LABORATOIRE INTERFACES CAPTEURS
ET MICROÉLECTRONIQUE

PROFESSÜR FÜR TECHNISCHE INFORMATIK

ÉCOLE DOCTORALE IAEM - LORRAINE
DÉPARTEMENT DE FORMATION DOCTORALE ÉLECTRONIQUE - ÉLECTROTECHNIQUE

Thèse en Co-Tutèlle

Présenté à l'Université P. Verlaine de Metz pour l'obtention du diplôme de
Docteur des l'Universités de Metz et Potsdam
Discipline : Systèmes électroniques

RECONFIGURABLE SELF-ORGANISED SYSTEMS : ARCHITECTURE AND IMPLEMENTATION

par

Kevin Cheng

Soutenue le 12 octobre 2011 devant le jury composé de :

A. Dandache	Pr., UPVM (FR)	Directeur de thèse et Examinateur
C. Bobda	Pr., Universität Potsdam (DE)	Co-directeur de thèse et Examinateur
C. Tanougast	MCF-HDR, UPVM (FR)	Superviseur et Examinateur
A. Bouridane	Pr., Northumbria University (UK)	Rapporteur
F. G. Moraes	Pr., PUCRS (BR)	Rapporteur
E. Bourennane	Pr., Université de Dijon (FR)	Examinateur
B. Granado	Pr., ENSEA (FR)	Examinateur

LICM - ISEA - UPVM - 7 rue Marconi, 57070 Metz Technopole, France
Tél. : +33387547428 - Fax : +33387547307 - www.licm.sciences.univ-metz.fr

Résumé

Afin de répondre à une complexité croissante des systèmes de calcul, de nouveaux paradigmes architecturaux basés sur des structures auto-adaptatives et auto-organisées sont à élaborer. Ces derniers doivent permettre la mise à disposition d'une puissance de calcul suffisante tout en bénéficiant d'une grande flexibilité et d'une grande adaptabilité, cela dans le but de répondre aux évolutions des traitements distribués caractérisant le contexte évolutif du fonctionnement des systèmes. Ces travaux de thèse proposent une nouvelle approche de conception des systèmes communicants, auto-organisés et auto-adaptatifs basés sur des nœuds de calcul reconfigurable. Autrement dit, ces travaux proposent un système matériel autonome et intelligent, capable de déployer et de redéployer ses modules de calcul, en temps réel et en fonction de la demande de traitement et de la puissance de calcul. L'aboutissement de ces travaux se traduit par la réalisation d'un *Système Auto-organisé Reconfigurable* (SAR) basé sur la technologie FPGA. L'architecture auto-adaptative proposée permet d'étudier l'impact des systèmes reconfigurables dans une structure distribuée et auto-organisée. Le système est réalisé pour étudier, à chaque niveau, les paramètres qui influencent les performances globales d'un réseau de calcul évolutif. L'étude de l'état de l'art a permis la mise en perspective et la formalisation des caractéristiques du concept d'auto-organisation matérielle proposé ainsi qu'une évaluation et une analyse de ces performances. Les résultats de ces travaux montrent la faisabilité d'un système complexe de calcul distribué dont l'intelligence repose sur les interactions des éléments reconfigurables le constituant.

Zusammenfassung

Bedürfnisse nach der Berechnungskraft vergrößernd, machen Biegsamkeit und Zwischenfunktionsfähigkeit Systeme immer schwieriger, zu integrieren und zu kontrollieren. Die Hochnummer von möglichen Konfigurationen, alternative Designentscheidungen oder die Integration von zusätzlichen Funktionalitäten in einem Arbeitssystem kann nicht nur an design-malig nicht mehr getan werden. In diesem Kontext, wo die Entwicklung von vernetzten Systemen ist, werden äußerst schnelle, verschiedene Begriffe mit dem Ziel studiert, mehr Autonomie und mehr rechnende Kraft zu versorgen. Diese Arbeit schlägt eine neue Annäherung für die Nutzbarmachung der reconfigurable Hardware in einem selbstorganisierten Kontext vor. Ein Begriff und ein Arbeitssystem werden unter der Form von Reconfigurable Selbstorganisierte Systeme (RSS) präsentiert. Die vorgeschlagene Hardware-Architektur hat vor, den Einfluß von reconfigurable FPGA gegründete Systeme in einer selbstorganisierten vernetzten Umwelt zu studieren, und partielle Wiederkonfiguration wird verwendet, um Hardware-Gaspedale an runtime durchzuführen. Das vorgeschlagene System wird dafür bestimmt, an jedem Niveau, Parameter dass Einfluß auf Leistungen der vernetzten selbstanpassungsfähigen Knoten zu beobachten. Die Resultate präsentiert zielen hier zu Zugang, wie Reconfigurable-Computerwissenschaft effizient verwendet werden kann, um einen Komplex zu entwerfen, erlaubten vernetztes Berechnungssystem und der Zustand der Kunst, zu erleuchten und Eigenschaften des vorgeschlagenen selbstorganisierten Hardware-Begriffes zu formalisieren. Seine Auswertung und die Analyse seiner Leistungen waren mögliches Verwenden ein kundenspezifisches Verwaltung: das Potsdam Vernünftige Kamera-System (PICSy). Es ist eine vollstandige Erfüllung vom elektronischen Verwaltung bis die Kontrollanwendung. Um die Arbeit zu ergänzen, erlauben Maße und Beobachtungen Analyse dieser Realisierung und tragen zur allgemeinen Kenntnis bei.

Abstract

Increasing needs of computation power, flexibility and interoperability are making systems more and more difficult to integrate and to control. The high number of possible configurations, alternative design decisions or the integration of additional functionalities in a working system cannot be done only at the design stage any more. In this context, where the evolution of networked systems is extremely fast, different concepts are studied with the objective to provide more autonomy and more computing power. This work proposes a new approach for the utilization of reconfigurable hardware in a self-organised context. A concept and a working system are presented as *Reconfigurable Self-Organised Systems* (RSS). The proposed hardware architecture aims to study the impact of reconfigurable FPGA based systems in a self-organised networked environment and partial reconfiguration is used to implement hardware accelerators at runtime. The proposed system is designed to observe, at each level, the parameters that impact on the performances of the networked self-adaptive nodes. The results presented here aim to assess how reconfigurable computing can be efficiently used to design a complex networked computing system and the state of the art allowed to enlighten and formalise characteristics of the proposed self-organised hardware concept. Its evaluation and the analysis of its performances were possible using a custom board: the *Potsdam Intelligent Camera System* (PICSy). It is a complete implementation from the electronic board to the control application. To complete the work, measurements and observations allow analysis of this realisation and contribute to the common knowledge.

Acknowledgement

I would like to express my gratitude to Professor Ahmed BOURIDANE and Professor Fernando Gehm MORAES from Northumbria University and Pontificia Universidade do Rio Grande do Sul respectively. It is my honour to have them as main reviewers of this thesis. I also would like thank Professor El-Bay BOURENNANE and Professor Bertrand Granado, respectively from the Université de Bourgogne and the École Nationale Supérieur de l'Électronique et de ses Applications to be examiners.

As research directors, I acknowledge Professor Abbas DANDACHE and Professor Christophe BOBDA. They gave me the opportunity to join their respective groups and moreover, the opportunity to meet a lot of great peoples who helped me to discover the world of scientific research.

To Doctor Camel TANOUGAST, I express special thanks for everything. As my direct supervisor, he shared with me his knowledge in the field of computer science and reconfigurable computing. Thank you for all great conversations about science and about life in general. I want to express my gratitude for giving me a part of his energy.

I also acknowledge Pr. Fabrice MONTEIRO, Dr. Camille DIOU and Mr. Philippe JEAN, for their advices during the years.

I would like to thank all my colleagues from Potsdam and Metz: Ali, Philipp, Felix, Clovis, Robert, Franck, Max, Cédric, Mohktar, Moshin, Maria, Wiebke and Francis.

I also would like to thank my friends Daniela and Ewelina who helped me to improve my English writing.

To conclude, I would like to thank my dad and mum, Davy CHENG and Patricia DI-SAÏA for their unfailing support over the years. To my brother, Stephane CHENG, who was at my side everyday. My sisters Séléna and Cassy DI-SAÏA, for their support. This could not have finished without my friends and special thanks go to

Carl, Stéphane, Pascal, Fragkiski, Thomas, Matthias, Andreas, Johanna, Sean, Maria, Diwertje, Anne, Giselle, Berit, Katharina, Mariam, Hélène, Nolwenn, Mathilde and Nadia.

I hear and I forget. I see and I remember. I do and I understand...
Confusius

To my parents Davy and Patricia...

Contents

Résumé ii

Zusammenfassung ii

Acknowledgement iii

Citation v

Dedicace vii

Contents ix

List of Figures xvii

List of Tables xix

List of Acronyms xxi

General Introduction: Context, Motivation and Contributions 1

1 Basic Concepts and Inter-Disciplinary Links 7

Introduction 8

1.1 Reconfigurable Computing and Systems 8

1.1.1 Reconfigurable Computing 9

1.1.2 Distributed Computing 9

1.1.3 Grid Computing 10

1.1.4 Cloud Computing 10

1.1.5 Complex Adaptive Computing 11

1.1.6 Natural Computing 12

1.1.7 Other Research Directions . . . 12
1.1.8 Computing Overview and Classification . . . 12
1.2 Self-organised Computing and Systems . . . 13
1.2.1 Self-Organization and Emergence . . . 13
1.2.2 Associated Properties . . . 14
1.2.3 The Three Properties: Autonomy, Dynamic and Interoperability . . . 15
1.2.4 Local and Regional Behaviour . . . 17
1.3 Networking and Communication . . . 17
1.3.1 Heterogeneous Environment . . . 18
1.3.2 Protocol Diversity . . . 18
1.3.3 Decentralised Systems . . . 19
1.3.4 Peer-to-Peer Application and Concept . . . 20
1.4 Electronic Technologies: Configuration Point of View . . . 20
1.4.1 General Purpose Processor Design . . . 21
1.4.2 Application-Specific Integrated Circuit Design . . . 22
1.4.3 Field Programmable Gate Array . . . 22
1.4.4 Reconfigurable System Versus Adaptive System . . . 26
1.4.5 Embedded Electronic Design . . . 26
1.4.6 Possible Combinations . . . 26
Conclusion . . . 26

2 Review of Recent Related Works 29
Introduction . . . 30
2.1 Classical and Historical References . . . 30
2.1.1 Classical references . . . 30
2.1.2 Point-to-point Connection and Communication . . . 30
2.1.3 Bus Interconnection . . . 31
2.1.4 Operating System and Reconfigurable Architecture Co-design . . . 32
2.2 High level Multi-Board Systems . . . 32
2.2.1 RecoNodes/ReCoNet Systems . . . 32
2.2.2 Scalable Self-Configurable Architecture for Reusable Space Systems (SCARS) . . . 33
2.2.3 iBoard System . . . 34
2.2.4 Multimedia Oriented Reconfigurable Array . . . 35
2.3 Low Level Network-On-Chip Structure . . . 36

2.3.1 OverSoC . . . 36
2.3.2 ReCoBus Project . . . 37
2.3.3 DyNoC . . . 38
2.3.4 C-U-NoC/Q-NoC . . . 39
2.4 Project Observations . . . 40
Conclusion . . . 41

3 The Reconfigurable Self-Organized System Concept: a Networked System 43
Introduction . . . 44
3.1 Concept Development . . . 46
3.1.1 The Local Market Global Symbiosis Concept . . . 46
3.1.2 The Local Intelligence Global Symbiosis Concept . . . 50
3.2 Reconfigurable Self-Organised System: A P2P Reconfigurable Technology . . . 51
3.2.1 Definition . . . 51
3.2.2 Environment and Application Field . . . 52
3.2.3 Computing Model . . . 52
3.2.4 Simplified modelling and formalisation of a RSS . . . 53
3.3 Using Reconfigurable Hardware in a Self-Organization context . . . 55
3.3.1 Task distribution Stream Concept . . . 56
3.3.2 Theoretical Stream Structure . . . 59
3.3.3 Generation of bitstream and data request . . . 61
3.3.4 Social Organisation . . . 62
3.4 RSS Properties and Limitations . . . 63
3.4.1 Autonomy at Local Level . . . 64
3.4.2 Dynamic Organisation . . . 64
3.4.3 Interoperability with other technology and network . . . 65
3.4.4 Other properties . . . 66
3.4.5 Requirements for the implementation of an RSS . . . 67
3.4.6 Technological and Conceptual Limitations . . . 68
3.5 Compatibility with the Peer-to-Peer concept and the best RSS . . . 69
3.5.1 P2P Compatibility . . . 69
3.5.2 The Best Theoritical RSS . . . 70
Conclusion . . . 70

4 Realization and Implementation of the RSS Concept: The Node's Design 73
Introduction . . . 74
4.1 Scientific Remark on Intelligence . . . 74
4.1.1 Intelligence and Artificial Intelligence . . . 75
4.1.2 Intelligence Transfers from human to *RSN* . . . 76
4.2 The design of Reconfigurable Devices . . . 76
4.2.1 Main Xilinx Technology Aspects . . . 76
4.2.2 System on FPGA Design . . . 77
4.2.3 FPGA Selection . . . 78
4.2.4 Board Architecture . . . 80
4.2.5 Configuration Tool Option . . . 80
4.3 The Design of Scalable Systems . . . 81
4.3.1 Device Selection . . . 82
4.3.2 Organization of Communication . . . 82
4.3.3 Bitstream Management . . . 84
4.4 The design of the *Potsdam Intelligent Camera System* . . . 84
4.4.1 Board Design . . . 86
4.4.2 SoFPGA of PICSy . . . 88
4.4.3 Embedded Operating System and Driver . . . 90
4.4.4 Resource Management Software . . . 91
4.4.5 Networked System and Communication . . . 94
4.4.6 Hardware Module Exchange and P2P . . . 94
4.4.7 Drawbacks and Advantages . . . 95
Conclusion . . . 97

5 Analysis of the proposed RSS Architecture 99
Introduction . . . 100
5.1 Video System using RSS . . . 100
5.1.1 Person Detection Scenario . . . 100
5.1.2 Reconfigurable Streaming Data Interface Controller . . . 102
5.2 PR Measurement: Quantitative Analysis . . . 104
5.3 RSN's behavior . . . 105
5.3.1 Starting Up . . . 105
5.3.2 User Tasks . . . 106
5.4 Network Qualitative Analysis . . . 107

5.4.1 Timing Analysis 108
5.4.2 Analysis of the Computation Power 110
5.4.3 Analysis of the Evolution of RSS 110
5.4.4 Comparison with other systems 111

General Conclusion **113**

Bibliography **122**

Annexe **123**
Publications 123
Reconfigurable Natural Computing World 125

List of Figures

1.1 A Complex Networked System: A sensor network. 8
1.2 Grids and Clouds Overview [FZRL08] including reconfigurable technology . . 11
1.3 Computing Classification Proposition. 13
1.4 Three important properties of self-organized system considered for this work . 17
1.5 Three observations associated with technology and the self-organized concept . 18
1.6 The three important properties of self-organized systems. 20
1.7 The three main technologies and possible combinations and extensions. 21
1.8 Virtex Local Routing [Inc09] 23
1.9 Simple Configuration System. 24
1.10 Simple Configuration System 25
1.11 Work Background and main properties for reconfigurable and self-organized system. 27

2.1 Evolution of reconfigurable architecture [MMP+03] 31
2.2 Multi board project from the ReCoNet Project. [HKT03] 32
2.3 Local and networked self-healing solution of the SCARS system [SJAS08]. . . 33
2.4 The IBoard Version 2 [HA10] 34
2.5 The MORA Architecture [LPC07] 35
2.6 The OverSoC exploration and refinement flow [MHV+09]. 36
2.7 The ReCoBus-Builder design flow [KBT08]. 37
2.8 DyNoC modules implementation and routing [BAM+05]. 38
2.9 C-U-NoC [JTBW09]. 40

3.1 Control examples: Centralized and decentralized. 44
3.2 Issue a request (a), Retrieve answers (b) and Commission job (c). [BCM+09]. . 46
3.3 LMGS element versus LIGS Node 50
3.4 How to use the concept's terms. 51

3.5 RSS Computation Model. 53
3.6 Self-Organization Criterion of a Reconfigurable System 54
3.7 Architecture of the networked Self-reconfigurable nodes. 56
3.8 Task Request Stream and temporary Data or bitstream fluxes. 57
3.9 Control flow graph of one self-organized node. 59
3.10 Request frames. 60
3.11 Packet format of a data flux. 61
3.12 Node Functionality that constitute the basis of RSN. 62
3.13 Three interdependent properties in a double point of view. 63
3.14 The dependency stack of technology. 64
3.15 Dynamic cooperation evolution 65

4.1 Architecture of System on *FPGA* 78
4.2 A floorplan of the *XC5FXT FPGA* architecture 79
4.3 Scalable Networked System 82
4.4 *RSE*'s internal communication fluxes. 83
4.5 Example of Self-Organization Scenario. 84
4.6 Description of the relation between bitstream files and PR-Region. 85
4.7 *PICSy* Board design 86
4.8 The PICSy Camera: a reconfigurable system 87
4.9 The *PICSy* platform architecture 88
4.10 Caption of SoFPGA designed for PICSy 89
4.11 Placement example into a R-SoFPGA. 90
4.12 First remote PR control. 91
4.13 *RSN* functionalities. 92
4.14 The Program Architecture. 93
4.15 Broadcasted Requests. 94
4.16 Communication Protocol between *RSNs* 95
4.17 Example of networked *RSN* using the proposed flux to communicate 96

5.1 Foreground/Background computation using color distortion and brightness distortion. 101
5.2 An example of a segmented image 102
5.3 Illustration of the SDI controller usage. 103
5.4 Proposed modification of the SDI controller 104
5.5 The *RSN*'s *Start Up* activities. 105

5.6 shows this insertion of a local task by a user. 106
5.7 shows this insertion of a local task by a user. 107
5.8 Build PR Command and execute it. 107
5.9 Example of external task accepted and processed by an RSN. 108
5.10 Map of subjects connections 126
5.11 Map of subjects connections 127

List of Tables

3.1 Comparison of two types of control. 45

4.1 Summary of Positive and negative aspects 97

5.1 Performance measurements of the image segmentation on different computing systems. 102

5.2 Timing measurements for different sizes of reconfigurable regions. 105

List of Acronyms

Acronym	Definition
AI	Artificial Intelligence
ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuit
CCD	Charge Coupled Device
CLB	Configurable Logic Block
CMOS	Complementary Metal-Oxide Semiconductor
CMT	Clock Management Tile
CPU	Central Processing Unit
CUNoC	Communication Unit Network on Chip
DDR-SDRAM	Double Data Rata Synchronous Dynamic Random Access Memory
DRU	Dynamically Reconfigurable Unit
DSP	Digital Signal Processing
DyNoC	Dynamic Network on Chip
EDK	Embedded Development Kit
FPGA	Field Programmable Gate Array
FSL	Fast simplex Link
GPIO	General Purpose Input Output
GPP	General Purpose Processor
GPU	Graphic Processor Unit
HDL	Hardware DEscription Language
HW	Hardware
ICAP	Internal Configuration Access Port
IO	Input Output
IP	Internet Protocol
IPS	Instruction per Second

Acronym	Definition
ISE	Integrated Software Environment
JTAG	Joint Test Action Group
LED	Light Emitting Diode
LIGS	Local Intelligence GLobal Symbiosis
LMGS	Local Market GLobal Symbiosis
LUT	Look Up Table
LVDS	Low Voltage Differential Signaling
MAC	Media Access Control
MORA	Multimedia Oriented Reconfigurable Array
MPSoC	Multi Processor System on Chip
NoC	Network on Chip
PoenCV	Open Source Computer Vision Library
OS	Operating System
OTG	On The Go
OverSoC	Over System on Chip
P2P	Peer to Peer
PICSy	Potsdam Intelligent Camera System
PCB	Printed Circuit Board
PCIE	Peripheral Component Interconnect Express
PPC	Power Personal Computer
PR	Partial Reconfiguration
PRM	Partial Reconfigurable Module
PROM	Programmable Read Only Memory
PRR	Partial Reconfigurable Region
QNoC	Q Network on Chip
RC	Reconfigurable Computing
ReCoBus	ReConfigurable Bus
ReCoNet	ReConfigurable Net
ReCoNode	ReConfigurable Node
RGB	Red Green Blue
RSE	Reconfigurable Self-Organised Entity
RSN	Reconfigurable Self-Organised Node
RSoFPGA	Reconfigurable System on Field Programmable Gate Array

Acronym	Definition
RSS	Reconfigurable Self-Organised System
SCARS	Scalable Self-organised Architecture for Reusable Space Systems
JTAG	Joint Test Action Group
SDIC	Streaming Data Interface Controller
SoC	System on Chip
SoFPGA	System on Field Programmable Gate Array
SPI	Serial Peripheral Interface
SR	Static Region
SRAM	Static Random Access Memory
SW	Software
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunication System
USB	Universal Data Bus
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
WiFi	Wireless Fidelity
XPS	Xilinx Platform Studio

General Introduction: Context, Motivation and Contributions

The Main Context

The possibilities offered by the technological entities are deeply transforming societies. People are linked over networks to communicate and exchange information. These networks are interconnected and form a worldwide structure that can be seen as an unique entity, composed of many different kinds of devices like supercomputers, personal computers or recently arrived netbooks, smartphones and tablets. All of them are able to process data in certain times and with more or less computation power. In other words, we are now facing huge network entities providing computation resources and data depending on time and space parameters.

The Starting Problematic and Limitations

Due to the scale and complexity of today's systems, it is important to give them the possibility to manage problems in an intelligent and autonomous way. To cope with all non-deterministic changes and events that dynamically occur in the system's environment, new *design paradigms* must be developed. In particular, networked approaches relating to system architecture can be a viable solutions for the high computation demand and self-organisation requirements. In the literature, we find many terms and definitions related to systems that exhibit this form of artificial life. They can be called as life-like computer systems. These systems are characterized by intelligence and autonomy and are able to evolve and behave by themselves.

The problematic that we study can be stated as follows: How to transfer a part of intelligence that characterises human being to a technological entity? How to give it an additional degree of freedom? How can this system be defined, studied, integrated and analysed? The purpose of this work is to find the answers to these questions.

Because the thematic of self organisation concepts is vast and relates not only to technology but also to domains like philosophy, sociology, biology and physics, the research presented in

this document is focussed on the computer science field and provides technological solutions in the form of microelectronic computer systems. By doing so, the research field is limited to more specific knowledge. Of course, this limitation is not exclusive and one or two notions from external fields are used to give an open-minded view of the problem. Computer science is still being a huge topic and this is why *Reconfigurable Computing* is selected as a specialised research field. It is a combination of logical and digital computations, informatics, with its art of problem solving and finally the knowledge about communication requirements and structures. More specifically, this field can be described using terms like scalability, self-adaptation and embedded system.

The Main Motivation

This research is important and must be carried out through different angles because it aims to contribute to the common knowledge of the engineering of complex and natural systems. The best implementation strategy is inspired by Nature because Nature is autonomous, dynamic and interoperable. It exhibits an large set of heterogeneous subsystems that are all combined in a self-organised manner. This complex entity exhibits emergent behaviours, characteristics and properties.

This work aims to study, integrate and analyse an engineering solution that is trying to show autonomy, dynamism and interoperability like a natural system. This solution can be described in terms of scalability, self-adaptation and embedded systems.

Concept and Solutions

The definition of this problematic leads to the proposition of a concept completed by a implementation solution. We demonstrate a method to design reconfigurable natural systems which are based on Reconfigurable Computing and its associated technology. The architecture used here is composed of the *Field Programmable Gate Array* (FPGA) technology that allows us to design *System on FPGA* (SoFPGA). It also includes common communication technologies like *Ethernet*, protocols like *TCP/IP* and structures like *Peer-to-Peer*. The main advantages are: P2P offers a decentralised and self-organised structure **(Node's Autonomy)**. Reconfigurable SoFPGA offers computation power with high flexibility **(Dynamic evolution)**. Self-Adaptive Networked Architecture offers common and massively used support **(Interoperability)**.

This proposed solution is to build an elementary reconfigurable node that can be the base of an intelligent networked entity (inspired by *Natural System*).

Results

Systems resulting from this way of thinking are networked designs and they are based on highly *integrated* elements that are called nodes in the rest of this document. Our node definition can be described by many other words in the computer sciences jargon. Literature can also refer to it as modules, elements, agents or individuals.

To prove that the proposed hardware/software co-design and the self-organised concept are realistic, we have realised such a specific node. This node can be described as reconfigurable, autonomous, dynamic, interoperable and intelligent. It relates a complete prototype including, FPGA multi-board design, operating system adaptation with specific drivers, software design, integration of electronic communication and file management.

Results also include performance measurements, an important analysis and a discussion about the design of the bio-inspired node.

Research always implies to open the discussion on existing knowledge and about what to do with the result of the work that have been done. Here and like in many other research approaches, it comes out that technological entities behave like *natural systems*. They are trying to evolve and reach a higher degree of intelligence and autonomy. The work presented here is a new answer that exhibit these properties. As demonstrated here, nodes are becoming more powerful, modular and scalable day after day. They are expanding into their environment which is also our environment. The provided results concern knowledge that characterise hardware details by enlighten links with low level elements and high level abstraction layer. We are presenting the direct links between reconfigurable logic resources, control applications and self-organisation paradigms.

Research Challenges and Advantages

During these years of work, we faced many challenges. The first one was to combine the knowledge of many different fields and to modify them to fit out our application. It was the starting point to set the problematic and the combined solution.

The second challenge set arises from the accessible technology. Today, systems are designed to be simple and static and unfortunately, many systems are designed with a programmed obsolescence. That is, a first step when considering their finite but dynamic environment and considering the last knowledge coming from this kind of research. It is now a challenge to evolve and go through dynamic design. We therefore have to develop autonomous, dynamic and interoperable systems with some technological limitation. It was the starting point to separate the goals from the material resources and it helps to refine the design to do.

Finally, the last set of challenges that is also the most important, is about methodology. Technological entities can be bio-inspired and it is why it is necessary to start by redefining the basic of what a node should be. This should be done according to reconfigurable technology peculiarity. It represents our involvement to the discussion and to the design of future embedded systems.

The limitations are the utilisation of logic resources and logic level, the establishment of elementary node model, the utilisation of inter-node communications and finally the most simple implementations. The method's goal is to provide the base of a reconfigurable natural system that can be describe as reconfigurable and self-organised hardware systems.

Cooperation Background

This document presents the research work of a thesis that was initiated by the *Collège Doctoral Franco Allemand* and entitle *Adaptive, Autonomous and Organic Reconfigurable Computing*, *Système de Calcul Reconfigurable Adaptable, Autonomique et Organique* in French, or *Adaptiv, Autonomische, Organische und Reconfigurable Rechensysteme* in German. This thesis started on February 2008 and within a collaboration between the *Computer Engineering department* from the *Institute für Informatik* (IFI) of *Universität Potsdam* (UP) in Germany and the *Laboratoire Interfaces Capteurs Microélectronique* (LICM) of the *Université Paul Verlaine de Metz* (UPVM) in France. The objective of this collaboration is to study new architectural paradigms for the development of self-organised systems based on reconfigurable technology.

Key researchers are Pr. Abbas Dandache from UPVM, as Ph.D. director, Pr. Christophe Bobda from UP, as Ph.D. co-director, and the associate professor (MCF-HDR) Camel Tanougast as research supervisor.

Chapter Organization

This thesis manuscript is structured in five chapters and organised as follows:

In chapter 1, we present the main set of properties defining the systems that we wish to develop including self-organisation and emergence, adaptivity, autonomy, sturdiness and flexibility, pre-emption and decentralised control that are all related to the observation of our scalable environment. Definitions of self-organised system and emergent systems are given. It is shown that FPGA dynamically reconfigurable technology can be useful to integrate some of those properties into the designed hardware systems and in addition, FPGA *Partial Reconfiguration* (PR) characteristics are discussed. Finally, the conclusion of this chapter highlights the relevance of using self-organisation technology in so called natural hardware systems.

In chapter 2, our literature review is given. Starting from historical and classical references, we continue with the state of the art, concerning reconfigurable technology, and in particulars some projects are presented with the will to expose networking solution and reconfigurable system on chip systems.

In chapter 3, we propose the concept of *Reconfigurable Self-organised System* (RSS) and its definition. This concept includes a transposition of the characteristics and properties like self-organisation that can be applied to FPGA based reconfigurable systems. This chapter mainly describes network aspects and we conclude a census of principal requirements for the conception of networked RSS. We show, on the first hand, the necessity to elaborate new architectural paradigms based on communication structure adapted to FPGA technology and on the other hand the necessity to develop learning mechanisms able to be used during run-time.

The chapter 4 discusses the selected architectural approach that allows to combine self-organisation and Reconfigurable Systems on FPGA. Firstly, a review on commonly distributed control paradigms used in the design of networked system is presented. Secondly, we propose our architectural approach characterised by a decentralised communication structure for networked reconfigurable nodes according to the RSS concept. This original concept allows us to exchange information and logic hardware modules between nodes that constitute a natural entity. These exchanges are accomplished without any centralised control and can also be described as an emerging intelligence that tries to respond to its environment. The concept is also demonstrated through a concrete example that is running on the *Potsdam Intelligent Camera System* (PICSy) platform which is dedicated to real-time image processing.

Chapter 5 presents results of the implementation of a reconfigurable node built to be the basis of a self-organised and networked architecture. It aims to enable decentralised processing over disseminated nodes that are providing either *General Purpose Processor* (GPP) time slot or logic electronic hardware (PR Region). It is made to validate the global architectural approach of the self-organised system and associated concepts, mechanisms and structures in an adapted network presented in the previous chapters. The objective of this chapter includes also an experimental validation of all conceptual aspects studied here to make systems able to manage themselves.

Finally, a conclusion sums up the work that have been carried out and it concludes important aspects and opens the discussion on the integration of RSS into our biological environment and on the new combination that we are proposing: *Reconfigurable Natural Computing* that is a combination between Reconfigurable Computing and Natural Computing.

Chapter 1

Basic Concepts and Inter-Disciplinary Links

Introduction

This chapter aims to present the research environment by pointing out three fields of work. The first one is *Reconfigurable Computing*, the second one is *Self-organized Computing* and the last one is *Networking and Communication*. Their particularities and characteristics are described to show their relations and their important links that constitute the basis of the concept presented in chapter 3. By the end of this chapter, readers will be able to follow the proposed work and will have a general view of the problems involved.

To present this point of view, the first section will describe the different computing paradigms and proposes a classification with the objective of situating Reconfigurable Computing into the context of all those ideas. Section two will provide information concerning Self-organized Computing and in particular its properties. Section three will show that communication is a major problem for the previously presented paradigms. Finally, section four will present some characteristics of Reconfigurable Technology. This last section will be focused on configuration aspects and show that the technology allows to combine different kind of computation architecture.

1.1 Reconfigurable Computing and Systems

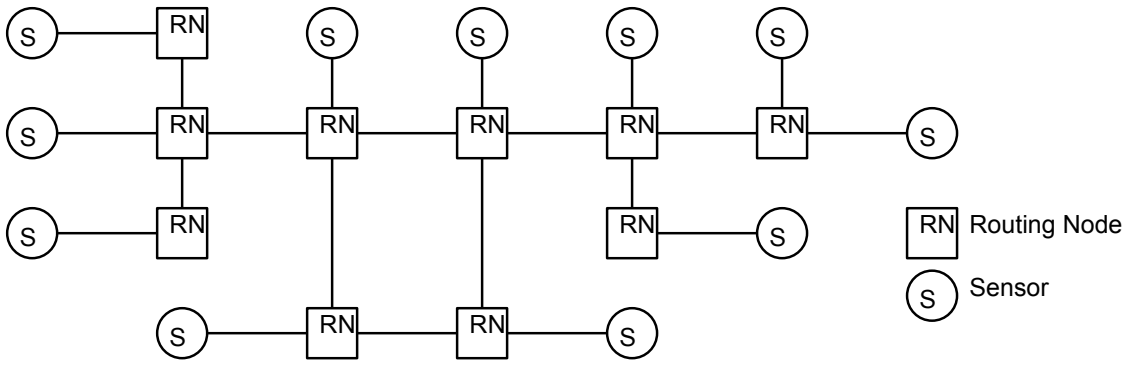


Figure 1.1: A Complex Networked System: A sensor network.

In this section, some research fields that study the system collaborations and interactions are set out. They are useful to characterize computing systems and in particular reconfigurable computing and systems. Different aspects are studied in order to find the best way to optimize the utilization of networked resources. The presented computing concepts are used to explain why interoperability is a major aspect of this work.

Figure 1.1 shows an example of a sensor network which is a complex networked system. Here, sensors must communicate with routing nodes to send data. Sensors and routing Nodes depend on each other. In spite of the fact that the subject is difficult to describe due to the diversity of definitions, some of them can be selected to fit into the topic of this work and to define its position.

1.1.1 Reconfigurable Computing

Reconfigurable computing is intended to fill the gap between hardware and software, achieving potentially much higher performance than software while maintaining a higher level of flexibility than hardware. Reconfigurable devices, including field-programmable gate arrays (FPGAs), contain an array of computational elements whose functionality is determined through multiple programmable configuration bits. These elements, sometimes known as logic blocks, are connected using a set of routing resources that are also programmable. In this way, custom digital circuits can be mapped to the reconfigurable hardware by computing the logic functions of the circuit within the logic blocks and using the configurable routing to connect the blocks together to form the necessary circuit. [CH02].

With this definition, a technological aspect is pointed out. Consequently, the evolution of the technology must be considered to redefine the concept if necessary. Technology is used to realize systems that are trying to make concepts real. As it is well known, the concept of reconfigurable computing was first described in [Est60] and it became possible to realize reconfigurable systems that really fit into this concept with the advent of FPGA technology. In the case of Reconfigurable Computing, the Reconfigurable Technology helps to rethink the original concept.

1.1.2 Distributed Computing

Distributed computing deals with large problems by decomposing a problem into small parts, send to nodes to solve them and then combining partial solutions to get a solution for the problem. In the scope of this work, distributed computing shows that it is possible to use all computers available world-wide. Besides this, it is a reminder that problems concerning undecidability and intractability are still governing computation problems [HMU01].

As an example of projects that apply the distributed computing concept, the folding@home and genome@home projects can be suggested. In [LSSP02] and related articles, distributed computing is used to compute an enormous data quantity with the help of a huge number of

personal computers. It means that simulated time access to processor resources can be shared for complex computing. Another example is the Condor Project which introduced the notion of flexibility. As distributed systems scale to ever larger sizes, they become more and more difficult to control or even to describe. International distributed systems are heterogeneous in every way: they are composed of many types and brand of hardware; they run various operating systems and applications; they are connected by unreliable networks; they change configuration constantly as old components become obsolete and new components are powered on. Most importantly, they have many owners, each with private policies and requirements that control their participation in the community. In this environment, flexibility is a key to surviving in such hostile environment [TTL05].

1.1.3 Grid Computing

Grid Computing is a type of parallel and distributed system that enables the sharing, selection and aggregation of geographically distributed "autonomous" resources dynamically at run-time depending on their availability, capability, performance, cost, and users' quality-of-service requirements

The grid computing concept can be seen as a proposition to share computation resources in order to reach a common goal. In this way, resources are at the front of the definition but without changing the main signification. In [FZRL08], a comparison between grid and cloud computing is proposed which positions these two topics considering different aspects. This comparison reveals that both concepts share part of their background such as architecture and technology. In addition, the authors propose a figure that tries to situate cloud and grid depending on architecture scale and function orientation. This figure could be completed with Reconfigurable Computing paradigm, see figure 1.2.

1.1.4 Cloud Computing

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [1].

As explained by this definition, one of the main problem in cloud computing is the possibility to hide the structure details from the users and provide resources on demand. This concept

1. Definition from the National Institute of Standards and Technology

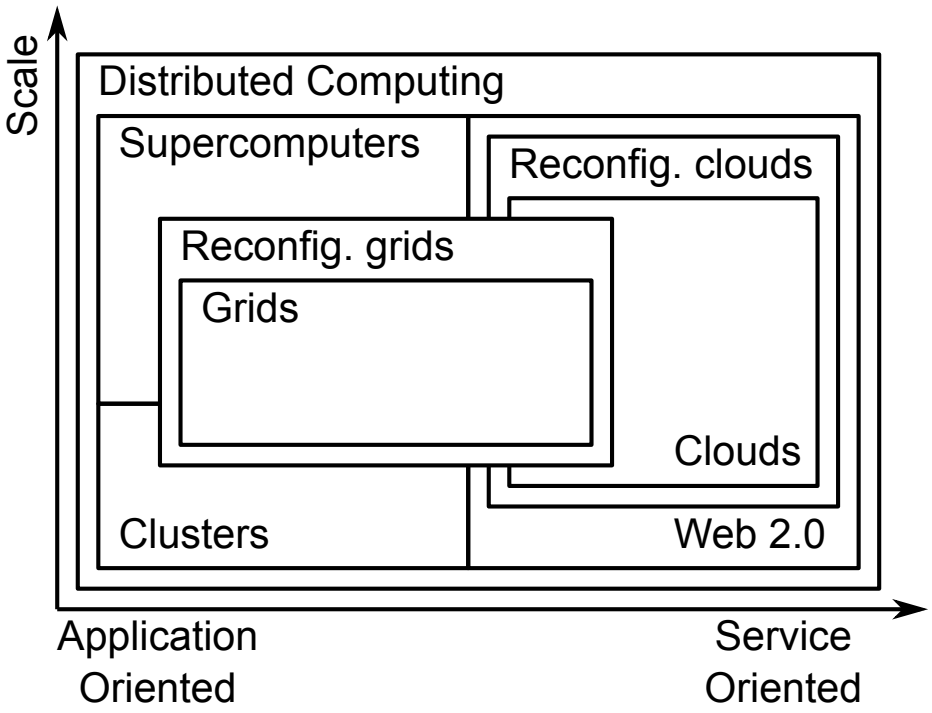


Figure 1.2: Grids and Clouds Overview [FZRL08] including reconfigurable technology

is also related to this work because it is the solution to see a networked system as a unique entity which can provide computation power. Ultimately, users do not need to know how their data are computed. They just need to obtain a result rapidly.

1.1.5 Complex Adaptive Computing

The basic elements of a Complex Adaptive System are agents. Agents are semi-autonomous units that seek to maximize their fitness by evolving over time. Agents scan their environments and develop schemas. Schemas are mental templates that define how reality is interpreted and what are appropriate responses for a given stimuli. These schemas are often evolved from smaller, more basic schema. These schemas are rational bounded: they are potentially indeterminate because of incomplete and/or biased information; and they differ across agents. Within an agent, schema exist in multitudes and compete for survival via a selection-enactment-retention process [Doo96].

As described in this definition, a Complex Adaptive System adds the notion of behaviour, awareness of the environment and capacity to learn from experience. These are fundamental characteristics to describe a technological entity that is designed to be autonomous and intelli-

gent. Applied to computer science, this notion can be called Complex Adaptive Computing.

1.1.6 Natural Computing

Natural Computing refers to computational processes observed in nature, and human designed computing inspired by nature. When complex natural phenomena are analyzed in terms of computational processes, our understanding of both nature and the essence of computation is enhanced. Characteristic for human-designed computing inspired by nature is the metaphorical use of concepts, principles and mechanisms underlying natural systems. Natural computing includes evolutionary algorithms, neural networks, molecular computing and quantum computing [Div02].

1.1.7 Other Research Directions

In addition to the entire topic of cooperative networking, a lot of different notions are also relevant to study Reconfigurable Computing. Bio-inspired Computing, Artificial Intelligence, Machine Learning, Informatics, Multi-Agent Systems and Biology provide common knowledge that should be observed, shared and reused. This common knowledge describes two aspects that are a collaboration and a competition and that define the relation between so called elements, cells, nodes, individuals or agents, depending on the application field. Because of the diversity of information, they will not be set out in detail in this work, which is not meant to minimize their importance. Moreover, as described in [Wal92], the question of complexity is important. It shows how sciences are interconnected and constitute knowledge.

1.1.8 Computing Overview and Classification

Finally, figure 1.3 tries to classify ideas related to the computing concept. It also includes the combination of Natural Computing and Reconfigurable Computing presented in this work. Two generations are revealed, the first of which consists of the historical ideas that are the basis of our technology. The ideas of the second generation are the actual ideas discussed by the science and technology communities.

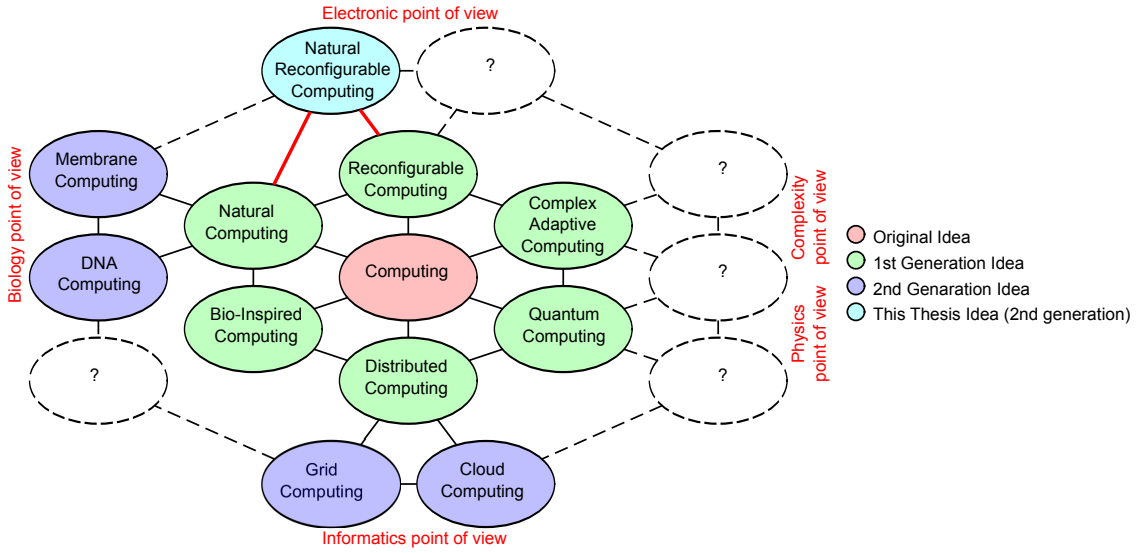


Figure 1.3: Computing Classification Proposition.

1.2 Self-organised Computing and Systems

1.2.1 Self-Organization and Emergence

The notions of self-organization and emergence are related and can be used to characterize the behavior of networked systems. Because of their similarity, they need to be clarified in order to specify their links. Self-organization can be defined as a dynamical and adaptive process where systems acquire and maintain structures themselves without external control. In addition, a system exhibits ”Emergence” when there is coherent emergent at the macro-level that dynamically arise from the interactions between the parts at the micro-level. Such emergents are novel with respect to the individual parts of the system [DH05].

These two definitions are part of the core necessary to understand this work. In [DH05], the conclusion of the authors points out that: *Both phenomena can exist in isolation, yet a combination of both phenomena is often present in complex dynamical systems. In such systems, the complexity is huge, which makes it infeasible to impose a structure a priory: the system needs to self-organise.[...] A combination of emergence and self-organisation, which is already applied in literature, is a promising approach to engineer large-scale multi-agent systems.*

There is an important reference used by the author of [Jov09] where a dissertation about the definition of self-organized systems is proposed which exactly fits the ideas of the work proposed here and can be summarized as follows: *A system defined as complex and modular*

able to restructure itself can be considered as self-organized if it shows new properties that emerged from the interaction of its elements and without external control. The objective is to adapt itself to unpredictable change of its environment or optimise its own behaviour depending on pre-set criterion.

1.2.2 Associated Properties

The previous definitions can be completed by a set of properties.

- **Micro/Macro Effect:** This effect is the main characteristic of emergence. It describes how macro behaviour can be the consequence of interactions at micro level. These interactions are somehow hidden by the macro level behaviour [Gol99] [Hey89].
- **Interactivity:** As described in Micro/Macro Effect, interactions are fundamental and they are related with communication aspects that are described in the next sections. The micro elements in interaction are responding to messages provided by each other with the objective to adapt themselves. In self-organized systems, interactions are by definition complex [Gol99] [Hol98].
- **Complexity:** A system can be considered to be complex if it is composed of dynamic and multiple interactive elements such that its behaviors cannot be understood only from the sum of individual behaviours [Ash47] [Edm95].
- **Dynamic:** Here, the property of dynamic concerns rather the evolution of the self-organized system with time and in addition, the fact that any system, in the widest sense, is trying to reach a certain equilibrium by adapting themselves. It can also be mentioned that this notion is closely related to the question of entropy which quantifies the expected value of the information contained in a message [Hey03] [NP77].
- **Adaptivity:** The capacity of this system to maintain its main functions in response to external perturbations coming from its environment. This capacity of adaptation requires autonomy in the behavior of systems [HJ01].
- **Autonomy:** Autonomy concerns local system behavior (micro level). It depicts the capacity of a system to organize its activities and communications spontaneously. Reaching this kind of autonomy causes it to develop sturdiness and flexibility [Hey89] [MFH$^+$03].
- **Sturdiness and Flexibility:** As elements are becoming stronger and stronger by accumulating characteristics in a virtuous circle, a certain sturdiness and flexibility can appear at both micro and macro level. In other words, a group can become stronger because elements are becoming individually stronger. In addition to this evolution, elements are capable to control themselves and their local communications [Ger05] [Hey03].

- **Decentralised Control:** Decentralized control is based only on local control strategy which is the distribution of the capacity to manage function. No element exists that can control explicitly an entire function for the entire group but the sum of those local controls can do it. To complete this later definition, the property of pre-emption can be added to describe the possibility to act on the control parameters in prediction of an future event [Ser04] [DH05].
- **Preemption:** It can be defined as a special case of adaptivity. Pre-emptive modification can be considered a pre-adaptation which marks the dependence of future events on past events. Beside this, other forms of dependence can be seen with the bi-directionality property [Ros85] [PHF07].
- **Bi-directionality:** This concerns links between low level and high level elements. It can be summarized as follows: micro-elements can influence macro structure as well as macro structure influences micro elements. Bi-directionality is also linked with Radical Novelty.
- **Radical Novelty:** It describes properties that are totally new in the sense that they emerge from a self-organized structure [dV97] [Gol99].
- **Coherence:** To complete this properties list, coherence needs to be defined. Emergence and self-organization properties are also part of this system. It concerns local behavior and structural behavior. Here, both of them are correlated. Moreover, coherence refers to the equilibrium between competition and cooperation possibilities that rules the system environment [Gol99] [Ode02].

This set of properties seeks to propose a logic statement that makes all individual properties a part of a complete description. This topic is only a part of the work proposed here and must be completed with many other notions that will be presented in the following. In particular, three properties are more interesting and are selected as subject of study: autonomy, dynamic and interoperability.

1.2.3 The Three Properties: Autonomy, Dynamic and Interoperability

Electronic technology is a basis of the integration of autonomous behavior and ultimately intelligence into self-organized systems. Robotics, learning machines and biological informatics are examples of directions that are studied currently and which provide information concerning intelligence, behavior and communication using electronic technologies. Those examples can be helpful to get a general idea and to locate the proposed work. To complete this situation, some definitions help to put limits to the work area.

The first one is the definition of Autonomy which can be defined as follows:

Autonomy is the capacity of someone or something to be not dependent from somebody or something else, it is the characteristic to function or evolve independently of anything.

This definition can be adapted to the field of Autonomous Systems as the possibility that is given to systems to operate without being externally controlled. It is the first important property that is at the center of this work. Within this definition, the autonomy has the main advantage to divide the global control problem into local control sub-problems that are easier to solve. This can be used to discuss decentralized control management. Moreover, local autonomy implicitly requires local intelligence.

The definition of intelligence is a question that is still hot. Many propositions are available and they constitute the basis of a fundamental topic. For this work, a definition oriented around computer science is selected. It describes Artificial Intelligence (AI) and it helps to distinguish what is real Intelligence and what is simple Automatism of systems. AI is the summation of theory and techniques used to realize machines able to simulate human intelligence Automatism (of a device or process) is the capacity to work by itself with little or no direct human control These two definitions are making a distinction between what can be considered intelligent and a simple automatic behavioral response due to precise circumstances.

Intelligent systems are defined by the integration of AI into systems. In other words, AI is the intelligence of machines and the branch of computer science that aims to create it. In addition, locally autonomous and intelligent systems should be communicative. This means that those systems should be interoperable with any other communicative systems.

Interoperability is the ability of systems to exchange and make use of information in a straightforward and useful way; this is enhanced by the use of standards in communication and data format.

This interoperability characteristic (the second important one) is used to integrate this kind of technology into existing systems which also means to follow the dynamic evolution.

Dynamics (of a process or system) is characterized by constant change, activity, or progress.

This is the third important characteristic and it is also a general truth about the global environment that represents our world.

Figure 1.4 shows the three important characteristics considered in this work. They are linked together into a virtuous circle: Dynamics requires Autonomy which requires Interoperability.

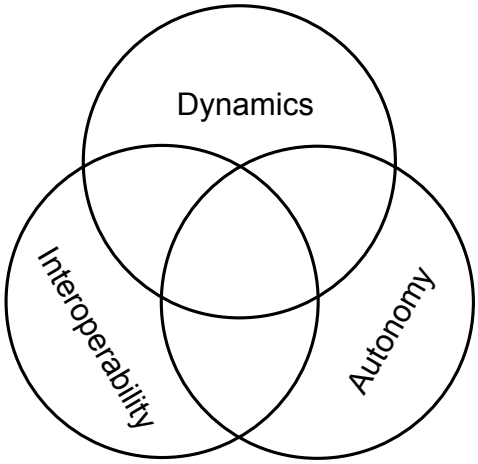


Figure 1.4: Three important properties of self-organized system considered for this work

1.2.4 Local and Regional Behaviour

Two remarks are interesting regarding behavior. The first remark is about the local behavior (micro-level) of a networked system where each element can be considered either a server or a client. It is a well-known concept taking into consideration the fact that users provide intelligent functionality to allow local elements have a dynamic behavior. Consequently, local elements are dependent of their electronic architectures which impact on performances.

The second remark concerns the regional level (macro level) where a system composed of multiple elements is the result of elements' interoperability. The inner collaboration of self-organization systems is as important as the node description. Properties of dynamics and autonomy can be completed with interoperability which combine everything into a good wheel cycle. Intelligence generates Autonomy which in return generates Interoperability. Here, the system is dependent on communication technologies and networking solutions.

1.3 Networking and Communication

This section is an observation of actual technology and knowledge. The goal is to show important points such as electronic device heterogeneity, communication protocol diversity and the general decentralization. Figure 1.5 shows three observations of actual technologies and systems.

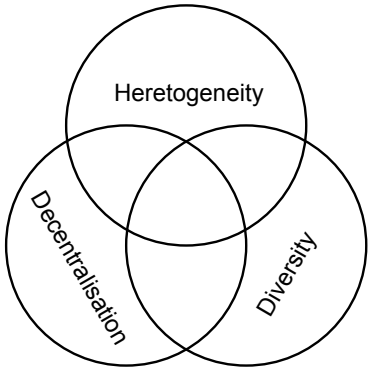


Figure 1.5: Three observations associated with technology and the self-organized concept

1.3.1 Heterogeneous Environment

Networked systems are designed to be connected to a global communication infrastructure. By doing so, they are thought to be connected with a wide variety of systems that constitute a heterogeneous environment. Notebooks, smart phones and now tablet devices are new examples that are easy to find. These examples show that communications have significant consequences on hardware development. Parameters concerning size and power consumption are key features. In addition, the flexibility and the possibility to share resources has become increasingly important.

In the environment, the heterogeneity has the main advantage of providing a huge quantity of solutions for many different problems. By allowing this, the best solutions can be found in order to improve the selected parameters. For example, using common personal computer architecture, a lot of solutions are discussed to provide access to CPU when it is not used by local users.

More precisely, heterogeneity means diversity in character or content: a large and heterogeneous collection. In chemistry, it denotes a process involving substances in different phases (solid, liquid, or gaseous). Using this definition, a parallel can be drawn between chemistry and networked computing where elements can be changed during time.

1.3.2 Protocol Diversity

Diversity is a characteristic of what is diverse, different; taste variety, plurality. Diversity of an ecosystem, size in relation with the number of species and the number of individuals of each species on a given territory. A great diversity is a guaranty of stability because each species can be in one or more food chains and by consequences, if one species does not contribute to

an aspect of the ecosystem, another one can probably do it.

To develop heterogeneous communicative systems, various protocols have been proposed. They are characterized by many parameters like media support, data transfer rate or security level. This diversity is also a problem because it means a possibility to see devices that necessitate special protocols to communicate with. A solution here is to develop devices that include different communication capacities, for example, smart phones are now built with *Universal Mobile Telecommunications System* (UMTS), *Wireless Fidelity* (WiFi) and Bluetooth interfaces. This kind of solution is the easiest way to provide a universal communication device but at the cost of redundant hardware. As for the heterogeneity observation, protocol diversity allows to study different solutions with the objective of finding a better way to communicate. When it is combined with the encyclopedic definition, it can be observed that networked systems use this diversity to extend their sizes and evolve in time and space. Based on this assumption, a definition that fits the proposed work is one that describes a characteristic of systems that are composed of multiple and different elements. It is guaranty that those systems are resilient and can survive in the time and in the space.

1.3.3 Decentralised Systems

Decentralization is the transfer or sharing of decision-making power from a central networked element toward lower-level elements. It signifies the division of power from the top down within any organizational hierarchy.

The last point concerns the field of command and management. At the global scale, systems are controlled by a completely decentralized structure. The internet is an example for decentralized systems. It shows that the entire structure can work without any central command system. Moreover the information routing systems allows to dynamically adapt data orientation depending on the system life. This kind of architecture is a powerful way to provide the capacity to change in the time and the space to any system. Based on this observation, many different aspects concerning networked system are studied and can be presented under the denomination of Complex Networked Computing. These are a set of different points of view connected to the general idea of sharing available resources to reach a common goal. This topic and its different orientations are presented in the next section.

1.3.4 Peer-to-Peer Application and Concept

On top of this structure, *Peer-to-Peer* (P2P) applications come as completely decentralized sharing system. They can be described as follows: In a P2P network, the "peers" are computer systems which are connected to each other via the network. Data to compute can be shared directly between systems on the network and without a central controller. In other words, each computer on a P2P network becomes a data provider as well as a data requirer.

To reach this goal, electronic research and engineering works are used to realize the infrastructures and the devices that integrate different functionalities. These functionalities are completely integrated in embedded electronics that have major constraints in terms of power consumption and logic areas.

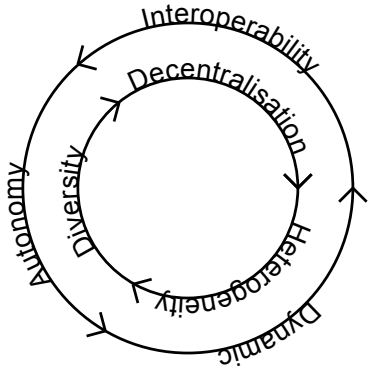


Figure 1.6: The three important properties of self-organized systems.

Figure 1.6 shows the three important characteristics considered in this work which are also related to the properties observed earlier: Heterogeneity, Diversity, Decentralization. They are linked together into a virtuous circle: Dynamic requires Autonomy requires Interoperability.

1.4 Electronic Technologies: Configuration Point of View

In this section, the main technology used for this work is presented with a focus on the configuration point of view of the FPGA technology.

Figure 1.7 shows the three main chip technologies that characterize electronic engineering. Moreover, it shows where FPGA is positioned considering the development paradigms and their combinations: programming, configuration and design.

Until now, most electronic systems are designed to be static. Here, static means that when a system is designed, there is no solution to modify its processing part except by adding a exten-

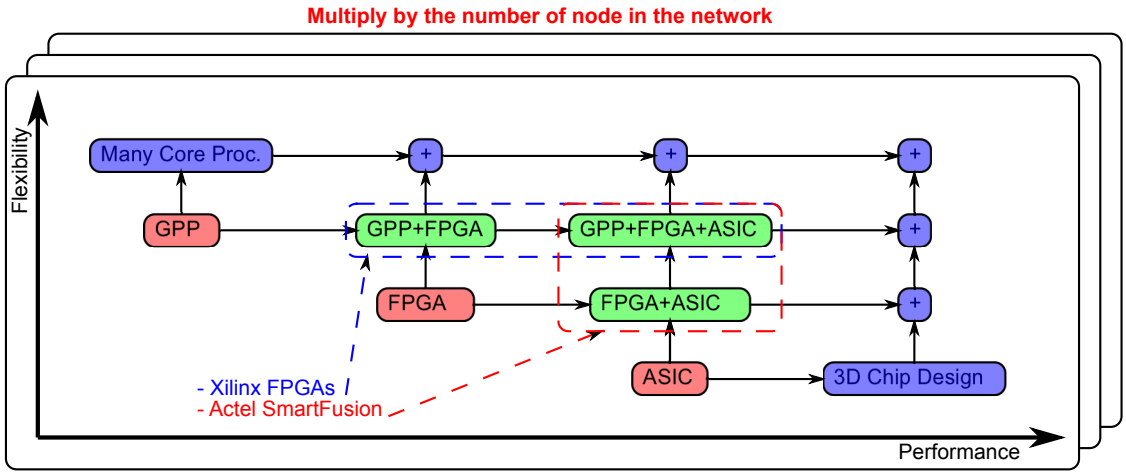


Figure 1.7: The three main technologies and possible combinations and extensions.

sion board using common interfaces. *Peripheral Component Interconnect Express* (PCI-E) is a good example of these interface. In this global concept of building systems, it is a main limitation regarding the dynamic environment where electronic systems should evolve. The main limitation is the necessity to shut down the local system when a new hardware computation functionality needs to be added. This context has led to the emergence of FPGA technology which is a descendant technology of the concept presented in [Est60]. This concept proposes an electronic architecture composed of *a fixed plus a variable structure*. The rest of this section presents the original chips technologies and the different FPGA configuration possibilities.

1.4.1 General Purpose Processor Design

It is well known that *General Purpose Processors* (GPP) offers flexibility. Using this kind of architecture with today's multi-core chips has the advantage of clearly separating the hardware part from the software part. Using an operating system to manage hardware resources, software solutions can be developed without the entire knowledge concerning the hardware. This fact is not completely true but can be generalized to most application developments. This is the main advantage of this kind of technology but unfortunately the hardware side is not flexible and it is based on a technology that leads to the obsolescence of the platform. On the other side, multi-core chips can be seen as coarse-grain reconfigurable system where core can be used to different purpose. Consequently, these architectures are therefore a limited solution for self-organized systems.

1.4.2 Application-Specific Integrated Circuit Design

On the other hand, *Application-Specific Integrated Circuit* (ASIC) designs are the best solution to provide the best performance for a given application. Unfortunately, performance has a cost, in terms of development and in terms of capacity restriction. But in spite of the best performance that can be achieved for any functionality implementation, it should be noticed that it is also possible to design mixed technology chips. This kind of chip includes static parts that can be designed for a specific application and beside this, reconfigurable structures can be implemented.

The Actel SmartFusion™technology is good example of this mix. As presented by Actel, SmartFusion intelligent mixed signal FPGAs are, at the moment, one of the rare devices that integrate an FPGA, ARM® Cortex™-M3 and programmable analogue component offering full customization, IP protection and ease-of-use. Based on Actel's proprietary flash process, SmartFusion FPGAs are ideal for embedded hardware designers who need a true system-on-chip (SoC) solution that gives more flexibility than traditional fixed-function micro-controllers without the excessive cost of soft-core processor cores on traditional FPGAs [2].

Following the evolution of the technology and the arrival of 3D chip design in addition to many-core processors, new technology combinations can be expected providing extreme flexibility and performances but also new challenges like hit dissipation in those complex chip and distributed memory management.

1.4.3 Field Programmable Gate Array

For this presentation of the FPGA structure, a focus is proposed on the FPGA *Static - Random Access Memory* (S-RAM) technology that allows to define an FPGA as reconfigurable as opposed to the one-time configurable FPGA based on anti-fuse technology. As example, the FPGA architecture of the Virtex series from Xilinx is presented.

Figure 1.8 shows how FPGA's *Configurable Logic Block* (CLB) resources are routed inside the FPGA [Inc02]. The Configurable Logic Blocks (CLBs) are the main logic resources for implementing sequential as well as combinatorial circuits [Inc10a]. It is based on an interconnected logic grid that can be configured in order to connect logic blocks. The goal is to realise a logic function. To achieve this, configuration tool chains are designed to offer a maximum of flexibility into the FPGA utilization.

2. Description from www.actel.com

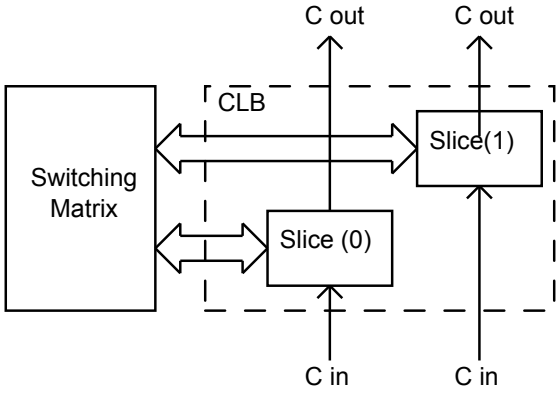


Figure 1.8: Virtex Local Routing [Inc09]

First-Configuration

Within this reconfigurable FPGA technology, different configuration modes are possible and are detailed in any FPGA configuration guide [Inc10b]. Precise technical aspects are not the main point here but instead, the focus will be on the description of reconfiguration aspects. To run an FPGA-based system, a first configuration is necessary to set up and start the FPGA. This step is possible by simply loading a bit file into the FPGA. Bit files contain FPGA configuration information. Considering a SoFPGA design, this step is used to set all communications with other electronic chips like Ethernet, memories and communication controllers. It is also used to boot a possible *Operation System* (OS) which is a key part of any computing machine. Unfortunately, SRAM based FPGA does not keep a configuration in memory when turned off.

Total-Reconfiguration

After the first configuration of the system and if no problem appears during this phase, it may be necessary to RE-configure the system to correct an error in the logic design. This possibility is one main advantages of SRAM-based FPGAs, and it is why they are very popular for development processes.

In electronic design build with at least one FPGA chip and one GPP chip, total reconfiguration is the possibility to reconfigure the entire FPGA with another configuration file. In this case, FPGA can be considered a reconfigurable co-processor. Such a design is already a powerful solution that allows to dynamically implement parallel hardware structures that are dedicated for any special computation.

Partial-Reconfiguration

FPGA technology provides the flexibility of on-site programming and re-programming without going through re-fabrication with a modified design. Partial Reconfiguration (PR) takes this flexibility one step further, allowing the modification of an operating FPGA design by loading a partial configuration file, usually a partial bit file. After a full bit file configures the FPGA, partial bit files can be downloaded to modify reconfigurable regions in the FPGA without compromising the integrity of the applications running on those parts of the device that are not being reconfigured [Inc10a].

With this capacity, a complete implementation in a SoFPGA that has the possibility to reconfigure itself is possible. In such design, the static region can integrate a processor used as a controller for the reconfigurable region. The integration is deeper in this case and allows to reduce a lot of electronic problems like the design of PCB, the power consumption or line delays.

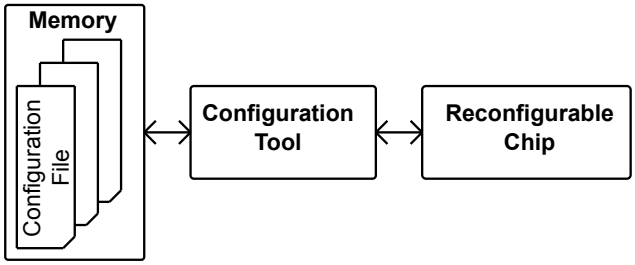


Figure 1.9: Simple Configuration System.

Figure 1.9 presents a simple configuration system that can be used in the case of first configuration, total configuration and partial reconfiguration. It simply shows a structure to load a bit file into an FPGA device.

Dynamic-Reconfiguration

Dynamic partial reconfiguration allows to reconfigure a FPGA part during run-time. It is an improvement in a way to think about reconfiguration and electronic design. That is the possibility to modify a circuit without the necessity to shutdown the entire electronic board. Of course, it is not a physical modification but the concept of reusing a grid of logic blocks to implement the more adapted function gives another degree of liberty in the design of electronic devices.

For example, design methodologies are in any case dependent on human decisions, and because making errors and mistakes are characteristics of human beings, they can be made at any moment in the development process. Having the possibility to modify part of a design after fabrication is a great advantage.

Auto-Reconfiguration or self-reconfiguration

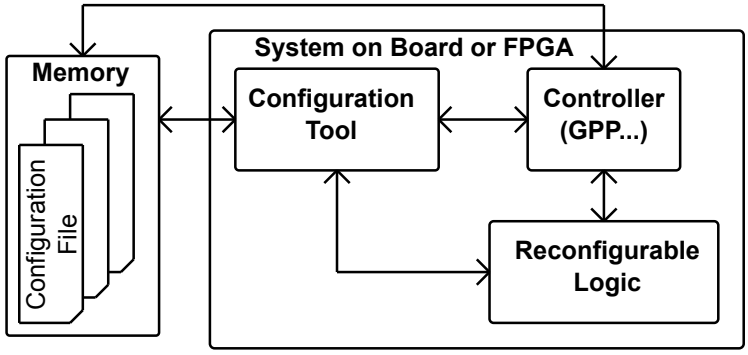


Figure 1.10: Simple Configuration System

Auto-reconfiguration states for SoFPGA that it can reconfigure itself at run-time. Ultimately, auto-reconfiguration allows systems to change themselves by using the same electronic resources to realize a new function at the hardware level. It is a combination of partial reconfiguration and dynamic reconfiguration. It is the main technological aspect that is used in this work. Figure 1.10 is a representation of a auto-reconfigurable system. To enable this auto-reconfiguration, an internal access to the logic is required.

System-Reconfiguration

System reconfiguration is a term used to describe how a system can be reconfigured independently of the technology. In other words, systems are able to reconfigure themselves with or without FPGA. For a networked system, reconfiguration is the possibility to change the configuration, not at the hardware functionality level but at system functionality level, where its organization of tasks can be managed. It represents computing using standard computer architecture which was described earlier.

1.4.4 Reconfigurable System Versus Adaptive System

Beside reconfigurable system, the field of adaptive systems which has the aim to propose systems able to adapt some of their characteristics is interesting. Adaptive systems are for example crucial for space application which are specially limited in the possibility to access the system due to its localization [Vis10]. Both reconfigurable and adaptive systems are trying to provide flexibility.

1.4.5 Embedded Electronic Design

Embedded electronic devices are characterized by some additional constraints in comparison with classical electronic system. These two main characteristics are logic area and power consumption. These two factors can be solved by FPGA reconfiguration technology where a region in the FPGA can be isolated in order to reuse its logic components for different functions or to completely remove a module when hardware acceleration is not necessary. The reusability and moreover the possibility for the system to do it alone is one of the best advantages that is given to design modern systems.

1.4.6 Possible Combinations

To improve the power of electronic design, some combinations can be made from GPP, FPGA and ASIC chips. As presented in figure 1.7, three possibilities can be proposed. By adding ASIC design and FPGA structure into the same chip, new architectures can be proposed like competitive analog interface combined with additional reconfigurable logic. Another solution which is already available is Reconfigurable SoFPGA where GPP can be combined with dynamically reconfigurable IPs. Beside that, the combination of all three GPP, FPGA and ASIC fabric could be an ultimate possibility to provide maximum flexibility and maximum performance in a unique chip. In this work, Reconfigurable SoFPGA are studied and an analysis is proposed in chapter 3, 4 qnd 5.

Conclusion

This chapter tries to give a general idea concerning the position of reconfigurable computing research and FPGA technology in the middle of computer science. Figure 1.11 shows the background of the concept proposed in the next chapter. It explains that the technological environment reveals some properties like heterogeneity, diversity, decentralization, for which

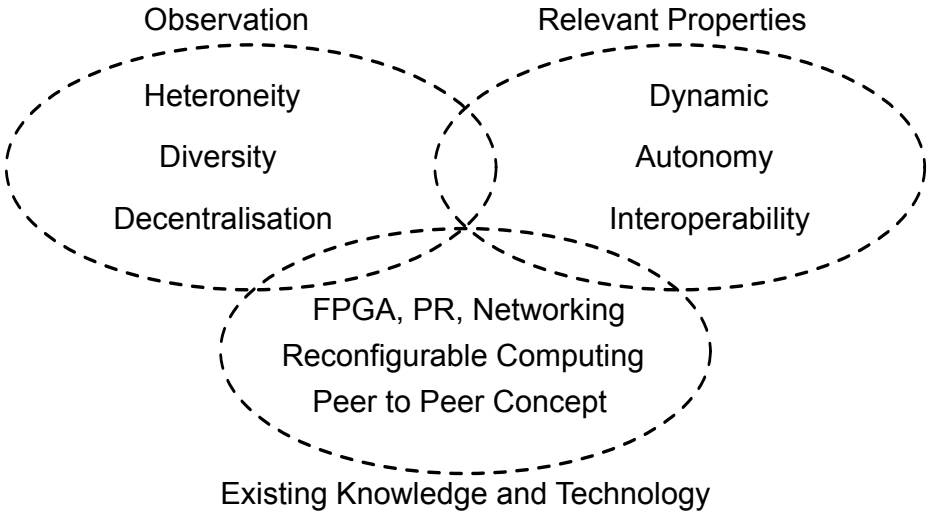


Figure 1.11: Work Background and main properties for reconfigurable and self-organized system.

it is possible to oppose other properties, such as Dynamic, Autonomy and Interoperability in addition to existing concepts and technologies, FPGA, networking, reconfigurable computing and peer-to-peer concept. After the presentation of the background, the state of the art will be presented in the next chapter.

Chapter 2

Review of Recent Related Works

Introduction

This chapter aims to present the state of the art of the topic. The projects presented do not constitute an exhaustive list of all existing projects but instead, this list tries to review the most recent research in the field of reconfigurable computing. Three different sections are used to precise some historical references, some high level on board systems and finally, some low level on chip systems.

2.1 Classical and Historical References

2.1.1 Classical references

To start this presentation two references can be cited. The first one is Static and Dynamic Configurable Systems (1999) by Eduardo Sanchez and Moshe Sipper. This paper describes four applications in the domain of configurable computing, considering both static and dynamic systems, including SPYDER (a reconfigurable processor development system), RENCO (a reconfigurable network computer) Firefly (an evolving machine), and the BioWatch (a self-repairing watch). [SSH+99]. In particular, the Firefly application can be mentioned [SGM+97]. It describes a cellular programming approach together with an FPGA-based implementation. The results shows that evolware systems exhibit enormous gains in execution speed compared to execution on high-performance workstation.

The second reference is *Dynamic Hardware Plugins in an FPGA with Partial Run-time Reconfiguration (2002)* by Edson L. Horta, John W. Lockwood and David Parlour [HLTP02]. It shows the utilization of partial-reconfiguration in a run-time reconfigurable architecture and a design methodology design of the static and the dynamic parts of the system. In addition, the authors use this kind of architecture in network oriented systems. In [Loc01], a prototype platform named Field Programmable Port Extender (FPX) is presented to experiment on evolvable internet hardware.

2.1.2 Point-to-point Connection and Communication

In the case of communication aspects and in particular point-to-point communication, the work of Matthias Dyer and Marco Platzner (2002), [DPP02] [DW02] is useful. In this work, a networked reconfigurable system is proposed. It shows the integration of a complete system onto FPGA, including on LEON core processor used as a controller and a Ethernet connec-

tion. At the time, the communication aspects were already discussed and their work showed a complete communicative and reconfigurable platform.

2.1.3 Bus Interconnection

In the case of an architecture of reconfigurable systems on FPGA, different systems can be presented: the tool-set presented in [MMP+03], the bus systems presented in [HBB04] [HUKB04], the generic model presented in [MCM+04] and the runtime environment for reconfigurable Hardware Operating Systems presented in [WP04].

All these systems describe different aspects of the *on Chip* technology. They showed bus interconnection for multiple reconfigurable region. These buses are designed to offer a maximum of flexibility in the management of reconfigurable modules.

In addition, these systems are based on a decade of research on reconfigurable computing that saw many different projects [Har01]. In [MMP+03], a figure tries to describes the reconfigurable architecture until the beginning of the century.

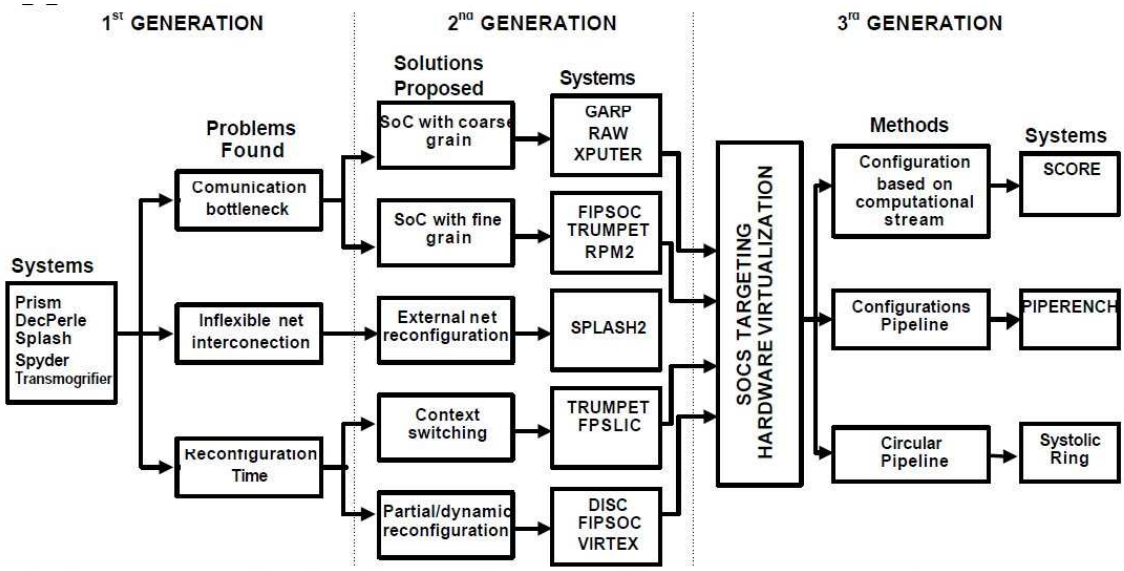


Figure 2.1: Evolution of reconfigurable architecture [MMP+03]

Three versions can be observed: the first one aims to increase GPP performances, the second one tries to minimize the bottleneck between GPP and FPGA and last one combines reconfigurable architecture and dataflow-based algorithms [MMP+03].

2.1.4 Operating System and Reconfigurable Architecture Co-design

On top of this architecture, a software part is necessary to control all functionalities. In [MMB+04], the Gecko and Gecko² platforms are presented. They combine a reconfigurable SoC implemented on a Virtex 2, an ARM processor embedded in a Compaq iPAQ 3760 and an extended operating system used to manage multitasking on multiple hardware resources. Results show the complexity of specific operating system built with a special scheduler.

2.2 High level Multi-Board Systems

2.2.1 RecoNodes/ReCoNet Systems

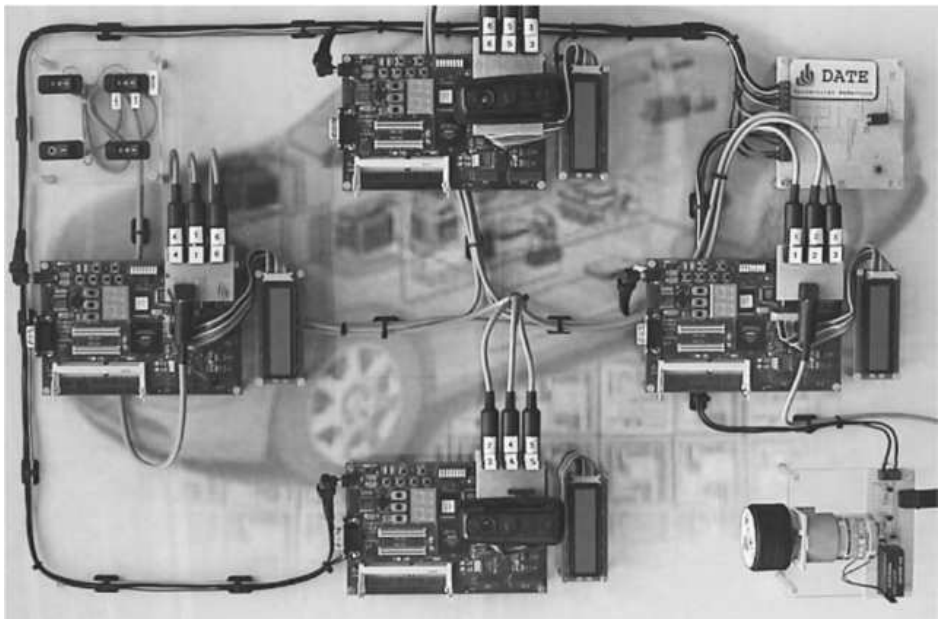


Figure 2.2: Multi board project from the ReCoNet Project. [HKT03]

The aim of this system is to overcome the deficiency of design automation of reconfigurable devices, in particular FPGA-based architectures. This goal is likely to be reached by supplying models and optimization methodologies for dynamic hardware reconfiguration. Those models and methods are part of a kind of operating system for hardware reconfiguration in charge of the resource management at run-time. Concretely, the investigations target mathematical optimization of strategies and methods for the optimal management and use of novel and future

generation of reconfigurable hardware. Those reconfigurable chips are currently used in different technical systems. Due to the practical barrier like the high reconfiguration overhead as well as the lack of theoretical models and methods, the potential of reconfigurable hardware could be only narrowly exploited. The goal is to show that existing technologies can be used to overcome many difficulties mentioned above. New impulses are therefore expected for the development of a new generation of chips.

Figure 2.2 shows the example selected for the ReCoNet project which is a self-healing networked system using reconfigurable technology. It is designed for an automotive application [HKT03]. The ReCoNet project is a full project from which a lot of other technology fields can be of benefit. It shows a number of properties like self-healing, autonomy and flexibility which make a system autonomous in its hardware utilization and management. For example, the ReCoNet project has contributed to the development of a set of tools that allows to easily realize Reconfigurable Systems.

2.2.2 Scalable Self-Configurable Architecture for Reusable Space Systems (SCARS)

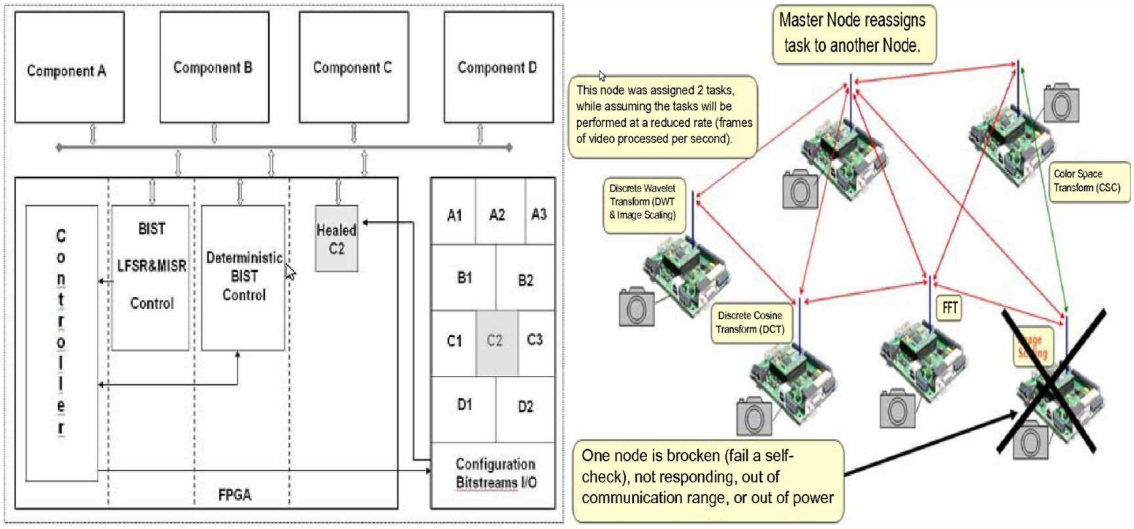


Figure 2.3: Local and networked self-healing solution of the SCARS system [SJAS08].

The *Scalable Self Configurable Architecture for Reusable Space Systems* (SCARS) is specifically focused on developing an architecture in which individual, modular components and subsystems are integrated in order to:

1. Coordinate their actions for a broader range of objectives, hence go beyond mission-specific requirements;
2. Adapt to changes in mission objectives over time and optimize computing and communication capability;
3. Respond to hardware/software anomalies automatically with self-healing action at both node and network levels.

It proposes a two-level self-healing methodology for increasing the probability of success in critical missions. This proposed system first undertakes healing at node-level. For this purpose, a built-in self-testing and fault detection, isolation and recovery capabilities to offer 100 percent node availability are developed. Failing to rectify the system at node-level, network-level healing is undertaken. The network automatically assigns the task of the faulty node to another node in the field. That field node then reconfigures itself to carry out the new task while running its original task. The prototype reconfigurable architecture demonstrates the network's capability for self-configuration and each node's capability for self-testing, fault-recovery/repair and computation optimization in the context of image processing [1].

Figure 2.3 shows the self-healing levels of the SCARS project. At the node level, the figure shows the organization of the structure that allows to use different resource sites as a possibility to repair itself. At the network level, the idea is to share modules and resources with the same objective. The properties shown here are self-healing, dynamic and adaptivity.

2.2.3 iBoard System

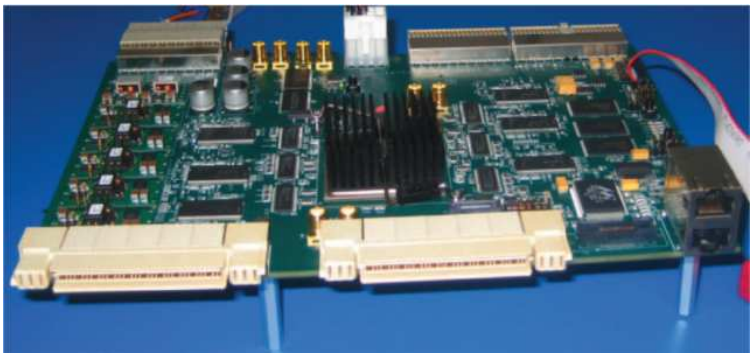


Figure 2.4: The IBoard Version 2 [HA10]

1. Description from the SCARS project team website: www2.engr.arizona.edu/ rcl/index.html and [SJAS08]

iBoard is a highly capable, highly reusable and modular FPGA-based common building block for instrument digital electronics. It is targeted to those space-borne instruments that require high-performance on-board processing capabilities. The design methodology is detailed. The requirements of flight instrument digital electronics is described and the implementation of the first prototype, iBoard 2 is presented in [HA10].

Figure 2.4 shows the i-Board system 2. The main advantage of this system is its integration into a complete vision of shared hardware resources. It is called *Instrument Shared Artifact for Computing*. It is here the question of the autonomy of different systems that can reuse common FPGA resources for computing.

2.2.4 Multimedia Oriented Reconfigurable Array

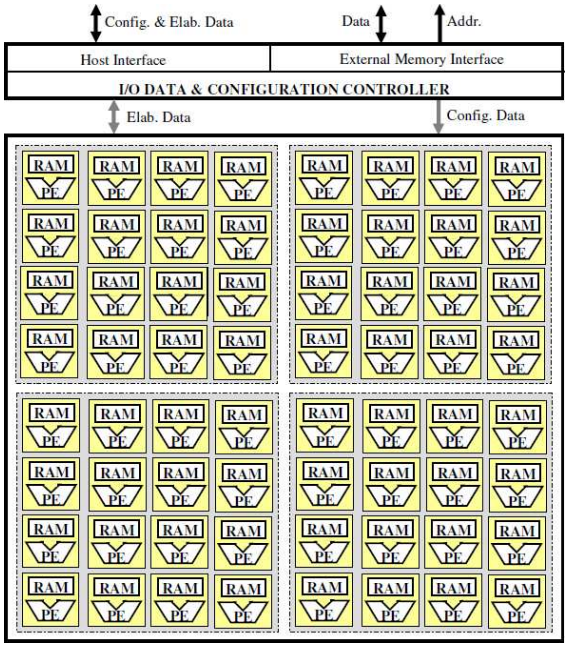


Figure 2.5: The MORA Architecture [LPC07]

The *Multimedia Oriented Reconfigurable Array* (MORA) project is a coarse-grain reconfigurable array optimized for multimedia processing. The system has been designed to provide a dense support for arithmetic operations, wide internal data bandwidth and efficiently distributed memory resources. All these characteristics are combined into a cohesive structure that efficiently supports a block-level pipelined dataflow which is particularly suitable for stream oriented applications. Moreover, the new reconfigurable architecture is highly flexible and easily

scalable. Thanks to all these features, the proposed architecture can be drastically more speed- and area-efficient than a state of the art FPGA in executing multimedia oriented applications [LPC07]. Figure 2.5 shows the architecture which is a array of processing units including both *Random Access Memory* (RAM) and *Processing Element* (PE).

2.3 Low Level Network-On-Chip Structure

2.3.1 OverSoC

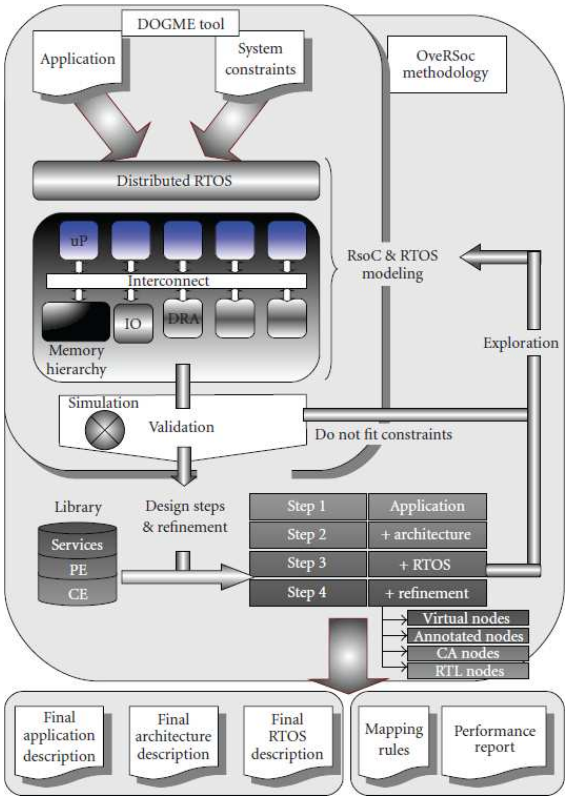


Figure 2.6: The OverSoC exploration and refinement flow [MHV+09].

The global methodology based on the original concepts addressed by OverSoC is the exploration of a distributed control of dynamic reconfiguration. In this way the methodology aims to explore the appropriate OS services that will be necessary to manage the RSoC platform. It relies on an iterative approach based on the refinement concepts as depicted in Figure 2.6.

Exploration is defined as an iterative process: modeling, simulation/validation and exploration. The inputs of the method are the specification of the application as a pure functional C

code and the system constraints. Once the system is validated, the design process starts again at a lower level of abstraction until the final system description. At each level of abstraction, the goal of the exploration depends on the separation of concerns paradigm. This paradigm is defined as a 4 steps process where the following concerns are successively addressed: application specification, architecture description, RTOS definition and platform refinement [MHV+09].

2.3.2 ReCoBus Project

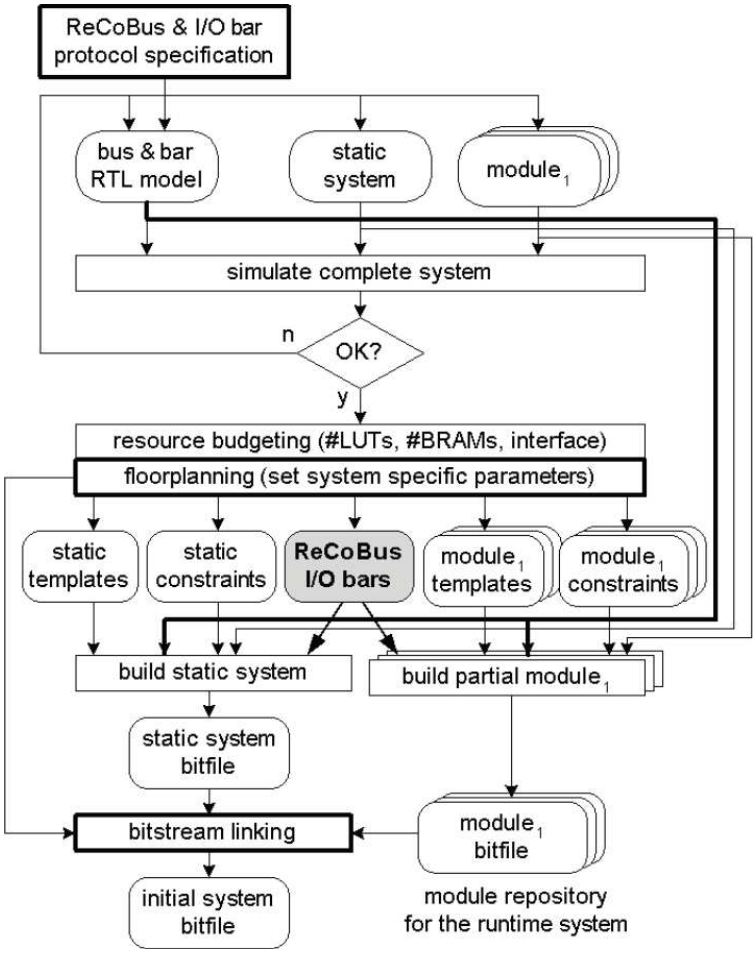


Figure 2.7: The ReCoBus-Builder design flow [KBT08].

To keep the design complexity in these days FPGA-based systems manageable, higher abstraction levels are highly desired. In future, FPGA designers will have to think more in modules and not so much in look-up tables. To assist this process, the ReCoBus technology provides a flexible back-plane bus that is highly optimized for regular structured FPGAs. This allows

for plugging pre-synthesized modules together in a manner that is comparable to the system integration known from PCB-based back-plane buses (for example VME, PCI, etc.). The integration of such modules is carried out without any time consuming synthesis runs or place and route iterations. Furthermore, by the use of partial reconfiguration, single modules can be hot-swapped in a few milliseconds at run-time [2].

Figure 2.7 shows the ReCoBus-Builder design flow. The processes in the fat boxes are performed by ReCoBus tools, while all other simulation, synthesis and place and route steps are based on external tools. This system provides an integrated design tool and the communication structure for reconfigurable modules [KBT08]. However, the specific function and paradigm for designing a self-organized system depends only on the designer. Using ReCoBus tools provides an easy to understand way to design reconfigurable systems. In combination with the dedicated bus that allows to connect reconfigurable modules, the complexity of the systems can be increased without making the development too difficult.

2.3.3 DyNoC

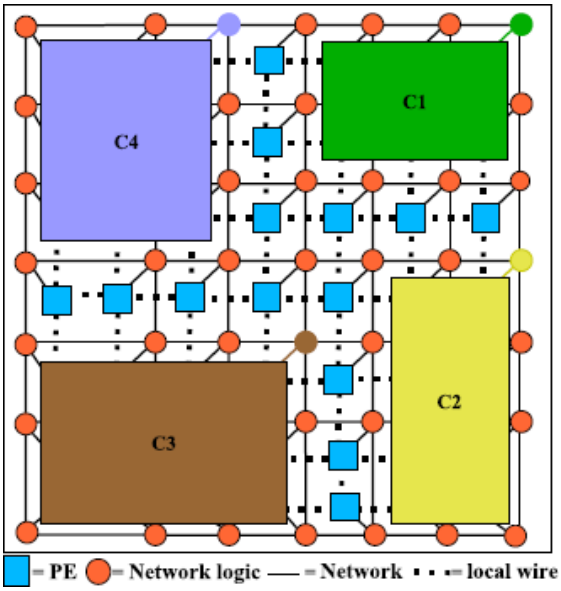


Figure 2.8: DyNoC modules implementation and routing [BAM+05].

The goal of the Dy-NoC project is to have a communication infrastructure in which the reachability of packets is ensured, independently from the changing topology which occurs

2. Definition from www.recobus.de

when components are placed and removed on the chip. In its basic state, the communication infrastructure is a normal NoC. *Processing Elements* (PEs) access the network via a network element. Additionally, direct communication paths exist between neighbor PEs. In this way, the network elements are only used for communication between non-neighbor PEs. As stated earlier, the placement of a module in a given region of the chip makes the routers in that region useless, since PEs belonging to the module are directly connected. The idea is then to implement routers as reusable elements which behave as routers in their basic configuration, but can be used by a component as part of its logic. Such router can be available as programmable hard macro on the chip. Whenever a component is placed in a given region, only one router is necessary for this element to access the network. Without loss of generality the router attached to the upper right PE of the module is used [BAM+05].

Figure 2.8 shows a DyNoc on board communication infrastructure. The DyNoC project is an example of FPGA internal communication structure.

The DyNoC Project shows a dynamic NoC implemented on FPGA. An aim of this project was to propose a communication structure that can be dynamically re-organized in order to provide a scalable interface between reconfigurable modules. It is an example used to show that communications are also studied at the logic level, inside the FPGA. Be consequence, the RSS proposes to extend this proposition to network level, outside of the FPGA. The selection of the DyNoC as reference for the RSS project is based on the following assumption:

- Extension of the discussion of communication: inside/outside the FPGA
- Discussion of dynamic configuration mode of “configuration computing” or using more actual terms: discuss the dynamic properties in the context of reconfigurable computing.
- Show that reconfigurable architectures are doing candidate to improve Complex Networked Computing.

It provides the idea that communication structure can evolve with the module placement and system activities.

2.3.4 C-U-NoC/Q-NoC

This centred communication approach allows communication between modules dynamically placed at the run-time on the chip. The C-U-NoC represents packet switched network of intelligent independent routers called Communication Unit (C-U). The main role is to route the address packet from the source to its destination in a dynamically changing network. The C-Us are characterized by novel routing algorithm based on the priority-to-the-right rule, their unique structure and specific connection with the computing resources [JTBW09]. Figure 2.9

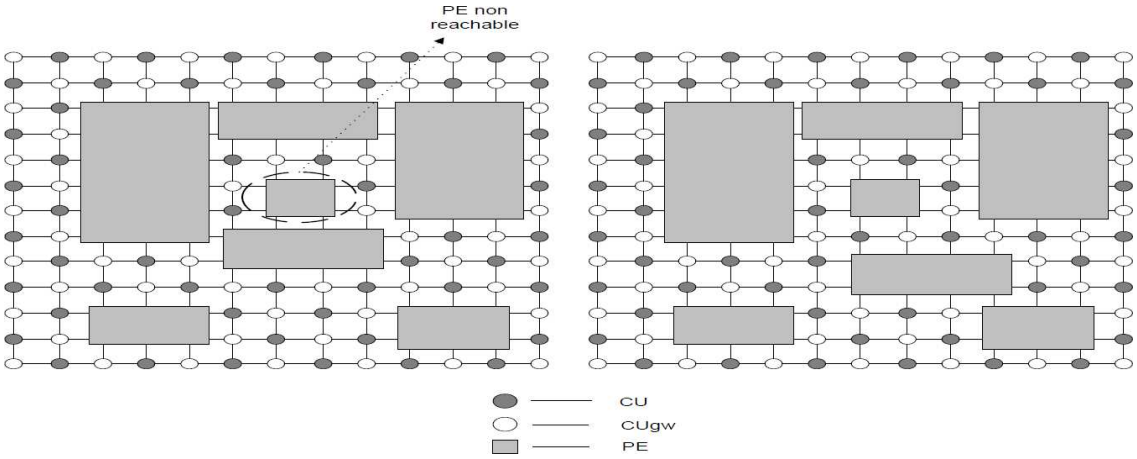


Figure 2.9: C-U-NoC [JTBW09].

shows dynamically placed modules and placement rules in C-U-NoC.

The Q-NoC project is an optimization of the C-U-NoC. It provides more powerful routing architecture. The *Q* is just the optimization of *C-U* which can also be pronounced [Q]. When the C-U-NoC provides a central buffer and a unique routing logic, the Q-NoC provides for each direction a separate buffer and a separate routing, making logic blocks more reactive and greatly improving the bandwidth.

Those projects are centered on the routing and rerouting capacity of NoC by thinking about C-U-Switches and Q-switches. Together with the ideas of Dy-NoC, its idea of reconfiguration shows that a NoC can also be partially reconfigured. Moreover the reconfiguration is here possible with dynamically placed module on the array.

2.4 Project Observations

Considering all those examples and the main requirements for designing a microelectronic self-organized system, it can be observed that the global environment where reconfigurable technology should be deployed is not at the center of the system specification and consequently, the inherent interactions are not really studied. The interoperability is a central question because the possibility to push reconfigurable computing to its limit depends on its association with other fields of research. In addition to these projects, our research tries to include this particularity into the design of reconfigurable systems.

Conclusion

Within this description, the RSS concept of will be proposed in addition to some experimental tests that demonstrate the relation between all notions that was presented earlier and in particular the triple: Autonomy, Dynamic and Interoperability properties.

To complete this chapter, a chronological view of reconfigurable computing is proposed (see figures 5.10 and 5.11 in annex 1). It shows links to other questionable aspects and the historical links between the concept, the technology and the integration of reconfigurable computing.

Chapter 3

The Reconfigurable Self-Organized System Concept: a Networked System

Introduction and Overview of the Problem

Classically, a networked system is organized around nodes based on a central GPP architecture running under an *Operating System* (OS). The main objective of this OS consists of controlling the hardware resources and scheduling the computation of tasks over time. Generally, in this type of controlled architecture, the entire knowledge and information are concentrated at the highest abstraction level. The decisions are diffused by a top to bottom approach. The lowest level is represented by modules that constitute the system. On the other hand, in a decentralized control structure where the decision making is realised by each node, information and knowledge go around in the inverse sense, i.e. from the bottom to the top of the organization. These two notions are dual and complete each other, one at the node level, the other at the network level. These two types of centralized and decentralized system are illustrated in figure 3.1.

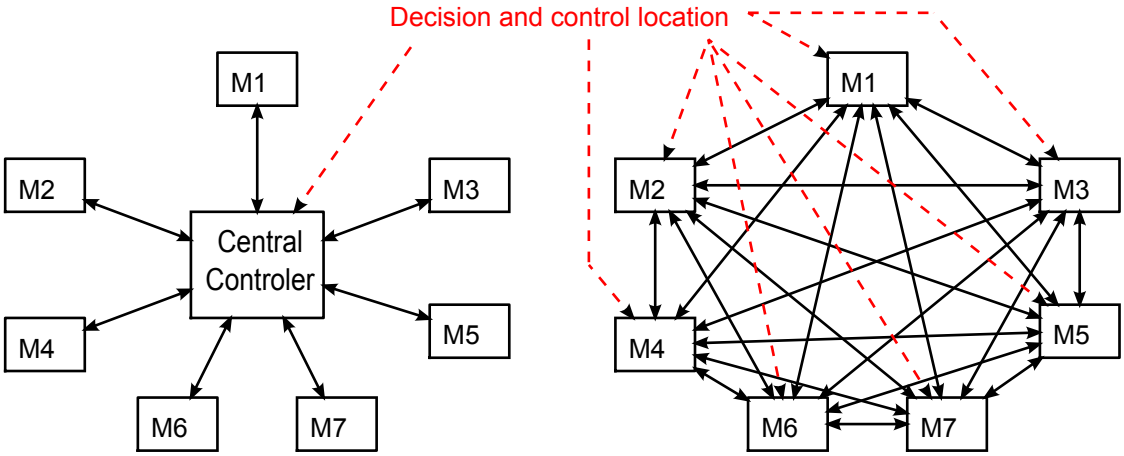


Figure 3.1: Control examples: Centralized and decentralized.

For a Reconfigurable Self-organized System, the number of tasks that can be simultaneously executed depend on the available resources within this network. Because the size of the Reconfigurable Self-organized System is variable in time and in space, the maximum number of simultaneous tasks is dynamic and depends on the number of Reconfigurable Self-organized Nodes. In this context, the scheduling problems are different and more complex than in a classical system where a centralized control is required. Indeed, the strong variability of the environment of these systems makes that the control system does not operate in an optimized and efficient way. Centralized systems present bad performance in terms of reactivity, flexibility, sturdiness and reconfigurability, because they are based on a rigid and not so adaptive

structure. Nevertheless, centralized systems present good performance in terms of productivity which is essentially due to internal optimization. In terms of sturdiness, if the central control system detects a failure during its working time, the entire system becomes inoperational. This is the main reason for traditional design approaches based on centralized systems to get a new decentralized orientation. In this decentralized orientation, the global management is distributed in all constituting nodes of the system. Their interactions allow the emergence of a global control. Table 3.1 summarizes these two integration cases and show the main differences between them.

	control centralized	**control decentralized**
architecture	rugged and static	flexible, programmable and dynamics
relation	system↔module	module↔module
approach	*top-down*	*bottom-up*
communication	one to many	many to many
response to perturbations	low	high

Table 3.1: Comparison of two types of control.

Within systems using a decentralized control management, multi-agent systems have a special place. A lot of different examples of these systems are available. These systems are mainly based on a GPP architecture and software solutions. Few examples exist of decentralized control systems based on dynamically reconfigurable hardware structures like FPGA. In general, FPGA based systems are mainly based on static control structures. To address the requirements imposed by the self-organization assumption for reconfigurable systems, a new architectural approach based on decentralized control should be conceptualized and developed. Nevertheless, an efficient management for reconfigurable network requires some specific services. Among the most important are the task scheduling and distributive services that ensure a correct execution of all application tasks on the resources of the network. To solve this problem, we propose a model of Self-organizable network systems used for on-line task scheduling and distribution on reconfigurable system-on-chip architectures.

The concept of *Reconfigurable Self-organized System* (RSS) is presented here. It is the combination of the reconfigurable technology and the peer-to-peer concept. It is used to contribute and to integrate *Reconfigurable Computing* into the world wide technological environment. *RSS* studies reconfigurable computing at two different levels, the node level and the network level.

This chapter is composed of four sections and it is mainly focused on network and commu-

nication details. The first one presents the origin of the RSS concept. The second one describes the concept itself. After that, properties and limitations are discussed and finally an analogy is made to compare the concept with a simple idea view.

3.1 Concept Development

The RSS concept comes from the study of networked system concept together with reconfigurable computing concepts and technologies and with the observation of related research projects.

3.1.1 The Local Market Global Symbiosis Concept

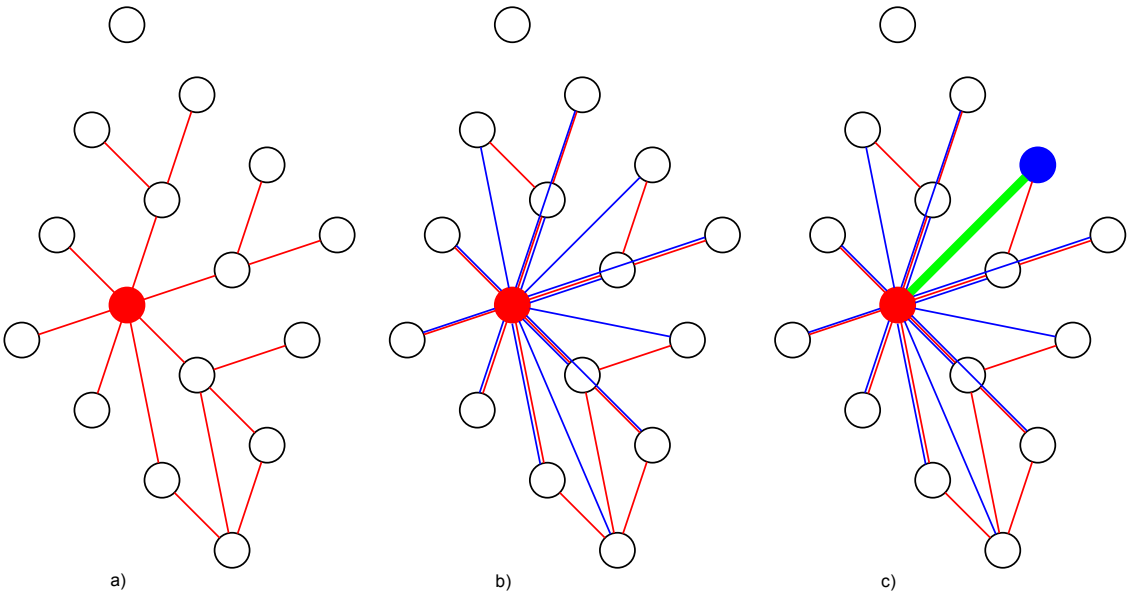


Figure 3.2: Issue a request (a), Retrieve answers (b) and Commission job (c). [BCM+09].

A simple concept was defined to deploy new features: *Local Marketplaces Global Symbiosis* (LMGS). The concept is distributed according to principles of supply and demand within the network (see figure 3.2).

The simple idea is that every device does exactly what it deems to be the best according to its stored parameters. The minimal software to actively take part in the process consists of two elements: a customer which issues requests to the network to have a certain job done and a purveyor that answers requests for jobs with the costs it charged if it would be commissioned.

The effort a device makes to create the answer can vary widely. Depending on its computation power, knowledge, and storage capacity, this ranges from a simple return of standard values to a complex measuring where the load, the utilization of the node's components, or the probability for the effectiveness of anticipated reconfiguration might be taken into account. For example, in sensor networks, communication is the most expensive action. A distribution of work should be chosen to minimize the overall traffic, maybe by executing a lot of tasks locally through to reconfiguration of the node.

Request

The requests issued by the customer component of a node consist of a tuple as follows:

$$Request = \{Source, Target, Requester, DataVolume, Task, MaxHops\} \tag{3.1}$$

where Source contains the data source for the task, Target the data sinks, and Requester the device which posted the inquiry. Data Volume holds the quantum of data to be processed with Task. Max Hops specifies the maximum number of times a request may be relayed by a node's neighbors.

Source, Target, and Requester may be partly or completely identical, we included mechanisms to distribute jobs within a channel between data source and target. The values can be one-to-one identifiers of nodes or might as well contain wild-cards to address groups of nodes so that, for example, modifications that have to be applied to a set of devices can be launched by a single command.

The Data Volume will be taken into account when an offer is generated for the request. It usually influences the decision whether it is more appropriate to solve a task in software or in hardware and if the data may be processed remotely or if the communication costs for that would be too high.

To be able to specify a task or a whole group of tasks, we suggest a hierarchical mechanism nomenclature which comprises every single service that can be rendered in the network under one root node.

As Task in the request structure is not limited to one atomic job, it may be a list of tasks. These lists may contain jobs that are to be processed sequentially or in parallel ranging from a single data-source to a single target, to complex data flows with multiple sources and targets.

Finally the restriction to Max Hops ensures that inquiries are not simply flooded through

the net but stay local working off jobs at a kind of in situ marketplace when sensible. This is of course only if the local neighbors provide appropriate solutions to tasks at a reasonable price. The composition of this "cost" will be presented in Section 4.2. The idea behind that is obvious: since we are able to reconfigure devices to serve virtually any need, even small localized groups are highly adaptable and will be able to cope with most challenges with optimal efficiency. Thus, data will in general be kept in a spatially narrow cloud, so that communication is minimized. In the current and the next section we will explain how this is reconciled with the claim of super-regional cooperation.

To publish and find services to fill in a valid request basically three mechanisms can be deployed. First one central directory server knows which device offers which services and has to be prompted for every job to work off. If it fails, the whole network is paralyzed. New services have to be entered, causing additional traffic.

Second, searches for certain services are flooded through the whole network. This is very flexible, but rather inefficient.

In this work, we have developed a third approach, a hybrid solution between the two previously mentioned ones. Here data is flooded only within the closest neighborhood. Only if the answers from them are insufficient, say because none of the next nodes wants to execute the requested task, the job is advertised again, this time with a higher number of maximum relays. Additionally devices keep a more or less extensive list of other remote hosts, to satisfy a certain request. In this manner not only local offers but also more distant ones will be taken into account. Distant ones possibly suit the current situation better. The maintenance of this list is closely related to the structure of offers replied to a request which will be covered in Section 4.2.

Generally, requests will be made to the direct neighbors and to the issuer itself. A neighbor that finds the Max Hops greater than one reduces that counter and sends the request to all its neighbours. Especially the answer from the node itself is interesting in this regard: with the possibility to reconfigure, scenarios can be managed where communication cost is very high and nodes have to cut back on transporting lots of data through the net.

Offers

The response to a concrete query contains two elements: the *cost-vector* that the replying device is estimating for supplying the service and the local *list of known providers* that are also capable of satisfying this particular request:

$$Offer = \{Costvector's, ListOfKnownProviders\} \tag{3.2}$$

Costs mean figures given as multiples of a base cost. For example, the transmission of one byte of data over a wireless Bluetooth connection will be a lot more expensive than over a wired Ethernet link. We identified three dimensions of costly actions: *time complexity*, *energy consumption*, and *space usage*. Thus, a device's purveyor calculates the cost vector for a task A according to:

$$C_A = (Z_K E_K P_K)^t . W \tag{3.3}$$

where Z_K denotes the cost for the local effort concerning time, E_K for energy, and P_K for the required space. W contains the willingness of the device to spend part of the specific resource to locally execute task A as a diagonal matrix. A battery driven device might, for example, want to lower its willingness to accept very energy consuming jobs as it runs low on battery power. The final cost-vector C_A is passed to the issuer as part of the offer. The list of additional service providers is being built up through logging of messages indicating the commission of a node for a particular task or through deliberate writing. On the one hand, devices that relay an accept message (basically a request with a specially formulated task) to a device store the target device and task together with an expiration time. On the other hand devices can advertise their capabilities by giving out a request to every other participant of the net to amend its local list of providers. If it intends to stay in the community for a longer period, the expiration time may be set accordingly, attracting all sorts of requesters, locally and remote.

Negotiation Example

The flow of a negotiation basis on sending requests, retrieving responses, determining the most appropriate solution, and commissioning a purveyor. When evaluating the replies, the contained list of alternatives, maybe remote, providers may be taken into account and the selected ones may be prompted for a bid. This enables the system to incorporate both: decentralized and distributed computing as well as central services like the storage of gathered and processed data. When answers came back in or after an amount of time, the customer determines the optimal partner to commission and it allows the device to emphasize a rather fast, energy-efficient or space preserving execution of a task. Depending on the computation-power

and -willingness the node might accept the earliest offer or may run a multi goal optimization on the large data retrieved to find the best trade-off.

The *Local Market Global Symbioses* (LMGS) Concept was presented in [BCM+09]. It proposes a concept of networked system where each node is placed to find tasks that must be processed by the nodes group. It can be summarized as a network where nodes are always trying to find a new job.

3.1.2 The Local Intelligence Global Symbiosis Concept

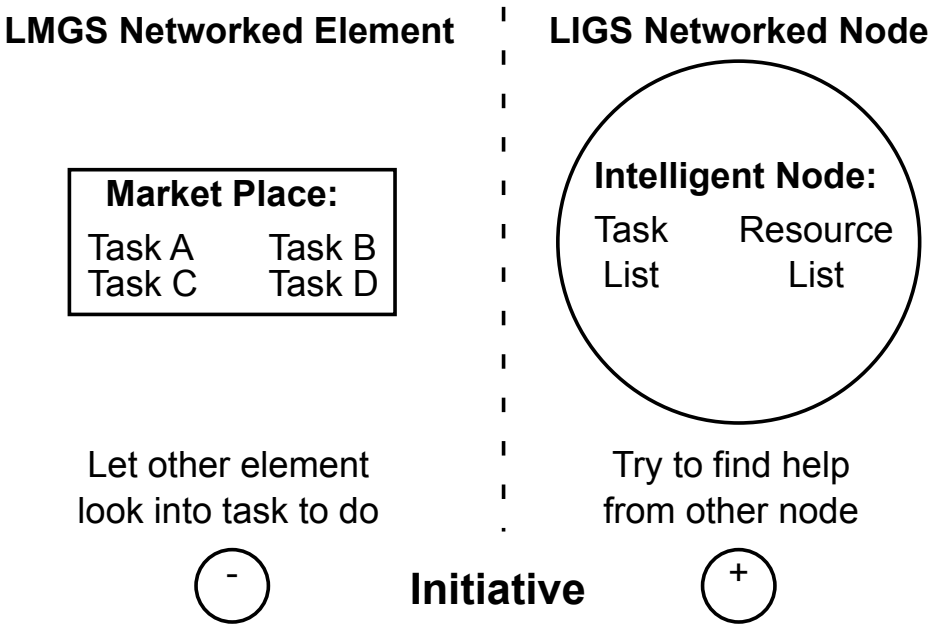


Figure 3.3: LMGS element versus LIGS Node

The first evolution of the LMGS concept is the *Local Intelligence Global Symbiosis* (LIGS). The LIGS proposes to look at the LMGS from another point of view where resource availability is at the center of the element collaboration instead of data quantity for the LMGS.

In other words, it proposes to reconsider the LMGS to see it as a network where each node is not looking for other tasks but can only ask for computation resources. Consequently, nodes have the possibility to reply or not to reply to any resource requests. The main advantage is that the communications are completely decentralized. Indeed, the LMGS concept is limited to a dynamic set of a small central task market place and the LIGS considers each node of the network as a fully communicative entity.

Figure 3.3 shows the LMGS element and the LIGS node, side by side. More precisely, the LMGS element is presented as a market place where tasks are available. The LIGS node is presented as a node able to manage tasks and resources together. Figure 3.3 also depicts the advantage of the LIGS which is to give the initiative to the nodes.

3.2 Reconfigurable Self-Organised System: A P2P Reconfigurable Technology

3.2.1 Definition

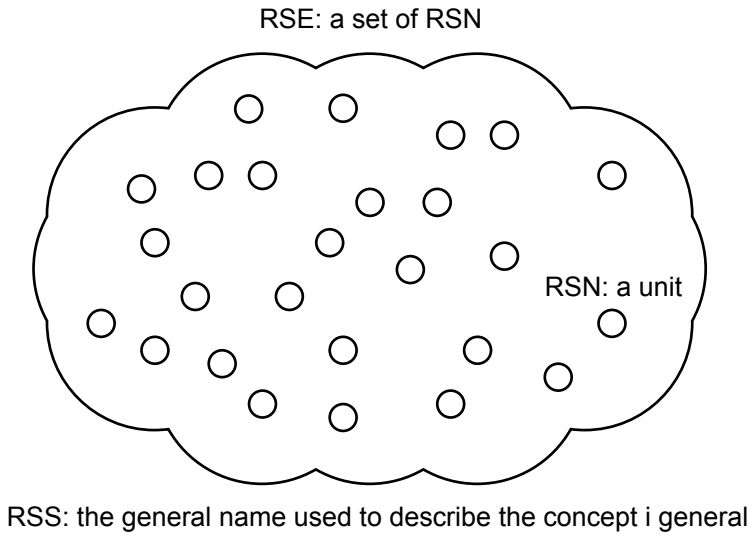


Figure 3.4: How to use the concept's terms.

Figure 3.4 shows a representation of the terms used to describe the proposed concept.

*A **RSS** is a modular and complex system based on reconfigurable hardware technology which is able to organize and restructure its own activity by interactions of its intelligent nodes and without external influences.*

Beside this definition, we also define:

– *Reconfigurable Self-Organized Node* (RSN)

– *Reconfigurable Self-Organized Entity* (RSE)

RSN and RSE are used to discuss the intelligent networked system based on the RSS concept, where RSN describes the intelligent nodes that are part of the RSE.

***RSN** is the base unit of RSS. They are intelligent nodes in the sense that they are able to communicate with other RSNs without being externally managed. Their main characteristics are dynamic behavior, autonomous capacity and interoperability with systems based on other electronic technologies.*

***RSE** is used to qualify how an RSN group interact and compute tasks generated by the users. By extension of the self-organization and emergence concept, RSEs can be distinguished from a simple RSN group by the emergence of properties developed due to the RSN's interactions.*

3.2.2 Environment and Application Field

RSS are thought and designed to be part of the actual networked infrastructure where nodes and any other system can communicate, exchange and work together. The Internet structure is good example of this kind of environment.

Beside the technological point of view, a parallel can be drawn to the standardization of communication protocols. In order to allow a communicative link between nodes, it is crucial to use the same protocol.

3.2.3 Computing Model

The computing model is based on the concept of reconfigurable computing used together with a peer-to-peer communication structure. The goal is to combine the advantages of auto reconfiguration provided to local systems with those of a completely decentralized communication system where each node or RSN is dynamic, autonomous and interoperable by definition.

Using this model, the problem is turned into a file management problem and a scheduling processing process. These problems have a lot of different solutions that are proposed in the research work on operating system and file systems. Under this description the computation model can be described as follows:

$$F(T_x) = f_1 \circ f_2 \circ f_3 \circ ... \circ f_n(T_x) = R_x \tag{3.4}$$

where, $\exists\, n \,\epsilon\, \mathbb{N}$, F is a functions applied to the task T_x (x denotes the task number). F can be decomposed into a set of f sub-functions. These sub-functions are applied sequentially to T_x. Finally R_x is the result.

In the computation model, f can be processed on any RSN of the RSE. RSNs are characterized by their existence in the time and in the space.

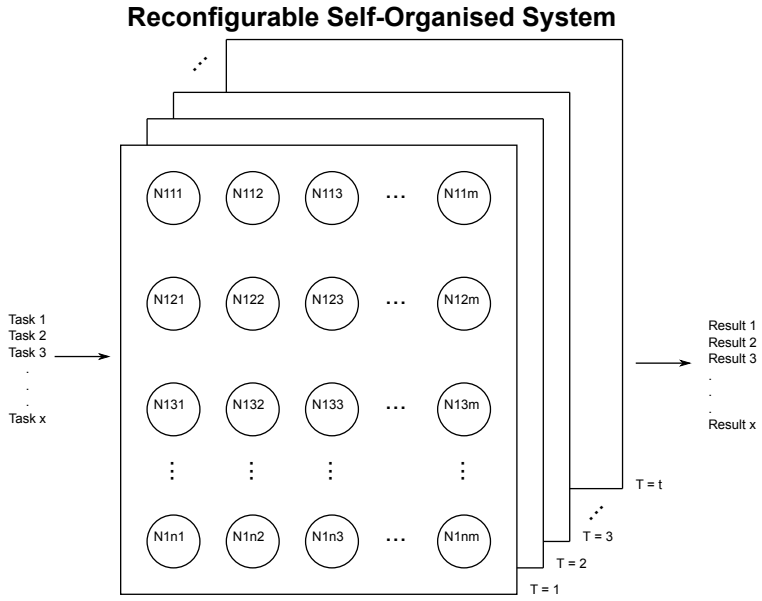


Figure 3.5: RSS Computation Model.

Figure 3.5 shows the organization of a RSE on a two-dimensional grid over the timesapce. RSN are symbolized by N_{tnm} where t is the instant time and n, m are used to give a position on the grid. Considering RSNs existence, the account of available resource can be defined as follows:

$$Re(t) = \int N_{tnm} dt \tag{3.5}$$

where $Re(t)$ represents the quantity of resources at a certain moment.

3.2.4 Simplified modelling and formalisation of a RSS

An RSS that integrates the general aspects presented above is illustrated by figure 3.6.

In a more simple manner, such a systems will:

– Adapt their behavior to any disturbance,

– Distribute, in case of a node failure, remaining functions or tasks to other nodes,

– Replace their modules using auto-reconfiguration,

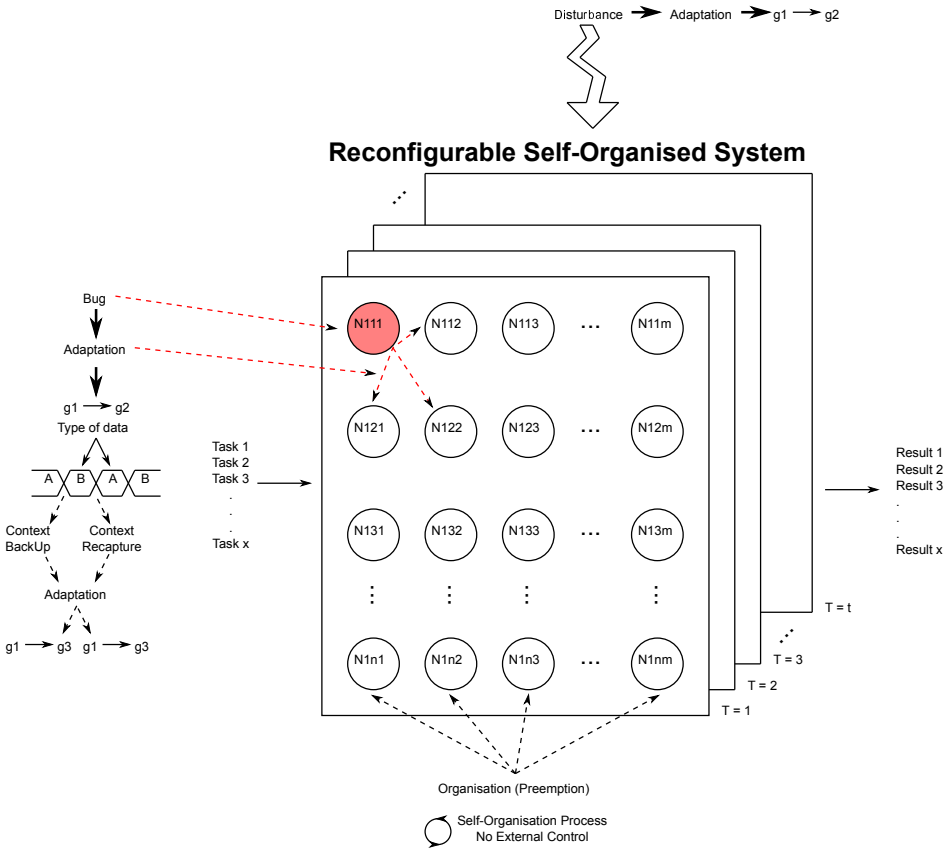


Figure 3.6: Self-Organization Criterion of a Reconfigurable System

– Detect the modification of input data and adapt their global behavior through the possible functions of the nodes for new computations,
– Organize in an autonomous way their nodes to make them ready to preempt future processing tasks.

By considering the *reconfigurability* of a system as a hardware architecture, the systems will be able to change its form or function during various processes over time, and the term *dynamic* that is attributed to reconfigurability as a characteristic of a system able to change its state, the definition of a dynamically reconfigurable system is reached. Moreover, due the fact that the dynamic reconfiguration of a system represents the most promising aspect of the realization of the self-organization principle, the explicit relations between that principle and the reconfigurable technology should be established.

A new cooperative intelligent hardware design concept making a system self-manage is proposed. The basic concept of this design is based on dynamic structures called *The Stream* that presents the main mechanisms of our self-organized design approach.

Let us suppose that there is a networked system *S* that is composed of *N* modules E_i ($1 \leq i \leq N$), that can communicate and exchange data. Each module E_i carries out a certain function or task $e_i(t)$ at a given time t. Each module's function presents a part of a set of the available functions (or services) S_i that could be accomplished by the given module and contributes to achieve a global function $g(t)$ such as $g(t) \in S$, where S represents the set of functions which can be implemented by the considered self-organized network nodes ($g(t) \in S$, $S = \{\emptyset, g_1, g_2, ..., g_{n_g}\}$). Defined in this way, the system can be described by the following set of expressions [Jov09]:

$$e_i(t) \in S_i,\ 1 \leq i \leq n,\ n \in N. \tag{3.6}$$

$$g(t) = \sum_{i,j=1}^{N} e_{ij}(t), 0 < i, j < N, n \in N. \tag{3.7}$$

where each task e_i can be localised and places on each reconfigurable RSN.

3.3 Using Reconfigurable Hardware in a Self-Organization context

The self-organized auto-reconfigurable system comes from the association of self organization and auto-reconfiguration concepts [MMDB07, JTW08]. To realize this concept, a real multi-FPGA system that is self-organized and composed of intelligent networked nodes is implemented. In this network, the nodes are able to choose their own tasks, depending on the resource requests sent by other nodes. These reconfigurable nodes ensure a certain flexibility and adaptability to the distributed system. Two nodes are also able to communicate directly to exchange data. In practice, each FPGA represents one self-organized node of the network. It allows to implement a complex embedded architecture that consists of processor cores, hardware blocks (i.e. *IP* blocks), dynamically reconfigurable area units (*DRU*) and Network on Chip (*NoC*) in the same circuit. Each node associated with memory (external and/or embedded) containing local bitstream configurations, is able to support dynamic applications and to ensure the execution of several tasks within the same resource. Furthermore, these nodes are associated with sensors to collect data and at least one data sink that gathers all the information and offers an interface to other nodes. Figure 3.7 shows an example of our self-organized networked architecture. It is composed of self-reconfigurable nodes.

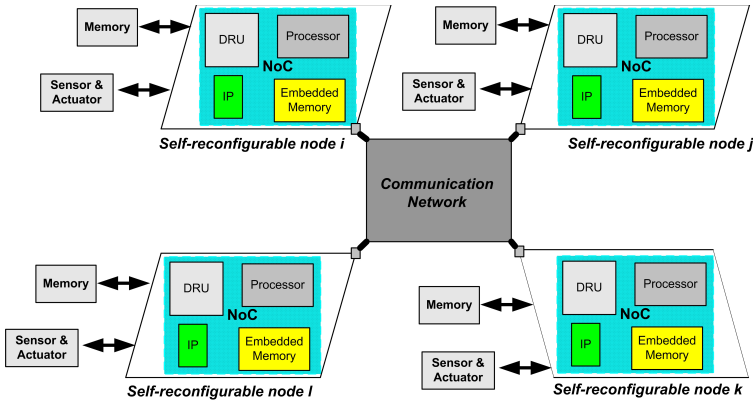


Figure 3.7: Architecture of the networked Self-reconfigurable nodes.

Our networked multi-FPGA system is based on an automatic task distribution by using a Stream and allowing each FPGA unit to open direct communications with any other FPGA unit with the objective of exchanging data. By exchanging data and the associated partial bitstream file, our self-organized system realizes the relocation of computation tasks with this objective to create a entity that can morph in real-time.

3.3.1 Task distribution Stream Concept

To ensure an efficient execution of the application, the self-organized network must manage tasks dynamically using the support offered by auto-reconfigurable nodes. Due to their nature, auto-reconfigurable nodes are limited not by the number of tasks they can process sequentially but by their reconfigurable regions that contain processing logic. These processing logic resources and regions must be configured at first, in order to realize a computation function. In this case, a configuration step is necessary before the beginning the processing. It can be a dynamic reconfiguration or an auto-reconfiguration. This technology allows the adaptation of the system using partial reconfiguration that does not disturb the rest of the design. Moreover, RSE must be able to perform multi-task and allow the dynamic management of several actions simultaneously.

Resource Requests are limited to information about the task itself and exclude completely the knowledge about the receivers. In this context, the awareness of the neighborhood nodes is emergent and comes from the potential replies.

For any reply received, the requester automatically discovers the existence of another node. Moreover, the requester node can start teamwork with the node that it knows about.

To answer these problems, the proposed cooperative intelligent approach is based on a stream that travels from node to node and which contains all information necessary: the task identification numbers and the names of the different files. This information allows self-organized nodes to decide by themselves if they want to help and how to establish direct communication links with the requesters. The direct communication links are established to exchange bitstreams files and data files. Moreover, the stream can be updated by all nodes that want to request help from other nodes. Figure 3.8 illustrates the mechanism of the self-organization based on the proposed stream concept. It shows the task distribution and the point-to-point dataflows used to process organized tasks over the networked entity. This is part of the main originality of this thesis work.

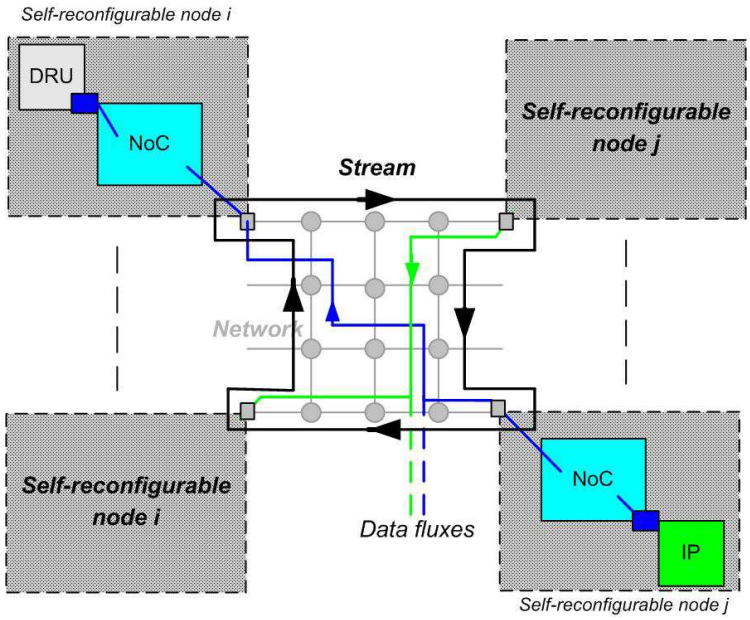


Figure 3.8: Task Request Stream and temporary Data or bitstream fluxes.

The main objective of this stream is to gather information about tasks that must be processed by nodes and by extension the networked entity. For example, if $node_i$ receives a request from $node_j$ and if $node_i$ has a free reconfigurable region, it can reply to $node_j$ by asking for the files associated with the concerned task. In other words, a resource request is followed by a file request.

It is important to consider that the stream is not a control structure. It does not impose any decision on the networked nodes. Its main role is to gather and inform requests. The stream allows a self-organized task distribution. It can be used without being completely aware about

the entire entity. Also, it can be realized using an underlying protocol, for example TCP/IP, and it is composed of two levels. The first level is the request stream and second level is the point-to-point dataflows.

The request level is made to connect nodes together but without any central control and with a minimal protocol. The data file level is made to exchange files in a reliable way. These two kinds of links can be designed using packet based protocols and technologies. Any communication standard can ensure additional quality of service for those communications. Figure 3.8 illustrates the different communication types. The request stream that reaches all nodes of the network is shown in black. The temporary fluxes created to transfer data or bitstreams between nodes are shown in blue and green.

In summary, when a node accepts to process a task, it asks for data and bitstream to reconfigure its *DRU*. For instance, a $node_i$ chooses to execute a task specified in the stream. It sends back a file request to exchange data and bitstream with $node_j$. Furthermore, the concept of resource and file request is completely dependent of the node's intelligence that constitutes a distributed intelligence and a non-centralized hardware control required in the design of a self-organized system. Indeed, by giving to each node the possibility to handle their own jobs, it is possible to create self-organization properties without any kind of node. This concept is particularly adapted to auto-reconfiguration mechanism of DRU (see chapter 5). Figure 3.9 depicts the flow control graph of the self-organization mechanism based on dynamically reconfigurable nodes and which produces a cooperative intelligent mechanism. In this graph, two mains direction can be followed: the allocation of a new task or the processing of a task.

In the case of a task's allocation (on the left side), it is first verified if available resources exist that can be shared. Therefore, the data file and the associated bitstream file are present. These verifications imply the discovery of the location of the task. If the files are not detected locally, it means that requests must be transmitted in order to exchange those files.

It corresponds with the second part of the flow (on the right side), where the node is waiting for the arrival of file. When the bitstream file is received, the node can reconfigure itself. After that, when the data file is received, it can be processed on the newly configured module.

The task distribution stream is the most important part of the self-organized reconfigurable entity. It is a self-managed system that unifies all nodes by forming a computing group. All information necessary to make a decision and communicate are provided to any node. Its main goal is to provide information in real time and create on emergent organisation. Secondly, the stream is designed to manage a scalable number of tasks according to the resources workload.

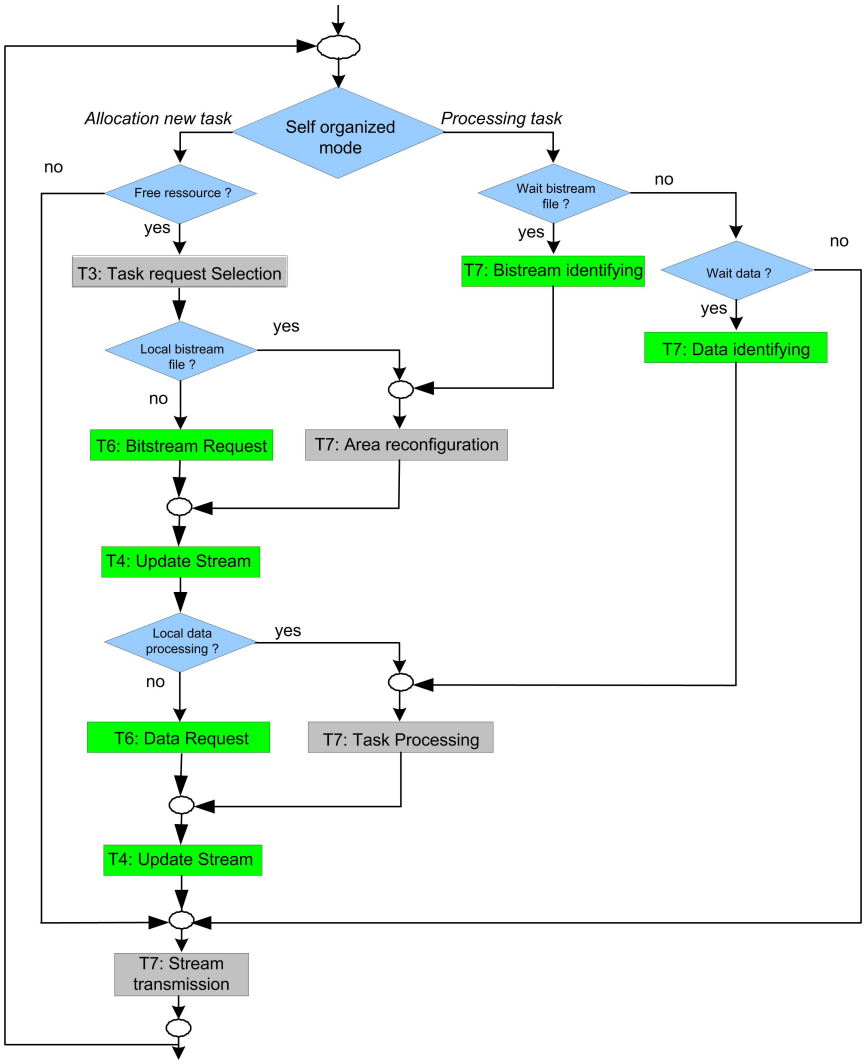


Figure 3.9: Control flow graph of one self-organized node.

3.3.2 Theoretical Stream Structure

Figure 3.10 presents the stream's frame used as information structure. It is composed of the following fields that correspond to three different kinds of requests:

- The number of tasks in the stream that have to be processed by the group.
- The data fields used to define the type and the status of each requests (*Ctrl*). This request field is used to identify the bitstream configuration file or the data themselves:
  - Task request,
  - Bitstream Request,
  - Data Request

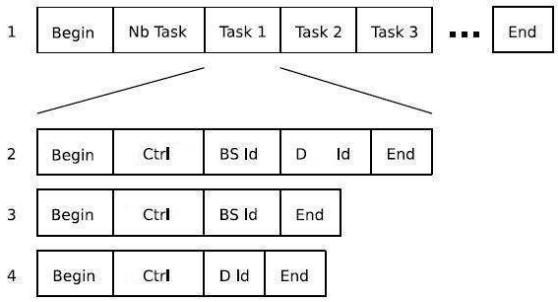


Figure 3.10: Request frames.

- The Bitstream Identifier (*BS_Id*) field indicates which bitstream has to be used to process data.
- The Data Identifier (*D_Id*) field indicates which data has to be processed.

As defined, there are three different types of request: task request, bitstream request and data request. For all of them it is not necessary to know the destination. It is a particularity of the concept. A node sends a request without any information about the entity. Each task has to be identified by the Control Bit, the *BS_Id* and the *D_Id*. This distribution stream achieves a cooperative intelligence allowing the census report, the management and the localisation of the tasks in each node of the system through a particular communicative mode. It concentrates the minimal information needed for a self-organizable and auto-reconfigurable node to collaborate with other nodes.

Point-to-point dataflows Structure - Data Exchange Communication

When a node accepts a task from the stream, all nodes are informed through the stream itself. It engages a file transfer with the requester. The goal is to get the configuration file necessary to implement the hardware function that is necessary to process data in a local node. Fluxes are created for short moments in order to transfer those files.

The point-to-point dataflows use a transfer protocol based on data packets for the stream and the data. The messages sent using the data fluxes are composed of a fixed number of packet Np ($Np \geq 1$).

The packets are composed by two fields. The first one is the *header* that contains the destination address and the packet's length. The second field is the *payload*, it contains data that are corresponding to a fraction of the files that are transmitted. Figure 3.11 describes the packet frames used in the fluxes. Therefore, if the appropriate bitstream is not already in the

memory of the local node that ensures a task computation, a new communication is established to download it. This strategy is important because it is not possible to store all bitstreams directly in the task distribution stream. Moreover, this approach has another advantage because each bitstream can be duplicated in a second unit. In this way, if one unit breaks down, the global system does not lose the considered bitstream thanks to this redundancy. This concept is used to protect the information and moreover for the security of the bitstreams.

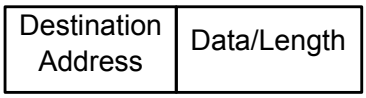


Figure 3.11: Packet format of a data flux.

The data exchange is established when a node accepts a task. In this case, this unit can ask to have the data which has to be processed after the self-reconfiguration.

3.3.3 Generation of bitstream and data request

Each global function F is composed of many sub-functions or sub-tasks: h, g, f. Those sub-functions are a pair of identifiers (*BS_Id*, *D_ID*) used to associate data and bitstreams.

$$F = h \circ g \circ f(\textit{BS_Id}, \textit{D_Id}) \tag{3.8}$$

It is the minimum information to transmit in order to organize tasks without any knowledge of the entity. Using this information, each node is able to configure itself with the best hardware functions.

Any node can send a request on the stream and when another node receives it, it can select a task in it, remove it from the stream, and send it back again. The stream can be seen as a list of request:

$$S = \{R_{T1}, R_{T2}, R_{BS5}, R_{D2}, ...\} \tag{3.9}$$

with S the stream, R_{T1} the request Task1, R_{T2} the request Task2, R_{BS5} the request for the Bitstream file 5, R_{D2} the request for the Data file 2. In this stream, each node can add or remove a request to participate in the entity's activities.

If a node needs to distribute work, it can send a request on the task distribution stream. Therefore, when another node accepts a request, it writes into the control field and generates two requests: a bitstream and a data request. For these two requests, the node uses bitstream and

data identifiers as its own identifier. For the auto-reconfiguration, the node takes the identity of the bitstream. Consequently, the bitstream request integrates (at the same time) the identifier of the bitstream and the address of the destination.

Examples:

1. Bitstream request: (four control bits, *BS_id* or node Id), the node takes the identifier of the data.
2. Data request: (four control bits, *D_Id* or node Id)

In this way, when the stream is read by the entity, each node can check the combination of identifiers. If there is a match between bitstream, data request and node identifier, the communication can be temporally established.

3.3.4 Social Organisation

The social organization of RSN is centered on the node intelligence. By giving enough capacity to manage their computation tasks and their communications, RSNs are free to evolve in the RSE and to be a part of any computation task. With this description each node has the same kind of functionality.

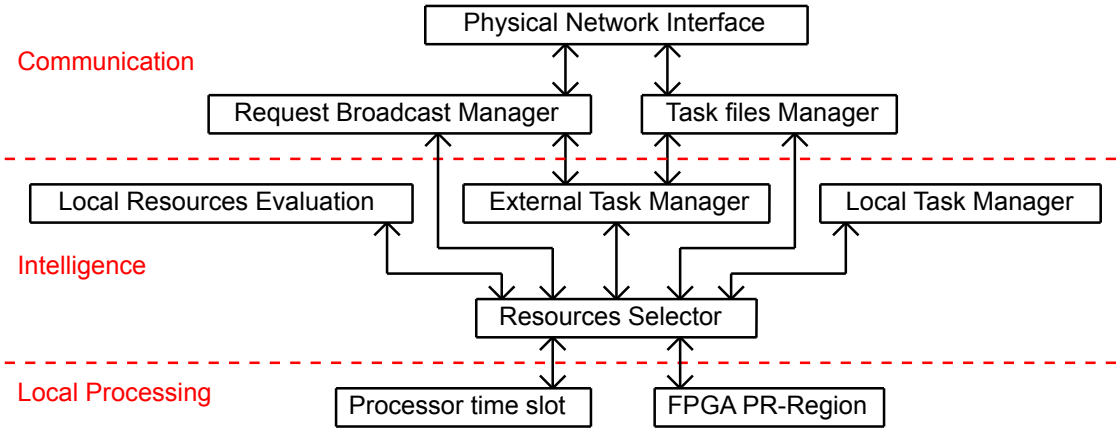


Figure 3.12: Node Functionality that constitute the basis of RSN.

Figure 3.12 shows the functionality that allow us to create a close relation between communication and local processing resources. By extension, it is the basis for the social organization of RSE. Based on the kind of structure, any RSN can cooperate or ask for cooperation from any other RSN. Within this communication point of view, all RSNs are designed to be fully autonomous and have the possibility of creating processing groups to solve tasks.

This description shows how a networked node can react depending on requests to organize itself. Depending on its own activity and available PR Regions, the node can decide to contribute to the execution of common processing tasks and reconfigure itself to compute data. This kind of description gives a certain autonomy to nodes and lets them decide what they can or cannot do. In other words, the control or management of networked tasks can be spread and then all the system can be seen as a decentralized control system.

Tasks dependencies add the notion of morphing the RSE structures. To solution tasks sequences, RSN are spontaneously forming little groups. These groups are formed to share resources in a more efficient way. In this complicated entity, the notion of self-organization and emergence need to be correctly determined to understand phenomena that are appearing. The parallelization of tasks is one of the main benefit of this concept.

3.4 RSS Properties and Limitations

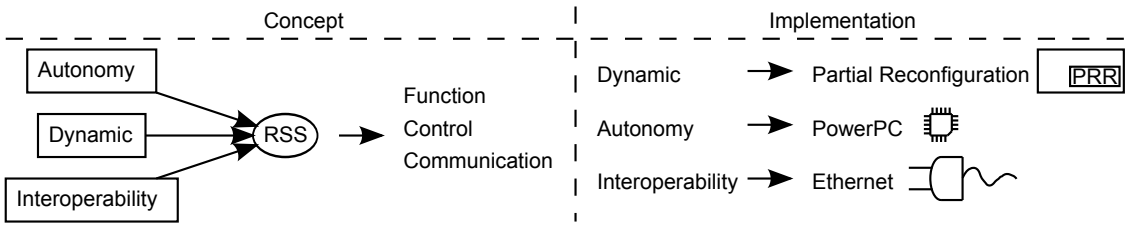


Figure 3.13: Three interdependent properties in a double point of view.

At this point, two remarks are necessary to include properties coming from chapter 2 into the concept presented here. The first remark is about the local behavior of a network system. At the local level, each node needs to provide server and client behavior at the same time. It is a well-known concept that takes into consideration the fact that a user provides intelligence to allow a RSN be autonomous and dynamic. This introduces intelligence into system This concept has evolved and can be seen as including also the notion of dynamic autonomy which is important for the self-organization and emergence properties of those projects.

The second remark is about collaboration and intelligent cooperative aspects. These aspects are as important as the node description. Properties of dynamics and autonomy can be completed with interoperability which can be summarized into a good wheel cycle.

Figure 3.13 shows the integration of the three key properties into this work. It is evident that there is a duality of the macro and the micro levels. Conceptualization can be applied to

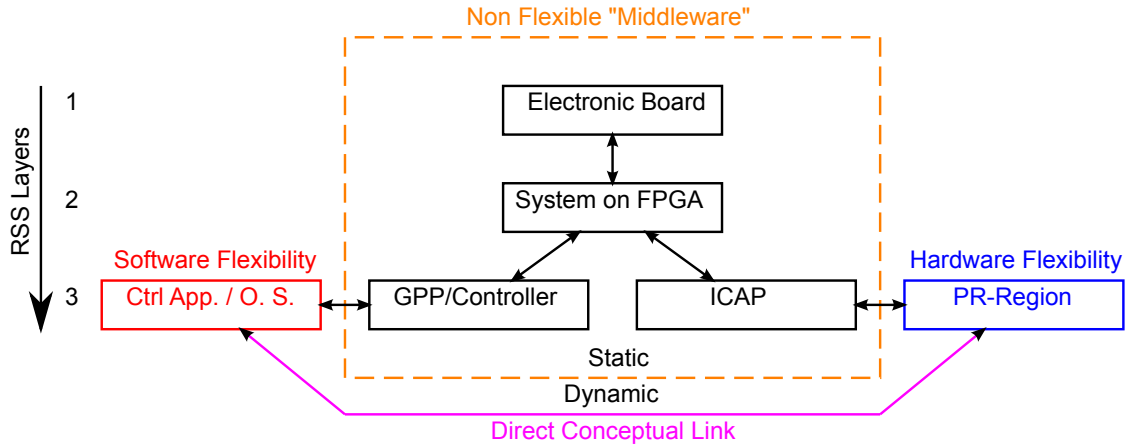


Figure 3.14: The dependency stack of technology.

both levels and it places the problem in perspective with the will to illuminate the complexity of their relation.

3.4.1 Autonomy at Local Level

The proposed concept attemps to respond to the three characteristics that describe what is necessary in a networked system to integrate new features and evolution capacity. For the autonomy at the local level of each RSN, the possibility to communicate tasks is provided by making a link between the low level hardware and high level software.

Figure 3.14, shows the stack of technology that defines the structure of RSS concept. In these concepts, and in layer 4: Partial Reconfiguration Region and Software Application are directly linked together. Layer 1, 2 and 3 can be seen as the structure to unify them. In addition, the four layers together allow to integrate the triple properties set: Autonomy, Dynamic and interoperability. At the local level, the concept is making RSN intelligent.

3.4.2 Dynamic Organisation

Figure 3.15, shows a scenario of RSN collaboration during the time. As it is shown, RSNs can be grouped at a time $t1$ and grouped differently at time $t2$ or $t3$.

Concerning the dynamic property of the system, and more precisely the dynamic organization, this characteristic is reached by providing the possibility to reply freely and depending on available resources.

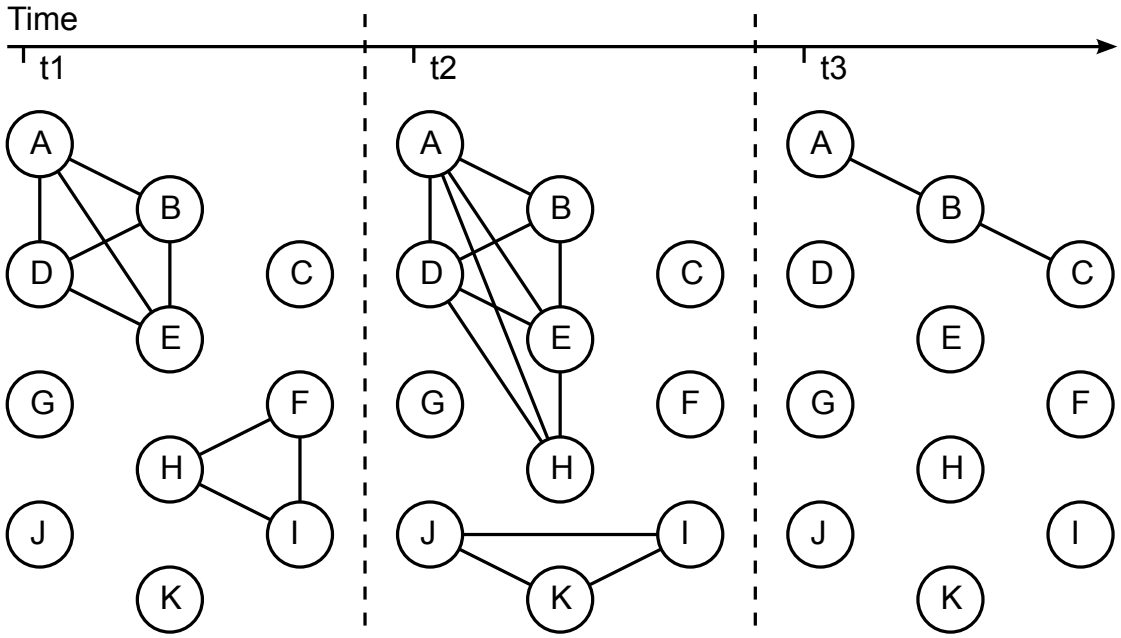


Figure 3.15: Dynamic cooperation evolution

Consequently, under a RSE, RSNs can produce an organization dynamically. Indeed, because of their ability to be independent, RSN can organize their jobs in a group spontaneously created.

This figure also shows a representation of self-organization and emergence. During the time, RSNs start collaboration in a self-organized way. It is also possible to see the locality of collaboration.

3.4.3 Interoperability with other technology and network

The two characteristics presented above lead to the third one related to the interoperability issue. This characteristic is provided by the possibility to use the software side of the application for other systems that do not include FPGA technology including reconfigurable technology.

By assuming the RSS as a combination of a control application on top of an existing electronic structure and *On chip* reconfigurable logic, it is possible to use this modularity to integrate RSE into an existing networked computation structure. This is due to the possibility given by actual main electronic architecture used to provide generalist computation machine that can compute any application. These structures are made to allow users to make them

evolve manually by adding new electronic devices to complete or replace old devices.

From these structures, the control application of RSS can be added as any other program. Consequently, computers can be seen as a half RSN without reconfigurable logic. Additionally, FPGA development boards including PCI-E interface can be also added into this computer to form a complete RSN made from a standard computer.

3.4.4 Other properties

The realization of a RSS leads to a list of expected properties. As explained before, they relate to the behavior at the local level but also at the behavior of the created RSE.

For a decade, the FPGA technology has been offering the possibility of building systems which are auto-reconfigurable. It is particularly obvious with the introduction of the partial reconfiguration technology that allows to reconfigure only a part of a FPGA at run-time. Using this advantage, a SoC can include any hardware accelerators that can fit into a predefined PR region. Today, auto-reconfigurable systems are starting to be integrated into networked systems.

The Five Self-Properties

Another set of properties exists that gives a good view of self-organization: the 5-self description. Considering the FPGA technology, the self-organization concept can be extended to a system where electronic hardware modules can be relocated. This makes it possible to add another degree of freedom to the management of resources. The concept of the self-stars described in [6] proposes a complete approach to implement self-organization properties. It is composed of five stars: Self-Reconfiguring, Self-Managing, Self-Optimizing, Self-Healing, Self-Protecting.

- *Self-reconfiguration.* The FPGA technology offers the possibility of building a system which is auto-reconfigurable. It is particularly obvious with the introduction of the partial reconfiguration technology that allows to reconfigure only a part of a FPGA at run-time. Using this advantage, a SoC can include any hardware accelerators that can fit into a predefined PR region.
- *Self-management.* The first improvement of the self reconfiguration stars is the self-management star. It explains how a networked system can organize itself by distributing tasks to the nodes that compose it. The management of partial-bitstreams is an example of self management. By exchanging partial bitstream with others systems, any system is able to physically evolve.

- *Self-optimization.* Software or hardware processing? This question is connected to load balancing methodology [2] and heterogeneity of the system. Most of the time, it is defined in advance because most systems are defined in a way that the processing is executed in software. In the proposed system, the definition of reconfiguration parameters is necessary to make a decision between software computation and hardware reconfiguration. In this context, it is the beginning of optimization.
- *Self-healing.* In the same way, the possibility for a self-organized system to heal itself allows to improve the working time of the system. Another scenario concerns the possibility to have a defective component in the system. Therefore, a multi-zone reconfiguration system could be a great advantage to re-implement a hardware accelerator on a viable partially reconfigurable region [7].
- *Self-protection.* The last star relates to the security of such complex systems. Considering the fact that there are a lot of faults in the security of common software system environment, the possibility to encrypt security components in a hardware module that is removable offers an advantage to such technology.

Starting from this point, the first self-star (Self-Reconfiguration) becomes a standard property of systems. This is due to the implementation that include reconfigurable technology and in particular the utilization of dynamic partial reconfiguration. Two degrees of reconfiguration are possible. Locally a node can adapt itself by reconfiguring hardware modules, and at the network level these modules can be exchanged by the nodes. The four others self-stars are properties that are studied in relation with the hardware structure. Projects like ReCoNet and SCARS give good hints concerning research in the field [4] [7]. In particular, the SCARS project proposes a complete demonstration of the utilization of reconfigurable hardware to realize a self-healed system.

3.4.5 Requirements for the implementation of an RSS

To create a reconfigurable system satisfying properties and expectations imposed by the concept of self-organization and/or emergence, research works concerning conception methodology for these systems must be followed. This section summarizes the main requirements for this kind of system characterized as self-organized and reconfigurable:

- *Some dynamic reconfiguration capacities* enabling to change the global or local structure in an effective and reliable way. Partial Reconfiguration technique presents an interesting aspect for the conception of various functions after the design time. Indeed, the possibility of modifying a hardware module during run-time without degrading the functionality

of the rest of the system allows self-restructuring of the system giving a realistic self-adaptive solution.

– *A distributed control mechanism* that is spread over the system and that enables a global management starting from node's local control mechanism. To assume a good coordination between those nodes, the control mechanism must essentially endorse a support for computation functionality.
– *An inter-node communication structure* that is robust, distributed and scalable. The nodes coordination can be assumed by a communication protocol that clearly defines the exchanges of information and data. It should allows exchanges with high-speed. The consequences of such communication technology are the optimization of the communication workload between nodes and with external entities. These interactions help to initiate emergent properties like collaborations between nodes that have a certain proximity.
– *A cognitive learning mechanism* that allows the system to learn from its previous experiences and that uses this skill for the resolution of new situations. This learning process allows the system to react to any known situation in an effective and rapid way. In addition, it helps to reinforce behavioral response. The learning technique must provide to the system a way to optimize its behavior to converge through an optimal solution concerning collaborative behavior.

3.4.6 Technological and Conceptual Limitations

The concept has many limitations concerning both technological and conceptual aspects.

Technological limitations

The first limitation is technological and it is partial reconfiguration that implies partial bit files. These bit files are dedicated to a specific kind of FPGA chip and they are also specific to a predefined area in this chip. This leads to the first division between either a homogeneous RSE with identical RSN or an heterogeneous RSE built with different kind of RSN. The result is the necessity to create for each logic functionality, a set of bit files which can be used in the largest set of FPGA type. It is the first limitation: the heterogeneity of FPGA and bitstream files for the same function. This work focuses on homogeneous RSN implemented into a standard network.

The second main limitation is the non-deterministic time necessary for the controller, usually a GPP, and particularly to access its *Arithmetic Logic Unit* (ALU). The OS is crucial for

this. Most systems are not running at real time *Operating System* (OS). Then, it is not precisely known when the tasks can be treated.

The third limitation concerns the diversity of communication protocols like Ethernet, WiFi, 3G and so on. In the best case, an RSN should be able to select the best protocol at any time and evolve with them.

The fourth limitation is related to RSS communication protocol itself, because it does not require any response. RSN must integrate a timer that allows to determine if it should help itself and process tasks sequentially.

The biggest technological limit is that electronic technology provides non-biological elements that are still not really a part of the natural environment. There is a phase difference between biological elements and technological elements.

Conceptual Limitations

The main conceptual limitation concerns the definition of a life-like system, the selection of the characteristis which should be integrated, and the determination of how to recognize it. It concerns the knowledge about the world behavior, cycles and interactions. Consequently, pretending to mix natural computing and reconfigurable computing is just a questionable solution that aims to determine what is a biological systems using technological systems.

To focus in computer science field, the concept of reconfigurable computing is mainly based on FPGA technology which is really limited because new ways to reconfigure a system must be discovered. It can be imagined, for example, a machine able to mechanically change some of its components, like a video processing card.

More importantly, the computation concept itself is a limitation to undecidability and an intractable problem. Notions like NP complete problems and polynomial spaces are important at this level.

3.5 Compatibility with the Peer-to-Peer concept and the best RSS

3.5.1 P2P Compatibility

The main characteristics of the P2P concept is the completely decentralized structure, the complete autonomy of each node and the complete decomposition of files that allows data transfer from different sources simultaneously.

The RSS concept integrates the two first characteristics and the third one is not implemented because it can be considered as an optimisation which does not fit into the research of the elementary node and system.

3.5.2 The Best Theoritical RSS

The theoretically best RSS can be compared to the actual Internet with its nodes that are a combination between human users and computation devices. In this model, nodes are at the same time intelligent and manually reconfigurable. Moreover, human users are able to use different devices relatively at the same time which can be seen as the possibility to upgrade hardware within being disconnected from the Internet.

Conclusion

After presenting the most important aspects of the reconfigurable technology, an analysis of the self-organization principle is proposed with an RSS synoptic definition. Therefore, RSS is defined as a system enabling its restructuring and its management without any external control. In addition, the RSS has the capacity to adapt itself to unpredicted modifications of the environment where it evolves using its nodes interactivity and hardware reconfigurability. The definitions of self-organization characteristics are given in the context of dynamically reconfigurable systems. Each characteristic has been formalized and illustrated using the example of an RSS. A synoptic schematic of this RSS is also proposed. After that, using an analysis of self-organization characteristics in the context of reconfigurable systems, the requirements for the design of an RSS are established. Also, the main architectural aspects that enable the realization of RSS are identified and summarized.

The first aspect to consider in the design of an RSS is the architecture approach. This aspect contains the distributed control mechanism and the way that the decisions, at the system level, are taken through the local mechanism. The cooperative intelligent aspects forming the system play a fundamental role in the design of such systems. Indeed, the characteristics of a scalable communication system are presented in order to adapt them to reconfigurable hardware systems. Finally, the last aspect to consider is the notion of self-learning that is integrated through the analysis of past activity.

The second part of this thesis details the two first requirements for the conception of an RSS, the general architectural approach to such systems and the inter-node communication structure that integrates the decentralized and self-management aspects. In chapter 4, a solu-

tion to achieve the implementation of an architectural approach enabling to decentralize the control of the system and its management in an autonomous and independent way is presented. It also validates the proposed approach through the design of the adopted concept as a part of the autonomous surveillance camera network denoted PICSy (Potsdam Intelligent Camera System project). Chapter 5 presents some results of such implementations and a discussion about the RSS as a possible implementation of natural reconfiguration computing system. In this design application, we show how to organize self-monitoring processing by driven cameras, sensors and detectors based on a network without a centralized management and a control of human operator. We also present one solution to add intelligence to the self-managed system. In practice, we propose a system that can select the appropriate information for processing in runtime which is based on FPGA Virtex 4 Xilinx technology exploiting the hardware reconfiguration ability and the adaptability in time and space.

As presented in this chapter, the RSS has a number of advantages but has one major drawback. It is complicated and uses a lot of different technologies. The topic of the RSS concept is also addressed in two interesting theses: *Architecture Reconfigurable de Système Embarqué Auto-Organisé* [Jov09] and *Architectures, Methods, and Tools for Distributed Run-time Reconfigurable FPGA-based Systems* [Koc09]. This thesis presents complete information concerning methodology and tools to manage auto-reconfiguration [Koc09] and link with MPSoC research [Jov09].

Chapter 4

Realization and Implementation of the RSS Concept: The Node's Design

Introduction

The main objective of this chapter is to show explicitly the engineering work to build the proposed RSS. It demonstrates a realistic solution that is *Autonomous, Dynamic and Interoperable*. Using new paradigms described in chapter 3, all details of the relation between the conceptual ideas and a real implementation are exposed. This chapter is focused on the node level implementation which is the starting point to establish an intelligent and scalable system. As described earlier, the aims of the work is to demonstrate these properties to the networked system by implementing them in each node.

To highlight the importance of the control distribution, some information about intelligence is given. Beside that point, different aspects are used to show hardware parameters, software and communication characteristics. Hardware parameters mainly determine the shared computation resources. During the design time, the selection of these resources is related to the decision of what can be shared. The software side concerns the design of a control application that is independently programmed in order to make it usable on any computer structure. It is a way to transform any machine into an *intelligent* machine. The communication options make *RSN* able to use any medium to send and receive information to other *RSNs*. Using Internet protocol stack, *RSNs* can to be integrated into a world wide web structure or any other sub-network.

In addition to autonomy and dynamics, the intention to make *RSNs* interoperable makes them efficient devices of any electronic devices. A parallel can be proposed here with biological units such as human beings. Both of them are able to communicate, they are able to behave as they wish and they are able to use their hardware for different purposes.

This chapter is organized in five sections. The first one introduces intelligence and artificial intelligence. It is used to show that making all individual *RSNs* more powerful is always a benefit for the created *RSE*. The follow twos sections give the general details about the design of *RSN* and *RSE* respectively. It is followed by a section related to the design realized based on dynamically reconfigurable *FPGA*. The control application built for *RSN* is then presented. Finally, to explain how to make *RSEs* using both *RSNs* and common computers, communication aspects are discussed.

4.1 Scientific Remark on Intelligence

The realization of such a complex system must be defined with wisdom and intelligence in order to follow the main purpose. It tries to combine the concept developed in chapter 3 and the

intention to give *RSEs* some intelligence. This can be achieved by using common distributed control structures and in particular the P2P communication protocols. In addition, the question of intelligence is raised and what it means in the domain of computer science.

4.1.1 Intelligence and Artificial Intelligence

The transfer of some functions from human to control systems is the aim of a lot of different working groups. It is also in the scope of the *RSS* concept. To achieve this, intelligence is defined as follows:

Intelligence is defined as the capacity for learning, reasoning, understanding, and similar forms of mental activity. This definition implies that the concept of intelligence is both multifaceted (i.e., reflective of many aspects of mental ability) and implicative of differences among people (i.e., reflective of degrees of capacity, ability, or aptitude among individuals). Yet this definition does not necessarily relate directly to the definition of intelligence used by scientists. In fact, there is no consensus on the definition of intelligence among professionals who study it (e.g., psychologists, educators, computer scientists) [Gal08].

To fit into the concerned application field, this definition can be completed with the definition of *Artificial Intelligence* (A.I.).

A.I., the ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings. The term is frequently applied to the project of developing systems endowed with the intellectual processes characteristic of humans, such as the ability to reason, discover meaning, generalize, or learn from past experience. Some programs have attained the performance levels of human experts and professionals in performing certain specific tasks, so that artificial intelligence in this limited sense is found in applications as diverse as medical diagnosis, computer search engines, and voice or handwriting recognition. [1]

In these two definitions, communication is implicitly connected with ideas developed. In our case, the property of *Intelligence* depends on data understanding and consequently how to get and send back information. It is an important aspect of this work to show that *RSE* has a communication capable system. Consequently, to realize a intelligent system, automatic communication functions are required. These functions help to show that the system has some intelligence. The goal is to approximate real human intelligence as much as possible.

1. www.britannica.com

4.1.2 Intelligence Transfers from human to *RSN*

To transfer intelligence to *RSN*, it is selected to give *RSN* the possibility to send broadcast messages containing resource requests when they are required. As described in chapter 3, *Dynamics, Autonomy and Interoperability* are the three properties that are necessary to develop the concept of RSS. They can be generated by thinking communication in a way that allows *RSN* to behave alone or in other words with Intelligence.

The relation between Intelligence and Control Distribution makes the necessity of the comparison with natural systems more clear. The more nodes are intelligent and have information, the more they can be free and autonomous. The node's *level of intelligence* is very difficult to estimate and finding references to do so is critical. However, the previous assumption is clear enough to validate the necessity to study this point. In this work, an intelligent node or *RSN* is designed to respond to this quest. It is focused on its implementation of the triple set *Autonomy, Dynamic and Interoperability*. In the next three sections, technology that allows to implement this concept in a way that fits this definition is detailed.

4.2 The design of Reconfigurable Devices

The goal of this section is not to give a profound course on FPGA technology, but it aims at guiding the reflection to a general idea about engineering *RSN*.

The design of a *reconfigurable device* is not so different from the design of a *static device*. The main methodology to build such a device can be described in a set of three steps: the definition of static functionality, known dynamic functionality, the development of the system and the realization of the device. To be more focused on the FPGA technology, it consists of the design of the main part *SoFPGA*, the selection of an *FPGA*, the design of the configuration tool chain and the design of an electronic board for it. The goal of this section is to show the limitations and constraints of reconfigurable design.

4.2.1 Main Xilinx Technology Aspects

The implementation of *RSS* and the design of the *RSN* is mainly based on Xilinx Technology that provides partial reconfigurable *FPGA* based on *SRAM* technology. The Virtex family is selected because it is the most powerful *FPGA* family from Xilinx.

Two Virtex devices were mainly used, the *Virtex 4 FX20* on *PICSy* platform and the *Virtex 5 FX70T* on the *Xilinx ML507*. Swapping from generation 4 to 5 has a certain impact. In Virtex

4 technology, the *LUTs* are designed with 4 inputs, while Virtex 5 technology allows up to 6 inputs per *LUT*. The consequence is the reduction of necessary *LUT* blocks for the same design and the reduction of the routing requirements. This example of technology evolution shows that the quantity of computation resources, logic components or more complex processor cores implemented into the same chip is always growing. This signifies that it is possible to build larger and more powerful computation systems. Nevertheless, these systems are built at instant t which means that they reply on the requirements for a certain time and they are limited in terms of structure evolution.

Xilinx *Partial Reconfiguration* (PR) technology is the key factor of this work. The proposed concepts and its implementations are derived from it. As presented in chapter 2, it allows to ***RE**-configure* a part of the *FPGA*. Moreover, the *ICAP* (Internal Configuration Access Port) pushes the idea to the limit and allows the *FPGA* to reconfigure itself during run-time [Inc10a].

In addition, Designs Tools provided by Xilinx allow us to play with a larger set of parameters and possibilities. They mainly include *ISE*, *EDK*, *ChipScope*, *PlanAhead*. *ISE* is the basis tool for *Hardware Description Language* (HDL) module design. *EDK* is composed of *XPS* and *SDK* that are dedicated to *SoFPGA* design. *ChipScope* is an internal signal analyzer for *FPGA*. *PlanAhead* is a *place and route* tool that also allows to implement *PR* characteristics and to generate *PR* bit files.

There are also others tools such as the FPGA editor from Xilinx that allows to play with FPGA logics directly. ReCoBus Builder [Koc09] is also useful to place and route modules inside the FPGA.

4.2.2 System on FPGA Design

Figure 4.1 shows a general structure of a SoFPGA. Three parts can be distinguished: an external memory, a static region and a reconfigurable region.

The static part has the main role in this structure. It is the heart of the system where a controller is in charge of all functionalities. Two links are crucial, the first one is the link to the memory that is used to create a computation architecture that is identical to a personal computer architecture and the second link concerns the reconfigurable regions. Here, the controller has a double interface that is used to configure the *PR* Region over the *ICAP* interface and to send and receive data from the reconfigurable logic circuit which can be seen as a co-processor. Moreover, the static region is mainly composed of a controller that can be a soft or a hard core processor, it can even be a custom hardware controller. The main bus interface is also present, it is a backbone unify the all IPs of the SoFPGA.

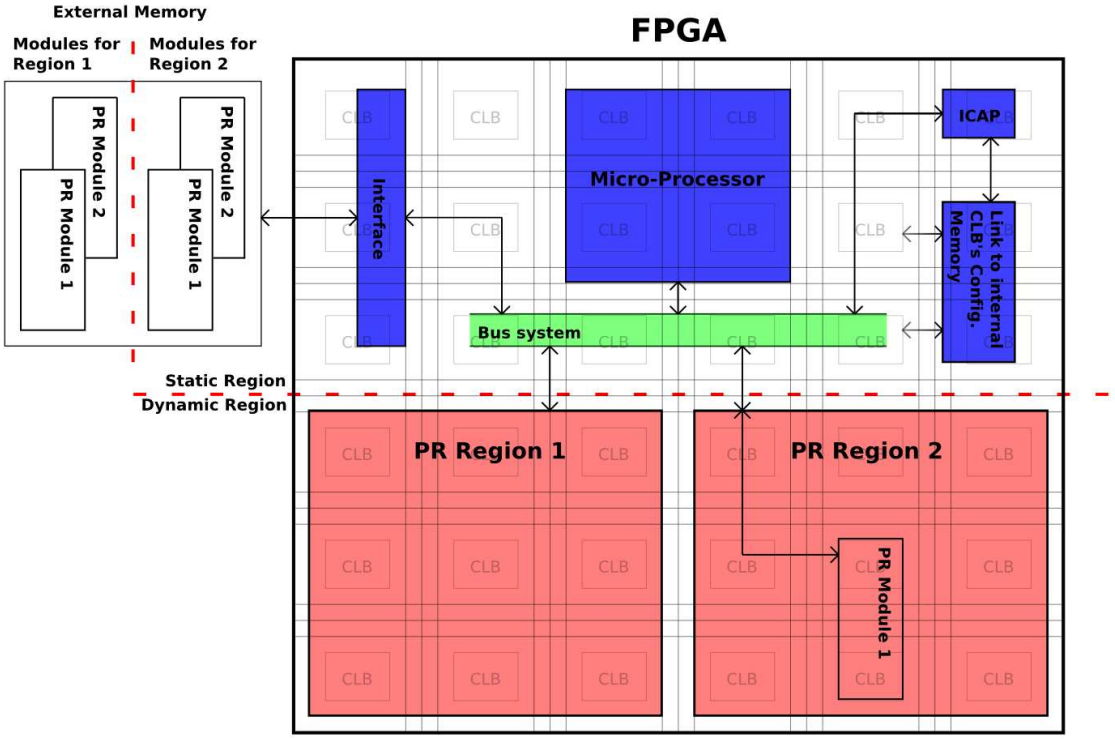


Figure 4.1: Architecture of System on *FPGA*

At this level, the limitation is the design itself. Building the *SoFPGA* means that restrictions are imposed on the system and specially the reconfigurable areas are selected inside the *FPGA*. By designing the *SoFPGA*, the quantity of resources required for the static region and the known *PR-modules* are defined. It gives an estimation of which *FPGA* should be considered for the design of an FPGA based self-organized system.

4.2.3 FPGA Selection

The *FPGA* selection is by definition a limited choice that generates the first constraints related to the possible static and reconfigurable region sizes. It also determines the type of logic resources that can be used including the maximum performances defined. Based on the *SoFPGA* resource estimation, a specific *FPGA* can be selected but with a certain imprecision with the idea that in the future new *PR-modules* can be designed for this platform.

As example, *Xilinx Virtex 5 FPGA Family* is defined by different characteristics.

– CLBs - Configurable Logic Blocks

– DSP48s

- Block RAM Blocks
- CMTs - Clock Management Tiles
- PPC Processor Blocks
- Ethernet MACs
- Max User Array I/O

CLBs, *DSP48s* and *Block-RAM* blocks are basic logic blocks that can be completed with *CMTs*, *PowerPC* Processor Blocks and *Ethernet MACs* blocks to realize a complete *SoFPGA*. In addition, the size of User *I/O* helps to determine how many electronic connections can be made between the *FPGA* and other on-board electronic components. Finally, the package type is determined in order to integrate the *FPGA* chip on the electronic board.

Figure 4.2 shows the architecture of the *Virtex XC5FXT FPGA* (this figure is rotated 90°on right side). In particular, the two blue columns are the *DSP48* blocks that can be really useful for any Partial Reconfigurable Modules because they include a set of hardware multipliers.

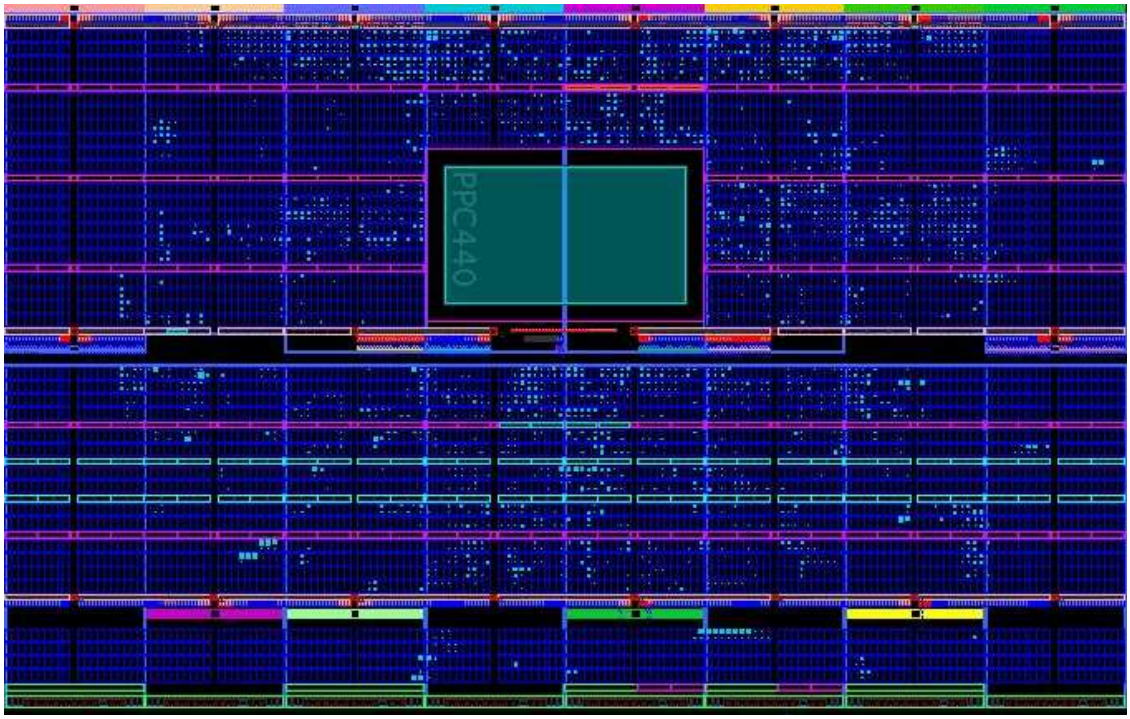


Figure 4.2: A floorplan of the *XC5FXT FPGA* architecture

In the case of this *FPGA* architecture, the placement of the partial reconfigurable regions can be more efficient in the right side of the FPGA. As presented here with this example, the area location of the *Partial Reconfigurable Region* (PR-Region) is limited by the *Static*

Region (SR) area location, the resources that are necessary in it and the connection interfaces between Static and Reconfigurable Region. To sum up the *FPGA* selectionis deduced from the knowledge of the resources that are required of the design and in which quantity.

4.2.4 Board Architecture

After the FPGA selection, the design of the board allows us to make a connection between the FPGA and the other required components like the *Ethernet controller* and the *memory chip*. The memory interface is crucial to provide the best performances and it should be particularly good due to the frequency of data transfers.

At this level, there is no difference to any other standard embedded electronic system. Nevertheless, PCB design is always important because it requires to determine all interfaces, power supply, input interfaces, communication, video output, custom interfaces, extension interfaces and so on.

This is one of the main limitation of electronic devices but it is not the aim of this work. However, for the global understanding, it is included and simple notify.

It should be noted that when an electronic board is designed, especially a multi-layer board, after manufacturing time it become extremely difficult to modify it. The board cannot be changed anymore.

4.2.5 Configuration Tool Option

FPGA technology offers many possibilities to configure FPGA chips. The configuration strategy should be selected depending on the performance required and board architecture. The different configuration tool chain possibilities are:

- Master-serial configuration mode
- Slave-serial configuration mode
- Master SelectMAP (parallel) configuration mode (x8 and x16 only)
- Slave SelectMAP (parallel) configuration mode (x8, x16, and x32)
- JTAG/Boundary-Scan configuration mode
- Master Serial Peripheral Interface (SPI) Flash configuration mode
- Master Byte Peripheral Interface Up (BPI-Up) Flash configuration mode (x8 and x16 only)
- Master Byte Peripheral Interface Down (BPI-Down) Flash configuration mode (x8 and x16 only)

Details concerning all those possibilities can be found in [Inc10b] but they are mainly concerned with "manual" configurations with hardware solutions. Another possibility is also described in this document which is necessary for this system, namely to use the internal access using the *Internal Configuration Access Port* (ICAP). This port is the key solution for offering auto reconfiguration, as described in chapter 2, to the *RSN*.

Auto-reconfiguration is possible by associating a complete set of technology that contains digital electronics, software design and informatics. Technically, this means a special architecture based on *FPGA*, a *SoFPGA* designed to be partially reconfigurable, an operating system layer and finally a software application that is used on top all this structure to manage everything. It concerns all information presented before. The design of such complex system must thought in global way.

4.3 The Design of Scalable Systems

The design of a so called *scalable system* depends on the selected definition. For this work, it is considered that a scalable system is a system that is networked and that can include or exclude nodes during run-time.

In addition to auto-reconfigurable capacity, *RSN* can also be designed as a scalable device. This means that it can be upgraded at the hardware or software level. For example, personal computer architectures are designed to receive other electronic cards like a more powerful graphical card or audio card and moreover all devices that are using a *USB* interface, *WiFi* interface, memory stick and so on. *RSN* are designed in the same way.

To summarize the global capacity of evolution, the following list is proposed:

HW	Update Upgrade	Use of connection protocol technology Use of hardware redundancy	PR allows run-time update PR allows run-time upgrade
SW	Update Update	Inherent to software layer Inherent to software layer	

Figure 4.3 shows a scalable network which is a mesh network. Devices integrate or quit the group over the time and space, making the system scalable. For example, the bi-color nodes are part of the group from time to time.

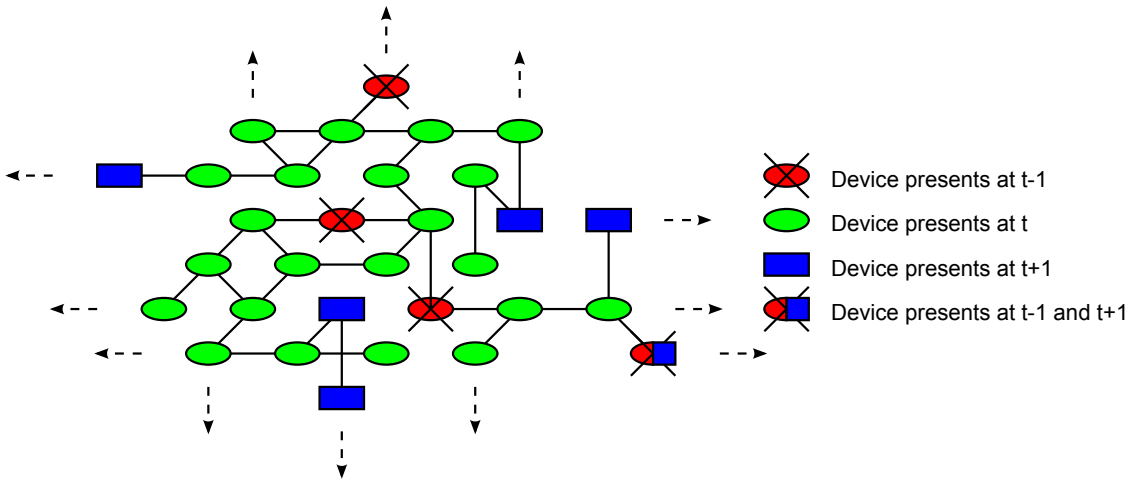


Figure 4.3: Scalable Networked System

4.3.1 Device Selection

The selection of nodes that can be the basis for *RSE* is a central requirement. *RSNs* are Autonomous, Dynamic, Interoperable, and also reconfigurable and self-organized. To fit into this description, *FPGA* based systems are the best system that exists at the moment to realize the *RSS* Concept.

Therefore, on open-minded view of the *RSS* concept allows to see the rest of existing devices as limited *RSN*. In fact, *RSNs* are composed of on three parts: computation resources, *HardWare/SoftWare* (HW/SW) block, and control application. With this structure, it is possible to consider only the control application part and run it on any computation devices.

The device selection becomes dynamic and allows us to make an *RSE* scalable. *RSN* can be seen as nodes that can be replaced by other more powerful nodes. It is the core of the *RSS* concept. The *RSN* scalability is the starting point for *RSE* scalability.

4.3.2 Organization of Communication

Currently, the main communication structure is the Internet and TCP/IP stack plays an important role. In order to make *RSN* interoperable, the communication structure of *RSN* should be based on the same stack.

On top of this communication stack, the internal communication of *RSE* is based on the idea of communication point-to-point dataflows travelling from one node to another node in order to propagate the information and data. A synoptic scheme of *RSE* internal communication is

presented in the Figure 4.4. When possible, the system solves the existing problem locally. If not, the rest of the system becomes aware of it and the fluxes are generated in order to progressively inform the rest of the system.

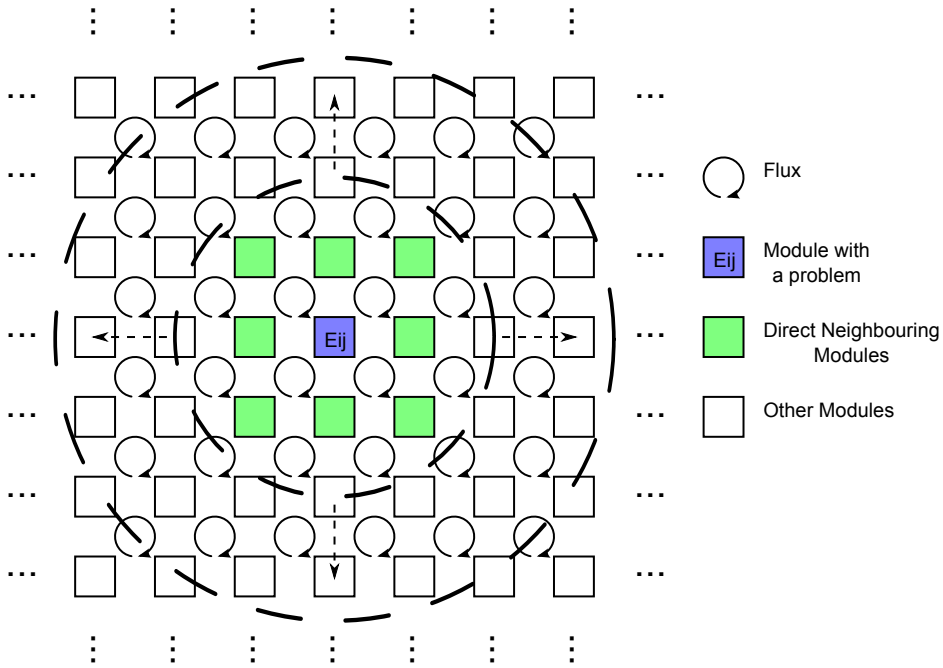


Figure 4.4: *RSE*'s internal communication fluxes.

Actually, the management aspects and the dynamic hardware structure in the context of networked self-organized modules has not been considered again. The control needed to achieve *RSE* must be distributed over all nodes. Indeed, the increasing needs of computation power, flexibility and interoperability, make the system more and more difficult to integrate and to control. To address this issu, the *RSNs* are designed to control themselves by making themselves more intelligent and by dividing the problem into smaller problems. With the ability to do what they are able to do, *RSNs* can select to request help from other *RSNs* and also they are free to service themselves. Figure 4.5 shows a scenario of resource sharing between *RSNs* connected over the Internet network. This scenario is the most basic and can be duplicated many times into a single *RSE* composed of variable number of *RSNs* .

By Handling the TCP/IP stack and common computer architecture allows the use of any communication medium, wired or wireless, and the use of most communication standards like WiFi, WiMax, Blutooth and ZigBee.

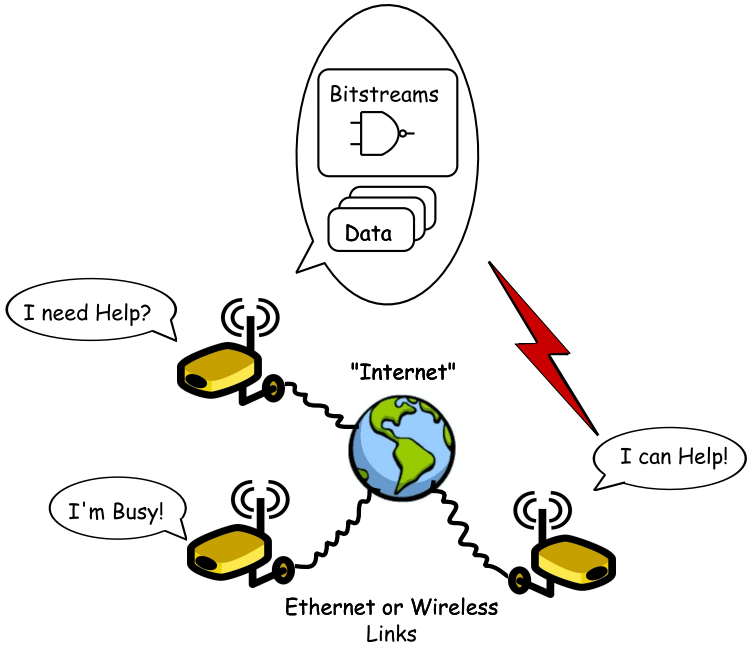


Figure 4.5: Example of Self-Organization Scenario.

4.3.3 Bitstream Management

Bitstream management is related to memory management. As explained before, the management of bitstream files results in a limitation of the *RSE*.

Since *RSE*s are thought to be heterogeneous, they can be based on different types of *FPGAs*. Different types of *FPGAs* means different kinds of *PR-Regions* thus it necessitates a set of bitstreams for each *PR-Module*, one partial bitstream file for each specific region.

This is another limitation of the *RSS* concept because the size of bitstream files to be generated can grow exponentially. The management of these files requires a more complex system than just copying all of them in the same folder. Figure 4.6 shows the considered relation bitstream/PR-region.

4.4 The design of the *Potsdam Intelligent Camera System*

This section shows the realization of an *RSN* based on a custom platform built for video application and processing. It aims to demonstrate the feasibility and the reality of *RSS* that tries to reach the goal of a highly natural system. The PICSy platform is developed at the Institute of Informatics at the University of Potsdam [BZM+10]. This group was created in

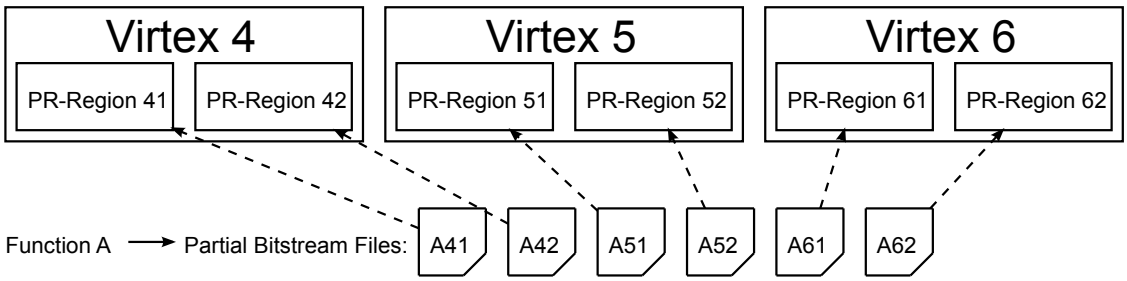


Figure 4.6: Description of the relation between bitstream files and PR-Region.

October 2007, the development of the PICSy project started in April 2008 and the platform was operational in 2010.

In most of the existing visual processing platforms, the computation is performed in software with high-end workstations to provide the required computational power. It is not unusual to find a cluster of *PCs* in a production chain or even in navigating robots. Sequential processors like the *Intel Xscale* or the *DSPs* used in embedded cameras do not provide enough computational power for complex image understanding algorithms. To overcome this limitation, an FPGA-based camera system that fulfils the previously mentioned requirements is proposed. The goal is to provide a miniaturized system with a rich set of interfaces and appropriate performance. The efficient partitioning of applications with the use of hardware accelerators in the image processing framework *OpenCV* has governed the design process.

Our motivation in designing a new platform was to provide a universal video processing machine for use in various domains. This would help researchers and programmers to develop and test their prototypes in the field without having to rely on the cumbersome evaluation boards or workstations. Video processing is a versatile domain where requirements on an architecture vary according to the application and the goals. The computation is usually streaming based. In front of the stream, either *CCD* or *CMOS* devices can be used for the image acquisition. After capturing the image, the processing is done in a given number of steps, some of which are iterated several times, according to the algorithm. To fulfil all these requirements, a modular system consisting of a set of hardware modules, the combinations of which define the possible system configurations, has been designed. According to the application and the target environment, the viable combination can be chosen.

4.4.1 Board Design

Figure 4.7 is a representation of the multi-board architecture that was selected to be built as the basis for the *PICSy* platform. This design is mainly composed of two boards, the main *FPGA* board, where the controller is implemented inside a *SoFPGA* and the Power Board which contains power supply components and communication elements. In addition, the *Sensor Board* makes the design suitable for video applications while the the Debug Board is set with *LED* and switch to facilitate the debugging of the platform. The design of the proposed platform aspires to be a general high performance image processing platform with real-time imaging capabilities. To address a very powerful system in terms of computation and scalability, the system is designed as a stackable board platform. In this design, additional *FPGA* boards can be added to create a multi-FPGA system which, additionally, offers the possibility to create a fully dynamic structure.

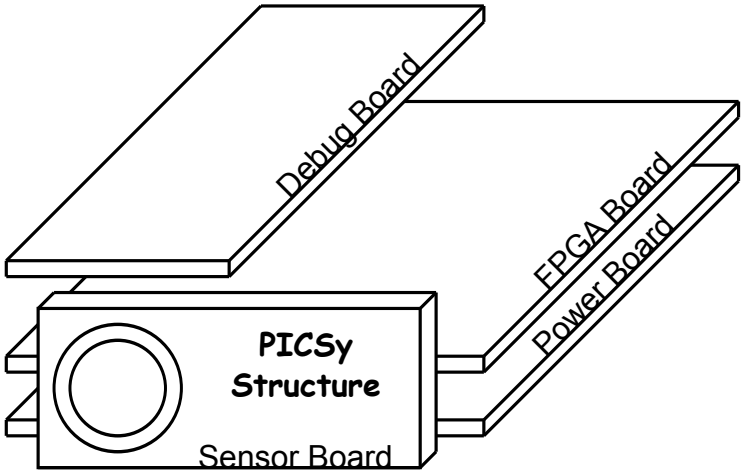


Figure 4.7: *PICSy* Board design

Figure 4.8 shows two photos of *PICSy*. The first one is a photo without the chassis where a fourth board can be seen. The fourth board is an additional board used to add multiple image sensors through the custom use of six *RJ45* connectors. The second photo shows the camera within its chassis. Such configuration includes only the main *FPGA* board, the power board and the sensor board.

The *Virtex-4 FX* series is selected as main technology, it is equipped with an embedded *PowerPC 405* processor. It has 19 224 logic cells and around 1 Mbits *BlockRAM*. The *FPGA* is mainly completed with a 64Mbytes *DDR-RAM* and Ethernet interface.

A single platform with the *FPGA* computation module and all the interfaces usually needed

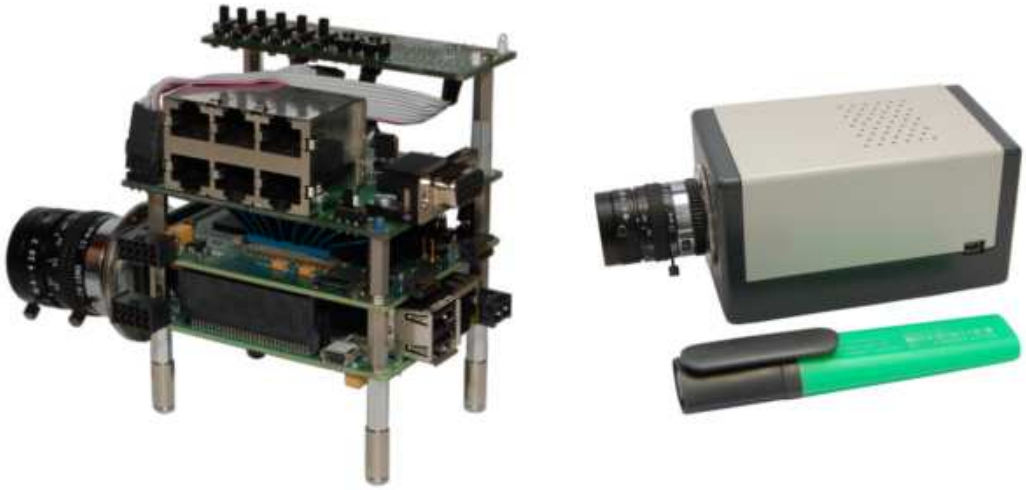


Figure 4.8: The PICSy Camera: a reconfigurable system

would have made the system on a single board too large and indistinguishable from existing prototype boards. Therefore, the computation was separated from the communication by using a dedicated interface module for the connection of the system with the outside world. With this, we void on the one hand the implementation of the *FPGA* and its configuration chain and thus save space on the board. On the other hand, the system is miniaturized because the growth occurs in the third dimension. The current interface module provides many connectors, two for *USB* devices and one for *USB OTG* connectors, one for Ethernet and one for a serial *RS323*, finally it has a connector for power supply and *JTAG* configuration chain.

Figure 4.9 shows more details concerning the components embedded on the different boards.

The main FPGA board, here called Coordinator Main board, is composed of a Virtex 4 FX 20 FPGA, a DDR Memory chip and connector to interface the board with the other. In addition, a JTAG end RS232 interface is implemented in order to allow the FPGA configuration and to set a communication link between users and the FPGA. Moreover, another interface allows to add a secondary portable memory and secondary System Management MCU chip is present as a secondary option to control the FPGA.

The Power Board, here called Coordinator Daughter board, is composed by power supply components, USB interface and most important the Ethernet interface. The sensor Board, here called Camera head board, simply integrates an image sensor. To complete this design and due to the intention to design a system that can physically evolve, it is possible to connect additional FPGA boards, here called Image sensor module. Those boards are composed of an additional

Virtex 4 FX 20 with its auxiliary components: DDR memory, clock and PROM.

The interface between the FPGAs is designed as a powerful backbone using LVDS technology. Also, the JTAG chain required to configure additional FPGA is scalably integrated into the design of the boards interface.

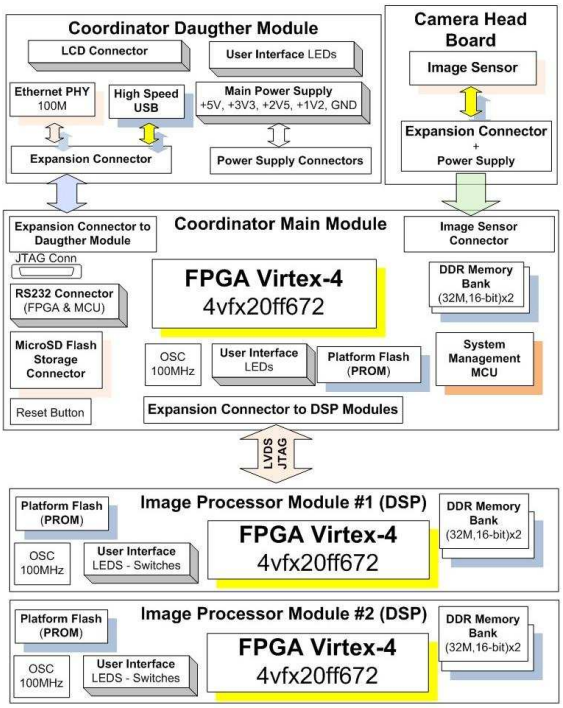


Figure 4.9: The *PICSy* platform architecture

4.4.2 SoFPGA of PICSy

Figure 4.10 shows a snapshot of the *SoFPGA* designed for testing the *RSS* concept on the *PICSy* Platform. In this picture, three kind of blocks are shown, the controller which is developed as a complete structure including memory, communication interfaces, clock management and so on. More precisely the processor is a *PowerPC* embedded into the *FPGA*. The *PR-Module* is placed into the *PR-Region* and *Bus Macros* are the interfaces between the static and the dynamic part of the design. *Bus Macro* are logic resources selected to be the interfaces between the static and the dynamic parts of a reconfigurable SoFPGA.

This SoFPGA manages simple reconfigurable arithmetic functions like adders or subtracters that are taking place in the single *PR-Region*. The goal is not to design and study a video application, but it should be useful to discover the technology and use of auto-reconfiguration

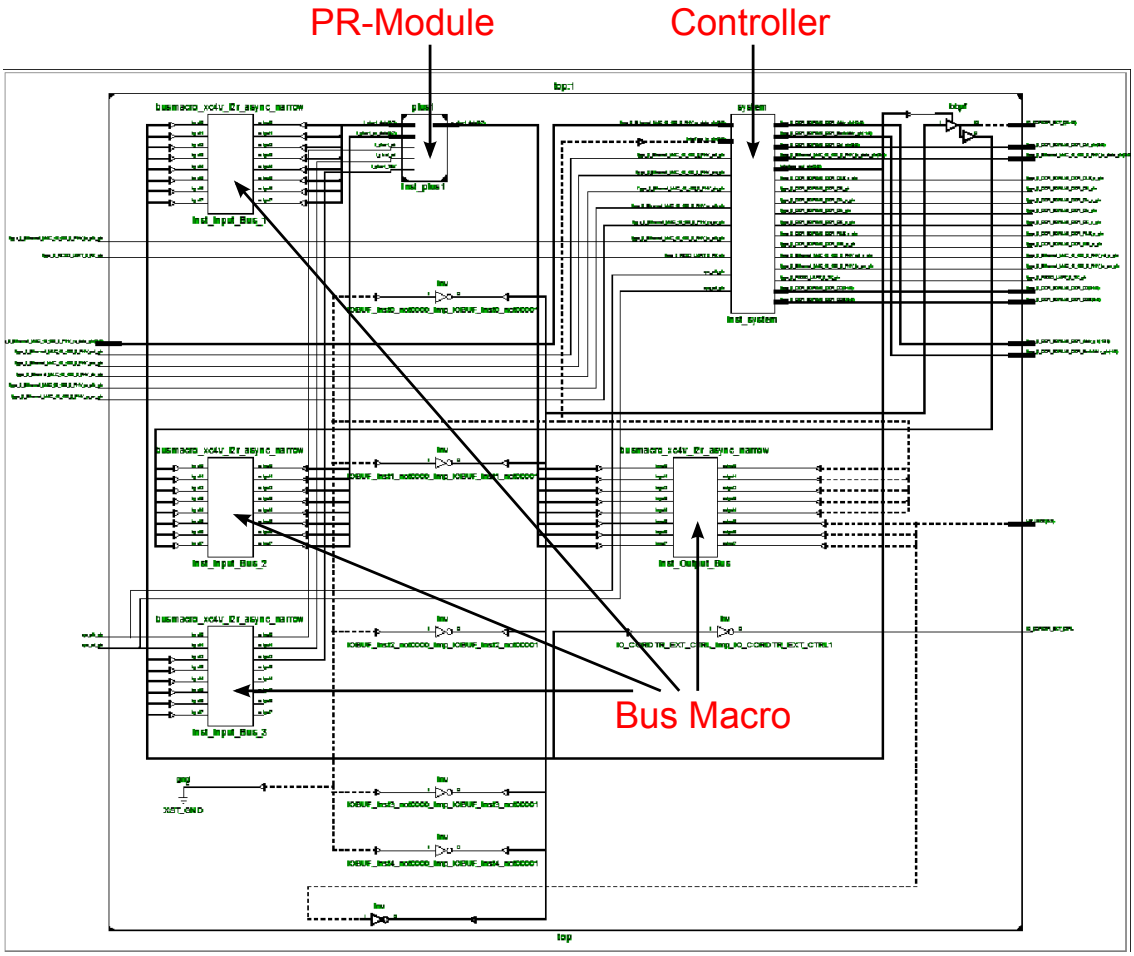


Figure 4.10: Caption of SoFPGA designed for PICSy

devices into a more complex structure. In [HBB04] and [HUKB04], the authors are working at the NoC level where only PR technology and application are considered. This work aims to integrate PR into existing macro networks like the internet. This means using standard node architecture in a networked structure. Consequently, our works are complementary: ON chip/OFF chip co-design. In addition, works of [HBB04] and [HUKB04] presente a NoC structure based on bus interconnection which can be extended to complex routing technology including TCP-IP protocols.

The input of this Region can be either a switch of the debug board or expansion output of the *PowerPC*. The computation loop ends by sending back the result coming from the PR-Module to external *LEDs* and on expansion input of the *PowerPC*. Data can be processed on the *PR-Module* and directly managed by the control application.

Figure 4.11 shows an example of region organization into the *Virtex 4 FX20*. The rectangle

on the right side is a PR-Region. The rectangle on the left side contents the static part of the design. The regions of the FPGA are real placement constraints, a minimum of them must be defined to facilitate and process the place and route tools. This means that describing only the PR-Region is the most optimized option. Nevertheless the proposed figure shows two regions to simplify the understanding of the design.

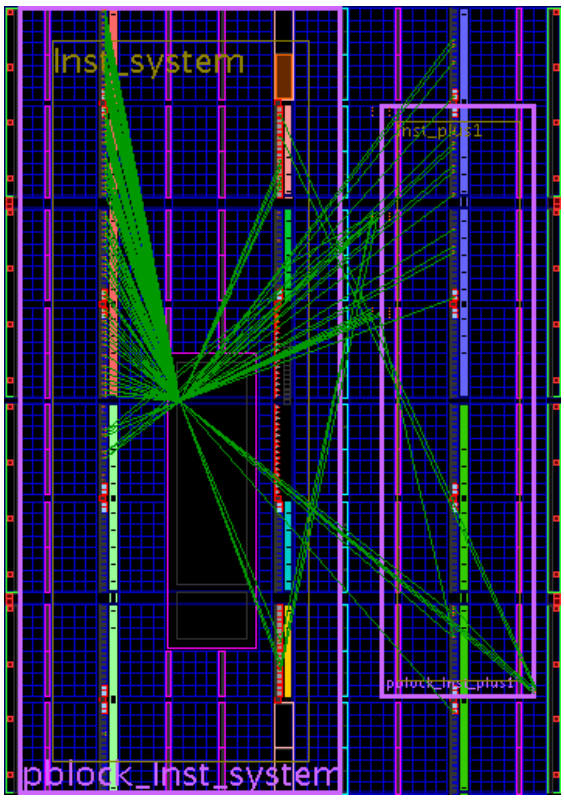


Figure 4.11: Placement example into a R-SoFPGA.

4.4.3 Embedded Operating System and Driver

Electronic and software technologies are clearly oriented to provide static hardware systems planned of obsolescence. This reality has a contradiction, the software part that is fully dynamic and scalable. This architecture is built to use these technologies rather than how they are defined. This engineering work introduces auto-reconfiguration and to manage this new feature, an *Operating System* is used on top of this hardware to control reconfigurable processes and create the links between reconfigurable logic components and the control application.

The *O.S.* is based on a Xilinx embedded distribution that is compiled with the Open-

Embedded framework [2]. Xilinx drivers are now already included into GNU/Linux kernel. They just require activation during the compilation time. For this example, two drivers are used, the ICAP driver originally written by John Williams [WB04] and the GPIO IP-core driver. This last one is used to control input and output of the embedded processor system in static part of the R-SoFPGA.

At this point, the utilization of the operating system is important for two reasons. The first one is the file system that allows file manipulation. Auto-reconfiguration technology allows to create electronic circuits using files to configure component arrays, thus making sense to reach and include a file system. The second one is the communication stack that allows communication over Ethernet is useful to use the Internet so that any *RSN* can be deployed anywhere at any time and in any kind of network. The created one can be the technological basis of an intelligent self-organized and networked system.

4.4.4 Resource Management Software

The design of the resource management software, which is the PICSy control application, started with two Xilinx development boards. The first goal is to control PR from one board to another using a simple control signal. Figure 4.12 shows this first implementation.

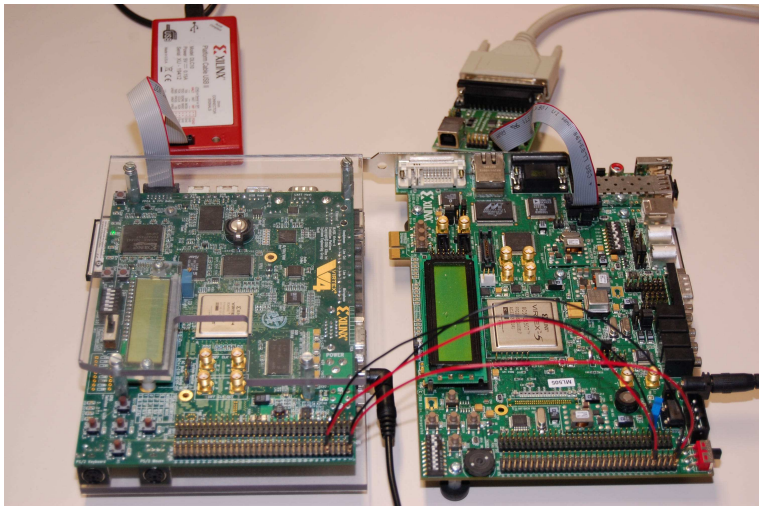


Figure 4.12: First remote PR control.

Figure 4.13 shows the structure of *RSN* functionalities, and how it combines communication, intelligence and resources management functions, as was already presented in chapter

2. OpenEmbedded at www.openembedded.org

3.

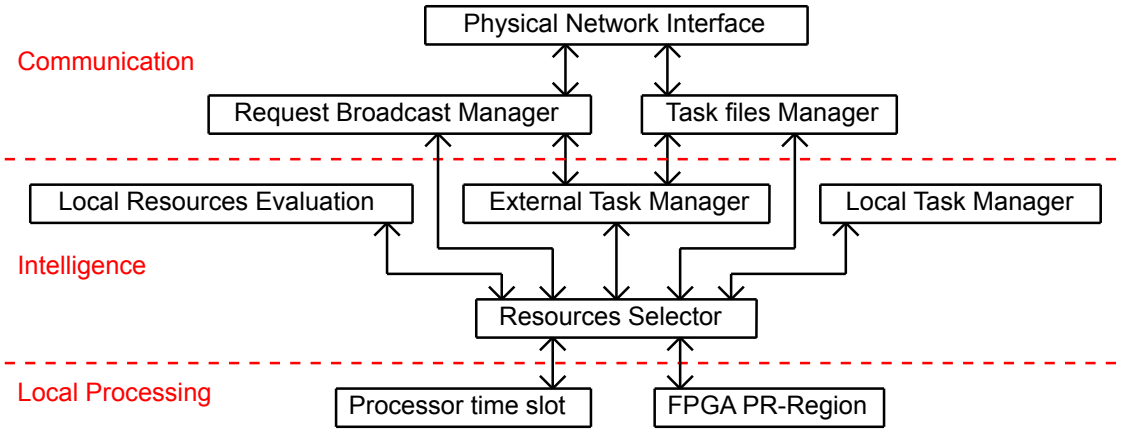


Figure 4.13: *RSN* functionalities.

The communication part is composed of two modules: one for the Broadcast service and one for the file transfer services. Both of them use the communication interfaces. The Intelligence part is in charge of associating tasks and resources using four modules. Two of them are used to manage the local tasks and resources. The third one is the resource selector itself. Finally, the last one is related to the external task manager responsible to link with external tasks and resources. In essence, the last one describes explicitly local hardware resources.

This structure is designed as simple as possible. It checks local the resources and local tasks to process in order to associate them and, if required, it immediately tries to find local hardware resources to accelerate the processing task.

RSN hardware structure is based on *FPGA* and like any other electronic device, after the design time, it is not possible to modify anything on the electronic board. But now, it is possible for a complete *SoFPGA*, designed like any other computer, to modify its own modules at run-time. As explained before, this run-time capacity is possible because of an internal access to the configuration structure: the *ICAP* (see [Inc10a], [Inc10b]).

The design of the control application for this project requires a controlled partial reconfiguration. Two main solutions are possible, writing a dedicated operating system scheduler or writing a completely independent scheduler. It was selected to program a portable application to keep the most simple solution which is also the most interoperable solution. The application is developed in C language to reply on the specifications presented in the previous section and in chapter 3.

Figure 4.14 shows the structure of the control application developed to add PR technol-

ogy and reconfigurable computing to any common computer system and that uses common communication system. It is based on RSN functionalities described in figure 4.13.

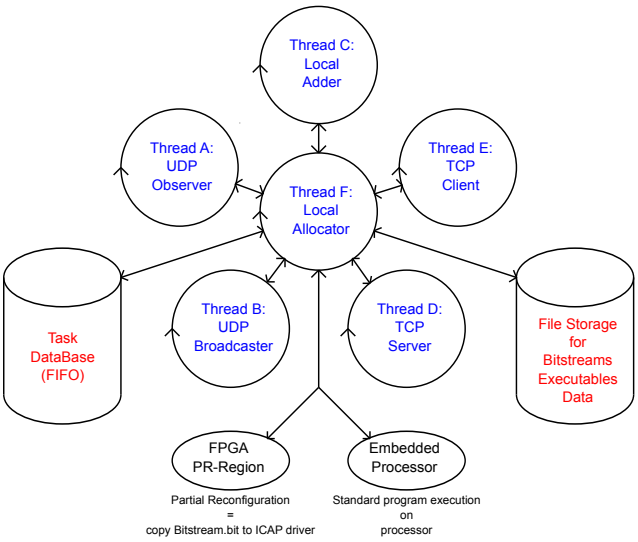


Figure 4.14: The Program Architecture.

The way in which this program operates is based on the assumption that it should be as simple as possible. Multi-thread programming is used to allow the program to constantly check evolution in the task database, resources availability and neighbourhood activity. By doing that, the *RSN* control application gives the appearance of a kind of intelligence where it only combines some automatic functions.

The six threads are used for a set of fundamental functions. The first thread is used to locally add tasks to process for the *RSN*. The second thread, *Local Allocator*, is responsible for the resource assignments. The four other threads are used to set a *UDP* broadcast service for resources request and a *TCP* service connection to exchange data like bitstreams, programs or data files.

To complete this application, a *FIFO* is used to manage multiple tasks as fast as possible and the *O.S.* file system is simply used to store and manage all files but without any kind of special processing or storage; it is a simple read/write storage utility. As proposed earlier, the control application allows to compute data on a GPP or on *PR-Region* depending on the resources availability. Tasks start to be processed by simply checking if data are processed at the same moment on resources. If all local resources are busy, then the concerned *RSN* has the possibility to request help using the *UDP* broadcast service.

On the other hand, any other *RSN* can propose help but only if they have free resources.

To achieve that, they listen to the broadcast channel and reply to requests that they receive but just if they can propose available resources. Therefore, a resource is assigned to a task which is reconfigured in the case of PR-Region. The size of the data bus connected to the PR-region is crucial and one of the main requirements during the design time.

4.4.5 Networked System and Communication

Figure 4.15 shows the basics information broadcast over the *RSE*. It just concerns task information. All addresses or acknowledgments are managed by the *TCP/IP* protocol layer and or not treated into this document. Nevertheless, *UDP* broadcast messages contain the sender's *IP* address information that is required for any *RSN* to reply to a resource request.

Request Informations send over the UDP broadcast

Task ID	Bitstream file name	Program file name	Data file name

Figure 4.15: Broadcasted Requests.

Figure 4.16 shows how *RSN* can communicate by asking nodes which can share computation resources and exchange files directly with one of the repliers. It is composed of two parts. The first one is the broadcast negotiation and the second one is the effective teamwork that means files exchanges and data processing.

The result of this implementation is also a completely non-centralized network. As proposed before, each node is Autonomous, Interoperable and able to start collaboration dynamically with other nodes. *RSNs* that are local systems can be now developed independently and without affecting the *RSE* behavior. This has the advantage of providing scalable performances from the *RSE* performances.

4.4.6 Hardware Module Exchange and P2P

Figure 4.17 shows how the communication is organized through the flux concept.

In a natural system based on *RSS* concept, the exchanges of hardware can be compared to *P2P* exchange because it is based on a complete decentralized networked system where nodes are autonomous. In addition, the nodes can operate client and server simultaneously.

In a real *P2P* exchange mechanism, file copies are located into many different places but this is not proposed to be in our system. Again, the goal of this work is to keep the system as simple as possible in order to set fundamental functions.

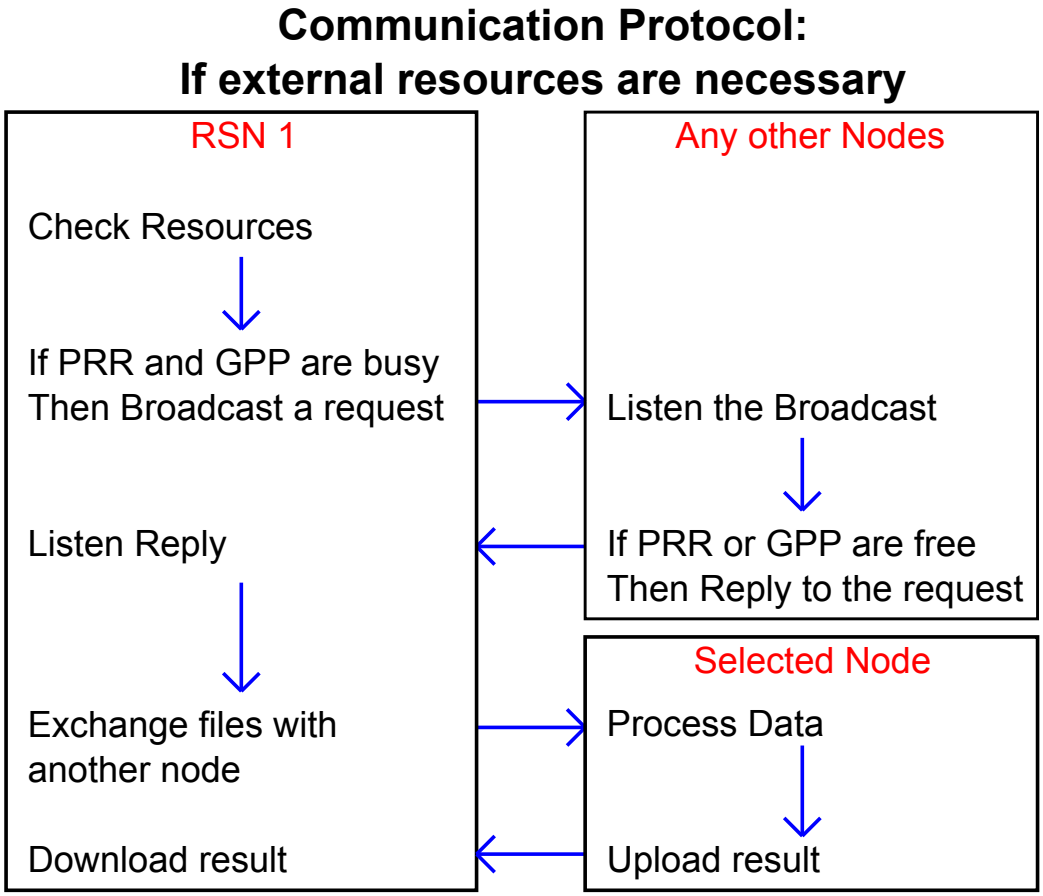


Figure 4.16: Communication Protocol between *RSNs*

4.4.7 Drawbacks and Advantages

Designing a *RSN* based on the *PICSy* platform helps to face real problems requiring to be controlled and corrected in order to give a realistic view of *RSN* devices and *RSE* systems. This allows to present advantages and drawbacks of the general concept. In this section, only engineering aspects are presented.

Drawbacks

The *Virtex 4 FX20* has a main limitation: the number of logic cells available. This limitation is reached fast and it confirms the benefit of partial reconfiguration capability for small *FPGA* platform. In the next chapter, examples of implementation show the limit of this specific *FPGA*.

A second limitation is the utilization of a *PowerPC* which implies the necessity to cool down the chip. Using the maximum frequency of this system (300MHz) requires to add a heat

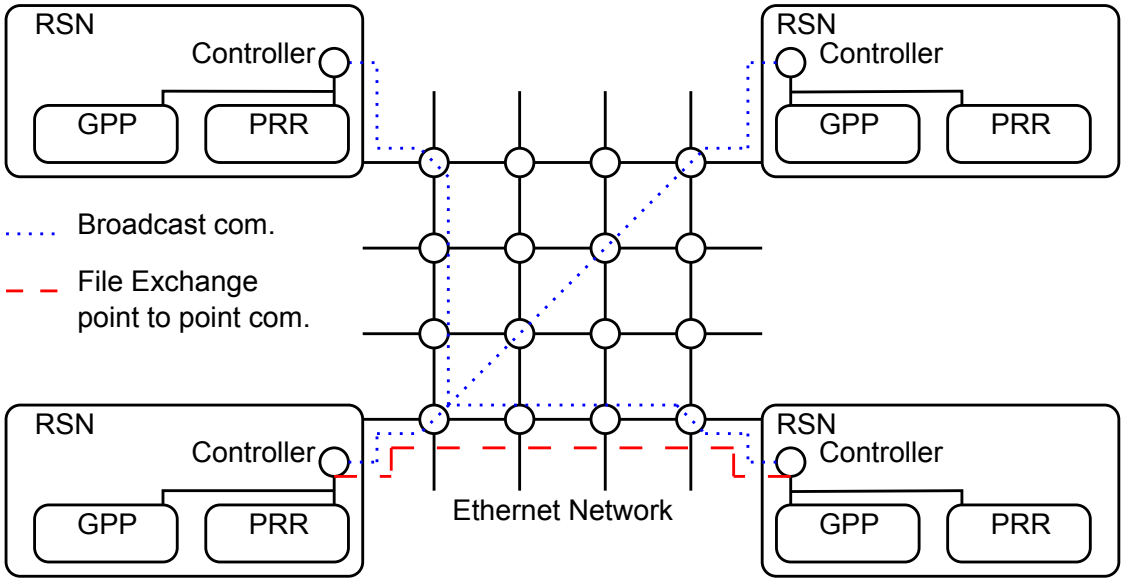


Figure 4.17: Example of networked *RSN* using the proposed flux to communicate

dissipation component which in return increases the size of the device. Another problem is that *PowerPC* hardcore is not used anymore in the last *FPGA* series from Xilinx.

Another limitation concerns the selected connectors for *JTAG* programmer and the serial *RS323* link. As non standard interface, they are really small but also remain fragile.

In the case of the *SoFPGA*, a drawback aspect is the *GPIO* interface used between the bus macros and the processor core. The problem requires to manage bytes word bit by bit. An alternative solution is the utilization of a *Fast Simplex Link* (FSL) bus which provides a direct connection to the processor.

At the software level, the utilization of C programming code limits the facility to program such control application which could fit completely into the field of C++ programming.

Advantages

Except PR capacity, the prototype provides many other advantages.

The Virtex 4 also includes a *PowerPC 405* used as the main controller running a custom *GNU/Linux* operating system. It has the advantage of rendering the platform a special computer that has *PR* capability. This means a file system to store bitstream files and data, standard network accesses and possibilities to design a software that can also be run on any other computer.

The second advantage is the modularity of the devices that keep the idea of stackable electronic board and common interfaces. In other words, additional *FPGAs* can make this platform

	Negative Aspects	Positive Aspects
Hardware	FPGA: Limited Resources Planned obsolescence	Common computer architecture Modularity of the platform
SoFPGA	GPIO IPs as Interface Hardcore Processor	Reconfigurable Usable into a NoC
Software	C Programming Complexity Non deterministic processing	Portable ICAP Driver

Table 4.1: Summary of Positive and negative aspects

really powerful and a *USB* interface enables it to integrate any kind of secondary devices like wireless transceivers and external hard drives.

The *SoFPGA* design is made using Xilinx suite and *VHDL* code which facilitate the transfer from one *FPGA* type to another. It also make the *PR-Region* scalable depending on the *FPGA* type.

In the case of the software part, it is possible to run the system on any device developed around computer architecture. This means that it is potentially impossible to run it on smart-phone or other kind of embedded devices.

Conclusion

In this section, details concerning how to design a *RSS* are discussed. The most important notion is the limitation of the technology where systems can not evolve physically. This limitation alone has an exception, namely the possibility to use plug-able electronic boards which however still do not allow the evolution of the main board. It does not fit into the biological description of an scalable system.

Considering the hardware level of the proposed *RSS* architecture, the major advantage is the reconfigurability. The run-time reconfigurability is a key feature to push into the direction of natural systems as described earlier. It allows to design systems that physically changes in order to adapt themselves.

At the software level, the possibility to provide a system's autonomy is given. Operating system and program can be used to add automatic functions that give the autonomous behavior aspect.

Chapter 5

Analysis of the proposed RSS Architecture

Introduction

After presenting the RSS concept and the integration of the *RSN* concept, we are now able to assess its performances. PICSy and the control application are just the support to realize an RSN. As described in figure 3.13, autonomy, dynamic, interoperability are implemented using PowerPC as a controller, PR to provide hardware dynamism and Ethernet/TCP/IP as the interoperability support.

This chapter proposes to realize three objectives. The first one is to present the performances of the *RSS* concept and more precisely the performances of the *RSN* device. Different results are presented here in order to demonstrate the usefulness of combining Natural Computing with Reconfigurable Computing and the RSS Concept. The second one is to show whether Autonomy, Dynamics and Interoperability are clearly visible. In addition, it should be noted that most of this work takes place into the *PICSy* teamwork of Potsdam.

More generally, this chapter presents the details of the decentralized processing over disseminated nodes employing either GPP time slots or logic electronic hardware (PR-Region). It attempts to validate the global architectural approach of the proposed self-organized system and associated concepts, mechanisms and structures in an adapted network that was presented in the previews chapters. The objective of this chapter is also an experimental validation of all conceptual aspects studied during this thesis to make systems able to manage themselves.

5.1 Video System using RSS

5.1.1 Person Detection Scenario

To demonstrate the power of the RSS concept, an implementation of a segmentation algorithm is proposed [BZM+10]. For almost all computer vision applications, image segmentation remains a basic step. The segmentation of Kim and Chalidabhongse [KCHD05] is selected. The method is very robust and includes shadow detection. However, a real-time implementation requires hardware implementation. The central aspect of the algorithm is illustrated in figure 5.1.

An initially trained background image is compared to the current image using the three *Red Green Blue* (RGB) color parameters. To determine if a pixel is a part of the background of foreground, two parameters are used: the color distortion δ and the brightness distortion α. To subtract the background, a reference image is selected with the threshold value selected to obtain the selected detection rate. This method allows to classify current pixels.

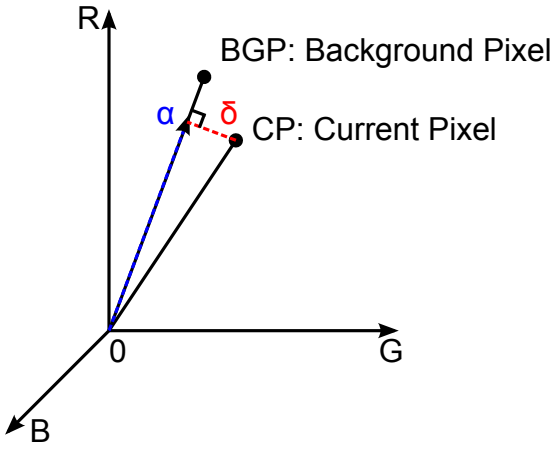


Figure 5.1: Foreground/Background computation using color distortion and brightness distortion.

The test of this algorithm is realized in three steps. First, we test the algorithm on a standard PC using *OpenCV*. The program needed approximately 23 ms to compute the foreground for a 640x480 sized image. The additional overhead to grab the image and display the result is not included. Second, we took the same implementation and tested it on our FPGA-based smart camera using the integrated *OpenCV* running on Linux, without hardware acceleration. Compared to the Intel *Core2Duo* running at 3.16, the *PowerPC* in the Virtex4 running only on 300MHz is quite slow. As expected, the pure software implementation on the smart camera could not compete with the PC. It took 1598 ms to compute the foreground for one frame. Finally we implemented the segmentation algorithm in hardware. The *DSP slices* of the Virtex4 make operations like multiplications very fast. The result is obtained in one clock cycle only. To compute δ and α for two pixels, 24 out of the 32 DSPs are required. The whole IP core uses 793 slices which is 9.28% of the FPGA. With a data width of 64 bits, two pixels can be processed per clock cycle. Due to the pipelined chain of IP cores (grabbing Bayer to RGB conversion segmenting), we have obtained the foreground for two pixels in every clock cycle (50 MHz).

The time measurements for the software-only and hardware/software implementations are summarized in table 5.1. The hardware implementation outperforms the other by a factor of 7 for the PC and 325 for the *PowerPC* respectively. The software solution on the smart camera suffers from the slow memory access of the PowerPC to load and store the image pixels. However grabbing and converting is fast because it is already made in hardware. The foreground segmentation is made independently for each pixel. Therefore the computation

System	Time (ms/frame)	frames/s
PC, $2 \times 3.2GHz$, OpenCV	≈ 23	44
Smart Camera, OpenCV in SW	≈ 1598	2
Smart Camera, HW/SW Co-Design	$3,07$	325

Table 5.1: Performance measurements of the image segmentation on different computing systems.

time for an image is linear with its size. These results are used to show that the system is really working, and able to process video data. In spite of the obviousness of these results they are useful to quantify the performance of the PICSy platform.

Figure 5.2 shows a snapshot example of video image segmentation.



Figure 5.2: An example of a segmented image

5.1.2 Reconfigurable Streaming Data Interface Controller

The second part of the results shows how to integrate partial reconfiguration into a video application. The *Streaming Data Interface* (SDI) is a set of skeleton components to allow hardware accelerators to be uniformly designed and easily integrated in the *OpenCV* environment.

It provides data and control signals to link hardware accelerators together and to access data source and data sink. A set of control and configuration registers is available to allow communication between the processors and the hardware accelerators. The major focus in developing the *SDI* controller was on the implementation of a concept which connects physical external memory to the several implemented application specific processor units, to supply them with required data and to store the processed results back to memory, without intervention of the main processor. The *SDI* is very useful in image processing where computation takes place in a streaming way. A data source supplies pixels one by one to the first processing unit in the chain. The computation takes place in the chain of *PUs* and finally the last *PU* supplies the data sink with results. Figure 5.3 shows an example of *SDI* controller usage. Incoming raw video data collected from an image sensor are placed into the *DDR* memory after a conversion from Bayer to *RGB*. The processed video is transferred to *VGA* output module.

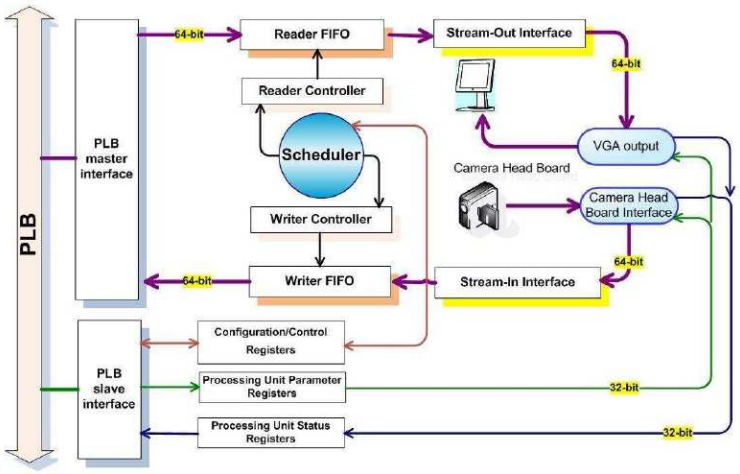


Figure 5.3: Illustration of the SDI controller usage.

To improve the design of the *SDI* system, processing modules are integrated using reconfigurable hardware logics separated from the control parts of the design. These keep exactly the same interface as the non reconfigurable *SDI* system. The *SDI* is in charge of managing the data that are processed by the hardware accelerators in the *PR-Region* while the *PowerPC* is still in charge of the PR procedure. Figure 5.4 describes the integration of partial reconfigurable hardware accelerators in the *SoC* using the *SDI*. Several reconfigurable processing units can be implemented in the region foreseen on the *FPGA* for this purpose.

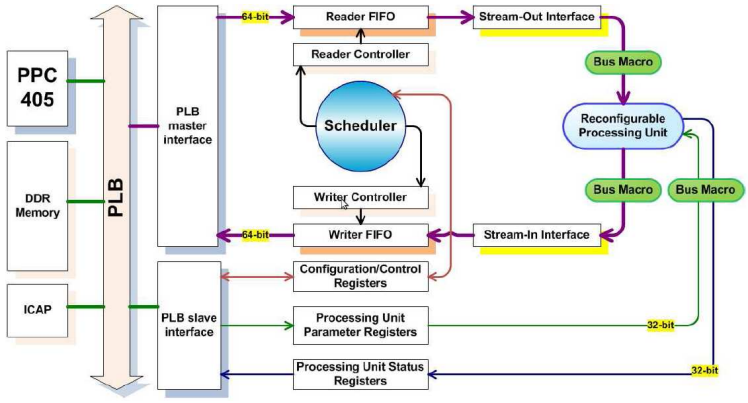


Figure 5.4: Proposed modification of the SDI controller

5.2 PR Measurement: Quantitative Analysis

This work is presented in [CZM+10] with the *RSS* concept. In this context, *Dynamic Partial Reconfiguration* is realized using a simple software application running on *Linux*. Series of 100 partial reconfigurations are made for different sizes of *PR-Region* without processing execution. The goal is to measure only the time required to reconfigure the hardware and to determine the corresponding necessary delay for it. The measurements are made at the software level. Consequently, they show the time required to execute the entire reconfiguration process, including the access to the non deterministic PLB bus and operating system scheduler. This comes with some limitations. The choice of software or hardware execution depends on the complexity of the *PR-Module* and on the volume of data that needs to be processed. These two parameters help to determine if the adaptation of the hardware with partial reconfiguration is significant or not, in terms of execution time. if the processing on a *GPP* is inferior to the time necessary to reconfigure the *FPGA*, then it is better to select *GPP* resources.

It is important to quantify qualitatively the performance of the processing unit. In addition, the design of the *SoFPGA* also introduces limitations. The size of the *PR-Region* is one of the limitations. *PR-Region* definition has an impact on the size of the partial bitstream that describes the PR-Module. Therefore, the time required to load a partial bitstream into an FPGA depends on the size of the PR Region. The results show the influence of the utilization of FPGA in an intelligent self-organized network. The main definition of the *SoC* and by consequence the size of partial bitstreams impacts the performance of the system directly. The table 5.2 gives the timing measurement for three different sizes of PR-Region. For three different sizes of reconfigurable region, the time necessary to load the bitstream is presented. As we can

see, this time stays below 50 ms for our implementation and it can be sensibly different for other implementation. Considering this measurement in a networked struture, we can consider that if the process can be executed under this 50 ms on the GPP, it is better to not choose to reconfiguration strategy. On the other hand, processes that requrie more then 50 ms can be efficiently accelerated.

	Small	Medium	Large
Number of CLB	96	192	288
Number of DSP48	8	16	24
Number of BRAM	4	8	12
Number of Clock Areas	1	2	3
Reconfig. Time (ms)	43.637	45.089	49.739
Time Std. Deviation (ms)	0.879	0.890	0.552

Table 5.2: Timing measurements for different sizes of reconfigurable regions.

5.3 RSN's behavior

This section shows how *RSNs* behave. The software activities reflect the behavior and communication between *RSNs*. Three main step can be followed: the start up of *RSN*, the local addition of tasks, the arrival of external requests.

5.3.1 Starting Up

When initialising the device, the six threads which composed the program are running and the RSNs wait for new task to handle. Figure 5.5 shows the starting activity of RSNs More precisely, it can be observed that the thread number 3 is asking for a new task to process. These tasks are inserted by a user.

```
Thread 3 LOCAL Add Task...
Thread 3: Do you want to add a new task? (1=Y or 0=N)
Thread 1 UPD Listener...
Thread 4 TCP server...
Thread 2 UPD Broadcast...
Thread 5 TCP client...
Thread 6 Allocation...
Thread 4: Socket port#12346
```

Figure 5.5: The *RSN*'s *Start Up* activities.

5.3.2 User Tasks

Figure 5.6 shows a manual tasks insertion into the *FIFO* of the *RSN*.

```
Thread 3: Do you want to add a new task? (1=Y or 0=N)
1
Thread 3: Enter task ID: (int)
1
Thread 3: Enter Bitstream name: (str)
bitstream.bit
Thread 3: Enter Application name: (str)
appli
Thread 3: Enter Data name: (str)
data.txt
Thread 3: 1, bitstream.bit, appli, data.txt, 0
Thread 3 LOCAL Add Task: New Counter Value: 1
Thread 3 LOCAL Add Task...
Thread 3: Do you want to add a new task? (1=Y or 0=N)
```

Figure 5.6: shows this insertion of a local task by a user.

Figure 5.7 shows the program information provided if an external request arrives and can be accepted. It starts with four packets that contains the task ID, bitstream file name, application file name and data file name. At this moment, the thread number 6 is aware of all information needed to decide if it can help or not. In this case, it accepts the request and the file transfer is negotiated.

After the arrival of a task into the *FIFO*, the thread in charge of the allocation Task/Resource checks if the *PR-Region* is available. In this case, *PR* command can be built from the file names and the process can be executed.

Figure 5.8 shows information sent by the user to create a new task. Once the program handles this information and depending an available resources, it creates the appropriate command to process the data.

Figure 5.9 shows an example of the control application activities. This is a description of some functions of the control application. To be more explicit on the thread functionalities, an example can show how they can handle a task. Figure 5.9 shows information that helps see thread actions.

At the beginning, the thread *UDP* listener (1) receives a task coming from another *RSN*. This task is managed by the thread allocator (6) that tries to associate it with an available resource. After the confirmation, the thread *TCP* client (5) requests the files from the requester *RSN*. When files are downloaded, they are handled by the thread allocator (6). In this case, a *PR-Region* is free, so the RSN reconfigures itself, processes the data files and sends the result back.

```
Thread 1: My address is 0.0.0.0
Thread 1: Received packet from  193.50.119.16
Thread 1: Received packet from  193.50.119.16
Thread 1: The packet is 1 byte long
Thread 1: The packet 1 contain "1"
Thread 1: The packet 1 contain "1"
Thread 1: Received packet from  193.50.119.16
Thread 1: The packet is 13 byte long
Thread 1: The packet 2 contain "bitstream.bit"
Thread 1: The packet 2 contain "bitstream.bit??"
Thread 1: Received packet from  193.50.119.16
Thread 1: The packet is 5 byte long
Thread 1: The packet 3 contain "appli"
Thread 1: The packet 3 contain "appli??"
Thread 1: Received packet from  193.50.119.16
Thread 1: The packet is 8 byte long
Thread 1: The packet 4 contain "data.txt"
Thread 1: The packet 4 contain "data.txt"
Thread 1: Origin of the request: 193.50.119.16
Thread 1 UPD Listener: New Counter Value: 1
Thread 6 Allocation: New Counter Value: 2
Thread 6: Task to allocate: 1, bitstream.bit??, appli??, data.txt from 193.50.119.16
Thread 6: New external task accepted, download files...
Thread 5: Copying from: 193.50.119.16
Thread 5: Copying from:
Thread 5: ID of the task that have been accepted: 1
Thread 5: The number of char is: 6148
Thread 5: Copying to file: bitstream.bit??
Thread 5: j= 6141
Thread 5: j= 6142
Thread 5: j= 6143
Thread 5: j= 6144
Thread 5: j= 6145
Thread 5: j= 6146
Thread 5: j= 6147
Thread 5: i= 0
Thread 5: The number of char is: 17
Thread 5: Copying to file: appli??
Thread 5: i= 0
Thread 5: The number of char is: 6
Thread 5: Copying to file: data.txt
Thread 5: i= 0
```

Figure 5.7: shows this insertion of a local task by a user.

```
Thread 6 Allocation: New Counter Value: 2
Thread 6: Task to allocate: 1, bitstream.bit, appli, data.txt from 0
Thread 6: Local PR Region Free
Thread 6: PR Command: dd if=bitstream.bit of=/dev/icap0
dd if=bitstream.bit of=/dev/icap0
```

```
Thread 6 Allocation: New Counter Value: 2
Thread 6: Task to allocate: 1, bitstream.bit, appli, data.txt from 0
Thread 6: Local GPP time slot Free
Thread 6: PR Command: ./appli data.txt
./appli data.txt
```

Figure 5.8: Build PR Command and execute it.

This last figure also helps to understand that the control application can be seen as *Peer-to-Peer* application. The capacity to be at the same time client and server, in addition to file exchange and autonomy, are characteristics of P2P application. Consequently, *RSS* can be considered as a kind of *P2P* hardware system. As explain earlier, *RSN* are able to accept or reject tasks as they want. Considering a group of many *RSNs*, it shows the feasibility of *RSE* using actual technology and based on interoperable nodes.

5.4 Network Qualitative Analysis

In this last section, a discussion about different aspects of this thesis helps to complete all information that can be reused in the research field.

```
Thread 1: Received packet from  193.50.119.16
Thread 1: The packet is 1 byte long
Thread 1: The packet 1 contain "1"
Thread 1: Received packet from  193.50.119.16
Thread 1: The packet is 13 byte long
Thread 1: The packet 2 contain "bitstream.bit"
Thread 1: Received packet from  193.50.119.16
Thread 1: The packet is 5 byte long
Thread 1: The packet 3 contain "appli"
Thread 1: Received packet from  193.50.119.16
Thread 1: The packet is 8 byte long
Thread 1: The packet 4 contain "data.txt"
Thread 1: Origin of the request: 193.50.119.16
Thread 6: Task to allocate: 1, bitstream.bit, appli, data.txt from 193.50.119.16
Thread 6: New external task accepted, download files...
Thread 5: Copying from: 193.50.119.16
Thread 5: ID of the task that have been accepted: 1
Thread 5: The number of char is: 6148
Thread 5: Copying to file: bitstream.bit
Thread 5: The number of char is: 17
Thread 5: Copying to file: appli
Thread 5: The number of char is: 6
Thread 5: Copying to file: data.txt
Thread 6: Local PR Region Free
Thread 6: PR Command: dd if=bitstream.bit of=/dev/icap0
dd if=bitstream.bit of=/dev/icap0
Thread 6: HW Processing
```

Figure 5.9: Example of external task accepted and processed by an RSN.

5.4.1 Timing Analysis

In this context of this work, measurements are focussed on timing, i.e. how long it takes to transfer data, find available resources and reconfigure hardware. It may be interesting to present an evaluation of timing required to reconfigure an FPGA region. As shown in [Wil01], there are two main solutions to measure the timing in a computer system, hardware-based measurement and software-based measurement. This last technique is the most common because it is simple and does not require expensive equipment.

The time required to reconfigure a *PR-Region*, viewed from the software point of view, is in the range of 50 ms. This value is observed for the PICSy project architecture that is described in [BZM+10] and [CZM+10]. To complete this measurement, these values can be put in perspective and used with any measurements concerning networked systems. This is possible, because the measurements are realized with the objective to show how long it takes for any standard architecture to reconfigure itself.

To measure and estimate timing in *RSS*, it is possible to reuse measurements coming from distributed computing and just add the particularity of partial reconfiguration. Because all communications are based on *TCP/IP* protocol and Ethernet, it does not make sense to measure timing again that was confirmed already known technology performances. Nevertheless, the determination of the time required to reconfigure a *PR-Region* can help to complete the information on using reconfigurable technology with *P2P* structure.

Using the software point of view (in ms) instead of the hardware point of view (in clock cycle) allows to see how reconfigurable technology impacts the performance of the computation.

The time necessary to process data on a remote RSN can be decomposed into 5 timing

steps:

$$T_{Total} = T_{Assos} + T_{Transfer1} + T_{Access} + T_{Proc} + T_{Transfer2} \tag{5.1}$$

1. T_{Assos}: Association Task/Resource
2. $T_{Transfer1}$: Files Transfer
3. T_{Access}: Resource Access
4. T_{Proc}: Processing Time
5. $T_{Transfer2}$: File Transfer

T_{Assos} can be decomposed in two cases: If a local resource is available, the association is direct and it requires the time necessary to execute one command line. If it requires external resources, T_{Assos} becomes non deterministic and it depends on the delay generated by the communications between nodes.

$$T_{Assos} \Rightarrow T_{command_execution}\ OR\ T_{com._transaction} \tag{5.2}$$

T_{Access} is the time required to access the resource, in other words the time to execute command and data register of the processor or the time required to reconfigure a PR-Region.

$$T_{Access} \Rightarrow T_{fill_reg}\ OR\ T_{PR} \tag{5.3}$$

T_{Proc} is the effective processing time that depends on the computation electronic structure and on the data quantity.

$$T_{Proc} = n.T_{execution} \tag{5.4}$$

where, n is the number of bytes and $T_{execution}$ is the time necessary to process these bytes.

Finally, $T_{Transfer1}$ and $T_{Transfer2}$ are determined by the networking parameters and the size of files that must be transferred over the network in case there are not enough local resources.

$$T_{Transfer1} = T_{bitstream_file} + T_{program_file} + T_{data_file} \tag{5.5}$$

$$T_{Transfer2} = T_{result_file} \tag{5.6}$$

They are corresponding on the TCP/IP protocol performances and they also depends on the network structure. T_{Assos},

$T_{Transfer1}$, $T_{Transfer2}$ and T_{Proc} are all observed in many different work like in [CSA00]. *RSS* introduces only the time required to reconfiguration a region. These results are presented in [CZM+10] for a specific platform.

5.4.2 Analysis of the Computation Power

The computation power of a *RSE* is not based on a fixed value that gives a simple idea of what should be the best system. Because of the dynamic evolution of the numbers of nodes during the time, it is not possible to give correct values like the equivalent of cycles or an *Instruction Per Second* (IPS) value. More useful data can be identified, namely when and where the biggest volume of resources is available for a computation problem. To do this, it is possible to count the number of nodes or computation resources at a certain moment as follow:

$$Total = \sum x_n(a_{PRR} + b_{GPP}) \tag{5.7}$$

where x_n is node number, a_{PRR} and b_{GPP} are respectively the number of *PR-Region* and the number of Processor time slot include in a single node.

The *RSN*'s control application evaluates which resource can be used to process a task by checking simple variables that are set to 1 if they are busy and 0 if they are free. The resource access is determined by the time required to reconfigure the *PR-Region* or to get a time slot access to the processor. These last parameters are not deterministic. After that, the effective processing time is a function of the quantity of data to process. Finally, the transfer file timing depends on the network characteristics.

To conclude this point, it can be noted that a main rule should be observed in any case. This rule is to prioritize the utilization of PRR. Nevertheless, hardware parallelization is often a good solution, but its limit is reached when the time to realize this parallelization becomes longer than the processing time or when the time required to reconfigure the region exceeds the time required to compute the data on a GPP.

5.4.3 Analysis of the Evolution of RSS

As a concept, the RSS can grow and evolve with the technology. It is not just the description of a fixed system and as explained by the triplet: *Autonomy, Dynamic and Interoperability*, it tries to characterize a kind of natural reconfigurable system. In the natural aspect, the evolution is at the basis of the structure. Two kinds of evolution can be discussed, the linear evolution and the rapid mutation. Linear evolution can also be called update within the computer science

vocabulary. It is a way to correct and improve the system with a few modifications and without changing the main structure of the system. Rapid upgrade can have a deep impact on the structure and exhibit huge evolution. Considering this observation and in addition to hardware and software technologies, it makes sense to say that the technology based an the *RSS* concept is also a part of the *natural* environment, in the sense that it is now possible, for the software part and the hardware part (PR), to change during the time.

Using the time parameter, it is possible to rewrite the preview equation:

$$Total = \int x_n(a_{PRR} + b_{GPP})dt \tag{5.8}$$

Now the total quantity of resources changes during the time and allows to make the system more or less powerful depending on the computation requirement. Moreover, because *RSNs* are autonomous they can evolve alone and provide help spontaneously. The collaboration between RSNs marks the apparition of *RSE*.

5.4.4 Comparison with other systems

Comparing the *RSS* concept with other concepts is difficult because *RSS* is designed to change during the time, at the micro-level and at the macro-level and regarding its reconfigurability.

A significant advantage of such systems introducing an applications that can manage *PR*, is the possibility to connect *RSNs* to the Internet network or any other network. In addition, it makes *RSNs* usable like any other computer over the existing infrastructure. It also shows that it can work independently.

By using this technology, this work shows a way to contribute and to discuss about interoperability which is one of the main goal in the communication world. *Interoperability* can be discussed in parallel with system heterogeneity. Today, a huge quantity of devices are interconnected over the Internet network. From a small smartphone to a supercomputer, a great diversity exists but most of them are designed with limited evolution capacity. The interaction between the very low level logic and the high level application allows to see the SoFPGA, the board design and the O.S. as unique layer which is already existing and that can be completed with reconfigurable technology.

In this work, *Interoperability* plays an important role at the computation level and reconfigurable logic offers a great advantage to obtain a greater computation power. Compared to simple *GPP* architecture, it is possible to implement during runtime dedicated parallel process-

ing modules that allow to process data in a limited number of clock cycles.

To allows processing either an *GPP* or *PR-Module*, programs and bitstreams are treated as a single processing package that contains all gathered information. When an *RSN* receive a package, it can decide whether to process data on available resources or not.

In spite the difficulty to quantify the performance of the RSS/PICSy, some comparisons can be presented with some similar implementations presented in chapter 2. The projects presented here are based on platform representing one node in a single level network. Three parameters present an interest, the number of PRR, the integration of the software including or not an OS and the communication capacity.

System	Board	PRR	Software	Communication
SCARS	ML501	1	Standalone App	DM2200 Wireless Module
ReCoNode	Excalibur	0	Standalone App	Custom Wire, HDLC Protocol
OverSoC	Cyclone 2	0	RTOS + Task App	No external communication
ReCoBus	Virtex-2	many	Standalone App	No external communication
RSS	PICSy	1	GNU/Linux + Ctrl App	Ethernet, (TCP – UDP)/IP

Conclusion

To conclude, the hybridization of computer systems is one of the most difficult challenges. The introduction of reconfigurable technology into the existing concepts such as distributed computing can be used to provide more parallel computing possibilities and more flexibility in the task processing. In running systems like any distributed computing network, it provides a new degree of freedom to compute data.

As a future work, run-time task pre-emption is going to be observed in order to propose an alternative solution to O.S. scheduler for reconfigurable computing. Beside this, using different kinds of architecture may be interesting. The study of heterogeneous RSE (with different RSN based on FPGA, GPU or GPP) offers the possibility to observe how they interact together. All this proposition are just an optimization of the main structure with must be improved to get the simplest architecture.

Conclusion and Future Work

The research that has been done combines them with Reconfigurable Computing and Technology with different other concept like self-organisation, inteligence. Offering the new point of view, it points out three properties: *Autonomy, Dynamic and Interoperability*.

In this work, the novelty is a new hardware design which is self-managed and adapted for FPGA based systems. The originality of our approach is that the self-management is based on decentralized control designed around a scalable communication protocol with is used on top of an existing communication standard. The basic concept of this design, as well as its main characteristics are both presented. We have defined the dynamic structures called the *Stream* which allows RSN to communicate their requests to other nodes and allows the establishment of spontaneous cooperation. These mechanisms are validated on an image-processing application through the PICSy platform. In practice, a complete RSN is designed. First, this is done using *Xilinx development boards*, and later based on the *Potsdam Intelligent Camera System* (PICSy). It is demonstrated that *Reconfigurable Computing Technology* can be completely integrated into massively used communication and computation structures. The link between low-level reconfigurable logic and high-level control application shows that the rest of the design (hardware platform, SoFPGA, OS) looks like a big middle-technology layer. Concerning this PhD contribution, the reconfigurable computing aspects and self-organisation properties were the most significant part of the implementation side and include:

- the elaboration of the FPGA configuration chain.
- the integration of partial reconfiguration in the SoFPGA design.
- the integration of the ICAP and GPIO driver.
- the development of the auto-reconfiguration application.
- the integration of the RSN node in a networked structure by developing a control application.
- the evaluation of the partial reconfiguration timing.

Moreover, the particularity of reconfiguration computing using FPGA implies a contribution to:

- the board design including schematic definition, PCB design, and circuit test and debugging.
- the video processing and hardware/software co-design (see SDI controller).
- the OS development.
- the development of the test board, in particular the FPGA's IP module and its debugging.

This implementation work must be considered with the general research work including:

- the research on the different fields and concepts related to reconfigurable computing.
- the concept development, discussion and formalization based on the ideas of my supervisor.
- the qualitative analysis of the integration of reconfigurable computing and technology into today's networked structures.
- the positioning of our work in the worldwide research activities.
- the proposition of a classification for computing paradigms and technology combinations.

To conclude, a very wide open-minded point of view is used to discuss technology. Natural Reconfigurable Computing provides an original research direction and can be associated to real hardware systems able to reconfigure themselves in a natural way. The *RSS* Concept is a pragmatic adaptive and self-organized system concept that aims at finding a new utilization and new way of thinking concerning obsolete devices and at grafting Reconfigurable Computing Technology on them. This goal is reached by integrating hardware aspects into well known software paradigms. Finally, the goal which consists of providing a degree of freedom to networked and self-organized system is reached by using intelligent auto-reconfiguration.

A video application based on Reconfigurable Computing Technology and RSN shows that it can be used for many different purposes. Timing measurements of partial reconfiguration show that reconfiguration delays have relatively few consequences from the software level and by consequence from the network level. Furthermore, the observation of communication exchanges shows that automatic and informatics systems can be seen as intelligent and communicative systems.

This integration concerns only the evolution part of the life cycle. By doing this, the RSS concept provides a solution for thinking of complex networked system as an item of Natural Reconfigurable Computing. At this level, the question of SoFPGA optimization becomes important because it is one of the main reasons for using partial reconfiguration that provides the possibility of modifying a partial reconfigurable module after design-time and directly on site

during run-time. As the pure behavioural description was the main way to design on FPGA systems, a mix of behavioural and structural descriptions could be a way to improve dynamically running functions or to integrate modules designed after the main design time.

For further research, the combination of RSS and MPSoC is a promising way to realize RSE completely on chip. Moreover, this kind of research can be a way to blur the border between *on Chip* and *off Chip* networks. Figure 1.7 shows promising technology combinations that can help to think about our future work.

As a completely dynamic solution, the platform will be used to discuss hardware/software partition, self-organization and fault tolerance. The study of the communication and the reconfiguration of a network and the reconfiguration of intelligent nodes is the main challenge of RSS and in particular of the Hardware/Software link.

On the other hand, task migration is a big part of our future work. Specially, how to optimize the processing of a task on either processor or reconfigurable module is one of our challenges. In addition, the issue of pre-emption and resource/task reassignment is a major question for both local and network level. The last point concerns the relation between O.S. scheduler and control application, scheduler is dedicated to task allocation and specially multi core processor are good example of how task can be dispatch and create a connection between our control application and the scheduler is other point is going to be study. As an ongoing work, failure-detection, fault-tolerance mechanisms and learning mechanisms are about to be considered. Morevore, learning from past experience and remember effective responses are specifically targeted.

Bibliography

[Ash47] William R. Ashby. Principle of the self-organisation dynamic system. *Journal of General Psychology*, Volume: 37:pages: 125–128, 1947.

[BAM+05] Christophe Bobda, Ali Ahmadinia, Mateusz Majer, Jurgen Teich, Sandor Fekete, and Jan van der Veen. Dynoc: A dynamic infrastructure for communication in dynamically reconfigurable devices. *Field Programmable Logic and Applications, 2005. International Conference on*, Volume: 2005:Pages: 153 – 158, 2005.

[BCM+09] Christophe Bobda, Kevin Cheng, Felix Mühlbauer, Klaus Drechsler, Jan Schulte, Dominik Murr, and Camel Tanougast. Enabling self-organization in embedded systems with reconfigurable hardware. *International Journal of Reconfigurable Computing*, Volume 2009:Article ID 161458, 9 pages, 2009.

[BZM+10] Christophe Bobda, Ali Akbar Zarezadeh, Felix Mühlbauer, Robert Hartmann, and Kevin Cheng. Reconfigurable architecture for distributed smart cameras. *Proceedings of the Engineering of Reconfigurable Systems and Algorithms (ERSA)*, Volume 2010:–, 2010.

[CH02] Katherine Compton and Scott Hauck. Reconfigurable computing: a survey of systems and software. *ACM Comput. Surv.*, 34:171–210, June 2002.

[CSA00] Neal Cardwell, Stefan Savage, and Thomas Anderson. Modeling tcp latency. *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, Volume 3:Pages 1742 – 1751, 2000.

[CZM+10] Kevin Cheng, Ali Akbar Zarezadeh, Felix Muhlbauer, Camel Tanougast, and Christophe Bobda. Auto-reconfiguration on self-organized intelligent platform. *Inproceedings of NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2010)*, Volume 2010:Pages 309–316, 2010.

[DH05] Tom DeWolf and Tom Holvoet. Emergence versus self-organisation: Different concepts but promising when combined. *Engineering Self-Organising Systems*, Volume 3464:Pages 1 to 15, 2005.

[Div02] Divers. *Natural Computing*. Springer, 2002.

[Doo96] Kevin Dooley. Complex adaptive systems: A nominal definition. *The Chaos Network*, Volume: 8:Pages: 2–4, 1996.

[DPP02] Matthias Dyer, Christian Plessl, and Marco Platzner. Partially reconfigurable cores for xilinx virtex. *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream*, 2438:81–92, 2002.

[dV97] G. Van de Vijter. *Emergence et explication*, volume 25. Intellectica, 1997.

[DW02] Matthias Dyer and Marco Wirz. *Reconfigurable System on FPGA*. PhD thesis, Ecole Polytechnique fédérale de Zurick, 2001/2002.

[Edm95] Bruce Edmonds. What is complexity? the philosophy of complexityper se with application to some examples in evolution. *The volution of Complexity*, Volume: 14, 1995.

[Est60] Gerald Estrin. Organization of computer systems: the fixed plus variable structure computer. *Papers presented at the May 3-5, 1960, western joint IRE-AIEE-ACM computer conference*, 1460365:33–40, 1960.

[FZRL08] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. *Grid Computing Environments Workshop, 2008. GCE '08*, Volume 2008:Pages: 1 – 10, 2008.

[Gal08] Thomson Gale. *International Encyclopedia of the Social Sciences*. Thomson Gale, 2008.

[Ger05] Carlos Gershenson. A genral methodology for designing self-organizing sytems. *CoRR*, 12, 2005.

[Gol99] J. Goldstein. *Emergence as a Construct: History and Issues*, volume 1.1. Ablex Publishing Corporation, 1999.

[HA10] Yutao He and Mohammad Ashtijou. iboard: A highly-capable, high-performance, reconfigurable fpga-based building block for flight instrument digital electronics. *Adaptive Hardware and Systems (AHS), 2010 NASA/ESA Conference on*, Volume: 2010:Pages: 73–74, 2010.

[Har01] Reiner Hartenstein. A decade of reconfigurable computing: a visionary retrospective. *Field Programmable Logic and Applications (FPL)*, Volume: 2001:Pages: 642–649, 2001.

[HBB04] M. Huebner, T. Becker, and J. Becker. Real-time lut-based network topologies for dynamic and partial fpga self-reconfiguration. *Brazilian Symposium on Integrated Circuits and Systems Design (SBCCI)*, Volume: 2004:Pages: 28–32, 2004.

[Hey89] F. Heylighen. Self-organisation, emergence and the architecture of complexity. *In Proceedings of the 1st European Conference on System Science*, Volume: 18:pages: 23–32, 1989.

[Hey03] Francis Heylighen. *The science of self-organisation and adaptivity*. Knowledge Management, Organisational Intelligence and Learning and Complexity in: Encylopedia of life Systems, 2003.

[HJ01] Francis Heylighen and Cliff Joslyn. *Cybernetics and second-order Cybernetics*, volume 15. Encyclopedia of PHysical Science and Technology (3rd edition), 2001.

[HKT03] Christian Haubelt, Dirk Koch, and Jurgen Teich. Reconet: Modeling and implementation of fault tolerant distributed reconfigurable hardware. *Integrated Circuit Design and System Design, Symposium on*, 0:343, 2003.

[HLTP02] Edson L. Horta, John W. Lockwood, David E. Taylor, and David Parlour. Dynamic hardware plugins in an fpga with partial run-time reconfiguration. *Proceedings of the 39th annual Design Automation Conference*, Volume: 2002:343–348, 2002.

[HMU01] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2001.

[Hol98] J.H. Holland. *Emergence: From chaos to order*. Addison Wesley, 1998.

[HUKB04] M. Huebner, M. Ullmann, A. Klausmann, and J. Becker. Scalable applications-dependent network on chip adaptivity for dynamical reconfigurable real-time systems. *Field Programmable Logic and Applications (FPL)*, Volume: 2004:Pages: 1037–1041, 2004.

[Inc02] Xilinx Inc. *Virtex 2.5 V Programmable Gate Arrays*, 2002.

[Inc09] Xilinx Inc. *Virtex-6 FPGA Configurable Logic Block*, 2009.

[Inc10a] Xilinx Inc. *Partial Reconfiguration User Guide*, 2010.

[Inc10b] Xilinx Inc. *Virtex-5 FPGA Configuration User Guide*, 2010.

[Jov09] Slavisa Jovanovic. *Architecture Reconfigurable de Système Embarqué Auto-Organisé*. PhD thesis, Université Henry Pointcarré de Nancy, 2009.

[JTBW09] S. Jovanović, C. Tanougast, C. Bobda, and S. Weber. Cunoc: A dynamic scalable communication structure for dynamically reconfigurable fpgas. *Microprocessors and Microsystems*, Volume 33, Issue 1:24–36, 2009.

[JTW08] S. Jovanovic, C. Tanougast, and S. Wber. A new high performance scalable dynamic interconnection for fpga based reconfigurable systems. *In Proceedings of the Application-Specific Systems, Architecture and Processors (ASAP'08)*, Volume:2008:Pages: 61–66, 2008.

[KBT08] Dirk Koch, Christian Beckhoff, and Jurgen Teich. Recobus-builder - a novel tool and technique to build statically and dynamically reconfigurable systems for fpgas. *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, Volume: 2008:Pages: 119 – 124, 2008.

[KCHD05] Kyunggnam Kim, Thanarat H. Chalidabhongse, David Harwood, and Larry Davis. Real-time foreground-background segmentation using codebook model. *Real-Time Imaging, Special Issue on Video Object Processing*, Volume 11, Issue 3:172–185, 2005.

[Koc09] Dirk Koch. *Architectures, Methods, and Tools for Distributed Run-time Reconfigurable FPGA-based Systems*. PhD thesis, Universität Erlangen-NÜrnberg, 2009.

[Loc01] John W. Lockwood. Evolvable internet hardware platforms. *Evolvable Hardware, NASA/DoD Conference on*, 0:0271, 2001.

[LPC07] Marco Lanuzza, Stefania Perri, and Pasquale Corsonello. Mora: a new coarse-grain reconfigurable array for high throughput multimedia processing. *Proceedings of the 7th international conference on Embedded computer systems: architectures, modeling, and simulation*, Volume: 2007:Pages: 159–168, 2007.

[LSSP02] Stefan M. Larson, Christopher D. Snow, Michael Shirts, and Vijay S. Pande. Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology. *eprint arXiv:0901.0866*, —:—, 2002.

[MCM+04] Leandro Möller, Ney Calazans, Fernando Moraes, Eduardo Brião, Ewerson Carvalho, and Daniel Camozzato. Fipre: An implementation model to enable self-reconfiguration applications. *Field Programmable Logic and Applications (FPL)*, Volume: 2004:Pages: 1042–1046, 2004.

[MFH+03] S. Mostefaoui, N. Foukia, S Hassas, G. Di Marzo Serugendo, C. Van Aart, and A. Karageorgos. *Self-organisation applications: A survey*, volume 18. 2003.

[MHV+09] Benoit Miramond, Emmanuel Huck, Francois Verdier, Bertrand Granado, Thomas Lefebvre, Mehdi Aichouch, Jean Christophe Prevotet, Yaset Oliva, Daniel Chillet, and Sebastien Pillement. Oversoc: A framework for the exploration of rtos for rsoc platforms. *International Journal of Reconfigurable Computing*, Volume 2009:Article ID 450607, 22 pages, 2009.

[MMB+04] T. Marescaux, V. Mignolet, A. Bartic, W. Moffat, P. Avasare, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins. Run-time support for heterogeneous multitasking on reconfigurable socs. *Integration The VLSI Journal*, Volume: 38 (1):Pages: 107–130, 2004.

[MMDB07] Dominik Murr, Felix Muhlbauer, Falko Dressler, and Christophe Bobda. Utilizing reconfigurable hardware to optimize workflows in networked nodes. *Embedded System Design: Topics, Techniques and Trends*, Volume: 231/2007:Pages: 373–386, 2007.

[MMP+03] Fernando Gehm Moraes, Daniel Mesquita, José Carlos Palma, Leandro Möller, and Ney Calazans. Development of a tool-set for remote and partial reconfiguration of fpgas. *Design, Automation, and Test in Europe Conference and Exhibition (DATE)*, Volume:2003:Pages: 1122–1123, 2003.

[NP77] G Nicolis and Ilya Prigogine. *Self-organisation in nonequilibrium systems*, volume 15. John Wiley and Sons, 1977.

[Ode02] J. Odell. *Objects and Agent Compared*, volume 2. Technology, 2002.

[PHF07] G. Pezzulo, J. Hoffmann, and R. Falcone. Anticipation and anticipatory behavior. *Cognitive Processing*, Volume 8(2)):Pages 67–70, 2007.

[Ros85] R. Rosen. *Anticipatory Systems: Philosophical, Mathematical, and Methodology*. Pergamon, 1985.

[Ser04] G.D.M. Serugendo. *Engineering Self-Organising Systems: Nature-Inspired Approach to Software Engineering*. Springer, 2004.

[SGM+97] Moshe Sipper, Maxime Goeke, Daniel Mange, Andre Stauffer, Eduardo Sanchez, and Marco Tomassini. The firefly machine: Online evolware. *IEEE International conference on Evolutionary Computation*, Volume: 1997:Pages: 181–186, 1997.

[SJAS08] Adarsha Sreeramareddy, Jeff G. Josiah, Ali Akoglu, and Adrian Stoica. Scars: Scalable self-configurable architecture for reusable space systems. *Adaptive Hardware and Systems, NASA/ESA Conference on*, 0:204–210, 2008.

[SSH+99] Eduardo Sanchez, Moshe Sipper, Jacques-Olivier Haenni, Jean-Luc Beuchat, André Stauffer, and Andrès Perez-Uribe. Static and dynamic configurable systems. *IEEE Transactions on Computer*, Volume: 48:pages: 556–563, 1999.

[TTL05] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: The condor experience. *Concurrency and Computation: Practice and Experience*, Volume 17:Pages 323–356, 2005.

[Vis10] Gianfranco Visentin. Needs for adaptive hardware in space robotics at esa. *Adaptive Hardware and Systems (AHS), 2010 NASA/ESA Conference on*, 2010.

[Wal92] Mitchell Waldrop. *Complexity: The Emerging Science At The Edge Of Order And Chaos*. Simon and Schuster Paperbacks, 1992.

[WB04] John A. Williams and Neil W. Bergmann. Embedded linux as a platform for dynamically self-reconfiguring systems-on-chip. *The International Conference on Engineering of Reconfigurable Systems and Algorithms*, Volume 2004:Pages 163–169, 2004.

[Wil01] Carey Williamson. Internet traffic measurement. *Internet Computing, IEEE*, Volume: 5 Issue: 6:Pages: 70 – 74, 2001.

[WP04] H. Walder and M. Platzner. A runtime environement for reconfigurable hardware operating systems. *Field Programmable Logic and Applications (FPL)*, Volume: 2004:Pages: 831–835, 2004.

Annexe

Publications

International Journal

C. Bobda, K. Cheng, F. Mülhbauer, K. Drechsler, J. Schulte, D. Murr and C. Tanougast, *Enabling Self-organisation in Embedded Systems with Reconfigurable Hardware*, International Journal of Reconfigurable Computing, 2009, doi:10.1155/2009/161458, pages 109-117.

International Conferences

K. Cheng, A. A. Zarezadeh, F. Mülhbauer, C. Tanougast and C. Bobda, *Auto-reconfiguration on Self-organised Intelligent Platform*, Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS), 2010, Volume 2010, pages 316-323.

C. Bobda, A. A. Zarezadeh, F. Mülhbauer, R. Hartmann and K. Cheng, *Reconfigurable Architecture for Distributed Smart Cameras*, Proceedings of the Engineering of Reconfigurable Systems and Algorithms (ERSA), 2010, Volume 2010, pages 166,175.

National Workshops (France)

K. Cheng, C. Tanougast, C. Bobda, A. Dandache, *Reconfigurable Self-organised Systems*, 4ème Colloque National du GDR System-on-Chip and System-on-Package (SoC-SiP), Juin 2010, ENSEA, Cergy.

K. Cheng, C. Bobda, C. Tanougast, A. Dandache, *Reconfigurable Hardware for a Network of Self-organised Nodes*, 3ème Colloque National du GDR System-on-Chip and System-on-Package (SoC-SiP), Juin 2009, Centre Scientifique d'Orsay, Orsay.

Reconfigurable Natural Computing World

Figures 5.10 5.11 shows an historical view of concepts related to this work. The original picture is from Brian Castellani, Sociology and Complexity Science, Kent State University, Kent Ohio, 2011. As a GNU free documentation, I completed it with Reconfigurable concept, technology and integration.

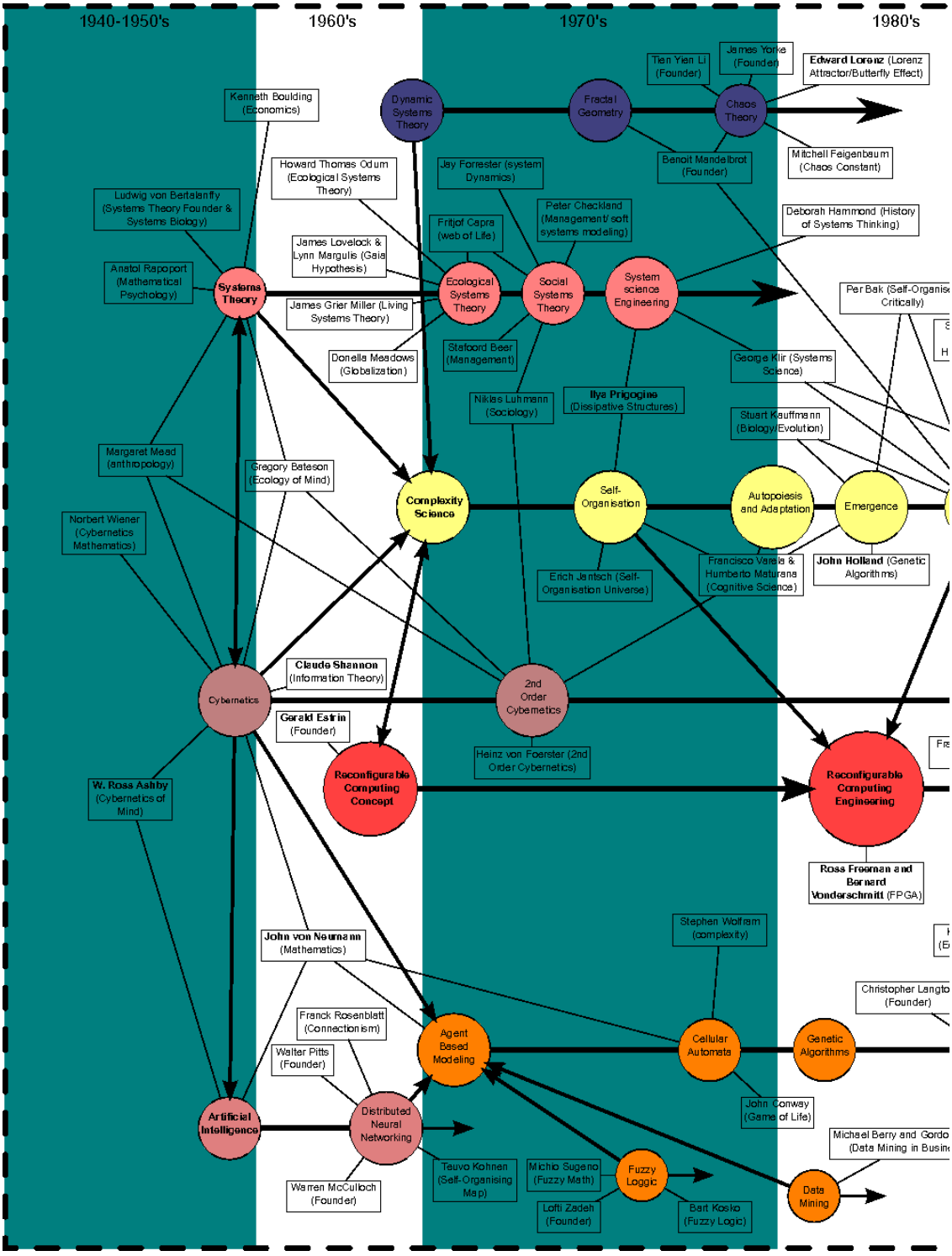


Figure 5.10: Map of subjects connections

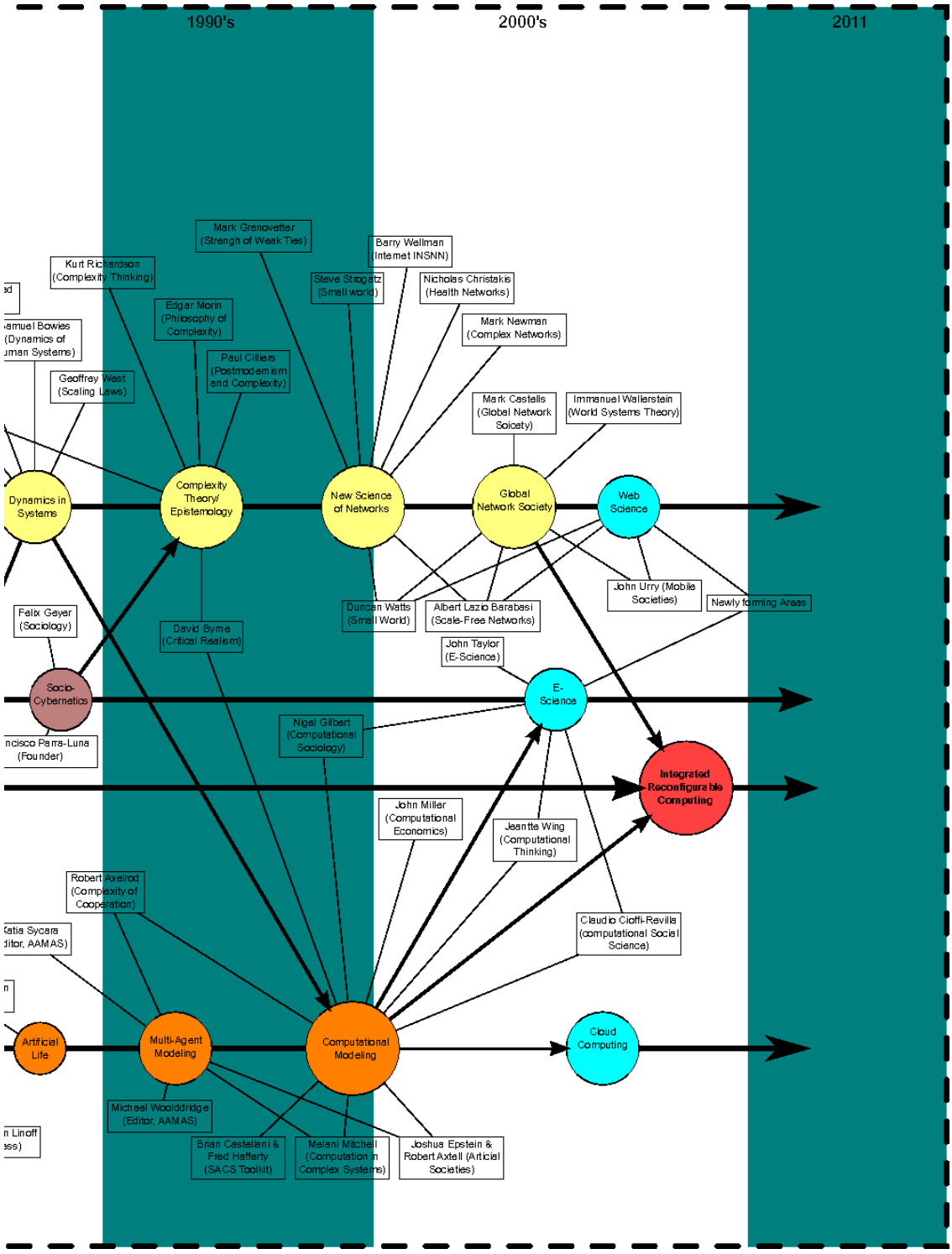


Figure 5.11: Map of subjects connections

Abstract

Increasing needs of computation power, flexibility and interoperability are making systems more and more difficult to integrate and to control. The high number of possible configurations, alternative design decisions or the integration of additional functionalities in a working system cannot be done only at the design stage any more. In this context, where the evolution of networked systems is extremely fast, different concepts are studied with the objective to provide more autonomy and more computing power. This work proposes a new approach for the utilization of reconfigurable hardware in a self-organised context. A concept and a working system are presented as *Reconfigurable Self-Organised Systems* (RSS). The proposed hardware architecture aims to study the impact of reconfigurable FPGA based systems in a self-organised networked environment and partial reconfiguration is used to implement hardware accelerators at runtime. The proposed system is designed to observe, at each level, the parameters that impact on the performances of the networked self-adaptive nodes. The results presented here aim to assess how reconfigurable computing can be efficiently used to design a complex networked computing system and the state of the art allowed to enlighten and formalise characteristics of the proposed self-organised hardware concept. Its evaluation and the analysis of its performances were possible using a custom board: the *Potsdam Intelligent Camera System* (PICSy). It is a complete implementation from the electronic board to the control application. To complete the work, measurements and observations allow analysis of this realisation and contribute to the common knowledge.

Résumé

Afin de répondre à une complexité croissante des systèmes de calcul, de nouveaux paradigmes architecturaux basés sur des structures auto-adaptatives et auto-organisées sont à élaborer. Ces derniers doivent permettre la mise à disposition d'une puissance de calcul suffisante tout en bénéficiant d'une grande flexibilité et d'une grande adaptabilité, cela dans le but de répondre aux évolutions des traitements distribués caractérisant le contexte évolutif du fonctionnement des systèmes. Ces travaux de thèse proposent une nouvelle approche de conception des systèmes communicants, auto-organisés et auto-adaptatifs basés sur des nœuds de calcul reconfigurable. Autrement dit, ces travaux proposent un système matériel autonome et intelligent, capable de déployer et de redéployer ses modules de calcul, en temps réel et en fonction de la demande de traitement et de la puissance de calcul. L'aboutissement de ces travaux se traduit par la réalisation d'un *Système Auto-organisé Reconfigurable* (SAR) basé sur la technologie FPGA. L'architecture auto-adaptative proposée permet d'étudier l'impact des systèmes reconfigurables dans une structure distribuée et auto-organisée. Le système est réalisé pour étudier, à chaque niveau, les paramètres qui influencent les performances globales d'un réseau de calcul évolutif. L'étude de l'état de l'art a permis la mise en perspective et la formalisation des caractéristiques du concept d'auto-organisation matérielle proposé ainsi qu'une évaluation et une analyse de ces performances. Les résultats de ces travaux montrent la faisabilité d'un système complexe de calcul distribué dont l'intelligence repose sur les interactions des éléments reconfigurables le constituant.