160854 | 0.2363034 | .02363030 |  |

*Table D.2.*

Levens’s Test Independent Samples Test

|  | | Levene's Test for Equality of Variances | |
| --- | --- | --- | --- |
| F. | Sig. |
| WR\_Comparisons | Equal variances assumed  Equal variances not assumed | 29.952 | .00 |

*Table D.3*

*T-test For Write Comparisons*

|  | | t-test for Equality of Means | | |  |
| --- | --- | --- | --- | --- | --- |
| T | df | Sig. | Mean Diff |
| WR\_Comparisons | Equal variances assumed  Equal variances not assumed | -3.215  -3.215 | 198  103.061 | .002  .002 | -.0767  -.0767 |

*Table D.4.*

*Descriptive Statistics For Read Latency Comparison.*

|  | WR-Group | N | Mean | Std. Devi | Std. error means | |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 100 | 1.278040 | 0.0868235 | 0.0086824 |
| Read-Comparison | 2 | 100 | 1.310480 | 0.0919551 | .0091955 |  |

*Table D.5.*

Levens’s Test Independent Samples Test

|  | | Levene's Test for Equality of Variances | |
| --- | --- | --- | --- |
| F. | Sig. |
| Read\_Comparisons | Equal variances assumed  Equal variances not assumed | 0.88 | .767 |

*Table D.6*

*T-test For Read Comparisons*

|  | | t-test for Equality of Means | | |  |
| --- | --- | --- | --- | --- | --- |
| T | df | Sig. | Mean Diff |
| Read\_Comparisons | Equal variances assumed  Equal variances not assumed | -2.565  -2.565 | 198  197.315 | .011  .011 | -.032440  -.0324400 |

# Appendix E

# A Scalable Interconnection Implementation

**NCSC Verilog Code**

// PROGRAM "Intel Quartus Prime - Version 20.1.0 Lite Edition" "Intel Corporation's design tools"

// CREATED by Allam Abumwais 5.8.2021

//DOWNLOAD https://fpgasoftware.intel.com/eula. "Full components"

//Top design file for 16-cores interconnection using NCSC interconnection network based on multi-cluster.

module NCSC( // input and output ports for NCSC design

WR,

RD,

clk,

D-core11,

D-core12,

D-core13,

D-core14,

D-core21,

D-core22,

D-core23,

D-core24,

D-core31,

D-core32,

D-core33,

D-core34,

D-core41,

D-core42,

D-core43,

D-core44,

Tagd-core11,

Tagd-core12,

Tagd-core13,

Tagd-core14,

Tagd-core21,

Tagd-core22,

Tagd-core23,

Tagd-core24,

Tagd-core31,

Tagd-core32,

Tagd-core33,

Tagd-core34,

Tagd-core41,

Tagd-core42,

Tagd-core43,

Tagd-core44,

Tags-core11,

Tags-core12,

Tags-core13,

Tags-core14,

Tags-core21,

Tags-core22,

Tags-core23,

Tags-core24,

Tags-core31,

Tags-core32,

Tags-core33,

Tags-core34,

Tags-core41,

Tags-core42,

Tags-core43,

Tags-core44,

Dout-core11,

Dout-core12,

Dout-core13,

Dout-core14,

Dout-core21,

Dout-core22,

Dout-core23,

Dout-core24,

Dout-core31,

Dout-core32,

Dout-core33,

Dout-core34,

Dout-core41,

Dout-core42,

Dout-core43,

Dout-core44

);

input wire WR;

input wire RD;

input wire clk;

input wire [31:0] D-core11;

input wire [31:0] D-core12;

input wire [31:0] D-core13;

input wire [31:0] D-core14;

input wire [31:0] D-core21;

input wire [31:0] D-core22;

input wire [31:0] D-core23;

input wire [31:0] D-core24;

input wire [31:0] D-core31;

input wire [31:0] D-core32;

input wire [31:0] D-core33;

input wire [31:0] D-core34;

input wire [31:0] D-core41;

input wire [31:0] D-core42;

input wire [31:0] D-core43;

input wire [31:0] D-core44;

input wire [15:0] Tagd-core11;

input wire [15:0] Tagd-core12;

input wire [15:0] Tagd-core13;

input wire [15:0] Tagd-core14;

input wire [15:0] Tagd-core21;

input wire [15:0] Tagd-core22;

input wire [15:0] Tagd-core23;

input wire [15:0] Tagd-core24;

input wire [15:0] Tagd-core31;

input wire [15:0] Tagd-core32;

input wire [15:0] Tagd-core33;

input wire [15:0] Tagd-core34;

input wire [15:0] Tagd-core41;

input wire [15:0] Tagd-core42;

input wire [15:0] Tagd-core43;

input wire [15:0] Tagd-core44;

input wire [15:0] Tags-core11;

input wire [15:0] Tags-core12;

input wire [15:0] Tags-core13;

input wire [15:0] Tags-core14;

input wire [15:0] Tags-core21;

input wire [15:0] Tags-core22;

input wire [15:0] Tags-core23;

input wire [15:0] Tags-core24;

input wire [15:0] Tags-core31;

input wire [15:0] Tags-core32;

input wire [15:0] Tags-core33;

input wire [15:0] Tags-core34;

input wire [15:0] Tags-core41;

input wire [15:0] Tags-core42;

input wire [15:0] Tags-core43;

input wire [15:0] Tags-core44;

output wire [31:0] Dout-core11;

output wire [31:0] Dout-core12;

output wire [31:0] Dout-core13;

output wire [31:0] Dout-core14;

output wire [31:0] Dout-core21;

output wire [31:0] Dout-core22;

output wire [31:0] Dout-core23;

output wire [31:0] Dout-core24;

output wire [31:0] Dout-core31;

output wire [31:0] Dout-core32;

output wire [31:0] Dout-core33;

output wire [31:0] Dout-core34;

output wire [31:0] Dout-core41;

output wire [31:0] Dout-core42;

output wire [31:0] Dout-core43;

output wire [31:0] Dout-core44;

wire [31:0] gdfx\_temp0;

wire [31:0] gdfx\_temp1;

wire [15:0] gdfx\_temp10;

wire [15:0] gdfx\_temp11;

wire [15:0] gdfx\_temp12;

wire [15:0] gdfx\_temp13;

wire [15:0] gdfx\_temp14;

wire [15:0] gdfx\_temp15;

wire [15:0] gdfx\_temp16;

wire [15:0] gdfx\_temp17;

wire [31:0] gdfx\_temp2;

wire [31:0] gdfx\_temp3;

wire [31:0] gdfx\_temp4;

wire [31:0] gdfx\_temp5;

wire [15:0] gdfx\_temp6;

wire [15:0] gdfx\_temp7;

wire [15:0] gdfx\_temp8;

wire [15:0] gdfx\_temp9;

//each cluster represent a MPCAM organization

Design cluster1 ( // MPCAM organization cluster 1

.WR(WR),

.RD(RD),

.clk(clk),

.D-core1(D-core11),

.D-core2(gdfx\_temp0),

.D-core3(gdfx\_temp1),

.D-core4(gdfx\_temp2),

.Tagd-core1(Tagd-core11),

.Tagd-core2(gdfx\_temp12),

.Tagd-core3(gdfx\_temp13),

.Tagd-core4(gdfx\_temp14),

.Tags-core1(Tags-core11),

.Tags-core2(gdfx\_temp6),

.Tags-core3(gdfx\_temp7),

.Tags-core4(gdfx\_temp8),

.Dout-core1(Dout-core11),

.Dout-core2(Dout-core21),

.Dout-core3(Dout-core31),

.Dout-core4(Dout-core41));

Design cluster 2( // MPCAM organization cluster 2

.WR(WR),

.RD(RD),

.clk(clk),

.D-core1(gdfx\_temp1),

.D-core2(gdfx\_temp3),

.D-core3(D-core33),

.D-core4(gdfx\_temp5),

.Tagd-core1(gdfx\_temp13),

.Tagd-core2(gdfx\_temp15),

.Tagd-core3(Tagd-core33),

.Tagd-core4(gdfx\_temp17),

.Tags-core1(gdfx\_temp7),

.Tags-core2(gdfx\_temp9),

.Tags-core3(Tags-core33),

.Tags-core4(gdfx\_temp11),

.Dout-core1(Dout-core13),

.Dout-core2(Dout-core23),

.Dout-core3(Dout-core33),

.Dout-core4(Dout-core43));

Design cluster 3( // MPCAM organization cluster 3

.WR(WR),

.RD(RD),

.clk(clk),

.D-core1(gdfx\_temp0),

.D-core2(D-core22),

.D-core3(gdfx\_temp3),

.D-core4(gdfx\_temp4),

.Tagd-core1(gdfx\_temp12),

.Tagd-core2(Tagd-core22),

.Tagd-core3(gdfx\_temp15),

.Tagd-core4(gdfx\_temp16),

.Tags-core1(gdfx\_temp6),

.Tags-core2(Tags-core22),

.Tags-core3(gdfx\_temp9),

.Tags-core4(gdfx\_temp10),

.Dout-core1(Dout-core12),

.Dout-core2(Dout-core22),

.Dout-core3(Dout-core32),

.Dout-core4(Dout-core42));

Design cluster 4(// MPCAM organization cluster 4

.WR(WR),

.RD(RD),

.clk(clk),

.D-core1(gdfx\_temp2),

.D-core2(gdfx\_temp4),

.D-core3(gdfx\_temp5),

.D-core4(D-core44),

.Tagd-core1(gdfx\_temp14),

.Tagd-core2(gdfx\_temp16),

.Tagd-core3(gdfx\_temp17),

.Tagd-core4(Tagd-core44),

.Tags-core1(gdfx\_temp8),

.Tags-core2(gdfx\_temp10),

.Tags-core3(gdfx\_temp11),

.Tags-core4(Tags-core44),

.Dout-core1(Dout-core14),

.Dout-core2(Dout-core24),

.Dout-core3(Dout-core34),

.Dout-core4(Dout-core44));

// N-conjugate shuffle based four clusters...

always @ (NCSC)

assign gdfx\_temp0 = D-core21;

assign gdfx\_temp0 = D-core12;

assign gdfx\_temp1 = D-core31;

assign gdfx\_temp1 = D-core13;

assign gdfx\_temp2 = D-core41;

assign gdfx\_temp2 = D-core14;

assign gdfx\_temp12 = Tagd-core21;

assign gdfx\_temp12 = Tagd-core12;

assign gdfx\_temp13 = Tagd-core31;

assign gdfx\_temp13 = Tagd-core13;

assign gdfx\_temp14 = Tagd-core41;

assign gdfx\_temp14 = Tagd-core14;

assign gdfx\_temp6 = Tags-core21;

assign gdfx\_temp6 = Tags-core12;

assign gdfx\_temp7 = Tags-core31;

assign gdfx\_temp7 = Tags-core13;

assign gdfx\_temp8 = Tags-core41;

assign gdfx\_temp8 = Tags-core14;

assign gdfx\_temp3 = D-core32;

assign gdfx\_temp3 = D-core23;

assign gdfx\_temp4 = D-core42;

assign gdfx\_temp4 = D-core24;

assign gdfx\_temp15 = Tagd-core32;

assign gdfx\_temp15 = Tagd-core23;

assign gdfx\_temp16 = Tagd-core42;

assign gdfx\_temp16 = Tagd-core24;

assign gdfx\_temp9 = Tags-core32;

assign gdfx\_temp9 = Tags-core23;

assign gdfx\_temp10 = Tags-core42;

assign gdfx\_temp10 = Tags-core24;

assign gdfx\_temp5 = D-core43;

assign gdfx\_temp5 = D-core34;

assign gdfx\_temp17 = Tagd-core43;

assign gdfx\_temp17 = Tagd-core34;

assign gdfx\_temp11 = Tags-core34;

assign gdfx\_temp11 = Tags-core43;

end

endmodule

// PROGRAM "Intel Quartus Prime - Version 20.1.0 Lite Edition" "Intel Corporation's design tools"

// CREATED by Allam Abumwais 5.6.2021

//DOWNLOAD https://fpgasoftware.intel.com/eula. "Full components"

//Top design file for 16 cores interconnection using MPCAM organization and NCSC based on multi-cluster.

module multicluster( // input/output ports

WR,

RD,

D-core11,

//cores 21,31,41....

D-core22,

//cores 12,32,42....

D-core33,

//cores 13,23,43....

D-core44,

//cores 14,24,44....

Tagd-core11,

Tagd-core22,

Tagd-core33,

Tagd-core44,

Tags-core11,

Tags-core22,

Tags-core33,

Tags-core44,

Dout-core11,

Dout-core12,

Dout-core13,

Dout-core14,

Dout-core21,

Dout-core22,

Dout-core23,

Dout-core24,

Dout-core31,

Dout-core32,

Dout-core33,

Dout-core34,

Dout-core41,

Dout-core42,

Dout-core43,

Dout-core44

);

input wire WR;

input wire RD;

input wire [31:0] D-core11;

input wire [31:0] D-core22;

input wire [31:0] D-core33;

input wire [31:0] D-core44;

input wire [15:0] Tagd-core11;

input wire [15:0] Tagd-core22;

input wire [15:0] Tagd-core33;

input wire [15:0] Tagd-core44;

input wire [15:0] Tags-core11;

input wire [15:0] Tags-core22;

input wire [15:0] Tags-core33;

input wire [15:0] Tags-core44;

output wire [31:0] Dout-core11;

output wire [31:0] Dout-core12;

output wire [31:0] Dout-core13;

output wire [31:0] Dout-core14;

output wire [31:0] Dout-core21;

output wire [31:0] Dout-core22;

output wire [31:0] Dout-core23;

output wire [31:0] Dout-core24;

output wire [31:0] Dout-core31;

output wire [31:0] Dout-core32;

output wire [31:0] Dout-core33;

output wire [31:0] Dout-core34;

output wire [31:0] Dout-core41;

output wire [31:0] Dout-core42;

output wire [31:0] Dout-core43;

output wire [31:0] Dout-core44;

NCSC b2v\_inst (

.WR(WR),

.RD(RD),

.D-core11(D-core11),

.D-core22(D-core22),

.D-core33(D-core33),

.D-core44(D-core44),

.Tagd-core11(Tagd-core11),

.Tagd-core22(Tagd-core22),

.Tagd-core33(Tagd-core33),

.Tagd-core44(Tagd-core44),

.Tags-core11(Tags-core11),

.Tags-core22(Tags-core22),

.Tags-core33(Tags-core33),

.Tags-core44(Tags-core44),

.Dout-core11(Dout-core11),

.Dout-core12(Dout-core12),

.Dout-core13(Dout-core13),

.Dout-core14(Dout-core14),

.Dout-core21(Dout-core21),

.Dout-core22(Dout-core22),

.Dout-core23(Dout-core23),

.Dout-core24(Dout-core24),

.Dout-core31(Dout-core31),

.Dout-core32(Dout-core32),

.Dout-core33(Dout-core33),

.Dout-core34(Dout-core34),

.Dout-core41(Dout-core41),

.Dout-core42(Dout-core42),

.Dout-core43(Dout-core43),

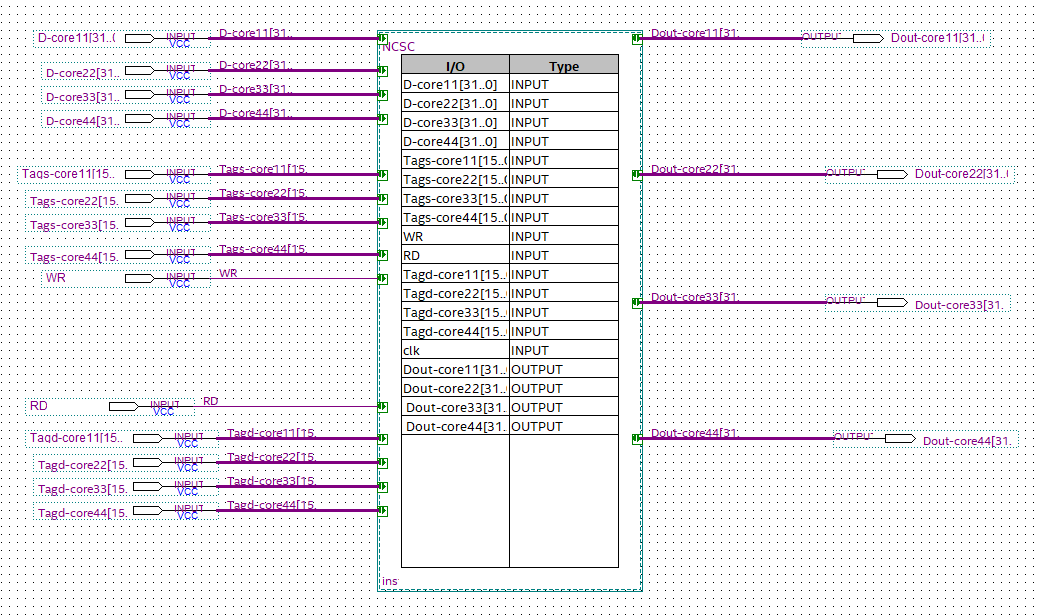
.Dout-core44(Dout-core44));

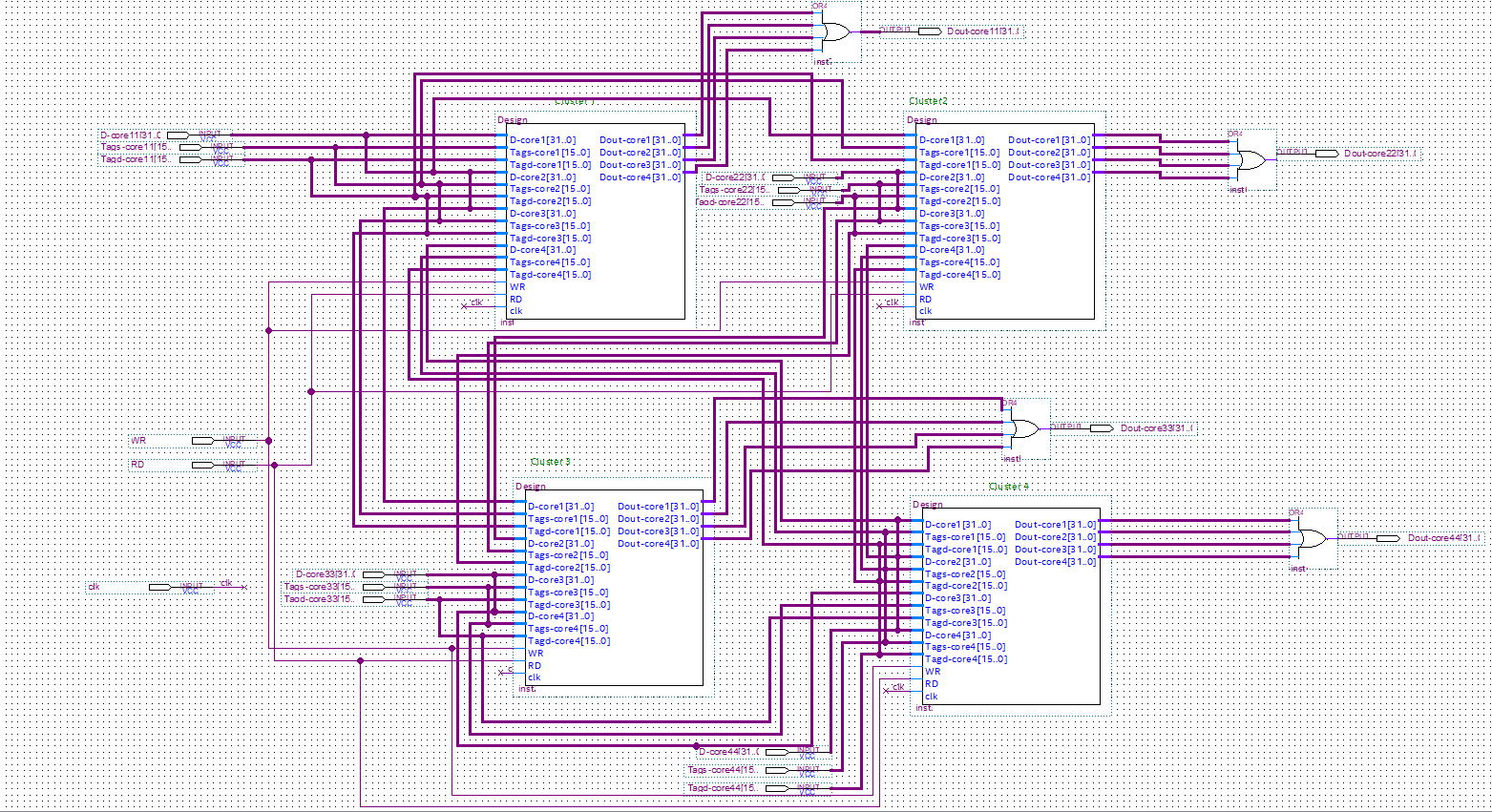
endmodule

**NCSC Design**

**Figure E-1.**

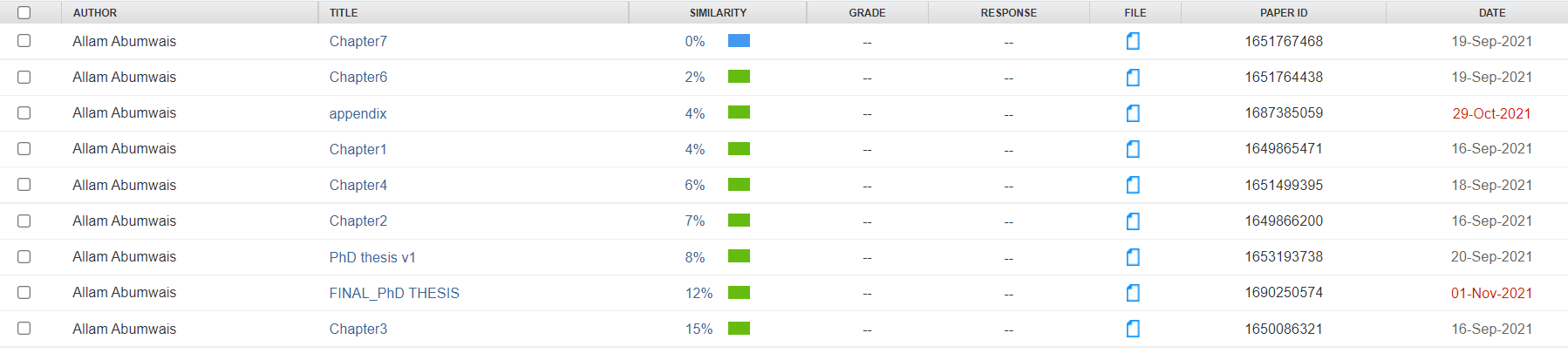
*NCSC Architecture*

****

****

# Appendix X

# Turnitin Similarity Report



# Curriculum Vitae

**PERSONAL INFORMATION**

Surname, Name : Allam R. M. Abumwais

Nationality : Palestinian

Date and Place of Birth : 8March 1987, Jenin.

Marital Status : Married

**EDUCATION**

1. **Schools Attended with Dates**

* Near East University 2018-
* Alquds University 2005-2013
* Science Secondary School Jalqamos, Jenin, 2003-2005
* Primary Almoger School 1993-2002

1. **Qualifications Obtained with Dates**

* PhD in view 2018-
* Masters in Electronic and Computer Engineering Feb, 2014
* BA in Computer Engineering

(Second Class Honors) Feb, 2010

* Secondary Examination (TAWJIHI), July, 2005
* First School Leaving Certificate (FSLC) June,2002

1. **Research Project/ Seminar**

* Location Detection Device, a hardware project.
* Master project “Water demand. neuron-fuzzy prediction method”
* National Diploma “A new Design of Large grain Data flow architecture”
* Master thesis “Non-blocking interconnection network in multicore system”

1. **Publications**

* **A. Mwais &A. Ayyad,“The MPCAM Based Multi-core Processor Architecture: A Contention Free Architecture”, “WSEAS TRANSACTIONS ON ELECTRONICS JOURNAL ”, Dec. 2018.**
* **A. ABUMWAIS, A. AMIRJANOV, K. UYAR and M. ElYYAT, “The Design of a Special Purpose Dual Port CAM- Implementation Using FPGAs”, “CMC-Computers, Materials & Continua”, June. 2021.**

1. **Membership of Professional Bodies/Affiliations**

* Member of the Palestinian & Jordanian Engineers Association.
* Member of Research and Scientific Innovation Society Journals as a Reviewer.

**EMPLOYMENT HISTORY**

* Research Assistant, Computer Engineering Alquds University 2010-2013
* Instructor, Vision Association for Culture and Arts 05/2013 - 01/2015.
* Arab American University (AAUP), Lecturer, Computer System Engineering 2014- Until now.

Here I acquired good Technical and teaching skills and became aware of good Laboratory practice.

**AWARD/PRIZES**

* I have a certificate for my Graduation project at 3rd Palestinian international Conference on computer and information Technology (PICCIT 2010) Organized by Palestine polytechnic university (ppu).

**INTERESTS**

I spend my spare time planning, executing, and love to see the end of achievements of such plans at the set time. I love history books, I also love reading, traveling, trading, meeting, and making friends.

**REFEREES**

**Prof. Dr. ADİL AMİRJANOV**

Computer Engineering

Near East University

**Mobile:** +90

**Email**: adil.amircanov@neu.edu.tr

**Prof. Dr. Mujahed Eleyat**

Deen of Engineering

Arab American University

Palestine

**Mobile:** +972568-513555

**Email**: mujahed.eleyat@aaup.edu

**Dr. KAAN UYAR**

Computer Engineering

Near East University

**Mobile:** +90

**Email**:kaan.uyar@neu.edu.tr