Università degli Studi di Ferrara

DOTTORATO DI RICERCA IN
SCIENZE DELL'INGEGNERIA

CICLO *XXV*

COORDINATORE Prof. Davide Bertozzi

Design and Validation of Network-on-Chip Architectures for the Next Generation of Multi-synchronous, Reliable, and Reconfigurable Embedded Systems

Settore Scientifico Disciplinare ING-INF/01

Dottorando
Dott. Strano Alessandro

(firma)

Tutore
Prof. Bertozzi Davide

(firma)

Anni 2010/2012

Dichiarazione di conformitá

AL MAGNIFICO RETTORE
UNIVERSITÁ DEGLI STUDI DI FERRARA

E-mail: alessandro.strano@unife.it
Oggetto: Dichiarazione di conformitá della tesi di Dottorato
Io sottoscritto: Dott. Strano Alessandro
Nato a: Ferrara
Provincia: FE
Il giorno: 25-12-1983
Avendo frequentato il Dottorato di Ricerca in: Scienze dell'Ingegneria.
Ciclo di Dottorato 25.
Titolo della tesi:
Design and Validation of Network-on-Chip Architectures for the Next Generation of Multi-synchronous, Reliable, and Reconfigurable Embedded Systems.
Tutore: Prof. Bertozzi Davide
Settore Scientifico Disciplinare: ING-INF/01.
Parole chiave della tesi:
Network-on-Chip, Fault-Tolerance, Synchronization, Built-in Self-Testing, Globally-Asynchronous Locally-Synchronous, FPGA, Reconfigurability.

Consapevole, dichiara:

CONSAPEVOLE: (1) del fatto che in caso di dichiarazioni mendaci, oltre alle sanzioni previste dal codice penale e dalle Leggi speciali per l′ipotesi di falsitá in atti ed uso di atti falsi, decade fin dall′inizio e senza necessitá di alcuna formalitá dai benefici conseguenti al provvedimento emanato sulla base di tali dichiarazioni; (2) dell′obbligo per l′Universitá di provvedere al deposito di legge delle tesi di dottorato al fine di assicurarne la conservazione e la consultabilitá da parte di terzi; (3) della procedura adottata dall′Universitá di Ferrara ove si richiede che la tesi sia consegnata dal dottorando in 2 copie di cui una in formato cartaceo e una in formato pdf non modificabile su idonei supporti (CD-ROM, DVD) secondo le istruzioni pubblicate sul sito: http://www.unife.it/studenti/dottorato alla voce ESAME FINALE disposizioni e modulistica; (4) del fatto che l′Universitá, sulla base dei dati forniti, archivierá e renderá consultabile in rete il testo completo della tesi di dottorato di cui alla presente dichiarazione attraverso l′Archivio istituzionale ad accesso aperto EPRINTS.unife.it oltre che attraverso i Cataloghi delle Biblioteche Nazionali Centrali di Roma e Firenze.

DICHIARO SOTTO LA MIA RESPONSABILITA′: (1) che la copia della tesi depositata presso l′Universitá di Ferrara in formato cartaceo é del tutto identica a quella presentata in formato elettronico (CD-ROM, DVD), a quelle da inviare ai

Commissari di esame finale e alla copia che produrró in seduta d′esame finale. Di conseguenza va esclusa qualsiasi responsabilitá dell′Ateneo stesso per quanto riguarda eventuali errori, imprecisioni o omissioni nei contenuti della tesi; (2) di prendere atto che la tesi in formato cartaceo é l′unica alla quale fará riferimento l′Universitá per rilasciare, a mia richiesta, la dichiarazione di conformitá di eventuali copie; (3) che il contenuto e l′organizzazione della tesi é opera originale da me realizzata e non compromette in alcun modo i diritti di terzi, ivi compresi quelli relativi alla sicurezza dei dati personali; che pertanto l′Universitá é in ogni caso esente da responsabilitá di qualsivoglia natura civile, amministrativa o penale e sará da me tenuta indenne da qualsiasi richiesta o rivendicazione da parte di terzi; (4) che la tesi di dottorato non é il risultato di attivitá rientranti nella normativa sulla proprietá industriale, non é stata prodotta nell′ambito di progetti finanziati da soggetti pubblici o privati con vincoli alla divulgazione dei risultati, non é oggetto di eventuali registrazioni di tipo brevettale o di tutela.

PER ACCETTAZIONE DI QUANTO SOPRA RIPORTATO

Ferrara, li 11/03/2013

Firma del Dottorando:

Firma del Tutore:

To Liisi, and the journey of our life together

Design and Validation of Network-on-Chip Architectures for the Next Generation of Multi-synchronous, Reliable, and Reconfigurable Embedded Systems

Alessandro Strano

Abstract

NETWORK-ON-CHIP (NoC) design is today at a crossroad. On one hand, the design principles to efficiently implement interconnection networks in the resource-constrained on-chip setting have stabilized. On the other hand, the requirements on embedded system design are far from stabilizing. Embedded systems are composed by assembling together heterogeneous components featuring differentiated operating speeds and ad-hoc counter measures must be adopted to bridge frequency domains. Moreover, an unmistakable trend toward enhanced reconfigurability is clearly underway due to the increasing complexity of applications. At the same time, the technology effect is manyfold since it provides unprecedented levels of system integration but it also brings new severe constraints to the forefront: power budget restrictions, overheating concerns, circuit delay and power variability, permanent fault, increased probability of transient faults.

Supporting different degrees of reconfigurability and flexibility in the parallel hardware platform cannot be however achieved with the incremental evolution of current design techniques, but requires a disruptive approach and a major increase in complexity. In addition, new reliability challenges cannot be solved by using traditional fault tolerance techniques alone but the reliability approach must be also part of the overall reconfiguration methodology.

In this thesis we take on the challenge of engineering a NoC architectures for the next generation systems and we provide design methods able to overcome the conventional way of implementing multi-synchronous, reliable and reconfigurable NoC. Our analysis is not only limited to research novel approaches to the specific challenges of the NoC architecture but we also co-design the solutions in a single integrated framework. Interdependencies between different NoC features are detected ahead of time and we finally avoid the engineering of highly optimized solutions to specific problems that however coexist inefficiently together in the final NoC architecture. To conclude, a silicon implementation by means of a testchip tape-out and a prototype on a FPGA

board validate the feasibility and effectiveness of the developed design methods in nano-scaled technology sub-systems enabling technology transfer from academia to industry.

Acknowledgements

I have waited for this moment for a long time, basically all my life of student. Now, I wonder what actually makes this moment unique. Today I feel like the first part of an exciting journey is overing and a new one is starting. Probably as a child spends his youth dreaming to become a man similarly I was dreaming to complete my studies to start my job career. However this moment is much more than this, more than the excitement of starting a new era of my life. This moment represents the chance to thank all the people that have helped and supported me on every step until here. This moment can repay a part, at least a small part, of the infinite faith put in me in all these years.

My PhD started when my advisor Davide Bertozzi gave me the opportunity to join the MP-SoC group in Ferrara. Needless to say, Davide has been the key person during my PhD. He has been more than a simple advisor, he did not only show me the way for a prolific research but he also provided me pure lessons of life. In Ferrara, I had the chance to work on ambitious topics in a great research group. A research group built around the harmony and the enthusiasm of challenging always new research problems. These latter group represented the best environment where I could ever imagine to develop my research. Together with endless scientific stimuli, Davide surrounded me by an unconditioned trust and pushed me to go beyond my limits. In these years, I had opportunities that I could neither imagine to have. I traveled around the world taking part to prestigious conferences, I met some of the pioneers in my research field, I exchanged opinions and point of view with persons of every culture and professional background, I had the chance to work in many topics and many countries. I'm really thankful for all these unrepeatable experiences that made me grow and I will always bring with me.

I want to thank all the persons of my research group. Simone and Daniele that have helped me immensely in the first period of my PhD. They represented my reference point for every problem, every doubt and every question. Thanks for all your patience, I have learned a lot from you guys. I'm thankful to Alberto, Luca, Hervé and Marco. It was fantastic to work side by side with you. I

have been impressed by your fairness, your sensitivity and your intelligence. I enjoyed all the moments that we had together. I even enjoyed the most difficult of them as the times when we had to work hard until the early morning hours. Together we have been able to meet the more challenging of the deadlines. Thanks to all of you my friends. I wish you the greatest satisfaction from the life.

A special thank you is due to Luca Benini that had me under his wing as co-advisor for the first year of research. I also owe Igor and Mohammad many thanks for their support with the technical problems I had during my PhD work. I could not forget the guys from Spain: Paco, Crispin and Francisco. I had many enjoyable moments during your internships in Ferrara. Last, I will never forget the help of Arnaud during my internship in Paris and the great collaboration and friendship born with Federico and Eleni during my internship in Lausanne.

Finally, I owe my parents and my whole family to have sustained me in all my choices and to have patiently followed me through all my sad and happy moments of this long way. Their invisible, but still strong and caring presence have encourage me several times.

My last thought is for the person without whom everything else could be meaningless. Liisi, sharing with you the amazing journey of the life is the most beautiful dream I've dreamed with eyes opened.

Alessandro Ferrara, Italy, March 2013

Table of contents

Dichiarazione di conformitá . . . . . **i**

Abstract . . . . . **v**

Acknowledgments . . . . . **vii**

List of Tables . . . . . **xv**

List of Figures . . . . . **xvii**

List of Acronyms and Symbols . . . . . **xxiii**

1 Introduction **1**
- 1.1 Problem Formulation 4
  - 1.1.1 The synchronization design issue 4
  - 1.1.2 The Built-In Self-Testing 6
  - 1.1.3 The Reconfiguration Framework 8
- 1.2 Approach 9
  - 1.2.1 Validation Strategy 10
- 1.3 Organization 11

2 Background **15**
- 2.1 The GALS Design Style 15
  - 2.1.1 The dual-clok FIFO Synchronization Interfaces 16
  - 2.1.2 The Mesochronous Synchronization Interfaces 18
- 2.2 Reliability 19
  - 2.2.1 Built-In Self-Testing and Diagnosis 20
  - 2.2.2 Fault-tolerance 21
- 2.3 Network Reconfiguration 23

3 Relaxing the Synchronization Assumption in Networks-on-Chip . . . **27**

3.1 Limitations of the Fully Synchronous Approach 27
3.2 A Possible Solution: the GALS Design Style 28
3.3 Target GALS Architecture 30
3.4 ×pipesLite switch architecture 33
3.5 The Mesochronous Interface 35
3.5.1 The *Loosely* Coupled Mesochronous Synchronizer . . 36
3.5.2 Tightly Integrated Mesochronous Synchronizer Architecture 38
3.5.3 Theoretical Analysis 40
3.5.4 Experimental Results 45
3.5.5 Mesochronous Link Design Characterization 49
3.6 The Dual-Clock FIFO Interface 56
3.6.1 Tight Integration into the Switch Architecture 60
3.6.2 Latency analysis 62
3.6.3 Throughput analysis 65
3.6.4 Specialized library components 67
3.6.5 Comparative assessment of bi-synch FIFO variants . . 68
3.7 Discussion 74
3.8 Conclusions 75

4 The Moonrake Chip 77
4.1 GALS Systems and Demonstrators 77
4.2 Moonrake Testchip Architecture 79
4.2.1 PIN Requirement 85
4.3 Floorplaning Constraints 86
4.3.1 Area results 89
4.4 Test Setup 91
4.5 Test Results 95
4.6 Conclusions 103

5 Design Space Exploration for Redundancy-Aware NoC Testing 104
5.1 Methodology and Taxonomy 104
5.2 Target Architecture 109
5.3 Testing framework based on handcrafted deterministic test patterns 112
5.3.1 The Testing Strategy 112
5.3.2 Testing Communication Channels 116
5.3.3 TPG for Communication Channels 118

5.3.4 Testing Other Internal Switch Modules 118
5.3.5 Fault Detection and Diagnosis 119
5.3.6 BIST-Enhanced Switch Architecture 120
5.3.7 Experimental results 122
5.4 Built-In Scan Chain-Based Testing Framework 124
5.4.1 The Scan Chain Tool-Flow 125
5.4.2 The Baseline Implementation 127
5.4.3 Customizations for the NoC Setting 128
5.4.4 Experimental Results 129
5.5 Built-In Pseudo-Random Self-Testing 133
5.5.1 The Testing Strategy 134
5.5.2 Testing communication channels 135
5.5.3 Testing multiplexers of the crossbar 136
5.5.4 Testing LBDR 137
5.5.5 Testing Arbiters 139
5.5.6 BIST-enhanced switch architecture 139
5.5.7 Experimental Results 140
5.6 Testing Framework Comparison 142
5.6.1 Stuck-at-faults coverage and testing latency 142
5.6.2 Routing delay 143
5.6.3 Area overhead 144
5.7 Testing Framework for Multi-Synchronous Networks 146
5.7.1 Extension to Multisynchronous Networks 147
5.7.2 Target GALS Architecture 149
5.7.3 Bisynchronous Channel Testing 150
5.7.4 Operating Principle 153
5.7.5 BIST-Enhanced Switch Architecture in a Multisynchronous Scenario 154
5.7.6 Experimental Results 155
5.8 Conclusions 160

6 OSR-Lite: NoC Reconfiguration Framework 163
6.1 Introduction 163
6.2 Native OSR technique 166
6.3 OSR-Lite 169
6.4 OSR-Lite implementation 171
6.4.1 OSR-Lite at the Input Ports 173
6.4.2 OSR-Lite at the Arbiters 173

6.4.3 OSR-Lite at the Output Ports . . . 174
6.5 System-Level Evaluation . . . 175
6.5.1 Propagation . . . 176
6.5.2 Time Overhead . . . 177
6.5.3 Comparison . . . 179
6.6 Synthesis results . . . 181
6.6.1 Area Comparison . . . 182
6.6.2 Routing Delay Comparison . . . 183
6.7 Conclusion . . . 184

7 Co-Optimized Design Methods for General Purpose System . . . 185
7.1 Introduction . . . 185
7.2 Switch Architecture Extensions for Fault-Tolerant NoC Design 192
7.2.1 The New Fault-Tolerant Flow Control: NACK/GO . . 193
7.2.2 Novel Low-Power Fault-Tolerant Arbiter . . . 194
7.2.3 Fault-Tolerance of Routing logic and Buffer FSMs . . 196
7.3 Reconfiguration Mechanism . . . 196
7.3.1 OSR-Lite at the Input Ports . . . 198
7.3.2 OSR-Lite at the Output Ports . . . 200
7.3.3 Fault-Tolerant Reconfiguration Mechanism . . . 202
7.4 Switch Extensions for System Level Notification . . . 203
7.5 The Built-In Self-Testing Framework . . . 206
7.5.1 The Data-Path . . . 207
7.5.2 The Control-Path . . . 209
7.5.3 BIST-enhanced switch architecture . . . 210
7.6 Experimental Results . . . 211
7.6.1 Area and Critical Path . . . 211
7.6.2 Optimized Reconfiguration Support . . . 214
7.6.3 Coverage for single stuck-at faults . . . 214
7.7 Conclusions . . . 215

8 The FPGA Demonstrator . . . 217
8.1 Introduction . . . 217
8.2 FPGA Platform . . . 220
8.3 The System Under Test . . . 221
8.3.1 Basic components: the on-chip network . . . 225
8.3.2 Basic components: the supervision subsystem . . . 227
8.3.3 Basic components: the reconfiguration algorithm . . . 229

8.3.4 The application 230
8.3.5 The physical platform implementation 233
8.4 Validating Built-in Self-Testing and NoC configuration 235
8.4.1 Protocol for BIST notification and configuration . . . 236
8.5 Validating Fault Detection and NoC Reconfiguration 238
8.5.1 Protocol for transient notification and reconfiguration . 240
8.6 Validating NoC Virtualization 241
8.7 Conclusions 243

9 Conclusions 245
9.1 Summary 246
9.2 Major Contributions 248

Bibliography 251

List of Publications 267

List of Tables

3.1 Switch crossing latency. 63
3.2 Dual-Clock FIFO throughput with parameterized buffer depth as a function of sender-receiver frequency ratio. 65
3.3 Throughput of specialized Dual-Clock FIFO variants. 68
3.4 2x2 switch critical path. 69
3.5 5x5 switch critical path. 70

5.1 Coverage for single stuck-at faults. 122
5.2 Test application time and coverage of different testing methods. 123
5.3 Coverage for multiple random stuck-at faults. 124

7.1 Coverage for single stuck-at faults. 215
7.2 Coverage breakdown of data and control path. 215

8.1 Resource utilization of the Virtex 7 chip. 234

List of Figures

3.1 Target Design Platform. 32
3.2 Baseline switch architecture. 33
3.3 GALS switch architecture. 35
3.4 Baseline mesochronous synchronizer architecture of [109]. . . 36
3.5 The loosely coupled mesochronous synchronizer of this work. 37
3.6 Proposed tightly coupled mesochronous synchronizer. 39
3.7 Waveforms example of the tightly coupled mesochronous synchronizer. 41
3.8 The hybrid architecture with a 1-bit mesochronous synchronizer on the receiver end. 43
3.9 Test-case platform under analysis. 45
3.10 Normalized cycle latency of the different synchronization schemes. 46
3.11 Area breakdown of a switch block with its synchronization scheme. 47
3.12 Normalized power consumption of different synchronization schemes in different traffic scenarios. 49
3.13 Operating frequency and tolerated link delay of different synchronizers. 50
3.14 Basic mechanisms affecting skew tolerance. 51
3.15 T_{setup} and T_{hold} for the loose coupled varying the skew tolerance. 52
3.16 T_{setup} and T_{hold} for the tight coupled varying the skew tolerance. 52
3.17 Setup time as a function of negative skew. 54

3.18 Dual-Clock FIFO Architecture. 56
3.19 Sampling of input data. 58
3.20 Vanilla switch and Dual-Clock FIFO integration into one input port of the NoC switch architecture. 60
3.21 64
3.22 Specialized Dual-Clock FIFO. 67
3.23 Post-layout normalized results of area (a) and power (b) for a switch with a dual-clock FIFO synchronizer. 71
3.24 Area (a) and power (b) consumption of baseline and specialized dual-clock FIFO architectures with different buffer depths. 73
3.25 Area occupancy of NoC switches with different synchronization interfaces. 74

4.1 Block diagram of the NoC testchip. 80
4.2 Synchronous sub-systems: (a) the *Synch_fast* design (on the left-side) and (b) the *Synch_slow* design (on the right-side). . 81
4.3 Loosely coupled sub-systems with mesochronous synchronizers: (a) the *Asynch_Loose_Slow* design (on the left-side) and (b) the *Asynch_Loose_Fast* design (on the right-side). 82
4.4 Hybrid coupled sub-systems: (a) the *Asynch_Hybrid_Slow* design (on the left-side) and (b) the *Asynch_Hybrid_Fast* design (on the right-side). 84
4.5 Dual-clock FIFO design. 85
4.6 NoC testchip floorplan. 87
4.7 Source synchronous communication in the hybrid coupled sub-systems and PnR constraints. 88
4.8 Source synchronous communication in the loosely coupled sub-systems and PnR constraints. 88
4.9 Source synchronous communication in the dual-clock FIFO sub-system. 89
4.10 Area breakdown of the seven sub-systems. 91
4.11 Verigy test platform. 92
4.12 Frequency and skew sweep in the *Asynch_Hybrid_Slow* sub-system. 97

4.13 Frequency and skew sweep in the *Asynch_Loose_Slow* subsystem. 98
4.14 Percentage of working chips in each test case. 101
4.15 Relative power comparison. 102

5.1 Modular structure of the baseline switch architecture. Not all connections are showed. 110
5.2 LBDR logic and requirements on the diagnosis outcome. . . . 111
5.3 The cooperative and concurrent testing framework saving TPG instances and covering their faults. 114
5.4 Practical implementation of communication channel testing. . 116
5.5 TPG for communication channels. 117
5.6 BIST-enhanced switch architecture. 121
5.7 Area overhead for BIST implementation as a function of target speed. 122
5.8 Coverage of TPG faults. 123
5.9 Latency breakdown of a scan chain-based test. 126
5.10 Practical implementation of the proposed scan chain-based test. 129
5.11 Maximum number of test patterns. 130
5.12 Latency of the scan chain-based test. 130
5.13 Total number of bits stored by the test patterns generator. . . . 131
5.14 Area overhead breakdown of the customized scan chain-enabled testing strategy. 131
5.15 Area overhead breakdown of the scan chain-based and deterministic test patterns-based solutions. 132
5.16 Latency of the scan chain-based and deterministic test pattern-based solutions. 132
5.17 Coverage of the scan chain-based and deterministic test pattern-based solutions. 133
5.18 Optimization steps of the pseudo-random testing framework. . 138
5.19 BIST-enhanced switch architecture. 140
5.20 Coverage for single stuck-at faults as a function of the test latency. 141

5.21 Area overhead for BIST implementation. 141
5.22 Coverage for single stuck-at faults as a function of the test latency. 143
5.23 Routing delay for BIST implementations. 144
5.24 Area overhead for BIST implementations. 145
5.25 Cooperative testing framework for bisynchronous communication channels. 148
5.26 Baseline bisynchronous communication channel. 149
5.27 Multisynchronous testing framework. 151
5.28 Proposed triple-stage brute-force synchronizer (a) and waveforms of synchronizers without (b) and with (c) *set* port. . . . 152
5.29 Bisynchronous channel operating principle. 155
5.30 Single stuck-at faults coverage as a function of test pattern count (a) and area overhead for BIST implementation (b). . . . 158
5.31 Bisynchronous framework test time as function of ATA and TPG frequencies. 159

6.1 Two NoC configurations where the routing algorithm needs to be adapted. 164
6.2 Channel dependency graph for two routing algorithms and the combination of both. 167
6.3 Reconfiguration steps performed in an OSR environment. . . 168
6.4 Token advance in a network: (a) check for absence of old messages and input ports epoch, (b) token signal propagation. The token separates old traffic from new traffic. 169
6.5 Reconfiguration steps performed in an OSR-Lite environment. 170
6.6 Switch input buffer enhanced with the OSR-Lite logic and a new set of routing mechanism. 172
6.7 Switch arbiter enhanced with the OSR-Lite logic. 174
6.8 Switch output buffer enhanced with the OSR-Lite logic. 174
6.9 Configuration information from neighbor switches and control network 176

6.10 OSR-Lite propagation over a 4 × 4 2D mesh topology: (a) scrolling up, and (b) scrolling down. 177
6.11 (a) Average message latency at different injection rates for SR routing on 8 × 8 2D mesh (b) OSR-Lite propagation over a 8 × 8 2D mesh topology at different injection rates. 178
6.12 Average message latency with (a) hotspot traffic and uniform traffic ((b) medium network load and (c) high network load). . 180
6.13 5x5 switch (a) area and (b) routing delay comparison. 182

7.1 Fault tolerant arbiter implementation. 195
7.2 Nack-Go switch 197
7.3 OSR-Lite logic extended for the NACK/GO input buffer. . . . 199
7.4 Counter of packets of the two epochs. 200
7.5 Configuration information from neighbor switches and control network 201
7.6 OSR-Lite logic extended for the NACK/GO output buffer. . . 201
7.7 Fault-tolerant OSR-Lite logic. 203
7.8 Transient fault notification. 204
7.9 Dual network routing primitive. 205
7.10 TMR approach with per-primitive voting system. 206
7.11 Pratical implementation of data-path testing. 207
7.12 BIST-enhanced switch architecture. 210
7.13 Area of the GP-NaNoC switch. 212
7.14 Routing delay. 213
7.15 Efficiency of Single-LBDR implementation. 214

8.1 VC707 baseline prototyping board. 220
8.2 FPGA platform overview. 222
8.3 Design flow for platform implementation. 224
8.4 Basic components of the on-chip network. 225
8.5 LBDR routing logic extension for two cores per switch support. 228
8.6 The matrix multiplication at work. 230

8.7 The semaphores of the matrix multiplication application. . . . 232
8.8 Layout of the full FPGA design. Green: data NoC; red: Network Interfaces; yellow: dual NoC; cyan: MicroBlazes and other logic. 234
8.9 Built-in-Self-Testing at work. 235
8.10 Transient fault detection and reconfiguration. 238
8.11 Network regions before (a) and after virtualization (b). Note that the arrows indicate the logical application flow, not necessarily the route followed by packets. For case (b), the arrows are only indicative of partitioning, but the pipeline sequence is in fact shuffled for verification purposes. 241
8.12 Virtualization request and reconfiguration. 242

List of Acronyms and Symbols

ASIC	Application-Specific Integrated Circuit
CMP	Chip MultiProcessor
CTS	Clock Tree Synthesis
DVFS	Dynamic Voltage and Frequency Scaling
FF	Flip-Flop
FHD	Full High Definition
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GALS	Globally Asynchronous Locally Synchronous
GPP	General Purpose Processor
GPU	Graphics Processing Unit
HDL	Hardware Description Language
IP	Intellectual Property
ITRS	International Technology Roadmap for Semiconductors
LEF	Library Exchange Format
LUT	Look-Up Table
MIN	Multi-stage Interconnection Network
MPSoC	Multi Processor System on Chip
MUX	Multiplexer
NI	Network Interface
NoC	Network-on-Chip
OCP	Open Core Protocol
P&R	Place and Route
PDA	Personal Data Assistant
RR	Round Robin
RTL	Register Transfer Level
SOCE	Cadence SoC Encounter
SR	Search&Repair
SoC	System on Chip
TLM	Transaction-Level Modeling
TTM	Time-to-Market
VLSI	Very Large Scale Integration
LBDR	Logic-Based Distributed Routing
TPG	Test Pattern Generator

ATA	Auto-Test Analyzer
BIST	Built-In Self-Test
BISD	Built-In Self-Diagnosis
DUT	Device Under Test
TRC	Two-Rail Checker
MISR	Multiple Input Signature Register
LFSR	Linear Feedback Shift Register
MTTF	Mean Time To Failure

1 Introduction

THE embedded system market is rapidly growing and features a rich variety of devices that are able to perform a wide multitude of diverse tasks. Nowadays, appliances such as mobile phones, personal data assistants (PDA) and ebook readers became mainstream in our everyday life. These mobile devices are ubiquitous, can be utilized everywhere and their applicability range span from pure computational tasks, through entertainment up to social network connectivity. The tremendous complexity reached by such devices represents a major challenge faced by engineers that have to design systems under a constant and relentless time-to-market (TTM) pressure. In order to shorten such TTM, the design of such devices is traditionally performed by integrating existing components in a plug–and–play fashion into a System-on-Chip (SoC) [1]. Therefore, a major challenge consists of interconnecting many different components with each other in an efficient way. According to ITRS roadmaps [103], thousands of cores will be integrated in a single chip during the next few years. Such scenario opens up many questions regarding scalability issues as all the cores in the single chip will have to be interconnected in a power efficient and scalable way.

Classically, intellectual properties (IPs) (e.g., memory controller, CPUs, GPUs, etc.) designed by different vendors are interconnected by dedicated buses. AMBA, AXI, AHB [75,76] represent well-established industrial examples of such interconnection architectures. Unfortunately, they do suffer from well known scalability problems due to arbitration penalties. This is one of the driver dictating the adoption of a more scalable interconnection scheme: Networks-on-Chip (NoCs). NoC architectures represent a viable, scalable packet-switched micro-network interconnect scheme alternative to classical bus architectures [110]. They are generally believed to be the long term solution to the communication scalability issue.

Today, networks-on-chips (NoCs) implement the communication backbone of virtually all large-scale system-on-chip (SoC) designs in 45nm and below. Despite their fast diffusion in products and roadmaps, today's NoC reality was not fully encompassed in the early position papers [2,4]. After twelve years of trial-and-error, design experiences and focused research ([3,5,7,8]), designers have gained the awareness of the profound difference of on-chip vs. off-chip interconnect design [13], of the tight constraints of an on-chip setting [6], of the challenges posed by nanoscale technologies [9] and of the differentiated requirements of specific application domains [10](e.g., application-specific heterogeneous NoCs vs. general-purpose homogeneous NoCs). As a result, NoC design principles have recently reached a stage where they start to stabilize, in correspondence to their industrial uptake [11, 12].

Unfortunately, the requirements on embedded system design are far from stabilizing and an unmistakable trend toward enhanced reconfigurability is clearly underway. Reconfigurability of the HW/SW architecture would in fact enable several key advantages, including on-demand functionality, on-demand acceleration, shorter time-to-market, extended product life cycles and low design and maintenance costs. Supporting different degrees of reconfigurability in the parallel hardware platform cannot be however achieved with the incremental evolution of current design techniques, but requires a disruptive approach and a major increase in complexity. At the same time, fault tolerance was previously an issue only for specific applications such as space or avionics. Today, due to the increased variability of components and breadth of operating environments, reliability becomes relevant to mainstream applications. Similarly, new reliability challenges cannot be solved by using traditional fault tolerance techniques alone: the reliability approach must be part of the overall reconfiguration methodology.

An even more daunting challenge for NoC designers consists of coming up with synchronization strategies for systems where the chip-level synchronization assumptions have been relaxed. Traditional globally synchronous clocking circuits have become increasingly difficult to design with growing chip size, clock rates, relative wire delays and parameter variations. Additionally, high speed global clocks consume a significant portion of system power and lack the flexibility to independently control the clock frequencies of submodules to achieve high energy efficiency. Therefore, ad-hoc counter measures must be adopted to relax the synchronization assumption within the system.

In the highly parallel landscape of modern embedded computing platforms, the system interconnect serves as the framework for platform integration and

is therefore key to materializing the needed flexibility and reliability properties of the system as a whole. Therefore, time has come for a major revision of current NoC architectures in the direction of relaxed synchronicity and increased reconfigurability and reliability.

In addition, a key property that novel NoCs cannot miss is to guarantee a potentially fast path to industry, since NoC deployment is today a reality. An important requirement for this purpose is the efficient testability of candidate NoC architectures. This property is very challenging due to the distributed nature of NoCs and to the difficult controllability and observability of its internal components. When we also consider the pin count limitations of current chips, we derive that NoCs will be most probably tested in the future via built-in self-testing (BIST) strategies.

Although there is still ample room to research novel approaches to the specific challenges of the NoC architecture, one key concern for this new revision round will be to co-design the solutions to different challenges in an integrated framework. In practice, interdependencies between different NoC features should be detected ahead of time so to avoid the engineering of highly optimized solutions to specific problems that however coexist inefficiently together in the final NoC architecture.

The major contribution of this thesis consists of taking on the challenge of engineering a NoC architectures for the next generation of multi-synchronous, highly reconfigurable and reliable systems. The thesis tackles the challenge in three ways:

- Provides design methods able to overcome the conventional way of implementing testability, reconfigurability and multi-synchronous designs. In fact, synchronization interfaces are proposed and optimized in terms of area, latency and power bringing a large energy savings that make a multi-synchronous NoC affordable at almost the same area and power cost of its synchronous counterpart. In the reliability context, the thesis presents four testing methods based on built-in self-test and self-diagnosis infrastructures making efficient use of NoC structural redundancy in cooperative testing frameworks. Ultimately, the reconfiguration is performed by means of a cost-effective, complete and transparent reconfiguration process that has never implemented on-chip before and is able to drain at run-time the network at link/router level.

- The thesis detects the interdependencies between the different design features and addresses them all in a coherently integrated final NoC

switch. The design methods for testability, reconfigurability and multi-synchronous designs are co-designed, co-optimized and finally integrated with state-of-art fault-tolerant techniques to coexist efficiently in the final NoC architecture.

- The proposed design methods have been validated by means of ASIC implementation and FPGA prototyping proving the practical relevance of the thesis research. The synchronization interfaces have been integrated in the first multi-synchronous ASIC demonstrator in 40 nm CMOS process. On the contrary, methods for testability and reconfigurability require non silicon-dependent strategies and they have been validated on a leading-edge Virtex-7 FPGA.

In the next section, we will start by looking at the problem of the synchronization design (Section 1.1.1). Then Section 1.1.2 presents the challenges related to reliable systems while Section 1.1.3 illustrates the issues of NoC reconfigurations. The key approach proposed in this thesis is described in Section 1.2 and finally, Section 1.3 provides an overview of the remaining chapters of the thesis.

1.1 Problem Formulation

While technology is providing unprecedented levels of system integration, it is also bringing new severe constraints to the forefront: power budget restrictions, overheating concerns, circuit delay and power variability, permanent faults affecting the system right from the beginning (manufacturing faults) or with progressive onset (wear-out faults), increased probability of transient faults. Such constraints are driving the migration into new forms and shapes of synchronicity, reconfigurability, testability and fault-tolerance. Hereafter challenges brought by the design of these latter features are detailed with emphasis on the inability of current NoC literature to effectively tackle them as a whole in a single integrated framework.

1.1.1 The synchronization design issue

The maximum operating speed of the network architecture should not constrain the speed of the networked IP cores. This calls for proper decoupling at the network boundary by means of synchronization interfaces. This is a key requirement also for power management strategies in the embedded computing

domain, requiring each core to run at an independent and runtime variable voltage and speed. Nowadays, both application requirements and technology effects call for a disruptive evolution of the synchronization architecture. In fact, distributing a global clock throughout the entire chip with tighly controlled skew is becoming increasingly power inefficient and even infeasible. Indeed, shifting the focus to a pure silicon technology viewpoint, there are some other very important challenges to be faced by both industry and academic research. In fact, as technology advances into aggressive nanometer-level scaling, several design challenges emerge from technology constraints which require a continuous evolution of the interconnection implementation strategy adopted at the circuit and architectural levels. Synchronization of current and future chips with a single clock source and negligible skew is extremely difficult if not close to be impossible [1]. Indeed, synchronization is today definitely among the most critical challenges in the design of a global on-chip communication infrastructure, as emerging technology variability, signal integrity, power dissipation limits are contributing a severe break-down of the global synchronicitiy assumption when logical structures spans more than a couple of *mm* on the die [14]. NoCs typically span the entire chip area and there is now little doubt on the fact that a high-performance and cost-effective NoC in 45nm and beyond can only be designed under relaxed synchronization assumptions [86]. A solution would be to design such systems using a fully asynchronous global intra-chip communication. Such choice would eliminate the clock distribution concern and would make designs more modular since timing assumptions are explicitly handled in the hand-shaking protocols. Unfortunately, current design tools and IP libraries heavily rely on the synchronous paradigm instead, thus making intermediate solutions more attractive and affordable in the short run. As previously anticipated, synchronizer-based globally asynchronous locally synchronous (GALS) systems represent an appealing solution in the mid term, and is therefore the focus of this thesis. In such systems, the design can be partitioned in different frequency islands and the interconnection infrastructure can be envisioned as mesochronous domain (isolated by dual-clock FIFOs at the boundary) where a single clock spans the whole communication infrastructure area relying on a loose synchronization assumption. Such mesochronous assumption allows to tolerate an arbitrary amount of space dependent time-invariant phase offset (i.e., skew) among the leaves of the clock signal hence resulting in a lower power clock tree synthesis. Unfortunately, the high cost of traditional synchronizer implementations in terms of area, power and latency is typically the main reason preventing their adoption as intermediate solution. Moreover, this is only one of the possible GALS implementation variants

within a large design space where it is difficult to select the best solution for the underlying design. Last, there is a general skepticism of industrial designers to relax the synchronization assumption in their chips due to the usage of unconventional tool capabilities, to the poor predictability of resulting designs and to the threat of process variations.

Aware of these challenging problems, the first goal of this thesis is to develop all the required support for the building process of such GALS systems. In particular, our contribution and emphasis is on the NoC infrastructure that is augmented with all the necessary library components in a sort of *galsification* process. Such GALS blocks are designed to considerably reduce area, power and latency issues. Furthermore, we perform a crossbenchmarking between implementation variants, resulting in actual guidelines for designers that want to migrate from synchronous to GALS solutions. Finally, the development effort of the synchronization library is validated with the tape-out of the first multi-synchronous ASIC demonstrator in 40 nm CMOS process.

1.1.2 The Built-In Self-Testing

On-chip interconnection networks are rapidly becoming the reference communication fabric for multi-core computing platforms both in high-performance processors and in many embedded systems [15, 24]. As the integration densities and the uncertainties in the manufacturing process keep increasing, complementing NoCs with efficient test mechanisms becomes a key requirement to cope with high defect rates [16, 43]. Above all, the NoC testing infrastructure should not be conceived in isolation, but should be coherently integrated into a reliability framework taking care of fault detection, diagnosis and network reconfiguration and recovery to preserve yield [23].

Moreover, wear-out mechanisms such as oxide breakdown, electro-migration and mechanical/thermal stress become more prominent in aggressively scaled technology nodes. These breakdown mechanisms occur over time, therefore the methodology and the infrastructure used for production testing should be designed for re-use during the system lifetime as well, thus enabling graceful degradation of the NoC over time.

The detection and identification of failures is the foundation of any reliability framework. Unfortunately, developing such a testing infrastructure for a NoC is a serious challenge. The controllability/observability of NoC links and sub-blocks is relatively reduced, due to the fact that they are deeply embedded and spread across the chip. Also, pin-count limitations restrict the use of I/O pins

dedicated for the test of the different NoC components. Traditional approaches to NoC testing rely on full-scan or boundary scan testing, but cannot avoid the significant overhead associated with the design for testability (DfT) infrastructure [45]. Moreover, the resulting test application times and test pattern generation times are hardly affordable for applications of practical relevance. A number of other concerns were raised in [44] on the use of external testers for nanoscale chip testing:

- First, the lack of scalability of test data volumes with the number of gates hidden behind each package pin.
- Second, the need for testing at full clock speed, which is overly expensive if even possible to accomplish with external testers.
- Third, the poor suitability of these latter for lifetime testing (and not just for production testing) and for a test-and-repair testing approach (beyond the baseline go/no-go philosophy).

As an effect, a migration from external testers to built-in self-test (BIST) infrastructures was envisioned in [44], and was later confirmed by the large amount of works in the open literature targeting scalable BIST architectures for NoC testing [26, 46, 47]. At the same time, the limited fault coverage that functional and pseudo-random testing can achieve on the control path of NoC switches when test generators are outside the switch has further pushed the adoption of BIST units at least for such control blocks [17].

While BIST techniques seem the best suitable solution to tackle permanent failures in NoCs, transient faults can not be handled by such strategies as they appear and disappear unpredictably. BIST techniques must be therefore integrated in fault-tolerant systems able to satisfy the high reliability constraints imposed by modern environments. In this direction, this thesis does not simply propose methods based on full BIST strategies but it also provides methods to integrate and optimize the BIST mechanisms in fault-tolerant systems.

In particular, the thesis proposes four different BIST strategies and the coherent integration of the most promising of them in a fault-tolerant switch micro-architecture thus able to address both permanent and transient failures. A key principle of the thesis approach consists of exploiting the inherent structural redundancy provided by NoCs. Each switch is comprised of input ports, output ports, arbiters and FIFOs that are duplicated for each channel. This feature is used to develop a very effective test strategy which consists of testing multiple identical blocks in parallel and of cutting down on the number of test

pattern generators. This is done both at the abstraction level of the switch micro-architecture (e.g., testing of the output port arbiters in parallel) and of the NoC architecture (i.e., testing of all NoC switches in parallel). The inherent parallelism of our BIST procedure makes our testing infrastructure highly scalable and best suited for large network sizes.

The proposed BIST procedures are suitable both for production and for lifetime testing, and is complemented by a built-in self-diagnosis logic distributed throughout the network architecture able to pinpoint the location of detected faults in each switch. This diagnosis outcome matches the reconfigurability requirements of logic-based distributed routing and is therefore the stepping stone into the novel network reconfiguration strategy that will be proposed in this thesis and described in next sections.

1.1.3 The Reconfiguration Framework

As technology advances chip multiprocessors (CMPs) and multiprocessor systems-on-chip (MPSoCs) have been accepted as the method to enhance the computing power and the functionality of current high-end systems. An ever increasing number of cores and devices can be added to the same chip enhancing its functionality and power. Current products already include tens and hundreds of devices (e.g. Tilera with 100 cores [25]). These systems rely on the Networks-on-Chip (NoC) concept where a high-performance interconnect is built inside the chip, allowing efficient interconnection between all the devices.

As complexity keeps increasing, there emerge new requirements that affect how the interconnect is designed. As mentioned in the previous sections, the system must support a wide range of faults. These latter impact the system configuration rendering the network affected. Besides reliability concerns, there is also an interest in providing further functionality to the system. Virtualizing the entire chip into sets of virtual regions and assigning them to different applications running concurrently is appealing in those systems. Similarly, powering down unused resources during most of the time is becoming compulsory to keep power consumption levels to reasonable bounds. In both cases, the NoC component is affected as it needs to be reconfigured to the changing environment.

To address the new functionalities, the NoC must be enriched with an efficient reconfiguration process which enables the smooth and transparent transition between system configurations.

When the topology of the network changes, either involuntarily due to failing/faulty components or voluntarily due to removal of addition of nodes, the network routing algorithm must be reconfigured in order to (re)establish full network connectivity among the attached nodes. In transitioning between the old and the new routing functions during network reconfiguration, additional dependencies among network resources may be introduced, causing what is referred to as reconfiguration-induced deadlock.

Current techniques typically handle this situation through static reconfiguration, meaning that application traffic is stopped and, usually, dropped from the network during the reconfiguration process. While this approach guarantees the prevention of reconfiguration-induced-deadlock, it can lead to unacceptable packet latencies and dropping frequencies for many applications, particularly realt-time and quality-of-service (QoS) applications.

As alternative it can be implemented a dynamic reconfiguration. This latter allows user traffic to continue uninterruptedly during the time that the network is reconfigured, thus reducing the number of packets that miss their real-time/QoS deadline. However such technique requires effort to avoid deadlock situations and typically brings extra resources to the network.

In this thesis we advance state-of-the-art in reconfiguration frameworks for NoC-based systems. Anyway, instead of designing a brand new reconfiguration mechanism, it is recognized the large amount of bibliography and proposals made for reconfiguration mechanisms in high-performance off-chip networks. In this sense, it is picked the approach that better suits the NoC domain and the tight resource budgets of the on-chip environment. However, in its native form its implementation in NoCs is out-of-reach. Therefore, we provide a careful engineering of the NoC switch architecture and of the system-level infrastructure to support a cost-effective, complete and transparent reconfiguration process.

1.2 Approach

This thesis takes on the challenge of designing the next-generation of multi-synchronous, reliable, and reconfigurable embedded systems by proposing new effective ad-hoc methods and by capturing their interdependencies in a co-designed and co-optimized final architecture.

Specifically, in order to tackle the *synchronization* problem, we design novel globally asynchronous locally synchronous (GALS) interfaces able to meet

different layout constraints. In particular, we design dual-clock FIFO interfaces to provide frequency decoupling between domains and mesochronous interfaces to relax the constraints of the clock tree within a frequency domain. The proposed interfaces are able to show that it is possible to migrate from the synchronous to the GALS paradigm with a negligible area and power cost without impacting performance.

The thesis also proposes and validates methods for *reliable and reconfigurable* embedded systems. Firstly, we present four scalable built-in self-test and self-diagnosis infrastructures making efficient use of NoC structural redundancy for testing and diagnosis purposes through the use of a cooperative testing framework. Secondarily the best suited proposed BIST framework has been integrated in an advanced multi-features switch, called GP-NaNoC. This latter integrates the most relevant and innovative design methods conceived throughout the thesis and makes sure they co-exist together. The switch is enhanced with a transparent system reconfiguration mechanism, called Overlapping Static Reconfiguration (OSR-Lite), and a novel flow-control protocol based on detection, correction and retransmission providing fault-tolerance to the switch.

1.2.1 Validation Strategy

All the proposed design methods have been validated by means of demonstrators able to prove their practical relevance. In particular, two ad-hoc demonstrators have been implemented in order to ensure the robustness and the effectiveness of the proposed design methods. The synchronization methods have been validated through an ASIC demonstrator in 40 nm CMOS process while the testability and reconfigurability methods have been implemented on a Virtex-7 FPGA prototype. Both technology and switch fabric adopted for the demonstrators have been consciously chosen to properly validate the underlining design methods that they integrate.

The synchronization interfaces have been integrated in the first multi-synchronous ASIC demonstrator in 40 nm CMOS process. The tape-out on silicon served as demonstrator of the robustness of the synchronization interfaces against process variability and operating conditions (e.g., speed ratio, clock phase offset). The demonstrator for synchronization methods improves the maturity of the developed GALS technology and bridges the final gap to actual silicon implementation. In particular, the testchip design and fabrication process validates the feasibility and effectiveness of the developed GALS NoC concept in nano-scaled technology sub-systems by comparing synchronous and GALS synchronization technology in a homogeneous experimental set-

ting: same baseline designs, same manufacturing process, same die. Finally, the testchip fabrication breaks the barriers that have prevented the success of synchronizer-based technology for GALS NoCs so far in alternative design experiences.

As mentioned, ASIC technology is adopted to prove the robustness of synchronization interfaces against timing constraints of integrated circuits. Indeed the strict dependency of these latter methods to constraints proper of ASIC technology did not allow us to consider alternative technologies for the first demonstrator. On the contrary, FPGA technology has been chosen to implement the second demonstrator integrating non-silicon dependent methods. In fact, testability and reconfigurability methods have been validated together with fault-tolerant techniques and control signaling strategies in a Xilinx Virtex-7 FPGA. In particular, a switching fabric based on the GP-NaNoC switch supporting network partitioning and isolation as well as irregularities has been designed and prototype on the second FPGA-based demonstrator. This latter especially validates boot-time testing and configuration, runtime detection of faults, runtime reconfiguration of the routing function, dynamic virtualization of the switching fabric. The NoC prototype implemented on the second demonstrator is a key enabler for the materialization of the needed flexibility and reliability properties of next-generation embedded systems.

1.3 Organization

The contributions of this thesis are organized in 9 chapters. Before presenting the contributions, Chapter 2 first provides the necessary background of the work in this thesis. It surveys built-in-self-testing, fault-tolerant and reconfiguration strategies as well as globally-asynchronous locally synchronous interfaces for the building of GALS systems. Finally, it summarizes the shortcoming of the presented work to be addressed in subsequent chapters.
Chapter 3 introduces the synchronization design issue. In a first step, the motivation for adopting synchronization mechanisms in the NoC environment will be discussed. Next, the target GALS platform of this thesis along with the architecture of the basic switch block required to build it will be highlighted. Last, it is presented the baseline mesochronous synchronizer and the dual-clock bi-synchronous FIFO with focus on all their improvements that led to a new fully integrated and flexible switch architecture.
Chapter 4 presents the GALS ASIC demonstrator in 40 nm CMOS process. The chip, named Moonrake, validates on silicon the robustness of the GALS

interfaces proposed in the previous Chapter. Firstly it is described the testing environment setup and the floorplanning constraints of the chip. Later, the different GALS subsystems integrated into the chip are presented with their own synchronization properties. Finally, the experimental section presents the performance of the subsystems.
Chapter 5 provides an exploration of built-in testing strategies customized for NoC-based systems comparing them under an area, coverage and latency point of view. In particular, the first section proposes a BIST mechanism exploiting architecture behavior knowledge to come up with a set of customized test patterns for NoC components. The second section presents a strategy based on a scan chain mechanism. In the third section, we design a testing based on pseudo-random patterns where we cut down on the test application time and we provide efficient testing of the control path. In the fourth section, we propose one of the first built-in self-testing and diagnosis framework for globally-asynchronous locally-synchronous NoC based on an asynchronous handshaking.
Chapter 6 presents the implementation of the fast and transparent reconfiguration mechanism called OSR-Lite. The Chapter describes the complete reconfiguration mechanism at work, involving the BIST mechanism, a central manager and a notification infrastructure. It is showed the implementation at micro-architecture level of the logic enabling the mechanism and the propagation of the reconfiguration tokens. Finally, the OSR-Lite mechanism is evaluated in terms of area and latency.
Chapter 7 extends the work presented in the previous chapters by proposing a switch (GP-NaNoC switch) integrating BIST, reconfiguration and fault-tolerance features in a single co-designed architecture. First of all, the switch architecture extensions for fault tolerant NoC design are presented. Especially, it is highlighted a novel protocol (NACK/GO) based on retransmission and a fault-tolerant arbiter. Thus the Chapter presents the optimizations introduced in the BIST mechanism and the OSR-Lite logic achieved by co-designing the reconfiguration and testing strategies with the fault-tolerant architecture. Next, the switch extensions for system level notification are described and coverage, area and critical path of the novel switch are finally reported.
Chapter 8 reports about the prototyping of the design methods proposed in the previous Chapters on a Xilinx Virtex-7 FPGA. The FPGA prototype comprises a large number of components that enables observability, controllability and debugability of the system under test. All this components are described in detail. Last, it is illustrated the application running on the system and the strategies adopted to validate the BIST, the fault-tolerance and the reconfigu-

ration features of the system.
Finally, Chapter 9 provides concluding remarks on the work presented. The chapter summarizes the thesis and outlines its contributions.

2
Background

THIS chapter starts by surveying several works in the domain of globally asynchronous locally synchronous (GALS) Networks-on-Chip. Furthermore, contributions concerning testing techniques and fault-tolerant systems are also surveyed. Last, works in the domain of network reconfiguration are reported. The chapter highlights also the contribution of this thesis with respect to the work available in the open literature.

2.1 The GALS Design Style

When the target system requires the interconnection of several components working at different frequency, the adopted synchronization scheme and its implementation is key for a successful result of the final system design.

A possible solution is to adopt the GALS philosophy which can be seen as an intermediate design style between the fully synchronous and fully asynchronous solutions. The GALS design paradigm was first proposed in [131] and consists basically of partitioning the system architecture in independent synchronous islands while the communication between them is achieved asynchronously. It is therefore a natural enabler for low-power dynamic voltage and frequency scaling (DVFS) and low noise. The GALS paradigm has been frequently experimented by using asynchronous logic [87, 89]. ETH labs developed a complete GALS methodology leveraging the use of pausable clocks [68]. IHP labs designed a 802.11a GALS baseband processor including various IP cores, a viterbi decoder, FFT, IFFT and a Cordic processor. Several asynchronous NoCs have been also proposed: Mango [89], QNoC, ANoC, CHAIN [130], FAUST [95], Alpin, Magali. A chip dedicated to flexible baseband processing for 3G/4G wireless telecommunication applications and making use of an asynchronous NoC is described in [83, 84]. However,

currently the intricacy of asynchronous design and its poor CAD tool support makes the design of hard macros with ad hoc techniques [77] the only viable solution for industrial exploitation [85]. This comes at the cost of large area and penalizes flexibility.

The practical viability of synchronizer-based GALS networks has been demonstrated in [95], where the hierarchical clock tree synthesis technique for such systems is detailed. A circuit switched source synchronous GALS link is described in [82], making use of long distance interconnect paths. It is then experimented on a 65nm reconfigurable NoC for an heterogeneous GALS many-core platform. However, there is no comparison whatsoever with alternative clocking styles and/or implementations. In [129] a many-core heterogeneous computational platform employing GALS compatible circuit switching on-chip network has been presented. [115] presents an example of mesh-connected GALS chip multiprocessor. The work shows that the typical performance penalties of GALS systems (mainly due to additional communication latency) can be hidden by using large FIFO buffers. In [95], the physical implementation of the DSPIN network-on-chip in the FAUST architecture has been presented. In [116] a cost effective solution for asynchronous delay-insensitive on-chip communication is proposed. The solution is based on the Berger coding scheme and allows to obtain a very low wire overhead. [119] proposes a new asynchronous NoC architecture aiming at low latency transfers. This architecture is implemented as a GALS system, where chip units are built as synchronous islands, connected together using a delay insensitive asynchronous Network-on-Chip topology.

2.1.1 The dual-clok FIFO Synchronization Interfaces

From the pure fully asynchronous world, there are several examples of FIFOs in literature [120, 121]. Since these designs do not utilize clocks, they are difficult to be adopted when two different clock domains have to be synchronized (e.g., two frequency islands of a GALS system). A reliable data transfer across unrelated asynchronous clock domains is accomplished by [122]. This work is demonstrated in both standard cell and full custom design used in a GALS array processor. In any case, the presented synchronizer does not account for timing implication when integrated in the NoC domain. The work in [123] illustrates a modular and easily configurable dual-clock FIFO for different NoC requirements (clocked or clock-less interfaces, synchronization latency for resolving metastability, FIFO capacity). This design was inspired by the modular FIFO from [124], composed of a ring of stages, where each

stage is composed of a storage cell, a *put* interface and a *get* interface. The solutions in [123] and [124] differ as [123] uses cells that are available from a typical standard cell library, whereas [124] requires custom, pre-charged cells in the control blocks. Nevertheless, [123] achieves performance which is comparable or even slightly higher than that reported in [124] when scaling for the different fabrication technologies. Furthermore, [123] separates the FIFO control logic from the synchronizers, allowing the synchronization latency to be chosen according to the clock frequency and reliability requirements. [136] proposed a synchronizing FIFO design based on *read* and *write* pointer comparison using Gray codes. The result of the comparison is synchronized to the sender's and receiver's clocks. Their design uses two binary counters for read and write addresses and a dual-port RAM for data storage which results in a larger and more complicated design for the modest FIFO capacities that are typically needed for NoC design. Furthermore, the design in [136] only provides FIFO capacities that are powers of two. The design in [137] uses the position of the *read* and *write* pointers to determine fullness and emptiness condition. It employees one synchronizer to detect fullness, and a synchronizer per-stage to detect emptiness. To prevent errors due to metastability when sampling the pointers, they use two tokens each as the read pointer and write pointer. This makes their empty detector a little bit more complicated, as it requires comparing two synchronized *write* pointer bits with two *read* pointer bits for each stage. In addition, to differentiate between a full and an empty FIFO, they only fill the FIFO up to the second-to-last position, which means that a n stage FIFO can only hold $n - 1$ items.

The operating principle of the dual-clock FIFO architecture proposed in this thesis resembles that of [123]. From an implementation viewpoint, we were inspired by [137] and [136]. From [137], we borrowed the token ring concept for implementing FSMs, since this is a simple yet robust solution with respect to Grey coding. From [136] we borrowed the idea of performing a comparison between read and write pointers asynchronously and then synchronizing the result in the target domains. With respect to previous work, this thesis presents not just a single dual-clock FIFO architecture for use in the general case, but specializes the architecture for the specific operating conditions.

In general, a tight integration of synchronization interfaces into NoC building blocks to cut down on latency, area and power is advocated. Finally this thesis addresses the co-design of synchronizers with NoC building blocks.

2.1.2 The Mesochronous Synchronization Interfaces

For the particular case where the clocks in two communicating domains have the same frequency but unknown phase offset, then a dual-clock FIFO would be too much of an overhead and custom tailored mesochronous synchronizers are normally more cost-effective. Several works leverage the timing determinism provided by GALS wrapper to facilitate debug and testing of GALS systems [133], [134]. A mesochronous link is integrated within a multi-processor tiled architecture based on a Network-on-Chip communication backbone on a CMOS 65nm technology in [80]. The work builds on a full-duplex link architecture illustrated in [79] and on integrated flow control [78]. The baseline mesochronous synchronizer is instead proposed in [96]. However, the synchronizer is still an external module to the NoC. The same drawback is exposed by a different synchronization architecture, detailed in [97].

Examples at industrial level are presented in [99], [98], [114], [96]. In [98] authors examine the use of GALS techniques to address on-chip communication between different synchronous modules on a bus. Issues related to validation, module interfaces and tool flows, while looking at advantages in power savings, timing closure and Time-to-Market/Time-to-Money (TTM) are explored. [114], [96] both suggest to implement the boundary interface with a source-synchronous design style and propose some form of ping-pong buffering to counter timing and metastability concerns.

A well established solution for mesochronous synchronization is illustrated in [128] consists of delay-line synchronizers, using a variable delay on the data lines. This delay is computed in such a way to avoid switching in the metastability window of the receiving registers. Variable delay lines make this solution expensive and not always available in standard cell libraries. This is the same problem of the works in [117, 118], which use voltage comparators.

Several periodic synchronizers are illustrated in [112], which avoid metastability by delaying either the data or the clock signal to sample data when the clock is stable. Configurable digital delay lines are again needed and experimented frequency is very low. The same authors in [132] illustrate many ways to "fool" a synchronizer thus showing several weaknesses of the proposed approaches. The works in [94, 100, 117] achieve mesochronous data synchronization by using Muller C-elements and digital delay lines that are typically designed with a full-custom approach. [100] presents a self-tested self-synchronization method for mesochronous communication. The scheme uses two clocks with a phase shift of 180° and a failure detector is used to select which one to use. In [94] a phase detector in place of a metastability detector is used in the same scheme.

Architectures based on FIFO synchronizers are proposed in [101, 123, 128]. FIFO size in [101] depends on the skew, hence is link-dependent or given in the worst-case. Implementation is also very expensive, as showed in [102]. More recently, an optimized bi-synchronous FIFO has been proposed in [137] featuring low-latency and small footprint. It can be adapted to the mesochronous needs while proving able to tolerate skew only up to 50% of the clock period. An early mesochronous scheme for the SoCBus NoC was proposed in [81], aiming at compact realization while still lacking of a validation on an NoC test case. A significant step forward comes from the OCN system [113], which uses a source synchronous scheme. A matched-delay architecture is used to compensate the strobe skew and enable high-speed mesochronous communication. A FIFO synchronizer is used at the receiver side.

Summing up, mesochronous synchronizers surveyed so far incur several disadvantages: high implementation overhead, use of non-trivial or full-custom components or low skew tolerance. Moreover, very few works are able to assess timing margins with layout awareness. In this thesis, the same approach to synchronization of [96, 109] is taken (source synchronous data transmission, safe storage of data at the receiver side, sampling in the receiver domain only when data is stable). However, the baseline architectures and circuits are improved by providing a more compact and equally robust solution. This is used just as the baseline architecture for the sake of comparison with the novel synchronization structure that it is proposed.

The tight synchronizer-NoC coupling design principle adopted for the dual-clock FIFOs is followed also for the mesochronous synchronizer to cut down on area, latency and power. Furthermore, a wide range of architecture alternatives and even port-level configuration capability is provided. Moreover, we identify the distinctive design constraints of the new schemes and perform a design space exploration of mesochronous NoC links and switches to capture how timing margins can be preserved for different combinations of synthesis and layout constraints.

2.2 Reliability

As faults will appear with increasing probability due to the susceptibility of shrinking feature sizes to process variability, age-related degradation, crosstalk, and single-event upsets, designing efficient reliable NoCs becomes a key requirement. However, errors can differently affect the system: they can produce permanent, intermittent or transient failures depending on their

nature. As a consequence, protection mechanisms may substantially vary as a function of the target fault to tackle. In particular, strategies envisioned ad-hoc for permanent failures usually do not reveal intermittent or transient failures and vice versa mechanisms for these latter failures do not efficiently tackle permanent ones. As a result, multiple strategies must be co-integrated and co-optimized in order to efficiently protect the system from a wide range of errors. Following such considerations, this thesis exploits off-line built-in self-testing strategies to reveal permanent failures while it relies on on-line fault-tolerant mechanisms to detect transient and intermittent ones.

2.2.1 Built-In Self-Testing and Diagnosis

Considering the regular and modular structure of on-chip networks, test strategies previously proposed for systems with identical cores [104, 105] can be applied to the NoC.

However, both approaches incur a significant overhead for DfT structures (full-scan and IEEE 1500 wrapped cores with registered I/O pins).

The work in [106] proposes an implementation of the IEEE 1149.1 boundary scan standard as a strategy to carry out hierarchical test, and enable diagnostics, of a 2D grid router. In this case, at-speed testing is not feasible, and the serial shift through all registers during test costs a lot of additional logic. In [108] a wrapper with scan-chains attached to each router is proposed. One of the routers is defined as a test access switch to receive test patterns from the external test source and broadcast these patterns to other routers. In [107] the NoC is progressively used for testing its own components in a recursive manner. Again, scan insertion is needed.

[47] illustrates a wide range of standard DfT techniques for NoC design, from BIST for FIFOs, to functional testing of wrapped switches. This approach has a high area overhead due to full scan and BIST.

[45] also proposes a partial scan technique in combination with an IEEE 1500-compliant test wrapper. Area overhead is greatly reduced, but test application times amount to tens of thousands of clock cycles and test pattern generation time does not scale.

As opposed to using scan paths and wrappers for test access, [18] considers the case where test patterns are applied at the border I/Os of the network. The method was then extended in [19] to support fault diagnosis, while the DfT infrastructure was developed in [17]. While very high fault coverage was achieved, the time complexity of the test configurations is square with respect

to the rank of the NoC matrix. Moreover, in order to apply test patterns from network boundaries at-speed, a large number of test pins are necessary.

In [20], it is proposed to add dedicated logic to enable analysis of response from each FIFO in the switch, however no test data is presented. In [22] the possibility to repair the NoC during testing is envisioned, however error information is computed once for all and thus cannot handle situations where the chip slowly degrades.

[21] proposes a built-in self-test and self-diagnosis architecture to detect and locate faults in the FIFOs and in the MUXes of the switches. Unfortunately, the control path is left out of the framework.

In [26] an automatic go/no-go BIST operation is proposed at start up of a 2D mesh NoC. Low fault coverage is achieved for the switch controller, moreover the methodology applies only to a 2D mesh. That idea is evolved in [42], where a fault coverage close to 100% is documented with a few thousand clock cycles. However, the area cost of the BIST architecture is the main concern of this work.

The pattern based testing section from the more general reliability framework presented in [23] reports a testing methodology relying on random test pattern generation and signature analysis. Unfortunately, testing takes as large as 200000 cycles with 10000 patterns per test.

The thesis presents four scalable built-in self-test and self-diagnosis infrastructures for NoCs providing a wide exploration of different testing strategies in a homogeneous experimental setting. With respect to previous work, we claim a more efficient use of NoC structural redundancy for testing and diagnosis purposes through the use of a cooperative testing framework. With respect to conventional scan-based approaches, we reduce area overhead while at the same time detecting TPG faults. With respect to state-of-art functional testing solutions, we provide efficient testing of the control path as well and provide better test time scalability. With respect to pseudo-random testing, we cut down on the test application time.

2.2.2 Fault-tolerance

Transient faults cannot be handled by off-line strategies as they appear and disappear unpredictably. Fault-tolerant systems must be therefore employed to satisfy the high reliability constraints imposed by modern systems. In this direction, there are three major approaches.

First, Modular Redundancy can be adopted. For instance, Constantinides et al. demonstrated the BulletProof router, which efficiently uses NMR techniques for router level reliability [41]. However, NMR approaches are expensive, as they require at least N times the silicon area to implement. Similarly, Time Redundancy (TR) [32] can be adopted to protect the NoC against faulty components. However, Time Redundancy decreases the performance of the NoC, since all information needs to be retransmitted.

Alternately, error detecting codes can be used. The detection phase is followed by a recovery one, for instance based on the retry of the unsuccessful operation. Simple retransmission schemes are described in [30, 31]. In terms of implementation, [10, 30] use a single transmission buffer that contains both sent and unsent flits together. [31] uses link-level retransmission together with the Unique Token Protocol (UTP) to ensure reliability. However, it requires at least two copies of a packet in the network, increasing buffer occupancy and flow control complexity. In contrast, [29] minimizes control logic by using a barrel shifter as retransmission buffer whose size is matched to the round trip notification latency of a NACK. The work in [28] targets virtual channel NoC implementations and uses dynamic packet fragmentation in tandem with a credit-based fault-tolerant flow control to recover from corrupted virtual channel states. In general, state-of-the-art in fault-tolerant flow control can be reviewed in [27], where the power inefficient ACK/NACK or the high-impact T-Error protocols are compared. The key take-away is that more research is needed in this domain, a challenge that for instance [28] takes on. Unfortunately, the solution in [28] comes with heavy throughput limitations.

Finally, error correcting codes (ECC) can be employed. They typically allow the correction of a limited amount of errors per codeword in order to contain complexity. Nonetheless, they are commonly reported to introduce a high timing penalty, because of the delay of the encoder/decoder and correction blocks. In [33] the router selects on the fly the most effective ECC scheme to send the data trough the link. The work in [35] proposes to use the Hamming Code on the input buffers to protect FIFO data. Similarly [23] protects the data-path via an ECC strategy. However, by using the fault tolerance techniques proposed in [23, 33, 35], but also in [34] and [38], only the links are protected, and incur large area and performance overhead. A retransmission scheme that enables graceful degradation of NoC communication performance under high failure rates is proposed in [36], but again the control path is not protected. Interestingly, erroneous behaviour in the functionality of the routing process or in output port arbitration may cause flit/packet misrouting. In the worst case, this results into loss of information or even into a deadlock condition. Clearly,

robust protection against such upsets should be provided. Finally, [37] adopts error correcting coding to perform on-line testing but achieves a quite low coverage (63%). [29] proposes a mechanism able to exploit an ECC strategy for single error correction and a retransmission procedure once a double error is revealed. Anyway, the area overhead to support such mechanism is really severe, in addition to the high switching activity suffered from the retransmission buffer.

In general, the use of ECC in the above approaches suffers from two main limitations. First, the corrector is used at each clock cycle and ends up in the critical path. Second, the proposed architectures are not robust to many transient faults affecting the corrector itself. Moreover, there is consensus on the fact that error detection followed by retransmission typically has a milder impact on network power than error correction. This is the assumption of the work in [36] and the result of an ad-hoc experimental framework in [39] and [40]. However, none of these works relies on accurate microarchitectural studies and on physical implementation efforts. This thesis integrates the proposed design methods in a fault-tolerant switch based on a flow control protocol with error notification capability (NACK/GO) integrating both a detector and a corrector in each switch buffer [58]. Control logic of input and output buffers is triplicated and voted, and so are their state registers to avoid misalignment of FSMs. By exploiting the retransmission capability provided by NACK/GO, the control path of the switch could be protected with Dual Modular Redundancy (DMR) instead of TMR.

2.3 Network Reconfiguration

During the last two decades, a large number of proposals have been presented about resilient routing for both off-chip and on-chip networks. These approaches are either non-reconfigurable fault-tolerant routing strategies which tolerate a limited number of faults [49–52], or reconfigurable routing mechanisms that allow unlimited changes to the network. This thesis performs a step forward with respect to literature. We focus on schemes of the second category, in particular on those based on reconfiguration processes that consider such changes to the network structure to obtain new routing paths replacing the previous ones. Finally, we enhance the proposed scheme with fault-tolerance capabilities.

In off-chip networks, such as those used in clusters, during a reconfiguration process, the topology resulting from the connection/disconnection or failure of

network components is discovered by a central node, which runs the reconfiguration algorithm in software. The management software computes new routing tables and distributes them to each node. Detecting the new topology and communicating the new routing tables can be completed with or without traffic into the network. Static reconfiguration first stops and drains all user traffic from the network before completing the reconfiguration process [53, 54]. This reconfiguration method is unable to provide real-time and quality-of-service support needed by some applications. On the contrary, dynamic reconfiguration updates routing tables without stopping user traffic. In this case, the main challenge is to guarantee deadlock freedom as old and new routing functions are simultaneously active [48, 55–57, 59–62].

In the context of networks on chip, new techniques have been proposed and other retain some features of the above. The Vicis NoC architecture [23] uses the turn routing model during fault-free operation, and a heuristic solution that makes exceptions to that routing model to maximize connectivity. Reconfiguration process rewrites the routing tables based on the information from built-in-self-test units in each router. When large number of faults occur, exceptions sometimes result in deadlocked routing paths.

A reconfigurable fault-tolerant deflection routing algorithm based on reinforcement learning for NoC has been proposed in [63]. The algorithm reconfigures the routing tables through reinforcement learning based on 2-hop fault information. In [64], a reconfigurable routing algorithm for a 2D-mesh NoC is presented. This algorithm introduces low hardware cost but can only be used in one faulty router or regular region topology. Other proposals can deal with irregular fault regions. A mechanism to tolerate failures in networks for parallel computers is described in [65]. It tolerates any number of failures regardless of their spatial and temporal distributions. Immunet is limited by the network connectivity and results in high area overhead because it requires three routing tables per router. In [66], a region-based routing has been proposed to handle irregular networks. This algorithm groups destinations into regions to make routing decision. However, it does not provide a reconfiguration method to migrate from one configuration to another.

Finally, [67] presented Ariadne, an agnostic reconfiguration algorithm for NoCs, capable of circumventing large numbers of simultaneous faults, and able to handle unreliable future silicon technologies. Ariadne utilizes up*/down* for high performance and deadlock-free routing in irregular network topologies that result from large numbers of faults.

Ariadne is implemented in a fully distributed mode. Thus it results in very sim-

ple hardware and low complexity although it comes with suboptimal solutions for lack of global view. The up*/down* routing will not perform optimally under certain configurations, specially in the absence of failures (in a 2D mesh). In addition, up*/down* routing is encoded in routing tables at switches. Unfortunately, the Ariadne latency badly scales with network size (the configuration latency increases with the square of the nodes number). This latter property has a severe impact on the network performance especially because Ariadne does not guarantee a transparent transition between configurations. The flits have to freeze in the network pipelines and the throughput drops to zero during reconfiguration. Even when the communication resumes, a high contention due to the fullness of injection queues strongly degraded the network performance for a long period.

As opposed to these solutions, the scheme proposed by this thesis (OSR-Lite) does not use routing tables at switches, allows coding any efficient routing algorithm (even DOR routing) and requires lightweight switch support to enable truly fast dynamic reconfiguration. Moreover its latency smoothly increases with network size, and the configuration transition is transparent, ultimately preserving the throughput of the system.

3

Relaxing the Synchronization Assumption in Networks-on-Chip

THIS chapter introduces the first challenge we aim to solve in this thesis: the synchronization design issue. In a first step, the motivation for the adoption of synchronization mechanisms in the Network-on-Chip environment will be discussed. Next, the target GALS platform of this thesis along with the architecture of the basic switch block required to build it will be presented. Then, the focus will be on the baseline mesochronous synchronizer and its improvements that led to a new integrated and flexible switch architecture. Last, a library of novel dual-clock FIFO synchronizers, where each architecture variant in the library has been designed to match well-defined operating conditions at the minimum implementation cost, will be presented.

3.1 Limitations of the Fully Synchronous Approach

Network-on-chip (NoC) communication architectures are being widely adopted in multi-core chip design to ensure scalability and facilitate a component-based approach to large-scale system integration. As technology advances into aggressive nanometer-level scaling, a number of design challenges emerge from technology constraints which require a continuous evolution of NoC implementation strategies at the circuit and architectural level.

Synchronization is today definitely among the most critical challenges in the design of a global on-chip communication infrastructure, as emerging variability, signal integrity, power dissipation limits are contributing to a severe breakdown of the global synchronicity assumption when a logical structure spans more than a couple of millimeters on die [14]. NoCs typically span the entire chip area and there is today little doubt on the fact that a high-performance and

cost-effective NoC can only be designed in 45nm and beyond under a relaxed synchronization assumption [86].

3.2 A Possible Solution: the GALS Design Style

One effective method to address the synchronization issue is through the use of globally asynchronous locally synchronous (GALS) architectures where the chip is partitioned into multiple independent frequency domains. Each domain is clocked synchronously while inter-domain communication is achieved through specific interconnect techniques and circuits. Due to its flexible portability and transparent features regardless of the differences among computational cores, GALS interconnect architecture becomes a top candidate for multi- and many-core chips that wish to get rid of complex global clock distribution networks.

In addition, GALS allows the possibility of fine-grained power reduction through frequency and voltage scaling [69]. Since each core or cluster of cores operates in its own frequency domain, it is possible to reduce the power dissipation, increase energy efficiency and compensate for some circuit variations on a fine-grained level.

Among the advantages of a GALS clocking style, it is worth mentioning [82]:

- GALS clocking design with a simple local ring oscillator for each core eliminates the need for complex and power hungry global clock trees.
- Unused cores can be effectively disconnected by power gating, thus reducing leakage.
- When workloads distributed to cores are not identical or feature different performance requirements, we can allocate different clock frequencies and supply voltages for these cores either statically or dynamically. This allows the total system to consume a lower power than if all active cores had been operated at a single frequency and supply voltage [73].
- We can reduce more power by architecture-driven methods such as parallelizing or pipelining a serial algorithm over multiple cores [74].
- We can also spread computationally intensive workloads around the chip to eliminate hot spots and balance temperature.

The methodology of inter-domain communication is a crucial design point for GALS architectures. One approach is the purely asynchronous clockless handshaking, that uses multiple phases (normally two or four phases) of exchanging control signals (request and ACK) for transferring data words across clock domains [70, 71]. Unfortunately, these asynchronous handshaking techniques are complex and use unconventional circuits (such as the Muller C-element [72]) typically unavailable in generic standard cell libraries. Besides that, because the arrival times of events are arbitrary without a reference timing signal, their activities are difficult to verify in traditional digital CAD design flows.

The so-called delay-insensitive interconnection method extends clockless handshaking techniques by using coding techniques such as dual-rail or 1-of-4 to avoid the requirement of delay matching between data bits and control signals [116]. These circuits also require specific cells that do not exist in common ASIC design libraries. Quinton et al. implemented a delay-insensitive asynchronous interconnect network using only digital standard cells; however, the final circuit has large area and energy costs [90].

Another asynchronous interconnect technique uses a *pausible* or *stretchable* clock where the rising edge of the receiving clock is paused following the requirements of the control signals from the sender. This stops the synchronizer at the receiver until the data signals stabilize before sampling [91, 92]. The receiving clock is artificial in the sense that its period can vary cycle by cycle, therefore, it is not suitable for processing elements with synchronous clocking that need a stable signal clock in a long enough time. Besides that, this technique is difficult to manage when applied to a multiport design due to the arbitrary and unpredictable arrival times of multiple input signals.

An alternative for transferring data across clock domains is the source-synchronous communication technique that was originally proposed for off-chip interconnects. In this approach, the source clock signal is sent along with the data to the destination. At the destination, the source clock is used to sample and write the input data into a FIFO queue while the destination clock is used to read the data from the queue for processing. This method achieves high efficiency by obtaining an ideal throughput of one data word per source clock cycle with a very simple design that is also similar to the synchronous design methodology; hence it is easily compatible with common standard cell design flows [93, 125–127]. Unfortunately, correct operation of source-synchronous links is very sensitive to routing delay mismatches between data and the strobe signals, and hence to delay variability. Therefore, it is very challenging in the context of nanoscale silicon technologies.

In general, each GALS paradigm has its own pros and cons. Fully asynchronous design techniques are a more disruptive yet appealing solution, although their widespread industrial adoption might be slowed down by the relevant verification and design automation concerns they raise. Moreover, they tend to be quite area greedy, at least when timing robustness is enforced. In this context, a full custom design style is still the safer strategy for their successful utilization, hence asynchronous NoC building blocks are often instantiated as hard macros.

For this reason, this chapter will not review fully asynchronous NoC architectures, but will rather focus on synchronizer-based GALS architectures, and on source synchronous communication in particular. Using synchronizers for GALS NoC design implies an incremental evolution of mainstream EDA design tools and paves the way for compatible architectures (with careful synchronizer engineering) with a standard cell design flow. This chapter is fully devoted to this kind of GALS architectures.

The remainder of this chapter is organized as follows: Section 3.3 will describe the target GALS architecture under analysis in this thesis. Section 3.4 will present the switch architecture of reference that has been evolved in a sort of *galsification* process till building a GALS NoC switch. In order to achieve this, Section 3.5 and Section 3.6 respectively present the architectures of the mesochronous and the dual-clock FIFO synchronizer and their optimizations that led to novel tightly integrated synchronizers. Section 3.5 and Section 3.6 also present theoretical analysis on the synchronization constraints, which highlights the distinctive features of each synchronization scheme studied in the chapter, and describe the experimental results for such architectures. Finally, Section 3.7 will compare the mesochronous and the dual-clock FIFO synchronizer under an area point of view and Section 3.8 summarizes the contribution of this chapter.

3.3 Target GALS Architecture

There exist several options when implementing a GALS architecture. From a pure implementation viewpoint, one method consists of asynchronous clock-less handshaking, which uses multiple phases of signal exchange to transfer data. Due to the round-trip signal exchange, the transfer latency between two consecutive data words is high. Besides that, the asynchronous clock-less circuits are difficult to verify in traditional CAD flows, and they also demand a comparatively large area and energy requirements [90, 125].

In the system we implemented in this thesis, the on-chip network is seen as an independent clock domain. Therefore, part of the circuitry is devoted to reliably and efficiently move data across asynchronous clock boundaries between NoC switches and connected network interfaces. These latter are assumed to be part of the clock domain of the IP core that they serve. Dual-clock FIFOs are an effective solution to provide asynchronous boundary communication, especially in throughput-critical interfaces. However, many designers are skeptical about their utilization due to the relevant latency, area and power overhead they incur. Beyond urging research activities aiming at the optimization of dual-clock FIFO architectures, this fact emphasizes the need for their conscious use in GALS systems.

Aware of this, we try to minimize their usage as much as possible by instantiating them only at IP core boundaries, after their respective network interfaces (see Figure 3.1). However, this choice moves many chip-wide timing concerns to the on-chip network. In fact, this latter ends up spanning the entire chip and might be difficult to clock due to the growing chip sizes, clock rates, wire delays and parameter variations. As previously anticipated, in 45nm and beyond, a global clock signal spanning the whole chip area will be difficult to control and even to realize with a controllable phase-offset.

In fact, we find that mesochronous synchronization can relieve the burden of chip-wide clock tree distribution while requiring simpler and more compact synchronization interfaces than dual-clock FIFOs. Hierarchical clock tree synthesis is an effective way of exploiting mesochronous links, as already experimented in [95]. During the first step, a clock tree is synthesized for each network switch with a tightly controlled skew (e.g.,5%). Next, each clock tree is characterized with its input delay, skew and input capacitance. This information is used by the clock tree synthesis (CTS) tool to infer a top clock tree balancing the leaves with a much looser skew constraint (e.g., 30/40%). The ultimate result is a global clock tree which consumes less power then the traditional one generated by enforcing chip-wide skew constraints. For future large multiprocessor systems-on-chip, the use of this methodology can be not just an issue of power efficiency but even of CTS feasibility.

However, power savings with this methodology should not be taken for granted, since it involves some overheads: the transmission of the clock signal across mesochronous links, the mesochronous synchronizers themselves (implementing power-hungry buffering resources) and the increased number of buffer slots needed at link end-nodes to cover the larger round-trip time (associated with the synchronization latency) for correct flow control management.

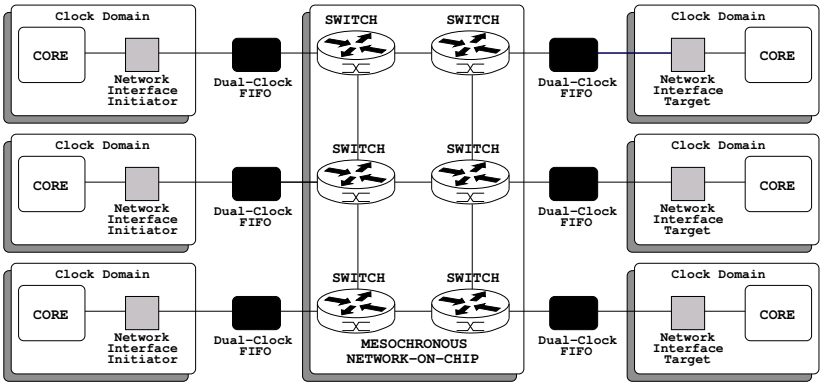


Figure 3.1: Target Design Platform.

Our design platform aims at minimizing such overheads through a novel mesochronous architecture taking advantage of the tight integration of the synchronizer into the switch architecture. However, since these solutions give rise to timing constraints that might not be verified for specific layout conditions, we provide architecture variants for these cases as well, thus coming up with a flexible switch suitable for many design instances.

Beyond mesochronous synchronizers, dual-clock FIFO architectures are still required at IP core boundaries.

Thus this thesis proposes a library of dual-clock FIFOs for cost-effective MP-SoC design, where each architecture variant in the library has been designed to match well-defined operating conditions at the minimum implementation cost. Each FIFO synchronizer is suitable for plug-and-play insertion into the NoC architecture and selection depends on the performance requirements of the synchronization interface at hand.

Finally we demonstrate that the design principle adopted for the mesochronous synchronizer, which we denote as “tight coupling” of synchronizer with NoC, can be applied also to dual-clock FIFOs. As a result, all components of our synchronization library have been conceived for tight integration into NoC building blocks.

In this direction, moving from a system-level to a switch block view, next section will present the fully synchronous baseline switch architecture that will be used as starting point for the implementation of our target GALS architecture. In fact, such an architecture will be augmented with various mesochronous and dual-clock FIFO synchronizer variants as we will see later on in this chapter.

3.4 ×pipesLite switch architecture

The ×pipesLite network-on-chip architecture has been conceived for the resource-constrained Multi-processor system-on-chip (MPSoC) domain. Therefore, it features a high degree of parameterization and compact implementation [110]. It is unpipelined, fully synthesizable with a standard design flow and achieves frequencies around 1.5GHz for the fastest configurations.

The ×pipesLite switch is conceived as a soft macro from the ground up. The possibility of design-time tuning of parameters such as flit width, number of I/O ports, buffer size and flow control policy makes it suitable to explore several topologies and architectural variants.

The baseline architecture implements an in/out buffered switch implementing wormhole switching and source-based routing, as depicted in Figure 3.2.

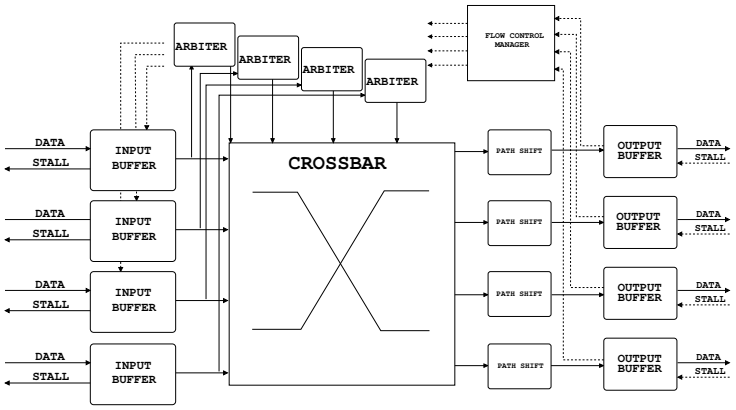


Figure 3.2: Baseline switch architecture.

The crossing latency is therefore equal to 1 cycle in the upstream/downstream link and 1 cycle inside the switch itself. The choice for retiming at input and output ports stems from the need to break the timing path across switch–to–switch links. The first chapters of this thesis show that in 65nm technology the delay of inter-switch links causes a significant performance drop for most regular NoC topologies depending on the intricacy of their connectivity pattern. This is certainly technology-, architecture- and topology-specific and depends on the specific link inference technique too, but a rule of thumb is that with target operating speeds above 700 MHz, even for 2D mesh topologies, the target speed is likely to be achieved by leveraging input and output retiming in

the switch. This is due to the increasing role of RC propagation delay across the interconnects and will hold even more in future technology nodes.

The ×pipesLite switch relies on a stall/go flow control protocol [27]. It requires two control wires: one going forward and flagging data availability ("valid") and one going backward and signaling either a condition of buffer filled ("stall") or of buffer free ("go"). With this scheme, power is minimized since any congestion issue simply results in no unneeded transitions over the data wires. Moreover, recovery from congestion is instantaneous. The input buffer serves as a retiming stage, while at the same time handling flow control. For this purpose, it needs to be 2 slots deep. During normal operation, only one slot is used. When a "stall" is notified by the downstream stage, output data of the buffer is frozen. However, it takes one cycle to propagate the "stall" upstream, during which a new input data is driven and needs to be stored in a backup slot. This justifies the need for 2 slots. The architecture can be seen as a synthesizable realization of the elastic buffer concept. The output buffer has a tunable size for performance optimization, and handles flow control as well. Therefore, it shares the underlying architecture design techniques with the input buffer. Arbitration is not centralized, i.e., there is a round-robin arbiter for each output port serializing conflicting access requests for that output port. The critical path of the switch starts from the finite state machine of the switch input buffer, goes through the arbiter, the crossbar selection signals, some header processing logic and finally includes a library setup time for correct sampling at the switch output port (see Figure 3.2). The combinational logic shifts the routing bits in the packet header in such a way that each switch reads the target output port for that packet in the first position.

In order to build our GALS platform, mesochronous and multi-synchronous communication needs to be implemented by means of synchronizers. We envision the design of such synchronizers as a seamless replacement of the switch input buffer of Figure 3.2 thus effectively coming up with a new GALS switch architecture, an example is depicted in Figure 3.3.

The architecture flexibility, provided by all the synchronizer interfaces variants, enables a fully configurable switch building block that can be tailored depending on different requirements. In fact, as it will be clear later on in this chapter, different synchronizer implementation choices lead to different tradeoff in terms of performance/link-delay toleration. We see this as a very efficient and cost-effective way to implement a GALS Network-on-Chip because only those switch input ports, that have specific synchronization requirements, can be equipped with a mesochronous or a dual-clock FIFO synchronizer whereas

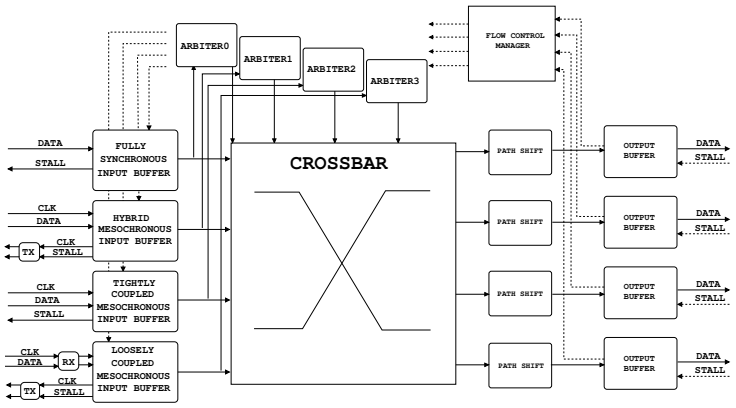


Figure 3.3: GALS switch architecture.

in the remaining ports, a fully synchronous input buffer can be instantiated.

All the developed mesochronous and dual-clock FIFO synchronizer architecture variants will be detailed in the remainder of this chapter starting from the baseline mesochronous synchronization architecture described in the next section.

3.5 The Mesochronous Interface

This work moves from the mesochronous synchronizer architecture presented in [109] and illustrated in Figure 3.4. The circuit receives as its inputs a bundle of NoC wires representing a regular NoC link, carrying data and/or flow control commands, and a copy of the clock signal of the sender. Since the latter wire experiences the same propagation delay as the data and flow control wires, it can be used as a strobe signal for them. The circuit is composed by a front-end and a back-end. The front-end is driven by the incoming clock signal, and strobes the incoming data and flow control wires onto a set of parallel latches in a rotating fashion, based on a counter. The back-end of the circuit leverages the local clock, and samples data from one of the latches in the front-end thanks to multiplexing logic which is also based on a counter. The rationale is to temporarily store incoming information in one of the front-end latches, using the incoming clock wire to avoid any timing problem related to the clock phase offset. Once the information stored in the latch is stable, it can be read by the target clock domain and sampled by a regular flip-flop.

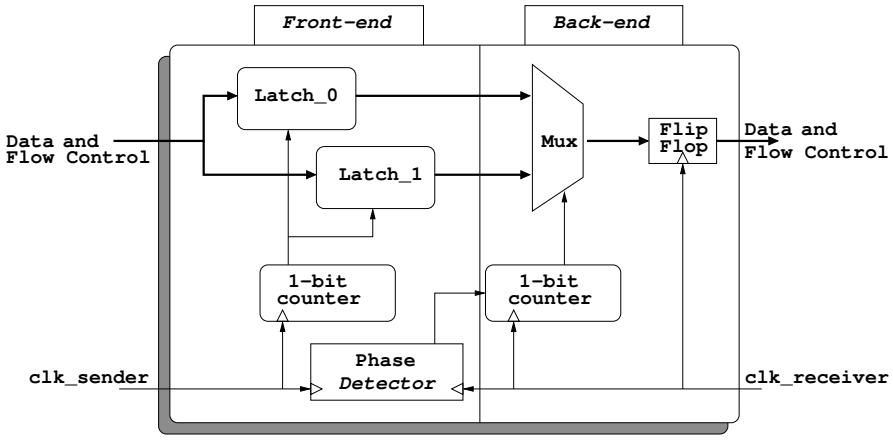


Figure 3.4: Baseline mesochronous synchronizer architecture of [109].

Counters in the front-end and back-end are initialized upon reset, after observing the actual clock skew among the sender and receiver with a phase detector, so as to establish a proper offset. The phase detector only operates upon the system reset, but given the mesochronous nature of the link, its findings hold equally well during normal operation.

Since few flow control wires are traveling backwards, another similar but much smaller synchronizer needs to be instantiated at the sender to handle them.

3.5.1 The *Loosely* Coupled Mesochronous Synchronizer

In agreement with [109], it is always possible to choose a counter setup so that the sampling clock edge in the back-end captures the output of the latches in a stable condition, even accounting for timing margin to neutralize jitter. Therefore, no more than 2 latches in parallel are needed in the front-end for short-range (i.e., single cycle) mesochronous communication with this scheme.

However, other considerations suggest that a different choice may be desirable at this point. In particular, by increasing the number of input latches by one more stage, it becomes possible to remove the phase detector (see the new architecture in Figure 3.5). This would be desirable due to the timing uncertainty or the high area footprint or the non-compliance to a standard cell flow that affects many phase detector implementations. A third latch bank allows to keep latched data stable for a longer time window and to even find a unique and safe bootstrap configuration (i.e., counters initialization) that turns out to be robust

in any phase skew scenario.

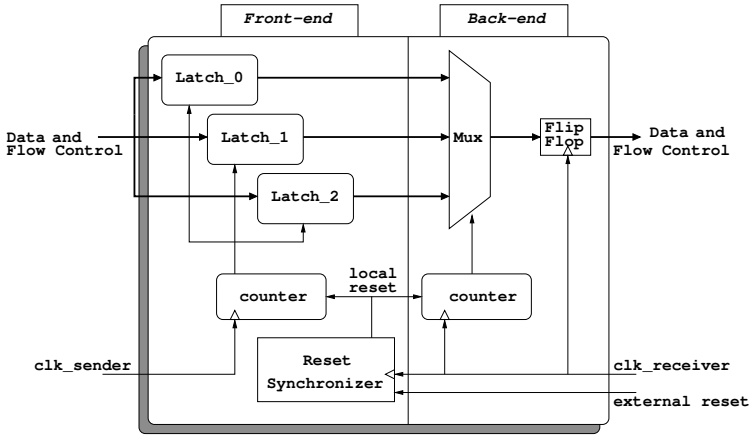


Figure 3.5: The loosely coupled mesochronous synchronizer of this work.

At regime, the output multiplexer always selects the output of the latch bank preceding the bank which is being enabled by the front-end counter. Rotating operation of both front- and back-end counters preserves this order. In contrast to [109], the reset architecture is designed, as Figure 3.5 shows. In most SoCs, the reset signal coming into the chip is an asynchronous input. Therefore, reset de-assertion should be synchronized in the receive clock domain. In fact, if a reset removal to a flip-flop occurs close to the active edge of its clock, flip-flops can enter a metastable state. We use a brute-force synchronizer (available in several new technology libraries as a standard cell, e.g. ST65nm) for reset synchronization with the receiver clock. Now, the challenge is how to reset the front-end. Typically, a reset can be sent by the upstream switch. In our architecture, we prevent metastability in the front-end by delaying the strobe generation in the upstream switch by one clock cycle after reset de-assertion. This way, on the first edge of the strobe signal, the receiver synchronizer is already reset. Such strobe signal generation delay is compliant with network packet injection delay after reset.

The transmitter clock signal is used as the strobe signal in our architecture. Differently than [109], a larger timing margin is enforced for safe input data sampling. In fact, the transmitter clock signal has to be processed at the receiver end in order to drive the latch enable signals. In actual layouts, this processing time adds up to the routing skew between data and strobe and to the delay for driving the latch enable high-fanout nets. As a result, the latch enable signal might be activated too late, and the input data signal might have already

changed. In order to make the synchronizer more robust to these events, we ensure that input data sampling occurs in the middle of the clock period. In fact, a switching latch enable signal opens the sampling window of the next latch during the rising edge, and closes the same during the falling one. As a result, the latch enable activation has a margin of half clock cycle to occur. Our post-layout simulations prove that this margin is largely met in practice. Finally, in agreement with [109], we computed the minimum size of the input buffer in the downstream switch to be 4 slots (flits). They are required by the stall/go flow control protocol in order to cover the round trip latency and not to drop flits in flight when a stall signal has to be propagated backwards. The original input buffer size is 2 slots, reflecting the requirements of stall/go without synchronization. Please refer to [109] for further details about this.

3.5.2 Tightly Integrated Mesochronous Synchronizer Architecture

The previous synchronizer, similarly to SKIL or to the Polaris one, is a module of the NoC architecture, thus loosely coupled with the downstream switch. The loose coupling stems from the fact that the flip-flop in the synchronizer back-end belongs directly to the switch input buffer. The mux output is therefore sampled like any other input in the fully synchronous scenario. However, our early exploration indicated that the area overhead induced in this input buffer, as an effect of the added synchronization latency, is much larger than the synchronizer area itself.

This hints that a tighter integration of the synchronizer into the switch input buffer is desirable. In particular, the latch enable signals of the synchronizer front-end could be conditioned with backward-propagating flow control signals, thus exploiting input latches as useful buffer stages and not just as an overhead for synchronization. In this case, input data is at first stored in the latches and then synchronized. This allows to completely remove the switch input buffer and to replace it with the synchronizer itself. The synchronizer output is then directly fed to the switch arbitration logic and to the crossbar. The ultimate consequence is that the mesochronous synchronizer becomes the actual switch input stage, with its latching stages acting as both buffering and synchronization stages (see Figure 3.6). As a side benefit, the latency of the synchronization stage in front of the switch is removed, since now the synchronizer and the switch input buffer coincide. The main necessary change to make the new architecture come true is to bring flow control signals to the front-end and back-end counters of the synchronizer. This solution would still

require 4 slot buffers, i.e., 4 latching banks. However, a further optimization is feasible. The backward-propagating flow control signal (the stall/go signal) could be directly synchronized with the strobe signal in the synchronizer front-end before being propagated to the upstream switch. This would save also the synchronizer at the transmitter side. In fact, the backward-propagating signal would be already in synch with the strobe, which in turn is in synch with the transmitter clock. The ultimate result is the architecture illustrated in Figure 3.6.

We are aware that this latter choice shrinks timing margins for the backward flow control signal. The reason is that it leaves the downstream switch with a generation delay across its synchronizer and also experiences the link propagation delay. This margin will be assessed post-layout in the experimental section, proving the applicability of the scheme. For this architecture solution, only 3 latching banks are needed in the synchronizer. In practice, only 1 slot buffer more than the fully synchronous input buffer. The tightly coupled synchronizer makes the mesochronous NoC design fully modular like the synchronous one, since no external blocks to the switches have to be instantiated for switch-to-switch communication. Please notice that the reset architecture remains unchanged with respect to Figure 3.5.

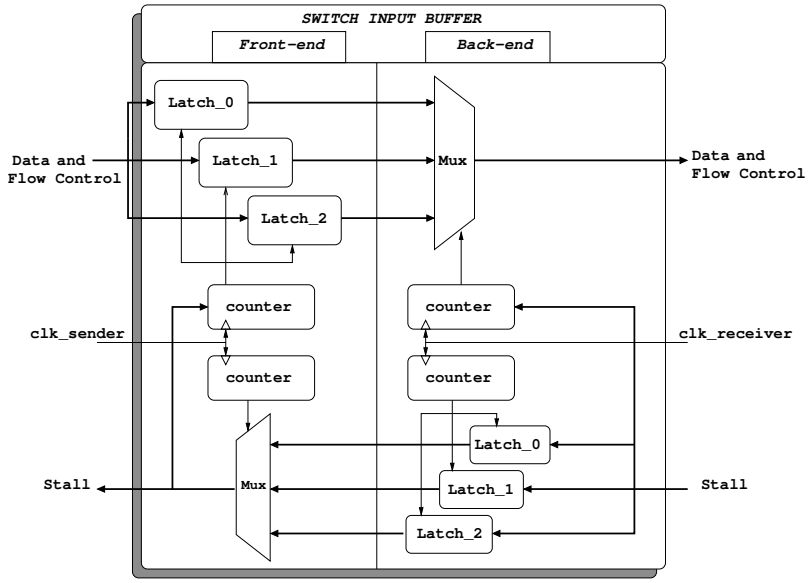


Figure 3.6: Proposed tightly coupled mesochronous synchronizer.

Operating principle

In case a *go* signal comes from the switch arbiter, at each clock cycle data are latched in the input buffers of the synchronizer, synchronized with the local clock and propagated to the switch arbiter and crossbar. When a *stall* occurs, the output mux keeps driving the same output until communication can be resumed. While the stall signal gets synchronized with the strobe and reaches the front-end, the front-end latches keep sampling input flits in a rotating way. When the stall signal finally leaves the synchronizer, it will stop the transmission of the upstream switch and the front-end counter operation at the same time. At this point, the situation is frozen. When then a *go* arrives, the output mux becomes operational again. Later, input latches and upstream switch resume their operation again at the same time. Please observe that this mechanism does not waste bandwidth on flow resumption, since the synchronizer backend can immediately start sweeping the output of front-end latches upon receipt of a *go*. Interestingly, flow control logic in the synchronizer is simplified with respect to that of the original switch input buffer. Before, a finite state machine used to generate a stall by monitoring the number of elements in the buffer. When it was equal to one and a stall came from the switch internal logic, than a stall was also generated for the upstream switch. In the new architecture, the synchronizer just synchronizes the stall signal from the switch logic with the transmitter clock and propagates it upstream. This way, a large amount of logic is saved.

Figure 3.7 reports the waveforms showing operation of the tightly coupled synchronizer. A delay from the strobe signal is assumed for the latch enable signals to account for their high-fanout.

3.5.3 Theoretical Analysis

In this section we analyze the previously presented synchronizer architectures (i.e., *loose* and *tight*) from a formal viewpoint.

The two synchronizer solutions presented so far have a series of common constraints regarding the correct sampling of data and flow control wires. In fact, the basic idea consists of sending *data* (or *stall* hereafter) and clock signal with a null phase offset, latching data at receiver end and performing the sampling only when data is stable. Nonetheless, as discussed in previous sections, there are some architectural peculiarities that differentiate the timing constraints of each solution with respect to another. Let us first analyze the common constraints considering Formula 3.1.

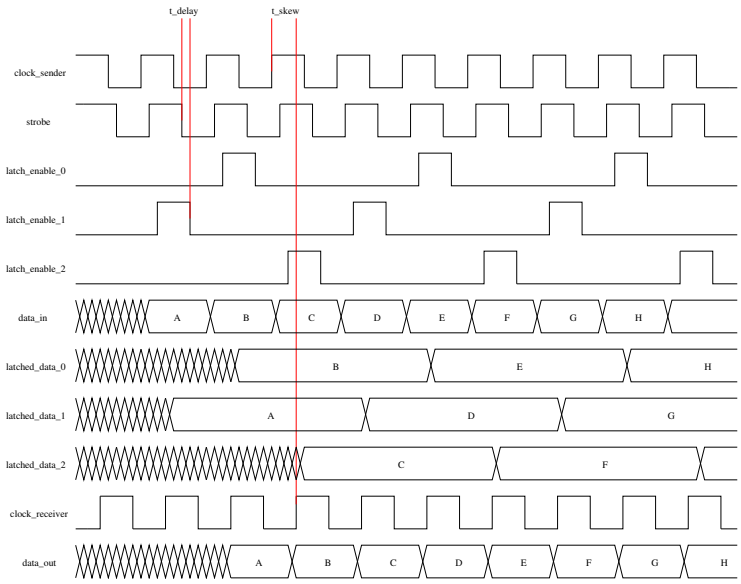


Figure 3.7: Waveforms example of the tightly coupled mesochronous synchronizer.

$$T_{setup} \leq \frac{T_{clock}}{2} + T_{latch\_enable} \tag{3.1}$$

For the sake of simplicity, we consider the case where clock and data signals are sent to the link channel without routing skew (perfect wires alignment). At receiver end, when a rising edge of the clock signal occurs, after a further T_{latch_enable} the latch window is transparent and data is captured. When a clock falling edge occurs (after half clock cycle) data is sampled correctly if and only if it was already stable for a T_{setup}. Of course, considering a non-zero routing skew between clock and data wires, a further $T\_routing\_skew$ has to be taken into account at left or right side of the equation depending on the relative position between clock and data signals.

Furthermore, for a correct sampling, data has to be kept stable at least for a T_{hold} before changing. Therefore, within a clock cycle, a $\frac{T_{clock}}{2}$ and a T_{latch_enable} are necessary to sample the data but a further T_{hold} is needed so that it can be considered stable (see Formula 3.2). As before, in a non-zero routing skew exists, it has to be taken into account as it additionally reduces the timing margin for a correct sampling.

$$\frac{T_{clock}}{2} + T_{latch\_enable} \leq T_{clock} - T_{hold} \tag{3.2}$$

So far, we have analyzed timing constraints common to all the proposed solutions. We will now consider, synchronizer by synchronizer, a further timing constraint that is specific for each architecture.

As our previous discussions already indicated, in the loosely coupled architecture, data sent from the upstream switch requires two clock cycles to be sampled by the downstream switch input buffer. Therefore, within two clock cycles, data has to traverse the link channel in T_{link}, it requires a T_{latch_enable} plus T_{mux} to correctly latch and output data from the external synchronizer towards the switch. A further T_{setup} is necessary to guarantee correct sampling by the downstream switch input buffer (see Formula 3.3). The entire data-path has to be traversed in two clock cycles, 1 cycle to synchronize data by the loosely coupled synchronizer plus a further clock cycle to sample data by switch input buffer. Obviously, a trade-off between clock frequency and link length is necessary for a correct operation of the entire system that would otherwise fail by violating the T_{setup} of the input buffer.

$$2 \cdot T_{Clock} \geq T_{skew} + T_{link} + T_{latch\_enable} + \\ + T_{mux} + T_{setup} \tag{3.3}$$

The same reasoning holds for the tightly coupled architecture with some differences though. In fact, sampling elements in the switch–to–switch data-path are the respective output buffers of the upstream and downstream switches. This path has to be traversed in two clock cycles: a first clock cycle is needed to send data from the upstream switch and to synchronize it by the multi-purpose switch input buffer. A second clock cycle is required to forward data from the switch input buffer to the output buffer of the same switch building block (see Formula 3.4).

$$2 \cdot T_{clock} \geq T_{skew} + T_{link} + T_{latch\_enable} + \\ + T_{mux} + T_{arbiter} + T_{crossbar} + \\ + T_{shift\_header} + T_{setup} \tag{3.4}$$

Differently from the loosely coupled architecture, Formula 3.4 points out the extra timing required to traverse the switch building block (e.g., arbitration time, crossbar traversal, etc.). In this case, a violation of the above would result in a sampling failure of the output buffer. A further timing constraint for the tightly coupled architecture is that illustrated by Formula 3.5.

$$T_{clock} \geq T_{generation} + 2 \cdot T_{link} + T_{counter} + \\ + T_{mux} + T_{setup} \tag{3.5}$$

We name this constraint as the *round-trip* dependency of the tightly coupled synchronizer. In fact, within a single cycle, the clock sent by the upstream switch triggers the bottom counter of the downstream switch synchronizer front-end (that is in charge of flow control synchronization, see Figure 3.6). Once a multiplexer output has been selected, the just synchronized stall signal can be forwarded to the upstream switch output buffer to stop data transmission. Obviously, for a correct sampling, the output buffer requires the stall signal to be stable for at least T_{setup}. The reason of this constraint is that in a single clock cycle, the stall signal has to be synchronized (at downstream switch end) and forwarded to the upstream switch to stop data transmission. The loosely coupled synchronizer is not affected by this constraint as the flow control synchronization is performed by a 1-bit synchronizer instantiated at upstream switch side. Intuitively, this constraint strongly limits the maximum operating frequency for a given link length of the tightly coupled architecture.

Architecture flexibility: the Hybrid solution

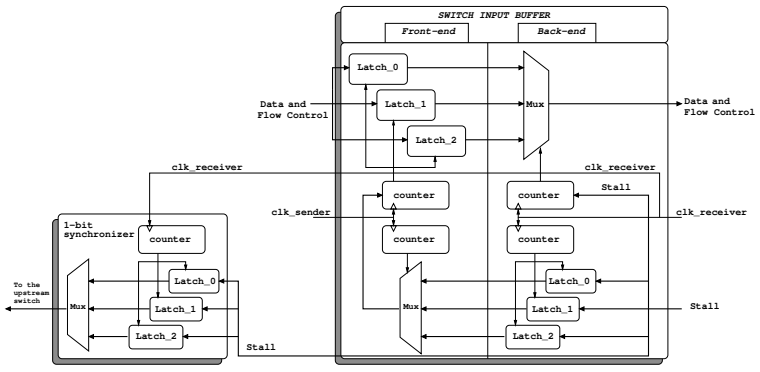


Figure 3.8: The hybrid architecture with a 1-bit mesochronous synchronizer on the receiver end.

In order to alleviate the limitation of the tightly coupled architecture, resembled by Formula 3.5, the hybrid architecture has been envisioned. In fact, the stall signal generated by the downstream switch arbiter is not synchronized in

the backend of the hybrid synchronizer but is directly forwarded to a 1-bit synchronizer instantiated in front of the upstream switch. This way, the *round-trip* dependency can be broken and Formula 3.5 does not hold for such an architecture anymore. Obviously, being the data-path of the hybrid architecture exactly the same as that of the tightly one, Formula 3.4 still remains valid but a new constraint for the stall signal path arises. The new constraint (see Formula 3.6) encompasses the time required to generate the stall signal by the arbiter (the higher the switch radix, the slower is the stall generation as the arbiter size increases); a T_{link} to traverse the link channel; the time necessary to the 1-bit synchronizer to latch and output the stall signal towards the upstream switch output buffer which requires a further T_{setup} to sample the stall signal.

$$\begin{aligned} 2 \cdot T_{clock} > T_{stall\_generation} + T_{skew} + T_{link} + \\ + T_{latch\_enable} + T_{mux} + T_{setup} \end{aligned} \tag{3.6}$$

Results in Section 3.5.5 will give an experimental evidence of the analysis presented above. While describing the specific constraints for each architecture, we omitted commenting T_{skew} parameter. It represents a misalignment between upstream and downstream clock (same clock frequency f, different phase offset). In all the presented architectures, T_{skew} is a decreasing factor for a correct operation as it reduces the timing margin of the analyzed system.

Another degree of flexibility of our architecture is that a switch can be assembled out of a mix of synchronous and mesochronous ports. In fact, the output architecture of the tightly coupled synchronizer resembles that of a synchronous switch input buffer, therefore for the switch data-path and control logic it is irrelevant whether the input port is synchronous or mesochronous. A flexible heterogeneous switch architecture can therefore be built, where input ports are either the conventional 2-slot buffer of synchronous switches or the tightly coupled synchronizer. Finally, an external mesochronous synchronizer can also be instantiated in front of the synchronous switch input ports to infer, whenever needed, the loosely coupled synchronization architecture.

Summing up, the three synchronizer architectures described so far enable the building process of GALS systems in a flexible and scalable way. In fact, depending on the link length (and an associated link delay), one of the different three solutions can be implemented in order to achieve a reliable timing margin for a correct operation of the system.

3.5.4 Experimental Results

This part of the chapter will characterize the mesochronous interfaces presented so far from latency, area footprint and power consumption viewpoint. In order to achieve such goal, both *loose*, *tight* and *hybrid* mesochronous interfaces have been implemented by means of the ×pipesLite NoC library [110]. Synthesisis and place&route have been performed through a backend synthesis flow leveraging industrial tools. The technology library utilized is a low-power low-Vth 65nm STMicroelectronics library (CMP project [135]).

Comparative latency Analysis

Since the tightly coupled mesochronous synchronizer not only changes the synchronizer implementation but also affects the entire network architecture, we performed basic tests to capture the macroscopic performance differences implied by the different synchronization architectures. We focus on synchronization latency, since the stall/go mechanism implemented in our synchronizer ensures that a stall-to-go transition of the flow control signal can be immediately propagated to the next stage. Hence, there are no wasted cycles at flow resumption, differently than [109]. Since what matters here is not a network-wide performance analysis, but just to investigate the latency of each scheme, this feature can be more conveniently stimulated and analyzed in a simple ad-hoc experimental test case for fine-grain performance analysis. We opted for a simple processor–NoC–memory topology (see Figure 3.9).

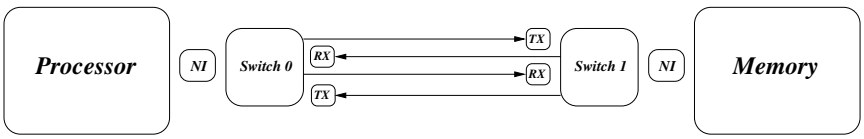


Figure 3.9: Test-case platform under analysis.

The investigated NoC is comprised of a couple of 2x2 switches respectively connected to the processor and the memory. Furthermore, each network switch is connected to its own RX-, TX-mesochronous part meant for synchronizing received data and flow control signals. For the sake of comparison, the SKIL synchronizer is considered as well. This is another loosely coupled module with the switch architecture [96].

The traffic pattern consists of full-bandwidth read and write transactions, i.e., the target memory never stops the access flow. Of course, the only perfor-

mance differentiation is seen for read transactions, since they are blocking for the processor core, hence they rely on the network ability to keep latency to a minimum. Performance results could be easily interpreted by means of a simple analytical model (Formula 3.7). It relates performance results to the intrinsic design characteristics of each synchronizer.

$$\texttt{cycles} = \texttt{n} + \textit{latency} \times 2 \times \texttt{\#transactions} \tag{3.7}$$

In fact in the best case, SKIL exposes two cycles synchronization overhead plus a further execution cycle for traversing the network switch; whereas our loosely coupled solution only requires one cycle latency in the mesochronous plus one cycle in the network switch. Even better, the tightly coupled mesochronous synchronizer requires the same computational resources of the vanilla switch (i.e., 1 execution cycle). The reason is that the tightly coupled solution seamlessly replaces the input buffer of the network switch thus providing a fast, reliable and robust mechanism for data synchronization. As for the tightly coupled, the hybrid solution as well only requires a single execution cycle in order to synchronize the data. The reason is that the synchronization circuit of the data path is the same as that of the tightly coupled interface (i.e., 1 cycle). Regarding the flow control, the stall signal in the hybrid architecture is directly forwarded from the arbiter to the TX- module instantiated in front of the upstream switch. This module is the one taking care of synchronizing the flow control signal in the same clock cycle when it has been forwarded.

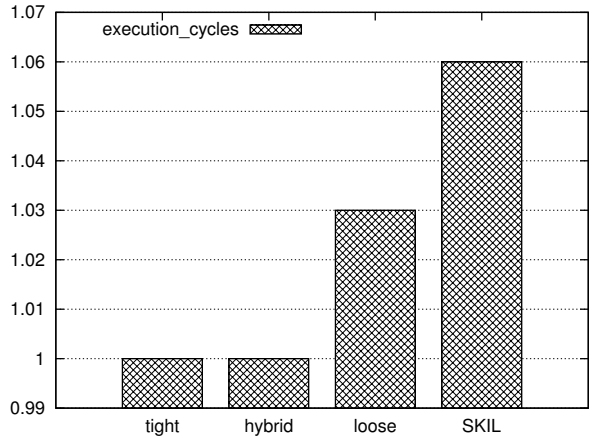


Figure 3.10: Normalized cycle latency of the different synchronization schemes.

Summarizing, whenever the system with the tightly (or hybrid) coupled

mesochronous synchronizer performs a computational task in n cycles, the alternative schemes, i.e., SKIL and the loosely coupled synchronizer respectively require a number of cycles equal to Formula 3.7, where *latency* is the number of clock cycles of the deployed mesochronous architecture whereas *#transaction* is the number of read operations performed by the processor unit. As depicted in Figure 3.10 there is a direct impact of the adopted synchronization solution on the overall system performance. While the tightly coupled and hybrid solutions keep the same performance as the vanilla network switch, a performance drop up ranging from 3% up to 6% incurs when using a loosely coupled or the SKIL scheme respectively.

Area Overhead

In order to estimate the area savings by using the tight integration design strategy, we went through a commercial synthesis flow and refined RTL description of the mesochronous switches (tightly, hybrid and loosely coupled) down to the physical layout. All the systems were synthesized, placed and routed at the same target frequency of 1GHz. In Figure 3.11, the area footprint of switches is reported along with a breakdown pointing out the contribution of synchronizers and/or input buffers.

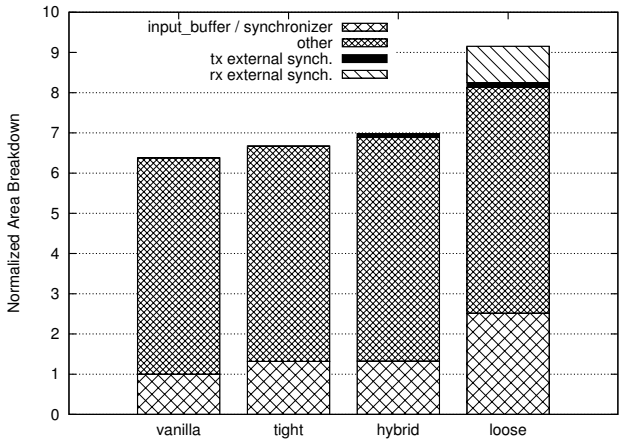


Figure 3.11: Area breakdown of a switch block with its synchronization scheme.

The tightly, hybrid and loosely coupled solutions are compared against a vanilla (i.e., fully synchronous) switch; *input_buffer* area for this switch only refers to the area occupancy of a normal 2 slot input buffer. For the loosely

coupled solution, a 4 slot buffer is needed to cover the round trip latency, and this is most of the overhead for this solution. As clearly pointed out by the area breakdown in Figure 3.11, the sum of the transmitter and of the receiver synchronizers is almost equal to that of a 2 slot buffer, i.e., of the input buffer in the vanilla switch. For the tightly coupled solution, *input_buffer/synchronizer* area refers to the multi-purpose switch input buffer (which is also the synchronizer). As for the latter, the hybrid architecture result refers to multi-purpose switch input buffers plus as many instances of TX- synchronizer as input ports. Clearly, there is almost no area overhead when moving from a fully synchronous to a tightly (or hybrid) integrated mesochronous switch as they employ similar buffering resources.

From the performance viewpoint, our post-layout synthesis results confirm that the critical path of the switch is not impacted by the replacement of the vanilla input buffer with the tightly integrated mesochronous synchronizer. By experimenting with different switch radix, the critical path deviates only marginally in the two cases, therefore no performance penalty should be expected for the mesochronous switch.

Power Consumption

A further step of our exploration was to contrast power consumption of the proposed mesochronous schemes. Our target design is a 5x5 switch in four different variants: the first is a fully synchronous switch block, the second has a tightly integrated mesochronous synchronizer per each input port; the third one is a switch utilizing a hybrid mesochronous synchronizer per input port thus requiring also a tx-synchronizer on the sender side; the last variant has a pair of loosely coupled rx- and tx-synchronizers per input port. Three different traffic patterns have been utilized to carry out an accurate power analysis: *idle*, request for a *random* output port and *parallel* communication flows. Post-layout simulation frequency was 700MHz for all the designs. As showed in Figure 3.12, in all the cases, the highest power consumption is consumed by the most buffer demanding solution, i.e., the loosely coupled design. Power consumption of the vanilla and tightly coupled designs are similar as expected; this is mainly due to the equivalent buffering resources deployed in both switches. The hybrid solution is a bit more expensive compared to the tight and the vanilla mainly because there is a small extra buffering due to the 1-bit synchronizer for each mesochronous port.

Next section will perform a design space exploration of the mesochronous link architecture in order to assess several quality metrics of the synchronizer

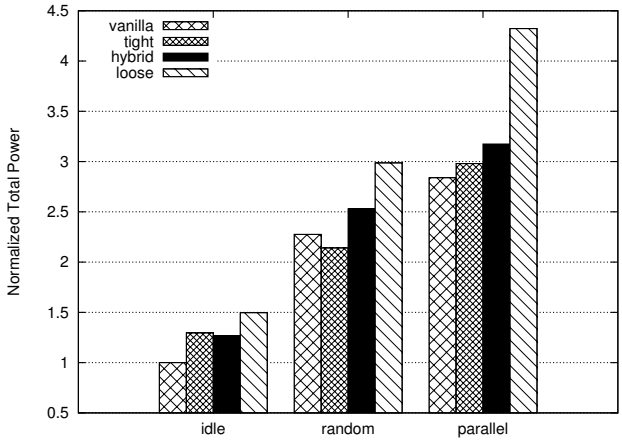


Figure 3.12: Normalized power consumption of different synchronization schemes in different traffic scenarios.

interface presented so far.

3.5.5 Mesochronous Link Design Characterization

The above architecture provides degrees of freedom for port-level selection of the most suitable mesochronous synchronization option based on timing and layout constraints. The following design space exploration of a mesochronous link implemented with our architecture will provide the guidelines for such port-level selection, and is therefore an essential enabler for automatic assembly of the target GALS NoC.

Design tradeoffs

A series of experiments have been carried out in order to characterize, for each synchronizer architecture, the maximum operating frequency for a given link length between two switch building blocks. In practice, a 5x5 switch, ideally extracted from the center of a 2D mesh, has been considered after place&route. The switch has been synthesized with a very tight timing constraints (1 GHz), so that after place&route the critical path for all the architectures will be in the switch–to–switch link. The switch is connected to a tester injecting clock and data with an increasing delay. The utilized testbench assumes an ideal alignment between clock signal and data as well as no routing skew. This

way, we can assess for a certain operating frequency, the relative link delay supported by each architecture. Obviously, a certain link delay corresponds to a relative channel length depending on how the link synthesis policy is chosen.

The theoretical analysis discussed in Section 3.5.3 is here confirmed by experimental results. In fact, Figure 3.13 reports maximum operating frequency of each synchronization scheme for a certain link delay. Obviously, being af-

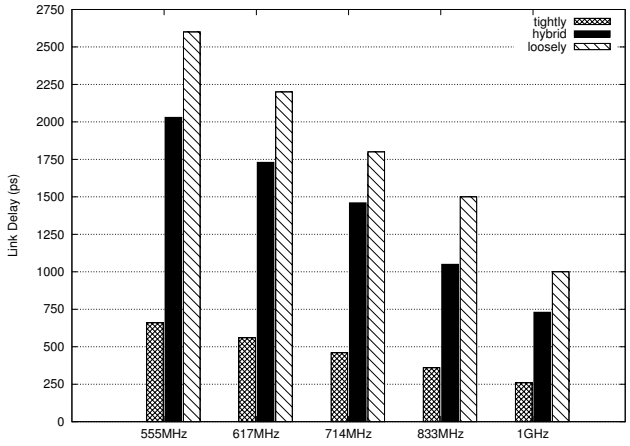
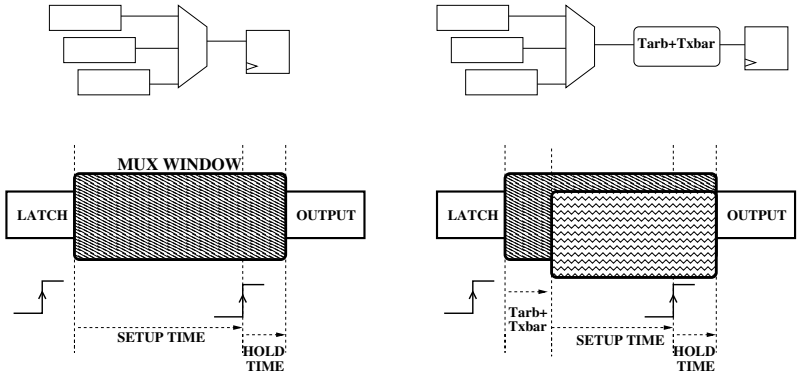


Figure 3.13: Operating frequency and tolerated link delay of different synchronizers.

fected by the round-trip constraint, the tightly coupled architecture turns out to be the less delay tolerant. Indeed, an increment of either frequency or delay would result in packets loss due to the late arrival of the backward propagating signal. However, such supported link delays enable the tightly coupled architecture to sustain a correct communication within a typical range of link length in nanoscale technologies. On the other hand, the hybrid alternative is quite effective as it relieves the round-trip constraint while keeping the area and power cost as low as the tightly coupled architecture. In fact, by using a small single-bit synchronizer at transmitter end for the backward signal, round trip dependency can be removed thus increasing maximum achievable frequency. Best results in terms of link delay toleration for a given frequency are achieved by the loosely coupled architecture. However, this result comes at a high area/power and also latency cost.

Skew Tolerance

Skew tolerance of our architecture schemes depends on the relative alignment of data arrival time at latch outputs, multiplexer selection window and sampling edge in the receiver clock domain. A few basic definitions help to assess the interaction among these parameters in determining skew tolerance. For the loosely coupled synchronizer, such definitions are pictorially illustrated in Figure 3.14(a).



(a) Loosely coupled synch. (b) Tightly coupled synch.

Figure 3.14: Basic mechanisms affecting skew tolerance.

During the *mux window*, data at latch outputs is selected for forwarding to the sampling flip-flop in the switch input port. Its duration closely follows that of the clock period. Sampling occurs *on the next rising edge of the receiver clock inside the mux window*. We denote the time between the starting point of the mux window and such sampling instant as the *Setup time*. Conversely, after an *Hold time* since the rising edge of the clock the mux window terminates. This is the time required by the counter to switch the multiplexer selection signals.

When we consider the tightly coupled architecture (Figure 3.14(b)), then the same metrics are taken at the switch output port rather than at the multiplexer output of the synchronizer. Therefore, the starting time of the *mux window* is delayed due to the worst case timing path between the synchronizer output and the switch output port, which includes the arbitration time, crossbar selection time and some more combinational logic delay for header processing. At the same time, the sampling rising edge of the receiver clock remains unaltered, therefore the ultimate effect is a shortening of the *Setup time* for the tightly integrated mesochronous switch architecture.

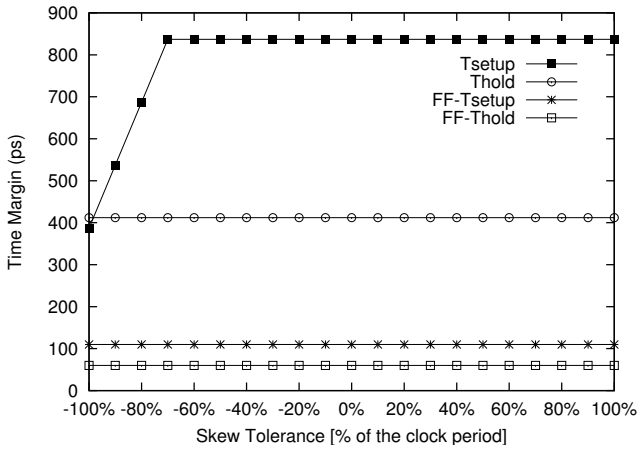


Figure 3.15: T_{setup} and T_{hold} for the loose coupled varying the skew tolerance.

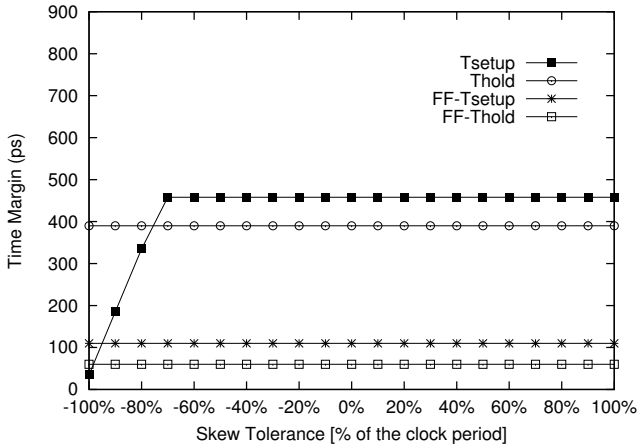


Figure 3.16: T_{setup} and T_{hold} for the tight coupled varying the skew tolerance.

Figure 3.15 quantifies these timing margins for the loosely coupled switch architecture. Results are referred to a 2x2 switch working at 660 MHz after place&route. X-axis reports negative and positive values of the skew, expressed as percentage of the clock period. Setup and hold times have been experimentally measured by driving the switch under test with a clocked testbench, by inducing phase offset with the switch clock and by monitoring waveforms at the switch. The connecting link between the testbench and the switch is assumed to have zero delay. A positive skew means that the clock at the switch is delayed with respect to the one at the testbench. The figure also

compares setup and hold times with the minimum values required by the technology library for correct sampling (denoted *FF-Tsetup* and *FF-Thold*).

First of all, we observe that both times are well above the library constraints, thus creating some margin against variability. For the whole range of the skew, the hold time stays the same. The result is relevant for positive skew, since its effect is to shift the *mux window* to the right, close to the region where latch output data switches. However, the stability window of the latch output data is long enough to always enable correct sampling of stable data before the point in time where it switches.

In contrast, a negative skew causes the *mux window* to shift to the left, therefore as the negative skew grows (in absolute values) the latch output data ends up switching inside the *mux window*, which corresponds to the knee of the setup time in Figure 3.15. From there on, the switching transient of data becomes closer to the sampling edge of the receiver clock and correct sampling can be guaranteed until the setup time curve equals the *FF-Tsetup* one. However, even for -100% skew synchronizer operation is correct.

Figure 3.16 illustrates the same results for the tightly coupled mesochronous switch. As anticipated above, the setup time is decreased by 370ps, corresponding to the time for arbitration, crossbar selection and shifting of routing bits. Interestingly, the knee of the setup time occurs for the same value of the negative skew, in that the switching instant of the latch output data enters the *mux window* at exactly the same point in time. The ultimate implication is that the tightly coupled synchronizer cannot work properly with -100% skew, since the crossing point with the *FF-Tsetup* occurs at around -95%. In practice, we can conclude that a 2x2 switch with tight coupling of the synchronizer on each port consumes 40% less area and power than its loosely coupled counterpart while incurring a 23% degradation of the maximum skew tolerance.

Target frequency

We now extend the above results to the case where the same RTL design (a 2x2 switch) is synthesized for a higher and lower target frequency and observe implications on the timing margins of a tightly coupled mesochronous NoC architecture. Figure 3.17 shows setup time as a function of negative skew for different target cycle times. The skew is expressed as percentage of the cycle time, and the assumption behind this plot is that as we relax the cycle time also the maximum skew constraint can be proportionally relaxed.

If we assume that by relaxing the target speed all delays scale proportionally

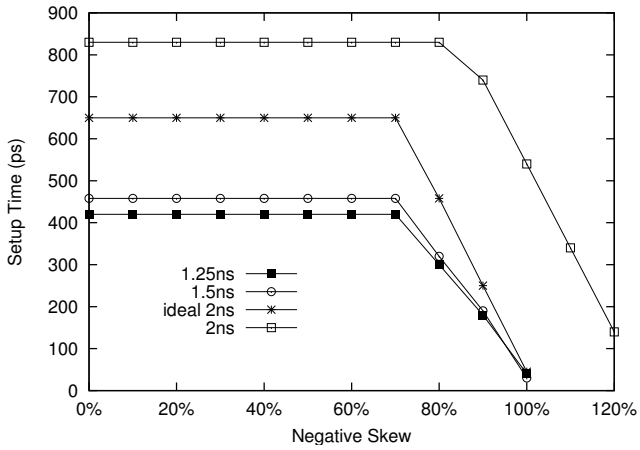


Figure 3.17: Setup time as a function of negative skew.

in the design, then we would expect that by relaxing the speed from 1.5ns to 2ns the setup time increases by 1.33x (see starred line). This is not the case, since the real setup time increases much more, as the figure shows. The reason for this is that the arbitration and switching logic inside the switch can be optimized for area as the timing constraint is relaxed only up to a certain point, beyond which no more netlist transformations are feasible. Therefore, the shifting of the *mux window* to the right, illustrated in Figure 3.14(b) for the tightly coupled architectures, scales ideally only up to a certain point, beyond which we observe a more-than-linear increase of the setup time. This is what happens for the 2ns target period.

Another deviation of the ideal curve from the real one regards the knees. In fact, although the target period increases from 1.25 to 2ns, a given skew percentage on the x-axis actually means a different absolute phase offset for the different cases. Therefore, the switching instant of synchronizer latch outputs should enter the *mux window* at the same percentage skew for all target periods (see starred line in Figure 3.17 for a 2ns target).

This is again not the case, indicating that a delay has not scaled proportionally to the clock period. The rationale for this is the time to generate the latch_enable signals in the synchronizer front-end. For a tight 1.25ns constraint, this netlist was already *non-critical*, therefore by relaxing the timing constraint its delay stays more or less the same. Therefore, the knee appears later for large cycle times, as the real curve for a 2ns target proves. The key

take-away from this characterization is that by relaxing the target clock frequency for the same RTL design an improvement of the skew tolerance and of the timing margins can be generally achieved for the tightly coupled architecture. In addition, a larger cycle time provides the physical synthesis tool more margin to enforce the feasibility constraint of Formula 3.4.

Switch radix

A last degree of freedom that we explored is the switch radix. We assume that for the same given target frequency, the switch radix is increased from 2 to 5. The effect on the timing margins is similar to what happens when we move from a loosely coupled to a tightly coupled mesochronous architecture. In fact, an increase of arbitration and crossbar selection time takes place, which results in a decrease of the setup time. Conversely, the knee occurs for the same amount of negative skew, since the modification concerns only the switch internal architecture, not the time at which latch outputs switch. Overall, by combining the two effects we have an additional reduction of the maximum (negative) skew tolerance, which is equal to the increase in delay of the combinational logic described above. In general, for high radix switches it has to be verified that the reduced setup time is still above the minimum value required by the technology library.

In contrast, synthesis constraints help relieve the above limitation. In fact, for both the 2x2 and the 5x5 switch, the synthesis tool tries to meet the same target cycle time and to exploit the available slack to save area. In practice, the syntheses of both the 2x2 and the 5x5 designs converge with almost no slack. Therefore, control logic with 5 and 2 inputs takes almost the same delay with a large difference in area. In this way, the setup time in the two cases is almost unaffected because of the netlist transformations performed by the synthesis tool in the area-performance plane. In our experiments, a 5x5 switch exhibits a setup time which is only 5% lower than the one in the 2x2 switch. For those NoC architectures where the above netlist optimizations are ineffective or for very high radix switches, it is necessary to verify that the reduced setup time is still above the minimum value required by the technology library.

Another implication of the switch radix concerns timing closure of the *hybrid* synchronization architecture. As already noted while commenting Figure 3.8, the critical path starts in the switch arbiter (which generates the *stall* signal) and includes the propagation of the *stall* signal to the upstream switch. As the switch radix increases, the delay for *stall* generation by the arbiter increases, and might make this timing path critical for the entire NoC.

3.6 The Dual-Clock FIFO Interface

Dual-clock FIFOs usually incur a significant area, power and latency overhead with respect to mesochronous synchronizer interfaces. However, while these latter simply guarantee skew compensation, dual-clock FIFOs also provide frequency decoupling. This thesis proposes a library of dual-clock FIFOs for cost-effective MPSoC design, where each architecture variant in the library has been designed to match well-defined operating conditions at the minimum implementation cost. Following the same principle adopted for the mesochronous synchronization, dual-clock FIFOs of our synchronization library have not been conceived in isolation, but have been tightly co-designed with the switching fabric of the on-chip interconnection network, thus making a conscious use of power-hungry buffering resources and leading to affordable implementations in the resource constrained MPSoC domain.

All the dual-clock FIFOs of our synchronization library are derived from a baseline dual-clock FIFO architecture, which is designed to be compatible with a standard cell design flow (i.e., without using custom cells).

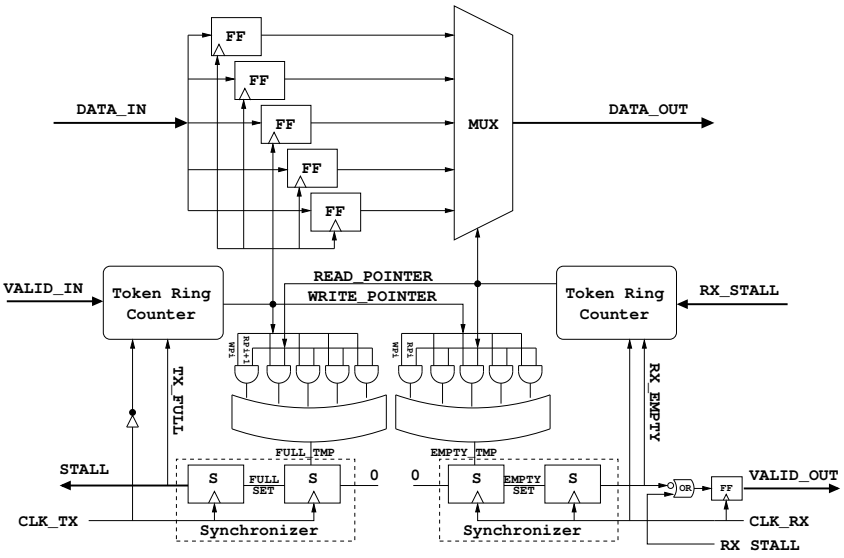


Figure 3.18: Dual-Clock FIFO Architecture.

The dual-clock FIFO (see Figure 3.18) is designed directly for use in a NoC setting. The xpipesLite NoC architecture already described in Section 3.4 is used as an experimentation platform. The NoC architecture determines the flow control protocol that the FIFO has to implement for correct interfacing

with the NoC. xpipesLite implements the stall/go flow control protocol requiring two control wires: one flags data availability while the other one propagates in the opposite direction and instructs the sender to stop or resume the flit transmission flow.

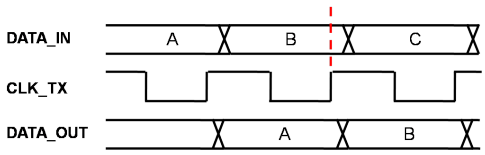
The circuit in Figure 3.18 receives as input a bundle of NoC wires representing a regular NoC link, carrying data and flow control commands, and a copy of the clock signal of the sender. Since the latter wire in principle experiences the same propagation delay as the data and flow control wires, it can be used as a strobe signal for them. The bi-synchronous FIFO has a sender and a receiver interface. Each interface has its own clock signal, *Clk_Tx* for the sender and *Clk_Rx* for the receiver.

Special care is devoted to enforcing timing margins for safe input data sampling. In fact, the transmitter clock signal has to be processed at the receiver in order to drive the write pointer. In actual layouts, this processing time adds up to the routing skew between data and strobe and to the delay for driving the clock input of the flip-flops. As a result, the strobe signal might be activated too late, and the input data signal might have already changed. In order to make the synchronizer more robust to these events, we make the strobe signal transition only on the falling edge of the transmitter clock. This way, each data is sampled approximately in the middle of the clock period (see Figure 3.19(a) and Figure 3.19(b) for a pictorial illustration).

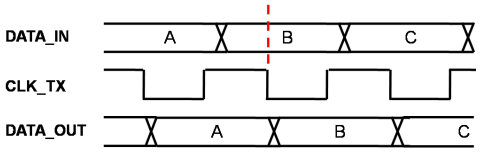
Queuing and de-queuing of data elements in the FIFO follow the protocol we describe hereafter. The *data_in* is queued into the FIFO, if and only if the *valid_in* signal is true and the *full* signal is false at the falling edge of *Clk_Tx*. Symmetrically, data is dequeued to *data_out*, if and only if the *RX_stall/go* signal is false (go) and the *empty* signal is false at the rising edge of *Clk_Rx*.

As shown in Figure 3.18, the bi-synchronous FIFO architecture is composed of 2 token ring counters. In the sender interface, the token ring counter is driven by the *Clk_Tx*, synchronous to incoming data. It generates the *write* pointer indicating the position to be written in the data buffer. In the receiver interface, the token ring counter is driven by the local clock, *Clk_Rx*. It generates the *read* pointer indicating the position to be read in the data buffer. Data buffer contains the data storage of the FIFO, which is parameterizable.

Full and *empty* detectors signal fullness and emptiness conditions of the FIFO. In our solution, these detectors perform an asynchronous comparisons between the FIFO *write* and *read* pointers that are generated in clock domains asynchronous to each other. The asynchronous FIFO pointer comparison technique uses few synchronization flip-flops to build the FIFO.



(a) Sampling on clock rising edge.



(b) Sampling on clock falling edge.

Figure 3.19: Sampling of input data.

The *full_detector* decides the value of the *full* signal depending on the content of *write* and *read* pointer. It requires N 2-input AND gates and one N-input OR gate, where N is the FIFO depth. The detector computes the logic AND operation between the *write* and *read* pointer and then collects the result along with an OR gate, thus obtaining logic value 1 if the FIFO is full, 0 otherwise. The FIFO is considered full when the *write* pointer points to the previous position of the *read* pointer. Vice versa, the FIFO is considered empty when the *write* pointer points to the same position of the *read* pointer.

Assertion of the *empty_tmp* signal is synchronous to the *Clk_Rx*-domain, since *empty_tmp* can only be asserted when the *read* pointer is incremented, but de-assertion of the *empty_tmp* signal happens when the *write* pointer is increased, which is an asynchronous event to *Clk_Rx*. On the contrary, assertion of the *full_tmp* signal is synchronous to the *Clk_Tx*-domain, since *full_tmp* can only be asserted when the *write* pointer is incremented, but de-assertion of *full_tmp* happens when the *read* pointer increments, which is an asynchronous event to *Clk_Tx*. As a consequence, the *full_tmp* and *empty_tmp* signals, coming out of an asynchronous comparison of read and write pointers, need to be synchronized by means of carefully engineered brute force synchronizers.

In particular, when the *read* pointer catches up with the *write* pointer, the *empty_tmp* signal presets the *rx_empty* flip-flops. When a FIFO write takes place, the *write* and *read* pointer contents are different, thus the *empty_tmp* can be de-asserted, releasing the preset control of the synchronizer. The *rx_empty* will be de-asserted after two rising edges of *Clk_Rx*. Since the de-assertion of *empty_tmp* occurs on a falling *Clk_Tx* edge while *rx_empty* is clocked by

Clk_Rx, the two-flop synchronizer is required to remove metastability associated with the asynchronous de-assertion of the preset control of the first flip-flop. Also the removal of the preset signal on the second flip-flop can violate the recovery time. However, in this case the second flip-flop will not go to a metastability state because the preset to the flip-flop has forced the output high so far and the input to the same flip-flop is also high, which is not subject to a recovery time instability [136].

The *tx_full* signal could be generated in an equivalent way with respect to *rx_empty* but an optimization is required to satisfy proper NoC constraints. In fact, the proposed FIFO is designed to support a STALL/GO flow control protocol. Since *tx_full* signal generated by the two-flop synchronizer coincides with the stall/go signal propagated upstream, it needs to be generated on the rising edge of *Clk_Tx*. This way, the time since the strobe edge occurs at the transmitter switch until the backward propagating flow control signal comes back should be one clock cycle. In order to meet this timing constraint, the two-flop synchronizer that generates *tx_full* samples on the rising edge of *Clk_Tx* and *full_tmp* does not preset the second *tx_full* flip-flop. In particular, when a FIFO-write operation causes a full condition on the falling edge of *Clk_Tx*, the *full_tmp* signal is consequently asserted and presets the first *tx_full* flip-flop. Therefore, assertion of the *tx_full* signal occurs on the next rising edge of *Clk_Tx*, and it can in turn be safely sampled by the counter on the next falling edge of the same clock. In practice, the token ring counter cannot progress any longer since the detection of the full condition. This mechanism relieves the round-trip dependency, since the stall/go signal leaves the FIFO as soon as the rising edge of *Clk_Tx* arrives and samples it, without waiting for the next falling edge.

Finally, when a FIFO-read operation takes place, the *read* pointer is incremented and the *full_tmp* signal is de-asserted, thus releasing the preset control of the first *tx_full* flip-flop. Therefore, due to the low logic value driving the input port of the first *tx_full* flip-flop, the FIFO will de-assert the *tx_full* signal after two rising edges of *Clk_Tx*. As the de-assertion of *full_tmp* occurs on a rising edge of *Clk_Rx* and because *tx_full* is clocked by the *Clk_Tx*, the two-flop synchronizer is required to remove metastability that could be generated by the first *tx_full* flip-flop.

The *tx_full* assertion process generates a critical timing path. The *full_tmp* critical timing path consists of (i) the $\texttt{tx\_clk} - \texttt{to} - \texttt{q}$ incrementing of the *write* pointer, (ii) comparison logic of the *read* pointer with the *write* pointer, (iii) presetting the first *tx_full* flip-flop, (iv) meeting the setup time of the second

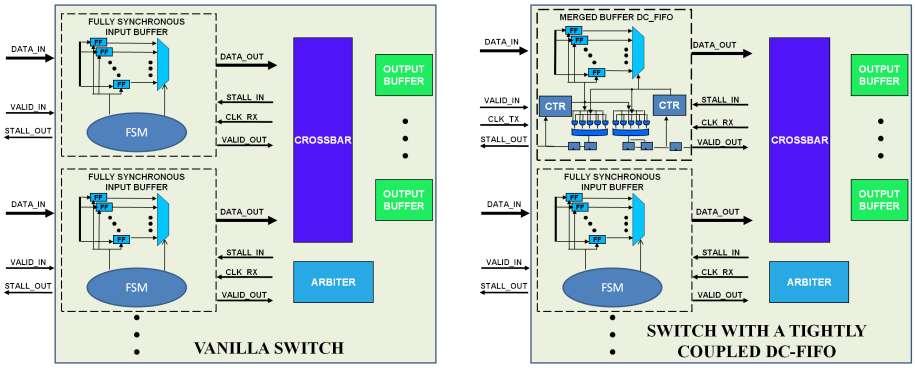


Figure 3.20: Vanilla switch and Dual-Clock FIFO integration into one input port of the NoC switch architecture.

tx_full flip-flop clocked with *Clk_Tx*. This critical path has to be covered in half clock cycle since *write* pointer is incremented on the falling edge while the *tx_full* flip-flop is sensitive to the rising edge. As regards the receiver domain, in the bi-synchronous FIFO in isolation, the empty assertion crosses a similar critical path but this time it can be properly covered in one clock cycle since the two-flop synchronizer is sensitive to the rising edge like the receiver token ring. Furthermore, the *rx_empty* assertion does not represent a critical path.

3.6.1 Tight Integration into the Switch Architecture

The dual-clock FIFO presented above can be used as a standalone component in a generic instance of the xpipesLite architecture, since it properly implements flow control. However, placing a FIFO in front of a NoC switch (like in Figure 3.1) implies a latency and buffering overhead for clock domain decoupling.

We found an alternative approach effective for mesochronous synchronizers: merging the synchronizer with the switch input buffer allows to share buffering resources for multiple tasks, namely switch buffering, flow control and synchronization, thus leading to a cost effective implementation of the synchronization interface. Unfortunately, this tight integration of architecture modules in some cases raises new timing constraints that have to be verified or even enforced with proper architecture-design techniques, as showed in Section 3.5.3.

In this section we extend the tight coupling design philosophy to dual-clock FIFOs, and prove that merging the architecture illustrated in Section 3.6 with the switch input buffer is straightforward (and far easier than for mesochronous

synchronizers) and does not fundamentally alter circuit timing.

As illustrated in Figure 3.20, the data buffer of the dual-clock FIFO is very similar to the architecture of the vanilla switch input stage (i.e. the baseline fully synchronous switch input buffer natively consisting of sampling elements, a mux and a FSM), therefore it does not bring major implications on the switch critical path. Moreover, the stall/go signal is provided by the arbiter, which receives the valid signal *valid_out* as an input.

The 2-slot input buffer of the vanilla switch has been replaced by a dual-clock FIFO that requires 5-slot buffers in the case where high performance needs to be guaranteed, thus leading to an area and power overhead. We will prove in Section 3.6.3 that in every frequency ratio scenario between sender and receiver, 100% throughput is guaranteed in the presence of a FIFO depth of at least 5 slots. Please notice that in any case the latency overhead of the synchronization interface is much reduced, since FIFO synchronizer and switch input buffer are not cascaded anymore.

The buffering overhead cannot be removed, but it is so marginal that in typical use cases it can be completely hidden. In fact, the xpipesLite architecture features input and output buffers, and the size of each of them is individually tunable. Generally, one buffer is kept to its minimum size (2 slots for correct support of flow control), while the other one is oversize to sustain performance and usually set to 6 slots based on past experience on system-level performance analysis [111]. In this scenario, the switch input buffer (merged with the FIFO synchronizer) can be used to implement performance-efficient buffering, while the output buffer can be retained just for retiming purposes. This way, the buffering overhead for synchronization is completely masked and there is fundamentally no area difference between a fully synchronous switch and a switch with synchronization-capable input buffers. Please observe that Figure 3.20 emphasizes the possibility to carry out a port-level configuration of the synchronization options to be supported by the switch. One or more input ports could decouple sender/receiver clock domains, while other ports might connect to switches/network interfaces in the same clock domain, thus giving rise to a large number of GALS NoC architecture variants.

Of note, the way the dual-clock FIFO architecture synchronizes the stall signal incurs a severe constraint on the round trip time. In fact, the transmitter clock used for stall synchronization at the downstream switch has already undergone a link delay T_{link}, and the stall signal itself takes another link delay to go back to the upstream switch:

$$T_{clock} \geq 2 \cdot T_{link} + T_{ff-propagation} + T_{setup} \tag{3.8}$$

Within a single cycle, the clock sent by the upstream switch triggers the flip-flop of the downstream brute-force synchronizer ($T_{ff-propagation}$) and the just synchronized stall signal is forwarded to the upstream switch output buffer to stop data transmission. Obviously, for a correct sampling, the output buffer requires the stall signal to be stable for at least T_{setup}. The reason of this constraint is that in a single clock cycle, the stall signal has to be synchronized (at downstream switch end) and forwarded to the upstream switch to stop data transmission. However, in such an architecture the most critical optimization is driven by the specific frequency ratio between two communicating domains rather than layout considerations (i.e., a large link delay may downgrade the final operaring speed of the whole design).

3.6.2 Latency analysis

Assertion and de-assertion latency of full/empty

Buffering requirements of the dual-clock FIFO can be limited when the number of clock cycles between the full/empty detection and the write/read suspension can be minimized. To meet this goal, the proposed architecture leverages optimized full/empty brute-force synchronizers having asynchronous preset control. Their runtime operation is detailed hereafter.

Figure 3.21(a) shows FIFO waveform when the transmitter frequency is higher than the receiver one. Since the receiver is slower than the sender, the dual-clock FIFO will generate periodically full assertion and full de-assertion. During T2 interval, value of the pointer is $WPi = RPi + 1$ and *full_tmp* is asserted, setting *full_set* on falling *tx_clk* edge. During T3, on the rising tx_{clk} edge, the second *tx_full* flip-flop will sample a logical high value and will drive *tx_full* signal high. During T3, on the falling *tx_clk* edge, the token ring counter will receive a high *tx_full* and will interrupt data write. Furthermore, the elapsed clock cycle between the detection of a full assertion and the write suspension is reduced to one clock cycle. During T3, *read_pointer* shifts and *full_tmp* is de-asserted on rising rx_{clk} edge. Consequently, during T4 *full_set* is de-asserted and during T5 also *tx_full* is de-asserted synchronously with tx_{clk}. Finally in T5, token ring counter in the sender domain restarts shifting regularly *write_pointer* on the falling tx_{clk} edge. Therefore, elapsed clock cycles

between *full_set* de-assertion and the *write_pointer* shift are reduced to one and a half clock cycles.

Let us now analyze a scenario when the transmitter frequency is lower than the receiver one. Since the *read_pointer* is faster than the *write_pointer*, the FIFO will generate periodically empty assertion and empty de-assertion. Symmetrically, the elapsed clock cycles between the detection of an empty assertion and the read suspension is only one clock cycle. Differently from the previous scenario, the elapsed clock cycles between *empty_set* de-assertion and the *read_pointer* shift is two clock cycles. This is due to the fact that the token ring counter in the receiver domain needs half clock cycle more to sample *rx_empty* due to the intrinsic 2-flop synchronizer latency. As the elapsed time between detection of empty/full and read/write suspension is one clock cycle, no quasi-full and quasi-empty detection need to be implemented and underflow/overflow is avoided by construction. On the contrary, slower empty/full de-assertion does not introduce errors in the FIFO activity but it can impact the FIFO throughput (see Section 3.6.3).

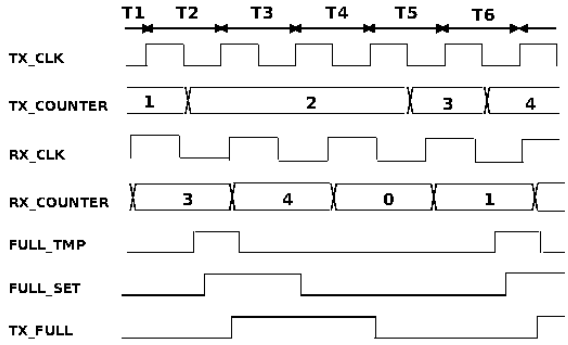
Switch crossing latency

As the sender and the receiver bridged by the dual-clock FIFO have different clocks, the crossing latency of the FIFO depends on frequency ratio and clock phase offset. Latency can be decomposed in two parameters: the first one is ΔT_{rx}, that is the time between the falling edge of the sender clock and the rising edge of the receiver clock. ΔT_{rx} can vary between 0 and 1 clock cycle depending on the offset between the clock signals. The second parameter is the number of clock cycles required by the read pointer to reach the location pointed by the writer. As reported in Table 3.1, three different scenarios have been analyzed in order to characterize the crossing latency of the switch with the integrated dual-clock FIFO interface.

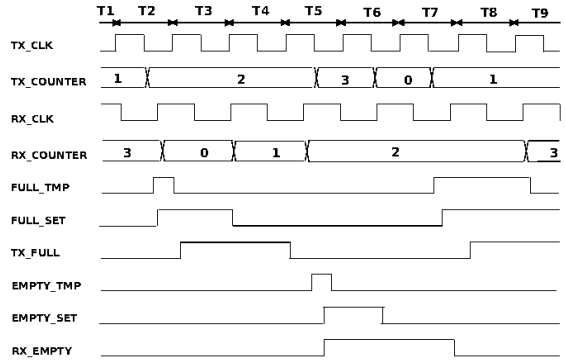
I°	minimum latency	$\Delta T_{rx} + 2Clock_{rx}$
II°	empty de-assertion	$\Delta T_{rx} + 3Clock_{rx}$
III°	maximum latency	$\Delta T_{rx} + Clock_{rx} \times (BufferDepth - 1)$

Table 3.1: Switch crossing latency.

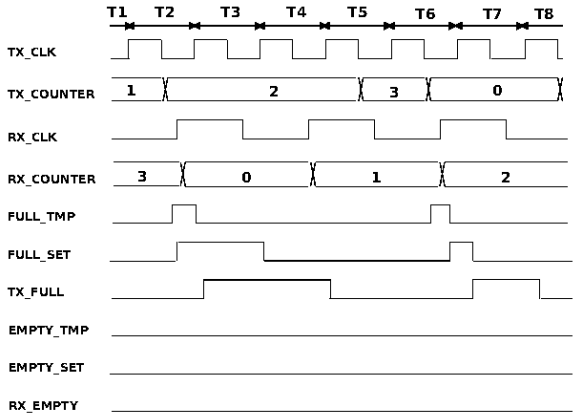
In the first scenario, when the *read* and *write* pointers are adjacent and the writer is preceding the reader, a minimum latency occurs. In this case, the *read* pointer opens the mux window after ΔT_{rx} for the data it is pointing to and the next data (which is the one being currently written) will be read after a further clock cycle. Therefore, the minimum latency to traverse the FIFO



(a) FIFO depth of 5, tx faster than rx.



(b) FIFO depth of 4, rx and tx freq. are similar.



(c) FIFO depth of 4, rx slower than tx.

Figure 3.21

synchronizer in this case is ΔT_{rx} + 1$Clock_{rx}$. In the second case, when the buffer is *empty* and a write operation occurs, a ΔT_{rx} + 1$Clock_{rx}$ is needed to clear the emptiness condition and a further clock cycle is required to enable the data at the multiplexer output. Finally, when the distance between the pointers is maximum (full condition), the required time for the reader to point the current write position is given by a ΔT_{rx} (offset between the clock signals) plus a contribution which depends on the number of buffer slots preceding the one currently pointed by the writer (which accounts to *BufferDepth-2*). Please note that in all latency results, 1 $Clock_{rx}$ cycle has been added to account for the time from the FIFO output to the input of the switch output buffer. In fact, Table 3.1 reports the overall switch crossing latency.

3.6.3 Throughput analysis

We consider our FIFO-based synchronizer to provide 100% throughput when the slowest end of the FIFO (either transmitter or receiver) can push/eject 1 data word per clock cycle.

In case of large FIFO depth, the de-assertion latency of the full signal (needed to notify the writer that further data can be stored) does not impact throughput, in that the reader has enough storage of past data words. This way, the reader avoids a blocking condition due to the delay needed to restart the writer. Similarly, the de-assertion latency of the empty signal does not lead to a blocking of the writer (which should wait till the reader resumes data consumption from the FIFO) due to the large availability of empty buffer slots in a deep FIFO.

By construction, we verified that the proposed dual-clock FIFO architecture guarantees 100% throughput when a FIFO depth of 5 slots is set, regardless of the frequency ratio between sender and receiver.

	FREQUENCY SCENARIO	FIFO DEPTH OF 5	FIFO DEPTH OF 4	FIFO DEPTH OF 3
I°	$3 \times f_{tx} > f_{rx}$	100%	100%	100%
II°	$1.5 \times f_{tx} > f_{rx}$	100%	100%	50% − 100%
III°	$f_{tx} \geqslant f_{rx}$	100%	50% − 100%	50% − 100%
IV°	$f_{rx} > f_{tx}$	100%	50% − 100%	50% − 100%
V°	$1.5 \times f_{rx} > f_{tx}$	100%	100%	50% − 100%
VI°	$3 \times f_{rx} > f_{tx}$	100%	100%	100%

Table 3.2: Dual-Clock FIFO throughput with parameterized buffer depth as a function of sender-receiver frequency ratio.

However, an interesting trade-off exists between throughput and FIFO depth (and consequently area and power footprint). Therefore, we now investigate

what happens when the FIFO depth is configured to be lower than 5 to save area. Figure 3.21(b) shows the system behavior when considering a FIFO depth of 4 and a *tx_clk* frequency similar (but higher) to the *rx_clk* frequency. During T2, value of the pointer is `WPi = RPi + 1` and *full_tmp* is asserted, thus setting *full_set*. During T3, the token ring will receive a high *tx_full* and will interrupt the data write. Consequently to *read_pointer* shifting, *full_set* and *tx_full* are de-asserted respectively during T4 and T5. During T5, token ring in the sender's domain restarts to shift regularly the *write_pointer*. Since we have a FIFO depth of 4, during T5 the value of the pointer is `WPi = RPi` and *empty_tmp* is asserted. Although transmitter frequency is higher than the receiver one, the FIFO generates an *empty_tmp* inducing a logic high *rx_empty* and a suspension of the regular read operation. This effect is clearly undesired and causes a reduction of throughput. In this condition, the FIFO is not able to deliver one word per cycle and the throughput is around 50%.

On the contrary, the FIFO will produce different results when considering a lower *rx_clk* frequency. Assuming the frequency ratio scenario showed in Figure 3.21(c) and a FIFO depth of 4, this time we obtain a throughput of 100% In fact, during T2, the value of the pointer is `WPi = RPi + 1` and *full_tmp* is asserted. In any case, during T5, when the token ring in the sender's domain restarts to shift regularly the *write_pointer*, the value of the pointer is not `WPi = RPi` and *empty_tmp* is not asserted.

Therefore, the FIFO throughput directly depends on the relative frequency between domains. Notice that examples could be envisioned by considering *rx_clock* frequency higher than the *tx_clock* one, thus obtaining symmetrical throughput results. Summing up, a FIFO with depth of 4 has 100% throughput if the sender clock cycle time (T_{tx}) and receiver clock cycle time (T_{rx}) meet one of these requirements:

$$3 \times T_{tx} < 2 \times T_{rx} \tag{3.9}$$

$$2 \times T_{tx} > 3 \times T_{rx} \tag{3.10}$$

As a result, the proposed FIFO with depth 4 guarantees 100% throughput when the transmitter module works with a frequency 33% lower or 50% higher than the receiver module. In presence of different frequency ratios, throughput is between 50% and 100%.

For the sake of further area and power optimizations, we analyzed the proposed FIFO with a depth of 3 slots. Following previous deductions, this solution

guarantees a throughput of 100% in case the transmitter is three times faster or slower than the receiver. In the remaining cases, the throughput is between 50% and 100%. Table 3.2 sums up the throughput results as a function of FIFO depth and frequency ratio scenarios.

3.6.4 Specialized library components

The dual-clock FIFO architecture can be specialized in order to reduce latency and area while sustaining throughput. In particular, an architecture specialization can be envisioned when synchronizing data from a sender domain which is permanently faster than the receiver one (Figure 3.22). In this architecture, the *valid_in* signal is sampled by the *data_buffer* as an additional data wire. This way, *data_in* and *valid_in* cross together the dual-clock FIFO without any additional logic needed to generate *valid_out*. Moreover, data is sampled by *data_buffer* in every possible condition, also when *valid_in* is low. Since an empty assertion can only occur when the sender is slower than the receiver, it is not possible to have an empty condition in our considered scenario. Therefore, the *empty_detector* is not required in this architecture.

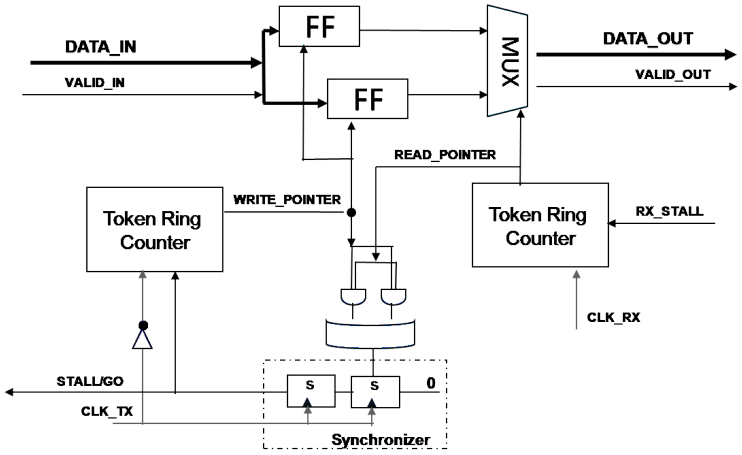


Figure 3.22: Specialized Dual-Clock FIFO.

This way, considering a scenario with the sender faster than the receiver and the specialized FIFO with a depth of 4, it is possible to achieve 100% throughput also when the condition in Formula 3.9 is not met. In fact, since no empty condition has to be detected, this architecture allows to read a safely sampled data in slot `i` also in case the `write_pointer`, waiting for full de-assertion,

has not shifted from `i` to `i + 1` position yet. Therefore, buffer resources are optimized and performance is preserved. In particular, 1 slot buffer is saved while guarantying the same throughput with respect to the architecture of Figure 3.18. Table 3.3 reports area and throughput results of the architecture depicted in Figure 3.22 as a function of buffer depth and frequency ratio of the end-nodes.

	FREQUENCY SCENARIO	FIFO DEPTH OF 4	FIFO DEPTH OF 3	FIFO DEPTH OF 2
I°	$3 \times f_{tx} > f_{rx}$	100%	100%	100%
II°	$1.5 \times f_{tx} > f_{rx}$	100%	100%	-
III°	$f_{tx} \geqslant f_{rx}$	100%	-	-

Table 3.3: Throughput of specialized Dual-Clock FIFO variants.

Please note that a similar optimization is not feasible when the receiver is permanently faster than the sender domain. The reason is that, the *full_detector* is still required because it is not possible to guarantee the absence of the full condition. In fact, when a contention takes place in the switch block, an input may loose its grant (from the arbiter) and consequently become stalled. In this case, a slower sender may succeed in filling up the dual-clock FIFO buffer, therefore, the *full_detector* is necessary to interrupt the communication.

As the baseline dual-clock FIFO, the specialized counterpart is meant to be directly tightly integrated in the switch architecture and it does not bring major implications on the switch critical path itself.

3.6.5 Comparative assessment of bi-synch FIFO variants

Experimental results for the dual-clock FIFO interface are structured into two subsections. In the first one, area benefits of the tightly coupled design principle applied to dual-clock FIFOs are quantified, while in the second one the implementation overhead for the different components in the synchronization library is characterized in relative terms. All physical synthesis experiments have been performed with a 65nm STMicroelectronics technology library.

Impact of NoC-synchronizer merging on the switch critical path

The switch configurations illustrated in Table 3.4 were synthesized, placed and routed. The first one is a fully synchronous switch with a 2 slot input buffer and a 6 slot output buffer. Moreover, the input buffer of the same switch is merged with a high-throughput dual-clock FIFO, thus augmenting the input

buffer storage to 5 slots. Finally, two remaining switch configurations implement 6 slot buffers in each input port: a fully synchronous one and one with merged dual-clock FIFOs. The last column of Table 3.4 reports the resulting maximum operating frequency.

	INPUT BUFFER	OUTPUT BUFFER	FREQUENCY
I°	2 fully synchronous	6 fully synchronous	1.43GHz
II°	5 bi-synch FIFO	6 fully synchronous	1.25GHz
III°	6 fully synchronous	2 fully synchronous	1.2GHz
IIII°	6 bi-synch FIFO	2 fully synchronous	1.2GHz

Table 3.4: 2x2 switch critical path.

The first configuration allows the switch to work at the highest frequency of 1.43GHz. In the second configuration, the switch, having the integrated FIFO synchronizer, features a lower frequency (1.25GHz). This result depends on the fact that the integration of the dual-clock FIFO has shifted the critical path from the switch crossbar and arbitration logic to the dual-clock FIFO itself in the input stage (see Figure 3.18).

In the third configuration, the switch performance decreases to 1.2GHz. This is due to the fact that the delay of the finite state machine in the input buffer is larger depending on the number of buffer slots, and it adds up to the propagation delay through the arbiter and the crossbar selection signals. Interestingly, the equivalent switch configuration with multiple clock domain support achieves the same speed and features the same critical path. As a result, the fourth configuration supports multiple clock domains while guaranteeing the same critical path of the fully synchronous switch having an equivalent overall amount of buffer storage.

The key take-away is that the tightly coupled FIFO synchronizer can determine the critical path in low radix switches (e.g., 2x2) when these latter could afford a speed higher than 1.25GHz with a typical fully synchronous input buffer. However, please notice that most topologies in use for NoC design typically require a larger number of switch I/Os.

In the following analysis we implement a 5x5 switch (used to build up a 2D mesh topology) in 2 different configurations. In particular we compare one fully synchronous switch configuration with one switch configuration integrating the FIFO synchronizers. Input and output buffers have the same size in both designs. Results are illustrate in Table 3.5.

The relevant result here is that, the increased switch radix decreases the maxi-

	INPUT BUFFER	OUTPUT BUFFER	FREQUENCY
I°	6 fully synchronous	2 fully synchronous	830Mhz
II°	6 bi-synch FIFO	2 fully synchronous	830Mhz

Table 3.5: 5x5 switch critical path.

mum operating speed, which ends up falling below the 1.25GHz threshold beyond which the dual-clock FIFO behaves as speed limiter. Below this threshold, the critical path moves somewhere else, hence the support for multiple clock domains does not bring any limitation to the maximum achievable performance.

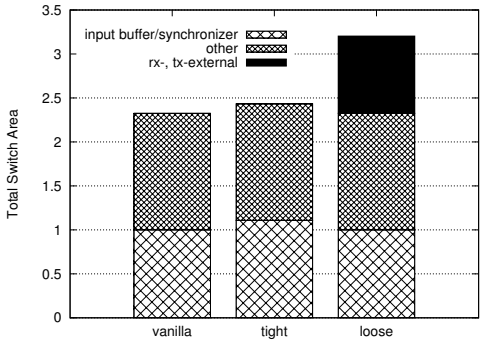
Area benefits of NoC-synchronizer merging

Three different designs have been compared. The first one is the conventional 5x5 vanilla (fully synchronous) switch with a 6-slot input buffer and a 2-slot output buffer per port. The second one is a switch where a dual-clock FIFO with 6 buffer slots has been merged with each input port. In order to carry out a fair comparison with the vanilla switch, total buffering resources have been kept equal, i.e., the output buffer size in the switch with the FIFO synchronizers has been reduced from 6 to 2 slots. The last configuration is a vanilla switch (6-slot inputs, 2-slot outputs) with external dual-clock FIFO (6 buffer slots) per input port.

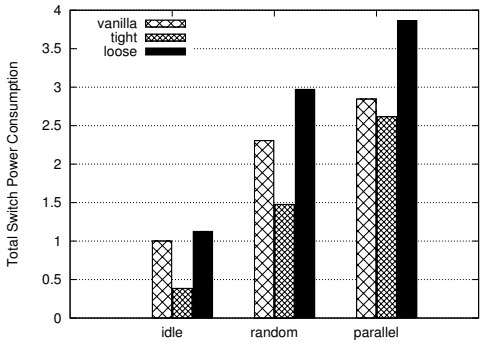
To assess area occupancy, all the above switch configurations have been synthesized, placed and routed at the same target frequency of 1GHz. Total area of the tightly coupled system exhibits almost the same area footprint of the vanilla switch. This is a direct consequence of the fact that exactly the same buffering resources have been deployed in a specular fashion (between input and output).

As showed in Figure 3.23(a), being the input buffer size of the three systems the same (6 slots), there is a similar amount of cell area devoted to either only buffering (vanilla switch) or buffering and synchronization (tightly and loosely coupled switch). Moreover, the loosely coupled system features the same area overhead (with the same distribution of *input_buffer* and *other* cell area) of the other switches plus a further synchronization area due to the external block implementing the dual-clock FIFO.

These results point that the merging approach applied to the dual-clock FIFO design achieves up to 24% of area saving with respect to the loosely coupled



(a) Area breakdown.



(b) Power consumption.

Figure 3.23: Post-layout normalized results of area (a) and power (b) for a switch with a dual-clock FIFO synchronizer.

design methodology.

To assess the power consumption of a switch integrating dual-clock FIFOs on the input ports, the vanilla, tightly and loosely coupled designs have been tested under different traffic patterns: idle (to measure standby power), random (target output port of input packets is randomly selected) and parallel (no switch internal conflicts). Post-layout simulations have been carried out at 800MHz. The switch with the external dual-clock FIFO is the most power greedy under all possible traffic patterns, as showed in Figure 3.23(b). This is due to a larger amount of buffering resources. From the power viewpoint, there is a substantial benefit when integrating the dual-clock FIFO in the switch architecture. In fact, the tightly coupled design is the most power efficient among those under test and achieves up to 51% power saving (under random traffic).

The motivation lies in the inherent clock gating that is implemented by our

dual-clock FIFO, which clocks only one bank of flip-flops at a time out of the total input buffer. If the incoming data is not valid, then the token ring circuit does not even switch thus gating the entire input buffer. Obviously a similar clock gating technique can be applied to the vanilla switch as well, and in fact the key take-away here should be that the dual-clock FIFO integration into the switch does not imply any major power overhead, as long as buffer depths of at least 6 flits are used in all switch variants for performance optimization.

Above all, these results indicate that with the proposed architecture design techniques it is possible to evolve a fully synchronous switch to a switch supporting relaxation of synchronization assumptions with marginal implementation overhead. This is a key enabler for the GALS paradigm in the context of NoC-centric MPSoCs.

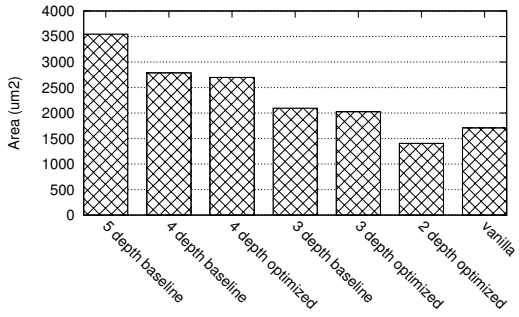
Characterization of synchronization library components

In this experiment a comparison between the baseline dual-clock FIFO architecture and its specialized version has been carried out. By varying buffer depth of each solution, we could span the entire range of components of the proposed synchronization library.

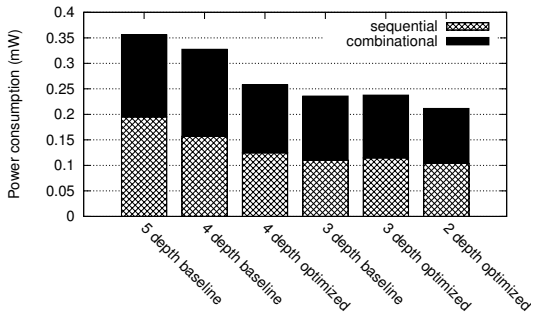
Buffer depth of the baseline architecture ranges from 5 to 3 slots while supporting all possible frequency ratio scenarios. On the other hand, the specialized version requires from 4 to 2 buffer slots and is able to work with all the sender/receiver frequency ratio where the sender is always faster than the receiver. Furthermore, for the sake of comparison, a fully synchronous input buffer (with minimum number of slots, i.e. 2) has been considered.

As reported in Figure 3.24(a), when the buffer depth is reduced in both the architectures, a corresponding reduction of overall cell area takes place. The reason lies in a reduction of both sequential and logic cell area. In fact, by reducing the buffer size, sequential elements are obviously reduced along with a simplification of the detector and all the combinational logic. It is interesting to note that when considering the two architectures with the same buffer depth (e.g., 4) there is a marginal area reduction of the specialized version. This is due to the absence of the empty detector which is no longer required. A further consideration stems from a comparison between the vanilla input buffer (2-slots) and the equivalent size specialized dual-clock FIFO. Of note, the dual-clock FIFO is more area efficient than the vanilla input buffer because the FSM taking care of *valid* signals is simplified.

Power consumption has been computed by considering all the architectures



(a) Area.



(b) Power consumption.

Figure 3.24: Area (a) and power (b) consumption of baseline and specialized dual-clock FIFO architectures with different buffer depths.

operating at the same frequencies that permit all them to correctly work while supporting full throughput communication. Experiments have been performed with a sender four times faster than the receiver. Sender clock frequency has been set to 1GHz. All the architectural variants have been tested under parallel traffic patterns.

As reported in Figure 3.24(b), a reduction of the buffer depth implies a corresponding reduction of power consumption in both baseline and optimized architectures. Interestingly, by comparing the result of the two architectures with the same buffer depth of 3, it is possible to note a similar power consumption. Conversely, when comparing the baseline and the optimized architectures with a buffer depth of 4, it is clear that a higher consumption takes place for the baseline one. In fact, the baseline dual-clock FIFO with 4 and 5 buffer slots

requires a larger token ring counter due to the high fanout net. This is not an issue in the optimized FIFO as there is no empty detector.

Following the previous considerations, experimental results proved that a clock-gated fully synchronous input buffer requires the similar power consumption of the baseline dual-clock FIFO when assuming the same buffer depth and a fully synchronous scenario.

3.7 Discussion

During the final step of our investigation, the previously presented mesochronous variants are compared with their analogous dual-clock FIFO ones. In particular, let us compare four NoC switch variants engineered to support different synchronization schemes. The baseline switch is again from the xpipesLite architecture (see Section 3.4). Its input stages have been augmented at first with a mesochronous 3 slot buffer interface able to tolerate up to 100% positive and negative skew,

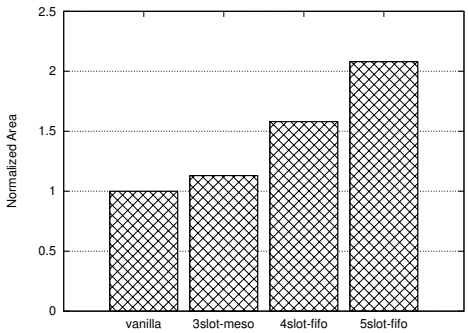


Figure 3.25: Area occupancy of NoC switches with different synchronization interfaces.

Among the baseline dual-clock FIFO options, the only interface that is able to work without a performance decrement (full throughput condition) in a mesochronous scenario is that with 5 slots. The reason is that, other solutions with smaller buffer slots, are not able to interface two clock domains working at the same frequency (see Table 3.2) retaining a full throughput condition. Moreover, the specialized 4-slot dual-clock FIFO variant (see Table 3.3) is the second solution that can work in a fully synchronous regime with arbitrary clock phase offsets. Summarizing, in the library of dual-clock FIFOs, there exist two possible solutions to be deployed in a mesochronous scenario: (i) a

baseline 5-slot and (ii) a specialized 4-slot interfaces. As depicted in Figure 3.25, the most specialized dual-clock FIFO is almost 30% more area expensive (in absolute term) with respect to the mesochronous solution.

Obviously, the reason stems from the fact that the dual-clock FIFO has been natively conceived for a multi-frequency application domain whereas the mesochronous interface is designed ad-hoc for the scenario under investigation (same clock frequency, unknown phase offset) and requires less control logic.

It should be finally observed that mesochronous NoCs are the reference solution for ultra-low cost synchronizer-based GALS systems. When observing the target platform of Figure 3.1, it makes a conscious use of area/power-hungry dual-clock FIFOs, which end up being instantiated only at network boundary. Instead, more compact mesochronous synchronizers are used inside the network, thus minimizing the area and latency overhead.

3.8 Conclusions

This chapter presented a library of synchronization interfaces for use in cost-effective MPSoCs. The need to avoid costly general-purpose interfaces fitting all kinds of performance requirements and layout constraints has been addressed by specializing the synchronizers for the different operating conditions. First of all, the configuration degrees of freedom exposed by the synchronizers are used (e.g., buffer size), but also fully customized architecture solutions are engineered.

In dual-clock FIFOs, architecture specialization can be carried out based on application performance requirements and on the ratios between transmitter and receiver speeds. When it comes to mesochronous links, even simple synchronizer schemes can achieve large skew tolerance, therefore the key requirement becomes the adaptation to the actual layout conditions (e.g., critical timing paths and link lengths and their interrelation). In all cases, our library synchronizers have been co-designed (merged) with the NoC building blocks thus helping designers meet tight area, power and latency budgets via carefully engineered switch input ports. A key take-away of this work is that in a mesochronous scenario, a generic dual-clock FIFO component is not the right choice for an area and power budget limited designer but a customized mesochronous interface (selected from our library) represents a more cost-effective alternative. The design techniques of synchronization interfaces presented in this work enable to replace a fully synchronous switch with an aug-

mented variant supporting the relaxation of synchronization assumptions at marginal implementation overhead, thus making GALS technology affordable for NoC-centric MPSoCs.

4

The Moonrake Chip

In this chapter we present a complex GALS ASIC demonstrator in 40 nm CMOS process further improving the maturity of the developed GALS technology presented in Chapter 3. This chip, named Moonrake, compares synchronous and GALS synchronization technology in a homogeneous experimental setting: same baseline designs, same manufacturing process, same die. The chip exploits the library of synchronization interfaces presented in Chapter 3 to validate GALS technology for network on-chip communications and bridges the gap to actual silicon implementation. In a first step, the advance with respect to state-of the-art demonstrators will be discussed. Next, the Moonrake architecture and its floorplan will be presented. Then, the focus will be on the test setup and the test results.

4.1 GALS Systems and Demonstrators

Globally Asynchronous Locally Synchronous (GALS) technology has been proposed many years ago as an alternative to the traditional synchronous paradigm for chip synchronization [138]. Although significant potential was reported by the academia, the GALS methodology has never taken off in the industry. However, the growing challenges, imposed by the unrelenting pace of technology scaling to the nanoscale regime, urge for an efficient and safe system-level integration methodology. Consequently, we have targeted the implementation of a chip in the advanced 40 nm CMOS process, aiming at the assessment of GALS technology for nanoscale designs. The chip was named Moonrake. Our intention was to evaluate GALS vs. standard synchronous technology on the same die, by implementing synchronous and GALS counterparts of the same baseline designs.

For on-chip networking applications, a GALS system results from the inter-

connection of domains with different synchronization assumptions. Chapter 3 was focus on the provision of flexible and cost-effective interfaces for arbitrary composability. In this direction, the novel mesochronous and dual-clock FIFO synchronization interfaces, aiming at low-area/power/latency overhead while preserving timing robustness, were integrated into NoC test structures exposing (and comparing) a range of flexible GALS solutions.

In past years several GALS chip implementations were reported. Many of them were focused on point to point GALS architectures (several designs from ETHZ [138], WLAN baseband processor from IHP [139]), while more recent implementations have also explored the GALS NoC concept (NEXUS chip [140], recent SpiNNaker [141], and three implementations from LETI, namely FAUST [144], ALPIN [142], and MAGALI [143] chip). The Magali chip is probably the most complex within the GALS demonstrators followed by the Moonrake chip. Improvement of this latter upon state-of the- art concerns the most aggressive manufacturing process, the higher industrial relevance of its `galsified` designs, the maturity of implemented synchronization interfaces. Above all, both the baseline synchronous and the GALS counterparts of the same designs are now available on the same chip, thus paving the way for benchmarking in a truly homogeneous experimental setting (both at the architecture and at the technology level).

The contributions of this chapter are as follows:

- The design flow followed for different GALS systems is illustrated and compatibility with mainstream standard cell libraries and design toolflows is discussed.
- The feasibility of GALS NoCs linking sub-systems with heterogeneous timing assumptions by means of area/power/latency optimized interfaces while preserving timing margins has been demonstrated.
- Synchronous and GALS counterparts of the same baseline designs, implemented in the same demonstrator chip, have been compared in terms of area, pointing out counterintuitive benefits of the GALS design style.

This chapter further improves the maturity of the developed GALS technology presented in Chapter 3 and bridges the final gap to actual silicon implementation on the 40nm technology. In this direction, it illustrates a test-chip design and fabrication process validating the feasibility and effectiveness of the developed GALS NoC concept in nano-scaled technology sub-systems. Also, the testchip fabrication was a valuable experience of technology transfer from

academia to industry, thus breaking the barriers that have prevented the success of mesochronous technology for GALS NoCs so far in alternative design experiences.

The remainder of this chapter is organized as follows: Section 4.2 will describe the testchip architecture composed of replicated sub-systems each validating a different feature of the new synchronization technology and it will describe the optimization required to meet the pin budget. Section 4.3 highlights the global and local constraints imposed in the testchip floorplan and the post place&route area results of each sub-systems. Section 4.4 illustrates the test environment and the testflow composed of continuity, functional and operating current tests. Next, Section 4.5 will present the testchip results evaluating the NoC subsystems in terms of skew and frequency robustness, power consumption and yield. Finally, Section 4.6 summarizes the contribution of this chapter.

4.2 Moonrake Testchip Architecture

The need to validate a new synchronization technology and the poor experimentation of the newly developed 40nm Infineon technology made the risk associated with testchip fabrication pretty high. In order to minimize such risk, the final decision for the testchip was to instantiate a number of small and replicated sub-systems each validating a different feature of the new synchronization technology. Replication of the sub-systems enables to amortize the risk of manufacturing concerns. To push this approach to the limit, the decision was to replicate sub-systems validating the same concepts even more in order to reflect different operating speeds. In practice, some sub-systems were selected for clocking from an external low-speed clock source through a JTAG interface, while other sub-systems with similar functionality were selected for higher-speed clocking from a PLL. The lower speed of clocking from an external source is associated not only with the inherent limitations of a possible test setup, but also with the limited driving capability of I/O pins. In both cases (internal vs external clocking), the sub-systems testing mesochronous interfaces were made capable of tolerating an increasing amount of clock skew between transmitter and receiver clock domains.

Figure 4.1 provides an overview of the testchip block diagram. Each subsystem replicates the same baseline network-on-chip template, a 2-ary 1-mesh topology, where every switch of the network is connected to 2 cores (a memory core and a tester block). The cores are connected to JTAG interfaces in order to

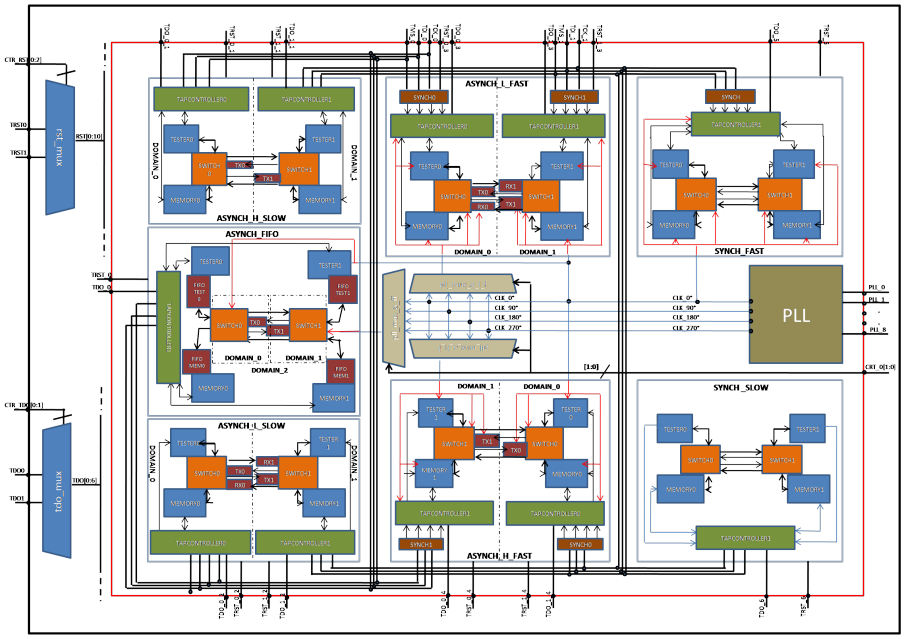


Figure 4.1: Block diagram of the NoC testchip.

be programmable by external input pins. In particular, the testchip compares a fully-synchronous NoC sub-system with two mesochronous NoC sub-systems (a first one composed by loosely coupled and a second one by tightly coupled mesochronous synchronizers) and a GALS NoC sub-system integrating also dual-clock FIFO interfaces. As mentioned above, in order to test the reliability of the source synchronous communication and to compare sub-system latency, area and power under different frequency constraints, every sub-system is designed twice to work at low and high frequency. The sub-systems designed to work at low frequencies (i.e. frequency lower than 260MhZ) are fed by an external clock injected through the input pins of the JTAG interface. On the contrary, the sub-systems designed to work at high frequency receive the clock by a PLL (Phase Lock Loop) integrated into the testchip itself.

The PLL is able to generate four different clock phases (0, 90, 180, 270) on distinct module output pins with a maximum speed of 400MhZ. The PLL frequency can be set through external pins. Moreover, each synchronizer-based sub-system receives two separated PLL clocks. In fact, the phase of the first clock is statically zero while the phase of the second clock can be selected by exploiting a 4x1 multiplexer and the external pins for mux programming. A dedicated multiplexer is implemented for every synchronizer-based sub-

system. As a result, the testing of the source synchronous interfaces can be performed also under different skew constraints. To notice that both the frequency and the phase of the PLL clock can be set during the power-on phase of the testchip. Also, the multiplexer delay ends up contributing to the clock phase offset between two mesochronous domains in the same sub-system, therefore such multiplexers have been synthesized for ultra-high speed of operation. The distinctive features of every sub-system can be described as follows:

- The **Synch_Slow** sub-system is demonstrating a fully synchronous NoC communication at low frequency. The sub-system is composed by 1 JTAG controller, 2 switches, 2 memory cores and 2 tester blocks and is clocked by a unique external clock (see figure 4.2.b).
- The **Synch_Fast** sub-system implements a fully synchronous NoC communication at high frequency. The sub-system is composed by 1 JTAG controller, 2 switches, 2 memory cores, 2 tester blocks and 1 synch block. The synch block implements a brute force synchronizer and it is used to synchronize the JTAG inputs with the fast and real sub-system clock which comes from the PLL (see figure 4.2.a).

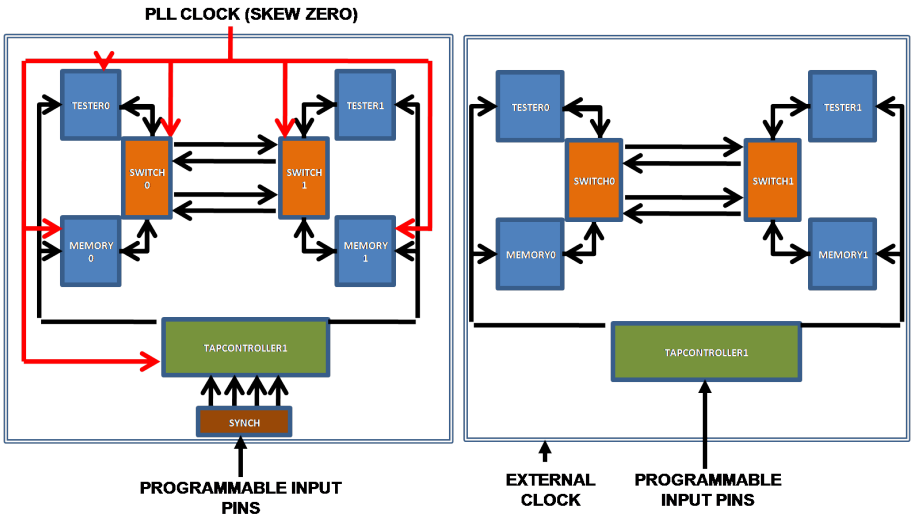


Figure 4.2: Synchronous sub-systems: (a) the *Synch_fast* design (on the left-side) and (b) the *Synch_slow* design (on the right-side).

- The **Asynch_Loose_Slow** sub-system has two mesochronous clock domains having the same low clock frequency but arbitrary phase offset.

This sub-system uses loose mesochronous synchronizers (RX, for the datapath, and TX, for the control path) placed in the bidirectional link next to the switches. The RX module synchronizes the data and the TX module synchronizes the flow control signal absorbing the phase offset between the domains. The sub-system is composed by 2 JTAG controllers, 2 switches, 2 memory cores, 2 tester blocks, 2 loosely coupled RX datapath synchronizers and 2 loosely coupled TX control path synchronizers (see figure 4.3.a).

- The **Asynch_Loose_Fast** sub-system has two mesochronous clock domains having the same high clock frequency but arbitrary phase offset. The domain0 is driven by PLL clock with 0 skew while the domain1 is driven by the output of the dedicated multiplexer, thus selecting one of the pre-configured clock phase offsets. This sub-system uses loosely coupled mesochronous synchronizer (RX and TX) placed into the bidirectional link next to the switches. The sub-system is composed by 2 JTAG controllers, 2 switches, 2 memory cores, 2 tester blocks, 2 loosely coupled RX synchronizers (for the data path), 2 loosely coupled TX synchronizers (for the control path) and 2 synch blocks for synchronization with timing of the input pins (see figure 4.3.b).

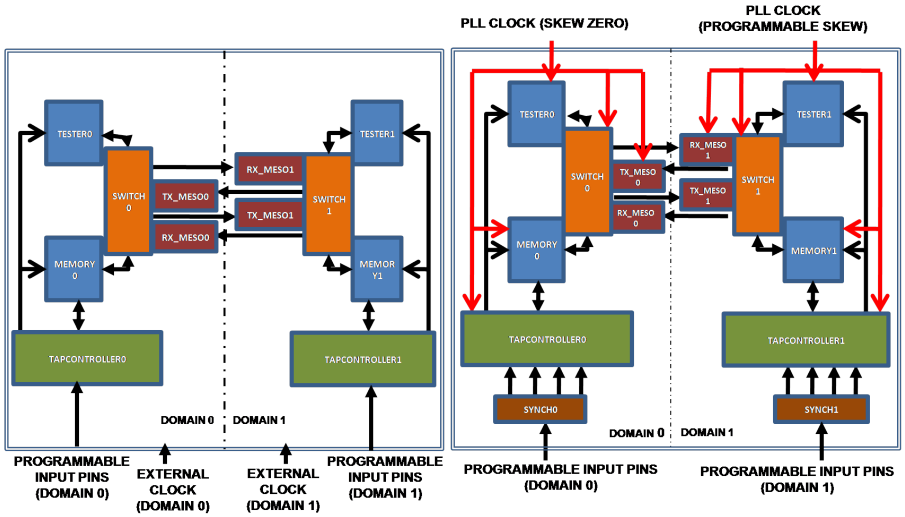


Figure 4.3: Loosely coupled sub-systems with mesochronous synchronizers: (a) the *Asynch_Loose_Slow* design (on the left-side) and (b) the *Asynch_Loose_Fast* design (on the right-side).

- The **Asynch_Hybrid_Slow** has two mesochronous clock domains having the same slow clock frequency but arbitrary phase offset. This subsystem has two tightly coupled mesochronous synchronizers integrated into the switch input stage (for the data path) and two loosely coupled TX synchronizers (for the control path) into the bidirectional link. The mix of the synchronizer implementation styles explains the name of this scheme (*hybridcoupling*) and is motivated by the need to break a long timing path originating in the upstream output buffer, going through the mesochronous interface at the downstream switch and going back to the upstream switch. By implementing the control path synchronizer as loosely coupled, a minor area overhead is incurred (this is a 1-bit synchronizer), while breaking the timing path across the link and resulting in higher operating speeds for a given link length (see Section 3.5.3). The sub-system is composed by 2 JTAG controllers, 2 switches, 2 memory cores, 2 tester blocks, 2 tightly coupled RX synchronizers and 2 loosely coupled TX synchronizers (see figure 4.4.a).

- The **Asynch_Hybrid_Fast** sub-system has two mesochronous clock domains having the same high clock frequency but arbitrary phase offset. This sub-system has two tightly coupled mesochronous synchronizers integrated into the switch input stage and two loosely coupled TX synchronizers into the bidirectional link. Again, this is a hybrid coupled synchronizer-based design. The domain0 is driven by PLL clock with 0 skew while the domain1 is driven by the output of the dedicated multiplexer. The sub-system is composed by 2 JTAG controllers, 2 switches, 2 memory cores, 2 testers, 2 tightly coupled RX synchronizers, 2 loosely coupled TX synchronizers and 2 synch blocks for synchronization with the timing of the input pins (see figure 4.4.b).

- The **Asynch_Fifo** sub-system has three clock domains: domain0 and domain1 have the same high-speed clock with arbitrary phase offset while domain2 has a distinct clock signal switching at a lower speed. The cores and the switches have true independent clocks with distinct frequency and offset. Since the sub-system has different clock frequencies, two tightly coupled dual-clock FIFOs are integrated into the switch input stage and two additional loosely coupled dual-clock FIFOs are instantiated next to the cores in order to synchronize the information coming from the cores and from the switches, respectively. Please notice that the loosely coupled dual-clock FIFOs have not been merged with their associated network interfaces to reflect the case where custom network

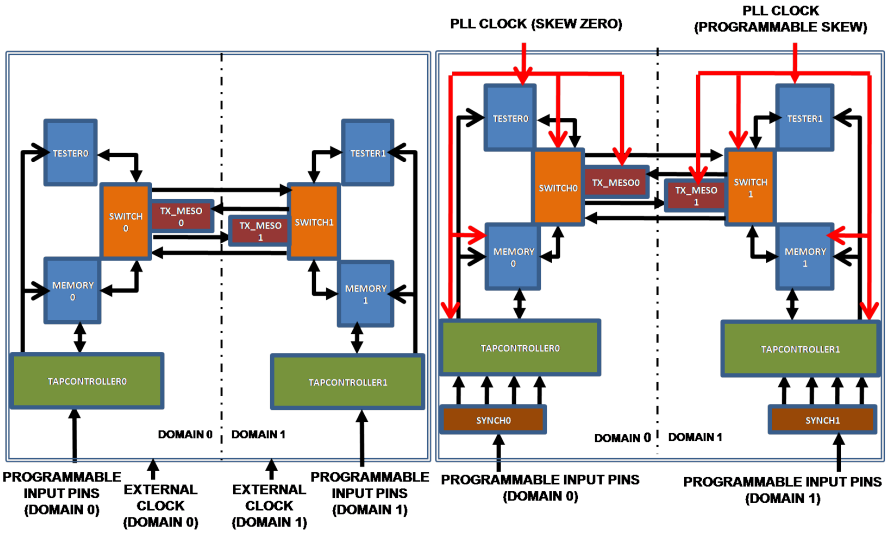


Figure 4.4: Hybrid coupled sub-systems: (a) the *Asynch_Hybrid_Slow* design (on the left-side) and (b) the *Asynch_Hybrid_Fast* design (on the right-side).

interfaces need to be rapidly reused in a GALS system, and therefore the merging process may turn out to be a lengthy or cost-ineffective process. In principle, nothing prevents from performing such a merging process with the network interface as well, even considering that the input stage of the network interface (and of its response path in particular) is composed by a FIFO. Moreover, the sub-system has two tightly coupled mesochronous synchronizers integrated into the switch input stage and two loosely coupled TX synchronizers into the bidirectional link. The domain0 is driven by PLL clock with 0 skew while the domain1 is driven by the output of the dedicated multiplexer, thus selecting a clock phase offset. Domain2 is clocked by an external clock through the JTAG interface. The sub-system is composed by 1 JTAG controllers, 2 switches, 2 memory cores, 2 tester blocks, 2 tightly coupled RX synchronizers, 2 loosely coupled TX synchronizers, 4 tightly coupled dual-clock FIFOs, 4 loosely coupled dual-clock FIFOs and 1 synch blocks (see figure 4.5). It is worth observing that this sub-system reflects a typical operating condition: the IP cores operate at a slower speed than the on-chip network, where this latter can be inferred as the collection of mesochronous sub-domains. This sub-system may be considered as the most complex one in the testchip and aims at validating the most advanced synchronizer-NoC merging techniques developed in the context

of the thesis.

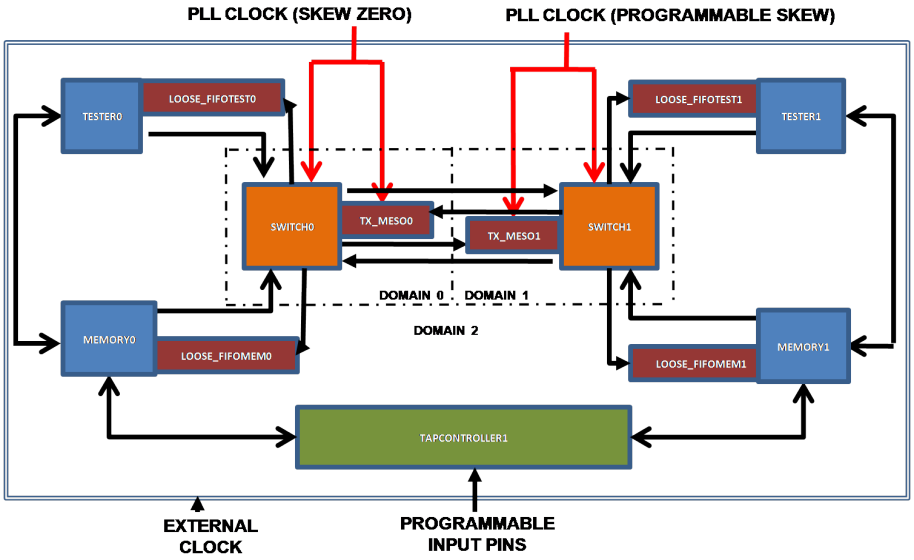


Figure 4.5: Dual-clock FIFO design.

4.2.1 PIN Requirement

As a final step, three further optimizations were required to meet the total pin budget. This latter was very tight since, as anticipated, the Moonrake architecture is based on the parallel implementation of the synchronous and the GALS variants of the same baseline designs. In order to reduce chip area, the same pad frame for both was re-used, and therefore data input and output pins were in most cases multiplexed.

First of all, the programmable input pins and the external clocks ended up being shared between the 7 sub-systems. Consequently, in order to avoid an undesirable parallel setting of all the 7 sub-systems, the reset input signal crosses a de-multiplexer designed to enable a single sub-system at a time (i.e., the active reset drives only one of the seven sub-systems). As a result, it is possible to select the operative sub-system by driving the de-multiplexer with three dedicated input pins.

Similarly, a multiplexer allows a single sub-system at a time to exploit part of the output pin resources (two shared output pins). In fact, each fast sub-system has both its output pins (one per JTAG) shared and each synchronizer-based

slow sub-system has the output pin belonging to the domain0 (zero skew) dedicated but the output pin of the domain1 shared. This strategy allows slight output pin resource saving with a marginal impact on the testability of the design. Indeed, each slow sub-system (which can be viewed also as the safe and backup version with respect to the fast counterpart) has still a dedicated output pin ensuring the sub-system testability, even when a failure in the output multiplexer occurs.

Moreover, as mentioned above, the phase of the PLL clock is selected through a 4x1 multiplexer. As a result, each sub-system tightly belongs to its own clock phase multiplexer (the whole sub-system fails when a failure affects its multiplexer). Then, an instance of the clock phase multiplexer is replicated in front of each synchronizer-based sub-system to avoid multiples sub-system failures due to single multiplexer error. Anyway, to further reduce the number of global input pins, the three clock phase multiplexers are still driven by two shared input pins. To note that the final goal is to test a sub-system at a time and the parallel clock phase setting of the synchronizer-based sub-system does not reduce the testability of the design.

As a conclusion, the input pins and the external clock sharing allows to save 33 input pins (they scaled from 44 to 11), the output pin sharing reduces from 11 to 8 the required output pins and, the clock phase multiplexers optimization saved 4 additional control pins. Finally, the global pin number scaled down from 70 to 30 (9 input/output pins are considered to drive the PLL). This approach enabled a massive optimization of the output/input pin requirements while marginally affecting the flexibility of the architectural test structures.

4.3 Floorplaning Constraints

The modularity of the testchip NoC architecture allowed an easier place-and-route. In fact, every sub-system was synthesized independently and treated as a soft-macro. Anyway, global and local constraints were imposed respectively in the testchip floorplan and in each sub-system. In particular, the final global testchip floorplan is represented in figure 4.6. The PLL is placed on the left side of the design and the seven NoC sub-systems are on the right side. To note that the PLL required a relevant percentage of the total testchip area. At a global floorplan level, the soft-macro of the fast sub-systems (in figure 16 denominated as Hybrid_Fast, Loos_Fast, Synch_Fast and Fifo_Fast sub-systems) were constrained to be placed as close as possible to the PLL. In fact, the distance between the PLL and the fast designs was minimized in order to reduce the

length of the clock tree branches. This strategy mitigated the unpredictability of the skew between the clock leaves and, as a result, it increases the accuracy of the sub-system intra-domain skew set by the external pins.

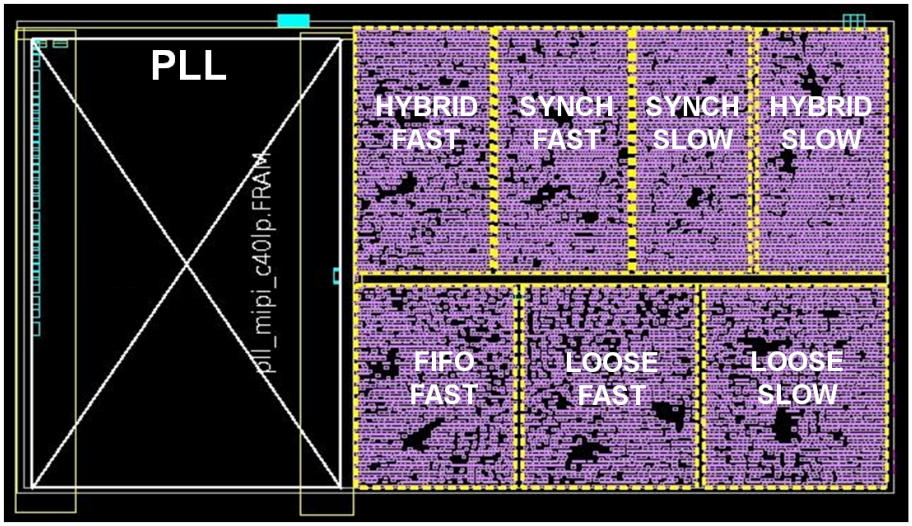


Figure 4.6: NoC testchip floorplan.

Additionally, at a sub-system level, the source synchronous communication was constrained as well. In particular, figure 4.7 depicts the source synchronous communication in the hybrid coupled sub-systems (*Asynch_Hybrid_Slow* and *Asynch_Hybrid_Fast*). In this case, the TX_MESO0 and TX_MESO1 modules were placed respectively close to the SWITCH0 and SWITCH1 modules. That is required to ensure that an additional link delay does not nullify the ideal synchronization of the TX_MESO module output, i.e. the flow control signal. Moreover, the links were constrained in order to have the same delay regardless of their crossing direction (i.e. D1 = D2). As a result, the skew test experiments can be performed with an additional degree of freedom regardless of the data crossing direction. Finally, the data crossing the source synchronous links was routed together with the strobe signal to match their propagation delay (bundled routing) with industry-available physical synthesis techniques.

As regards the source synchronous communication in the loosely coupled sub-systems (*Asynch_Loose_Slow* and *Asynch_Loose_Fast*), also the RX_MESO0 and RX_MESO1 modules together with the TX_MESO0 and TX_MESO1 modules were placed respectively close to the SWITCH0 and SWITCH1 modules (see Figure 4.8). As mentioned before, that is required

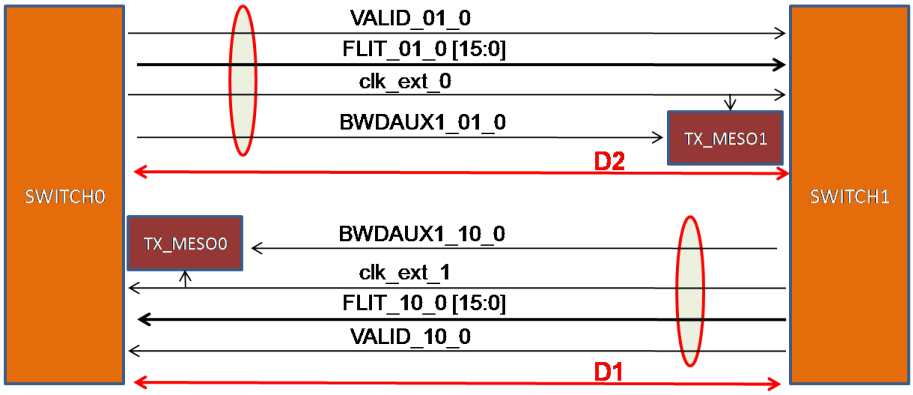


Figure 4.7: Source synchronous communication in the hybrid coupled sub-systems and PnR constraints.

to ensure the ideal synchronization of the data/flow control of the RXMESO/TXMESO output. Similarly to the hybrid coupled communication, the information on the link are bundled routed and the D1 crossing delay corresponds to the D2 crossing delay.

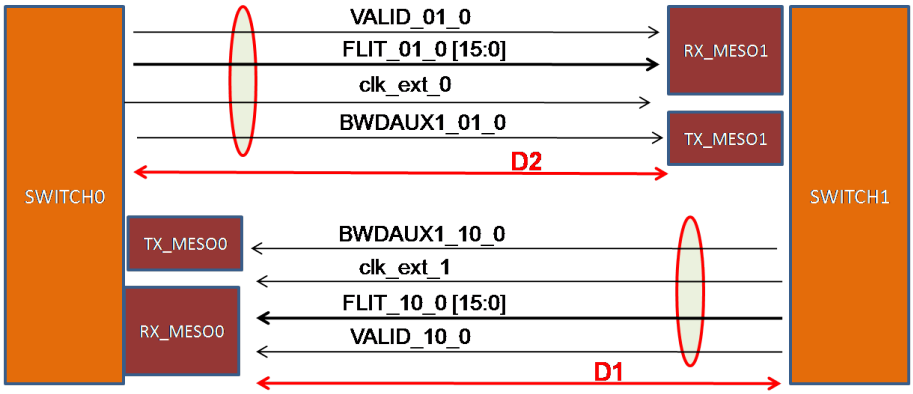


Figure 4.8: Source synchronous communication in the loosely coupled sub-systems and PnR constraints.

As final step, the source synchronous communication by means of dual-clock FIFOs was constrained. In particular, since in the `Asynch_FIFO` sub-system the communication between switches is constrained as in the hybrid coupled sub-system, figure 4.9 focuses on the source synchronous communication between the switches and the cores. In this case, the external dc-FIFO modules

(i.e. the interfaces taking care of synchronizing the data to the cores) were placed close to their respective memories (MEMORY0 and MEMORY1) or testers (TESTER0 and TESTER1). As before, that is required to ensure the ideal synchronization of the dc-FIFO data output. Similarly to the switch communication scenario, the information on the dc-FIFO link are bundled routed and the D1 crossing delay corresponds to the D2 crossing delay.

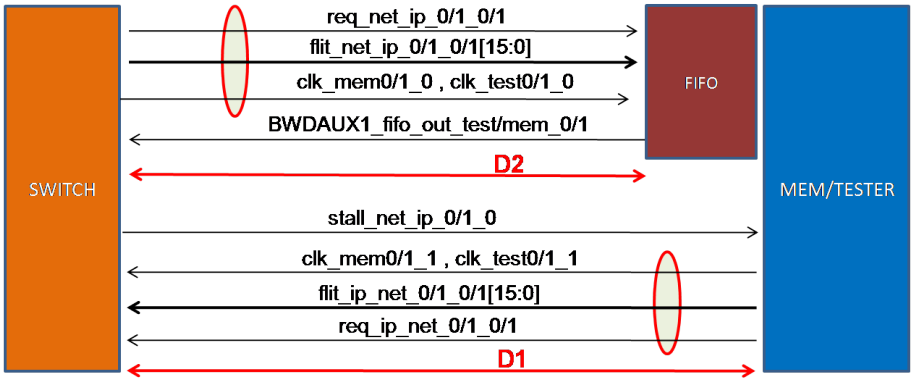


Figure 4.9: Source synchronous communication in the dual-clock FIFO sub-system.

4.3.1 Area results

This section compares the seven sub-systems under an area point of view. Every sub-system has been synthesized independently by means of the 40nm standard cells Infineon Technology and treated as a soft-macro. In particular, the slow sub-system (*Asynch_Hybrid_Slow*, *Asynch_Loose_Slow* and *Synch_Slow*) were synthesized at 100Mhz while the fast sub-systems (*Asynch_Hybrid_Fast*, *Asynch_Loose_Fast*, *Asynch_FIFO* and *Synch_Fast*) at 500Mhz.

As mentioned before, the key idea of the testchip was to instantiate a number of small and replicated sub-systems each validating a different feature of the new synchronization technology. Basically, all the seven sub-systems integrate the same modules (2 memory, 2 tester, 2 switches and 1/2 JTAG) but the communication between these modules is enabled by different source synchronous interfaces. As a result, the source synchronous interfaces provide the main contribute in terms of area overhead with respect to the fully synchronous baseline sub-system (*Synch_Slow* and *Synch_Fast*). See figure 4.10 for area results.

In particular, the fast solutions present a similar area footprint with respect to the slow counterpart. In fact, although the fast sub-systems were synthesized at a higher frequency than the slow sub-systems, both the solutions meet the target frequency constraint with a large slack and as a result the synthesis tool was able to provide a well optimized gate-level netlist in terms of area footprint in both the cases. To note that the additional *synch* modules integrated in the fast sub-systems provided a negligible area overhead. Interestingly, since the goal of the testchip was to evaluate the NoC synchronization technology, the instantiated memories and testers are elementary and as a result the switches required more than the 50% of the total area footprint of each sub-system. Following a sub-system comparison in terms of area:

- The *Asynch_Hybrid_Slow* sub-system required the 7% of additional area than the fully synchronous baseline sub-system (*Synch_Slow*). In particular, this area overhead is due to the additional Jtag module required to configure the second frequency domain (3%), the 2 Tx_Meso flow control synchronizers (1.5%) and the 2 switches integrating the data mesochronous synchronizer (2.5%).

- The *Asynch_Loose_Slow* sub-system required the 17% of additional area than the fully synchronous baseline sub-system (*Synch_Slow*). In particular, this area overhead is due to the additional Jtag module (3%), the 2 Tx_Meso flow control synchronizers (1.5%), the 2 Rx_Meso data synchronizers (5%) and the area overhead of the 2 switches (5.5%). To note that the input buffers of the two switches were extended by two slots since 2 clock cycles of latency are added in the round-trip from the transmitter and receiver synchronizers and vice versa on the link. Interestingly, the *Asynch_Loose_Slow* sub-system had a 10% of area overhead with respect to the *Asynch_Hybrid_Slow* sub-system.

- The *Synch_Fast* sub-system required 4% of additional area than the *Synch_Slow* sub-system mainly due to the higher synthesis frequency.

- The *Asynch_Hybrid_Fast* sub-system and the *Asynch_Loose_Fast* sub-system followed the same area trend than their slow counterparts. In fact, they feature respectively 7% and 17% of area overhead with respect to the *Synch_Fast* sub-system.

- The *Asynch_FIFO* sub-system required 30% of additional area than the fully synchronous fast sub-system (*Synch_Fast*). In particular, this area overhead is mainly due to the 4 loosely coupled dual-clock FIFOs

(17%) and the 4 tightly coupled dual-clock FIFOs (9%). To note that all the dual-clock FIFOs were with 5 buffer slots and, the tightly coupled dual-clock FIFOs have been merged with the switch input buffer. Then, since the xpipes switches are natively composed of 2 input buffer slots, the total amount of buffering resources is increased. In order to keep the total buffering resources and reduce the final source-synchronous switch area overhead the output buffer size could be reduced from 6 to 3 slots, since additional buffer slots come with the synchronization-augmented input buffer.

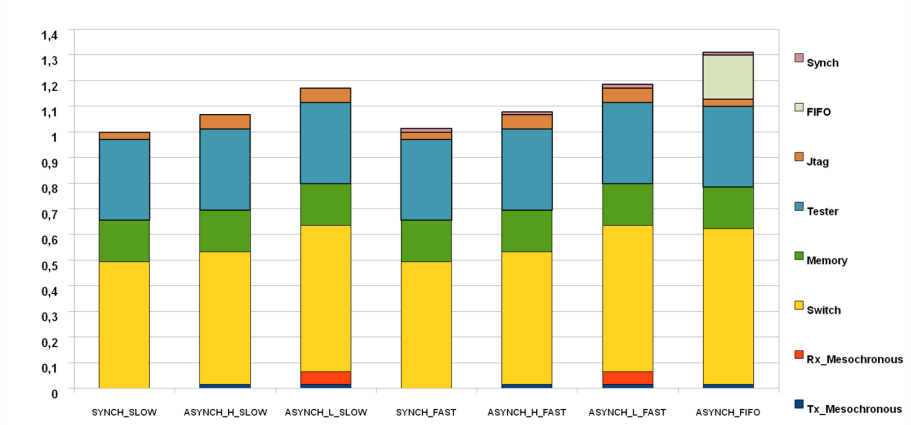


Figure 4.10: Area breakdown of the seven sub-systems.

As a final consideration, the global area footprint of the testchip was composed by two additional main contributions: the PLL module, and the pin pads. In particular, almost half of the total required area was devoted to the PLL instantiation.

4.4 Test Setup

In this section we describe the test performed on the testchip and the test specification that had to be provided to enable test engineers to create a DUT board design specification and implement the tests themselves. The testing phase took place at IHP and took profit by IHPs testing equipment.

The Moonrake tests were mainly conducted on the Verigy 93000 SOC (see Figure 4.11). The Verigy 93000 SOC is a high performance production test

system. It has a digital-dominant configuration with licensed speeds up to 800MB/s. The hardware is capable of up to 3.6GB/s per channel. The test system provides a set of commonly used standard test functions such as functional test, current measurements, sweep tests etc. Low level programming for user/device specific requirements is available through a rich C++ API as well as direct firmware access. The 93000 system was used for interactive functional verification of the NOC tests, sweep tests on parts of the NOC and for the standard current measurements.

Figure 4.11: Verigy test platform.

The test specification had to be provided by filling in appropriate templates and supplying vector data files (EVCD). The vector data files are generated from logging the data flow on the design ports during a functional simulation. As the simulation is event-driven also the resulting test data files are event-based and need to be converted into a new format in order to be read by the test system software. This vector translation (conversion from event based format to cycle based format) extracts the actual vector data at defined points following the timing provided in the test specification. The successful completion of this task depended on several parameters and, as a result, a special attention was devoted to provide the test specifications and to define the minimum number of simulation runs able to capture the real testchip quality metrics.

In particular, for each test/vector set using a different timing, a separate timing set specification was provided. As an example, some of the main information required in the test specification for each timing set are the following:

- Source cycle frequency: the frequency used in the simulation defining the period to be used for the cycle representation of the simulation data.
- Target cycle frequency: the frequency to be used during the test.
- Sample point for source vector data: the sample point is related to the simulation since it specifies the point within the cycle where the logical value should be sampled during vector translation.
- Exact timing position of read strobe impulses to capture outputs.

Moreover, if the tests have to be executed at several different frequencies or clock phase, timing data is required for each frequency/phase offset to be tested. An additional step for the success of the testing is represented by the definition of the type of test/measurement to execute and, as a consequence, the simulation to perform.

The *continuity* test was the first performed. This test function checks for continuity of the test signal paths and for short circuits. Normally, it is executed in all testflows since it gives first information about the integrity of the chip under test. Basically, it forces a test current into the chip and measures the voltage between the pin and vdd/ground. The test is passed if the measured voltage is in the range specified in the test function setup. A second executed test is called *IDDQ*. This test function measures the quiescent power supply current (IDDQ) of the chip when it is stimulated by an EVCD without events. In this case, it was provided an EVCD where the external clocks were kept constant, the reset signals enabled and the PLL switched-off. Then, the *FunctionalTest* was performed to analyze the behavior of the chip under specific input patterns. This test function examines the DUT following the timing set and performing a real time comparison between expected and received data (specified by the vector set). Through this test, the functionalities of the seven sub-systems were fully validated. Each sub-system of the testchip was evaluated under several operative frequencies and skew scenarios through frequency and clock phase sweep.

Moreover, the EVCDs were generated in order to stimulate every possible NoC state. In fact, each EVCD was composed by the following six transactions:

1. The tester0 performs 2 writes to the memory1 (memory slot 0 and 1). The flits cross both the switches and the NoC link before to reach the target memory. The data move from domain0 to domain1. In the synchronizer-based sub-systems, the source synchronous synchronizers are stimulated.

2. The tester0 performs 2 writes to the memory1 (memory slot 0 and 1). The flits cross both the switches and the NoC link before to reach the target memory. The data move from domain0 to domain1. In the synchronizer-based sub-systems, the source synchronous synchronizers are stimulated.

3. The tester1 performs 2 writes to the memory0 (memory slot 0 and 1). As before, the flits cross both the switches and the NoC link before to reach the target memory but in this case the data moves from domain1 to domain0.

4. The tester1 and the tester0 perform 2 writes to the memory0 (memory slot 2,3,4 and 5) at the same time. These transactions force congestion in the switch0. As a result, the flow control taking care of the interruption/resumption of the communication on the link was tested.

5. The tester1 and the tester0 perform 2 writes to the memory1 (memory slot 2,3,4 and 5) at the same time. In this case, the congestion is forced in the switch1. The `c` and `d` transactions test together the flow control of the NoC in both the directions.

6. The tester0 requires a read from the memory1. The tester packets move from domain0 to domain1 and the memory packets from domain1 to domain0.

7. The tester1 requires a read from the memory0. In this case, the tester flits move from domain1 to domain0 and the memory flits from domain0 to domain1.

To note that the memory locations are not overwritten by the transactions in order to allow the full evaluation of the test during the final memory scan-out. Finally, the *`OperatingCurrent`* test was performed. The operating current test measures the current supplied to the chip while it is undergoing a functional test. This test allows to estimate the power consumption as a function of the injected vector data files. In particular, the test function programs a sequencer

to execute the provided vectors in a loop until the operating current measurement is completed. In this case, a specific area to loop was indicated in the provided EVCDs. In particular, the power consumption of every sub-system was evaluated during idle time and under low and high traffic conditions. The idle time EVCDs was written to enable a sub-system at a time without injecting transactions. On the contrary, the high/low traffic EVCDs were composed of writes transaction (6 at a time) performed sequentially by the 2 testers into the memories of their opposite domains. The number of clock cycles to wait between 2 sequence of writes transactions was higher for low traffic than high traffic condition.

4.5 Test Results

As mentioned above, the first performed test was the *continuity* test. Since several testchip copies were printed (19 versions), this first test allowed to discard the chips with defects and to perform the further tests in the remaining reliable chips. Only 1 chip presented process defects in the design under test. The *functionaltest* was executed as a second test and it was performed in two steps. In the first step, the slow sub-systems were analyzed (*Asynch_Hybrid_Slow*, *Asynch_Loose_Slow* and *Synch_Slow*). Since the slow sub-systems are clocked from an external low-speed clock, the PLL was no longer required and it was switched-off. The goal of this test was to verify the functionalities of a sub-system at a time in each frequency and skew scenario.

The frequency sweep was performed increasing gradually the speed of the injected external clocks. The control input signal and the data input signal were injected in compliance with the target operating frequencies. The strobe setting at the output pins was performed similarly. The selected lower bound for the frequency sweep was 25MhZ. Clearly, the test could also be performed below this frequency but we did not expect relevant results under this threshold. On the contrary, the upper bound of the frequency sweep was not specified a priori since the test was also meant to determine the maximum operating frequency that the slow sub-systems were able to achieve. This maximum speed is associated not only with the inherent limitations of the test setup, but also with the limited driving capability of the I/O pins. To note that the slow sub-systems were synthesized at 100 MHz although the synthesis tool met the frequency constraints with a large slack.

Additionally, a skew sweep was performed for the *Asynch_Hybrid_Slow* and

Asynch_Loose_Slow at every operating frequency under test. In this case, the domain1s clock phase was gradually increased until 100% of the clock period was achieved. To note that the applied skew can be seen as positive skew in the `a` and `d` transactions (see the previous section), while on the contrary as negative skew for the `b` and `c` transactions. As a result, a functional test for a given clock phase passes when the NoC source synchronous link is able to absorb both the positive and the negative skew. This approach allows to reduce the final effort required to define the test specification and to perform the testing. Anyway, in order to perform a safe test, the skew was not only applied to the domain1 external clock but it was also applied to the control/data input signal and to the output pin strobe of the same domain.

As a result, the *Synch_Slow* sub-system passed every functional test in the range of 25Mhz and 265Mhz. Interestingly the sub-system was able to work at a frequency significantly higher than the frequency of synthesis. That can be due to two main reasons:

- A significant slack was reported after the 100Mhz synthesis. Then the failure of the sub-system is actually expected when the slack is completely absorbed.
- The designs were instantiated by means of the worst case standard cell library. The worst case library was probably assuming overly pessimistic conditions.

Concerning the synchronizer-based sub-systems, the skew and frequency sweep results of the *Asynch_Hybrid_Slow* and the *Asynch_Loose_Slow* sub-systems are depicted in Figure 4.12 and in Figure 4.13, respectively.

As demonstrated by Fig. 4.12 and Fig. 4.13, the two synchronizer-based slow sub-systems were able to pass the functional tests in the range of 25Mhz and 265Mhz as the *Synch_Slow* sub-system. Moreover, they well absorbed the applied skew tolerating 100% of the clock period offset in most of the frequency scenarios. In particular, the *Asynch_Loose_Slow* sub-system was able to tolerate every injected skew up to 265Mhz. On the contrary, the *Asynch_Hybrid_Slow* sub-system presented some failures at low frequency and high skew (but in a domain with poor practical relevance). Anyway, these latter failures do not find an easy justification in theory. In fact, although the *Asynch_Loose_Slow* sub-system is supposed to have relaxed intrinsic timing margins compared to the *Asynch_Hybrid_Slow* counterpart, the failures were expected to show up at high frequency rather than at slow frequency. Then, we

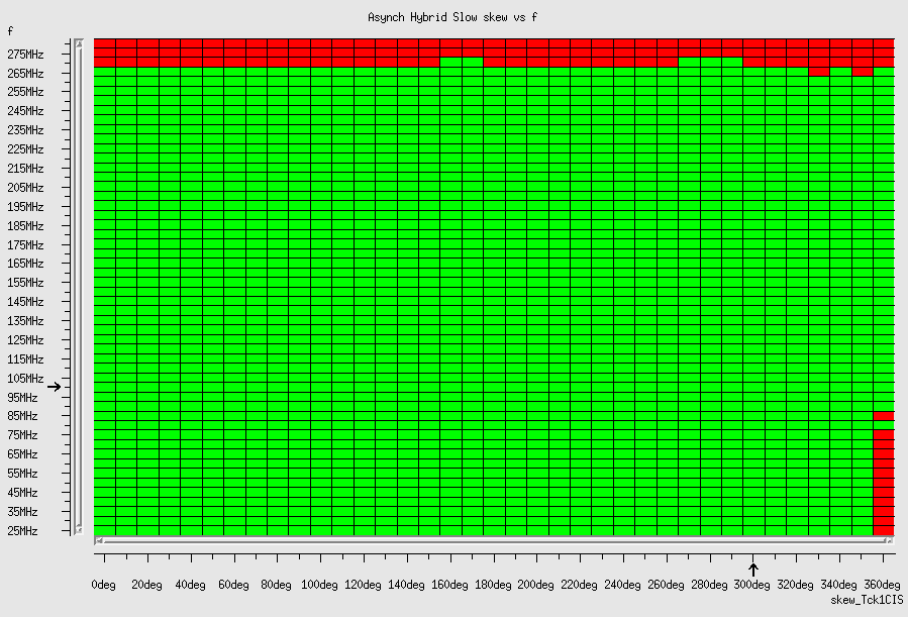


Figure 4.12: Frequency and skew sweep in the *Asynch_Hybrid_Slow* sub-system.

are induced to think that this low frequency failures are actually due to problems in the test equipment and/or specification. Moreover, a slow degradation of the skew tolerance was expected with increasing frequency. Since the sub-systems under test fail before showing the expected skew tolerance degradation and they fail all together (*Synch_Slow* sub-system included) at the same frequency, the max performance is probably dictated by the driving capability of the I/O pins or by a critical path violation. On the contrary, a violation of the timing margins in the source synchronous interfaces should be excluded.

In the second part of the *functionaltest*, the fast sub-systems were analyzed (*Asynch_Hybrid_Fast*, *Asynch_Loose_Fast*, *Asynch_FIFO* and *Synch_Fast*). The goal of this second testing part was still to verify the functionalities of the sub-systems (a sub-system at a time) since in this case the sub-systems were clocked by the PLL. As a result, the frequency and the skew sweep should be performed according with the limitations dictated by the PLL specification. In particular, the PLL provides the following 4 clock phases: 0, 90, 180 and 270 (i.e. the highest provided skew corresponds with the 75% of the clock cycle). Moreover, the PLL clock frequency is generated according with the following formula:

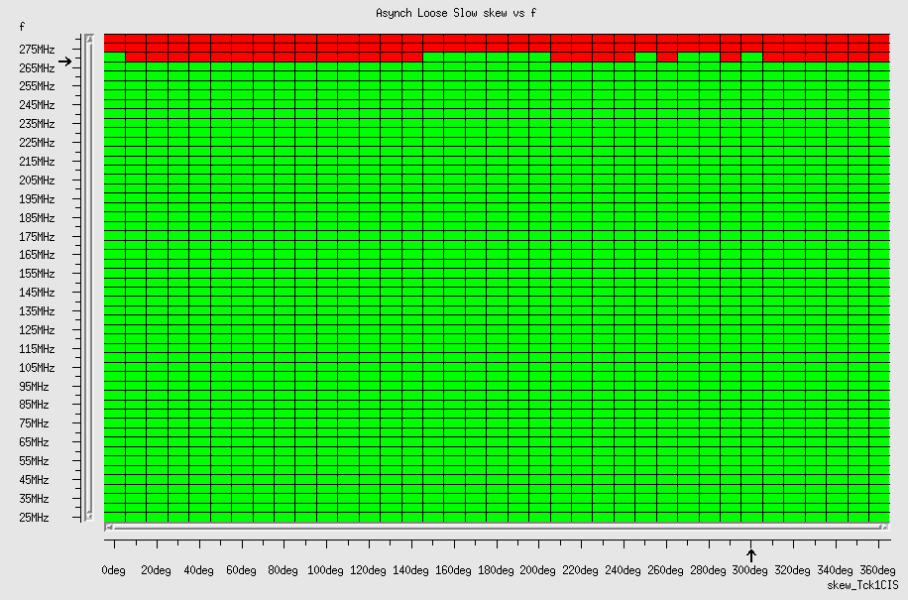


Figure 4.13: Frequency and skew sweep in the *Asynch_Loose_Slow* sub-system.

$$PLLfreq = [PLLexternal_clock * (n + 1)]/2 \tag{4.1}$$

To note that the *n* parameter can be set through dedicated external pins. Following the reported formula, a sweep of the *PLLexternalclock* frequency could directly generate the desired sweep of the PLL clock. Anyway, the PLL has been designed to properly work with an external clock of around 25Mhz. As a consequence, the experiments were performed with a fixed external clock (25Mhz) and the *n* parameter was used to modify the PLL frequency. Theoretically, 8 bits are available to set the *n* parameter but three of these bits were hard-wired at a low value in order to meet the maximum pin number requirements (i.e., 5 input pins are still available). As a result, the PLL can run at the maximum frequency of 400Mhz (supposing n=00011111 and PLLexternal_clock=25Mhz).

Unluckily, the frequency and the skew sweep in the fast sub-systems could not exploit the same timing set and vector data file since the frequency and the clock phase had to be set statically by driving the dedicated external pins. On the contrary, the clock phase and the clock frequency in the slow sub-systems were modified by means of the same EVCDs through straightforward steps. In fact, the clock phase in the slow sub-systems was simply set by delaying the

injection of the clock and control/data input signals while the clock frequency was modified by scaling the timing of the vector data files.

Then, a few test frequencies were selected to perform the functional test of the fast sub-systems. The selected PLL frequencies for test were 200Mhz and 400Mhz in the 4 available clock phase variants. It is useful to recall that the input pins were supposed to inject the data/control signals with a frequency of 25Mhz (going through the synchronizer) and the fast sub-systems were synthesized at 500Mhz. The results with the test frequencies were as follows:

- The *Synch_Fast* functional test passed with the PLL frequency set at 200Mhz and 400Mhz with all the chips that previously were able to pass the continuity test. To note that only 1 chip out of 19 chips was discarded during the continuity test.

- The *Asynch_Loose_Fast* functional test passed with the PLL frequency set at 200Mhz in each of the 4 skew scenarios for the 18 chips. The same sub-system operating at a frequency of 400Mhz passed the test with 17 chips. The sub-system was not able to work at 400Mhz in only 1 chip previously able to pass the continuity test.

- The *Asynch_Hybrid_Fast* functional test passed with the PLL frequency set at 200Mhz in each of the 4 skew scenarios for the 18 chips. The same sub-system operating at a frequency of 400Mhz passed the test with 12 chips. The sub-system was not able to work at 400Mhz in 6 chips previously able to pass the continuity test. Anyway, each of these 6 chips passed the functional test with a phase offset of 90, 180 and 270, on the contrary, they did not pass the test with 0 of skew. The 0 of skew represents the fully synchronous scenario and intuitively the source synchronous interfaces should provide the best timing margins in this condition. As future work, further explorations of the timing set specification should be performed to justify this result, which might be most probably ascribed to the inexperience with the testing equipment and/or to the complex testing procedure rather than to a true problem in the technology.

- The *Asynch_FIFO* functional test passed with the PLL frequency set at 200Mhz in each of the 4 skew scenarios for the 18 chips. To note that this sub-system should not only absorb the skew into the source synchronous link but also synchronize data from 25Mhz to 200Mhz and vice versa. The same sub-system operating at a frequency of 400Mhz

passed the test with only 11 chips out of 18. The output of the remaining 7 chips was not stable. In fact, although the sampled output was in most of the cases matching the expected one, the correct output was still not guaranteed when repeating the test several times. Moreover, the failing clock phase was usually only one at a time and, surprisingly, it was randomly changing although the test was performed on the same chip. Interestingly, the failing tests presented few unstable output bits always located in the same position of the output vector. On the contrary, when the *Asynch_Hybrid_Fast* and the *Asynch_Loose_Fast* sub-systems were failing the output vectors presented the anomalous bits spread homogeneously all over the output vector. As a result, these information led us to suppose that specific patterns bring few bits of the *Asynch_FIFO* sub-system to a metastability condition that results into unpredictable results.

At a better analysis, the *Asynch_FIFO* sub-system is the only one that integrates brute force synchronizers, composed of a sequence of 2 flip-flops, where the output bit can fall into a metastability condition. The probability of solving the metastability in these bits follows the MTTF law and tightly depends on the technology node and the operative frequencies. As reported in literature (see [88]), the failure probability is not negligible anymore at the scaling of the technology node and with the increase of the frequencies. Then, the MTTF law applied to the dual-clock FIFO brute force synchronizers provides a valid explanation for the instability of the few output bits in the *Asynch_FIFO* sub-system at 400Mhz. To note that the brute force synchronizers are stimulated only during the congestion conditions of the switches (i.e. when assertion of the STALL/GO flow control is required). The congestion of the switches is achieved in only two of the six transactions and it always affects the switches during the same clock cycles (in compliance with the EVCD patterns).

Summing up, Fig.4.14 illustrates on the y-axis the percentage of chips (out of the 18 ones which passed the continuity test) that proves functional in each test case (identified by the operating frequency, skew and architecture variant parameters).

The success percentage, for a first time implementation, is quite high. We start observing some issues at 400 MHz, as we approach the target synthesis frequency of 500 MHz. However, the failure pattern is such that improving upon such results should not be an issue in future work. In fact, the zero skew

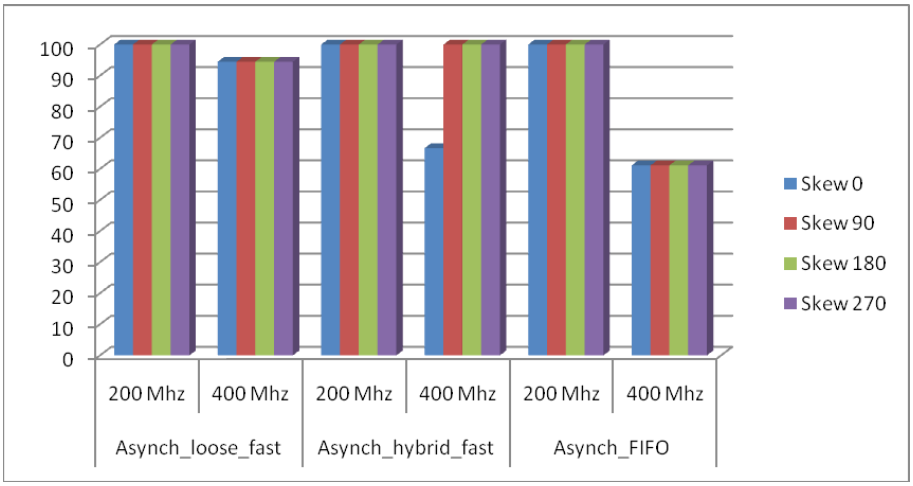


Figure 4.14: Percentage of working chips in each test case.

failure for the Asynch_hybrid_fast design does not seem to be related to the robustness of the interface. Also, failures at 400 MHz for Asynch_loose_fast design are independent of the applied skew, thus denoting a different problem than the robustness of the mesochronous synchronizers. Finally, the failures for the Asynch_FIFO design could be easily solved by cascading more flip flops in the brute-force synchronizers used inside dual-clock FIFOs.

We then performed some power measurements on the test structures. Fig.4.15 shows the relative comparison between power of the designs under test clocked by the PLL. Taking a look at the plot, it is clear that it was not possible to inject significant traffic due to the constraints of the measurement setup. In fact, in order to achieve precise dynamic current measurements, it is essential that the pattern has a constant current profile and that it can be repeated in a loop to make sure the pattern is running as long as the current measurement is active (and there is no way of predicting how long the measurement will be. The tool uses auto-ranging, i.e., it executes several measurements to find the minimum measurement range where no overflow occurs.) Unfortunately, the NoC structures under test consist of a quite time consuming programming step of a much shorter execution phase, therefore most likely the programming step is the prevailing contribution to measured power figures.

Idle power is taken by selectively removing the reset but without giving any input traffic to the platform itself. Clearly, the power difference between the architectures under test is not significant at all: the synchronizer-based solutions tend to add 0.38mW overhead to the power of the synchronous sub-system

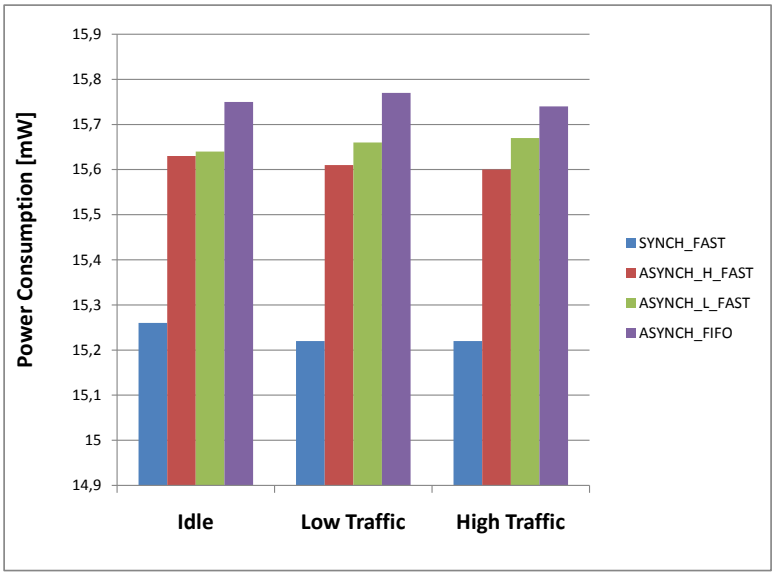


Figure 4.15: Relative power comparison.

(slightly more for the loosely coupled solution), which grows to 0.5mW for the Asynch_FIFO design. The overhead in percentage terms would be 2.8% and 3.7% with respect to the synchronous sub-system for the mesochronous and for the dual-clock FIFO-based ones, respectively. Ultimately, we feel the key take away of these results is that synchronizer-based GALS technology comes at a marginal power overhead.

The above power results indicate just a trend and should be handled very carefully for the following reasons:

- The error margin of the test equipment is about 0.3mA. The current absorbed by the NoC structures differs by an amount which is close to the uncertainty of the measurement tool. In addition, a single measurement of the current was taken at regime. However, by sampling the current multiple times we were getting a further uncertainty of around 0.5mW. In spite of all this, we were able to observe a clear relative trend between the architectures under test which is orthogonal to the 18 working chips. Also measurements for slow designs confirm the same power trends.
- Injecting heavy traffic into the test structures was virtually impossible, given the small size of the structures themselves and the large time needed to program the structures, prevailing with respect to the actual

packet exchange time. For this reason, the most accurate power results certainly concern the idle power tests.

- The Asynch_FIFO design has memories, tester and JTAG interfaces working at 25 MHz instead of 200 MHz like in the other fast platforms. However, this does not seem to favor this design much, from a power viewpoint.

4.6 Conclusions

The Moonrake chip was as illustrated successfully tested. It was the first implementation of synchronizer-based GALS NoC technology in 40nm CMOS process. The objective was to validate the synchronization technology, containing key innovations such as the tight coupling of the synchronizer with the switch input buffer, and, implicitly the design technology supporting it. The testchip was structured in a modular way, thus amortizing the manyfold risks associated with testchip fabrication: in essence, porting a new synchronization technology to a new manufacturing process, and then using new features of the test equipment to test such an advanced chip implementation.

Overall, results are very promising. NoC test structures getting the clock from the external world provided an excellent result: frequencies from 25 to 265 MHz were swept, while at the same time varying the clock phase offset from 0 to 360 degrees. This means that the synchronization mechanisms, considered by themselves, can be ported to the 40nm technology and prove functional in such an environment. When it comes to the designs synthesized for and operating at a higher frequency, then a set of issues comes into play: not only higher stress of the technology platform, but also larger complexity of the test procedure. 91% of the performed tests completed successfully, and the failures seem more related to testing or *systematic* issues in the technology platform rather than to the robustness of the synchronization interfaces. Also, an interesting experimental phenomenon was observed: as technology scales down, the resolution time constant of brute force synchronizers degrades, hence calling for a larger number of cascaded flops to resolve metastability. This is of special interest to many dual-clock FIFO realizations. Finally, our power measurements indicate that the power overhead when moving from a fully synchronous test system to a mesochronous one or even to a multi-synchronous one is marginal. To conclude, we find that this testchip experience validates the feasibility and effectiveness of the developed GALS NoC concept in nano-scaled technology sub-systems bridging the final gap to actual silicon implementation.

5

Design Space Exploration for Redundancy-Aware NoC Testing

THIS chapter provides an exploration of testing strategies for NoC-based systems. In particular, it presents four scalable built-in self-test and self-diagnosis infrastructures making efficient use of NoC structural redundancy for testing and diagnosis purposes through the use of a cooperative testing framework. While the first three strategies are applied to fully synchronous NoCs, the last one tackles the testing of multisynchronous systems. The optimization of the stuck-at-fault coverage and the minimization of area and latency of the testing and diagnosis strategy will be the guideline of the chapter. The proposed testing strategies will be compared between each others and the most promising of them will be thus exploited as the reference testing and diagnosis strategy in the final next-generation system presented in the last chapters of this thesis.

5.1 Methodology and Taxonomy

Embedded systems have been shifting to multi-core solutions (Multiprocessor System-on-Chips; MPSoCs). A clear example of high-end MPSoCs are the products offered by Tilera [24] where multi-core chips provide support to a wide range of computing applications, including high-end digital multimedia, advanced networking, wireless infrastructure and cloud computing.

Main microprocessor manufacturers are also shifting to chip multiprocessors (CMPs) for their latest products. In CMPs many cores are put together in the same chip and, as technology advances, more cores are being included. Recently, Intel announced a chip with 48 cores, under the Tera-scale Computing Research Program [145]. Previously, Intel also developed a chip proto-

type [146] that included 80 cores (known as TeraFlops Research chip).

Current trends indicate that Multi-core architectures will be used in most application domains with energy efficiency requirements exceeding 10GOPS/Watt. However, aggressive CMOS scaling accelerates transistor and interconnect wearout, resulting in shorter and less predictable lifespans for CMPs and MP-SoCs [149]. It has been predicted that future designs will consist of hundreds of billions of transistors, with upwards of 10% of them being defective due to wearout and process variation [147]. Consequently, in order to support the current technology trends we must develop solutions to design reliable systems from unreliable components, managing both design complexity and process uncertainty [148].

Network on Chip is being increasingly investigated by researchers and designers to address the issues of interconnect complexity in both CMPs and MP-SoCs [150]. The reliability of NoC designs is threatened by transistor wearout in aggressively scaled technology nodes. Wear-out mechanisms, such as oxide breakdown and electromigration, become more prominent in these nodes as oxides and wires are thinned to their physical limits. These breakdown mechanisms occur over time, so traditional post burn-in testing will not capture them.

As a result, an important requirement for this purpose is the efficient testability of candidate NoC architectures. This property is very challenging due to the distributed nature of NoCs and to the difficult controllability and observability of its internal components. When we also consider the pin count limitations of current chips, we derive that NoCs will be most probably tested in the future via built-in self-testing (BIST) strategies. These latter are also capable of tackling wearout failures since they are suitable both for production and for lifetime testing.

Finally, the chapter focuses on Built-In-Self-Testing techniques. All the testing logic of these latter techniques is encapsulated inside the chip, working autonomously from the exterior of the chip. In BIST-based systems, the generation of the test cases and the check of the results is performed inside the chip, instead of being done in an external device.

In this chapter, we take the challenge of performing a design space exploration for BIST-based testing for NoCs. In order to perform such exploration, we have identified the main degrees of freedom extended to a designer implementing a testing framework for NoCs. First of all, the degree of controllability exploited by the testing framework plays a crucial role since it denotes the ability to move the design under test (DUT) around in its entire configuration space. Secondly, the kind of test patterns adopted to stimulate the DUT has a rele-

vant impact on the performance of the framework. Finally, the aforementioned degrees of freedom are constrained by the target performance that the testing framework is required to achieve. In particular, the framework performance is driven by three main reference metrics: stuck-at-fault coverage, area overhead and testing time. As a result, this manuscript wants to explore the trade-off exposed by these latter metrics at the swapping of network controllability and test patterns kind.

In general, a network is divided in sub-blocks and these latter represent the elementary blocks around the testing framework is built. Thus the testing framework controls all the inputs of each sub-block. Finally the degree of controllability is dictated by the granularity of the network sub-blocks. While high controllability requires to divide the network in several micro blocks on the contrary low controllability comes with few macro blocks. On one hand, when controllability of the network increases then stuck-at-fault coverage arises and testing time decreases. On the other hand, a higher controllability comes with a higher area overhead due to the inherent complexity of the BIST infrastructure. Our purpose is to analyze different controllability scenarios in order to identify the ones better performing.

In addition, different kinds of test patterns can be generated and combined with testing frameworks at each stage of controllability. We identify the following main groups of test patterns:

1. *Deterministic-Handcrafted*. Test patterns are totally created by hand exploiting the knowledge of the DUT. The underlining idea consists in bringing the DUT into all the possible functional states.

2. *Deterministic-Algorithmic*. Algorithmic deterministic test patterns are generated by automatic tools (like Tetramax). Tools produce a set of heuristic test patterns by taking as an input the DUT.

3. *Pseudo-Random*. Linear-Feedback-Shift-Registers (LFSR) generate pseudo-random test patterns. Such patterns randomly stimulate the DUT until the expected performance is achieved.

At a first glance, a loose control seems appealing for a testing framework. In fact, this latter comes with light and low intrusive frameworks. However, a loose controllability does not typically achieve acceptable stuck-at-fault coverage and testing time regardless of the adopted test patterns. Deterministic-handcrafted test patterns are applied at the *network-boundaries* in [18] anyway the time complexity of the test configurations is square with respect to

the rank of the NoC matrix and the control path achieves a low coverage. Differently, deterministic-algorithmic or pseudo-random test patterns can be adopted. However the automatic tools are not able to converge to a set of algorithmic deterministic test patterns when the whole network is given as input due to the complexity of the design. Similarly, pseudo-random test patterns injected at the boundaries of the network do not achieve relevant results in terms of testing time and stuck-at-time coverage. Even when the controllability of the network is increased by means of traditional full-scans, the testing framework incurs an hardly affordable area overhead (like in [47]).

Such considerations brought us to start our testing framework exploration in scenarios with higher degree of controllability. In particular, a first set of testing frameworks was developed for each kind of test patterns by controlling switch sub-blocks. Each sub-block is composed by a chain of two or more switch modules. In particular we consider the following switch modules: the input and output buffer, the switch-to-switch link, the arbiter, the multiplexers of the crossbar and the routing mechanism. The testing frameworks stimulate the boundaries of the chain. Once we have evaluated the first set of results then we moved to a higher or lower degree of controllability in accord with the achieved stuck-at-fault coverage, area overhead and testing time. The key idea is to migrate to a higher controllability solution whenever coverage or testing time are inadequate while to migrate to a lower degree of controllability whenever the area overhead skyrockets.

The design space exploration is completed by an analysis of the diagnosis structure alternatives. The diagnosis structure represents the infrastructure in charge of check the correctness of test responses. In this case, three mainly diagnosis options are reported in literature:

1. *Signature analysis* is widely adopted in digital systems to compress multiple bit streams. This latter is usually implemented by means of a Multiple-Input-Signature-Register (MISR).

2. *Comparators trees* can be adopted to perform an effective diagnosis able to exploit the inherent structural redundancy of the design under test. Once stimulated by the same test patterns, replicas of the same modules outputs the same results if there is no fault. Thus test responses of multiple replicas feed comparator trees for the diagnosis.

3. *ROM registers* store the expected test responses which are compared with the output of the DUTs.

As a matter of fact, the choose of the diagnosis structure does not represent a degree of freedom for testing framework but it is rather dictated by the design under test and the adopted test patterns. Indeed MISR must be combined with long pseudo-random patterns sequences to avoid aliasing effects. Thus signature analysis is usually coupled with LFSR solutions since it results unreliable when alternatives test patterns are adopted. Concerning ROM registers, they are inefficient in redundant environments. In fact, their information can be replaced by test responses generated by in-situ replicas of the DUT. Finally, the manuscript follows a clear strategy as regards the choose of the diagnosis structure. Since the NoC exposes a high degree of redundancy then ROM registers are discarded in favor of comparators trees able to effectively exploit the NoC environment. Anyway, comparators trees introduce interdependencies between DUTs replicas complicating the failure analysis with respect to MISRs. As a result, MISRs are preferred to comparator trees when the testing framework is based on pseudo-random patterns. On the contrary, comparators trees are still adopted in the other test patterns scenarios.

Overall, the manuscript provides an exploration of built-in testing strategies customized for NoCs comparing them under an area, coverage, routing delay and latency point of view. The test of NoC switches will occur in parallel, thus making test application time independent of network size. Notice that communication channels between switches are tested as a part of the switch testing framework from the switches themselves in most of the proposed techniques.

A key principle of our approaches consists of exploiting the inherent structural redundancy provided by NoCs. Each switch is comprised of input ports, output ports, arbiters and FIFOs that are duplicated for each channel. This feature is used to develop very effective test strategies which consists of testing multiple identical blocks in parallel and of cutting down on the number of test pattern generators. This is done both at the abstraction level of the switch micro-architecture (e.g., testing of the output port arbiters in parallel) and of the NoC architecture (i.e., testing of all NoC switches in parallel). The inherent parallelism of our BIST procedure makes our testing infrastructure highly scalable and best suited for large network sizes.

Finally our BIST procedures are suitable both for production and for lifetime testing, and are complemented by a built-in self-diagnosis logic distributed throughout the network architecture able to pinpoint the location of detected faults in each switch. This diagnosis outcome matches the reconfigurability requirements of logic-based distributed routing.

The rest of the chapter is organized as follow. Section 5.2 presents the base-

line switch architecture which will be the baseline for the testing frameworks. Section 5.3 takes on the challenge pointed by [26] of exploiting architecture behavior knowledge to come up with a set of customized test patterns for NoC components. Thus we present a deterministic test patterns-based BIST/BISD framework with low latency and high coverage while at the same time detecting TPG faults. Section 5.4 presents a strategy based on a scan chain mechanism and algorithmic deterministic test patterns. With respect to the conventional scan-based approaches, we built-in the test pattern generators and the diagnosis modules and we reduce the area overhead. In Section 5.5, we design a testing based on pseudo-random patterns where we cut down on the test application time and we provide efficient testing of the control path. Section 5.6 compares between each others the proposed testing frameworks in terms of area, stuck-at-fault coverage and latency. In Section 5.7, we propose one of the first built-in self-testing and diagnosis framework for Globally-Asynchronous-Locally-Synchronous Network-on-Chip based on an asynchronous handshaking. Finally, the main conclusions are presented in Section 5.8.

5.2 Target Architecture

Performance of testing frameworks in literature are hardly comparable between each others since whenever numbers are available, network under tests are very different. On the contrary, we want to perform a design space exploration in a homogeneous setting in order to build up a fair comparison between testing techniques. As a result, we use the xpipesLite switch architecture [110] with a 5×5 configuration to prove viability of all our solutions.

The baseline switch architecture illustrated in Fig.5.1 represents a variant of the architecture presented in section 3.4. As this latter switch, it implements both input and output buffering and relies on wormhole switching. The crossing latency is 1 cycle in the link and 1 cycle in the switch itself. Flit width assumed in this thesis is 32 bits, but can be easily varied. Without lack of generality, the size of the output buffers is 6 flits, while it is 2 flits for the input buffers. This switch relies on a stall/go flow control protocol. It requires two control wires: one going forward and flagging data availability (”valid”) and one going backward and signaling either a condition of buffer filled (”stall”) or of buffer free (”go”).

The switch architecture is extremely modular and exposes a large structural redundancy, i.e., a port-arbiter, a crossbar multiplexer and an output buffer are instantiated for each output port, while a routing module is cascaded to the

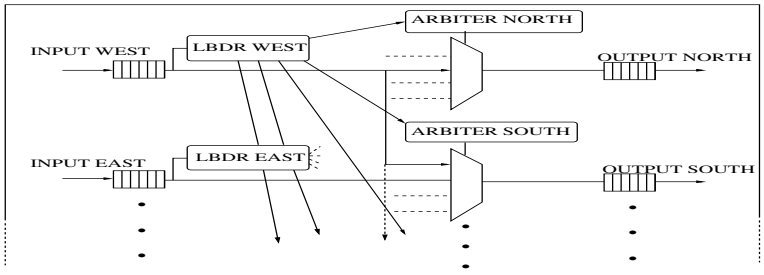


Figure 5.1: Modular structure of the baseline switch architecture. Not all connections are showed.

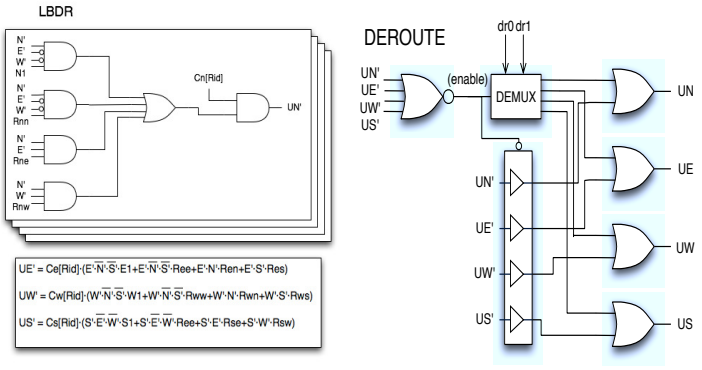
buffer stage of each input port. This common feature to all switch architectures will be intensively exploited in this work.

On the contrary of the switch architecture presented in section 3.4 based on source-based routing, the target switch of this chapter implements distributed routing. We implement distributed routing by means of a route selection logic located at each input port. Forwarding tables are usually adopted for this purpose, although they feature poor area and delay scalability with network size [151]. The possibility to implement logic-based distributed routing (LBDR) while retaining the flexibility of forwarding tables has been recently demonstrated in [152]. In practice, LBDR consists of a selection logic of the target switch output port relying on a few switch-specific configuration bits (namely routing R_{xy}, connectivity C_z and deroute bits dr_t). The number of these bits (14 in this case) is orders of magnitude less than the size of a forwarding table, yet makes the routing mechanism reconfigurable.

The core of LBDR logic is illustrated in Fig.5.2(a), illustrating the conditions that select the output port north UN' for routing. The pre-processed direction of packet destination $N'/S'/W'/E'$ is an input together with the routing and the connectivity bits. In some cases (see [152] for details), deroutes are needed to properly route packets, and the associated logic is reported in Fig.5.2(b).

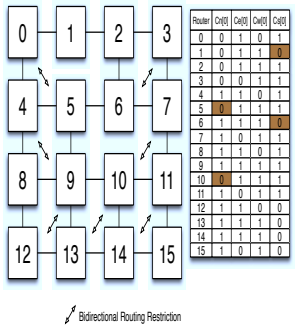
LBDR supports the most widely used algorithms for irregular topologies and can be used on a 2D mesh as well as on roughly 60% of the irregular topologies derived from a 2D mesh, like in Fig.5.2(c). Irregularity of the connectivity pattern can be an effect of manufacturing or wearout faults, but also of power management or thermal control decisions or of virtualization strategies. Switch configuration bits need to be updated whenever the topology evolves from one connectivity pattern to another (e.g., when a fault is detected).

Our testing and diagnosis frameworks have been all conceived to enable a net-



(a) Core logic for route computation

(b) Deroute logic



Router	Cn[0]	Ce[0]	Cw[0]	Cs[0]
0	0	1	0	1
1	0	1	1	0
2	0	1	1	1
3	0	0	1	1
4	1	1	0	1
5	0	1	1	1
6	1	1	1	0
7	1	0	1	1
8	1	1	0	1
9	1	1	1	1
10	0	1	1	1
11	1	0	1	1
12	1	1	0	0
13	1	1	1	0
14	1	1	1	0
15	1	0	1	0

(c) Connectivity bit setting because of failed links.

Figure 5.2: LBDR logic and requirements on the diagnosis outcome.

work reconfiguration strategy leveraging the cost-effective flexibility offered by the LBDR routing mechanism. An algorithm is reported in [152] for computation of the switch configuration bits given the topology connectivity pattern. As an example, updated connectivity bits are illustrated in Fig.5.2(c). This algorithm might be executed by a centralized NoC manager and in practice needs the list of failed links to recompute the configuration bits for correct routing with the available communication resources. Failure of a switch input or output port can be viewed as the failure of the connected link. Our diagnosis strategies will therefore target this requirement and will provide an indication of whether input and output ports of a switch are operational.

5.3 Testing framework based on handcrafted deterministic test patterns

This section proposes a built-in self-test/self-diagnosis procedure at start-up of an on-chip network based on deterministic test patterns. Concurrent BIST operations are carried out after reset at each switch, thus resulting in scalable test application time with network size. The key principle consists of exploiting the inherent structural redundancy of the NoC architecture in a cooperative way, thus detecting faults in test pattern generators too.

Four main features differentiate the testing framework proposed in this section from most previous work. First, we take on the challenge of generating deterministic test vectors on-chip at a limited area overhead. At the same time, this enables us to report much shorter test application times than typical pseudo-random testing frameworks and larger fault coverage in the control path than most functional testing frameworks for NoCs. Second, we account for the tedious problem of faults affecting test pattern generators (TPGs) and provide large coverage for them. This is done without implementing more hardware redundancy but fully exploiting the existing one by means of a cooperative testing framework among switches. Third, our testing framework targets double and triple stuck-at faults from the ground up, and not as an afterthought. Fourth, our framework is not limited to regular 2D meshes, but can be applied to a much wider range of network topologies.

As a result, the coverage for single stuck-at faults closely tracks 100% in both the control and data path of the network. This latter is achieved by means of deterministic test patterns handcrafted for the specific block under test by exploiting knowledge of the architecture behavior. Finally testing of stuck-at faults can be performed in less than 1200 cycles regardless of their size, with an hardware overhead of less than 26%.

5.3.1 The Testing Strategy

The key idea of our BIST/BISD framework consists of exploiting the inherent structural redundancy of an on-chip network. We opt for testing the NoC switches in parallel, thus making test application time independent of network size. Communication channels between switches are tested as a part of the switch testing framework.

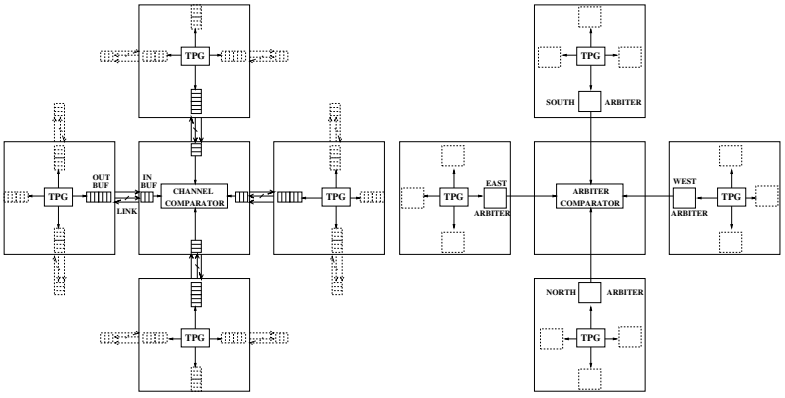
Each switch can in turn test its manyfold internal instances of the same sub-blocks (crossbar muxes, communication channels, port arbiters, routing mod-

ules) concurrently. In fact, all the instances are assumed to be identical, therefore they should output the same results if there is no fault. As a consequence, the test responses from these instances are fed to a comparator tree. This makes the successive diagnosis much easier. There is a unique test pattern generator (TPG) for all the instances of the same block, thus cutting down on the number of TPGs. Although the principle is similar to what has been proposed in [45, 46, 104], there is a fundamental difference. If the TPG of a set of block instances is affected by a fault, then the comparison logic will not be able to capture this since all instances provide the same wrong response. To avoid this, a cooperative framework is devised, such that each switch tests the block instances of its neighboring switches.

As an example, a switch tests the incoming communication channels from its north/south/west/east neighbors (i.e., it feeds their test responses to its local comparator tree), thus checking the responses to distinct instances of the same TPG. This way, a non-null coverage of TPG faults becomes feasible. Fig.5.3(a) clearly illustrates the cooperative testing framework for communication channels and the need for a single TPG instance per switch to feed test patterns to all of its output ports. Faults in the TPG, in the output buffer, in the link and in the input buffer will be revealed in the downstream switch. Each switch ends up testing its input links, while its output links will be tested by their respective downstream switches.

The same principle can be applied for the testing of switch internal block instances associated with each output port: crossbar muxes and output port arbiters. Fig.5.3(b) shows the case of port arbiters. The main requirement for testing these instances is that the communication channels bringing test responses to the comparators in the downstream switches are working correctly. Clearly, testing these modules can only occur after communication channels have been tested. Therefore, the procedures in Fig.5.3(a) and Fig.5.3(b) occur sequentially in time. Should one communication channel result defective, this would not be a problem, since it would not make any sense to test and use a port arbiter when the corresponding port is not operational. Crossbar multiplexers associated with each output port are tested in the same way and are hereafter not illustrated in Fig.5.25 for lack of space.

Finally, the methodology can be extended to test block instances associated with each switch input port with some modifications. This is the case of the LBDR routing block. The key idea to preserve the benefits of cooperative and concurrent testing is to carry test patterns rather than test responses over the communication channels to neighboring switches, where the LBDR instances



(a) Testing communication channels.

(b) Testing output port arbiters.

(c) Testing LBDR routing logic.

Figure 5.3: The cooperative and concurrent testing framework saving TPG instances and covering their faults.

are stimulated and their responses compared (see Fig.5.3(c)). If the channel is not working properly, than testing and use of the downstream routing block is useless, since it is associated with an input port which will not be used.

A BIST engine is embedded into each switch and regulates the testing procedure. This latter is in fact split into four phases in time:

- testing of communication channels.
- testing of the crossbar.
- testing of the arbiters.

- testing of the LBDR routing blocks.

The serial execution of test phases for the switch internal components is dictated primarily by the limited flit width, constraining the amount of test patterns that can be transmitted at the same time over the communication channel, and also by the limited availability of comparators, although in our case the former effect comes into play first. As the flit width increases, then we can perform more testing operations in parallel, starting from those components that have a limited amount of primary input/outputs (e.g., the arbiter with the LBDR).

A fundamental difference with respect to a lot of previous work is that we do not rely on pseudo-random testing (like in [23]), which usually gives rise to large testing times. We use deterministic test patterns instead, which are handcrafted for the specific block under test by exploiting knowledge of the architecture behavior. This way, the reduced number of test patterns enables the serialization of test phases without making test application time skyrocket (see section 5.3.7).

On a cycle by cycle basis, comparator outputs are fed to a diagnosis logic which identifies where exactly the fault occurred. In our diagnosis framework, each switch checks whether test responses from its input ports are correct or not. As a consequence, the outcome of the diagnosis is coded in only 5 bits, one for each input port of the current switch (they would be of course doubled if a two-rail code is implemented to protect them against stuck-at faults). A '1' indicates that the port is faulty. In practice, the fault may be located either in the input buffer or in the LBDR module, in the connected communication link or even in the output buffer and associated port arbiter and crossbar multiplexer of the upstream switch. This further level of detail is not needed, since in any case the meaning is that the link is unusable, and this is enough for a global controller to recompute the reconfiguration bits for the LBDR mechanism.

In the final implementation, other 5 bits will be needed to code the diagnosis outcome because of practical implementation issues, as discussed in section 5.3.2.

Common to most current NoC testing frameworks, the underlying assumption for correct operation of our BIST/BISD infrastructure is that the reset signal can be synchronously deasserted in all switches of the network at the same time.

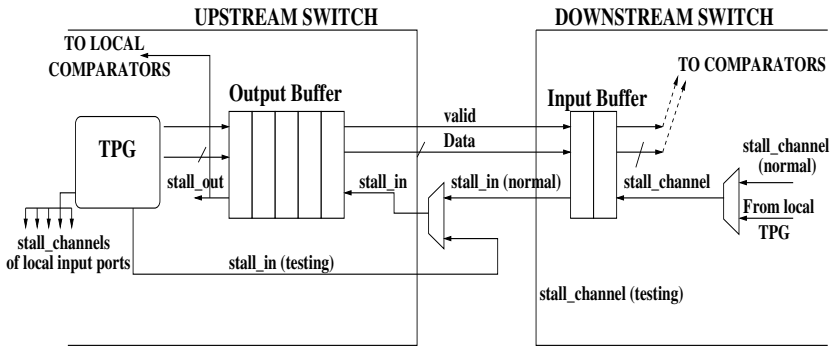


Figure 5.4: Practical implementation of communication channel testing.

5.3.2 Testing Communication Channels

Communication channels include input/output buffers and their intermediate links, as illustrated in Fig.5.4: all these elements are jointly tested by means of a single TPG and the test patterns are handcrafted for them based on knowledge of their behavior.

Our approach in this direction was to expand the finite state machine (FSM) of the device under test (DUT) into all its possible states. Therefore, we have defined a sequential test pattern that drives the FSM to each of its states. In this way, we can ensure that if the FSM reaches the expected state for all the test patterns there are no faults inside the DUT. As an example, the FSM of the buffers defines that if the Stall signal is asserted and the buffer receives a set of valid flits, the buffer has to store the flits that it receives until it becomes full. One test pattern to check this behavior would fill up the buffer by asserting the Stall signal, and would in the end check whether the output buffer correctly asserts the Full signal. The datapath is obviously much easier to test by means of only few test patterns.

From an implementation viewpoint, there are several practical issues. On one hand, we had to make the stall input of the output buffer directly controllable to the TPG to raise its stuck-at fault coverage to almost 100% (see Fig.5.4).

On the other hand, the *stall_channel* signal of the input buffer, which lies in the downstream switch, should be driven by the TPG as well. This would require an additional wire in the switch-to-switch link. A similar concern is that the *stall_out* signal from the output buffer should be brought to the comparators in the downstream switch, again requiring an additional wire in the link.

To avoid the extra wires, we opted for the solution in Fig.5.4: *stall_channel* is

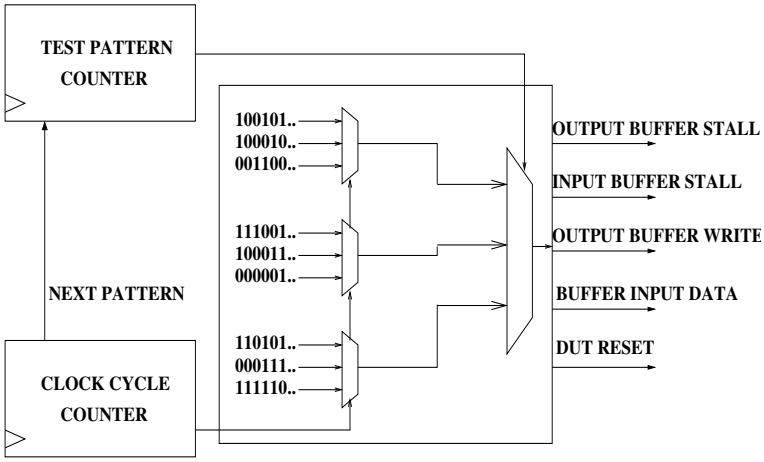


Figure 5.5: TPG for communication channels.

driven by the TPG of the downstream switch, while *stall_out* is brought to the comparator tree in the upstream switch. From the testing viewpoint nothing changes, since all channel TPGs inject the same patterns synchronously, and so do the comparators. The only difference lies in the fault coverage of TPG faults, which is likely to be decreased a bit. In fact, those (upstream) TPG faults that can be detected by only monitoring *stall_out* will not be detected, since all the *stall_out* signals brought to the local comparators will be driven by the same TPG. Similarly, some faults in the (downstream) TPG will not be detected, since the comparators compare responses to *stall_channel* signals generated by the same faulty TPG: the responses will look like the same. These implementation variants, needed to adapt the conceptual testing scheme to the constraints of the real implementation, will be proven in section 5.3.7 to only marginally decrease fault coverage of the TPGs, while leaving fault coverage for the communication channel obviously unaffected.

The only major implication is that the fault detection framework becomes even more collaborative: some (very few) faults in the channel and/or TPGs are now detected in the upstream switch comparators instead of the downstream ones. Therefore, other 5 additional diagnosis bits are needed, flagging a fault in the output port of a switch. The global controller will combine this (OR operation) with the faults detected at the input port of the downstream switch to get the complete indication of a fault across the entire channel.

5.3.3 TPG for Communication Channels

A test pattern can be easily generated in hardware by using a clock cycle counter and some logic to generate the values of the input signals for the DUT. In order to extend this approach to a TPG able to generate all the test patterns for a given DUT, we can include an additional counter. This latter will indicate the current test pattern within the test sequence. Figure 5.5 depicts the resulting conceptual scheme for the channel TPG. The actual gate-level implementation depends on the logic synthesis tool and on the synthesis constraints. The two counters act as a FSM driving the control signals of two levels of multiplexing: the first one selects the current test pattern, while the second one selects the current clock cycle and associated input vector for the buffer.

It is however possible to easily compact the combinational logic, because there are a lot of test patterns that include other test patterns. For instance, by checking the response not only at the end of the test pattern, but also somewhere in the middle, it is often possible to detect another fault. This perfectly matches with the capability of our BIST framework, which even performs check response at each clock cycle. Therefore, it is possible in our implementation to perform a compaction of test patterns by generating in hardware only those patterns including a subset of the other ones, thus largely saving test time and TPG area.

5.3.4 Testing Other Internal Switch Modules

A similar process is followed to generate deterministic test patterns for the port arbiters, the LBDR modules and the crossbar. Also the implementation of their TPGs is identical, and so are the optimization techniques.

Again, the most relevant practical implementation issue concerns the communication of test patterns or responses across the switch-to-switch links for the crossbar and LBDR module. The crossbar outputs 34 bits in response to a test vector: 32 data bits, 1 valid bit and 1 stall bit. The communication channel can only carry 32 bits (the valid bit of the channel needs to be permanently set to 1 during test vector transmission, while the stall signal travels in the opposite direction). The two remaining crossbar signals (valid and stall) which do not fit into the link can be either transmitted by means of additional lines used only during testing, or alternatively checked by local comparators, similarly to what has been done for the communication channel. We took the latter approach, and the results in section 5.3.7 again confirm the marginal coverage reduction on TPG faults. Fault coverage of the crossbar is not affected at all by this

choice.

Unlike other modules, test vectors for the LBDR modules should be transmitted across the link, and they take 31 lines (the primary inputs of the LBDR module). So, they perfectly match with the current flit width, provided the number of network destinations does not exceed 64. From there on, the test vector width starts growing logarithmically with the number of destinations, and additional lines may be required on the link.

In contrast, the use of a larger flit width in the network (e.g., 64 bits) would automatically solve the problem. In that case, the test patterns of the LBDR block and the test responses of the arbiter could even be communicated at the same time over the link. Also, since LBDR module and arbiters have only few outputs, their response checking could be performed at the same time on the available tree of comparators, thus cutting down on the test application time (see section 5.3.7).

5.3.5 Fault Detection and Diagnosis

The core of the diagnosis unit is given by comparators which can be implemented in two different ways, by:

- using a level of XORs and an OR gate to provide a single output encoding of the equality test;
- using a two-rail checker TRC (with the second word which is negated);

We opted for the TRC approach, which achieves the self-testing and fault-secure properties [153] although leading to a more complex circuit.

In the diagnosis unit we use 10 different comparators to compare data from all the possible pairs of switch input ports. A smaller number of comparators could be used. Unless time multiplexing is exploited, this would trade cost for diagnosis capability. The maximum number of usable comparators also depends on the number of switch I/O ports. In what follows, we will focus on the internal switches of a 2D mesh for the sake of simplicity (featuring 5 I/O ports, including the local connection to the network interface), however *all irregular topologies supported by LBDR and making use of switches with at least 3 I/O ports* are suitable for our methodology. Obviously, the lower the number of ports, the lower the diagnosis capability.

If we denote two faults in different ports under comparison as *equivalent* if they produce the same output sequence in response to the same input stimuli,

then our comparator and diagnosis logic is able to:

- diagnose the correct position of 1 or 2 faulty channels affected by equivalent or non-equivalent faults;
- diagnose the correct position of 3 faulty channels affected by non-equivalent faults; [1]
- detect the presence of 4 and 5 faulty channels. Anyway, since a 5x5 switch affected by 4 or 5 faults has to be discarded, we don't distinguish between these two scenarios.

One might argue that when a communication channel fails, then the following testing phases have less inputs available and diagnosis capability reduces. In practice, this effect plays only a minor role, since a fault on a communication channel means that also (say) the arbiter of that channel should be considered faulty (unusable). So, the diagnosis capability reduces, but also the number of input ports to be checked reduces as well.

When a switch features only three I/O ports, then the detection and diagnosis capabilities change as follows. Single stuck-at faults can be diagnosed while double faults can be detected, provided they are not equivalent. If they are equivalent, then diagnosis fails. However, when two faults are detected in two ports out of three, the switch should be discarded anyway.

As regards the possible presence of faulty comparators, let us first note that any input vector producing less than four ones corresponds to faults in less than four comparators (we are neglecting the case where all 5 channels are faulty and 4 of them have equivalent faults, which is very unlikely). In case the number of faulty comparators is larger than 3, some configuration exists which may produce a wrong diagnosis. Let us note, however, that it is sufficient to have a single test vector (not a test sequence) featuring less than four ones to immediately recognize the presence of faulty comparators because no combination of faulty channels may produce such response.

5.3.6 BIST-Enhanced Switch Architecture

The switch architecture enriched with the BIST infrastructure is illustrated in Fig.5.6. Only one section is reported. The figure is necessarily at a high ab-

[1] The probability that more than two faulty channels produce the same output sequence in response to the same input stimuli is here neglected.

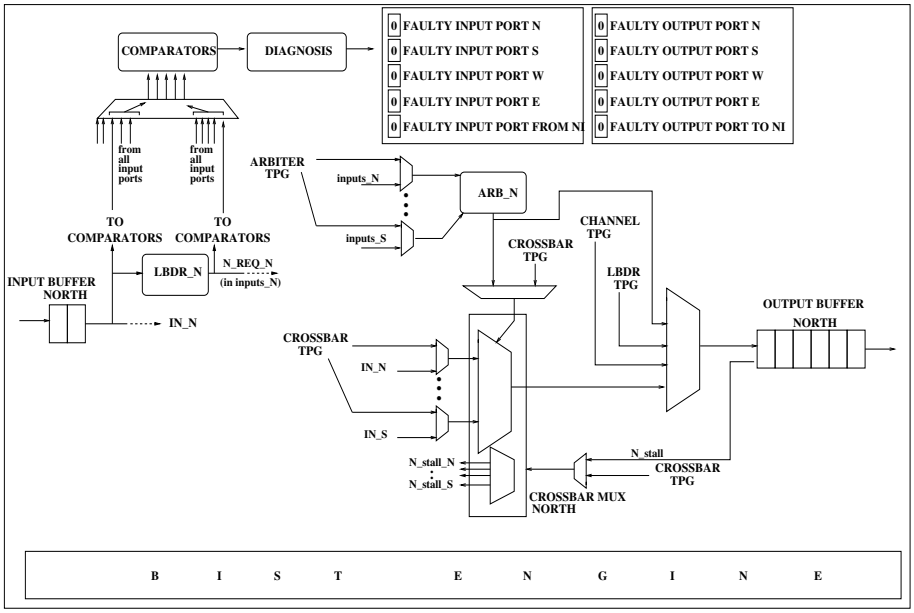


Figure 5.6: BIST-enhanced switch architecture.

straction level, and signal-level connection details previously illustrated in sections 5.3.2 and 5.3.4 are purposely omitted.

A test wrapper consisting of multiplexers can be clearly seen, which enables test pattern injection of TPGs in the modules they test. At the output of the input buffer, test patterns are directly fed to the LBDR module, since they are carried by the communication channel as normal network traffic. A multiplexer in front of each output buffer selects between the switch datapath, the test patterns from the LBDR TPG (feeding the LBDR module of the downstream switch), the channel TPG (directly feeding the channel) and the arbiter test responses (checked in the downstream switch). A BIST engine drives the 4 phases of the testing procedure by acting upon the control signals of the test wrapper.

During the first three phases (communication channel, crossbar, arbiter testing), outputs of the input buffers are selected to feed the comparator tree, while in the last phase (LBDR testing), all LBDR outputs are selected. Test response check and diagnosis are performed at each clock cycle, and result in the setting of 10 bits, indicating whether each input/output port is faulty or not.

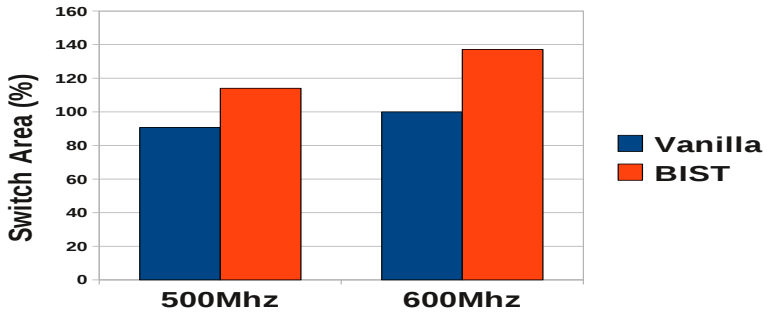


Figure 5.7: Area overhead for BIST implementation as a function of target speed.

Switch sub-block	Test patterns	Test vectors	Coverage
Comm. channel	58	464	99.4%
Arbiter	82	328	97.1%
Crossbar	72	72	99.8%
LBDR	240	240	98.7%

Table 5.1: Coverage for single stuck-at faults.

5.3.7 Experimental results

We performed logic synthesis of a 5x5 switch on the 40nm Infineon low-power technology library. The baseline switch architecture of Fig.5.1 is compared with its BIST-enhanced counterpart.

Fig.5.7 shows the area overhead for BIST implementation as a function of the target speed. Area overhead is 25.27%, which peaks at 37.1% when maximum performance is required. In this latter case, the multiplexers on the critical path are primary targets for delay optimization in exchange for more area. It should be pointed out that this overhead is tightly technology-library dependent.

When considering the BIST infrastructure in isolation (at 600 MHz) most of the overhead comes from the on-chip generation of test patterns (almost 31%) and from the multiplexers (44%) of the test wrapper. Interestingly, although arbiters and LBDR require less test vectors than the communication channel, their TPGs are far more complex due to higher irregularity of their test patterns.

Fault Coverage

Tab.5.1 reports the total number of deterministic test patterns (and test vectors) generated for each tested module, and the associated coverage. This latter was

	Test Cycle	Coverage
Our	864 - 1104	99.3%
[21]	3.88×10^2 - 2.89×10^3	97.79%
[108]	4.05×10^5	95.20%
[42]	2.74×10^3	99.89%
[45]	9.45×10^3 - 3.33×10^4	98.93%
[46]	5×10^4 - 1.24×10^8	N.A.
[17]	320	99.33%
[23]	200×10^3	full (no exact numbers)

Table 5.2: Test application time and coverage of different testing methods.

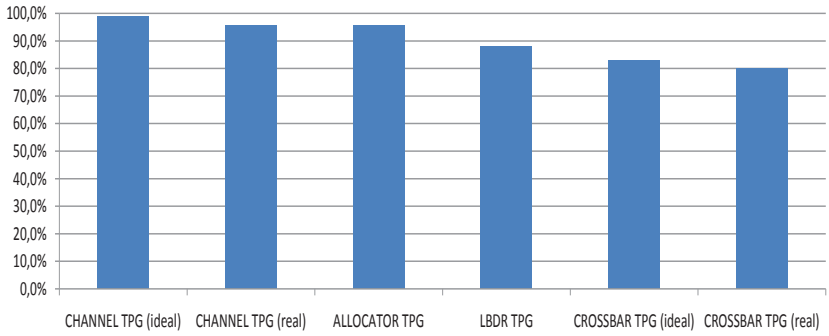


Figure 5.8: Coverage of TPG faults.

derived by means of an in-house made gate-level fault simulation framework: (one or more) faults are applied to any or selected gate inputs, then our testing procedure is run on the affected netlist and the diagnosis outcome is compared with the expected one.

It can be seen that in all cases the coverage for single stuck-at faults closely tracks 100%. The number of test vectors provides the test application time (in clock cycles). A network with a flit width of 32 bits, as assumed so far, would therefore take 1104 clock cycles for testing, regardless of the network size. If we assume 64 bit flits, then LBDR testing occurs in parallel with arbiter testing and total test time reduces to 864 cycles.

These numbers compare favorably with previous work, as Tab.5.2 shows. Only [21] and [17] in some cases do better. However, [21] does not test the control path while [17] reports 320 cycles for a 3x3 mesh (made of a simplified switch architecture) which however grow linearly with network size. Also, this latter approach makes additional use of BIST logic for the control path not accounted for in the statistics.

Multiplicity of Fault Injection	2	3	4	5
Coverage	99.2%	96.4%	96.6%	96.6%

Table 5.3: Coverage for multiple random stuck-at faults.

We feel that area overhead is hardly comparable with previous work since whenever numbers are available, features of the testing frameworks are very different (e.g., control path not tested [21], test patterns generated externally [45,108], diagnosis missing [42,45,46,108], lack of similar test time scalability [17,18], NoC architecture with overly costly links [42]). Moreover, the impact of synthesis constraints is never discussed.

Fig.5.8 reports the coverage of TPG faults. While single stuck-at faults in the allocator and channel TPGs feature a coverage of roughly 95%, worse results are obtained for the LBDR and especially for the crossbar TPGs. We verified that their lower coverage is a direct consequence of the low number of test patterns they generate. The designer can then choose whether increasing crossbar TPG area and having it generate more patterns or dedicating a separate test phase to TPGs. Also, when comparing real vs ideal coverage of channel and crossbar TPGs, it is possible to assess the marginal reduction of TPG fault coverage as an effect of the local (instead of remote) check of some signals of these modules in the switch they belong to (see section 5.3.2 and 5.3.4).

Since our BIST infrastructure targets multiple stuck-at faults from the ground up, we characterized fault coverage for multiple faults as well. We have injected multiple faults randomly in the gate-level netlist of the switch and checked the diagnosis response. Fault multiplicity was 2,3, 4 and 5 and fault injections for a given multiplicity were repeated 1000 times, as in [23]. As Tab.5.3 shows, the proposed BIST framework provides a higher than 96% coverage in every scenario. Interestingly, the coverage saturates with 4 and 5 faults since the probability to inject errors in a module already affected by an error becomes high.

5.4 Built-In Scan Chain-Based Testing Framework

The test of the integrated circuits is traditionally performed by means of scan chains. The scan-chain still features a wide adoption in the industrial environment although several alternative testing techniques have been proposed. Indeed the success of the scan-chain is justified by some key reasons. First of all, the main synthesis tools support a semi-automatic insertion of the scan-chain

in the logic under test. In addition, it achieves an extremely high stuck-at-fault coverage of both the data and the control path. Anyway, it is showed in [45] that traditional full-scan and boundary scan strategies like [47, 106–108] incur an hardly affordable area overhead. [45] also proposes a partial scan technique in combination with an IEEE 1500-compliant test wrapper. Area overhead is greatly reduced, but test application times amount to tens of thousands of clock cycles and test pattern generation time does not scale. Then the advantages in terms of coverage and design time of the scan-chain technology are offset by severe area overhead and high latency. Furthermore, the scan-chain is typically associated with test patterns injected by off-chip sources. The injection of external test patterns becomes infeasible in modern integrated circuits due to the limited pin budget and the difficulties in reaching the widespread logic of the NoC. Finally, life-time testing is not supported by this solution because of the need for an industrial test environment.

Aware of the relevance of the scan chain-based mechanism, we enrich our design space exploration of testing strategies by implementing this latter technique in the NoC switch. Scan chain-based mechanisms are conventionally coupled with algorithmic deterministic patterns self-generated by Automatic Test Pattern Generator (ATPG) tools. Thus we rely on ATPGs to stimulate the DUT although we do not finally evaluate a conventional implementation but we custom-tailor the scan chain for the target NoC setting in order to alleviate its intrinsic limitations.

As a result, the next sections present the switch architecture enhanced with the customized scan chain infrastructure. In particular, we implement a testing framework based on test patterns generated by means of an ATPG tool, Tetramax. This latter tool requires as input the post-synthesis netlist together with the list of the primary inputs/output and generates as output a sequence of algorithmic deterministic patterns. Finally, the performance of the proposed solution is compared in terms of area, coverage and latency with a baseline (unoptimized) scan-chain strategy and with the testing framework based on handcrafted deterministic patterns in Section 5.3.

5.4.1 The Scan Chain Tool-Flow

In order to implement a scan chain in our design under test we exploited the Synopsys tool enhanced with some dedicated commands (`set_dft_signal`, `set_scan_configuration`, `set_scan_path`, etc.). Thus, the post-synthesis design was automatically augmented with scan-enable sequential cells and scan-in/scan-out ports for the write/read of the scan chain control bits. Finally,

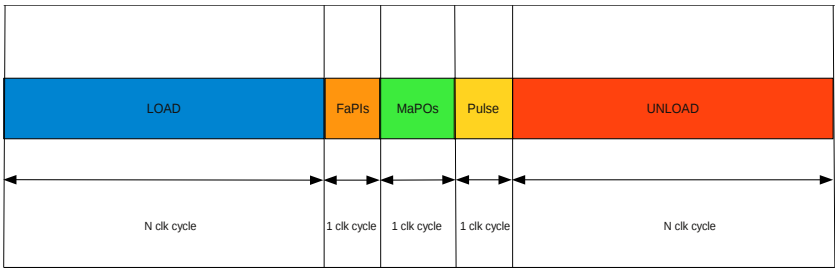


Figure 5.9: Latency breakdown of a scan chain-based test.

the test patterns were generated by means of the Tetramax tool. Tetramax requires as input the post-synthesis netlist together with the list of the primary inputs/outputs and the scan-in/scan-out ports of the design.

The injection of a test pattern in a scan chain-enabled design follows a sequence of 5 phases (see figure 5.9). During the first phase (*load*), *n* test bits are driven through the scan-in port. Since a single bit for clock cycle is injected in each chain, the length of this phase is equal to *n* clock cycles where *n* is the number of sequential cells of the chain. During the next two phases (*force_all_pis* and *measure_all_pos*), the primary inputs are stimulated with the test data pattern and the primary outputs' response is read out. Both these phases last 1 clock cycle. The reset ports are tested during the third phase (*pulse*) and finally the scan-out bits are read in the last phase (*unload*). The *unload* phase lasts *n* clock cycles. Then the *unload* and the *load* phases dominate the latency required to inject a test pattern. Intuitively the number of sequential cells (*n*) for chain has a relevant impact on the final latency of the test.

It is possible to model the total latency (*TotLat*) of a test based on scan chains by means of the following formula:

$$TotLat = (TotCells/TotChains + 4)X(NumPat) + TotCells/TotChains \quad (5.1)$$

The formula is parametrized by the number of test patterns (*NumPat*), the number of total design cells (*TotCells*) and the number of scan chains (*TotChains*). To notice that it is taken into account the overlapping of the *unload* phase with the *load* phase of the next test pattern.

Similarly, the number of bits stored by the test pattern generator can be modeled by the following formula:

$$NumBit = (PIs + ScanChainPort + TotCells)XNumPatterns \quad (5.2)$$

In this case, *PIs* represents the number of primary inputs of the design and *ScanChainPort* takes into account the additional inputs required by the scan chain test framework.

The number of bits of Equation 5.2 gives the size of the test pattern generator, thus its area footprint. Modeling the latency and the number of bits of the test generator represents a key step for the further optimizations presented in the following sections.

5.4.2 The Baseline Implementation

As in the cooperative testing framework of section 5.3, we use the xpipesLite switch architecture [110] to implement a baseline scan chain-enabed testing strategy in a realistic NoC setting. When implementing a scan chain framework, it is possible to set few Synopsys parameters to specify the number of chains in the design, the number of cells for each chain and group cells in a single chain.

We did not constrain the synthesis tool during the first baseline implementation. As a result, the tool created 48 scan chains for the whole switch design. Each scan chain was composed of 40 sequential cells and required 147 test patterns. The test achieved a high stuck-at-fault coverage (99.95%) and a reasonable latency of 6500 cycles. Anyway, each scan chain differs from the other scan chains thus a dedicated set of test patterns was required for every chain. As a consequence, 310k bits were stored into the test pattern generator and the resulting area footprint was too severe when enhancing the scan chain framework with a built-in self-testing approach. Therefore alternative solutions are required in order to alleviate the area overhead.

We constrained Synopsys to modify the number of scan chains in the switch. Interestingly the synthesis tool balances the number of sequential cells within each scan chain. As expected, the latency linearly decreased with the increase of the number of scan chains following Equation 5.1. However, the area overhead stayed approximately constant since the same number of test patterns was required despite the modification of the scan chain length (the remaining parameters of Equation 5.2 did not change as well).

In addition, the whole switch needs to be discarded when a single fault is detected with the above implementations. Although diagnosis logic can associate

a faulty cell to a single faulty scan chain, the position of the cell in the switch is still unknown. In fact, the cells of a switch port are not grouped in a single scan chain and every scan chain collects cells from multiple switch ports. Therefore a faulty scan chain does not carry the information required to enable a graceful degradation of the switch as in section 5.3.

5.4.3 Customizations for the NoC Setting

A generic implementation of a scan chain-based testing strategy proved area hungry and resulted in poor fault tolerance. Thus customizations for a NoC setting were envisioned to overcome these latter concerns.

Following the intuition of Section 5.3, the key idea of our BIST framework based on scan chains consists of exploiting the inherent structural redundancy of an on-chip switch. We collected the cells of each input and output ports in distinct groups. In particular, a port-arbiter, a crossbar multiplexer and an output buffer group were instantiated for each output port, while a routing module and an input buffer group were instantiated for each input port. Then we performed a dedicated synthesis for each group before to glue all the synthesis results in a single design (see Figure 5.10). In this way every scan chain is associated with a cell group and failure of a chain can be viewed as the failure of the associated switch input or output port. Our diagnosis strategy will therefore provide an indication of whether input and output ports of a switch are operational or not. Moreover, all the scan chain instances are assumed to be identical, therefore there is a unique test pattern generator (TPG) for all the instances of the same block, thus cutting down on the total number of bits stored in the TPGs. Finally all the instances of the same scan chain should output the same results if there is no fault. As a consequence, the test responses from these instances are fed to a comparator tree similar to the tree of Section 5.3. As already proved, this makes the successive diagnosis much easier.

The core of the diagnosis unit is given by comparators which are implemented using a two-rail checker TRC achieving the self-testing and fault-secure properties. When we consider a 5x5 switch then we use 20 different comparators to compare data from all the possible pairs of switch input ports and switch output ports. Thus the diagnosis logic diagnoses the correct position of 3 faulty input ports and 3 faulty output ports affected by non-equivalent faults. Finally, it detects the presence of 7, 8, 9 and 10 faults located in different input or output ports. Anyway, since a 5x5 switch affected by more than 6 faults has to be discarded, we don't distinguish between these latter scenarios.

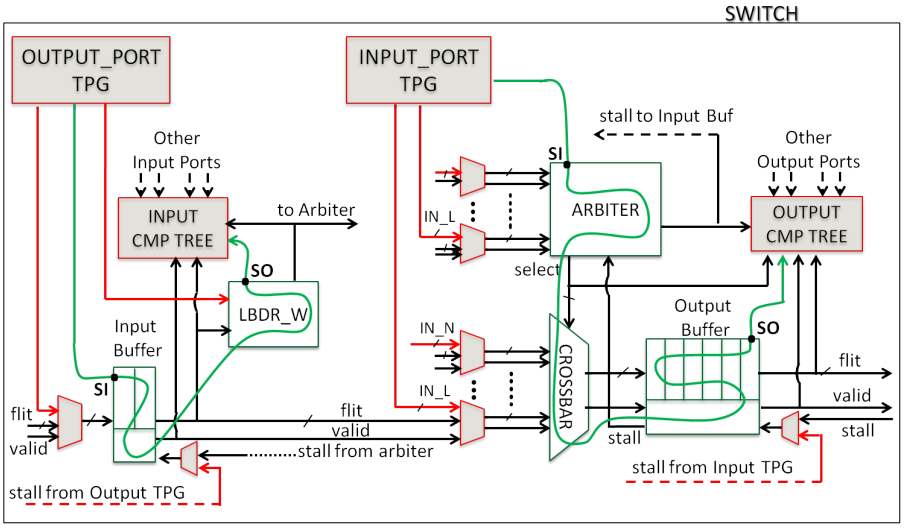


Figure 5.10: Practical implementation of the proposed scan chain-based test.

Although the principle is similar to what has been proposed in Section 5.3, this solution presents a fundamental drawback. If the TPG of a set of block instances is affected by a fault, then the comparison logic will not be able to capture this since all instances provide the same wrong response. Furthermore the communication channels between switches are not tested as a part of the switch testing framework.

5.4.4 Experimental Results

The switch enhanced with the proposed scan chain was synthesized by means of a 40nm Infineon technology library. The test patterns were generated by Tetramax. A test pattern can be generated in hardware by using a clock cycle counter and some logic to generate the values of the input signals for the DUT. Then we exploited the Tetramax test patterns to implement a test pattern generator built-in into the switch. A test wrapper consisting of multiplexers enables test pattern injection of TPGs in the scan chain they test.

Scan Chain Results: Custom Vs Baseline Implementation.

Both the baseline and the proposed solution well perform in terms of coverage: they achieve the 99.9% of single stuck-at-fault coverage. However,

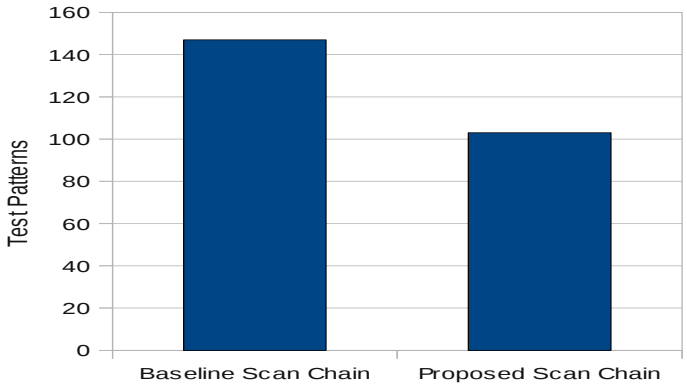


Figure 5.11: Maximum number of test patterns.

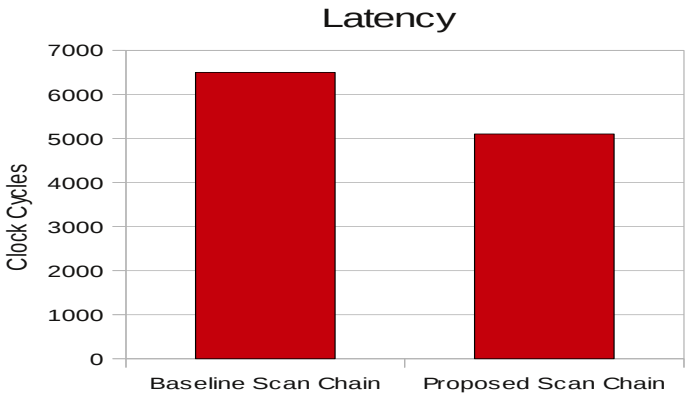


Figure 5.12: Latency of the scan chain-based test.

the proposed scan chain customization brought an interesting reduction of the number of test patterns. In fact, Tetramax exploited respectively 103 and 65 test patterns to test the scan chains of the output and the input port. On the contrary, 147 test patterns were required to test the baseline scan chains (see Figure 5.11). Interestingly the pseudo-random clustering of the cells in the baseline solution did not allow Tetramax to perform the test patterns optimization achieved in the custom-tailored solution.

The reduction of the test patterns enables shorter `load` and `unload` phases. Thus the latency of the proposed solution decreases by 20.2% with respect to the baseline counterpart (see Figure 5.12). Furthermore, the area overhead benefits from both the test pattern reduction and the proposed optimizations (i.e. single sets of test patterns test all the instances of the same port). Then, the area

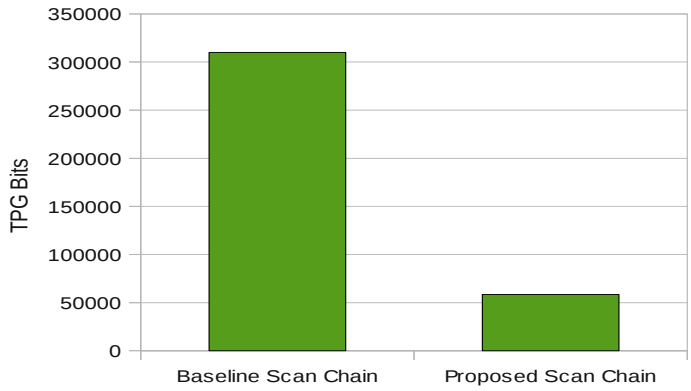


Figure 5.13: Total number of bits stored by the test patterns generator.

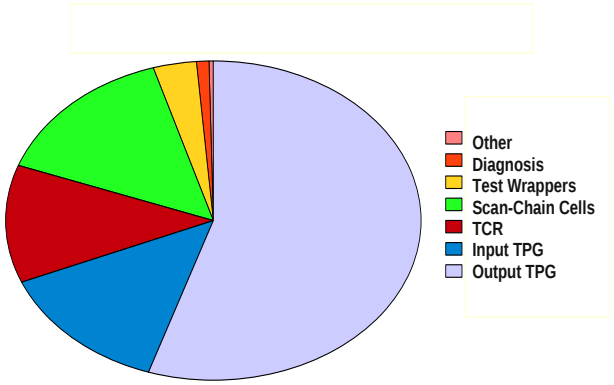


Figure 5.14: Area overhead breakdown of the customized scan chain-enabled testing strategy.

overhead of the proposed solution was cut down by the 82% with respect to the baseline one (see Figure 5.13). Although this result was achieved through an aggressive reduction of the built-in test pattern generators' area, these latter still introduce the highest area overhead (as reported in Figure 5.14).

Comparison with Handcrafted Deterministic Test Pattern-Based Framework

In this section the scan chain-based proposed solution is compared with the deterministic test patterns-based solution of Section 5.3 in terms of coverage, latency and area overhead. As showed by Figure 5.15, the area of the deterministic test patterns-based solution outperforms the scan chain-based frame-

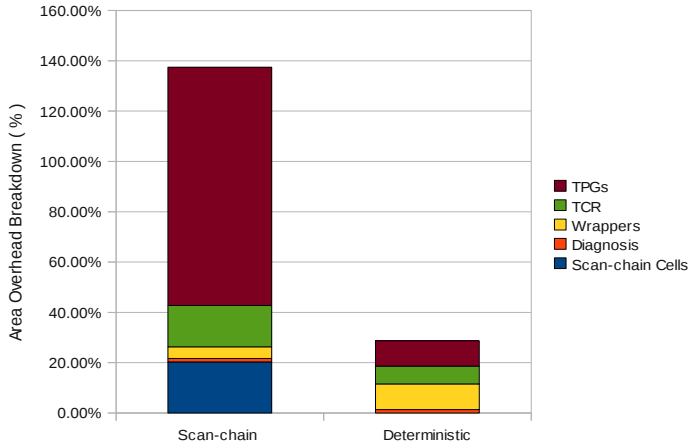


Figure 5.15: Area overhead breakdown of the scan chain-based and deterministic test patterns-based solutions.

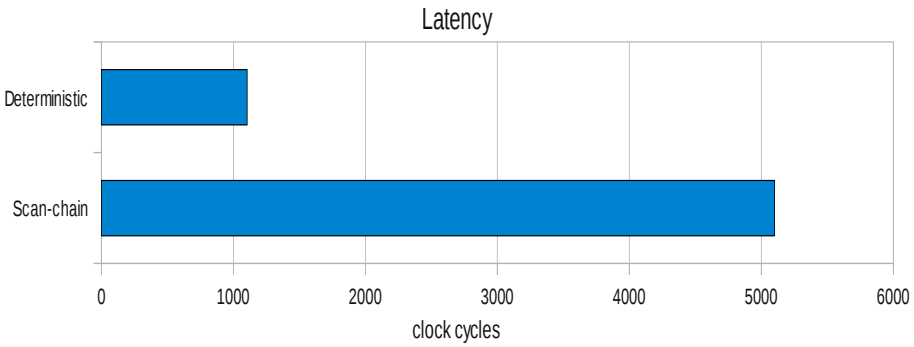


Figure 5.16: Latency of the scan chain-based and deterministic test pattern-based solutions.

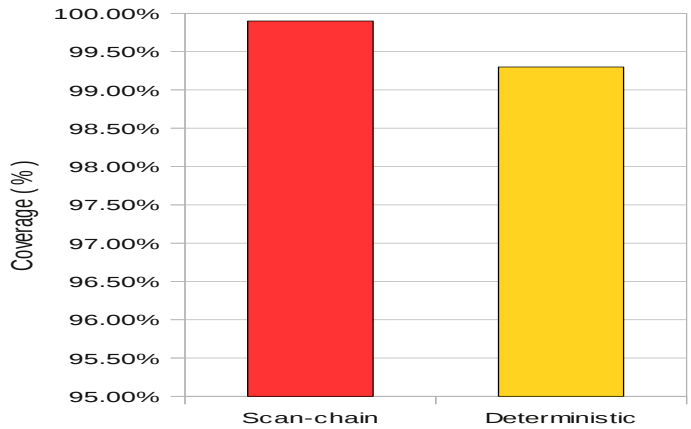


Figure 5.17: Coverage of the scan chain-based and deterministic test pattern-based solutions.

work. They introduce respectively an area overhead lower than the 30% and higher than the 130% with respect to a switch without BIST/BISD capability. Although the above mentioned customizations alleviates the scan chain test pattern generators' area overhead, the area footprint of these latter is still similar to the whole switch area confirming how the scan chain technique is not well suited for built-in approaches.

Furthermore, the latency of the scan chain solution is five times higher than the deterministic counterpart underlining the intrinsic slowness of the `load` and `unload` phases of a scan chain-based test (see Figure 5.16). Finally, both the BIST frameworks achieve a single stuck-at-fault coverage higher than 99% as reported in Figure 5.17.

5.5 Built-In Pseudo-Random Self-Testing

Most BIST architectures in literature use pseudo-random test pattern generators. However, whenever this technique has been applied to on-chip interconnection networks, overly large testing latencies have been reported. On the other hand, the alternative approaches proposed in the previous sections either suffer from large area penalties (like scan-based testing) or high design effort for the use of deterministic test patterns. This section presents the optimization of a built-in self-testing framework based on pseudo-random test patterns to the microarchitecture of network-on-chip switches. As a result, we will demonstrate that that through proper customizations for an on-chip setting

fault coverage and testing latency approach those achievable with deterministic test patterns while materializing relevant area savings and enhanced flexibility.

This section aims at a low area footprint and low-latency testing framework for NoCs by optimizing built-in pseudo-random self-testing for the microarchitecture of a NoC. The choice of pseudo-random testing potentially reduces the area overhead, although the test time concern arises. This latter was tackled by reusing test responses of switch sub-blocks as test patterns for cascaded blocks combined with specific test pattern optimizations for selected blocks to preserve coverage.

The efficiency of the developed testing framework is demonstrated by the comparison with the cooperative testing strategy (section 5.3) relying on deterministic test patterns and strictly aiming for low testing latencies.

5.5.1 The Testing Strategy

The baseline switch architecture is the same as showed in Fig.5.1. It relies on a stall/go flow control protocol and implements distributed routing by means of a route selection logic located at each input port. Failure of a switch input or output port and their associated switch internal sub-blocks can be viewed as the failure of the connected link. The diagnosis strategy proposed in this section will therefore target this requirement and will provide an indication of whether input and output ports of a switch are operational or not. As the cooperative testing strategy of Section 5.3, each switch can in turn test its several internal instances of the same sub-blocks (crossbar muxes, communication channels, port arbiters, routing modules) concurrently by means of pseudo-random patterns. The testing framework of Section 5.3, which comes up with one of the lowest testing latencies reported in the open literature for similar single stuck-at fault coverages, will be used for the sake of comparison as a lower bound for testing latency.

Deterministic test patterns come with their own drawbacks, especially the large effort to define the test patterns and the poor adaptation to technology library and/or node migrations. For these reasons, we extended our analysis to a framework relying on a pseudo-random testing. However, naive application of such testing strategy to NoCs like [23] results in unacceptable testing latencies of hundreds of thousands of cycles. In fact, a baseline pseudo-random testing usually tests the modules under test in sequence to exploit a single test pattern generator thus reducing the area overhead of the framework. Anyway, the total testing latency represents the potential killer of this approach. In fact,

the testing latency does not scale at the increase of the number of modules to test. Furthermore, random test patterns do not allow an efficient testing of the control-path. Indeed, some states of the FSMs typically have a low probability to be reached through pseudo-random input patterns.

We propose NoC test optimization based on the three following ideas:

- To increase the fault coverage within reasonable times, we exploit the knowledge of the architecture under test to rise the probability of driving the FSMs to test-relevant states. We thus foster an hybrid approach (combining deterministic and pseudo-random patterns) for these machines.
- To reduce the area overhead we reuse the test responses of a switch sub-block as test patterns for the cascaded ones.
- To minimize the latency we test the redundant replicas of the same modules in parallel, like the solution in section 5.3.

We progressively optimize a baseline pseudo-random testing framework for the NoC in incremental steps. First of all, we decompose the network (i.e. the switch and the channel) into its building blocks: the arbiters, the crossbar multiplexers, the input buffer, the output buffer, the LBDR and the link.

Then, we exploit a Linear-Feedback-Shift-Register (LFSR) for test pattern generation and a Multiple-Input Signature Register (MISR) for compression of test responses.

Finally we progressively cascade switch sub-blocks to the LFSR and monitor the variations of testing latency and fault coverage. For cascaded sub-blocks, test responses of the upstream block are test patterns for the downstream one. Whenever possible, we try to compensate coverage degradation by means of ad-hoc optimizations thus cascading as many blocks as possible and minimizing the number of test phases. This process is detailed hereafter.

5.5.2 Testing communication channels

As a first step a stand-alone output buffer was tested. A 34 bit LFSR was required to drive the flit, the valid and the stall signal. A high fault coverage was achieved (99.5%) in 250 clock cycles. The coverage for single stuck-at faults was derived by means of the Tetramax tool.

As a next step, the full communication channel was tested. Then the link and the input buffer of the downstream switch were cascaded to the output

buffer, as reported in Fig.5.18(a). In this case, the testing coverage saturated to 91.2% after 250 clock cycles. This coverage degradation pointed to the need of exploiting the knowledge of buffer implementation to increase test efficiency.

As showed in Fig.5.18(a), we increased the controllability of the channel by driving the stall signals. From an implementation viewpoint, there are several practical issues. In fact, the *stall_channel* signal of the input buffer, which lies in the downstream switch, should be driven by the LFSR as well. This would require an additional wire in the switch-to-switch link. A similar concern is that the *stall_out* signal from the output buffer should be brought to the MISR in the downstream switch, again requiring an additional wire in the link. To avoid the extra wires, we opted for the solution in Fig.5.18(a): *stall_channel* is driven by the LFSR of the downstream switch, while *stall_out* is not observed directly by a MISR but it is still connected to the arbiter affecting its behavior. From the testing viewpoint nothing changes, since all the pseudo-random LFSRs inject the same patterns synchronously.

As a result, we obtained a fault coverage of 99.7% in 250 clock cycles.

5.5.3 Testing multiplexers of the crossbar

A similar process was followed to include the multiplexers of the crossbar in the testing framework. The channel modules were cascaded to the multiplexer as showed in Fig.5.18(b). The multiplexer was directly fed by the LFSR while the channel was crossed by the test responses of this latter module.

This incremental testing step required interesting optimizations to limit the LFSR area overhead and to preserve a high fault coverage. First of all, the multiplexer of the crossbar presents 165 inputs (33 inputs for port) when taking into account a 5x5 switch. Thus a baseline testing environment could require a relevant area overhead due to a 165 bits LFSR. As a second concern, few control signal configurations (see *select* in Fig.5.18(b)) allow the multiplexer input signals to cross the multiplexer logic. As a consequence, we experimented a high degradation of the fault coverage of the cascaded channel due to the low amount of testing packets forwarded by the multiplexer.

In order to tackle the LFSR area overhead, we fed each input port of the multiplexer with the same 34 pseudo-random bits exploiting a data shift of $6 \cdot N$ bits for every input port. To note that the N parameter corresponds to the multiplexer port ID. As a result, we preserved the fault coverage and the LFSR extension from 34 bits to 165 bits was no longer required.

Concerning the issue related to the multiplexer control signals, we de-

signed a ring counter driving them to data transparent configurations (`10..0, 01..0, 00..1`). Furthermore, the randomness of the configurations is preserved by exploiting an LFSR bit to drive the enable signal of the ring counter. Thus the counter moves from a configuration to the next one when the LFSR pseudo-random bit is set to `1`.

The above mentioned optimizations finally guaranteed a fault coverage and a testing period for the cascaded crossbar-channel similar to the results obtained by the stand-alone channel (i.e. a 99.5% of coverage in 250 clock cycles).

5.5.4 Testing LBDR

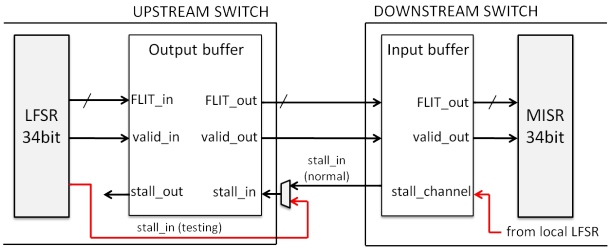
Unlike the crossbar module, the LBDR block associated with the port under test lies in the downstream switch. Thus, it is fed by the input buffer and it is directly connected to the MISR as showed in Fig.5.18(c). In this case, the MISR was extended to 39 bits in order to compress also the LBDR outputs.

However, since the LBDR consists of a many-input hard-to-test combinational logic, the implementation of an effective testing is challenging. As a result, the following three optimizations were introduced to rise the fault coverage of the routing mechanism:

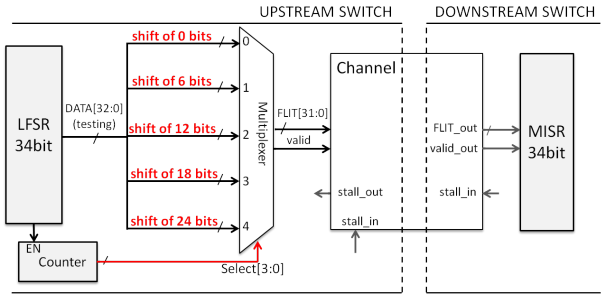
1. Some of the LBDR inputs were directly driven by the local LFSR to restore the highest randomness of the test patterns.

2. The probability to inject a packet to the local port is negligible when exploiting pseudo-random patterns. In fact the LBDR routes the packet in the direction of the local port only when the local ID (*SID bits*) is equal to the destination ID of the packet. Then we connected the SID bits to the input 0 of the crossbar multiplexer. As a result, all the packets forwarded by the port 0 of the crossbar multiplexer are routed to the local port of the receiver switch. Thus, this solution allows to test all the routing scenarios.

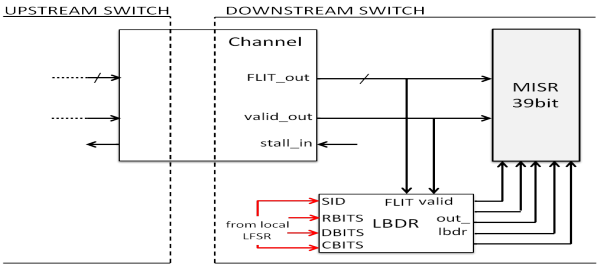
3. Since the LBDR logic performs its computation only when stimulated by header flits, the test patterns bits at the input 0 of the crossbar multiplexer were connected in order to generate exclusively header or tail flits (i.e. the flit type bits were driven by the same MISR output bit thus assuming only the 11/00 configuration associated to a header/tail flit type). As a consequence, payload flits are not longer forwarded by the port 0 of the multiplexer. In this way, the number of header flits was increased and the routing logic efficiently stimulated and tested.



(a) Testing communication channel.



(b) Testing crossbar.



(c) Testing LBDR routing logic.

Figure 5.18: Optimization steps of the pseudo-random testing framework.

Finally the testing framework achieved a 97.2% of total fault coverage for the cascaded blocks in 1000 clock cycles.

5.5.5 Testing Arbiters

The test did not achieve a high coverage when the arbiter was directly connected to the existing testing framework following the approach taken so far. The reason lies in the poor efficiency in testing the arbiter FSM. As a result, in the final testing framework the arbiter logic is directly driven by the LFSR and its test responses feed a dedicated 11 bits MISR. Although some area overhead was introduced with the additional MISR, we found it necessary to achieve the maximum coverage for such a strategic module.

5.5.6 BIST-enhanced switch architecture

The switch architecture enriched with the BIST infrastructure is illustrated in Fig.5.19. A test wrapper consisting of multiplexers can be clearly seen, which enables test pattern injection of the LFSR in the modules it tests. A unique 34 bits LFSR generates the pseudo-random patterns to test in parallel every switch port. Moreover a dedicated 11 bits MISR for every port collects the test response from the output port arbiters and a 38 bits MISR for every port performs the signature analysis of the test responses from the crossbar, the channel and the LBDR blocks.

Test diagnosis results in the setting of 10 bits (one for every MISR), indicating whether each input/output port is faulty or not. This meets the requirements of the LBDR configuration algorithm in [152]. Interestingly, the testing framework is able to reveal the correct position of multiple faulty channels since a MISR is dedicated to each port. Obviously, it is not possible to distinguish the elementary faulty module inside the faulty port. However, the proposed testing framework is based on the assumption that the functionally coupled modules for an input port are its routing block, its upstream communication channel, the port arbiter and the crossbar multiplexer in the upstream switch associated with that channel. Thus, when one of the above mentioned functionally coupled modules fails then the associated safe logic would be unusable anyway.

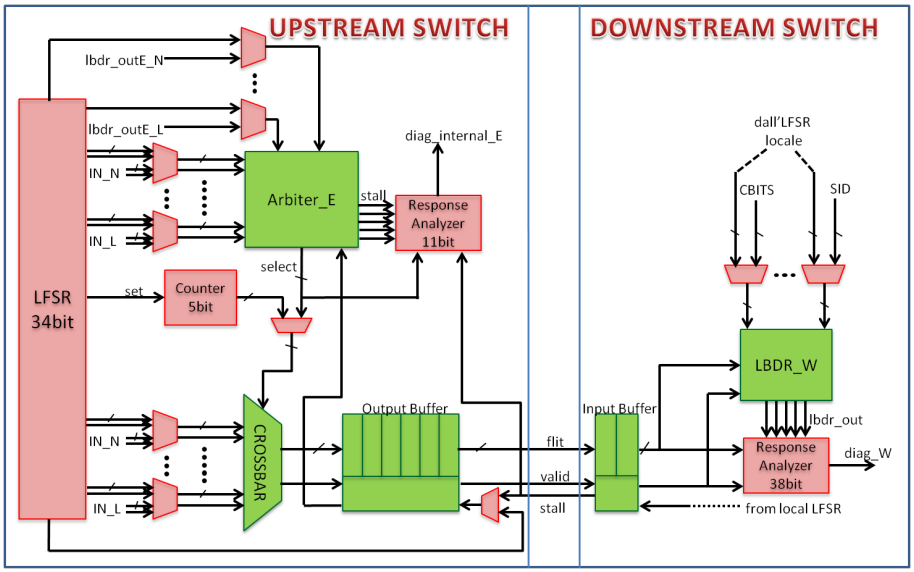


Figure 5.19: BIST-enhanced switch architecture.

5.5.7 Experimental Results

We performed logic synthesis of a 5x5 switch on an industrial 40nm Infineon technology library. The baseline switch architecture of Fig.5.1 and the proposed switch augmented with the pseudo-random testing framework give approximately the same maximum operating speed of 600 MHz when synthesized for maximum performance, thus proving that our BIST-enabled switch is capable of at-speed testing.

Fig.5.30(a) reports the total number of test patterns (clock cycles) generated for the 5x5 switch and the associated coverage for both the deterministic and the pseudo-random testing framework. It can be seen that the proposed pseudo-random framework exploits a further degree of freedom with respect to the deterministic solution. In fact, it can trade latency for coverage. Interestingly, in all the analyzed latency scenarios, the coverage for single stuck-at faults is above 94%. Especially the deterministic framework achieves a 99.3% of coverage in 1104 clock cycles while the coverage of the proposed framework ranges between 94.2% and 98.2% achieved in 500 and 10.000 clock cycles respectively.

These numbers prove that it is possible to achieve a coverage and a testing latency with pseudo-random test vectors that approach those of deterministic

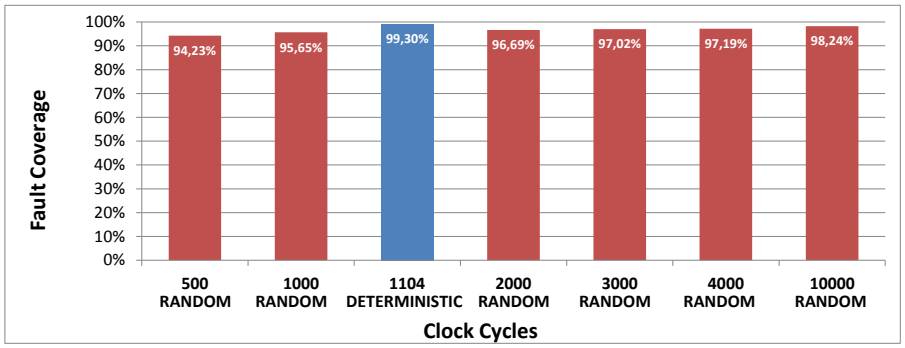


Figure 5.20: Coverage for single stuck-at faults as a function of the test latency.

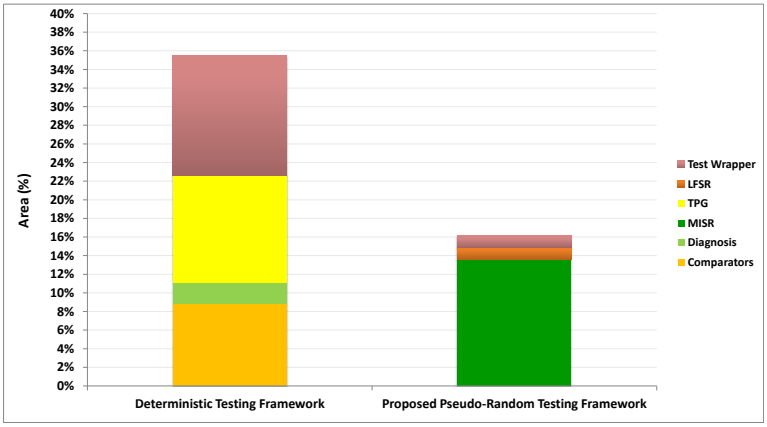


Figure 5.21: Area overhead for BIST implementation.

vectors, although not entirely achieving the same quality metrics. This was made possible by the performed optimizations which take advantage of the knowledge of the architecture under test to some extent, without reverting to full deterministic vectors. The remaining difference in quality metrics with respect to them can be considered as the price to pay for reduced area overhead, as proved hereafter. Also, it should be mentioned that a pseudo-random approach to testing is more appealing in terms of shorter design time and adaptivity to architecture, library and technology changes.

Fig.5.30(b) shows the area overhead for the proposed testing framework and the reference one based on deterministic patterns when applied to a 5x5 NoC switch. Both of them were synthesized on an 40nm Infineon technology li-

brary. Area overhead is referred to the baseline BIST-less switch in Fig.5.1. The area overhead of the proposed framework based on pseudo-random patterns is 16.3%, which rises to 37.1% for the framework based on deterministic patterns. Interestingly, in the proposed framework most of the overhead comes from the MISR modules; on the other hand, the test wrappers and the TPGs (i.e. the LFSRs) prove extremely lightweight when compared with their deterministic framework counterparts. To note that the removal of most of the test wrappers in the pseudo-random framework was possible thanks to the proposed exploitation of test responses to test other cascaded modules.

5.6 Testing Framework Comparison

All the testing frameworks proposed so far have been synthesized on an industrial 40nm Infineon technology library. This section performs a comparison between them in terms of area overhead, stuck-at-fault coverage, testing latency and routing delay. Thus, the section evaluates (1) the testing framework with algorithmically generated deterministic test patterns and scan-chain control, (2) the testing framework based on handcrafted deterministic test patterns and (3) the testing framework based on pseudo-random test patterns.

5.6.1 Stuck-at-faults coverage and testing latency

Figure 5.22 reports the total number of test patterns (clock cycles) generated for a 5x5 switch and the associated coverage for the 3 testing frameworks above mentioned. It can be seen that the solution (3) based on pseudo-random test patterns exploits a further degree of freedom with respect to the other solutions. In fact, it trades latency for coverage without affecting TPG architecture and area footprint of the framework. The highest coverage is achieved by the frameworks based on deterministic test patterns: (1) and (2) achieve respectively 99.9% and 99.3% of coverage in 5104 and 1104 clock cycles. Handcrafted generated deterministic test patterns achieve a worst coverage than algorithmically generated deterministic test patterns but they guarantee a lower testing latency. The coverage achieved by frameworks based on pseudo-random test patterns approaches those of deterministic vectors, although not entirely achieving the same quality metrics. Indeed, the coverage of (3) ranges between 94.2% and 98.2% in 500 and 10.000 clock cycles respectively.

These numbers prove that it is possible to achieve a high coverage and a low testing latency with both pseudo-random test vectors and deterministic vectors.

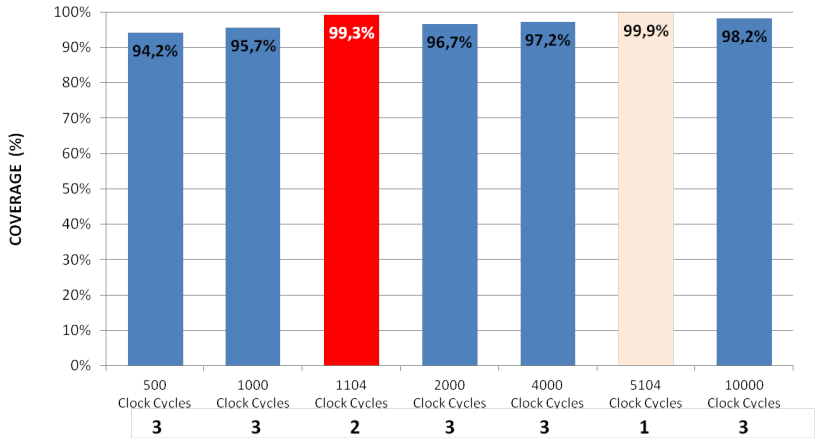


Figure 5.22: Coverage for single stuck-at faults as a function of the test latency.

This was made possible by exploiting a high degree of testing control and taking advantage of optimizations based on the knowledge of the architecture under test.

5.6.2 Routing delay

The baseline switch architecture of Figure 5.1 and the switch augmented with the 3 proposed testing frameworks have been synthesized for maximum performance to estimate their critical paths. The critical path of the baseline switch architecture starts from the input buffer, crosses the internal switch logic and ends into the output buffer. As represented by Figure 5.23, the time penalty introduced by the testing frameworks is low. Indeed the critical path of (1) and (3) lays on the same path as the baseline solution although additional test wrappers worsens the final routing delay. Differently, the testing logic complexity brought by the testing phases in (2) brings a new critical path. This latter starts in the arbiter TPG and ends in the diagnosis logic after crossing the comparator tree. Finally (2) comes with a 37% longer critical path and results the slowest testing alternative.

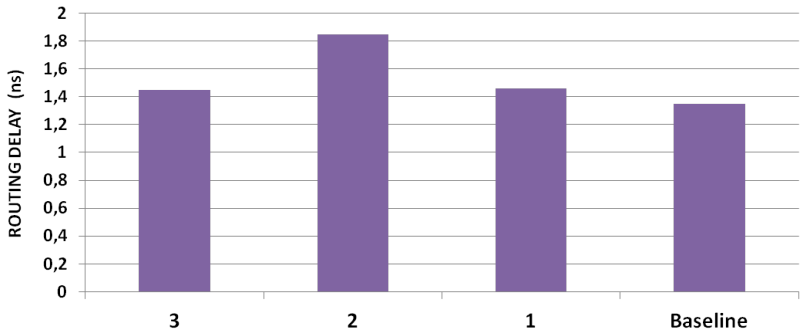


Figure 5.23: Routing delay for BIST implementations.

5.6.3 Area overhead

Figure 5.24 shows the area overhead for the 3 testing frameworks when applied to a 5x5 NoC switch. The area overhead is referred to the baseline BIST-less switch in Figure 5.1.

Algorithmically generated deterministic test patterns lack of efficiency when exploited to test a complex design. In order to restore their efficiency in terms of coverage then wide scan-chains must be adopted. (1) exploits algorithmically generated deterministic test patterns augmented with scan chains. In order to reduce the area overhead introduced by the associated test patterns generators, the intrinsic redundancy of the switch was exploited to implement dedicated scan chains to each switch input/output ports and finally a single TPG was required for testing all the switch ports. Despite the effectiveness of the proposed optimizations, a built-in self-test can not still be envisioned due to the severe area overhead required by TPGs for algorithmically generated deterministic patterns associated with scan chains. In fact, (1) requires 137% of area overhead with respect to a switch without testing capabilities.

The alternative approaches outperform the deterministic algorithmic patterns-based solution in terms of area footprint as depicted in Figure 5.24. Although the effort required by a hardware designer to implement (1) by means of automatic tools is low, the resulting design does not represent a valid solution for a highly-constrained and performance-critical NoC.

In particular, the deterministic handcrafted patterns-based solution cuts down the area overhead to 37.1%. Concerning pseudo-random approaches, (3) results the lightest testing framework with 16.3% of area overhead. In partic-

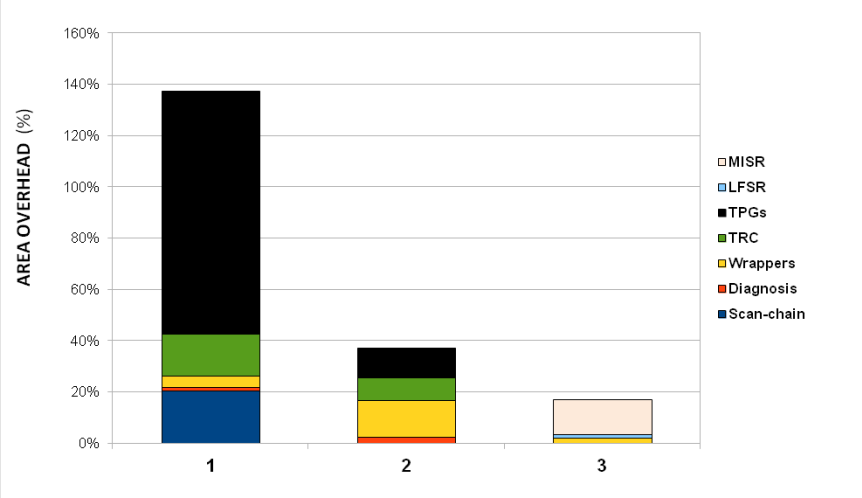


Figure 5.24: Area overhead for BIST implementations.

ular, the most of area overhead comes from TPGs, test wrappers and comparators modules in (2). On the contrary, test wrappers and the TPGs prove extremely lightweight in (3). In fact, (3) implements TPGs by means of a single LFSR furthermore it removes the most of the test wrappers by exploiting longer chains of modules than (2).

Awareness of the architecture under test enabled testing framework optimizations that improved coverage while reducing latency. As a result, the quality metrics of a testing framework based on handcrafted deterministic test patterns were approached by pseudo-random patterns alternatives while materializing significant area savings and enhanced flexibility. Getting precisely the same coverage numbers is however not affordable for pseudo-random testing within reasonable testing times. Therefore, the small percentage increase in fault coverage that deterministic test patterns are able to provide represents an advantage that should be strictly traded off with a larger area footprint and a lower flexibility. Finally, it should be mentioned that a pseudo-random approach, like (3), is more appealing than a handcrafted deterministic approach, like (2), in terms of shorter design time and library and technology changes.

5.7 Testing Framework for Multi-Synchronous Networks

Traditional globally synchronous clocking circuits have become increasingly difficult to design with growing chip size, clock rates, relative wire delays and parameter variations. The globally asynchronous locally synchronous (GALS) clocking style [131] separates processing blocks such that each block is clocked by an independent clock domain and is an effective strategy to address global clock design challenges [82]. In this context, the multi-synchronous design style is as a special case of GALS systems where the system is partitioned into individual islands of synchronicity, each operating at a different frequency and connected with each other by an on-chip network which implements synchronization interfaces at island boundaries. In this section, we provide testing support for multi-synchronous networks.

Although it is an appealing alternative to the common design practice, the GALS NoC paradigm heavily impacts the architecture of the chip-wide communication infrastructure. When a multi-synchronous design style is considered then circuitry to reliably and efficiently move data across clock domain boundaries needs to be integrated into the NoC channels. For this purpose, source synchronous interfaces are adopted. Source synchronous interfaces route the source clock along with the data for correct synchronization at the receiving end. The source transmits data words without individual acknowledgments and it halts transfers when the destination indicates it can no longer accept new data via a backward propagating stall signal. We will hereafter refer to a source synchronous channel connecting two islands of synchronicity together as a bisynchronous channel.

This link architecture has a number of implications on the BIST/BISD framework for bisynchronous channel testing. First, the framework must tackle the test of a complex control logic introduced by the source synchronous interfaces. Second, the test pattern analyzer lies in a different frequency domain with respect to the test pattern generator, when the same cooperative testing strategy considered so far is selected. Since the frequency ratio of the domains can change during the NoC life-time, the test pattern analyzer has not a priori knowledge of the arrival time of the test responses. Thus, the testing framework needs to support additional mechanisms for the flow control of the test patterns.

This section presents a built-in testing and diagnosis framework for multi-synchronous NoCs. Following the implementations of the previous sections,

it exploits a cooperative testing framework redesigned under relaxed synchronizations constraints. Since the pseudo-random and the deterministic-based approaches were the best performing in the fully synchronous system then both of them are considered for the sake of comparison in the final experimental section.

5.7.1 Extension to Multisynchronous Networks

The fundamental difference between synchronous and multisynchronous systems consists of the clock domain crossing between clock domains. A clock domain may comprise a single switch (and attached IP cores) or span multiple switches. The testing methodologies presented so far can be applied in a straightforward way inside each clock domain, while the proper course of action should be taken at domain boundary. In fact, a switch tests the incoming communication channels from its north/south/west/east neighbors and local port. Since connected switches may potentially operate at different speeds, then test data may arrive at different rates from the different switch ports.

While this is not an issue for testing arbiters, since their TPGs and response analyzers are local to the switch and hence do not incur any clock domain crossing, this is a problem for the testing of all other switch sub-blocks. The key concern is the different relative speed between TPGs and response analyzers that are placed in different switches although they cooperate for the NoC testing.

Whenever comparator trees are used as response analyzers, the comparator tree needs to assess its inputs only when all switch input ports have delivered a test response, which should be in principle the same. Unfortunately, this occurs at different points in time, depending on the speed ratio between upstream switches and the local one. The workaround we propose in this manuscript consists of an asynchronous handshaking protocol regulating the transmission of test data on bisynchronous channels and synchronizing the comparison of test responses in the local switch.

Even in case MISRs are used to analyze responses at each input port, such handshaking protocol is needed. In fact, a pseudo-random sequence from an upstream TPG would be guaranteed to provide a specific coverage only under given frequency settings in the upstream and downstream switch. Should the frequency ratio change, the coverage would change as well and sequences of different lengths might be needed depending on the operating conditions. In the end, designers would have no other choice other than conservatively

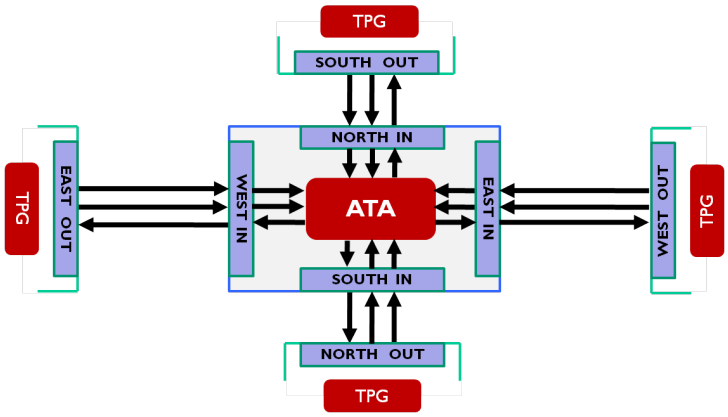


Figure 5.25: Cooperative testing framework for bisynchronous communication channels.

implementing the worst case sequence length, which is a clearly inefficient approach. Going for the asynchronous control signaling across bisynchronous channels would be the only way to get the expected coverage out of a pseudo random sequence of test patterns independently of the frequency ratio between TPGs and response analyzers.

Observe that in fine-grained multisynchronous NoCs (i.e., each switch is associated with a different frequency domain) comparator trees or MISRs can be used interchangeably, while in coarse-grained (more switches in each frequency domains) ones MISRs should be preferred, since with comparator trees even the synchronous channels should be augmented with the handshaking protocol for the sake of correct comparison with bisynchronous ones.

Finally, the proposed multisynchronous testing framework can be seen as an extension of the synchronous frameworks presented so far. In particular, it extends the synchronous framework based on pseudo-random patterns (see Section 5.5) with an extra asynchronous handshaking protocol to tackle the testing of the bisynchronous channels. It integrates TPGs for pseudo-random test patterns generation and Auto Test Analyzers (ATA) where comparator trees or MISRs are integrated to perform the test response diagnosis. Following the strategy of the synchronous mechanism, a cooperative framework is devised, such that each switch tests the block instances of its neighboring switches. Fig.5.25 illustrates the cooperative testing framework for bisynchronous communication channels. Faults in the TPG, in the output buffer, in the link and in the input buffer will be revealed in the downstream switch. Each switch ends up testing its input links, while its output links will be tested by their respective

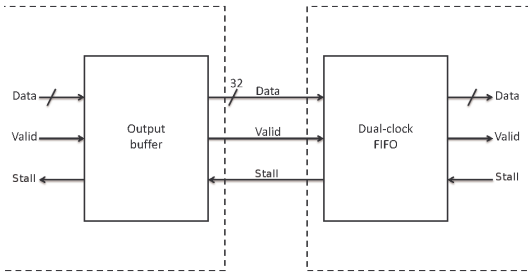


Figure 5.26: Baseline bisynchronous communication channel.

downstream switches.

5.7.2 Target GALS Architecture

The bisynchronous channel under test includes input/output buffers and their intermediate links. We consider two neighboring switches belonging to different clock domains thus the link is bisynchronous and communication in it is asynchronous. Bi-synchronous FIFO synchronizers are therefore used to connect the switches in a reliable way (see Figure 5.26).

The xpipesLite switch architecture introduced in Section 5.2 and the source-synchronous interface presented in Section 3.6 are used as baseline experimental setting to implement the multi-synchronous communication. As described in Chapter 3, the synchronizers are typically inserted between two connected network building blocks belonging to different clock domains. In practice, they break the switch-to-switch or the network interface-to-switch connections depending on the decisions about clock domain partitioning. However, there is typically no co-optimization of the synchronizer with the following/preceding NoC sub-module, therefore a significant latency, area and power overhead materializes. This design practice can be denoted as the loose coupling of synchronization interfaces with the NoC. In contrast, the bisynchronous channel under test exploits a synchronizer tightly integrated into the NoC switch taking care of synchronization but also of other tasks such as switch input buffering and flow control (see Figure 3.20). The consequent reuse of buffering resources for different purposes in turn leads to large energy savings that make a GALS NoC affordable at almost the same area and power cost of its

synchronous counterpart. Moreover, by moving the synchronizer inside the switch, the communication latency at the clock domain boundary reduces to the ideal synchronization latency. While a tightly coupled solution results extremely appealing in terms of performance, it comes with a challenging testing framework. This latter needs to test both the buffer, the synchronization and the flow control logic.

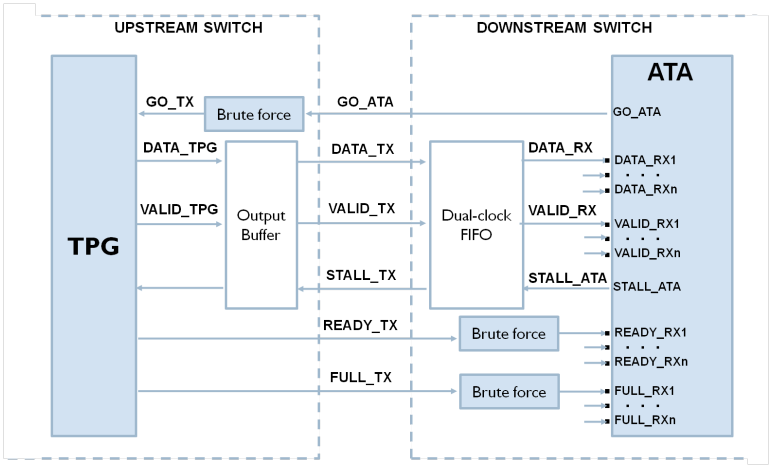
5.7.3 Bisynchronous Channel Testing

In our multi-synchronous testing framework, bisynchronous channels are tested by means of a Test Pattern Generator (TPG), an Auto Test Analyzer (ATA) and some brute force synchronizers as illustrated in Fig.5.27(a). The test phases are regulated by three asynchronous signals (*full*, *ready* and *go*). The TPG generates the test patterns for the channel while the ATA reads its responses.

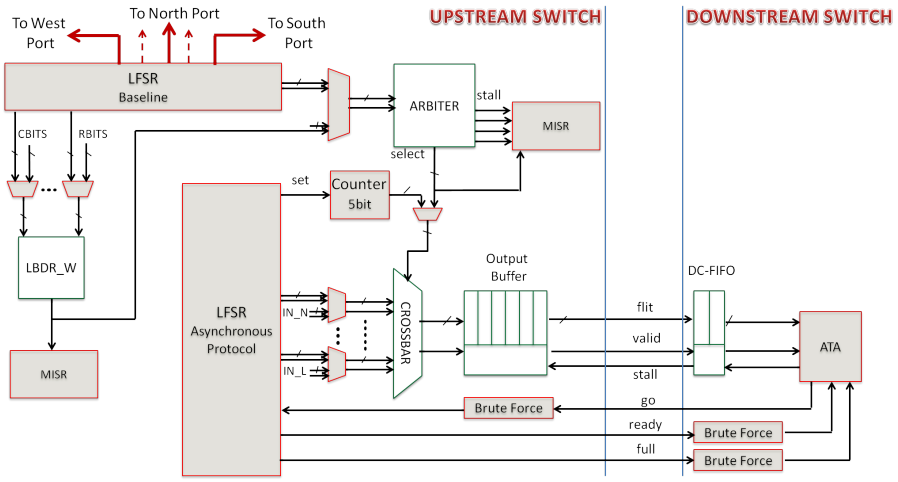
A TPG dedicated to each channel is placed at the sender side. It injects the test traffic to the channel output buffer exploiting some control signals (the *valid_TPG* and the *stall_ATA*) and communicates with the ATA through two forward signals (the *full_TX* and the *ready_TX*) and one backward signal (the *go_ATA*). These latter signals cross two frequency domains thus they are synchronized by dedicated brute force synchronizers before feeding the ATA and the TPG. At the receiver side, the ATA drives the *stall_ATA* signal and analyzes the *valid_RX* and the *data_RX* signals from the channels. It manages asynchronous control signals and implements a timer and a response analyzer.

A reliable synchronization of the *ready*, the *full* and the *go* control signals is essential for a successful communication between TPG and ATA. In our framework, we synchronize channel control signals with a three stage brute-force synchronizer, which should guarantee a sufficiently high MTTF despite the degrading resolution time constant of synchronizers with technology scaling [88].

However, the straightforward implementation of the brute-force synchronizer did not guarantee a correct TPG-to-ATA communication. In fact, an input signal asserted for few clock cycles was filtered by the synchronization circuit if the receiver domain was slower than the sender counterpart. The synchronizer was not able to sample the incoming signal if any positive receiver clock edge occurred during its assertion (see Figure 5.28(c)). In this latter scenario, the TPG/ATA missed the control signal and the testing framework operations were

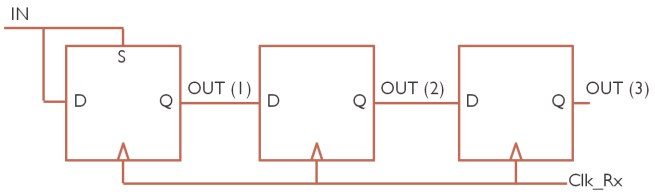


(a) Bisynchronous channel enhanced with the BIST framework.

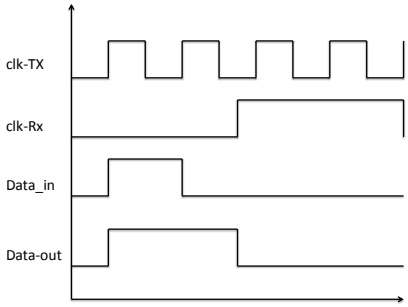


(b) BIST-enhanced switch architecture.

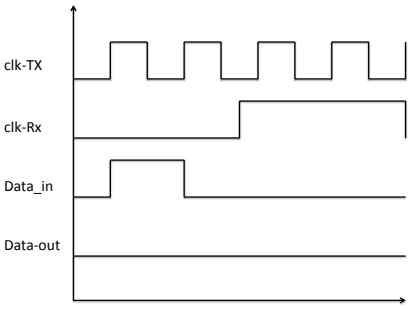
Figure 5.27: Multisynchronous testing framework.



(a)



(b)



(c)

Figure 5.28: Proposed triple-stage brute-force synchronizer (a) and waveforms of synchronizers without (b) and with (c) `set` port.

compromised. As a solution, we designed the first flip-flop of the brute-force synchronizer enabled with a set port. Thus the incoming control signal presets the first flip-flop as soon as it is asserted at the synchronizer input. As Figure 5.28(b) shows, this latter solution allows the correct control signal sampling even when the receiver is slower than the sender and a positive receiver clock edge does not occur. Afterwards, the synchronizer output is deasserted at the next positive receiver clock edge.

The brute-force synchronizers have a further objective in the testing framework. The correctness of the operation is preserved only when the *ready* and the *full* control signals arrive together with or later than the test patterns. Thus the triple-stage synchronizer avoids the arrival of those control signals in advance with respect to the test patterns. The proposed brute-force synchronizer is illustrated in Figure 5.28(a).

5.7.4 Operating Principle

The testing framework of the bisynchronous channel is designed to start at boot-time after the reset phase of the system completes. Thus both permanent faults due to fabrications defects and wear-out effects that arise during device lifetime are tackled.

At the falling edge of the reset signal, all ATAs send a *Go* signal to the TPGs integrated into the neighboring switches in order to communicate the start of the testing phase. Afterwards, each ATA enters into a wait state and when the *Go* signal reaches the receiver (after being synchronized by means of the brute-force synchronizer) then the TPG starts the injection of the first test pattern. The patterns are composed of 32 bits of data, the *valid* and the *full* signal. When the TPG injects the pattern then it asserts the *ready* and enters an idle state waiting for the next *go* assertion by the ATA. During the idle state, the TPG deasserts the *valid* signal.

Each ATA enables a timer when it sends the *go* signal. Therefore the timer generates a timeout flag if the *ready* signal is not received within a maximum period of time, indicating that the communication channel is unreliable.

On the contrary, when the pattern arrives, then the diagnosis phase can start. In this case, the incoming *full* signal is exploited by the ATA to drive the *stall* signal of the dc-FIFO for a single clock cycle. The *full* signal has to be considered as an integral part of the test pattern and is used by the TPG to randomly span the FSM states of the input buffer by making its input stall signal controllable.

The diagnosis procedure then depends on the chosen response analysis strategy. In case of a MISR, the buffer output is sampled by the MISR one cycle after the arrival of the synchronized *ready* signal.

In case of comparator trees, port-ATAs can be merged into a single ATA (this is the case of Fig.5.27(a)) since operations on each port need to be synchronized. The incoming full signals are stored by the ATA and applied to the dc-FIFOs as stall signals only when all test patterns have been received (i.e., all synchronized *ready* signals on each port have been sampled). Thus, test responses are compared between each other in the global comparator tree and the diagnosis is performed. Interestingly the dc-FIFO *stall* signals are driven high by default (forcing a stall condition) if the ATA is not evaluating the channel test responses in order to freeze the channel state avoiding misalignments between the other channel states of the same switch.

In all cases, after a response analysis step the ATA resets its timer and asserts the *go* signal to the TPG to allow the dispatch of the next test pattern. When an error is revealed, the information about the failure is stored in a local register and the faulty channel output is no longer considered at the test restart. Figure 5.29 models the above mentioned procedure in a purely illustrative scenario where an ATA collects the test responses from a slower (*TX*2) and a faster (*TX*1) bisynchronous channel. Comparator trees are assumed for test response analysis. For this reason, the stall signal (a go condition in this case) is applied to the two dc-FIFOs only when both synchronized *ready* signals are sampled by the unified switch ATA.

5.7.5 BIST-Enhanced Switch Architecture in a Multisynchronous Scenario

The switch architecture enriched with the BIST infrastructure is illustrated in Fig.5.27(b). Similarly to the synchronous scenario, test wrapper consisting of multiplexers enables test pattern injection and the test of every switch port is performed in parallel. However while a unique 34 bits LFSR generates the pseudo-random patterns in the synchronous scenario, multiple LFSRs are required in a multisynchronous environment. In fact, 5 dedicated LFSRs to each switch channel are required to support the asynchronous handshaking protocol. The internal switch sub-blocks of the control-path (i.e. arbiters and LBDRs) can still exploit a single LFSR shared among all the instances. Finally the whole switch comes with 6 LFSRs.

In the multisynchronous solution, the LBDR is tested in isolation without be-

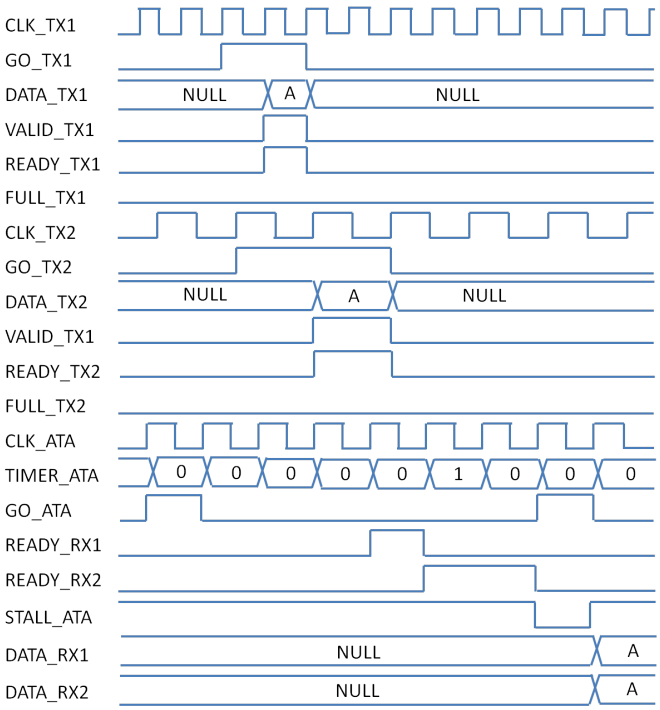


Figure 5.29: Bisynchronous channel operating principle.

ing fed by the channel test responses. In such a way, the LBDR testing was unrelated to the asynchronous handshaking of the channel increasing the final LBDR coverage. Clearly, this latter strategy requires additional resources for the sake of the diagnosis. This latter can be performed implementing response analyzers either with MISRs or with comparator trees. To note that the testing of the arbiter was not modify with respect to the synchronous solution and the multiplexers of the crossbar are cascaded to the channel testing.

5.7.6 Experimental Results

We performed placement-aware logic synthesis of a 5x5 switch on an industrial 40nm technology library. The baseline switch architecture of Fig.5.1 and the proposed switch augmented with the multisynchronous testing framework gave approximately the same maximum operating speed of 600 MHz when synthesized for maximum performance, thus proving that asynchronous testing extensions are decoupled from normal operation and do not impact the

at-speed testing capability of the switch.

In the following characterization experiments, two fully synchronous NoC testing frameworks are considered for the sake of comparison: the framework based on pseudo-random test patterns of Section 5.5 and the framework based on handcrafted test patterns of Section 5.3.

Coverage Vs Test Pattern

Fig.5.30(a) reports the total number of test patterns generated for the 5x5 switch and the associated coverage for both the deterministic and the pseudo-random testing framework, this latter in its fully synchronous and bisynchronous variants. It can be seen that the proposed synchronous pseudo-random framework exploits a further degree of freedom with respect to the deterministic solution. In fact, it can trade latency for coverage without affecting the test pattern generator design. The deterministic framework achieves 99.3% coverage in 1104 clock cycles while the coverage of the proposed fully-synchronous framework ranges between 94.2% and 98.2% achieved in 500 and 10.000 test patterns respectively.

As regards the multisynchronous testing framework, we notice that a high coverage ($\sim$98%) is achieved with few test patterns ($\sim$500). On the other hand, the coverage remains approximately constant with the injection of further test patterns. Interestingly, the bisynchronous framework performs better than the fully-synchronous counterpart when few test patterns are considered while the gap between the two frameworks is closed when a high amount of patterns is injected. This effect is mainly due to the high coverage achieved for the dc-FIFO of the bisynchronous channel. This latter result confirms the efficiency of the pseudo-random test patterns when applied to buffer-intensive data paths. Finally, the dc-FIFO comes with a relevant area footprint thus its coverage contributes to increase the total coverage of the switch.

Area Overhead

Fig.5.30(b) shows the area overhead of the considered testing frameworks for the 5x5 switch with respect to its baseline BIST-less implementation. The area overhead of the proposed fully-synchronous framework based on pseudo-random patterns is 16.3%, which raises to 37.1% for the framework based on deterministic patterns. The test wrappers and the TPGs (i.e. the LFSRs) of the pseudo-random based synchronous solution prove extremely lightweight when

compared with the deterministic framework counterpart.

When the bisynchronous framework is considered then the area overhead hits the 27.9% since the contribution of LFSRs to switch area footprint is no longer negligible. In fact, LFSRs dedicated to each switch port were required in the multisynchronous solution to support the asynchronous handshaking, as opposed to a unified LFSR per switch in the fully synchronous scenario.

To note that most of the comparator trees area of the fully-synchronous framework is replaced by the ATA module taking care of test response analysis in the bisynchronous channel for the cascaded crossbar multiplexer and channel link.

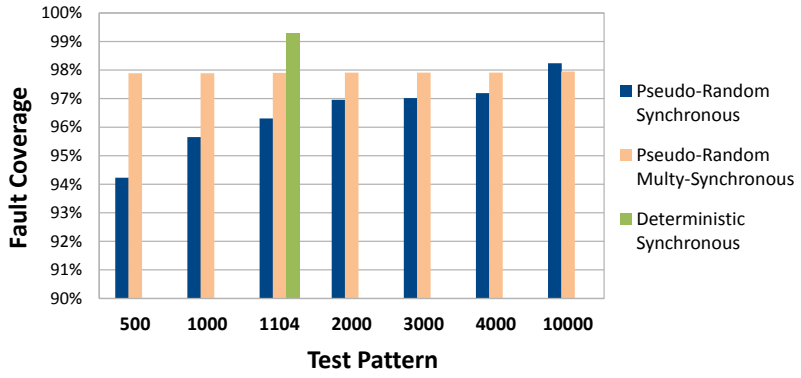
In all architectures, we have experimentally verified that the use of MISRs in place of comparator trees for response analysis only marginally affects switch area figures and their breakdown, hence such results are not reported for lack of space.

Latency

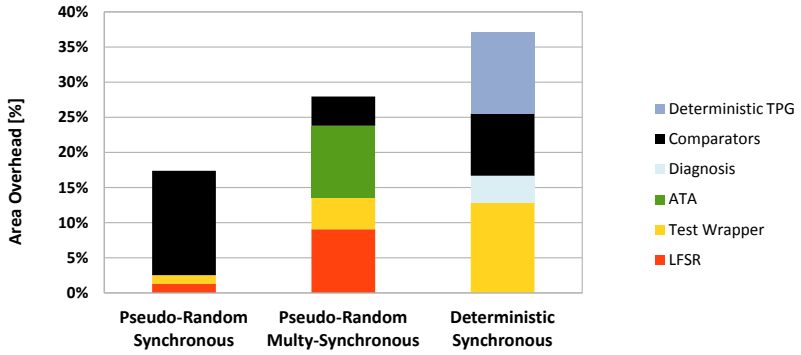
In the fully-synchronous frameworks a test pattern is injected every clock cycle while in the bisynchronous solution some latency overhead is introduced by the asynchronous protocol. As a result, the number of clock cycles required for testing the fully-synchronous switches is equal to the number of injected test patterns. On the contrary, the test of the switches in the multisynchronous scenario does not only depend on the number of test patterns but also on additional parameters, such as the receiver/sender frequency ratio. Figure 5.31 shows the test time as a function of the TPG (Test Pattern Generator) and ATA (Auto-Test Analyzer) operating frequencies. As expected, when the frequencies of the TPG and the ATA are both low (as an example, 10MHz), the test time is very high. Vice versa, when the frequencies are both high (as an example, 1GHz), the test time is the smallest. In the case that only one of the frequencies is low than this latter dictates the final test time.

Comparison with State-of-the-Art Framework

The results achieved by the proposed multisynchronous testing framework compares favorably with previous work in [154]. In fact, the proposed framework introduces almost half of the area overhead required by the counterpart (50% of area overhead is required by [154] when the BIST-enhanced channel is considered with respect to the baseline one without testing support).



(a)



(b)

Figure 5.30: Single stuck-at faults coverage as a function of test pattern count (a) and area overhead for BIST implementation (b).

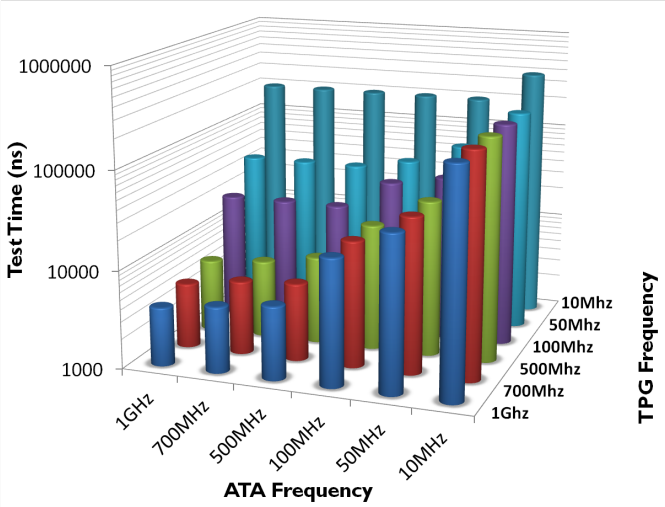


Figure 5.31: Bisynchronous framework test time as function of ATA and TPG frequencies.

On the contrary, coverage and latency are not directly comparable. In fact, they depend on the number of test patterns which is strictly associated with the microarchitecture under test. Especially, [154] implements both output buffers and dc-FIFOs with 4 slots while we exploit 6 slots output buffers [110] and 5 slots dc-FIFOs to guarantee the highest throughput at runtime despite of the testing complexity. The work in [154] took the opposite approach. Moreover, our and [154]'s dc-FIFOs are implemented by means of different architectures (namely [137] and [155] respectively).

However, some key considerations can be drawn just at the same. When a standalone channel is considered, the latency of the two frameworks under test scales in the same way with varying transmitter/receiver frequency ratios. Differently, when the entire network is analyzed then our approach delivers all the network diagnosis information at the end of the test due to the switch cooperation strategy while each channel in [154] completes independently the test. Anyway, the two frameworks require an equal number of clock cycles to test the whole network since the total test time in both solutions is ultimately dictated by the slowest frequency domain. We can get a similar mechanism by using MISRs in place of comparator trees in our test architecture, which therefore turns out to be extremely flexible. Overall, our approach saves half of the area overhead introduced by [154] while providing the same network test

latency and latency scalability with operating speeds of NoC clock domains.

5.8 Conclusions

Modern Network-on-Chip should be integrated into a reliable framework taking care of fault detection, diagnosis and network reconfiguration. In this scenario, external testers for nanoscale chip testing face severe concerns: lack of scalability of test data volumes, high cost for full clock speed testing, poor suitability for the extension of production testing to lifetime testing. As an effect, a migration from external testers to built-in self-test (BIST) infrastructures becomes a must.

In this direction, this chapter presented four scalable built-in self-test and self-diagnosis infrastructures for NoCs providing a wide exploration of different testing strategies. Table-less logic based distributed routing was the foundation of all our approaches, and enabled network reconfiguration with only 10 diagnosis bits per switch. Customizations for the NoC environment of conventional testing strategies were proposed taking full advantage of the intrinsic network structural redundancy through cooperative testing and diagnosis frameworks:

- First, NoC switch was enhanced with a conventional scan chain-based mechanism. When the scan chain was automatically implemented by means of synthesis tool then a huge amount of test patterns was required to support the *load* and *unload* of the chains. Thus, the switch area overhead was not affordable due to the size of the built-in TPG.

  Therefore, a novel customized framework was envisioned to reduce the test pattern number. Dedicated scan chains were reserved for each switch port. Then the same test patterns could be reused for multiple scan chains cutting down the TPG area footprint. Although 5 times less test patterns were required for a 5x5 switch, the area overhead was still higher than the 130%. As a result the scan-based approach proved unsuitable for use within a built-in self-testing strategy in NoCs.

- Second, the switch was tested by means of conventional pseudo-random patterns generated by a built-in LFSR module. The solution materializes relevant area savings and enhanced flexibility. However, pseudo-random patterns suffer from low coverage on the control path. The coverage only increases logarithmically with the number of clock cycles. A low coverage is achieved with a high testing latency.

In order to tackle the intrinsic drawbacks of the pseudo-random patterns, test responses of switch sub-blocks were reused and test pattern optimizations were introduced. Finally the 96% of switch coverage was reached in 2000 clock cycles with a 16% of area overhead. The pseudo-random pattern framework proved an appealing solution for a low-area highly-flexible NoC solution, provided the devised customizations are applied.

- Third, a low-latency alternative to the pseudo-random approach was proposed. Differently from the previous solutions, the framework relied on a non-conventional strategy based on deterministic test patterns. The test patterns were handcrafted exploiting the knowledge of the device under test. Thus a coverage close to 100% was achieved by means of few test patterns (i.e. 1104 clock cycles). Clearly, these latter results were achieved at the cost of lower flexibility and higher area overhead (37%) with respect to the pseudo-random counterpart. As a conclusion, the deterministic test pattern-based strategy represents the best solution for high-performance high-reliability NoCs.

- Finally, we proposed one of the first BIST/BISD framework for GALS Network-on-Chips. In particular, we extended the proposed pseudo-random pattern framework through an asynchronous handshaking protocol on bisynchronous channels. We exploited the outcome of the previous testing explorations to achieve a high coverage and a low area overhead by means of cooperative diagnosis and efficient test strategies. The final implementation is more area-efficient than state-of-the-art solutions while providing comparable performance, thus paving the way for testable on-chip networks in mixed timing nanoscale integrated systems.

The testing exploration performed in this chapter will drive the implementation of the final fault-tolerant and self-testable GP-NaNoC switch for next-generation embedded systems that will be presented in Section 7.

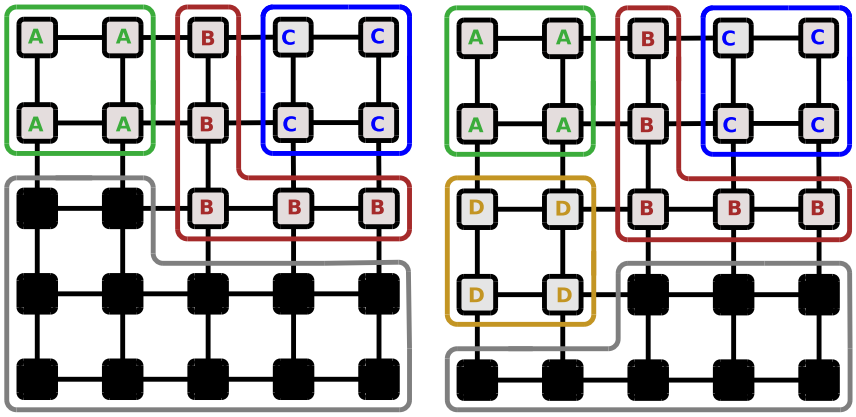
6

OSR-Lite: NoC Reconfiguration Framework

CURRENT and future on-chip networks will feature an enhanced degree of reconfigurability. Power management and virtualization strategies as well as the need to survive to the progressive onset of wear-out faults are root causes for that. In all these cases, a non-intrusive and efficient reconfiguration method is needed to allow the network to function uninterruptedly over the course of the reconfiguration process while remaining deadlock-free. The work presented in this chapter is inspired by the overlapped static reconfiguration (OSR) protocol developed for off-chip networks. However, in its native form its implementation in NoCs is out-of-reach. Therefore, we provide a careful engineering of the NoC switch architecture and of the system-level infrastructure to support a cost-effective, complete and transparent reconfiguration process. Performance during the reconfiguration process is not affected and implementation costs (critical path and area overhead) are proved to be fully affordable for a constrained system. The chapter will provide the implementation details for the proposed reconfiguration protocol and present both high-level and synthesis results.

6.1 Introduction

The new mobile usage models that are coming about require the execution of multiple use cases on the same device while optimizing resource consumption for each of them. On the other hand, the hardware and software design convergence in todays complex embedded systems call for an upgrade of architecture building blocks in the direction of runtime reconfigurability and adaptivity. As a result, applications should be able to frequently reconfigure the underlying



(a) Configuration A (b) Configuration B

Figure 6.1: Two NoC configurations where the routing algorithm needs to be adapted.

hardware platform on-the-fly and in a cost-effective way, thus selecting the most convenient operating point that suits their needs and allows an efficient use of system resources.

Modern multi-core integrated systems achieve scalable computation horsepower and power efficiency by integrating a large number of processing cores on the same silicon die. This trend is unmistakable since current products already include tens and even hundreds of processing cores, such as the Tilera multicore processor [25]. In this context, on-chip interconnection networks (networks-on-chip, NoCs) are typically used to provide communication parallelism and the reference integration infrastructure for the whole system.

To address the new functionalities, the NoC must be enriched with an efficient reconfiguration process which enables the smooth and transparent transition between system configurations. For instance, Figure 6.1 shows two different configurations of a multicore system over time. In the first one (configuration A) different applications are mapped to the NoC nodes and execute concurrently, while other resources are powered down. Later, the resource manager may trigger a chip reconfiguration to power on unused resources and thus activate a new application (configuration B).

The transition between configurations needs a careful design of the NoC routing algorithm, which establishes the paths for every packet in the network. At each configuration a different routing algorithm is needed. In both cases, the algorithm must be deadlock-free (should not introduce cycles in its channel

dependency graph). However, in the transition between configurations, both algorithms can induce extra dependencies that lead to deadlock.

Therefore, in order to migrate from one configuration to the other, one possible approach is to drain the network, then changing the routing algorithm to the new one and finally resuming traffic injection with the new algorithm. This is the case of the so called traditional static reconfiguration (TSR). In this case system performance is likely to be heavily impacted by the reconfiguration process due to the temporarily low resource utilization. Alternatively, the network can be dynamically reconfigured, in the sense that traffic is not stopped during the reconfiguration process, but an effort is needed to avoid deadlock situations. This is typically achieved by devoting extra resources to the network. We refer to this case as the dynamic reconfiguration.

In this chapter we advance state-of-the-art in reconfiguration frameworks for NoC-based systems. However, instead of designing a brand new reconfiguration mechanism, we recognize the large amount of bibliography and proposals made for reconfiguration mechanisms in high-performance off-chip networks. In this sense, we pick the approach that better suits the NoC domain and the tight resource budgets of the on-chip environment.

The Overlapping Static Reconfiguration process (OSR) in [48] enables a transparent system reconfiguration process. However, in [48] only the protocol was described while at the same time highlighting the key architectural requirements to properly support it (namely virtual channels, routing tables, event notification, involvement of end-nodes in the reconfiguration process). Unfortunately, no practical implementation insights were provided, thus raising the reader's skepticism on the applicability of OSR to an on-chip setting.

In this chapter we report on the first-time implementation of the native OSR protocol in an on-chip network, proving that the needed network overprovisioning is such to make the protocol not viable in practice. As a consequence, the thesis targets the modification of OSR to better match the requirements of the resource-constrained NoC setting, thus resulting into the OSR-Lite framework. Such modifications concern both selected protocol features (without giving up the goodness of the underlying idea) and relevant implementation techniques.

With OSR-Lite in place, it is possible to reconfigure a whole 64-node network in a few hundreds of cycles, enabling the entire and transparent transition between any pair of independent and unrelated configurations. Moreover, this is achieved with no impact on network latency and with no impact on switch delay. The reconfiguration performance of OSR-Lite makes it the enabling

tool for planned reconfigurations in multicore systems. The following specific scenarios can be therefore materialized by the outcome of this work:

- *Virtualization of the system*. Our method enables the runtime division of the entire network into sets of virtual regions for assignment to different applications running concurrently.
- *Power management*. The reconfiguration mechanism can be exploited for powering down unused resources; such functionality becomes compulsory to keep power consumption levels to reasonable bounds.
- *Reliability*. When a NoC is augmented with transient fault tolerance, then this kind of faults can be tolerated without any loss of information. However, intermittent faults are likely to be an indicator of the gradual onset of a permanent fault (typically, a wear-out fault). In this case, OSR-Lite can be used to reconfigure the network so to exclude the affected link/switch component, before the permanent fault shows up and causes packet loss.

The rest of the chapter is organized as follows. In Section 6.2 the OSR technique is briefly described. Section 6.3 focuses on the OSR-Lite proposal and shows the reconfiguration steps performed in an OSR-Lite environment. Section 6.4 deals with implementation issues by describing how the mechanism affects the micro-architecture of the switch sub-modules. Then, Sections 6.5 deal with high-level results thus it shows the time propagation of OSR-Lite and the average message latency during reconfiguration. Section 6.6 proposes area and routing delay results of the mechanism. The chapter is concluded in Section 6.7 with conclusions and future work.

6.2 Native OSR technique

Typically, a routing algorithm is deadlock-free when its channel dependency graph (CDG) is acyclic (we do not consider fully adaptive routing algorithms). The CDG is set by representing the resources of the network by vertices (mainly the buffers associated with the ports of each switch) and the dependencies between two resources by arcs. There is a dependency between two resources r_1 and r_2 if a message can use r_1 and request r_2.

Two routing algorithms R_1 and R_2 are deadlock-free when they have an acyclic channel dependency graph. However, when using both algorithms at the same

time new extra dependencies are induced potentially leading to deadlock. This can be seen in Figure 6.2 where a cycle is formed when using two routing algorithms (*XY* and *YX*) at the same time. During a reconfiguration process we refer to R_{old} as the old routing function and R_{new} as the new routing function. Similarly, packets routed with R_{old} will be referred to as old packets and packets routed with R_{new} will be referred to as new packets.

The native OSR method is based on the fact that those cycles are created only when old packets using R_{old} are routed after new messages using R_{new}. If old packets are guaranteed to never go behind new packets the extra dependencies do not occur in practice and then no deadlock can be formed. Indeed, in a static reconfiguration process the entire network is drained thus guaranteeing old packets will never go behind new ones.

OSR is a static reconfiguration process but localized at link/router level, and not at network level. Indeed, it guarantees that new packets are only forwarded via links that have been drained from old packets. This is achieved by triggering a token that separates old packets from new packets. The token is triggered by all the end nodes and tokens advance through the network hop by hop. Indeed, tokens follow the CDG of the old routing function, draining the network from old packets. However, in contrast with static reconfiguration, the new packets can enter the network at routers where the token already passed. Figure 6.3 shows the complete native OSR mechanism, involving a central manager. In a first step, a reconfiguration action is triggered, either by the detection of a malfunctioning component or by a higher level manager in the system stack requiring a reconfiguration, e.g. a new application is admitted. In any case, when needed the central manager may receive event notifications through the

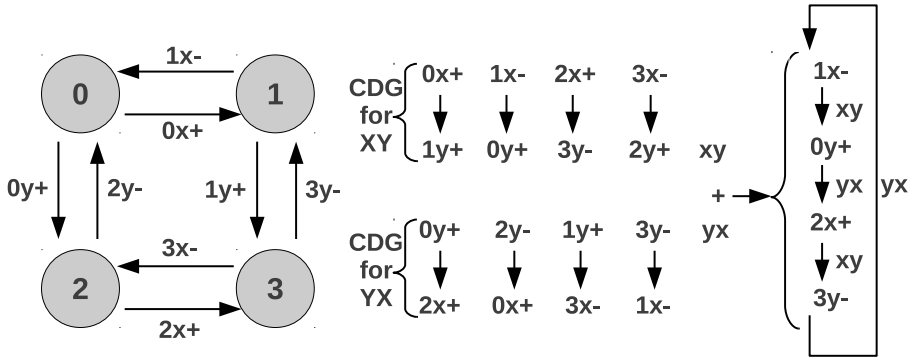


Figure 6.2: Channel dependency graph for two routing algorithms and the combination of both.

network (step 1). Then, in step 2, the new algorithm for the new configuration is computed by the central manager. The resulting information is disseminated to all the switches in step 3. In step 4 the end nodes trigger the token and the OSR reconfiguration spreads throughout the network (step 5).

Figure 6.4 shows how tokens advance in a network. At a given output port, a token is triggered to the next downstream router indicating the output port has been drained from old packets. This is guaranteed when the token has been received through all the input ports of the switch that have old (R_{old}) output dependencies with the output port. These port dependencies can be extracted from the R_{old} routing algorithm. However, how to perform this is not explained in [48], although it is key to obtaining an efficient implementation.

Notice that the token divides two epochs in the network, the old epoch (when packets are routed with the R_{old} routing function) and the new epoch (when packets are routed with the R_{new} routing function).

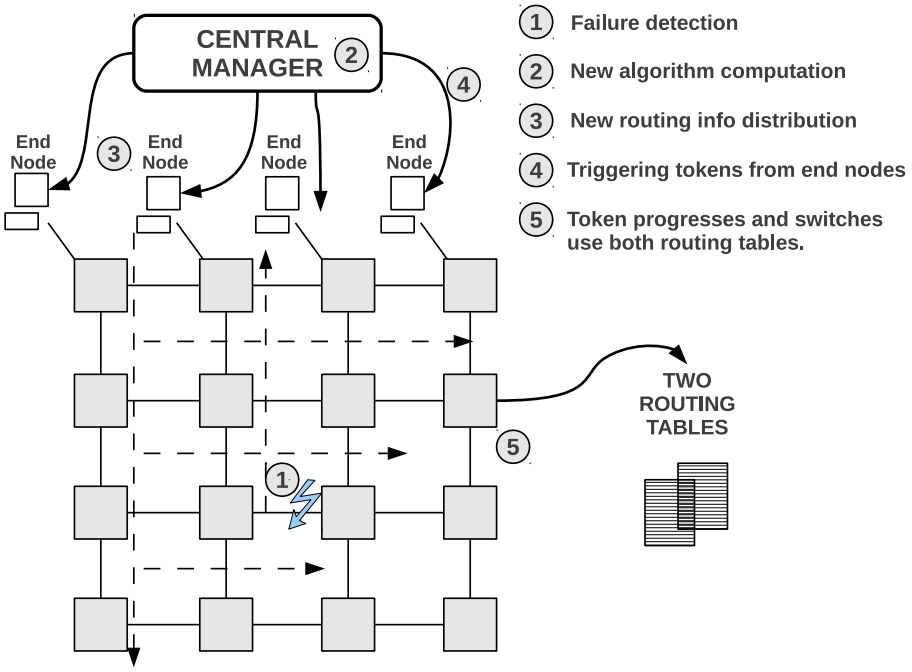


Figure 6.3: Reconfiguration steps performed in an OSR environment.

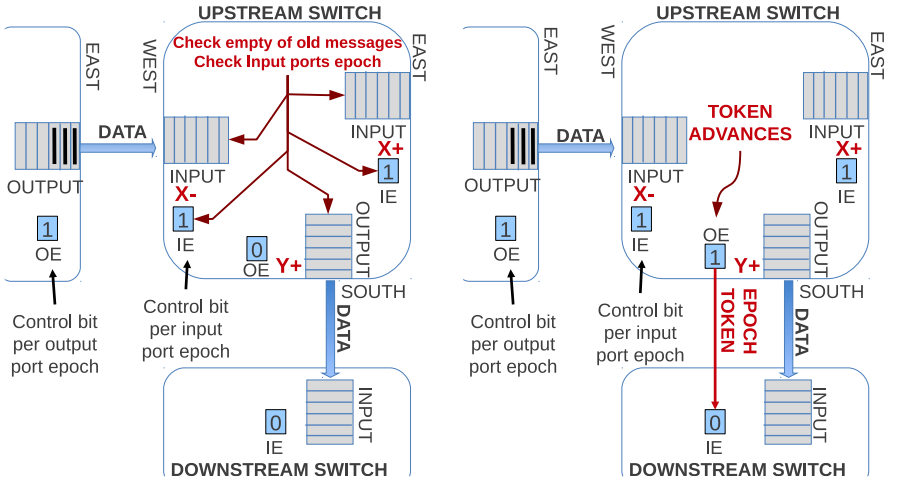


Figure 6.4: Token advance in a network: (a) check for absence of old messages and input ports epoch, (b) token signal propagation. The token separates old traffic from new traffic.

6.3 OSR-Lite

The OSR mechanism needs to be modified in order to better suit the NoC environment so to become an efficient and plausible mechanism for planned reconfigurations. Indeed, the main issues addressed in this chapter are the following:

- *Codification of the routing information.* During the reconfiguration process both routing algorithms coexist at the same time at routers. This means resources need to be sized for both algorithms. In OSR, routing tables were used to store the routing info. In NoCs, however, routing tables are an expensive resource in terms of access time, area, and power consumption. Therefore, hosting two routing tables per switch input port does not appear to be a cost-effective solution for OSR-Lite.

- *Control virtual channel (VC) used in OSR.* Different actions (sending routing information to routers, triggering the reconfiguration process) are performed during the OSR reconfiguration which imply the exchange of information between a central manager and the routers or the

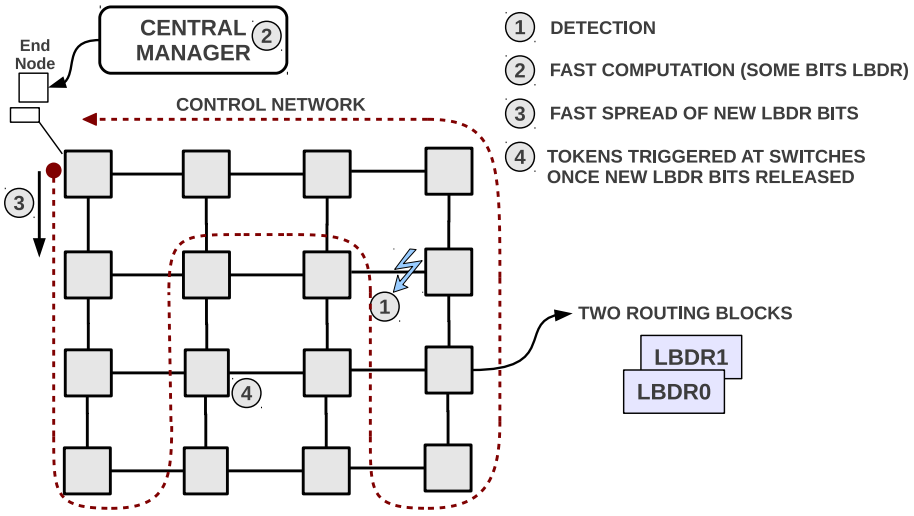


Figure 6.5: Reconfiguration steps performed in an OSR-Lite environment.

endnodes. In [48] this was implemented by means of a control VC. Unfortunately, using VCs only for that purpose has a large impact on router implementation (will be seen later) and is not fully justified in an on-chip.

- *Reliable control VC assumed in OSR.* A different (spanning-tree) algorithm is assumed in OSR to effectively route control packets through the control VC.

- *Involvement of end nodes in the reconfiguration process.* In OSR the end nodes were notified to trigger the reconfiguration. This is done by end nodes injecting the token directly as a new packet. In NoCs, reaching the end nodes via dedicated packets from the central manager would be a time-consuming course of action. In order to cut down on the reconfiguration latency, involving only switches and not endnodes in the reconfiguration would be an appealing property in a NoC setting.

In order to address all these issues, we propose the OSR-Lite approach. Figure 6.5 shows all the steps and the main modifications performed. In particular, we exploit a control network through which routers can inform about expected topology changes (e.g., an output link is having frequent transient failures and is going to fail soon, or a region of the NoC is overheated and needs to be powered down). The control network collects all the notification events and

sends them to a central manager (step 1). If the reconfiguration is instead initiated by a resource manager in the context of power management or virtualization strategies, step 1 can be skipped. The central manager then computes the new configuration (step 2) and disseminates the new routing information to the switches (step 3). Then, every switch starts the OSR-Lite reconfiguration process in step 4. Notice that end nodes are not involved in the reconfiguration process.

The control network can be used also in step 3 for routing bit dissemination to the switches. In [156], we have presented the design of a dual network for switch-to-global manager bidirectional signaling, thus offloading critical control tasks from the main data network. In that work, the dual network was used to notify diagnosis information to the manager following the main NoC testing phase, and to notify configuration bits of the routing mechanism to the switches. The same network could be reused for other purposes, such as congestion management, deadlock recovery and software debugging. In [156] it is showed to be a cost-effective solution for control signaling, which can be easily and effectively made reliable through a combination of fault-tolerant and online testing strategies. For this reason, this work relies on such a fault-tolerant dual network to convey control information of the reconfiguration process. Furthermore, [156] also reports an efficient computation algorithm that comes up with the routing configuration bits of a new network partitioning or topology shape. This is the algorithm the controller runs in step 2. Given that the control network and the computation algorithm are covered by previous work, from now on we focus on the core reconfiguration process of the network and on the microarchitectural support for that. The reader should keep in mind that all these mechanisms will work together in the complete reconfiguration framework. In the next section we describe the router implementation in more detail.

6.4 OSR-Lite implementation

Without lack of generality, we use the xpipesLite switch architecture [110] already illustrated in Figure 5.1 to prove viability of our OSR-Lite mechanism. The switch implements both input and output buffering, relies on wormhole switching and on a stall/go flow control protocol. As described in Section 5.2, the switch architecture is extremely modular and implements logic-based distributed routing (LBDR) [152]: instead of relying on routing tables, each switch has simple combinational logic that computes target output ports from

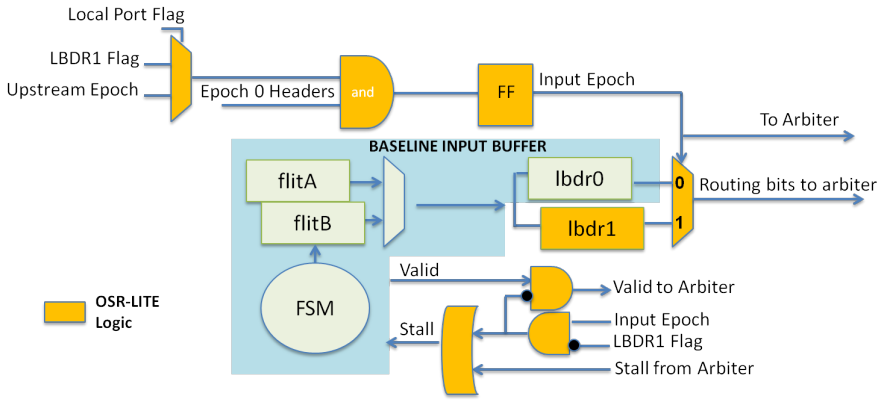


Figure 6.6: Switch input buffer enhanced with the OSR-Lite logic and a new set of routing mechanism.

packet destinations. The support for different routing algorithms and topology shapes is achieved by means of 16 configuration bits for the routing mechanism of the switch (hereafter denoted as LBDR bits). LBDR bits carry the routing algorithm information (expressed in terms of routing restrictions), the connectivity information of switch output ports and special detour bits. Such bits make LBDR a flexible routing mechanism while at the same time significantly cutting down on the memory requirements of routing tables. LBDR bits are computed by a central NoC manager and disseminated to the switch input ports through the dual control network. Indeed, two sets of LBDR bits are allocated at each router for OSR-Lite. Upon receiving the new routing bits, a router triggers the reconfiguration process by auto-generating initial tokens at its local input port (port connected to an end node) and processing the tokens accordingly.

The logic enabling the OSR-Lite mechanism was integrated into the above mentioned baseline switch taking care to preserve its modularity together with its performance. Thus, the OSR-Lite logic was designed in new modules plugged into the switch without affecting the existing blocks. Moreover, the new modules were instantiated for each switch port following the modularity of the baseline blocks (the OSR-Lite mechanism can be extended for switches of every arity by means of simple logic replication).

6.4.1 OSR-Lite at the Input Ports

As a first step, the baseline switch was enhanced with a second routing logic unit (LBDR1) collecting the new routing info coming from the central manager. This unit is connected to the input buffer as the baseline LBDR0 block (see Figure 6.6) although is used exclusively for routing packets in the new epoch (new packets). The switch arbiters need to select the routing info from the appropriate routing logic block (either LBDR0 or LBDR1). This is obtained from a multiplexer configured by the current epoch of the input port (in a flip-flop). In order to reduce the reconfiguration latency, the input port evolves to the new epoch as soon as there are no stored header flits at the input port with the epoch bit set to zero (*Epoch 0 headers* signal) and the token has been received from the upstream switch (*upstream epoch* signal). Notice that in the case of the ports connected to end node (*local* port; *local port* flag), the token is assumed to arrive with the arrival of the new configuration bits (*LBDR1 flag*). In this case, the header flits located in the buffers are considered of the new epoch when the new configuration bits have arrived and the routing mechanism (LBDR1) is set. Notice that local ports do not introduce dependencies between channels that may lead to deadlocks, therefore is safe to assume all the injected flits as belonging to the new routing function. To notice that the token propagation will always start from local ports at switches, not involving end nodes.

The number of flit headers to be routed by LBDR0 and stored in the buffer is detected by a 2 bits counter monitoring the incoming and outgoing headers of the input buffer module. The counter increases its value when a header is accepted and the incoming token is low and decreases its value when a header is sent. In order to preserve the max performance of the baseline switch, sequential logic stages were exploited to avoid impacting the critical path in the OSR-Lite mechanism.

Notice that the implementation prevents possible race conditions from occurring. For instance, a token may be received from the upstream switch before the new routing bits are received. In that case, the header flits in the input buffers are stalled and declared not valid to the internal switch logic until LBDR1 is set.

6.4.2 OSR-Lite at the Arbiters

OSR-Lite requires a lightweight new module plugged around the baseline arbiters. The logic is reported in Figure 6.7. Basically, a set of *AND*/*OR* logic

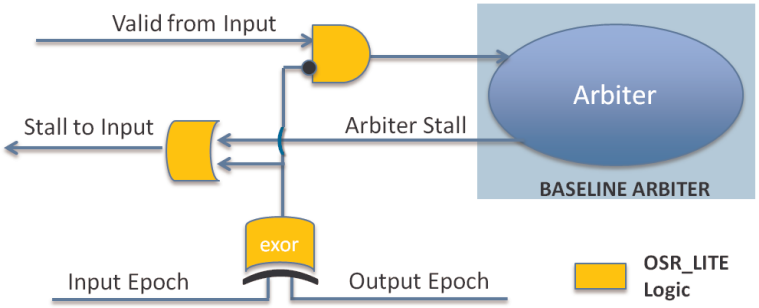


Figure 6.7: Switch arbiter enhanced with the OSR-Lite logic.

blocks together with a set of *EXOR* blocks allow the arbiter to process an incoming header exclusively when the epoch of the switch input port is the same as the one of the destination output port. On the contrary, a packet residing in an input port with the new epoch is stalled until the output port evolves to the new epoch (guaranteeing old packets go first and then new packets).

6.4.3 OSR-Lite at the Output Ports

Concerning the output port, an output port evolves to the new epoch when all the input ports with output dependencies to this output port have evolved to

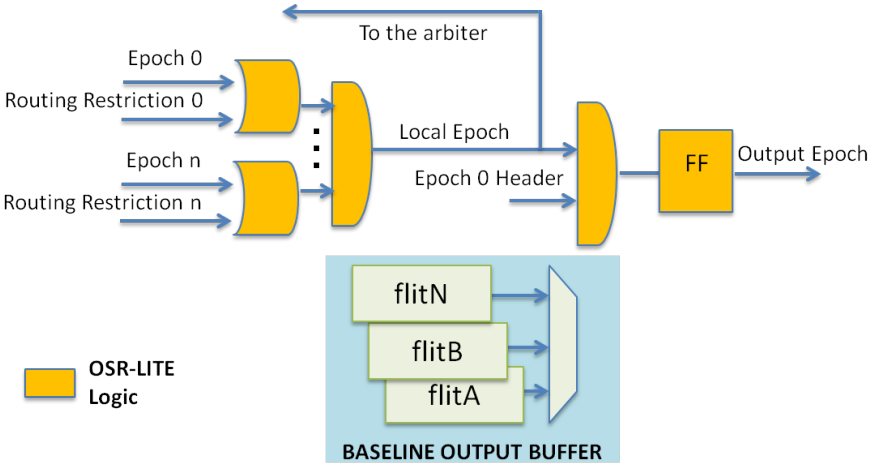


Figure 6.8: Switch output buffer enhanced with the OSR-Lite logic.

the new epoch. In order to efficiently deal with the dependencies, OSR-Lite takes profit of the routing bits used in LBDR. Routing bits indicate the routing restrictions that exist at neighboring switches. Therefore, they can be seen also as channel dependencies. If the R_{xy} bit is set it means that there is a link dependency between the output port x and the output port y at the next switch. On the contrary, if the bit is reset it means there is no dependency and in that case we can safely assume no packets will come through the port x requesting output port y. Therefore, the output port needs to receive both the epochs of the input ports and the routing restrictions located at the neighboring switches. The mechanism is enabled by a set of *OR* blocks (each of them belonging to a different input port) followed by an *AND* block, as represented in Figure 6.8.

In contrast with the baseline OSR technique (where the routing restriction information was saved in the routing table), the OSR-Lite mechanism needs to obtain channel dependencies from the routing logic located at neighbor switches. As a result, three additional routing bits are sent by the LBDR0 logic of the upstream switch together with the token bit. To note that LBDR0 received its routing bits information through the control network in an earlier configuration stage. In addition, the input port needs to send the incoming routing restriction signals to the appropriate output ports. Thus every link is extended by 4 additional wires (i.e. 1 token wire + 3 routing restriction wires). See Figure 6.9.

Finally, the token is sent by the output port to the downstream switch when all the input ports with dependencies with the output port have evolved to the new epoch, meaning all these input ports have drained all the old packets from their buffers (see the *LocalEpoch* signal in Figure 6.8).

Once the network has completely migrated to Epoch 1, the central manager can safely fill LBDR0 bits with a copy of LBDR1 bits, and instruct all the switches to safely swap to Epoch0 again. This allows for the system to be ready in few cycles for a new reconfiguration process.

6.5 System-Level Evaluation

In this section, we evaluate OSR-Lite. First, we show how the OSR-Lite propagates over the network. Then, we evaluate the reconfiguration time overhead under different injection rates using synthetic traffic. Moreover, we compare the proposed reconfiguration with a static reconfiguration process in terms of network latency.

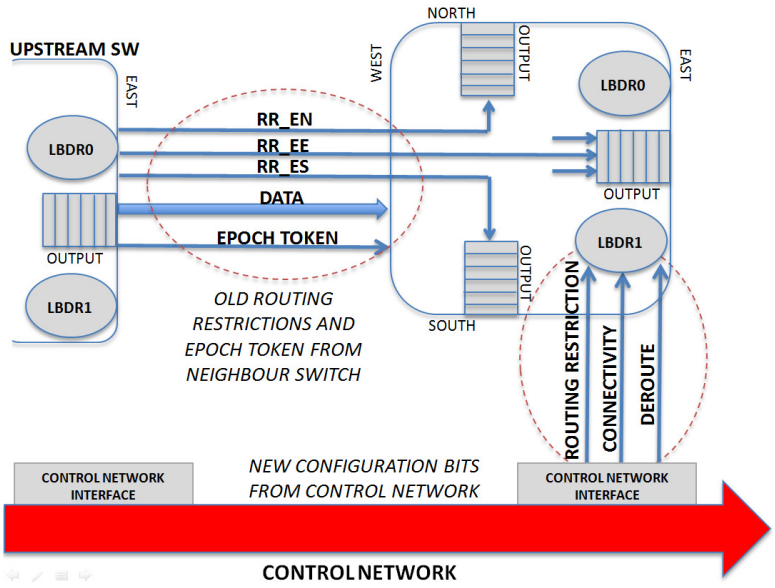


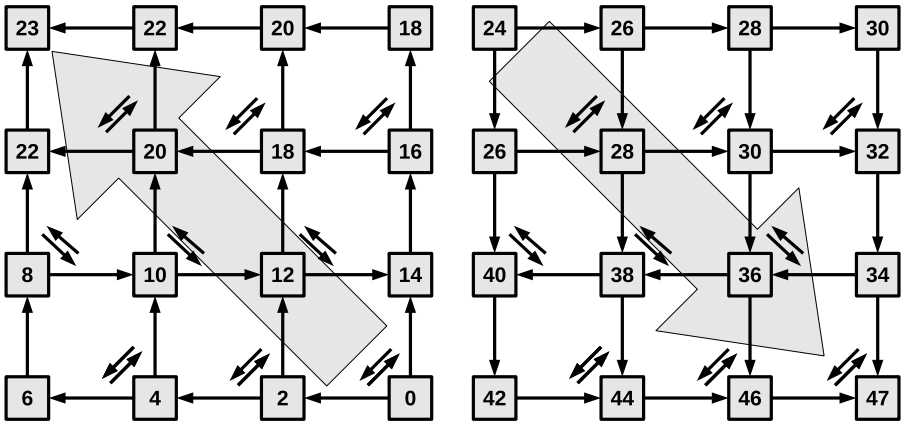
Figure 6.9: Configuration information from neighbor switches and control network

6.5.1 Propagation

In order to simulate the reconfiguration process, we have modeled the OSR-Lite scheme in our event-driven cycle-accurate network simulator. A 8×8 mesh is used with wormhole switching (although the proposed method also works for virtual cut-through switching). Flit size is set to 4 byte and messages are 5-flit long. For the transient state, 50K messages are assumed and results are collected after 50K messages are received.

Figure 6.10 shows how OSR-Lite tokens propagate over a mesh when there is no traffic traveling through the network. The diagonal arrows represent the bidirectional restrictions imposed by the routing algorithm (Segment-Based routing [157] in this case). In this figure, the numbers inside the switches represent the cycle when the token signal is propagated to its neighbors. Moreover, the arrows among switches depict the direction of the token signal propagations. As we can see, the token signals propagate among switches throughout the network in the order of the routing channel dependency graph, where Figure 6.10.(a) follows a scrolling up zig-zag direction, and Figure 6.10.(b) follows a scrolling down zig-zag direction.

When no messages are traveling through the network and a regular 2D mesh is considered then the number of clock cycles required for the OSR-Lite recon-



(a) (b)

Figure 6.10: OSR-Lite propagation over a 4×4 2D mesh topology: (a) scrolling up, and (b) scrolling down.

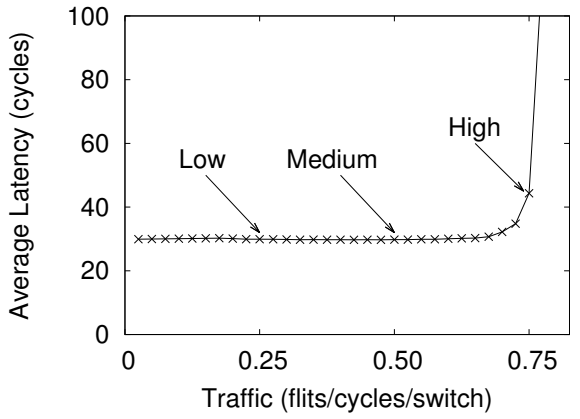
figuration process is modeled by the following formula:

$$PropagationTime = (4xDx(D-1)) - 1 \tag{6.1}$$

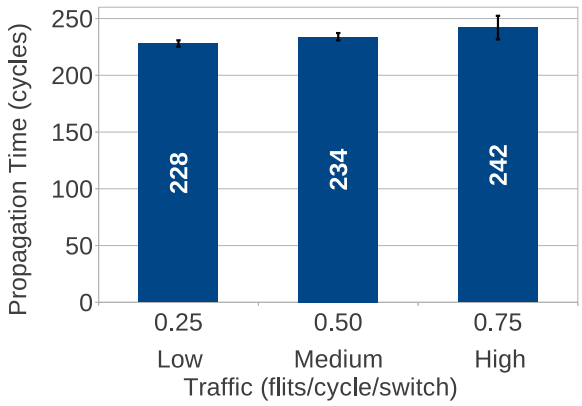
where D represents the mesh dimension. As we can see, it is a very fast process as the protocol uses only 223 cycles when a 8×8 mesh is considered. The high speed of the OSR-Lite reconfiguration process allows to perform frequent planned reconfigurations without affecting the integrity of the system operations. However, when there are messages traveling through the network the switches must drain the input queues of old messages before propagating the token signal as explained in Section 6.2. This fact delays the OSR-Lite propagation depending on the network load. In the following, we analyze this effect taking into account different injection rates.

6.5.2 Time Overhead

In order to analyze the impact of the network load over the OSR-Lite reconfiguration framework, we have performed different simulations varying the injection rate. For each rate, we assume a constant packet generation rate for all end nodes. Moreover, in order to ensure that start-up instabilities do not affect our evaluation results, reconfiguration is not invoked until the network is completely stabilized. Figure 6.11.(a) shows the performance obtained in a



(a)



(b)

Figure 6.11: (a) Average message latency at different injection rates for SR routing on 8×8 2D mesh (b) OSR-Lite propagation over a 8×8 2D mesh topology at different injection rates.

8×8 2D mesh network under uniform traffic when no reconfiguration process is triggered. The figure indicates the three network injection rates that are used in the simulations. In what follows, the three rates are referred to as Low, Medium, and High, respectively.

Figure 6.11.(b) shows the number of cycles involved in the propagation of the OSR-Lite process taking into account the three different injection rates.

Each bar depicts the mean of 30 simulations varying the seed. Moreover, we show the error bars that represent the 95% confidence interval. As we can see, the propagation time does not exceed 242 cycles for the High injection rate. Moreover, the difference between both the minimum and the maximum network loads is only 14 cycles, and therefore, the network traffic condition has a minimal effect on the OSR-Lite token propagation.

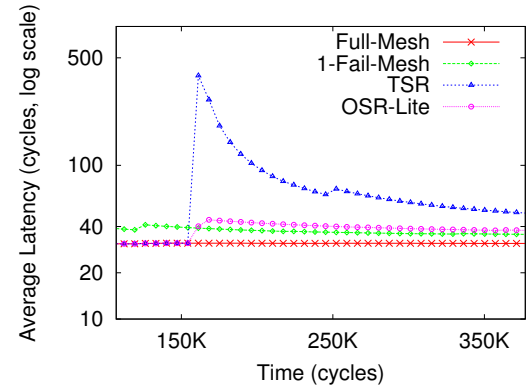
Finally, the contribution in terms of cycles for event notification (A), algorithm computation (B), configuration bits delivery (C) and OSR-Lite propagation (D) should be taken into account to determine the total latency for a complete reconfiguration process. In particular, (A) and (C) latencies depend on the position of the components to reconfigure with respect to the central manager. On the other hand, (B) and (D) latencies are related to the number of components to reconfigure and the traffic injection rate respectively. As an example, when we consider a 8×8 mesh then at most 66 cycles are required to cross the control network. Moreover, if 7 switches need to be reconfigured (i.e. the scenario of Figure 6.1) then 195 cycles are required by the computation algorithm in [156]. Finally, 242 cycles are spent by (D) in a High injection rate scenario. Summing up, the total amount of cycles for a complete reconfiguration process is the following:

$$66(A) + 195(B) + 66(C) + 242(D) = 569\ Cycles \tag{6.2}$$

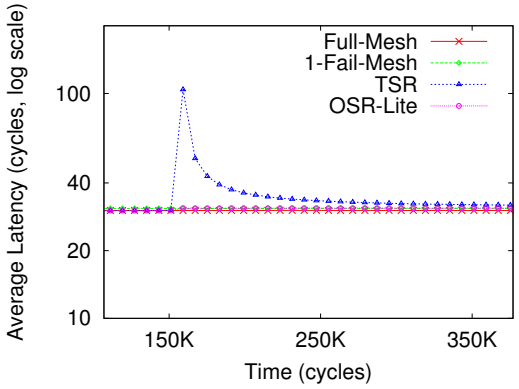
For dissemination of new LBDR bits to the switches, the dual network has to carry 17 bits per switch. However, not all switches need to be reconfigured, since the algorithm in [156] is able to evolve a system configuration into a new one while updating the minimum amount of LBDR configuration bits.

6.5.3 Comparison

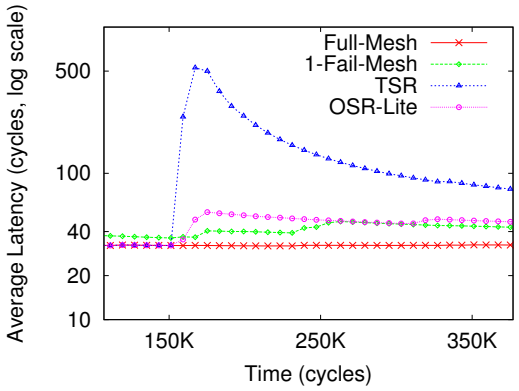
In this section we compare the OSR-Lite protocol and the traditional static reconfiguration process (TSR). Figures 6.12.(a), 6.12.(b), and 6.12.(c) represent the average network latency respectively under hotspot traffic and uniform traffic with Medium and High injection rates, where both reconfiguration processes (OSR-Lite and TSR) are invoked after 150K cycles. Moreover, we have plotted two additional lines: the average message latency for the full mesh (Full-Mesh), and the average message latency for the mesh which has one link disabled from the beginning of the simulation (1-Fail-Mesh). Notice the y-axis is in logarithmic scale. Moreover, we have selected a random link in the 8×8 mesh as faulty. Under hotspot traffic pattern, 5 nodes are randomly chosen as



(a)



(b)



(c)

Figure 6.12: Average message latency with (a) hotspot traffic and uniform traffic ((b) medium network load and (c) high network load).

hot spots which receive an extra proportion of traffic (30%) in addition to the regular uniform traffic.

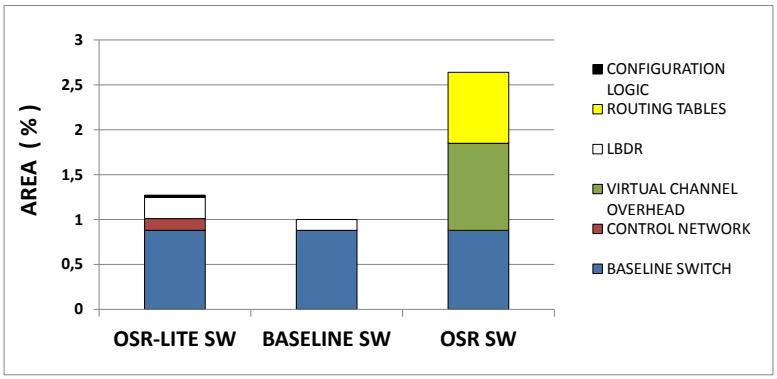
The first observation is that both Full-Mesh and 1-Fail-Mesh obtain a different message latency. This is normal because the 1-Fail-Mesh suffers a latency degradation due to the disabled link. On the other hand, the two reconfiguration processes (OSR-Lite and TSR) start at the same time at the 150K cycle. At this point, the reconfiguration process moves from the Full-Mesh to the 1-Fail-Mesh topology. This effect can be estimated by the figures as the latency evolves from the latency obtained for the Full-Mesh to the latency obtained for the 1-Fail-Mesh. However, an important result based on the figures is that OSR-Lite performs the reconfiguration without degrading the obtained performance. In this case, the obtained latency grows up to the 1-Fail-Mesh line. Therefore, the latency is always near the maximum obtained with the 1-Fail-Mesh topology. In the TSR case, on the contrary, the latency is degraded due to the reconfiguration process overhead (need to drain the network). In the three cases, the latency grows above the 1-Fail-Mesh latency until it stabilizes. Specifically, in the Figure 6.12.(c) the latency of the TSR line grows to more than 500 cycles, and then stabilizes after 350K cycles. In this period of time, the TSR reconfiguration is degrading the obtained latency more than the link failure degradation produces. On the other hand, the OSR-Lite latency is upper bounded by the 1-Fail-Mesh latency.

Interestingly, the hotspot traffic and the uniform traffic with a High load have similar reconfiguration performance. Then, we can observe that the OSR-Lite has no impact on the message generation while the TSR process does. In fact, the TSR process increases considerably the obtained latency for all the cases. The main reason is that TSR queues all the messages at end nodes during reconfiguration while that need disappears in the OSR-Lite scheme.

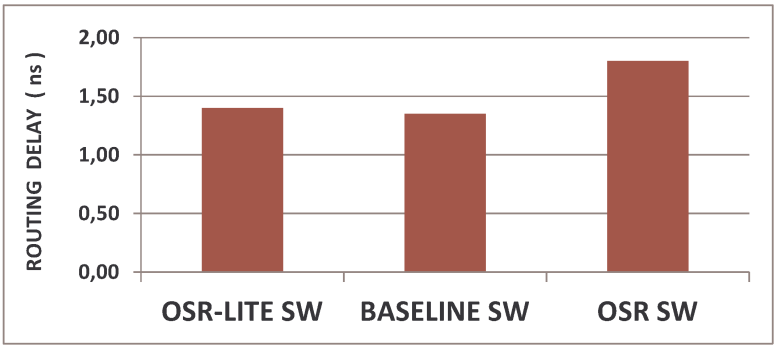
6.6 Synthesis results

The implementation of a switch enhanced with the OSR-Lite mechanism has been compared in terms of area and routing delay with a switch based on the native OSR mechanism described in Section 6.2 and the baseline xpipesLite switch architecture [110]. The evaluation will demonstrate the infeasibility of the native OSR mechanism for an on-chip setting because of the need for VCs and the low scalability of routing tables.

For the experiments, an industrial memory compiler for a 40nm process technology was used to generate the memory macros required by the routing ta-



(a)



(b)

Figure 6.13: 5x5 switch (a) area and (b) routing delay comparison.

bles of the OSR mechanism. The switches together with their reconfiguration mechanisms were synthesized for the same 40nm industrial library.

6.6.1 Area Comparison

The description of the OSR mechanism in [48] focuses on the protocol details and it lacks of practical implementation details. Thus we exploited the information provided in [48] to model the OSR mechanism at RTL level and evaluate this latter solution in an on-chip constrained system. Especially, the OSR mechanism relies on 1 data VC supported by an additional control VC, and it adopts routing tables. As a result, we implemented the OSR mechanism into a 5x5 switch augmented with VCs by following the design techniques for area efficiency in [158] and we enhanced the switch with the 40nm memory

macros to model the routing tables.

The 8×8 mesh topology of Section 6.5 was considered. Thus, 64 end-nodes are the total number of destinations in the system. When routing tables are used for distributed routing, each switch input port has a memory module with a number of words equal to the amount of destinations. Every word is composed of 3 bits, matching the switch radix. Given a destination ID, the switch selects the target output port based on look-up table. The minimum word width that the memory compiler, at the 40nm technology node, can generate is 4 bits. As a result, above all the available memory cuts, a single-port low-power RAM with 64 words of 4 bits was the memory cut showing the lowest routing delay and area footprint.

Finally, Figure 6.13.(a) shows the area footprint of this latter solution (the *OSR SW*) with respect to a baseline switch and our proposed solution (*OSR-LITE SW*). In particular, the OSR-Lite area overhead takes into account also the contribution of the control network carrying the information from the global manger to the routing mechanisms. For this purpose, we exploited the fault-tolerant control network proposed in [156].

The OSR-Lite reconfiguration mechanism requires a 14% of area overhead with respect to the baseline switch. This result is mainly due to the additional LBDR routing mechanism (+12%) contribute. On the other hand, the area overhead of the remaining reconfiguration logic (detailed in Section 6.4) is negligible when integrated into the switch.

Interestingly the OSR-Lite switch outperforms the baseline OSR switch: this latter requires approximately two times larger area than the counterpart solution. This result is mainly due to the severe area penalty introduced by the VCs and the 65% area saving achieved by the LBDR mechanism with respect to the routing table.

As a last consideration, the routing mechanism of the OSR-Lite solution scales with network size. In fact, while the memory macro suffers from increasing area and delay penalties, the logic complexity of the distributed routing algorithms does not depend on the number of destinations, hence it stays constant. Indeed, the distributed routing algorithms just grow with the switch radix.

6.6.2 Routing Delay Comparison

In order to evaluate the effects of the OSR-Lite mechanism on the switch routing delay, we performed the 5x5 switch synthesis for maximum performance. The same experiment was repeated for both the baseline switch and the switch

augmented with the baseline OSR mechanism. The *OSR-LITE* switch and the baseline switch achieved a similar maximum operating speed of 750 MHz. As described in Section 6.4, the reconfiguration scheme was designed to avoid long critical path and preserve the baseline switch performance. Our OSR-Lite-enabled switch is thus capable of an at-speed reconfiguration.

On the other hand, the *OSR* switch is the 35% slower than our proposed solution as showed by Figure 6.13.(b). This result is mainly due to the intrinsic complexity added by the VC logic and the delay required to access the 64 words RAM routing tables.

6.7 Conclusion

The goal of this chapter was to provide reconfigurability design methods for next-generation embedded systems and drive the implementation of the configurable GP-NaNoC switch presented in the next chapter. Finally, we have proposed a fast and transparent reconfiguration mechanism in NoCs. The strict constraints found in on-chip networks demand for efficient and compact mechanisms that do not excess in area and latency demands. The native OSR reconfiguration mechanism has been proven to be unsuitable for this purpose although proposing an interesting intuition. Therefore, in this chapter we have engineered the proper protocol and implementation modifications to fit reasonable area budgets, with no impact on performance while retaining the same underlying principle for fast reconfiguration. The final mechanism, OSR-Lite, is able to support a transparent NoC reconfiguration with as little as less than 250 cycles when an 8x8 2D mesh is considered.

7

Co-Optimized Design Methods for General Purpose System

THIS chapter presents the GP-NaNoC switch: it integrates the most relevant and innovative design methods conceived throughout the thesis and makes sure they co-exist together in a runtime reconfigurable switching fabric supporting network partitioning and isolation as well as irregularities stemming from power/thermal/fault-tolerance management frameworks. Finally, the GP-NaNoC switch guarantees support for static and dynamic irregularities, for detection of network status and for control signaling. The OSR-Lite reconfiguration method proposed in Chapter 6, the outcome of the exploration of built-in self-testable strategies in Chapter 5 and state-of-the-art fault-tolerance techniques are co-designed together and integrated in the final GP-NaNoC switch presented in this chapter.

7.1 Introduction

In past years, the differentiation between network-on-chip switch architectures was achieved through relevant parameters such as supported topologies, switching technique, flit size, buffering styles, supported routing algorithms, etc.

The reason for this stems from the fact that NoC technology took only ten years from the research vision to the industrial uptake. Such a fast evolution has been driven by two converging trends. First, Moore's law has maintained its pace in terms of logic (and storage) density, but it has finally reached hard limits in terms of power consumption and synchronization. Hence, SOCs today heavily rely on multi-core parallelism and locally synchronous, globally asynchronous power domains. Second, the complexity of large-scale SoCs

combined with the ever-increasing time-to-market pressure have pushed for a strong componentization and modularization of silicon platforms, to reduce design effort though massive reuse combined with hierarchical, divide-and-conquer design flows. NoCs meet all the key requirements imposed by these converging trends: they facilitate the modular construction of heterogeneous multi- and many-core architectures, they provide communication abstractions and services across component boundaries and they enable the top-down design of highly power-manageable architectures.

Today, NoCs are a mainstream industrial interconnect solution and basic design techniques for switch architectures are consolidated. Therefore, we are at the stage where the features of the on-chip network can be matched with the requirements of the system management framework(s) for cross-layer optimization and efficient system control. This is bringing awareness of new key requirements that the interconnect fabric should meet and that are out-of-reach of current NoC realizations.

In fact, microelectronic system design, as never before, is evolving under the influence of its two main drivers, the broadening complexity of applications and the opportunities along with the constraints of nanoscale technologies. The concept of use case is becoming mainstream in embedded computing: not only applications can run in several modes, but also the combinations of applications that can be executed concurrently in a system are manyfold and changing over time. Moreover, some mainstream concepts from the high-performance computing domain such as system virtualization are making inroads also in the embedded computing domain, thus calling for partitioning and isolation capabilities in system resources. At the same time, while technology is providing unprecedented levels of system integration, it is also bringing new severe constraints to the forefront: power budget restrictions, overheating concerns, circuit delay and power variability, permanent faults affecting the system right from the beginning (manufacturing faults) or with progressive onset (wear-out faults), increased probability of transient faults.

The inherent tension between these trends leads to the explosion not only in the design space, but also in the runtime reconfiguration space. The on-chip interconnection network is obviously affected by these converging trends, which are driving it to the migration into new forms and shapes of synchronicity, reconfigurability, testability and fault-tolerance. A reconfigurable interconnect fabric is in fact at the core of a system aiming at runtime adaptation; the capability to reliably bring data across component boundaries is key to the reliable operation of the system as a whole; the support for multi-synchronous fre-

quency domains enables power optimizations and guarantees the integration of heterogeneous components featuring differentiated operating speeds.

This thesis wants to propose a timely answer to the above concerns. It has identified the basic design requirements needed to augment NoC architectures with an enhanced degree of dynamism and flexibility and to let them successfully cope with the increasing uncertainty of the technology platform. In essence, such requirements are:

- **Support for static irregularities**. These are deviations of the physical topology from the logic topology planned at design time and that can be found out during post-silicon testing or at boot-time testing. The proper course of recovery action is somehow simplified by the fact that no traffic is crossing the network yet and that in this pre-use mode of the electronic device a somewhat larger activation latency can be supported.

- **Support for dynamic irregularities**. These are intentional topology reconfiguration decisions taken at runtime in the context of more general system management strategies. For instance, power management decisions or the need to contain overheating of selected areas may lead to power them off at runtime. Also, some links or components may be detected to start failing too often, thus denoting the onset of permanent faults in them that requires their prompt disconnection. Finally, the topology may be broken down into smaller subsets of irregular shape requiring traffic isolation, as requested by virtualization strategies. In all these cases, the NoC has to deal with wanted reconfigurations that affect ongoing traffic, thus requiring smooth and safe transitions between network states within stringent latency requirements.

- **Support for the detection of network status and for control signaling**. Unwanted effects in the network (manufacturing faults, intermittent faults) have to be promptly detected to allow the resource management framework to take the proper course of recovery and/or reconfiguration action. This goes through the implementation of network status detection mechanisms specialized for the target event they want to detect. For instance, testing frameworks are key to gain awareness of permanent faults in the network, while an error notification framework should be able to capture transient and intermittent faults. The final outcome is the network status information that needs to be delivered to the final decision point. This brings the issue of a system-level notification infrastructure to the forefront, with extension capability to carry reconfiguration com-

mands to network components.

- **Support for multiple frequency domains.** Application-specific systems are typically composed by assembling together heterogeneous components featuring differentiated operating speeds. The maximum operating speed of the system should not be constrained by the speed of the slowest component. This calls for proper decoupling at the network boundary by means of synchronization interfaces. This is a key requirement also for power management strategies in the embedded computing domain, requiring each core to run at an independent and runtime variable voltage and speed.

The above requirements led us to design a next-generation switch, called GP-NaNoC switch, with the following characteristics:

- **Logic-based distributed routing**. It has been proven the poor scalability of routing tables with the silicon technology node and with the number of network nodes. Using distributed routing logic instead of routing tables is appealing for its better scalability properties but also challenging when the network connectivity pattern is (or becomes) highly irregular. LBDR is an appealing solution in this domain and is therefore implemented in the GP-NaNoC switch. Obviously, system management frameworks, manufacturing issues or physical degradation effects may affect the regularity of the topology, but this thesis has conceived robust design methods to effectively handle this in the presence of a logic based distributed routing framework.

- **Fault-tolerant flow control**. The simple stall/go flow control protocol was known to effectively backpressure in case of congestion but to lack of any kind of support for fault-tolerance. This thesis has therefore integrated the Nack/go flow control protocol [58], which is implemented in the GP-NaNoC switch. Nack/go combines the pros of stall/go (prompt flow resumption, low power) with the pros of ack/nack (fault-tolerance support through the retransmission of faulty flits with go-back-N policy). Nack/go envisions retransmission of corrupted flits on the datapath, thus minimally affecting the critical path of the switch (no error correction on it), while introducing only a few cycles overhead in the rare cases of actual retransmissions. Nack/go targets transients faults and single event upsets in switch buffers. Nonetheless, in case a permanent fault shows up at runtime, it may end up in an endless retransmission loop. This

was however not considered to be a problem, since the target NoC has reconfiguration capability to work around the faulty link or switch component. In practice, such an unfortunate case would not affect system lifetime but would result only into its graceful degradation.

- **Fault-tolerant control path.** While Nack/go effectively protects the datapath, the traditional solution to protect the control path against transient faults consists of triple modular redundancy. However, we devised a joint fault-tolerance framework that exploits the Nack/go flow control protocol to use dual modular redundancy instead of the triple modular one in the control path. In essence, since control information travel inside flits or together with flits (i.e., as their extensions), in case discrepancies are found between the behaviour of duplicated control paths, retransmissions are scheduled thus feeding refreshed inputs to the switch FSMs.

- **Boot-time built-in self-testing support making use of pseudo-random test patterns.** Chapter 5 has performed a design space exploration of testing strategies for NoC switches. The experimental setting for this study was an overly simple switch, far away from the complexity of the GP-NaNoC switch. Given a new switch architecture, we gave priority to the designer's need of coming up with a testing framework in the shortest possible time frame. Therefore, we excluded the use of deterministic test patterns in spite of their proven capability to materialize lower test application times and implementation overheads. In contrast, we reverted to pseudo-random test patterns, and applied them following the guidelines devised in Chapter 5 to contain testing latency and to feed as many blocks as possible with the same LFSR. Nonetheless, the GP-NaNoC switch deals with the testing of complex FSMs and of retransmission circuitry and an interesting testing case study for future switches. GP-NaNoC supports built-in post-silicon and boot-time testing as well as distributed diagnosis, resulting in the indication of which switch input or output port (and associated links) is corrupted.

- **Centralized resource manager.** The GP-NaNoC architecture assumes a centralized entity which collects state information from the network and notifies reconfiguration commands backs to the NoC components. Intuitively, a distributed control would result in faster reaction times to network changes but in an overly more complex implementation. This is confirmed by the substantial network overprovisioning that works in the open literature have reported so far whenever distributed control has

been implemented: virtual channels, timers, additional inter-switch signaling, etc. At the same time, such solutions are not able to avoid sub-optimal or inefficient or even wrong reconfigurations in some corner cases, due to the lack of visibility of global network state. For these reasons, in NaNoC we have developed an entire system level infrastructure and reconfiguration methodology for a system with centralized control. In the future, we plan to overcome scalability limitations through hierarchical control schemes.

- **Dual NoC for control signaling.** Given the item above, the GP-NaNoC switch implements a read and a write interface to a dual NoC which is used to convery diagnosis information from switches to the global controller and reconfiguration instructions back from the controller to the switches. The dual NoC consists of a ring which connects all the switches together and where the global controller closes the ring, so that it can either receive diagnosis packets and send reconfiguration ones. Each GP-NaNoC switch therefore implements a routing primitive (essentially a 2x2 input-buffered mini-switch), which is the basic building block of the dual NoC.

- **Fault-tolerant control signaling.** The routing primitive of the dual NoC illustrated above is quite small with respect to the whole GP-NaNoC switch, therefore TMR could be implemented in it without significantly impacting the switch footprint. Voting is performed at the output of each routing primitive, since this gives more reliability than voting at the global controller only. Inter-primitive links are therefore also triplicated, with negligible impact on the sizing of the routing channel. The reader and writer interfaces also implement a protocol for the trustworthy reconfiguration of the network, combining a mix of fault-tolerance and online testing strategies that will be discussed in Chapter 8.

- **Segment-based routing.** Three key components are required to properly deal with a NoC system reconfiguration. The first component required to properly deal with a NoC system reconfiguration is the routing algorithm. It is very important to select the right routing algorithm in order to maximize its support to new topologies emerging from the occurrence of failures or of wanted changes of the connectivity pattern. In this work, without lack of generality we selected segment based routing, which features a good match with logic based distributed routing mechanism and which lends itself to some simple case for ultra-fast reconfigurations (e.g., a network could support one failure for each seg-

ment without the need for any resegmentation process). The GP-NaNoC switch can be instructed to implement the SR algorithm (or any other of choice) by simply setting the connectivity, routing and deroute bits of LBDR.

- **Overlapped static reconfigurations.** The second component needed for NoC reconfigurations is a nonintrusive and efficient reconfiguration mechanism to allow the routing algorithm to change uninterruptedly over system lifetime (to match associated changes in the connectivity pattern) while always remaining deadlock-free. The GP-NaNoC switch implements OSR-Lite (overlapped static reconfigurations), which enable such a deadlock free network reconfiguration without stopping network traffic or draining the network. The guiding principle consists of preventing packets with the new routing function from crossing links which packets with the old function still have to cross. While OSR-Lite was illustrated in Chapter 6, its integration with the other switch components and into a real network setting required a number of optimizations and enhanced flexibility provisions to support a truly arbitrary number of safe reconfigurations over time.

The GP-NaNoC switch supports also the multi-synchronous design methods proposed in Chapter 3. However such design methods have been already validated on silicon in the 40nm testchip presented in Chapter 4. Since the proposed methods consist of a straightforward integration of dual-clock FIFOs and mesochronous synchronizers into the input buffers of downstream switches, they have not been implemented in this chapter to simplify the design of the GP-NaNoC switch.

As regards the usage model, the GPNanoC switch is conceived for boot-time testing. In case a manufacturing fault is detected, diagnosis information is notified through the dual NoC to the global controller, which runs an online configuration algorithm to adapt the routing function to the actual connectivity pattern. The new LBDR configuration bits (coding the new routing function) are dispatched to the switches again via the dual NoC. For all control signaling, fault-tolerance and online testing ensure that the probability of wrong configurations is marginal (to stay on the safe side, the controller may want to discard switches in case doubts arise on the capability of the manager to reliably communicate with it). At runtime, when traffic is crossing the network, two kinds of events may occur. On one hand, wanted reconfigurations may be initiated by the global controller to selectively switch off part of the network or to partition the network into isolated domains. On the other hand, unwanted

reconfigurations may be needed when intermittent faults are captured by the fault-tolerance support of the GP-NaNoC switch and notified to the controller via the dual NoC. In both cases, the online configuration algorithm iterates and devises the routing function to evolve the connectivity pattern from the old one to the new one. An interesting option consists of using the OSR-Lite mechanism to partition the network into isolated regions, where different applications may be mapped without any traffic interference with each other. Examples of this usage model on an FPGA implementation will be provided in Chapter 8.

The rest of the chapter is organized as follow. Section 7.2 presents fault tolerant extensions introduced into the switch architecture. The focus will be on a novel fault-tolerant arbiter and the fault-tolerant NACK/GO flow control [58]. Section 7.3 optimizes the proposed reconfiguration method, OSR-Lite, for the target fault-tolerant switch. Therefore, the new input and output buffer will be illustrated. Next, Section 7.4 shows the system level notification framework enhanced with a fault-tolerant dual-network and Section 7.5 presents a novel built-in self-testing framework handcrafted for the GP-NaNoC switch based on pseudo-random patterns. Then, results in terms of area, stuck-at faults coverage and routing delay are highlighted in Section 7.6. Last, the conclusions are presented in Section 7.7.

7.2 Switch Architecture Extensions for Fault-Tolerant NoC Design

Fault-tolerance represents the first feature required by a general-purpose switch in order to satisfy the high reliability constraints imposed by modern systems. As a result we picked the ×pipesLite switch of Figure 5.1 and we incrementally extended its baseline architecture.

The switch has been enhanced with the fault-tolerant flow control protocol (NACK/GO) proposed in [58]. It uses the NACK/GO protocol on the data path to notify error detection and trigger link-level data retransmissions, and also on the internal switch data path, to ask for data retransmissions from switch input to output buffers (see Section 7.2.1). Interestingly, on-demand correctors are used to repair corrupted values in the source buffers, which a retransmission would not fix. Thanks to the retransmission capability of the data path, the control path can implement a simpler dual-modular redundancy: in case of differences between the replicated paths, the current transfer is invalidated and a retransmission is required in the next cycle (see Section 7.2.2). Logic-based distributed routing (LBDR) is implemented. The support for different routing

algorithms and topology shapes is achieved by means of 16 configuration bits for the routing mechanism of the switch (hereafter denoted as LBDR bits). LBDR bits are computed by a central NoC manager and disseminated to the switch input ports through the dual control network proposed in [156]. Next, the NACK/GO protocol proposed to trigger data retransmissions is derived, and later on the control path is detailed accordingly.

7.2.1 The New Fault-Tolerant Flow Control: NACK/GO

STALL/GO is one of the simplest flow control protocols that can be found in the open literature. It leverages only one forward signal, that flags the availability of new valid data (Valid signal) and one backward signal, used to stop the communication flow when a new flit cannot be accepted due to congestion in the downstream node (Stall signal). Conversely, ACK/NACK is a flow control protocol with error detection/notification capabilities. It exploits a Go-back-N policy to manage and control correctness of the transmitted data. When an error is detected in a transmitted flit, the receiver signals this event to the sender, who will retransmit the flit with the corrupted information and all the (N) successive ones.

Unfortunately, the ACK/NACK protocol does not make a clear distinction between the backpressure phenomenon and the occurrence/detection of transient faults. As a consequence, a nack received by the upstream switch means that the flit should be retransmitted for some reason. In case of congestion of downstream paths, the protocol keeps retransmitting the same flit indefinitely regardless of the receiver state, thus proving power-inefficient. On the contrary when a STALL/GO protocol is considered then transmission freezes until a go arrives, in case a stall notification is received upstream. This protocol is much more power efficient but does not provide any kind of support for fault tolerance and data retransmission.

Augmenting the protocol in this direction we comes up with the NACK/GO flow control protocol. NACK/GO is a new protocol, and associated switch implementation, that offers full error detection and notification capabilities of ACK/NACK while preserving the power efficiency of STALL/GO for error-free operation.

NACK/GO leverages four control signals to control transmission of data and achieve fault tolerance. There are two signals going in the same direction of the data stream, and two backward propagating signals.
Valid: this signal flags availability of new data, and it triggers the data transfer.

Trash: this control signal notifies that the data currently being transmitted is corrupted and should be discarded. The reason why the valid signal is not used for this is to optimize the internal critical path of the switch and avoid multi-cycle switch traversal.
Stall: it stalls the transmission in case of traffic congestion. The Stall signal stops the communication flow, freezing all the forwarding control signals and data to their current value.
Nack: Nack signal is de-asserted low when a valid flit has been received in the previous clock cycle (Acknowledgement). In contrast, Nack is asserted high whenever no valid flit is received, either because no transmission took place or because the accepted flit is detected as corrupted.

NACK/GO combines the best of STALL/GO and ACK/NACK. Like STALL/GO, it exploits an efficient methodology to block the communication traffic in case of congestion, avoiding the unnecessary switching activity for flit retransmission as in ACK/NACK. It uses a signal to notify the availability of free buffer positions inside the receiver, so communication can be frozen when congestion occurs. Furthermore, a Stall signal avoids the roundtrip necessary to resume communication from the packet that was not accepted like in ACK/NACK, leading to a better average performance.
In addition, it features the error flagging capability not exposed by STALL/GO. In this way the system can re-establish a correct working point and resume its correct operating condition.

These advantages come at the cost of extra channel wiring, since a total of 4 control signals are needed, rather than the 2 of STALL/GO, and a more complex control logic inside the buffers and the switches. NACK/GO features worse minimum buffer slot requirements than STALL/GO, since it requires a minimum of 3 buffer slots. In addition, every pipeline stage inside the link should have not only flow control capabilities, but also error detection/correction capabilities. For this reason, every repeater must have at least three buffer positions, more than the two required for STALL/GO.

7.2.2 Novel Low-Power Fault-Tolerant Arbiter

By exploiting the retransmission capability provided by NACK/GO, we designed the fault-tolerant control path that supports NACK/GO flow control operation.

The novel fault-tolerant arbiter was designed following Figure 7.1. It represents an effective variant of a baseline TMR arbiter. As showed in Figure

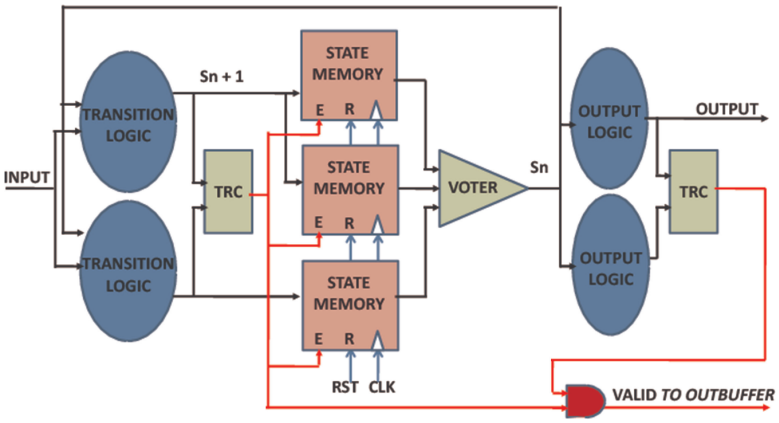


Figure 7.1: Fault tolerant arbiter implementation.

7.1, the *transition logic*, representing the arbiter combinational logic for computing the next FSM state, is doubled. Therefore, the outputs of the two *transition logic* instances are compared in a Two-Rail Checker (TRC) module (i.e. a redundant comparator block with fault tolerance capability) and feed the state memory register of the arbiter. The state memory register is triplicated as in a conventional TMR strategy although it is enabled by the TRC module output. When the two *transition logic* blocks generate the same result, then they are not affected by an error and the state memory register can sample the new state. On the other hand, the TRC block freezes the state of the arbiter registers when the *transition logic* blocks provide different results. In this latter case, the *valid* signal to the output port is deasserted. As a consequence, the output port will not read the incoming information affected by errors, and since no valid data has been stored in the current clock cycle, the Nack signal will be asserted high in the following clock cycle. This will be interpreted by the input buffer as a request of retransmission.

Thus, the outputs of the three state memory registers are voted before feeding two instances of the output combinational logic with the arbiter current state. Then, the outputs of the combinational logic modules are compared in a TRC comparator following the previous implementation adopted for the *transition logic* modules. Finally, when the above mentioned comparator reveals an error, then the *valid* signal to the output port is deasserted and the information coming from the output combinational logic is discarded.

The baseline behavior of the arbiter remains the same: it selects between different inputs competing for the same output port, operating with a round robin

arbitration policy in order to enforce fairness between all the requests. New conditions and events had to be managed in order to cope with error detection: whenever an error is detected in the output buffer, the Nack signal must be routed to the correct destination. If this takes place on the head flit of a packet, that stores the information about destination, in order to avoid misrouting the current source will lose the grant acquired by the arbiter. In the following clock cycle, the route will be re-computed and the arbitration process will take place again. On the other hand, when an error is signaled for the last flit of a packet (tail flit), the sender has already lost the grant given by the arbiter and has no longer exclusive access to the output port. For this reason, the arbiter does not consider any new request, and the sender that had the grant in the previous clock cycle acquires again access to the output port. In this way the correct status of the various actors is re-established, and the corrected flit can now be retransmitted. Lastly, when a flit is signaled as corrupted while being transmitted (Trash signal asserted high), the arbiter must propagate this condition to the output port, and ignore the current transaction taking place. The trash signal is forwarded to the output buffer, that will ignore the current incoming flit.

7.2.3 Fault-Tolerance of Routing logic and Buffer FSMs

In order to protect routing logic, two LBDR replicas directly feed the combinational logic cascaded to the TRC of the arbiter. A failure in the LBDR logic will be tackled by the arbiter's Two Rail Checkers, thus exploiting the cooperation with the arbiter to achieve fault-tolerance by means of an effective lightweight solution.

The status registers of buffer FSMs need additional protection. For this purpose, the simplest thing was to implement TMR in light of the low-complexity of replicated circuits.

Figure 7.2 depicts the final NACK/GO switch architecture, where emphasis is given to the fault-tolerance support.

7.3 Reconfiguration Mechanism

The OSR-Lite mechanism, initially proposed in Chapter 6 for the baseline ×pipesLite switch, needs to be modified in order to better suit the target NoC environment. In fact, it should be upgraded to guarantee fault-tolerance, effective support for frequent reconfigurations and tight exploitation of the in-situ

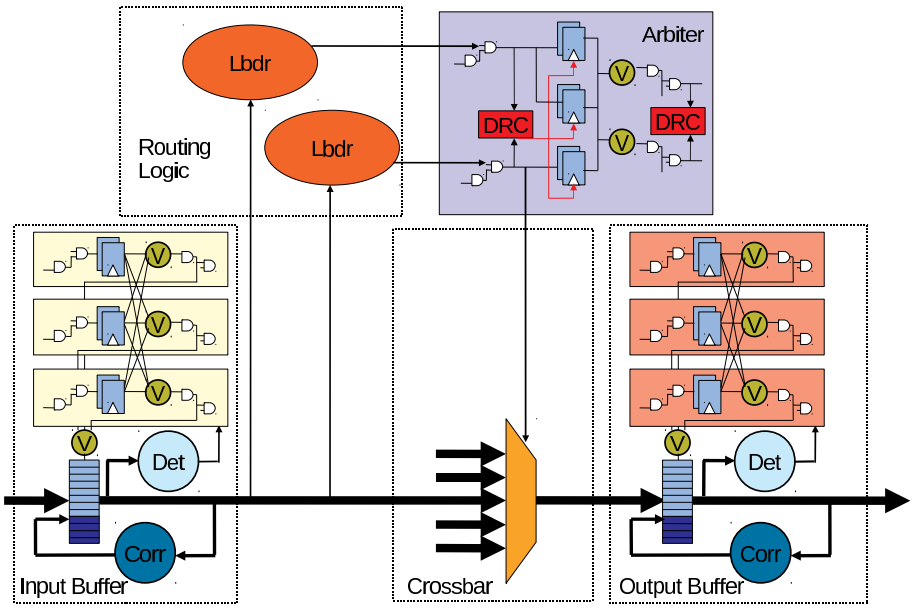


Figure 7.2: Nack-Go switch

control network. Indeed, this chapter targets such upgrades especially addressing the following issues:

- *Codification of the routing information.* During the reconfiguration process both routing algorithms coexist at the same time at routers. This means resources need to be sized for both algorithms. In OSR-Lite, two logic-based distributed routing blocks (LBDR) per input port were used to store the routing info. Anyway, optimizations can be envisioned to further reduce the area overhead introduced by routing blocks.

- *Involvement of neighboring switches in the reconfiguration process.* Three additional routing bits are sent by each upstream switch to expose the dependencies between output and input ports. Thus every link needs to be extended with 3 additional wires. Exploiting the in-situ control network, such link extension can be avoided and the link width of the main data-network preserved.

- *Transparent support for multiple reconfigurations.* Once the network has been reconfigured and migrated to Epoch 1 then all the switches need to swap to Epoch 0 again in order to support a new reconfiguration

process. OSR-Lite should be capable to migrate from an epoch to the next one without need to bring its logic back to initial state.

- *Extension for fault tolerant design.* The OSR-Lite mechanism should be extended in order to suit the fault-tolerant environment in which it is integrated. In particular, its logic should be reliable and support the fault-tolerant NACK/GO protocol of the target switch.

The switch architecture described in Section 7.2 is extremely modular. A fault-tolerant port-arbiter, a crossbar multiplexer and an output buffer are instantiated for each output port, while a fault-tolerant routing module is cascaded to the buffer stage of each input port. The logic enabling the OSR-Lite mechanism was integrated into the Section 7.2 switch taking care of preserving its modularity together with its performance. Thus, the OSR-Lite logic was designed in new modules plugged into the switch without affecting the existing blocks. Moreover, the new modules were instantiated for each switch port following the modularity of the baseline blocks (the OSR-Lite mechanism can be extended for switches of every arity by means of simple logic replication). In the next sections we describe in more detail how the above mentioned upgrades are materialized and how they affect the baseline OSR-Lite implementation.

7.3.1 OSR-Lite at the Input Ports

In Chapter 6, the OSR-Lite mechanism was designed on top of a baseline switch without fault-tolerance features based on a STALL/GO flow-control. In this latter scenario, the STALL/GO switch is enhanced with a second routing logic unit (LBDR1) collecting the new routing info coming from the central manager. This unit is connected to the input buffer as the baseline LBDR0 block (see Chapter 6) although is used exclusively for routing packets in the new epoch (new packets).

The switch arbiters need to select the routing info from the appropriate routing logic block (either LBDR0 or LBDR1). This is obtained from a multiplexer configured by the current epoch of the input port (in a flip-flop).

The baseline OSR-Lite architecture should be now upgraded in order to suit the NACK/GO switch of Section 7.2 and allow effective frequent reconfigurations. We firstly remove a routing block instance for input port. In fact, two routing functions can coexist in the same switch but they will never run concurrently at input port level. Thus, we can exploit the same LBDR instance in all epochs. In particular, the LBDR is uploaded with new routing function information

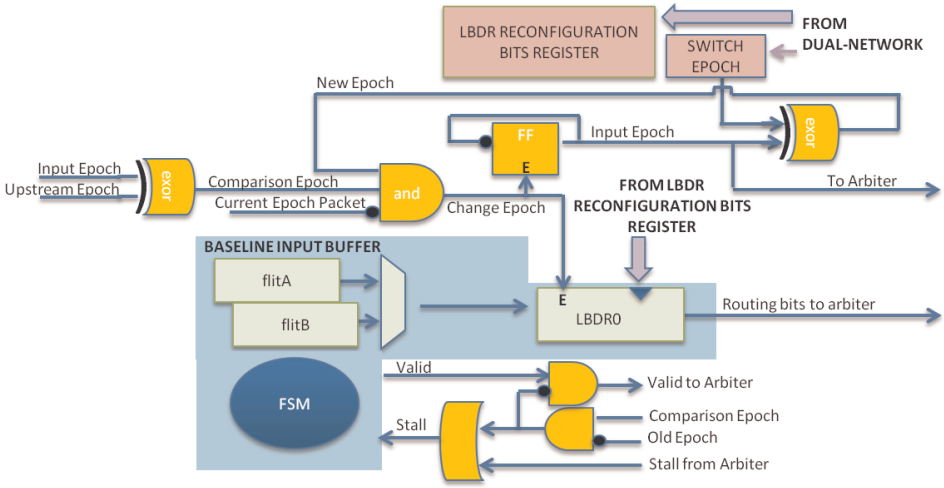


Figure 7.3: OSR-Lite logic extended for the NACK/GO input buffer.

as soon as its input port migrates to new epoch. The new routing function information are contained in a register which is shared between all the LBDR instances and stores the incoming configuration bits from control network.

A local register stores the current epoch of the input port while an additional register triggered by control network stores the switch epoch. The switch and input port epoch are compared in an exor logic block to reveal a reconfiguration request (*New_Epoch* flag). When the reconfiguration request is asserted then the input port evolves to the new epoch if there are no stored flits from current epoch (*Current_Epoch_Packet* signal) and the token in compliance with current switch epoch has been received from the upstream switch (*Comparison_Epoch* signal). Finally the *Change_Epoch* flag enables to set the LBDR with new configuration bits and to switch the local epoch register to new epoch.

Figure 7.3 depicts the new OSR-Lite mechanism at the input port. To note that it shows a single routing block for the sake of clarity but the routing block is actually fault-tolerant (i.e., it is composed by two replicas of LBDR following the fault-tolerant solution of Section 7.2). Interestingly, we would need to double the routing resources by means of four replicas of LBDR if we would want to still exploit the baseline OSR-Lite mechanism.

The OSR-Lite mechanism for NACK/GO switch requires extensions also to the counter monitoring incoming and outgoing packets. In particular, two counters are integrated to respectively monitor epoch 0 and epoch 1 packets. The counter 0 (*counter 1*) increases its value when a header is accepted, the

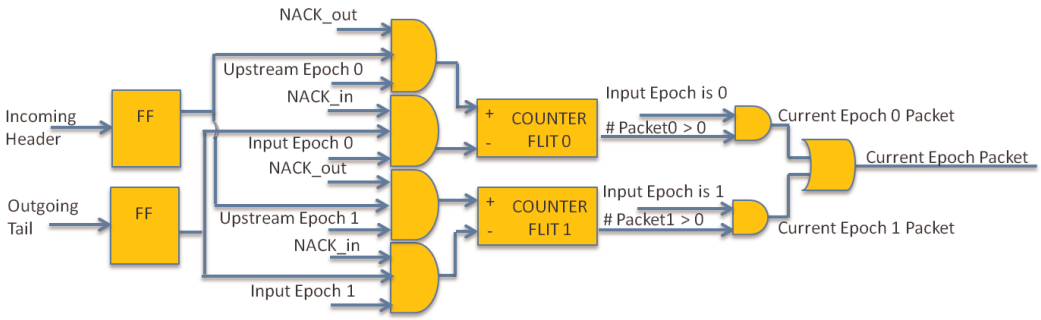


Figure 7.4: Counter of packets of the two epochs.

incoming token is 0 (*token is 1*) and the input port does not assert a NACK during the next clock cycle. On the contrary, the counter 0 (*counter 1*) decreases its value when a tail is sent, the local epoch is 0 (*local epoch is 1*) and the input port does not receive a NACK during the next clock cycle. Finally, the *Current_Epoch_Packet* flag is asserted if the counter 0 or the counter 1 register packets in accord with the current input port epoch. Figure 7.4 depicts the two epoch counters.

7.3.2 OSR-Lite at the Output Ports

An output port evolves to the new epoch when all the input ports with output dependencies to this output port have evolved to the new epoch. In order to efficiently deal with the dependencies, OSR-Lite takes profit of the routing bits used in LBDR. Therefore, the output port needs to receive both the epochs of the input ports and the routing restrictions located at the neighboring switches.

The baseline OSR-Lite mechanism presented in Chapter 6 obtains channel dependencies directly from the routing logic located at neighbor switches. As a consequence, three additional routing bits are required to be sent by the LBDR0 logic of the upstream switch together with the token bit. To note that LBDR0 received its old routing bits information through the control network in an earlier configuration stage. In addition, the input port needs to send the incoming routing restriction signals to the appropriate output ports. Thus every link is extended by 4 additional wires (i.e. 1 token wire + 3 routing restriction wires).

However, the OSR-Lite mechanism needs to be upgraded to support the NACK/GO protocol and exploit the in-situ control network. In fact, the routing

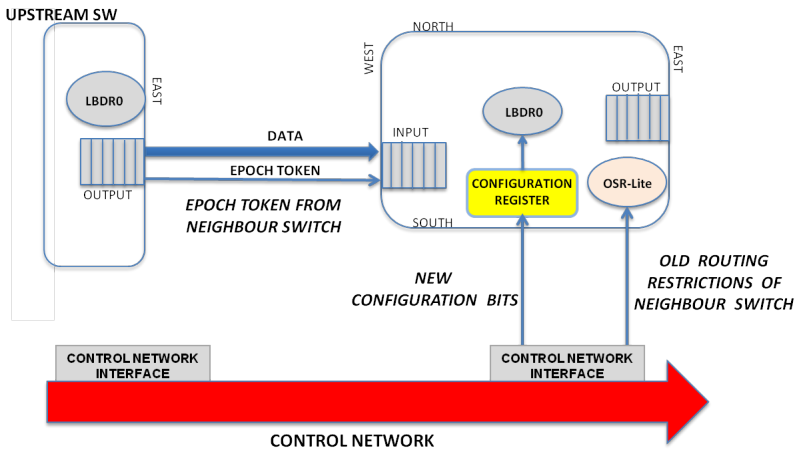


Figure 7.5: Configuration information from neighbor switches and control network

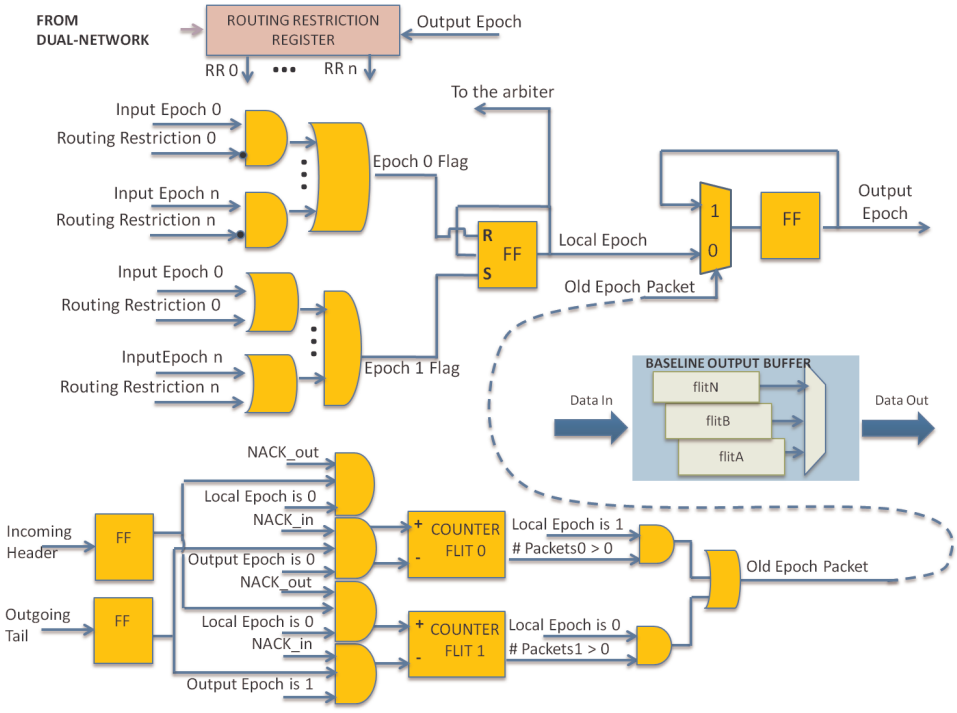


Figure 7.6: OSR-Lite logic extended for the NACK/GO output buffer.

restriction information of the upstream switch can be delivered together with new configuration information through the control network. As a result, the links do not longer need to be extended with 3 additional wires cutting down the area overhead of the OSR-Lite mechanism. Finally, only the epoch token needs to travel in the communication channels. See Figure 7.5 for the final solution.

In order to effectively support frequent reconfigurations, a set of *OR* blocks followed by an *AND* block signals the migration from epoch 0 to epoch 1 (*Epoch_1_Flag*) and, vice versa, a set of *OR* blocks followed by an *AND* block signals the migration from epoch 1 to epoch 0 (*Epoch_0_Flag*). A local register triggered by the *Epoch_1_Flag* and the *Epoch_0_Flag* stores the local epoch value. The local epoch will change when all the input ports with dependencies with the output port have evolved to the new epoch, meaning all these input ports have drained all the old packets from their buffers.

Similarly to input ports also output ports integrate two counters for monitoring incoming and outgoing packets. In this case, a flag is asserted (*Old_Epoch_Packet*) when output ports have migrated to new epoch but their buffers have still not drained from old packets. As soon as, output ports have drained then the *Old_Epoch_Packet* flag drives a multiplexer which enables the propagation of the new epoch token (*Output_Epoch*) to the downstream switch. Figure 7.6 depicts the architecture of OSR-Lite for NACK/GO output ports.

As regards the lightweight OSR-Lite module plugged around the arbiters, it is preserved when a NACK/GO switch is considered instead of its STALL/GO counterpart.

7.3.3 Fault-Tolerant Reconfiguration Mechanism

The OSR-Lite mechanism must be reliable in order to meet the NACK/GO switch requirements. For this purpose, the simplest way to guarantee fault-tolerance was to implement TMR in light of the low-complexity of replicated circuits. Finally, the circuit outputs are voted before feeding the switch internal logic as showed in Figure 7.7. This solution is the same adopted for FSMs of the input and output ports of the switch.

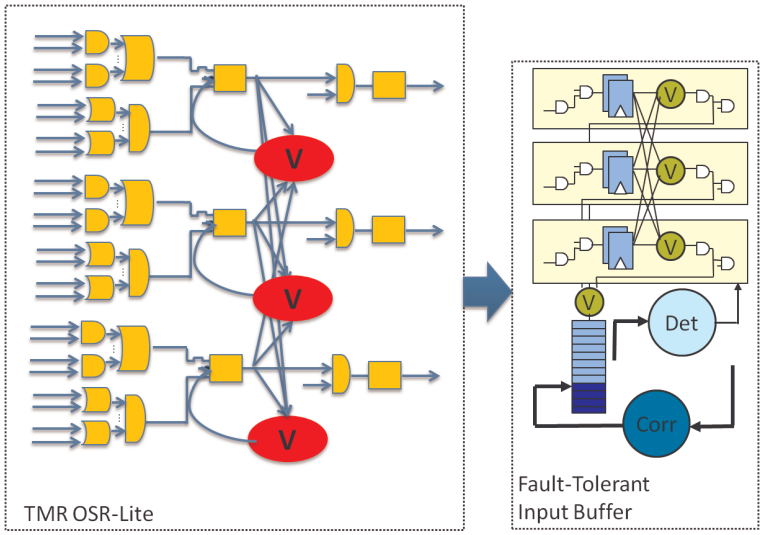


Figure 7.7: Fault-tolerant OSR-Lite logic.

7.4 Switch Extensions for System Level Notification

If a link or a switch experience frequent transient errors with an occurrence probability that exceeds that expected in the case of uncorrelated transient faults, they could be disabled for safety reasons. In order to do this, the switch should be able to know the frequency of error occurrences, to be able to disable itself, and be smart enough to notify to all the neighboring switches its new state, so that the whole system can be reconfigured accordingly. This is a lot of work to be done by an element that should be as simple as possible. In order to avoid large counters and complex control logic inside each switch, we exploit the control network infrastructure to notify a global controller every time a transient fault is detected inside the switch. In Section 7.3, the control network was used to notify configuration bits of the routing mechanism to the switches. The same network could be reused for other purposes, such as congestion management, deadlock recovery and software debugging. In [156] it is showed to be a cost-effective solution for control signaling, which can be easily and effectively made reliable through a combination of fault-tolerant and on-line testing strategies. For this reason, this work relies on such a fault-tolerant control network for transient faults notification.

Transient fault detection can be performed by the error detectors at switch ports or by the TRC blocks and voters in the control path. See Figure 7.8. For

instance, when a Detector of an input buffer reveals an error, the error report contains only the associated input channel. On the other hand, if a detector inside an output buffer reveals an error, not only the associated output channel is notified, but also the input port from where the flit was coming is included in the error report. The reason of this is that, while for an input port the place of origin of the flit is always the output port of the previous switch, the source of a flit inside a switch is not unique, and different errors revealed by different output channels can be caused by the same faulty input port. Signaling the path the flit was following allows the global controller to gather a more accurate and effective information about the status of the switch. Similarly, each replicated arbiter inside the switch has a one-to-one correspondence with a switch output port and its intermittent malfunctioning may lead the controller to disable that output port. On the other hand, each replicated LBDR belongs to a switch input port and its malfunctioning contributes to disable that input port. Notifications from voters of OSR-Lite mechanism take part to the diagnosis of associated ports. Given this, the global controller can compute the frequency of fault events and take the appropriate course of action before a permanent fault shows up.

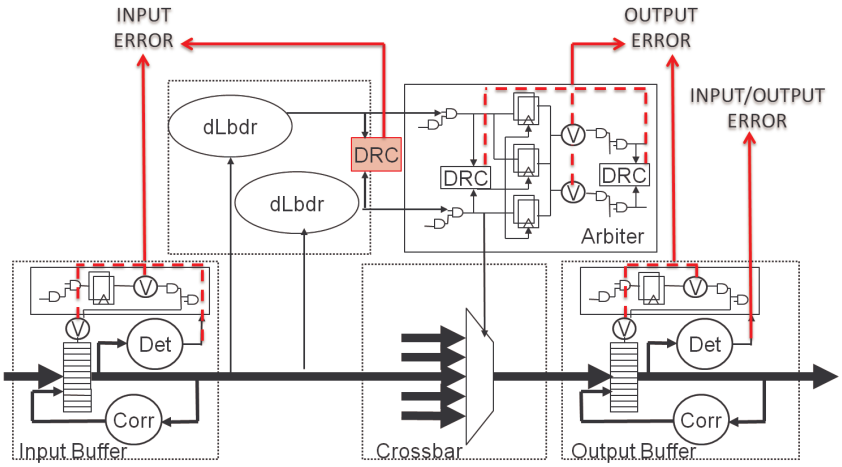


Figure 7.8: Transient fault notification.

Switches and global controller communicate through a dual network. The dual network is composed by a set of replicated *routing primitives*, each one associated with, and connected to, a switch of the main data NoC. Unlike the full featured and richly connected main NoC, the dual NoC implements a straightforward ring topology where the routing primitives are simply cascaded. From an

architecture viewpoint (see Fig.7.9), the routing primitive resembles an over-simplified version of an input buffered clocked switch. An input decoder discriminates between packets destined to the controller (collision between these latter and packets originating from the local switch is solved by an allocator) and those destined to the local switch. All packets on the dual network have a fixed 3 flit length. Flit width is 15 bits. The input buffer has 2 slots (for stall/go flow control management) while the allocator implements a fixed priority algorithm in light of the operating principle of the dual network.

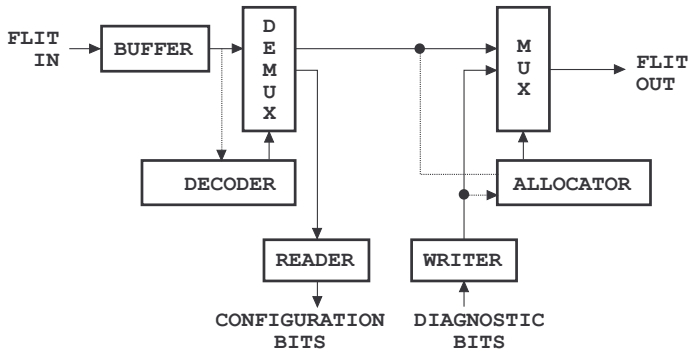


Figure 7.9: Dual network routing primitive.

As an example, the controller transmits configuration bits in 3 flits. The header contains target switch ID and flit type. The second and third flits contain LBDR bits. Whenever the primitive receives a configuration packet, the *reader* interface eliminates the header information and extracts the LBDR bits.

The dual network is a critical component for the correct configuration of the system. Therefore, we decided to provide fault-tolerance capability to it by means of triple modular redundancy (TMR). This latter has been preferred to information redundancy (e.g., error correcting codes, ECCs) for ease of design.

The simplest approach to TMR implementation consists of replicating the dual-network three times and to vote it at the input of the controller interface (for switch-to-controller signaling). This approach, however, has a probability of failure that does not scale well with the size of the network. To overcome this limitation, the scheme in Fig. 7.10 is adopted, where voting is performed at the output of each stage of the dual network. It should be noted that also flow control signals (valid and stall/go) are voted. Voting of the incoming configuration bits is also demonstrated in the figure.

Interestingly, switches are designed to send back the received configuration bits to the controller, so that this latter can check whether the transmission was

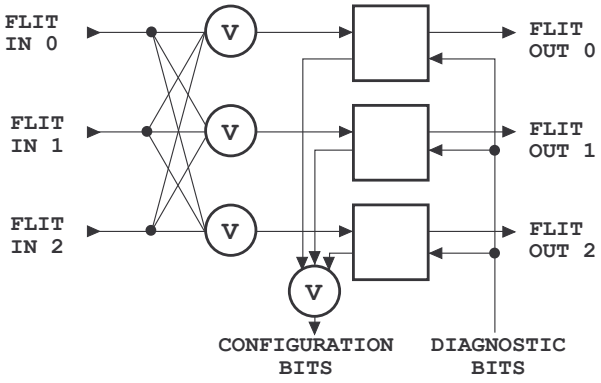


Figure 7.10: TMR approach with per-primitive voting system.

successful. Finally, the reconfiguration procedure is able to detect when a fault affect dual network voters, since configuration bits by the controller would not match the bits that the switch sends back to the controller for a double check (see Chapter 8 for more details).

7.5 The Built-In Self-Testing Framework

This section presents a low area footprint and low-latency testing framework designed by optimizing the built-in pseudo-random self-testing framework presented in Chapter 5 for the microarchitecture of the fault-tolerant and reconfigurable GP-NaNoC switch. The choice of pseudo-random testing over the other strategies was primarily dictated by its potentially lower area overhead and higher flexibility. Following the outcome of Chapter 5, the test time concern has been tackled by reusing test responses of switch sub-blocks as test patterns for cascaded blocks combined with specific test pattern optimizations for selected blocks to preserve coverage.

First of all, we decompose the network (i.e. the switch and the channel) into its building blocks: the arbiters, the crossbar multiplexers, the input buffer, the output buffer, the LBDR, the OSR-Lite mechanism and the link. Then, we exploit a Linear-Feedback-Shift-Register (LFSR) for test pattern generation and Multiple-Input Signature Registers (MISR) for compression of test responses. Finally we cascade switch sub-blocks to the LFSR: then test responses of the upstream block are test patterns for the downstream one. The testing framework is detailed hereafter.

7.5.1 The Data-Path

Data-path includes crossbar, input/output buffers and their intermediate links. We cascaded the circuits of the data-path under test exploiting the synergies between them in order to cut down on test wrapper and TPGs complexity. Especially the cascade is composed by crossbar and output buffer of the upstream switch, input buffer of the downstream switch and inter-switch link. All these elements are jointly tested by means of a single LFSR placed at the beginning of the cascade. Since test responses of one block become test patterns for the next one then the LFSR is responsible for the injection of test vectors for the whole cascade. Such optimization allowed us to exploit a single test-wrapper for the data injection and avoid additional data test-wrappers located in front of the input/output buffer.

In practice, testing of NoC switches is engineered in such a way that all switches are tested in parallel. The key enabler is to implement a cooperation mechanism between switches for testing their inter-switch links. In fact, each switch sends test patterns across its outgoing links and response analysis will be performed in the neighboring switches. The opposite holds for incoming links, which are analyzed locally. See Figure 7.11.

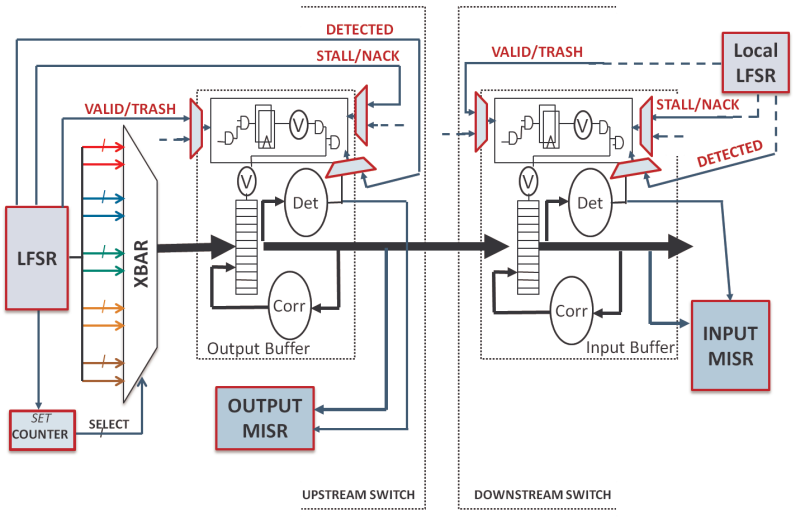


Figure 7.11: Pratical implementation of data-path testing.

When naive application of pseudo-random test patterns from LFSR was adopted to test the cascaded data-path elements a low coverage was achieved.

This coverage result pointed to the need of exploiting the knowledge of elements implementation to increase test efficiency.

Testing multiplexers of the crossbar

The multiplexers of the crossbar are directly fed by the LFSR while its test responses feed the output buffer of the channel. Following the outcome of Chapter 5, interesting optimizations were introduced to limit the LFSR area overhead and to preserve a high fault coverage.

In order to tackle the LFSR area overhead, we fed each input port of the multiplexer with the same 39 pseudo-random bits exploiting a data shift of $7 \cdot N$ bits for every input port. We designed a ring counter driving multiplexer control signals to data transparent configurations (`10..0`, `01..0`, `00..1`).

Testing communication channel

Communication channels include input/output buffers and their intermediate links. All these elements are cascaded to the crossbar and fed by the crossbar test responses. As first optimization to guarantee a high coverage, we increased the controllability and the observability of the channel by driving and reading the incoming and outgoing flow control signals.

In particular, the LFSR drives the *stall*, the *nack*, the *trash* and the *valid* signal of both the input and output buffer. From an implementation viewpoint, there are several practical issues. In fact, the flow control signals of the input buffer, which lies in the downstream switch, should be driven by the LFSR as well. This would require additional wires in the switch-to-switch link. To avoid the extra wires, we opted for the solution in Fig.7.11 where the flow control signals of the input buffer are driven by the LFSR of the downstream switch. From the testing viewpoint nothing changes, since all the pseudo-random LFSRs inject the same patterns synchronously. In order to achieve a high observability, two MISR are adopted. A first MISR observes the outputs of the output port while a second MISR observes the outputs of the input port.

Interestingly, encoded words cross the data path during nominal operation and detector and corrector modules guarantee the correctness of transmitted data. In particular, detectors reveal failures in the data and trigger retransmissions and corrections. During testing time, we could suppose to inject encoded words by means of a LFSR and an encoder. However, this latter solution would never properly stimulate correctors since the encoded words are intrinsically

safe. In order to guarantee an effective testing also for correctors, we inject random bit sequences in the data path. As a result, the corrector will actually find failures to correct when the *NACK* signal is asserted. However, detectors would never allow data to progress through the data-path since errors will be always revealed in the packets. Thus, we randomly control the *detection* flag from detectors to buffer FSMs. As a consequence, the data can progress although the *detection* flag of the detector is asserted. Finally, the *detection* flags are still observed by MISRs to increase detector observability.

7.5.2 The Control-Path

Control-path includes arbiters, routing blocks and OSR-Lite logic. To note that the test of the control-path modules result challenging due to the intrinsic FSMs and combinational logic complexity.

Testing LBDR

Unlike the crossbar module, the LBDR block associated with the port under test lies in the downstream switch. It is fed by the input buffer and it is directly connected to the input MISR. In this case, the input MISR of the data path was extended in order to compress also the LBDR outputs.

The implementation of an effective testing for fault-tolerant LBDRs is not straightforward. As a result, optimizations similar to the ones already proposed in Chapter 5 for the baseline LBDR were introduced to rise the fault coverage of the fault-tolerant routing mechanism (see Chapter 5 for further details).

Testing Arbiters and OSR-Lite logic

The test did not achieve a high coverage when the arbiter and the OSR-Lite logic was directly connected to the existing testing framework following the approach taken so far. The reason lies in the poor efficiency in testing the arbiter FSM and the OSR-Lite combinational logic. As a result, in the final testing framework the arbiter and the OSR-Lite logic are directly driven by the LFSR and its test responses feed dedicated MISRs.

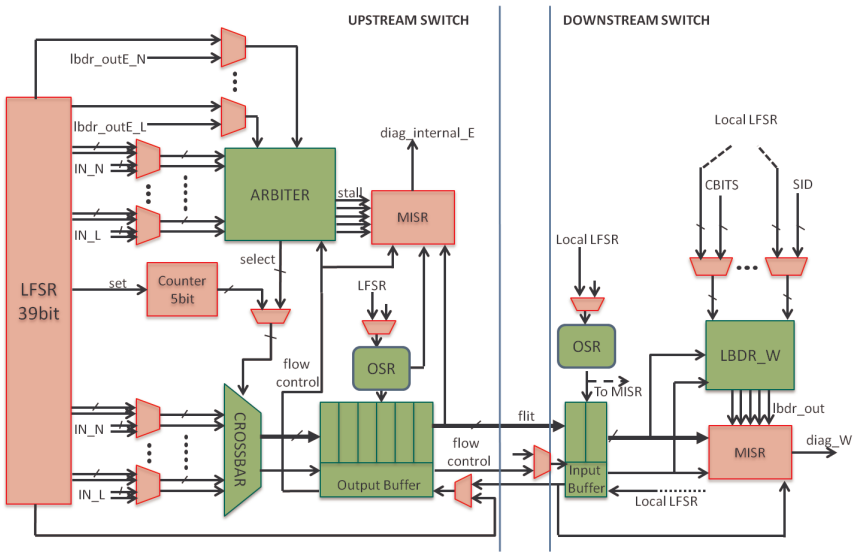


Figure 7.12: BIST-enhanced switch architecture.

7.5.3 BIST-enhanced switch architecture

The switch architecture enriched with the BIST infrastructure is illustrated in Fig.7.12. A test wrapper consisting of multiplexers can be clearly seen, which enables test pattern injection of the LFSR in the modules it tests. A unique 39 bits LFSR generates the pseudo-random patterns to test in parallel every switch port. Moreover two dedicated MISRs for every port collects the test response from the output and input ports. On one hand, the first MISR performs the signature analysis of the test responses from the link, the LBDR blocks and the input buffer together with its OSR-Lite logic. On the other hand, the second MISR performs the signature analysis of the test responses from the crossbar, the output buffer, the arbiter and their associated OSR-Lite logic.

Test diagnosis results in the setting of 10 bits (one for every MISR), indicating whether each input/output port is faulty or not. This meets the requirements of the LBDR configuration algorithm. The testing framework is able to reveal the correct position of multiple faulty channels since a MISR is dedicated to each port.

7.6 Experimental Results

In this section we illustrate the experimental results carried out for the GP-NaNoC switch. Especially, we progressively add features to the switch pointing out the area and routing delay penalty each feature comes with. Finally, we will compare 5 switches:

- The fault tolerant switch.
- The fault tolerant switch with reconfiguration capabilities.
- The fault tolerant switch with reconfiguration capabilities enhanced with control network.
- The fault tolerant switch capable to reconfigure, interact with a control network, perform a built-in self-test at boot-time.
- The reference STALL/GO TMR switch illustrated in Chapter 6 without any support for reconfiguration, notification and testing.

Detector and corrector modules were implemented by means of a Hsiao code. They respectively guarantee a double-error detection and a single error correction exploiting 7 check bits. These latter bits are bundled with the 32 data bits of the network. As previously mentioned, each input and output switch port integrates a detector and a corrector instance. The correctors associated to the input ports correct the data transmitted in the switch data path while the correctors into the output ports correct data routed through the link. Vice versa, the detectors integrated into the input ports detect the errors of the incoming data from the link and the detectors of the output ports detect the errors of the switch data path. The corrector modules have been implemented with a single-stage pipeline, because this is not affecting the critical path. It is anyway worth recalling that the three clock cycles latency penalty required for error recovery (error signaling, flit correction and retransmission) will be paid only on the rare occasions where a transient fault is detected.

All the analyzes discussed in this work have been carried out by means of a low-power standard-Vth 40nm Infineon technology library.

7.6.1 Area and Critical Path

The area results are shown in figure 7.13, while critical path comparison is presented in figure 7.14. Please note that all the results are normalized with

respect to the TMR switch solution.

As far the fault-tolerant Nack-Go switch is concerned, a non-negligible area contribution comes from detector and corrector modules. These count for almost 13% of the total area. In many works in the literature targeting transient faults the problem of data corruption inside buffers is typically omitted, therefore the overhead of correctors is not accounted for. When the reconfiguration mechanism is integrated into the NACK/GO switch, a 11% of area overhead is introduced. Interestingly the OSR-Lite mechanism proposed for the NACK-/GO switch outperforms the baseline OSR-Lite mechanism illustrated in Chapter 6 for a baseline ×pipesLite switch. In fact, the novel OSR-Lite solution introduces a lower area overhead (11% instead of 14%) thanks to the removal of the second routing logic required by the baseline OSR-Lite solution for each port. In addition, the novel mechanism guarantees fault-tolerance, an efficient support for multiple reconfiguration and better preserves the nominal data link width. Concerning the switch enhanced with notification system, it comes with an additional 5% area overhead. In particular, the notification system results lightweight since it takes advantages by the diagnosis logic already made available for fault-tolerance purposes. Finally, the switch capable of built-in self-test and self-diagnosis brings a 27% of area overhead. The area penalty mainly comes from MISRs and test-wrappers. This latter result exposes a area trade-off between the testing frameworks based on pseudo-random and deterministic test patterns proposed in Chapter 5 which respectively introduce a 16% and a 37% area penalty. In fact, the testing framework for the general purpose switch is based on pseudo-random patterns but it exploits a high amount of testing logic to increase the controllability and the observability as the deterministic approach. To note that the testing frameworks in Chapter 5 were designed for

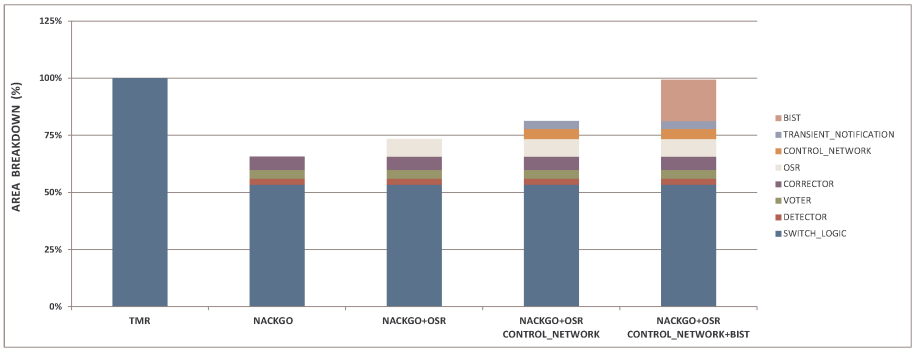


Figure 7.13: Area of the GP-NaNoC switch.

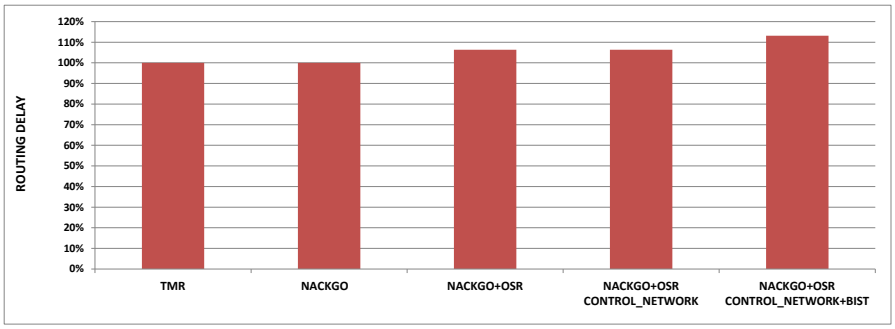


Figure 7.14: Routing delay.

a baseline STALL/GO switch with simpler FSMs and control logic. When we consider a baseline TMR switch which implements only fault-tolerance on top of a baseline xPipeslite switch, we can clearly see that the GP-NaNoC switch (rightmost bar in the plot) provides many more functionalities and provisions at even lower area footprint.

In order to evaluate the effects of each additional feature on the switch routing delay, we performed a 5x5 switch synthesis for maximum performance for all the 5 solutions. The switch with *OSR-Lite* mechanism and the fault-tolerant NACK/GO switch achieved a similar maximum operating speed. As described in Section 7.3, the reconfiguration scheme was designed to avoid long critical path and preserve the baseline switch performance. The OSR-Lite-enabled switch is thus capable of an at-speed reconfiguration. Moreover, the routing mechanism of the OSR-Lite solution scales with network size. In fact, while memory macro-based solutions could suffers from increasing area and delay penalties, the logic complexity of the distributed routing algorithms does not depend on the number of destinations, hence it stays constant. Indeed, the distributed routing algorithms just grow with the switch radix. The maximum operating speed is similarly preserved also when the switch with notification system is considered. As already demonstrated by the area results, the notification mechanism is seamless introduced in the design. Finally, the testing framework degraded by 13% the maximum performance of the NACK/GO switch. The performance of the switch is limited by the test-wrappers placed on the critical path. When we look at the TMR solution, we notice that the GP-NaNoC switch delivers far more functionalities at the cost of a longer critical path (+13%) but within the same area budget.

7.6.2 Optimized Reconfiguration Support

First of all, we assess the efficiency of the routing resource optimization performed in section 7.3 to enable reconfigurability of the routing function. Fig. 7.15 illustrates the normalized area results of three 5x5 switches that make use of a stall/go flow control protocol. From left to right we have respectively the baseline xpipesLite switch, the same switch endowed with a normal OSR-Lite mechanism (two LBDR modules, like in Chapter 6) or with the single-LBDR implementation and support for multiple reconfigurations proposed in this chapter. Clearly, our optimization reduces the area overhead from 14% to 6%. Further support for fault-tolerance would then only amplify the savings, since less logic would have to be replicated.

7.6.3 Coverage for single stuck-at faults

Table 7.1 reports the total number of cells associated to each tested module, and the related achieved coverage. This latter was derived by means of an in-house made gate-level fault simulation framework: faults are applied to any or selected gate inputs, then our testing procedure is run on the affected netlist and the diagnosis outcome is compared with the expected one.

It can be seen that in all cases the coverage for single stuck-at faults exceeds the 90%. While single stuck-at faults in the crossbar, input and output buffer feature a coverage higher than 97%, worse results are obtained for the OSR-Lite mechanism and especially for the LBDR. Their lower coverage is a direct consequence of FSMs and combinational logic complexity. Such complexity brings a gap in terms of coverage between control and data path. In fact, Table 7.2 reports the coverage breakdown of data and control path. The coverage for data path closely tracks the 100% while the coverage for control path tracks

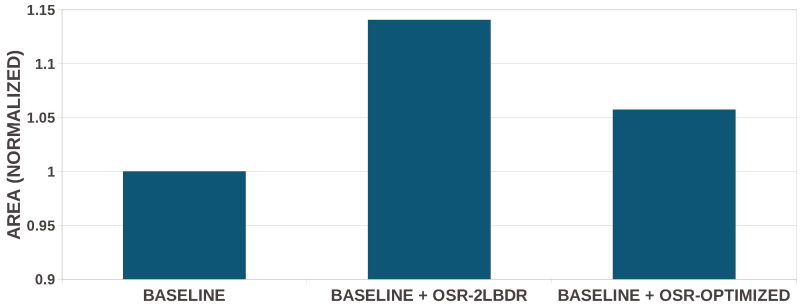


Figure 7.15: Efficiency of Single-LBDR implementation.

Switch sub-block	Cells	Coverage
OSR-Lite	360	95.0%
Arbiter	334	96.7%
Crossbar	154	97.4%
Input Buffer	1272	97.6%
Output Buffer	1855	97.4%
LBDR	289	91.0%
TOT	4264	96.8%

Table 7.1: Coverage for single stuck-at faults.

	Cells	Coverage
Data-Path	2337	99.1%
Control-Path	1927	94.0%

Table 7.2: Coverage breakdown of data and control path.

94%. To note that the output and input buffers belong to the data path of the switch but they internally consist of an actual data path (data registers with selection input demux and output mux) and of a control FSM driving the read and write pointers of the mux and demux. In order to build Table 7.2, these two internal blocks have been separated and respectively associated to data and control path.

Concerning the testing latency, a network composed of the general purpose switches, as assumed so far, would take 10.000 clock cycles for testing, regardless of the network size.

7.7 Conclusions

Modern computing systems require an enhanced degree of (re-)configurability and robustness/tolerance with respect to the uncertainty of the technology platform. For this reason, switch architectures cannot be compared any more only in terms of their microarchitectural parameters, but especially in terms of support or lack of support for such advanced features. The GP-NaNoC switch integrates design methods especially targeted to runtime configurability and reconfigurability of the routing function, to fault-tolerance (permanent faults, transient faults, intermittent faults) and to testing. The GP-NaNoC switch can therefore target the advanced system management requirements of gen-

eral purpose systems such as selective power down of unused or overheated regions, disconnection of malfunctioning components and links, and/or network partitioning and possibly isolation. The proposed switch has provisions for event notification to a central manager and a complete system infrastructure is provided to disseminate reconfiguration commands of the distributed routing logic.

In the next chapter, the GP-NaNoC switch will serve for demonstrating on FPGA the capability of a general purpose NoC to perform boot-time testing, runtime detection of faults, runtime reconfiguration of the routing function and dynamic virtualization.

8

The FPGA Demonstrator

TODAY the converging trend toward multifunction integrated architectures is slowed down by the lack of a proper runtime reconfiguration framework of the on-chip interconnect. A runtime reconfiguration is needed whenever the occurrence of events at runtime causes the need for a different resource allocation, such as in the cases for graceful degradation of system performance, power management, thermal control, etc. This thesis has been focused on developing design methods to introduce such a dynamism into the on-chip network. This chapter reports about the prototyping of such design methods on a Xilinx Virtex-7 FPGA. Boot-time testing and configuration, runtime detection of faults, runtime reconfiguration of the routing function, dynamic virtualization of the interconnect fabric are especially validated on the FPGA prototype, where a 4x4 multi-core system has been implemented and managed. The advanced form of platform control is achieved via hardware/-software co-design and co-optimization.

8.1 Introduction

NoC design principles have recently reached a stage where they start to stabilize, in correspondence to their industrial uptake. In fact, NoCs are an indisputable reality since they implement the communication backbone of virtually all large-scale system-on-chip (SoC) designs in 45nm and below.

On the other hand, the requirements on embedded system design are far from stabilizing and an unmistakable trend toward enhanced reconfigurability is clearly underway. Reconfigurability of the HW/SW architecture would in fact enable several key advantages, including on-demand functionality, on-demand acceleration, shorter time-to-market, extended product life cycles and low design and maintenance costs. Supporting different degrees of reconfigurabil-

ity in the parallel hardware platform cannot be however achieved with the incremental evolution of current design techniques, but requires a disruptive and holistic approach, and a major increase in complexity. At the same time, fault tolerance was previously an issue only for specific applications such as aerospatial. Today, due to the increased variability of components and breadth of operating environments, reliability becomes relevant to mainstream applications. Similarly, new reliability challenges cannot be solved by using traditional fault tolerance techniques alone: the reliability approach must be part of the overall reconfiguration methodology.

In the highly parallel landscape of modern embedded computing platforms, the system interconnect serves as the framework for platform integration and is therefore key to materializing the needed flexibility and reliability properties of the system as a whole. Therefore, time has come for a major revision of current NoC architectures in the direction of increased reconfigurability and reliability.

In addition, a key property that novel NoCs cannot miss is to guarantee a potentially fast path to industry, since NoC deployment is today a reality. An important requirement for this purpose is the efficient testability of candidate NoC architectures. This property is very challenging due to the distributed nature of NoCs and to the difficult controllability and observability of its internal components. When we also consider the pin count limitations of current chips, we derive that NoCs will be most probably tested in the future via built-in self-testing (BIST) strategies.

Finally, there is an increasing need in embedded systems for implementing multiple functionalities upon a single shared computing platform. The main motivation for this are the constraints set for systems size, power consumption and/or weight. This forces tasks of different criticality to share resources and interfere with each other. Integration of multiple software functions on a single multi- and many-core processor (multifunction integration) is the most efficient way of utilizing the available computing power. For a mixed-criticality multifunction integration, the NoC should be augmented to support partitioning and isolation, so that software functions can be protected from unintended interferences coming from other software functions executing on the same hardware platform. This feature is a key enabler for the virtualization of embedded systems, that is, an effective and clean way of isolating applications from hardware.

This chapter reports on the first-time prototyping of a Network-on-Chip capable of supporting all of the advanced features described above. The presented

prototype builds on the GP-NaNoC switch (i.e., the outcome of the design methods presented in this thesis) and raises the level of abstraction to the network as a whole. Then, it validates the (re-) configuration capabilities that preserve safe network operation in the presence of wanted (e.g., virtualization) and unwanted (e.g., manufacturing defects, intermittent faults) effects. The prototyped system implemented inside the FPGA is a homogeneous multicore processor, which resembles programmable hardware accelerators of hierarchical, high-end embedded systems, or basic computation clusters of many-core processors. The validated design methods include:

- Boot-time testing and diagnosis of the 4x4 2D mesh NoC, targeting permanent faults;
- Switch-level and network-level fault-tolerance, targeting transient faults and intermittent faults (i.e., those faults that rapidly anticipate the breakdown of links or switch components);
- Runtime reconfiguration of the network routing function, with logic-based distributed routing as the underlying routing mechanism. The validated reconfiguration procedures are twofold: at boot-time, without background traffic, and at runtime, with background traffic.
- Dynamic virtualization, i.e., partitioning of the whole NoC into isolated partitions running different applications.

The rest of the chapter is organized as follow. The prototyping platform is represented by the Xilinx Virtex-7 evaluation board named VC707, described in Section 8.2. The high-level view of the platform can be found in Section 8.3. This section provides an overview of the design-flow for platform implementation; then it presents the basic components adopted in the demonstrator that is composed of a NoC-based system and a supervision system; finally it describes the application and the reconfiguration algorithm running respectively on top of these latter systems. Next, Section 8.4 validates the boot-time testing and configuration while Section 8.5 demonstrates runtime reconfiguration of the routing function upon transient failures. These sections also highlight the fault-tolerant protocol exploited by the dual-network for control signaling. Last, Section 8.6 describes the dynamic virtualization of the interconnect fabric and Section 8.7 presents the conclusions.

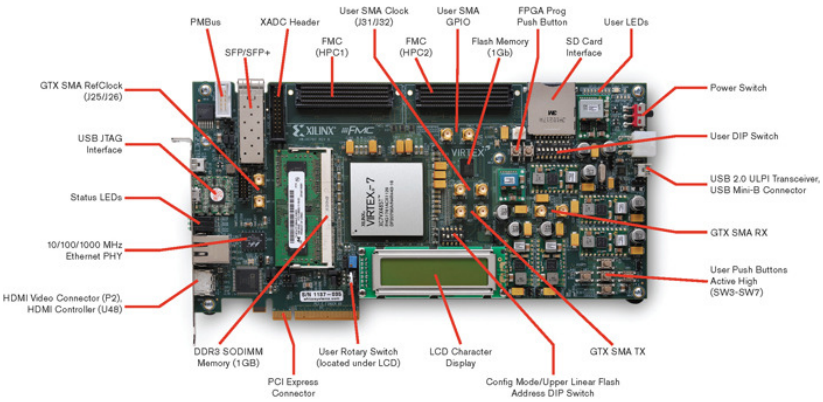


Figure 8.1: VC707 baseline prototyping board.

8.2 FPGA Platform

The target system to prototype is overly complex, hence calling for high-end FPGAs and development boards, not to incur integration capacity limits.

The Virtex-7 FPGA VC707 Evaluation Kit was selected for our task. It is a full-featured, highly-flexible, high-speed serial base platform using the Virtex-7 XC7VX485T-2FFG1761C and includes basic components of hardware, design tools, IP, and pre-verified reference designs for system designs that demand high-performance, serial connectivity and advanced memory interfacing. The included pre-verified reference designs and industry-standard FPGA Mezzanine Connectors (FMC) allow scaling and customization with daughter cards. The XC7VX485T FPGA features 485760 logic cells, 75900 CLB slices, 2800 DSP slices, 37080 kb of block RAM, 14 total I/O banks and 700 max. user I/O.

The key features of the evaluation board (see Figure 8.1) are as follows:

- **GA VC707 Evaluation Kit**: ROHS compliant VC707 kit including the XC7VX485T-2FFG1761 FPGA
- **Configuration**: Onboard JTAG configuration circuitry to enable configuration over USB, JTAG header provided for use with Xilinx download cables such as the Platform Cable USB II, 128MB (1024Mb) Linear BPI Flash for PCIe Configuration, 16MB (128Mb) Quad SPI Flash.
- **Memory**: 1GB DDR3 SODIMM 800MHz / 1600Mbps, 128MB (1024Mb) Linear BPI Flash for PCIe Configuration, SD Card Slot, 8Kb

IIC EEPROM.

- **Communication and Networking**: GigE Ethernet RGMII/GMII,SGMII, SFP+ transceiver connector, GTX port (TX, RX) with four SMA connectors, UART To USB Bridge, PCI Express x8 gen2 Edge Connector (lay out for Gen3).
- **Display**: HDMI Video OUT, 2 x16 LCD display, 8X LEDs.
- **Expansion Connectors**: FMC1 - HPC (8 XCVR, 160 single ended or 80 differential, user-defined pins), FMC2 - HPC (8 XCVR, 116 single ended or 58 differential user-defined pins), Vadj supports 1.8V, IIC.
- **Clocking**: Fixed Oscillator with differential 200MHz output used as the system clock for the FPGA, programmable oscillator with 156.250 MHz as the default output, default frequency targeted for Ethernet applications but oscillator is programmable for many end uses, differential SMA clock input, differential SMA GTX reference clock input, Jitter attenuated clock used to support CPRI/OBSAI applications that perform clock recovery from a user-supplied SFP/SFP+ module.
- **Control and I/O**: 5X Push Buttons, 8X DIP Switches, Rotary Encoder Switch (3 I/O), AMS FAN Header (2 I/O).
- **Power**: 12V wall adapter or ATX, Voltage and Current measurement capability.
- **Debug and Analog Input**: 8 GPIO Header, 9 pin removable LCD, Analog Mixed Signal (AMS) Port.

8.3 The System Under Test

An ambitious Virtex 7 FPGA-based platform was conceived for this reserach project. The high-level view of the design can be found in Figure 8.2.

The system comprises a large number of components within the FPGA. As can be seen on the left side of the diagram, a relatively standard Xilinx subsystem is instantiated first; this comprises an AXI interconnect linking together a MicroBlaze (to run the supervision software), a small memory and an external DRAM controller, and several peripheral controllers required to run software on the MicroBlaze and to communicate with a laptop.

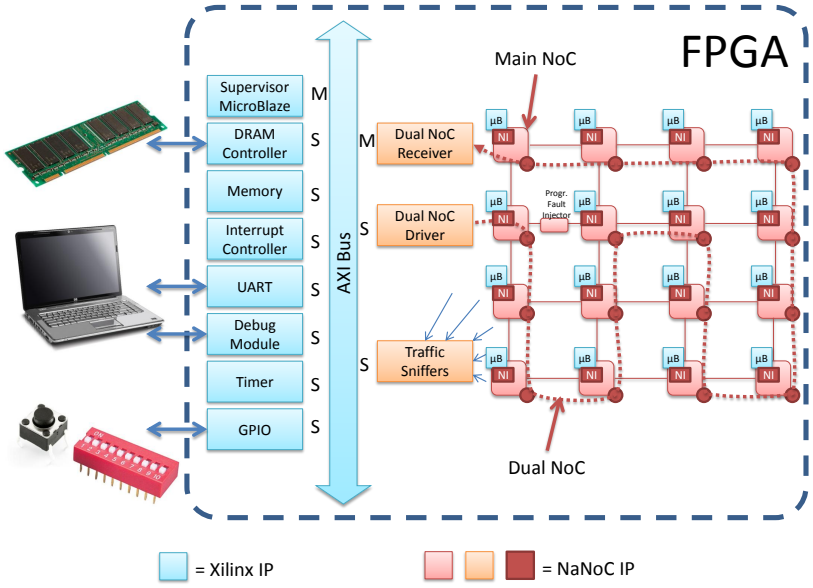


Figure 8.2: FPGA platform overview.

The right side of the diagram depicts the components that have been proposed in this thesis, including some that were specially developed for the FPGA prototype and will be presented in this chapter. This part of the system is the "Device Under Test" (DUT) of the platform, whose functionality is to be verified. It comprises mainly:

- The main NoC, built as a 4x4 mesh of the GP-NaNoC switch architecture presented in the previous chapter.
- The dual NoC, built as a chain that follows the topology of the main NoC. The dual NoC is in charge of configuring the main NoC and of collecting status information (e.g. fault detections) from the main NoC.
- At each node of the main NoC (see also Figure 8.4), a MicroBlaze and a memory (by Xilinx) are connected to the switch by means of Network Interfaces. The MicroBlaze NI has an AMBA AXI NI while the memory is given an AMBA AHB NI.
- Two special blocks have been designed in this thesis to connect the dual NoC to the supervision subsystem. These blocks allow the supervision

MicroBlaze to receive notifications by the dual NoC, and to reprogram it.

- A sniffer module monitors traffic along all links of the main NoC mesh, computing link utilization. It is designed so that the supervision subsystem can probe it at regular intervals and transfer its contents towards a user's laptop.
- A fault injection module has been instantiated along a mesh link. This simple module, connected to a physical button on the FPGA board, provides a method to inject faults on that link to test the platform's fault tolerance and the NoC reconfiguration capability.

To build this platform, we proceed in steps (Figure 8.3). First, we instantiate within Xilinx Platform Studio (XPS) a complete design comprising all the supervision subsystem, the 16 additional MicroBlazes, and the corresponding 16 memories. At this stage, no NoC is instantiated yet. Using XPS for this task allows us to efficiently connect and configure all the Xilinx blocks, and facilitates the instantiation of the toplevel HDL files. Additionally, this makes it possible to subsequently load the applications into all 17 MicroBlazes' memories, and to debug those processor step-by-step, directly through the Xilinx toolchain, which is Eclipse-based. After the first pass of synthesis, however, we remove from the design the Xilinx AXI subsystem which is connecting the 16 additional MicroBlazes and memories, and swap in the NoC (main and dual) in its place. We then proceed to finish the implementation flow within Xilinx ISE by performing mapping, placement and routing, and generating the final bitstream.

We leverage some key features of the Virtex 7 board, apart from the FPGA chip. The on-board DRAM is used to provide sufficient space for the software running on the supervision MicroBlaze to work. Physical buttons and switches of the board are connected to an on-chip GPIO controller to allow the user to interact with the platform. Finally, a laptop can be connected to the board by means of two cables to monitor the platform's operation; one cable carries serial port signals (piggybacked onto a USB port) and the other carries JTAG signals (also piggybacked onto a USB port). The former is used to read the board's outputs, while the latter allows for programming the board and interactively debugging the on-FPGA MicroBlazes. An Ethernet cable had initially been considered instead of the serial interface, in light of its higher throughput, but the Ethernet PHY of the board was found to be defective.

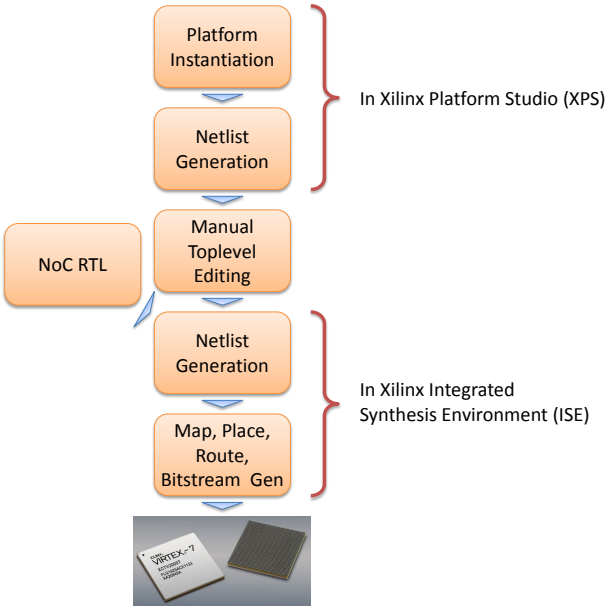


Figure 8.3: Design flow for platform implementation.

Custom-written software runs in three locations of the system: on the supervision MicroBlaze, on the 16 MicroBlazes connected to the main NoC, and on the external laptop.

- The software on the supervision MicroBlaze is tasked with oversight of the main NoC and data NoC, with regular polling of the Traffic Sniffers, and with interfacing with the external world through the serial interface.
- The 16 MicroBlazes connected to the mesh run micro-benchmarks developed in this thesis. These micro-benchmarks have the main role of generating traffic on the mesh, so that the various platform features can be tested. Real functional behaviour was implemented: the nodes perform pipelined matrix multiplications, exchanging data in producer-consumer fashion. More advanced applications could not be implemented due to the lack of I/O interfaces on these nodes and due to lack of memory to instantiate a full C library.
- The user's laptop is connected to the board through a JTAG-over-USB cable and a serial-over-USB cable. The former can be leveraged mainly by the Xilinx toolchain, allowing for board programming and debug-

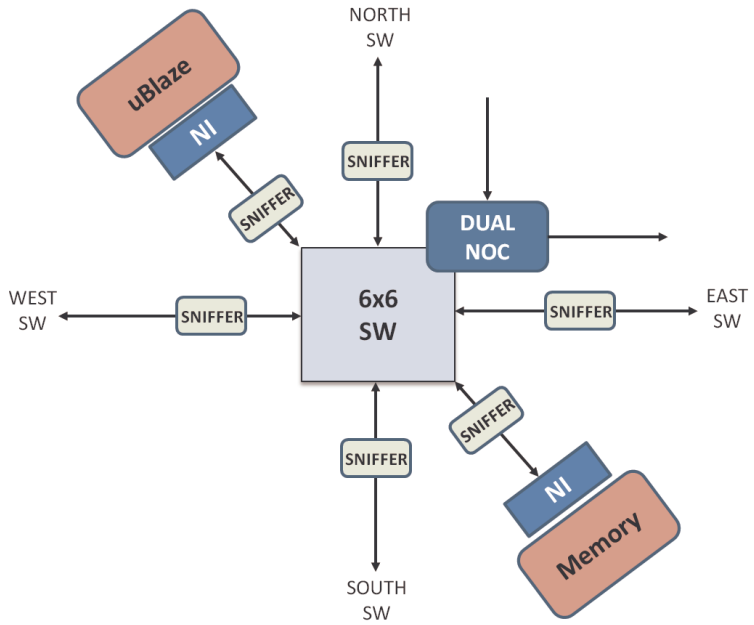


Figure 8.4: Basic components of the on-chip network.

ging. The latter is monitored to display in real-time the platform status and link utilization.

8.3.1 Basic components: the on-chip network

A 4x4 mesh with one core and one memory per switch has been chosen as target on-chip network of the FPGA platform. In particular, Figure 8.4 represents the basic components instantiated to realize the 4x4 mesh. A MicroBlaze and a memory are connected to each switch through two Network Interfaces. Finally, a sniffer is placed on each bidirectional network link to monitor the network traffic. The sniffers collect information about the traffic crossing the switch-to-switch and NI-to-switch links and deliver such information to the global manager (i.e., the supervision MicroBlaze).

Both the NIs and the switches have been designed ad-hoc to support the target on-chip network where fault-tolerance, testing capability and reconfigurability features are guaranteed.

Note that the MicroBlaze also includes a directly-connected BRAM of 128 kB (not shown in the figure) to store its application software; loading the binary

image of the application into the AHB memory would be unnecessarily problematic from the toolchain viewpoint. However, we explicitly use the AHB memory as storage and for inter-processor communication in the application (Section 8.3.4).

The Network Interfaces

We instantiate two types of NIs: an AXI initiator NI to interface with the MicroBlaze, and an AHB target NI to interface with the memory. This choice was deliberate (e.g., both could have been AXI) to demonstrate interoperability among the two.

Due to the relatively simple needs of the MicroBlaze core, which does not support multiple transaction IDs, we save area by instantiating a small AXI initiator NI with support for only one such ID. However, the NI is still supporting all AXI features. Both AXI and AHB NIs, and their interoperability, were extensively tested in RTL and on the FPGA.

For integration into the platform, a few tweaks to the NI were needed:

- NIs embed routing tables to statically perform source routing. In this platform, routing is distributed and reconfigurable to work around faults or to enforce virtualization. Therefore, the routing tables are modified to instead encode the XY coordinates of the destination core; these will be processed at the switches. The coordinates are expressed as strings of 9 bits: 4 bits for each coordinate (slightly overprovisioned for a 4x4 mesh) plus one bit to differentiate among the two local cores at each node, i.e. MicroBlaze and memory.
- The input and output buffers of the NIs are extended to support the NACK-GO flow control protocol used by the GP-NaNoC switches proposed in this thesis.
- The AXI initiator NIs are extended with two extra pins, directly connected to FPGA pads, in turn connected to physical switches of the FPGA board. This means that the user, manually flipping those switches, can change the value of two bits inside each NI. The NI in turn exposes these two bits to the MicroBlaze at the reserved address `0x11000000`. The MicroBlaze can poll this location to change among operating modes, e.g. staying idle, or executing one of multiple pre-programmed applications. As can be inferred from Figure 8.2, note that in the plat-

form, the 16 MicroBlazes attached to the mesh have no way to communicate with the external world except for this facility (although 7 of them are also connected to the Debug Module - see Section 8.3.2).

The switch

The GP-NaNoC switch architecture adopted in this work is an extension of the GP-NaNoC switch presented in the previous chapter. The switch implements logic-based distributed routing (LBDR [152]), relies on wormhole switching and implements both input and output buffering. The crossing latency is thus 1 cycle in the link and 1 cycle inside the switch. This section briefly summarizes the key features of the switch together with the extensions introduced to meet the target FPGA platform requirements.

Routing logic extension. As previously mentioned, the switch implements logic-based distributed routing by means of LBDR logic modules. The LBDR modules natively support a single core per switch. Thus this latter logic had been extended for the target FPGA platform where two cores belong to each switch, a MicroBlaze and a memory.

In principle, the LBDR selection logic computes the destination output port by reading the destination address information contained in the header flit of each packet. In particular, the LBDR logic performs a comparison between the destination address (*Dest_ID*) and the local switch ID (*Local_ID*). When the local switch ID matches the destination address, the packet is forwarded to the local port (i.e., to the core). However, the FPGA platform is enhanced with two nodes per switch thus each switch integrates two local ports. As a result, further information must be added to the incoming destination address of the header flit and the LBDR logic must be extended to determine whether the packet should be routed to the first or the second local port. The destination address information has been extended by 1 bit (*Core_Flag*). The additional bit is exploited to determine the target core at the destination switch. If *Dest_ID* matches *Local_ID*, the *Core_Flag* bit is used to distinguish between the two local ports. Figure 8.5 shows the logic gates added to the native LBDR logic.

8.3.2 Basic components: the supervision subsystem

In order to demonstrate the NoC functionality, a supervision subsystem is required. We choose to instantiate it within Xilinx Platform Studio, and using as

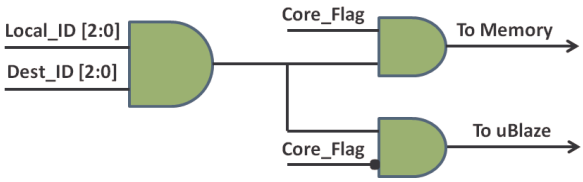


Figure 8.5: LBDR routing logic extension for two cores per switch support.

many Xilinx IP cores as possible, for convenience. The subsystem (see Figure 8.2) includes a Xilinx AXI interconnect, with two masters (a Microblaze and the Dual NoC Receiver) and numerous AXI, AXI Lite and AHB slaves.

At the heart of this subsystem is a Microblaze running software developed in this thesis. This software is tasked with:

- Probing the status of the NoC, e.g. after BIST and upon fault occurrences.
- In response to the above, configuring or reconfiguring the NoC.
- Awaiting for possible user requests to reconfigure the NoC in a virtualized manner.
- In response to the above, reconfiguring the NoC.
- Polling the link sniffers periodically to monitor activity on the NoC links.
- Transferring key information about the platform's functioning outside the FPGA through the serial port (or, potentially, an Ethernet port).

To perform these actions, multiple support controllers and devices are needed. First of all, since the supervision software and the required underlying C library have a non-negligible footprint, incompatible with on-chip resources, a DRAM controller is advisable to be able to store the software. To support the basic functionality of the Xilinx C library, a timer and an interrupt controller must also be present. (Note that, in contrast, the 16 Microblazes connected to the NoC mesh do not have access to external memory, timer or interrupt controller; this limits the capabilities of the software that can be run on those).

In order to monitor the NoC, it is necessary for the Microblaze to be able to access the dual NoC. This is done via three components plugged to the

AXI bus: a Dual NoC Driver, a Dual NoC Receiver, and a memory. The first two blocks have been designed in this thesis to exchange packets in the specific format expected by the dual NoC. The Microblaze can directly write to the Dual NoC Driver, which is a slave on the AXI bus, to program the main NoC. Due to the way the dual NoC was designed, the reverse operation cannot be done with a read to the same device; instead, whenever there is a message requiring attention (e.g., upon BIST completion or fault detection), the dual NoC sends a packet to the Dual NoC Receiver, which converts it into an AXI transaction directed at the on-bus memory (a standard Xilinx core). The Microblaze can periodically poll this memory to check all notifications.

To supervise the NoC activity, the Microblaze can also poll the Traffic Sniffers. These blocks can be connected to up to 16 links of the main NoC on one side, and to the AXI bus on the other. For maximum thoroughness, we choose to monitor as many as 80 links of the NoC (almost all, disregarding just a few whose information is redundant), with five Traffic Sniffers in parallel. The sniffers include a counter that is incremented at the passage of any flit; whenever the counter is read by the MicroBlaze, it automatically resets itself. A simple division yields a utilization metric.

Finally, the FPGA needs external interfaces. First of all, a GPIO controller allows the Microblaze to periodically check the status of a few physical buttons and switches on the FPGA board. This allows the user to change operating modes of the platform; for example, we use this feature to instruct the software on the Microblaze to initiate the reconfiguration to get into virtualized application mode. Two extra blocks are used to communicate with the user's computer. A UART controller is an output-only interface that allows the platform to transfer information to the laptop. A debug module, relying on a JTAG-over-USB electrical connection, allows for bidirectional communication: the user can program the supervision Microblaze, step through its software, and check the content of certain on-FPGA registers and memories. Since the debug module allows for monitoring of up to 8 Microblazes, we connect it to the supervision Microblazes and to selected 7 other Microblazes out of the 16 attached to the main NoC mesh.

8.3.3 Basic components: the reconfiguration algorithm

The supervisor MicroBlaze is constantly monitoring the status of the NoC through the dual NoC. Whenever a notification is received about a fault on a link, if deemed necessary (e.g. unless it is assumed to be a transient), the supervisor triggers the reconfiguration algorithm. This algorithm computes

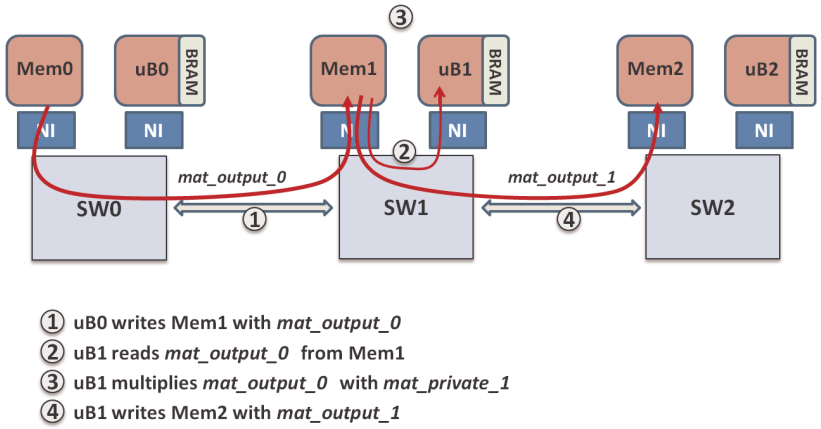


Figure 8.6: The matrix multiplication at work.

the required changes in the LBDR bits at specific switches in order to migrate from the current routing algorithm to a new one that avoids the use of the notified link. The algorithm is a simple access to two precomputed tables containing all the changes for every possible link failure. Those bits are encoded and transmitted through the dual NoC together with a triggering notification to switches to launch the reconfiguration process (the generation of tokens and their advance through the network).

8.3.4 The application

The MicroBlazes have been programmed in order to start their application after the 4x4 mesh is configured, upon flipping a physical switch on the board. The application run by the MicroBlazes is a matrix multiplication consisting of the product of a pair of matrices. The MicroBlazes sequentially forward the results to each other in a pipelined producer-consumer fashion. Each MicroBlaze performs the multiplication of a private matrix and a matrix delivered by the previous MicroBlaze of the sequence. Once the matrix product is computed the resulting matrix is forwarded to the next MicroBlaze. The lack of I/O interfaces and memory does not allow the implementation of more advanced applications.

The private matrix (`mat_private`) is stored by each Microblaze into its local (pertaining to its local switch) AHB scratch memory of 4kB and it is simply initialized as follows:

```
for (row = 0; row < LINES; row ++)
  for (column = 0; column < LINES; column ++)
    (*mat_private)[row][column] = row + column;
```

Notice that the matrix size can be tuned by means of the *LINES* parameter. Moreover the same matrix multiplication can be set to run for a specific number of times or in an endless fashion. The AHB memory is used as storage and for inter-processor communication in the application. Indeed the incoming matrix from the previous Microblaze (*mat_input*) is stored in the AHB memory connected to the local switch. Each MicroBlaze stores its matrix product result (*mat_output*) into the AHB memory connected to the switch to which the next MicroBlaze of the sequence is connected. The first MicroBlaze of the pipeline initializes its own local AHB memory before performing the matrix product. Each MicroBlaze has local registers storing the address of the local AHB memory, the addreses of the remote AHB memory of the next MicroBlaze, the position within the pipeline, and the pipeline length. Figure 8.6 depicts the application at work.

The application run by each Microblaze is simply the following:

```
for (i = 0; i < LINES; i ++)
 for (k = 0; k < LINES; k ++) {
  (*mat_output)[i][k] = 0;
  for (j = 0; j < LINES; j ++)
   (*mat_output)[i][k]+=(*mat_input)[i][j]*(*mat_private)[j][k];
  }
```

In order to guarantee the synchronism between the MicroBlazes, custom semaphores are implemented. Interestingly, these are purely software and do not need dedicated hardware support. Such a solution slightly increases the complexity of the code but clearly simplifies the hardware design effort and the area overhead. Of course this approach is possible only since the application is fixed and known upfront; more sophisticated synchronization capabilities would demand hardware-level atomicity support.

The goal of these semaphores is to avoid reading the same incoming matrix multiple times, and to avoid overwriting output matrices before the next MicroBlaze has been able to process them. Each MicroBlaze has been enhanced with 4 semaphores:

- *Semaphores_read* is stored in the local AHB memory. It is polled (and therefore set) by the local MicroBlaze, while it is reset by the next MicroBlaze of the sequence to notify that it is available to receive new input. As soon as polling reveals that the semaphore is reset, the local

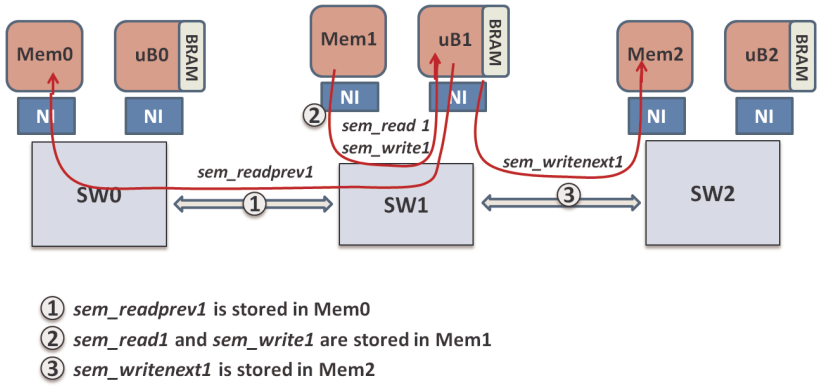


Figure 8.7: The semaphores of the matrix multiplication application.

MicroBlaze can forward the matrix product result to the next MicroBlaze.

- `Semaphores_write` is stored in the local AHB memory. Similarly to the previous semaphore, it can be polled and set by the local MicroBlaze while it is reset by the previous MicroBlaze of the sequence to notify that a new input matrix is incoming.

- `Semaphores_readprev` is stored in the AHB memory of the previous MicroBlaze. The current MicroBlaze resets it as soon as it has read the matrix incoming from the previous MicroBlaze.

- `Semaphores_writenext` is stored in the AHB memory of the next MicroBlaze. The current MicroBlaze reset it as soon as it has forwarded the matrix product result to the next MicroBlaze.

Notice that semaphores to be polled have been placed in the local AHB memory in order to reduce congestion in the network. Figure 8.7 shows the semaphores location.

Since hardware support to semaphores has not been implemented, special care was devoted to guarantee the proper semaphore initialization. Indeed `Semaphores_write` and `Semaphores_readprev` needs to be respectively initialized to 1 and 0. Also, semaphore initialization must occur in synchronism otherwise the first MicroBlaze could start operations before the next MicroBlaze is actually ready. Thus, the initialization sequence progresses in reverse order with respect to the pipeline: each MicroBlaze only initializes its

own semaphores once the semaphores of the next MicroBlaze have been already initialized. The last MicroBlaze of the pipeline is therefore the first allowed to initialize itself, while the first will be the last. This ensures that the matrix multiplication starts only when all the MicroBlazes are properly initialized.

8.3.5 The physical platform implementation

Some steps of the implementation flow described in Figure 8.3 can be parallelized; for example, the initial platform description involves several blocks which can be independently synthesized in parallel. Even after joining all the pieces together, the mapping stage can be run on two threads in the Xilinx toolchain, and the placement and routing in four. Despite this, we measure end-to-end flow runtimes of about 7 hours on a dual-chip Opteron 6378 (16 threads/core) server with 128 GB of RAM. We observe peak memory utilization close to 10 GB during implementation. The layout of the platform implementation can be seen in the screenshot of Figure 8.8.

The platform fills the FPGA almost completely, as can be seen in Table 8.1. The left column reports the utilization when the template system generated in XPS is implemented, the right one is for the same system where the NoC, dual NoC and associated NaNoC IP (e.g. sniffers, dual NoC interface blocks, etc.) have been instantiated to replace the simple AXI interconnect. It can be seen that the NoC represents approximately 17% of the FPGA's sequential resources and 66% of the combinational resources (or a little bit more, since this is the overhead on top of the default bus); it does not occupy any BRAM nor require external pins.

Due to development timing constraints, no specific optimizations could be taken to reduce the area of the design; given the large number of blocks and the redundancy features (e.g. triplication of some components, BIST, datapath encoding) built into the NoC, we perceive the resource utilization figures as very positive. Note that triplicated logic in the adopted switch has to be marked with special annotations embedded in the RTL, otherwise the Xilinx synthesis tools would detect it as redundant and prune it away.

The design is not aimed at, and not optimized for, high performance. The very high resource utilization features also impose a significant timing overhead as routing necessarily becomes more convoluted and less efficient. We record a maximum operating frequency of 38 MHz; the critical path is in the BIST logic of the switch.

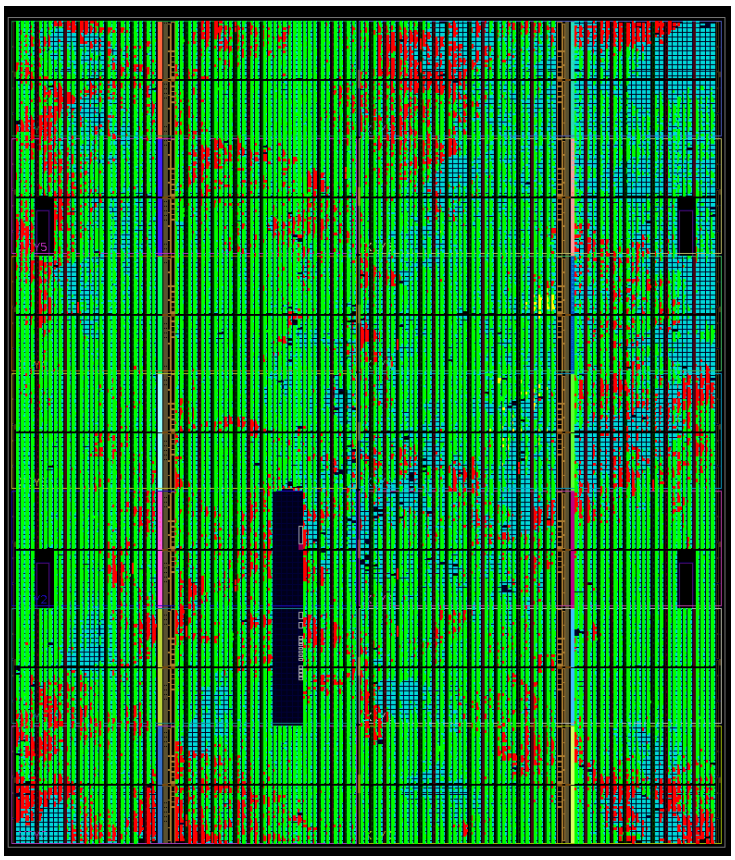


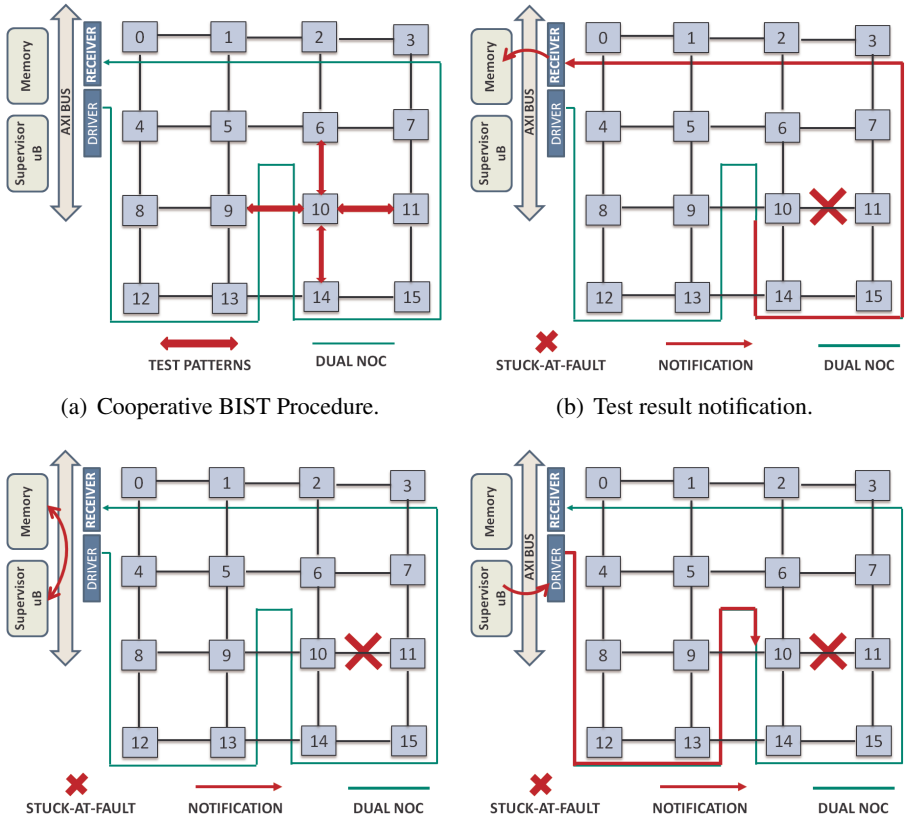
Figure 8.8: Layout of the full FPGA design. Green: data NoC; red: Network Interfaces; yellow: dual NoC; cyan: MicroBlazes and other logic.

Resource utilization	Supervisor subsystem only	Full platform
Slice Registers	5%	22%
Slice LUTs	16%	88%
IOs	20%	20%
36-bit BRAMs	61%	61%

Table 8.1: Resource utilization of the Virtex 7 chip.

8.4 Validating Built-in Self-Testing and NoC configuration

In order to validate the Built-in Self-Testing implemented in the 4x4 mesh, a permanent failure was forced in the network by hard-wiring to zero a link wire. In this implementation, the failure was injected in the link between switch 11 and 10. However, it could have been freely injected in different locations since the 4x4 mesh has been based on a switch that guarantees around 97% of stuck-at-fault coverage.



(a) Cooperative BIST Procedure.

(b) Test result notification.

(c) Supervisor polls the memory and computes configuration bits

(d) Network configuration

Figure 8.9: Built-in-Self-Testing at work.

As soon as the FPGA board is booted the BIST automatically starts and the switches cooperatively exchange test patterns as shown in Figure 8.9(a). When

the BIST procedure is completed, the BIST managers integrated in each switch send to the dual NoC the diagnosis information related to the switch they belong to. In the FPGA platform under test, the BIST manager of Switch 10 reveals an error on its East input channel where the error has been injected. Thus it notifies the dual NoC, which takes care of delivering all the BIST diagnosis information to the global manager (i.e., the supervision MicroBlaze). In particular, the diagnosis information crosses the dual NoC and the Dual NoC Receiver before being stored in the memory connected to the supervision subsystem (Figure 8.9(b)).

The supervision MicroBlaze has been programmed to periodically check for dual NoC notifications by polling the control bus memory (Figure 8.9(c)). It recognizes when the BIST notification information has been stored in the memory (i.e., the BIST procedure is completed) and it runs the configuration algorithm described in Section 8.3.3. Thus, it computes configuration bits able to guarantee deadlock-free routes despite the failed link. The configuration bits are sent to the Dual NoC Driver through the AXI bus. They cross the dual NoC and configure the routing mechanism of each switch (Figure 8.9(d)).

8.4.1 Protocol for BIST notification and configuration

The 4x4 mesh is properly configured only once the configuration bits are sent twice by the supervision MicroBlaze. The exchanged information follows a sophisticated protocol envisaged to meet multiple requirements:

- The dual NoC is a highly simplified version of the main NoC. The information travels in small packets composed of 2 or 3 flits as a function of the notification type. Only head and tail flits are required when a notification related to transient errors is delivered.
- The information must be exhaustive in order to allow the supervision MicroBlaze to take a wide range of diagnosis and recovery actions. As an example, it should be able to identify the kind of fault (`fault_type`) that has occurred (permanent or transient) and the address of the failed switch (`address_sender`).
- The notification information should be able to pinpoint the exact location of the error inside the switch (`bist_result`) to enable an effective network configuration.
- The protocol must be intrinsically fault tolerant to guarantee a reliable

communication. Thus, the information is sent multiple times or encoded within the packets.

- The configuration bits generated by the supervision MicroBlaze must match the specifications of the switch routing mechanism (Section 8.3.1).

In the case of BIST notification and configuration the protocol consists of 4 phases. In the first phase, the switches notify the BIST results by means of three 22-bit flits with the following format:

Head:
flit_type(2), *priority*(2), *address_sender*(8), *unused*(10);
Body:
flit_type(2), *fault_type*(2), *ctrl*(2), *bist_result*(10), *unused*(6);
Tail:
flit_type(2), *ctrl_ng*(2), *fault_type_ng*(2), *bist_result_ng*(10), *unused*(6);

Note that the tail flit contains the body information in its negated version for fault tolerance reasons.

During the second phase, the supervision MicroBlazes sends the configuration bits in compliance with the LBDR specification by means of three flits per switch:

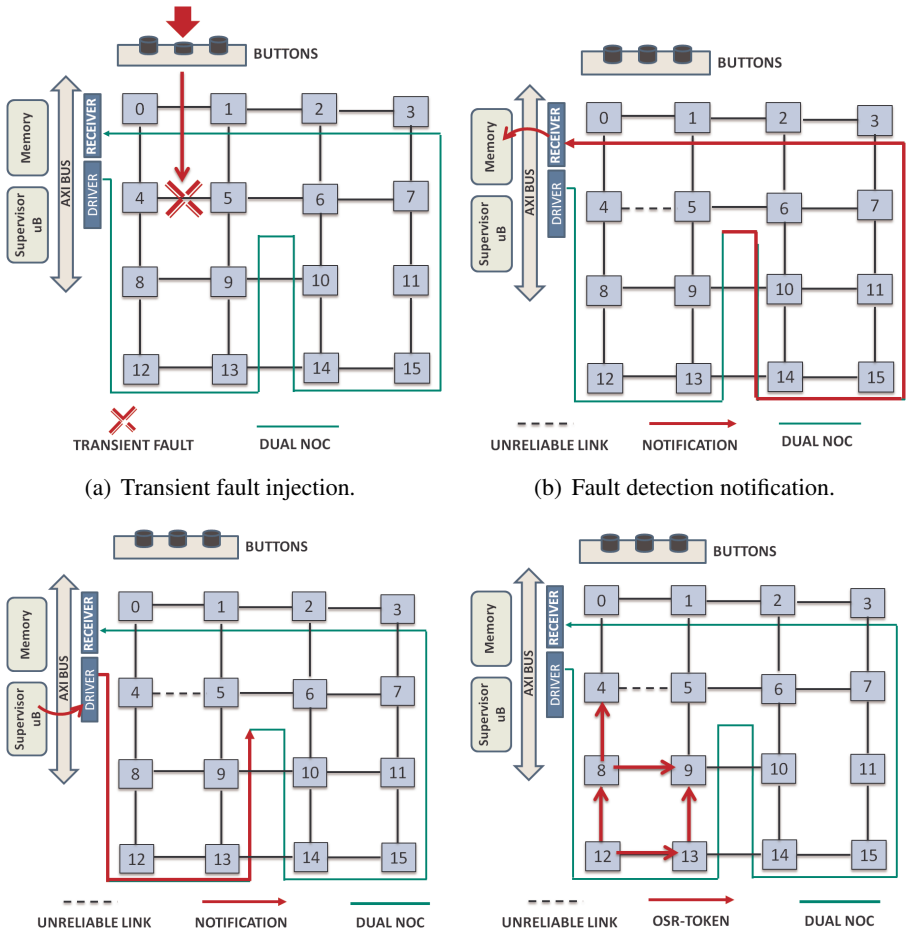
Head:
flit_type(2), *priority*(2), *address_receiver*(8), *unused*(10);
Body:
flit_type(2), *ctrl*(2), *lbdr_configuration*(18);
Tail:
flit_type(2), *lbdr_configuration*(20);

In the third phase, the switches forward back to the MicroBlaze the configuration information just received following the same above reported format.

In the last phase, the MicroBlaze checks the incoming information to ensure that the configuration information delivered in the second phase has been correctly received by the switches. If the delivery is confirmed as safe, the MicroBlaze notifies the switches by sending for the second time a replica of the second phase information.

Once the 4 phases have been executed, the 4x4 mesh is configured and the matrix multiplication application described in Section 8.3.4 can run properly.

8.5 Validating Fault Detection and NoC Reconfiguration



(a) Transient fault injection.

(b) Fault detection notification.

(c) Notification of network reconfiguration.

(d) Overlapped static reconfiguration (OSR-Lite) at work.

Figure 8.10: Transient fault detection and reconfiguration.

Once the network has been tested and permanent faults have been detected

and tackled by the off-line configuration, the system can be still affected by run-time transient and intermittent faults. Such faults cannot be handled by off-line strategies as they appear and disappear unpredictably. As a result, the network has been designed as fault tolerant to satisfy the high reliability constraints imposed by modern systems. In particular, the fault-tolerant flow control protocol (NACK/GO) is used on the data path to notify error detection and trigger data retransmissions. Although this protocol has been primarily designed to tackle SEUs (Single Event Upset), the system is also able to tackle physical effects such as wear-out. Indeed wear-out effects end up in permanent faults but they are known to have a gradual onset. In practice, frequent transient faults affecting the same circuitry denote the possible onset of a permanent fault. Before this happens, the network routing function could be modified to exclude the affected circuit from communication traffic. NACK/GO lends itself to such a policy, since its retransmission and/or voting events may be notified to the supervision MicroBlaze which may monitor the distribution and frequency of transient faults over time and eventually take the proper course of recovery action. This exact policy is supported and validated by the FPGA platform.

Physical buttons and switches of the board are connected to an on-chip GPIO controller to allow the user to interact with the platform. The physical buttons have been leveraged to inject transient faults in the network and validate the above mentioned fault tolerance policy. For this purpose, a fault injection module has been instantiated along the link routed from switch 4 to 5. This module is connected to a physical button on the FPGA board and provides a method to inject transient faults on that link (see Figure 8.10(a)). Since the link may be idle when the button is pressed, the fault injection module integrates a simple FSM that waits until a valid flit is crossing the link to inject the fault, ensuring that actual important information is corrupted. Therefore, every time the button is pushed, the error is revealed and notified to the supervision MicroBlaze (Figure 8.10(b)).

Similarly to the procedure followed by the BIST notification described in Section 8.4, the transient notification crosses the dual NoC and it is stored into the control bus memory. The supervision MicroBlaze periodically polls the memory also during run-time operations. Thus it reads the transient notification and updates its register with distribution and frequency of transient faults over time. Only when the number of transient notifications from the same link reaches a threshold its recovery action taken. For the sake of the demonstration, the MicroBlaze's software is set to run its reconfiguration procedure after three notifications (i.e., after the button has been pushed three times). Note that

the 4x4 mesh at this stage is irregular since a link has been already disabled due to a previously detected permanent failure. Thus, the algorithm computes the reconfiguration bits for this irregular network and delivers them to the dual NoC (Figure 8.10(c) and Section 8.3.3).

The new reconfiguration bits coming from the dual NoC cannot directly update the existing routing strategy, as during off-line operations, since applications are now running. Thus, the network implements the OSR-Lite reconfiguration mechanism which avoids stopping or draining network traffic during the transition from one network configuration to another. As described in Chapter 6, the switches of the FPGA platform start to inject tokens into the network. The tokens follow the channel dependency graph of the old routing function and progressively drain the network from old packets, as represented in Figure 8.10(d).

8.5.1 Protocol for transient notification and reconfiguration

As described in the case of BIST notification, also transient notification and reconfiguration follow a sophisticated protocol able to meet the platform requirements underlined in Section 8.4.1. The protocol consists again of 4 phases. The protocol moves to the second phase only when the switches notify multiple times a transient fault on the same link/switch port. Otherwise, the first phase is repeated until the notification threshold value is reached. The transient notification is forwarded to the supervision MicroBlaze by means of two flits with the following format:

Head:
flit_type(2), *priority*(2), *address_sender*(8), *time_info*(10);
Tail:
flit_type(2), *fault_type*(2), *ctrl*(2), *trans_pos*(10), *unused*(6);

Differently from BIST notification, the head flit contains information about the error time occurrence (*time_info*), which can be useful to compute the frequency of the transient. The *trans_pos* field pinpoints the location of the fault within the switch.

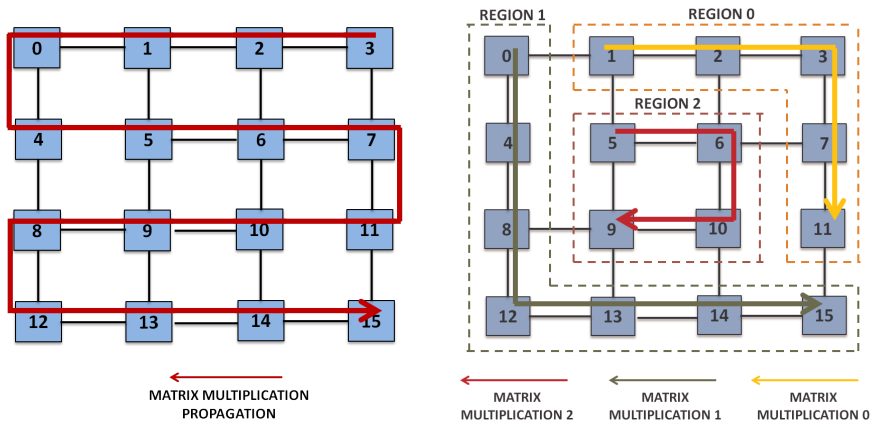
When the number of transient notifications reaches the threshold the second phase starts. The MicroBlaze sends reconfiguration bits following the same format of the off-line configuration, previously described:

Head:

flit_type(2), *priority*(2), *address_receiver*(8), *unused*(10);
Body:
flit_type(2), *ctrl*(2), *lbdr_configuration*(18);
Tail:
flit_type(2), *lbdr_configuration*(20);

As during configuration, the switches forward back the reconfiguration information to the MicroBlaze (third phase) and finally the latter sends for a second time a replica of the reconfiguration information (fourth phase). Once the 4 phases have been executed, the 4x4 mesh starts to inject the OSR-Lite tokens to drain the network and migrate to the new routing strategy.

8.6 Validating NoC Virtualization

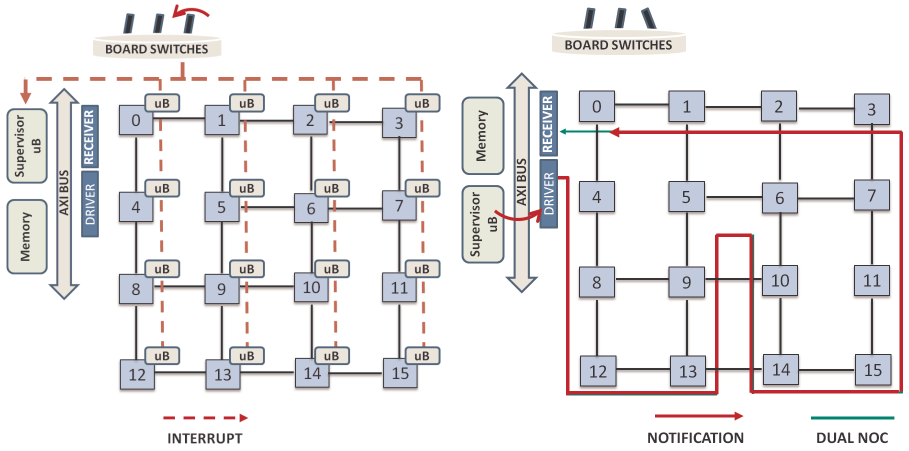


(a) (b)

Figure 8.11: Network regions before (a) and after virtualization (b). Note that the arrows indicate the logical application flow, not necessarily the route followed by packets. For case (b), the arrows are only indicative of partitioning, but the pipeline sequence is in fact shuffled for verification purposes.

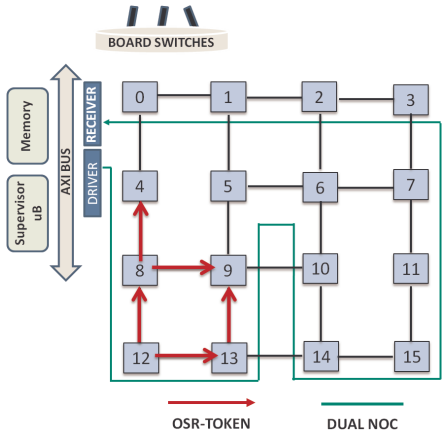
The 4x4 mesh of the FPGA platform is also able to support virtualization. In order to validate such a feature, the platform is virtualized at run-time. In particular, the network has a single region when the operations start after the boot procedure. Thus, the same matrix multiplication application is run by all the 16 MicroBlazes and it propagates from Switch 0 to Switch 15 in a zig-zag

fashion, as illustrated in Figure 8.11(a). Since virtualization is not in place, packets follow the best deadlock-free path available. Note that packets do not strictly respect the arrow of Figure 8.11(a) but they are routed in compliance with the actual routing strategy which takes into account also the faults of the considered scenario. If the user requires a virtualization, the network is split in three regions, as represented in Figure 8.11(b). In this new mode, a dedicated matrix multiplication application runs into each region and packets are constrained to respect *virtual boundaries*.



(a) Notification of virtualization request.

(b) Notification of network reconfiguration.



(c) Overlapped static reconfiguration (OSR-Lite) at work.

Figure 8.12: Virtualization request and reconfiguration.

In order to enforce virtualization at the user's request, external interfaces are required. Dedicated physical switches on the board serve this purpose. These are directly connected to the initiator NIs of the 16 data NoC MicroBlazes (Section 8.3.1). All 16 MicroBlazes periodically check the status of the physical switches. When they detect a change, they transition operating mode, interrupting the current application, reading the address map for the new matrix multiplication sequence of Figure 8.11(b), and finally running the new application.

The status of the physical switches can be polled by the supervision MicroBlaze too, through the GPIO controller. Therefore, in parallel, the software on the supervision MicroBlaze detects the request to get into virtualized application mode (Figure 8.12(a)). The reconfiguration bits required to virtualize the network are then computed and sent through the dual NoC (Figure 8.12(b)). In particular, the routing algorithm modifies the connectivity bits of the switch routing functions to shape the network in three virtual regions. The token propagation of the OSR_Lite reconfiguration mechanism is triggered (Figure 8.12(c)) as described in the previous section. The packets are eventually forced to propagate within the region boundaries.

Note that the sequence of the application propagation of Figure 8.11(b) is only indicative; if the pipeline were as depicted, communication would be strictly point-to-point across a single hop, and virtualization would be unnecessary. To verify its effectiveness, each of the three pipelines has been shuffled, so that producer-consumer traffic would tend to stray out of the geometric shape of each region, if not for virtualization enforcement. As a consequence, it has been possible to prove that the packets do not violate the region constraints even when the theoretical best deadlock-free path goes through a neighboring region.

The protocol adopted to communicate the reconfiguration bits for virtualization matches that for transient reconfiguration (Section 8.5.1). However the protocol is 3-phase only; the first phase is missing since the reconfiguration is triggered by external physical switches instead of by a network notification.

8.7 Conclusions

This chapter reports on the prototyping of a 16-core homogeneous multi-core processor with a reliable, runtime reconfigurable and dynamically virtualizable on-chip network exploiting the design methods proposed in this thesis. The prototyped system has been successfully validated in its capability of boot-

time testing and configuration, transient or intermittent fault detection, runtime reconfiguration of the routing function, and dynamic partitioning and isolation proving the effectiveness and the maturity of the proposed design methods.

The validated NoC prototype is a key enabler for the future evolution of embedded systems. First, it enables the integration of multiple software functions on a single multi- and many-core processor (multifunction integration). This is the most efficient way of utilizing the available computing power. Second, the proposed design methods enable advanced forms of platform control, especially power management and thermal control. In fact, parts of the network can be easily powered off, while preserving its global functionality and guaranteeing safe transitions between network configurations. Third, the proposed technology paves the way for an effective graceful degradation of the system in the presence of permanent and intermittent faults. The routing function can be reconfigured at runtime to avoid faulty links and switches. Finally, a comprehensive reliability framework has been set into place, from switch-level to network-level, while covering all design aspects (e.g., reliable control signaling) and effectively co-optimizing different architectural features together (fault-tolerance, testing, runtime reconfiguration, control signaling).

9 Conclusions

TODAY, microelectronic system design, as never before, is evolving under the influence of its two main drivers, the broadening complexity of applications and the opportunities along with the uncertainties of nanoscale technologies. On one hand, while technology is providing unprecedented levels of system integration, it is also also bringing new severe concerns (overheating, power budget, permanent and transient faults). On the other hand, the complexity of applications calls for large-scale SoCs and support for an increasing number of functionalities that must be materialized by advanced interconnect fabric providing high communication bandwidth together with an enhanced degree of dynamism and flexibility.

NoCs, as mainstream industrial interconnect solution, are generally believed to be the answer to such challenges. However relevant parameters such as supported topologies, switching technique, flit size, buffering styles, supported routing algorithms, etc. cannot longer represent the key differentiation between network-on-chip architectures. On the contrary, we are at the stage where the features of the on-chip network must match with the new complex requirements driven by application and technology scaling constraints that are out-of-reach of current NoC realizations.

The constraints introduced by technology scaling require design methods able to provide fault-tolerance and testability to tackle the uncertainties of aggressive technology node and design methods able to support locally synchronous, globally asynchronous frequency domains to meet the power budget restrictions and the overheating concerns. Finally, NoCs must be envisioned to support combinations of applications that can run in several modes and that can be executed concurrently in a system changing over time. Such requirements call for design methods able to support system virtualization, partitioning and isolation capabilities of system resources.

This thesis is a timely answer to the above concerns. The thesis has identified the basic design requirements needed to augment NoC architectures with an enhanced degree of dynamism and flexibility and to let them successfully cope with the increasing uncertainty of the technology platform. Such requirements have been thoroughly investigated throughout the thesis leading to novel design techniques that have been integrated in a single NoC architecture. The thesis validates the novel design techniques while at the same time proves their co-existence in the same switching fabric. In essence, the thesis contributes to the evolution of the NoC concept providing architectures for the next-generation of multi-synchronous, reliable and reconfigurable embedded systems relying on:

- Built-in self-test and diagnosis frameworks, to address post-production and lifetime permanent failures.
- Support for frequent and dynamic reconfiguration processes.
- Reliable synchronization in a multi-frequency environment.

The NoC proposed in this thesis addresses all the modern requirements, by co-designing all the above mentioned new features on top of a fault-tolerant NoC architecture. All these features have never been addressed all together, in a single NoC. Moreover, interdependencies between different NoC features have been detected ahead of time so to avoid the engineering of highly optimized solutions to specific problems that however coexist inefficiently together in the final NoC architecture. Finally, the proposed design methods have been validated by means of ASIC implementation and FPGA prototyping. The robustness of multi-synchronous methods against process variability and operating conditions have been proved in the first multi-synchronous ASIC testchip in 40 nm CMOS process while non silicon-dependent design methods for reliability and reconfiguration have been validated on a leading-edge Virtex-7 FPGA.

This chapter is structured in three sections. Section 9.1 summarizes the work presented in this thesis. In Section 9.2, we present the major contributions of the thesis.

9.1 Summary

This dissertation began by providing the necessary background of the work in Chapter 2. It surveyed the architectures of globally-asynchronous locally

synchronous interfaces for the building of GALS systems, the strategies for built-in self-testing and fault-tolerance for designing reliable systems and the reconfiguration mechanism for NoCs. Finally, it summarized the shortcoming of the presented work to be addressed in subsequent chapters.

Chapter 3 introduced the synchronization design issue. In a first step, the motivation for the adoption of synchronization mechanisms in the Network-on-Chip environment was discussed. Next, it was presented the target GALS platform of this thesis along with the architecture of the basic switch block required to build it. Last, it is presented the baseline mesochronous synchronizer and the dual-clock bi-synchronous FIFO with focus on all their improvements that led to a new fully integrated and flexible switch architecture.

Chapter 4 presented the implementation of the Moonrake chip in 40nm CMOS process. The design flow followed for the different GALS systems integrated in the Moonrake chip have been illustrated highlighting compatibility with mainstream standard cell libraries and design toolflows. Next, the tests successfully performed on Moonrake chip have been illustrated. Last, the chapter presented the testchip results evaluating the NoC subsystems in terms of skew and frequency robustness, power consumption and yield.

Chapter 5 provided an exploration of built-in testing strategies customized for NoC-based systems comparing them under an area, coverage and latency point of view. In particular, the chapter proposed four testing design methods. Deterministic-handcrafted, deterministic-algorithmic and pseudo-random test patterns have been considered and enhanced with different diagnosis mechanism based on ROM, signature analysis and comparator trees. The testing design methods are provided for both fully-synchronous and multi-synchronous networks.

Chapter 6 presented the implementation of a novel fast and transparent reconfiguration mechanism called OSR-Lite. The goal of this chapter was to provide a reconfigurability design method matching the requirements of next-generation embedded systems. The proposed OSR-Lite mechanism is a static reconfiguration process localized at link/router level and engineered to fit area and performance NoC constraints. The chapter described the complete reconfiguration mechanism at work and the implementation at micro-architecture level of the logic enabling the mechanism and the propagation of the reconfiguration tokens.

Chapter 7 collected the outcome of the previous chapters and proposed the next-generation GP-NaNoC switch. Testability and reconfigurability methods have been co-designed together with fault-tolerant strategies in a single optimized architecture. They co-existed together in a runtime reconfigurable

switching fabric supporting network partitioning and isolation as well as irregularities stemming from power/thermal/fault-tolerance management frameworks.
Chapter 8 reported about the prototyping of the design methods proposed in the previous chapters on a Xilinx Virtex-7 FPGA. The FPGA prototype implemented a 4x4 multi-core system and comprised a large number of components that enables observability, controllability and debugability of the system under test. The 4x4 multi-core platform has been described in detail together with the advanced form of platform control and the testcases validating the design methods.

9.2 Major Contributions

This section discusses the major contributions of our study.

- **GALS design methods:** Among several implementation variants, we selected a source synchronous design style as the choice for implementing our target GALS platform. Moreover, in order to migrate from a synchronous to a GALS design style at a negligible area and power cost, we opted for mesochronous synchronization within the network domain. In light with this, we designed a novel mesochronous synchronizer that is fully merged with the switch input buffer. We explored the design space of the mesochronous link by characterizing the skew tolerance of such architecture as a function of several NoC parameters such as switch radix, interswitch link length, switch operating frequency. Beyond mesochronous synchronizers, dual-clock FIFO architectures are still required at IP core boundaries. Thus we designed a library of dual-clock FIFOs for cost-effective MPSoC design, where each architecture variant in the library has been designed to match well-defined operating conditions at the minimum implementation cost. Similarly to the proposed mesochronous synchronizer also the dual-clock FIFO is fully merged with the switch input buffer. In particular, each FIFO synchronizer is suitable for plug-and-play insertion into the NoC architecture and selection depends on the performance requirements of the synchronization interface at hand. Critical-path, area, power consumption and throughput have been provided for each dual-clock FIFO variant of the library. Finally, the mesochronous and the dual-clock FIFO synchronizer have been

integrated in a testchip, the Moonrake chip. This latter represents the first implementation of synchronizer-based GALS NoC technology in 40nm CMOS process. The Moonrake chip validated on silicon the robustness of the multi-synchronous design methods proposed in the thesis. The new synchronization technology was successfully tested and ported to a new manufacturing process. The experience validated the feasibility and effectiveness of the developed multi-synchronous NoC concept in nano-scaled technology sub-systems bridging the final gap to actual silicon implementation.

- **Testability and reconfigurability design methods:** We provided support for static irregularities performing an exploration of testing strategies for NoC-based systems. In this direction, we presented four scalable built-in self-test and self-diagnosis infrastructures for NoCs providing a wide exploration of different testing strategies. Customizations for the NoC environment of conventional testing strategies were proposed taking full advantage of the intrinsic network structural redundancy through cooperative testing and diagnosis frameworks. First, the NoC switch was enhanced with a conventional scan chain-based mechanism. Although the mechanism was customized to reduce the test pattern number, we proved that the area overhead was non affordable and the scan-based approach is unsuitable for use within a built-in self-testing strategy in NoCs. Second, the switch was tested by means of conventional pseudo-random patterns generated by a built-in LFSR module. The pseudo-random pattern framework proved an appealing solution for a low-area highly-flexible NoC solution, provided the devised customizations are applied. Third, a low-latency alternative based on handcrafted-deterministic test patterns was proposed. Lower flexibility and higher area overhead with respect to the pseudo-random counterpart was achieved. As a conclusion, the deterministic test pattern-based strategy represents the best solution for high-performance high-reliability NoCs. Finally, we proposed one of the first BIST/BISD framework for GALS Network-on-Chips. In particular, we extended the proposed pseudo-random pattern framework through an asynchronous handshaking protocol on bi-synchronous channels.

  Next, we provided design methods for NoC reconfigurations. Thus we proposed an overlapped static reconfigurations strategy called OSR-Lite. It represents a non-intrusive and efficient mechanism to allow the routing algorithm to change uninterruptedly over system lifetime (to match

associated changes in the connectivity pattern) while always remaining deadlock-free. OSR-Lite enables such a deadlock free network reconfiguration without stopping network traffic or draining the network.

Last, the thesis integrated the proposed design methods with fault-tolerant strategies to provide support for dynamic irregularities. The adopted fault-tolerant strategies are based on the Nack/go flow control protocol. Nack/go envisions retransmission of corrupted flits on the datapath, thus minimally affecting the critical path of the switch (no error correction on it), while introducing only a few cycles overhead in the rare cases of actual retransmissions. While Nack/go effectively protects the datapath, we devised a joint fault-tolerance framework that exploits the Nack/go flow control protocol to use dual modular redundancy instead of the triple modular one in the control path

As a result, the most relevant and innovative design methods conceived throughout the thesis are integrated into the GP-NaNoC switch. This latter captures the interdependencies between different NoC design methods and co-optimizes them in a single effective architecture. The GP-NaNoC switch targets the advanced system management requirements of next-generation systems such as selective power down of unused or overheated regions, disconnection of malfunctioning components and links, and/or network partitioning and possibly isolation. The proposed switch has provisions for event notification to a central manager and a complete system infrastructure is provided to disseminate reconfiguration commands of the distributed routing logic.

To conclude, we reports on the prototyping of a 16-core homogeneous multi-core processor on a Xilinx Virtex-7 FPGA with a fault-tolerant, runtime reconfigurable and dynamically virtualizable on-chip network exploiting the GP-NaNoC switch. The prototyped system has been successfully validated in its capability of boot-time testing and configuration, transient or intermittent fault detection, runtime reconfiguration of the routing function, and dynamic partitioning and isolation proving the effectiveness and the maturity of the proposed design methods. The validated NoC prototype is a key enabler for the future evolution of embedded systems.

Bibliography

[1] L. Benini, G. De Micheli, “Networks on Chips: a New SoC Paradigm”, IEEE Computer 35(1), pp. 70–78, 2002.

[2] Andriahantenaina, A. et Al, ”SPIN: a Scalable, Packet Switched, On-chip Micro -network,” DATE03, 2003, pp. 70 73.

[3] Jian, L.; Swaminathan, S.; Tessier, R. ”aSOC: A Scalable, Single -Chip communications Architecture.“ PACT 2000, pp.37-46.

[4] Dally, W.; Towles, B. ”Route packets, not wires: on-chip interconnection networks,” DAC01, 2001, pp. 684-689.

[5] Moraes, F. et Al., ”A Low Area Overhead Packet-switched Network on Chip: Architecture and Prototyping.” VLSI-SOC, 2003.

[6] Erland Nilsson, ”Design and Implementation of a hot-potato Switch in a Network on Chip,” Master Thesis, IMIT/LECS 2002

[7] Sgroi, M. et Al., ”A. Addressing the System-on-Chip Interconnect Woes Through Communication-Based Design.” DAC01, 2001, pp. 667 672.

[8] Karim, F.; Nguyen, A.; Dey S. ”An interconnect architecture for network systems on chips.” IEEE Micro v. 22(5), Sep.-Oct. 2002, pp. 36-45.

[9] M.Radetzki, C.Feng, X.Zhao, and A.Jantsch. ”Methods for fault tolerance in networks on chip.“ ACM Computing Surveys - 2012

[10] Dall’Osso, M. et Al. ”Xpipes: a latency insensitive parameterized network-on-chip architecture for multiprocessor SoCs” ICCD 2003.

[11] Rijpkema, E.; Goossens, K.; Radulescu, A. ”Trade Offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip.“ DATE03, Mar. 2003, pp. 350-355.

[12] Jose Flich, Davide Bertozzi ”Designing Network On-Chip Architectures in the Nanoscale Era“ 2010 by Chapman and Hall/CRC

[13] D.Ludovici, et Al. ”Assessing fat-tree topologies for regular network-on-chip design under nanoscale technology constraints.“ DATE 2009.

[14] E. Flamand, “Strategic Directions Towards Multicore Application Specific Computing”, Proceedings of Design, Automation and Test in Europe (DATE’09), pp. 1266, 2009.

[15] D. A. IIitzky, J. D. Hoffman, A. Chun and B. P. Esparza, "Architecture of the Scalable Communications Core's Network on Chip", IEEE MICRO, 2007, pp. 62-74.

[16] M. Mishra and S. Goldstein, "Defect tolerance at the end of the roadmap", in ITC, pages 1201-1211, 2003.

[17] J.Raik, V.Govind, R.Ubar, "DfT-based External Test and Diagnosis of Mesh-like NoCs", IET Computers and Digital Techniques, October 2009.

[18] J.Raik, V.Govind, R.Ubar, "An External Test Approach for Network-on-a-Chip Switches", Proc. of the IEEE Asian Test Symposium 2006, pp.437-442, Nov. 2006.

[19] J.Raik, V.Govind, R.Ubar, "Test Configurations for Diagnosing Faulty Links in NoC Switches", Proc. ETS, 2007.

[20] Panda et al., "Design, Synthesis and Test of Networks on Chips", IEEE Design and Test of Computers, vol.22, issue 8, pp.404-413, 2005.

[21] S.Y.Lin, C.C.Hsu, A.Y.Wu, "A Scalable Built-In Self-Test/Self-Diagnosis Architecture for 2D-mesh Based Chip Multiprocessor Systems", IEEE Int. Symp. on Circuits and Systems, pp.2317 - 2320, 2009

[22] B.Vermeulen, J.Delissen, K.Goossens, "Bringing Communication Networks on a Chip: Test and Verification Implications", IEEE Communications Magazine, vol.41-9, pp.74-81, 2003.

[23] V.Bertacco, D.Fick, A.DeOrio, J.Hu, D.Blaauw, D.Sylvester, "VICIS: A Reliable Network for Unreliable Silicon", DAC 2009, pp.812-817.

[24] D.Wentzlaff et al., "On-Chip Interconnection Architecture of the Tile Processor", IEEE Micro, vol.27, no.5, pp.15-31, 2007.

[25] "Tilera Tile-Gx Product Brief," 2011. Available: `http://www.tilera.com/products/processors/TILE-Gx_Family`

[26] K.Petersen, J.Oberg, "Utilizing NoC Switches as BIST-Structures in 2D Mesh Network-on-Chip", Future Interconnects and Network on Chip Workshop, 2006.

[27] A. Pullini, F. Angiolini, D. Bertozzi, L. Benini, "Fault Tolerance Overhead in Network-on-Chip Flow Control Schemes," Proceedings of 18th

Annual Symposium on Integrated Circuits and System Design (SBCCI) 2005, Florianpolis, Brazil, Sep 4-7, 2005, pp. 224-229

[28] Young Hoon Kang, Taek-Jun Kwon, Jeffrey Draper. "Fault-Tolerant Flow Control in On-Chip Networks." Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip.

[29] Nicopoulos, C. ; Kim, J. ; Vijaykrishnan, N. ; Das, C.R. "Exploring Fault-Tolerant Network-on-Chip Architectures ". International Conference on Dependable Systems and Networks, 2006.

[30] W. J. Dally and B. Towles, Principles and practices of interconnection networks: Morgan Kaufmann, 2003.

[31] W. J. Dally, L. R. Dennison, D. Harris, K. Kan, and T. Xanthopoulos, "Architecture and implementation of the reliable router," Proc. of the Hot Interconnects II, pp. 197-208, 1994

[32] Nicolaidis M., "Design for soft error mitigation", IEEE Transactions on Devices and Materials Reliability, vol. 5, Issue 3, pp. 405-418, 20

[33] Yu Q., Ampadu P. "Adaptative Error Control for NoC Switch-to-Switch Links in a Variable Noise Environment." IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems, pp. 352-360, 2008.

[34] Rossi D., Metra C., Nieuwland K., Katoch A., "Exploiting ECC Redundancy to Minimize Crosstalk Impact," IEEE Desing & Test of Computers, vol. 222, Issue1, pp. 59-70, 2005.

[35] Frantz A., Kastensmidt F., Cota E., Carro L., "Dependable Network-on-Chip Router Able to Simultaneously Tolerate Soft Errors and Crosstalk." Proceedings International Test Conference (ITC), vol. 1, pp. 1 9, 2006.

[36] Adn Kohler, Gert Schley, and Martin Radetzki "Fault Tolerant Network on Chip Switching With Graceful Performance Degradation.", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, June 2010.

[37] Concatto, Caroline; Matos, Debora; Carro, Luigi; Cota, Erika; Kastensmidt, Fernanda; Susin, Altamiro. "Fault Tolerant Mechanism to Improve Yield in NoCs Using a Reconfigurable Router", Symposium on Integrated Circuits and Systems Design (SBCCI), 2009.

[38] D. Rossi et al., ”Configurable Error Control Scheme for NoC Signal Integrity,“ Proceedings 12th IEEE International On-Line Testing Symposium, LOS ALAMITOS, M. Nicolaidis, A. Paschalis, 2007, pp. 43 - 48.

[39] D. Bertozzi et al., ”Error control schemes for on-chip communication links: The energy-reliability tradeoff, IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 24, no. 6, pp. 818831, Jun. 2005.

[40] S. Murali et al., ”Analysis of error recovery schemes for networks on chips, IEEE Design Test Comput., vol. 22, no. 5, pp. 434442, Sep. Oct. 2005.

[41] Kypros Constantinides, Stephen Plaza, Jason Blome, Bin Zhang, Valeria Bertacco, Scott Mahlke, Todd Austin and Michael Orshansky. ”BulletProof: A Defect-Tolerant CMP Switch Architecture” International Symposium on High-Performance Computer Architecture (HPCA), Austin, TX, February 2006

[42] K.Peterson, J.Oberg, ”Toward a Scalable Test Methodology for 2D-mesh Network-on-Chip”, DATE 2007, pp.75-80, 2007.

[43] Y.Zorian, ”Embedded Memory Test and Repair: Infrastructure IP for SoC Yield.”, International Test Conference, pp.340-349,2002.

[44] Y.Zorian, ”Testing the monster chip”, IEEE Spectrum, pp.54-60,1999.

[45] A.M. Amory, E.Briao, E.Cota, M.Lubaszewski, F.G.Moraes, ”A Scalable Test Strategy for Network-on-Chip Routers”, Proc. of ITC 2005.

[46] C.Grecu, P.Pande, A.Ivanov, R.Saleh, ”BIST for Network-on-Chip Interconnect Infrastructures”, VLSI Test Symposium, page 6, 2006.

[47] R.Ubar, J.Raik, ”Testing Strategies for Network on Chip”, in Book: ”Network on Chip”, edited by A.Jantsch and H.Tenhunen, Kluwer Academic Publisher, pp.131-152, 2003.

[48] O. Lysne, J. Montanana, J. Flich, J. Duato, T. Pinkston, and T. Skeie, “An efficient and deadlock-free network reconfiguration protocol,” *IEEE Transactions of Computers*, vol. 57, no. 6, pp. 762–779, 2008.

[49] W. Dally, L. Dennison, D. Harris, K. Kan, and T. Xanthopoulus, "The reliable router: A reliable and high-performance communication substrate for parallel computers," in *Proceedings of the Workshop on Parallel Computer Routing and Communication (PCRCW)*, May 1994, pp. 241–255.

[50] C. Glass and L. Ni, "Fault-tolerant wormhole routing in meshes without virtual channels," *IEEE Transactions Parallel and Distributed Systems*, vol. 7, no. 6, 1996.

[51] M. Gómez, J. Duato, J. Flich, P. López, A. Robles, N. Nordbotten, O. Lysne, and T. Skeie, "An efficient fault-tolerant routing methodology for meshes and tori," *Computer Architecture Letters*, vol. 3, no. 1, pp. 3–3, January-December 2004.

[52] C.-T. Ho and L. Stockmeyer, "A new approach to fault-tolerant wormhole routing for mesh-connected parallel computers," *IEEE Transactions on Computers*, vol. 53, no. 4, pp. 427–439, 2004.

[53] K. M. et al., "Fibre channel switch fabric-2 (fc-sw-2)," NCITS 321-200x T11/Project 1305-D/Rev 4. 3 Specification, Tech. Rep., March 2000.

[54] M. Schroeder, A. Birrell, M. Burrows, H. Murray, R. Needham, T. Rodeheffer, E. Satterthwaite, and C. Thacker, "Autonet: a high-speed, self-configuring local area network using point-to-point links," *IEEE Journal on Selected Areas in Communicartions*, vol. 9, no. 8, pp. 1318–1335, October 1991.

[55] R. Casado, A. Bermúdez, , J. Duato, F. Quiles, and J. Sánchez, "A protocol for deadlock-free dynamic reconfiguration in high-speed local area networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 2, pp. 115–132, February 2001.

[56] O. Lysne and J. Duato, "Fast dynamic reconfiguration in irregular networks," in *Proceedings of the 2000 International Conference of Parallel Processing (ICPP)*. IEEE Computer Society, 2000, pp. 449–458.

[57] T. Pinkston, R. Pang, and J. Duato, "Deadlock-free dynamic reconfiguration schemes for increased network dependability," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 8, pp. 780–794, 2003.

[58] A. Ghiribaldi, A. Strano, M. Favalli, D. Bertozzi. "Power Efficiency of Switch Architecture Extensions for Fault Tolerant NoC Design," *Third International Green Computing Conference (IGCC'12)*. San Jose, (USA), 2012, pp. 1 – 6.

[59] J. Duato, O. Lysne, R. Pang, and T. Pinkston, "Part I: A theory for deadlock-free dynamic network reconfiguration," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 5, pp. 412–427, May 2005.

[60] O. Lysne, T. Pinkston, and J. Duato, "Part II: A methodology for developing deadlock-free dynamic network reconfiguration processes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 5, pp. 428–443, May 2005.

[61] D. Avresky and N. Natchev, "Dynamic reconfiguration in computer clusters with irregular topologies in the presence of multiple node and link failures," *IEEE Transactions Computers*, vol. 54, no. 5, pp. 603–615, May 2005.

[62] J. Acosta and D. Avresky, "Intelligent dynamic network reconfiguration," in *Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society, 2007, pp. 1–9.

[63] C. Feng, Z. Lu, A. Jantsch, J. Li, and M. Zhang, "A reconfigurable fault-tolerant deflection routing algorithm based on reinforcement learning for Network-on-Chip," in *Proceedings of the International Workshop on Network on Chip Architectures (NocArc)*, 2010.

[64] Z. Zhang, A. Greiner, and S. Taktak, "A reconfigurable routing algorithm for a fault-tolerant 2D-mesh Network-on-Chip," in *Proceedings of the 46th Design Automation Conference (DAC)*. ACM, June 2008, pp. 441–446.

[65] V. Puente, J. Gregorio, F. Vallejo, and R. Beivide, "Immunet: A cheap and robust fault-tolerant packet routing mechanism," in *Proceedings of the 31th Annual International Symposium on Computer Architecture (ISCA)*, June 2004, pp. 198–209.

[66] A. Mejia, J. Flich, J. Duato, S.-A. Reinemo, and T. Skeie, "Segment-based routing: An efficient fault-tolerant routing algorithm for meshes

and tori," in *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS)*, 2006, pp. 1–10.

[67] K. Aisopos, A. DeOrio, L.-S. Peh, and V. Bertacco, "Ariadne: Agnostic reconfiguration in a disconnected network environment," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2011.

[68] F. Gurkaynak, S. Oetiker, H. Kaeslin, N. Felber, W. Fichtner, "GALS at ETH Zurich: Success or Failure?", Proceedings of Asynch'06, pp. 150–159, 2006.

[69] S. Borkar, "Thousand Core Chips: a Technology Perspective", Proceedings of the Design Automation Conference (DAC), pp. 746–749. 2007.

[70] C. J. Myers et al., "Asynchronous Circuit Design", Wiley, 2001.

[71] J. Sparso, S. Furber, "Principles of Asynchronous Circuit Design: A System Perspective", Kluwer, 2001.

[72] J. M. Rabaey, "Digital Integrated Circuits: a Design Perspective", Prentice-Hall, 2003.

[73] S. Herbert, D. Marculescu, "Analysis of Dynarmic Voltage/Frequency Scaling in Chip Multiprocessors", Proceedings of the International Symposium on Low Power Electronics and Design, pp. 38–43. 2007.

[74] A. P. Chandrakasan et al, "Low Power CMOS Digital Design", IEEE Journal of Solid State Circuits, Vol. 27, pp. 437–484, 2007.

[75] ARM Ltd., "AMBA AHB Overview", `http://www.arm.com/products/solutions/AMBA_Spec.html`, 2005.

[76] ARM Ltd., "AMBA 3 AXI Overview", `http://www.arm.com/products/solutions/AMBA3AXI.html`, 2005.

[77] A. J. Martin, M. Nystrom, "Asynchronous Techniques for System-on-Chip Design", Proceedings of the IEEE, vol.94, no.6, pp. 1089–1120, 2006.

[78] S. Saponara, F. Vitullo, R. Locatelli, P. Teninge, M. Coppola, L. Fanucci, "LIME: a Low-Latency and Low-Complexity On-Chip Mesochronous Link with Integrated Flow Control", Proceedings of Euromicro Conference on Digital System Design (DSD), pp. 32–35, 2008.

[79] D. Mangano, G. Falconeri, C. Pistritto, A. Scandurra, "Effective Full-Duplex Mesochronous Link Architecture for Network-on-Chip Data-Link Layer", Proceedings of Euromicro Conference on Digital System Design (DSD), pp. 519–526, 2007.

[80] F. Vitullo et al. "Low-Complexity Link Microarchitecture for Mesochronous Communication in Networks-on-Chip", IEEE Trans. on Computers, Vol.57, issue 9, pp. 1196–1201, 2008.

[81] D. Wiklund, "Mesochronous Clocking and Communication in On-Chip Networks", Proceedings of the Swedish System-on-Chip Conference, 2003.

[82] A.T.Tran, D.N.Truong, B.Baas, "A Reconfigurable Source-Synchronous On-Chip Network for GALS Many-Core Platforms", IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, Vol.29, no.6, 2010.

[83] F. Clermidy, R. Lemaire, X. Popon, D. Ktenas, Y. Thonnart, "An Open and Reconfigurable Platform for 4G Telecommunication: Concepts and Application", Proceedings of Euromicro Conference on Digital System Design (DSD), pp. 62–74, 2009.

[84] F. Clermidy, C. Bernard, R. Lemaire, J. Martin, I. Miro-Panades, Y. Thonnart, P. Vivet, N. Wehn, "A 477mW NoC-based Digital Baseband for MIMO 4G SDR", ISSCC'2010, pp. 278–279, 2010.

[85] Y. Thonnart, P. Vivet, F. Clermidy, "A Fully-Asynchronous Low-Power Framework for GALS NoC Integration", Proceedings of Design, Automation and Test in Europe (DATE'10), pp. 33–38, 2010.

[86] S. Borkar, "Design Perspectives on 22nm CMOS and Beyond", Proceedings of the Design Automation Conference (DAC), 2009.

[87] R. Dobkin, V. Vishnyakov, E. Friedman, R. Ginosar, "An Asynchronous Router for Multiple Service Levels Networks on Chip", Proceedings of ASYNC'05, pp. 44–53, 2005.

[88] S. Beer, R. Ginosar, M. Priel, R. R. Dobkin, A. Kolodny, "The Devolution of Synchronizers", Proceedings of ASYNC, pp. 94–103, 2010.

[89] T. Bjerregaard, J. Sparso, "A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip", Proceedings of Design, Automation and Test in Europe (DATE), pp. 1226–1231, 2005.

[90] B. R. Quinton, M. R. Greenstreet, S. J.E. Wilton, "Asynchronous IC Interconnect Network Design and Implementation Using a Standard ASIC", Proceedings of the International Conference of Computer Design (ICCD), pp. 267–274, 2005.

[91] K. Y. Yun, R. P. Donohue, "Pausible Clocking: a First Step Toward Heterogeneous Systems", Proceedings of the International Conference of Computer Design (ICCD), pp. 118–123, 1996.

[92] R. Mullins, S. Moore, "Demystifying Data-Driven and Pausible Clocking Schemes", Proceedings of the International Symposium on Asynchronous Circuits and Systems, pp. 175–185, 2007.

[93] Z. Yu, B. M. Baas, "Implementing Tile-Based Chip Multiprocessors with GALS Clocking Styles", Proceedings of the International Conference on Computer Design, pp. 174–179, 2006.

[94] B. Mesgarzadeh, C. Svensson, A. Alvandpour, "A New Mesochronous Clocking Scheme for Synchronization in SoC", ISCAS, pp.605–609, 2002.

[95] I. M. Panades, F. Clermidy, P. Vivet, A. Greiner, "Physical Implementation of the DSPIN Network-on-Chip in the FAUST Architecture", Proceedings of International Symposium on Networks-on-Chip (NOCS), pp. 139–148, 2008.

[96] F. Vitullo, N. E. L'Insalata, E. Petri, L. Fanucci, M. Casula, R. Locatelli, M. Coppola, "Low-Complexity Link Microarchitecture for Mesochronous Communication in Networks-on-Chip", IEEE Trans. on Computers, Vol.57, no.9, pp. 1196–1201, 2008.

[97] S. Vangal et al.; "An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS", IEEE Journal of Solid-State Circuits, Vol.43, Issue 1, pp. 29–41, 2008.

[98] A. M. Scott, M. E. Schuelein, M. Roncken, J. Hwan, J. Bainbridge, J. R. Mawer, D. L. Jackson, A. Bardsley, "Asynchronous on-Chip Communication: Explorations on the Intel PXA27x Processor Peripheral Bus",

Proceedings of the 13th International Symposium on Asynchronous Circuits and Systems, pp. 60–72, 2007.

[99] M. B. Stensgaard, T. Bjerregaard, J. Sparso, J. H. Pedersen, "A Simple Clockless Network-on-Chip for a Commercial Audio DSP Chip", Proceedings of the 9th EUROMICRO Conference on Digital System Design, pp. 641–648, 2006.

[100] F. Mu, C. Svensson; "Self-Tested Self-Synchronization Circuit for Mesochronous Clocking", IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing, Vol.48, no.2, pp.129–141, 2001.

[101] A. Edmanand, C. Svensson, "Timing Closure through Globally Synchronous, Timing Portioned Design Methodology", Proceedings of Design Automation Conference (DAC), pp. 71–74, 2004.

[102] P. Caputa, C. Svensson, "An On-Chip Delay- and Skew-Insensitive Multicycle Communication Scheme", IEEE Solid-State Circuits Conference (ISSCC), pp. 1765–1774, 2006.

[103] SIA Semiconductor Industry Association "The International Technology Roadmap for Semiconductors", `http://public.itrs.net/`

[104] K.Arabi, "Logic BIST and Scan Test Techniques for Multiple Identical Blocks", IEEE VLSI Test Symnposium, pp.60-68, 2002.

[105] Wu, Y. and MacDonald, P., "Testing ASICs with Multiple Identical Cores", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 22-3, 2003, pp. 327-336.

[106] C.Aktouf, "A Complete Strategy for Testing an on-Chip Multiprocessor Architecture", IEEE Design and Test of Computers, vol.19-1, pp.18-28, 2002.

[107] C.Grecu, P.Pande, B.Wang, A.Ivanov, R.Saleh, "Methodologies and Algorithms for Testing Switch-Based NoC Interconnects", IEEE DFT 2005, pp.238-246, 2005.

[108] M.Hosseinabady, A.Banaiyan, M.N.Bojnordi, Z.Navabi, "A Concurrent Testing Method for NoC Switches", DATE, pp.1171 - 1176, 2006

[109] I. Loi, F. Angiolini, L. Benini, "Developing Mesochronous Synchronizers to Enable 3D NoCs", Proceedings of International Conference on VLSI Design, 2007.

[110] S. Stergiou, F. Angiolini, S. Carta, L. Raffo, D. Bertozzi, G. De Micheli, "xpipesLite: a Synthesis Oriented Design Library for Networks on Chips", Proceedings of the Design Automation and Test in Europe Conference (DATE), pp. 1188–1193, 2005.

[111] F. Angiolini, L. Benini, P. Meloni, L. Raffo, S. Carta, "Contrasting a NoC and a Traditional Interconnect Fabric with Layout Awareness", Proceedings of Design, Automation and Test in Europe (DATE), 2006.

[112] Y. Semiat, R. Ginosar, "Timing Measurements of Synchronization Circuits", Proceedings of the International Symposium on Advanced Research in Asynch. Circuits and Systems, pp. 68–77, 2003.

[113] D. Kim, K. Kim, J. Y. Kim, S. Lee, H. J. Yoo, "Solutions for Real Chip Implementation Issues of NoC and Their Application to Memory-Centric NoC", Proceedings of the International Symposium on Networks-on-Chips (NOCS), 2007.

[114] M. Ghoneima, Y. Ismail, M. Khellah, V. De, "Variation-Tolerant and Low-Power Source-Synchronous Multi-Cycle On-Chip Interconnection Scheme", VLSI Design, 2007.

[115] Zhiyi Yu, Bevan M. Baas, "High Performance, Energy Efficiency, and Scalability with GALS Chip Multiprocessors", IEEE Trans. VLSI, vol.17, no.1, pp. 66–79, 2009.

[116] G. Campobello, M. Castano, C. Ciofi, D. Mangano, "GALS Networks on Chip: a new solution for asynchronous delay-insensitive links", Proceedings of Design, Automation and Test in Europe (DATE), pp. 160–165, 2006.

[117] S. Kim, R. Sridhar, "Self-Timed Mesochronous Interconnections for High-Speed VLSI Systems", Proceedings of GLSVLSI, pp. 122–128, 1996.

[118] M. R. Greenstreet, "Implementing a STARI chip", Proceedings of ICCD, pp.3, 1995.

[119] E. Beigne, F. Clermidy, P. Vivet, A. Clouard, M. Renaudin, "An Asynchronous NOC Architecture Providing Low Latency Service and Its Multi-Level Design Framework", Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems, pp. 54–63, 2005.

[120] J.Ebergen, "Squaring the FIFO in GasP", Proceedings of International Symposium on Asynchronous Circuits and Systems, pp.194–205, 2001.

[121] C.E.Molnar, I.W.Jones, W.S.Coates, J.K.Lexau, "A FIFO ring performance experiment", Proceedings of International Symposium on Asynchronous Circuits and Systems, pp.279–289, 1997.

[122] R.Apperson, Z.Yu, M.Meeuwsen, T.Mohsenin, B.Baas, "A scalable dual-clock FIFO for data transfers between arbitrary and haltable clock domains", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 15(10), pp.1125–1134, 2007.

[123] T.Ono, M.Greenstreet, "A Modular Synchronizing FIFO for NOCs", Proceedings of International Symposium on Networks-on-Chip (NOCS), 2009

[124] T.Chelcea, S.M.Nowick, "Robust Interfaces for Mixed-Timing Systems", IEEE Transactions on Very Large Scale Integration Systems, 12(8): 857-873, 2004.

[125] E. Beigne, P. Vivet, "Design of on-chip and off-chip interfaces for a GALS NoC architecture", Proceedings of the 12th IEEE International Symposium on Asynchronous Circuits and Systems, pp. 172, 2006.

[126] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, S. Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor", IEEE Micro, Volume 27:5, 2007.

[127] B. Stackhous et al., "A 65nm 2-Billion Transistor Quad-Core Itanium Processor", IEEE Journal of Solid State Circuits, Volume 44, pp. 18–31, 2009.

[128] W. J. Dally, J. W. Poulton, "Digital Systems Engineering", Cambridge University Press, 1998

[129] A. Tran, D. Truong, B. Baas, "A GALS Many-Core Heterogeneous DSP Platform with source-Synchronous On-Chip Interconnection Network", Proceedings of the International Symposium on Networks-on-Chip, 2009.

[130] J. Bainbridge, "CHAINWorks", Silistix, http://www.silistix.com

[131] D. M. Chapiro, "Globally-Asynchronous Locally-Synchronous Systems". PhD Dissertation, Stanford University, October 1984.

[132] R. Ginosar, "Fourteen Ways to Fool Your Synchronizer", Proceedings of International Symposium on Asynchronous Circuits and Systems, pp. 89–97, 2003.

[133] Xuan-Tu Tran, J. Durupt, F. Bertrand, V. Beroulle, C. Robach, "A DFT Architecture for Asynchronous Networks-on-Chip", Proceedings of the IEEE European Test Symposium, pp. 219–224, 2006.

[134] Xuan-Tu Tran, Y.Thonnart, J.Durupt, V.Beroulle, C.Robach, "A Design-for-Test Implementation of an Asynchronous Network-on-Chip Architecture and its Associated Test Pattern Generation and Application", Proceedings of the International Symposium on Networks-on-Chip, pp. 149–158, 2008.

[135] Circuits Multi-Projects, Multi-Project Circuits; http://cmp.imag.fr

[136] C.Cummings, P.Alfke, "Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparison", SNUG-2002, San Josè, CA, 2002.

[137] I.M.Panades, A.Greiner, "Bi-Synchronous FIFO for Synchronous Circuit Communication Well Suited for Network-on-Chip in GALS Architectures", Proceedings of International Symposium on Networks-on-Chip (NOCS), pp.83–94, 2007.

[138] M. Krstic, et al., Globally Asynchronous, Locally Synchronous Circuits: Overview and Outlook, IEEE Design & Test of Computers, pp: 430 - 441 , Volume: 24 Issue: 5, Sept.-Oct. 2007

[139] M. Krstic, et al., OFDM Datapath Baseband Processor for 1 Gbps Datarate, Proc. IFIP/IEEE VLSI-SoC Conference 2008, pp. 156-159.

[140] Lines, A., Asynchronous interconnect for synchronous SoC design, IEEE Micro, 2004, 24, (1), pp. 3241.

[141] L. Plana, et al., A GALS Infrastructure for a Massively Parallel Multiprocessor, IEEE Design & Test of Computers, Sept.-Oct. 2007, Volume: 24 Issue:5, pp.: 454 - 463

[142] E. Beigne, et al, An asynchronous power aware and adaptive NoC based circuit, 2008 IEEE Symposium on VLSI Circuits, pp. 190 191.

[143] R. Lemaire, Y. Thonnart, Magali, A Reconfigurable Digital Baseband for 4G Telecom Applications based on an Asynchronous NoC, In Proc ACM/IEEE International Symposium on Networks-on-Chip, 2010.

[144] P.Vivet, et al, FAUST, an Asynchronous Network-on-Chip based Architecture for Telecom Applications, DATE, 2006.

[145] http://techresearch.intel.com/articles/Tera-Scale/1421.htm

[146] http://techresearch.intel.com/articles/Tera-Scale/1449.htm

[147] S. Borkar, Microarchitecture and design challenges for gigascale integration. Proc. ACM/IEEE *MICRO*, keynote address, pp. 3-3, 2004.

[148] S. Borkar, Designing reliable systems from unreliable components: the challenges of transistor variability and degradation, IEEE Micro, Vol. 25, No. 6, pp. 10-16, 2005.

[149] J. W. McPherson, Reliability challenges for 45nm and beyond, Proc. ACM/IEEE *DAC*, pp. 176-181, 2006.

[150] T. Bjerregaard, Shankar Mahadevan, A survey of research and practices of network-on-chip, ACM Computer Survey, Vol. 38, No. 1, 2006.

[151] S.Rodrigo, S.Medardoni, J.Flich, D.Bertozzi, J.Duato, "Efficient Implementation of Distributed Routing Algorithms for NoCs", IET-CDT, pp.460-475, vol.3, issue 5, 2009.

[152] S.Rodrigo, J.Flich, A.Roca, S.Medardoni, D.Bertozzi, J.Camacho, F.Silla, J.Duato, "Addressing Manufacturing Challenges with Cost-Effective Fault Tolerant Routing", NoCs 2010, pp.35-32, 2010.

[153] P.K. Lala, "Self-checking and fault tolerant Digital Design", MK Publishers 2001.

[154] Zhang, Z., Greiner, A., Benabdenbi, M.: Fully Distributed Initialization Procedure for a 2D-Mesh NoC, Including Off-Line BIST and Partial Deactivation of Faulty Components The 16th IEEE International On-Line Testing Symposium (IOLTS), 2010, pp.194-196.

[155] Strano, A., Ludovici, D., Bertozzi, D.: A Library of Dual-Clock FIFOs for Cost-Effective and Flexible MPSoCs Design, Proceedings of the International Conference on Embedded Computer Systems: Architectures, MOdeling and Simulation (SAMOS), 2010, pp.20-27.

[156] A. Ghiribaldi, D. Ludivici, F. Triviño, A. Strano, J. Flich, J. Sánchez, F. Alfaro, M. Favalli, and D. Bertozzi, "A complete self-testing and self-configuring noc infrastructure for cost-effective mpsocs," *ACM Transactions on Embedded Computing Systems*, 2011.

[157] A. Mejia, J. Flich, and J. Duato, "On the potentials of segment-based routing for nocs," in *Parallel Processing, 2008. ICPP '08. 37th International Conference on*, sept. 2008, pp. 594 –603

[158] F. Gilabert, M. E. Gómez, S. Medardoni, and D. Bertozzi, "Improved utilization of noc channel bandwidth by switch replication for cost-effective multi-processor systems-on-chip," in *Fourth ACM/IEEE International Symposium on Networks-on-Chip*, 2010, pp. 165–172.

List of Publications

Book Chapters

1. D. Bertozzi, **A. Strano**, D. Ludovici, V. Pavlidis, F. Angiolini, M. Krstic, **The Synchronization Challenge**, Chapter 6 in *"Designing Network-on-Chip Architectures in the Nanoscale Era"*, pp. 177-235, December 2010, CRC Book, ISBN: 978-1-4398-3710-8.

2. D. Bertozzi, **A. Strano**, F. Gilabert, D. Ludovici, **Technology-Aware Communication Architecture Design for Parallel Hardware Platforms**, Chapter in *"Advanced Circuits for Emerging Technology"*, CRC Book, 2011, *in press*.

International Journals

1. F. O. S. Jacobsen, S. Rodrigo, **A. Strano**, T. Skeie, D. Bertozzi, F. Gilabert. **Enabling Power Efficiency through Dynamic Rerouting on-Chip**. *ACM Transaction on Embedded Computing Systems (TECS), in press*.

2. M. Krstic, X. Fan, E. Grass, L. Benini, M. R. Kakoee, C. Heer, B. Sanders, **A. Strano**, D. Bertozzi. **Evaluation of GALS Methods in scaled CMOS Technology - Moonrake Chip Experience**. textitSpecial issue of the International Journal of Embedded and Real-Time Communication Systems (IJERTCS), 2012.

3. F. O. S. Jacobsen, S. Rodrigo, T. Skeie, **A. Strano**, D. Bertozzi. An Efficient, **Low-Cost Routing Framework for Convex Mesh Partitions to Support Virtualisation**. *ACM Transaction on Embedded Computing Systems (TECS), in press*.

4. **A. Strano**, N. Caselli, S. Terenzi, D. Bertozzi. **Optimizing Pseudo-Random Built-In Self-Testing of Fully Synchronous as well as Multisynchronous Networks-on-Chip**. *Special issue of the International Journal of IET Computers & Digital Techniques, 2012. in press*.

5. A. Ghiribaldi, D. Ludovici, F. Triviño, **A.Strano**, J. Flich, J. L. Sanchez, F. Alfaro, M. Favalli, D. Bertozzi, **A Complete Self-Testing and Self-Configuring NoC Infrastructure for Cost-Effective MPSoCs**, *ACM Transaction on Embedded Computing Systems (TECS), in press*.

6. **A. Strano**, D. Ludovici, D. Bertozzi, **A Library of GALS Interfaces for Cost-Effective and Flexible MPSoC Design**, *Transaction on HiPEAC*, *in press*.

7. **A. Strano**, C. Hernández, F. Silla, D. Bertozzi. **Self-Calibrating Source Synchronous Communication for Delay Variation Tolerant GALS Network-on-Chip Design**. *Special issue of the International Journal of Embedded and Real-Time Communication Systems (IJERTCS), 2011*.

International Conferences (with proceedings)

1. **A. Strano**, F. Triviño, J. L. Sánchez, F. J. Alfaro, D. Bertozzi, J. Flich. **OSR-Lite: Fast and Deadlock-Free NoC Reconfiguration Framework**. *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XII)*. Samos (Greece), 2012, pp. 86 – 95. **BEST PAPER AWARD**.

2. N. Caselli, **A. Strano**, D. Ludovici, D. Bertozzi. **Cooperative Built-in Self-Testing and Self-Diagnosis of NoC Bisynchronous Channels**. *IEEE 6th International Symposium on Embedded Multicore SoCs (MC-SOC12)*. Aizu (Japan), 2012, pp. 159 – 166. **BEST PAPER AWARD**.

3. H. Tatenguem, **A. Strano**, V. Govind, J. Raik, D. Bertozzi. **Ultra-Low Latency NoC testing via Pseudo-Random Test Pattern Compaction**. *International Symposium on System on Chip (SoC), 2012*. Tampere (Finland), pp. 1 – 6.

4. A. Ghiribaldi, **A. Strano**, M. Favalli, D. Bertozzi. **Power Efficiency of Switch Architecture Extensions for Fault Tolerant NoC Design**. *Third International Green Computing Conference (IGCC'12)*. San Jose, (USA), 2012, pp. 1 – 6.

5. **A. Strano**, D. Bertozzi, F. Angiolini, L. Di Gregorio, F. O. Sem-Jacobsen, V. Todorov, J. Flich, F. Silla, T. Bjerregaard. **Quest for the ultimate Network-on-Chip: the NaNoC project**. *Interconnection Network Architecture: On-Chip, Multi-Chip (INA-OCMC)*. Paris (France), 2012, pp. 43 – 46.

6. S. Terenzi, **A. Strano**, D. Bertozzi. **Optimizing Built In Pseudo-Random Self-Testing for Network-on-Chip Switches**. *Interconnection Network Architecture: On-Chip, Multi-Chip (INA-OCMC)*. Paris (France), 2012, pp. 21 – 24.

7. H.F. Tatenguem, D. Ludovici, **A. Strano**, H. Reinig, D. Bertozzi. **Contrasting Multi-Synchronous MPSoC Design Styles for Fine-Grained Clock Domain Partitioning: the Full-HD Video Playback Case Study**. *4th International Workshop on Network on Chip Architectures (NoCArc'11)*. Porto Alegre, 2011, pp. 37 – 42.

8. **A. Strano**, C. G. Requena, D. Ludovici, M. E. Gómez, M. Favalli, D. Bertozzi, **Exploiting Network-on-Chip Structural Redundancy for A Cooperative and Scalable Built-In Self-Test Architecture**, *Proceedings of Design, Automation and Test in Europe 2011 (DATE)*, pp. 661–666, Grenoble, France, 2011.

9. **A. Strano**, D. Bertozzi, A. Grasset, S. Yehia. **Exploiting structural redundancy of SIMD accelerators for their built-in self-testing/diagnosis and reconfiguration**. *IEEE International Conference on Application-specific Systems, Architectures and Processors 2011 (ASAP)*, Santa Monica. pp. 141–148.

10. M. Krstic, X. Fan, E. Grass, L. Benini, M. R. Kakoee, C. Heer, B. Sanders, **A. Strano**, D. Bertozzi. **Moonrake Chip - GALS Demonstrator in 40 nm CMOS Technology**. *International Symposium on System on Chip (SoC), 2011*. Tampere (Finland), pp. 9 – 13.

11. D. Ludovici, **A. Strano**, G. Gaydadjiev, D. Bertozzi. **Mesochronous NoC Technology for Power-Efficient GALS MPSoC**, *Proceedings of the Fifth ACM Interconnection Network Architecture, On-Chip Multi-Chip Workshop (INA-OCMC)*, pp. 27–30, Heraklion, Greece, 2011.

12. **A. Strano**, C. Hernández, F. Silla, D. Bertozzi. **Process Variation and Layout Mismatch Tolerant Design of Source Synchronous Links for GALS Networks-on-Chip**. *International Symposium on System on Chip (SoC), 2010*. Tampere (Finland), pp. 43 – 48.

13. **A. Strano**, D. Ludovici, D. Bertozzi, **A Library of Dual-Clock FIFOs for Cost-Effective and Flexible MPSoCs Design**, *Proceedings of the International Conference on Embedded Computer Systems: Architectures, MOdeling and Simulation (SAMOS)*, pp. 20–27, Samos, Greece, 2010.

14. D. Ludovici, **A. Strano**, G. N. Gaydadjiev, L. Benini, D. Bertozzi, **Design Space Exploration of a Mesochronous Link for Cost-Effective and Flexible GALS NOCs**, *Proceedings of Design, Automation and Test in Europe 2010 (DATE)*, pp. 679–684, Dresden, Germany, 2010.

15. D. Ludovici, **A. Strano**, D. Bertozzi, **Architecture Design Principles for the Integration of Synchronization Interfaces into Network-on-Chip Switches**, *Proceedings of the 2nd ACM/IEEE International Workshop on Network-on-Chip Architectures (NoCArc)*, pp. 31–36, New York, USA, 2009.

16. D. Ludovici, **A. Strano**, D. Bertozzi, L. Benini, G. N. Gaydadjiev, **Comparing Tightly and Loosely Coupled Mesochronous Synchronizers in a NoC Switch Architecture**, *Proceedings of the 3rd ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, pp. 244–249, San Diego, USA, 2009.