D 5.3 *Report on the evaluation of the network-level testing and/or (re-)configuration strategy.*

Due date of deliverable: Month 36
Actual submission date: Month 36

Organization name of lead beneficiary for this deliverable: UNIFE
Contributing partners to this deliverable: UNIFE, INOCS, UPV, SIMULA
Work package contributing to the Deliverable: WP5

<table>
<thead>
<tr>
<th>Dissemination Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU Public</td>
</tr>
<tr>
<td>PP Restricted</td>
</tr>
<tr>
<td>RE Restricted</td>
</tr>
<tr>
<td>CO Confidential</td>
</tr>
</tbody>
</table>

**APPROVED BY:**

<table>
<thead>
<tr>
<th>Partners</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>All partners</td>
<td>January 25th, 2013</td>
</tr>
</tbody>
</table>
INDEX

1 Introduction 5

2 FPGA Platform 7

3 The System Under Test 8
   3.1 Basic components: the on-chip network 11
      3.1.1 The Network Interfaces 12
      3.1.2 The switch 13
   3.2 Basic components: the supervision subsystem 18
   3.3 Basic components: the reconfiguration algorithm 19
   3.4 The GUI infrastructure 20
   3.5 The application 22
   3.6 The physical platform implementation 24

4 Validating Built-in Self-Testing and NoC configuration 25
   4.1 Protocol for BIST notification and configuration 27

5 Validating Fault Detection and NoC Reconfiguration 28
   5.1 Protocol for transient notification and reconfiguration 30

6 Validating NoC Virtualization 31

7 Conclusions 33
ABSTRACT

Today the converging trend toward multifunction integrated architectures is slowed down by the lack of a proper runtime reconfiguration framework of the on-chip interconnect. A runtime reconfiguration is needed whenever the occurrence of events at runtime causes the need for a different resource allocation, such as in the cases for graceful degradation of system performance, power management, thermal control, etc. The NaNoC project has developed design methods to introduce such a dynamism into the on-chip network. This deliverable reports about the prototyping of such design methods on a Xilinx Virtex-7 FPGA. Boot-time testing and configuration, runtime detection of faults, runtime reconfiguration of the routing function, dynamic virtualization of the interconnect fabric are especially validated on the FPGA prototype, where a 4x4 multi-core system has been implemented and managed. The advanced form of platform control is achieved via hardware/software co-design and co-optimization.
GLOSSARY

AMBA: Advanced Microcontroller Bus Architecture
AHB: Advanced High-Performance Bus
AXI: Advanced Extensible Interface
BISD: Built-In Self-Diagnosis
BIST: Built-In Self-Test
BRAM: Block Random Access Memory
CMP: Chip Multi-Processor
DC_FIFO: Dual-Clock First In First Out
DMR: Dual Modular Redundancy
DRC/TRC: Dual/Two-Rail Checker
DUT: Device Under Test
FIFO: First In First Out
FIFO: First In First Out
FSM: Finite-State Machine
GALS: Globally Asynchronous Locally Synchronous
GPIO: General Purpose Input Output
GPU: Graphics Processing Unit
HDL: Hardware description language
ID: Identifier
JTAG: Joint Test Action Group
LBDR: Logic-Based Distributed Routing
LFSR: Linear Feedback Shift Register
MISR: Multiple Input Signature Register
MPSoC: Multi-Processor System-on-Chip
NI: Network Interface
NoC: Network-on-Chip
OSR: Overlapped Static Reconfiguration
PHY: Physical Layer
SEU: Single Event Upset
SoC: System-on-Chip
TMR: Triple Modular Redundancy
TPG: Test Pattern Generator
USB: Universal serial bus
XPS: Xilinx Platform Studio
1 Introduction

NoC design principles have recently reached a stage where they start to stabilize, in correspondence to their industrial uptake. In fact, NoCs are an indisputable reality since they implement the communication backbone of virtually all large-scale system-on-chip (SoC) designs in 45nm and below.

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

In the highly parallel landscape of modern embedded computing platforms, the system interconnect serves as the framework for platform integration and is therefore key to materializing the needed flexibility and reliability properties of the system as a whole. Therefore, time has come for a major revision of current NoC architectures in the direction of increased reconfigurability and reliability.

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

Finally, there is an increasing need in embedded systems for implementing multiple functionalities upon a single shared computing platform. The main motivation for this are the constraints set for systems size, power consumption and/or weight. This forces tasks of different criticality to share resources and interfere with each other. Integration of multiple software functions on a single multi- and many-core processor (multifunction integration) is the most efficient way of utilizing the available computing power. For a mixed-criticality multifunction integration, the NoC should be augmented to support partitioning and isolation, so that software functions can be protected from unintended interferences coming from other software functions executing on the same hardware platform. This feature is a key enabler for the virtualization of embedded systems, that is, an effective and clean way of isolating applications from hardware.

This deliverable reports on the first-time prototyping of a Network-on-Chip capable of supporting all of the advanced features described above, and represents a prove of the success of the NaNoC project. The presented prototype builds on the GPNaNoC switch presented in deliverable D5.1 and raises the level
of abstraction to the network as a whole. Then, it validates the (re-) configuration capabilities that preserve safe network operation in the presence of wanted (e.g., virtualization) and unwanted (e.g., manufacturing defects, intermittent faults) effects. The prototyping platform is represented by the Xilinx Virtex-7 evaluation board named VC707, described in Section 2. The prototyped system implemented inside the FPGA is a homogeneous multicore processor, which resembles programmable hardware accelerators of hierarchical, high-end embedded systems, or basic computation clusters of many-core processors. The validated design methods include:

- boot-time testing and diagnosis of the 4x4 2D mesh NoC, targeting permanent faults;
- switch-level and network-level fault-tolerance, targeting transient faults and intermittent faults (i.e., those faults that rapidly anticipate the breakdown of links or switch components);
- runtime reconfiguration of the network routing function, with logic-based distributed routing as the underlying routing mechanism. The validated reconfiguration procedures are twofold: at boot-time, without background traffic, and at runtime, with background traffic.
- Dynamic virtualization, i.e., partitioning of the whole NoC into isolated partitions running different applications.

As such, this deliverable validates:

- the design methods for supporting NoC static irregularities developed in WP1 (routing methods, testing and diagnosis methods);
- the design methods for supporting dynamically virtualized NoCs developed in WP2 (error detection and signaling mechanisms, runtime reconfiguration methods, virtualization methodology);
- methodologies for seamless integration of NoC topologies within IP cores, from WP3.

The remaining technical WPs (the bulk of WP3, and WP4) will be validated in deliverable D5.2.

Looking beyond the validation goal, as this Deliverable introduces a fully-functional, innovative FPGA platform, it also represents a powerful exploitation vehicle that will be leveraged as such by the Consortium (see D6.1). For example, INOCS will use the design as-is to claim FPGA-level validation of part of its RTL portfolio, namely the AXI and AHB NIs and their interoperability; derivatives of this platform based on all-INOCs IP are possible and planned. UNIFE is also planning to reuse this work heavily for internal training and for both teaching and research activities, on NoCs and further. The Consortium believes that this platform could serve as a product seed for concrete applications especially in reliability-conscious industries (aerospace, military, healthcare, etc.) and the partners commit to looking for possible commercial outcomes, direct or indirect. In this perspective, numerous of the salient features of the design could certainly be tailored to the specific applications, e.g. by tuning up or down the level of fault tolerance vs. the performance and area overheads.
The target system to prototype is overly complex, hence calling for high-end FPGAs and development boards, not to incur integration capacity limits.

The Virtex-7 FPGA VC707 Evaluation Kit was selected for our task. It is a full-featured, highly-flexible, high-speed serial base platform using the Virtex-7 XC7VX485T-2FFG1761C and includes basic components of hardware, design tools, IP, and pre-verified reference designs for system designs that demand high-performance, serial connectivity and advanced memory interfacing. The included pre-verified reference designs and industry-standard FPGA Mezzanine Connectors (FMC) allow scaling and customization with daughter cards. The XC7VX485T FPGA features 485760 logic cells, 75900 CLB slices, 2800 DSP slices, 37080 kb of block RAM, 14 total I/O banks and 700 max. user I/O.

The key features of the evaluation board (see Figure 1) are as follows:

- **GA VC707 Evaluation Kit**: ROHS compliant VC707 kit including the XC7VX485T-2FFG1761 FPGA
- **Configuration**: Onboard JTAG configuration circuitry to enable configuration over USB, JTAG header provided for use with Xilinx download cables such as the Platform Cable USB II, 128MB (1024Mb) Linear BPI Flash for PCIe Configuration, 16MB (128Mb) Quad SPI Flash.
- **Memory**: 1GB DDR3 SODIMM 800MHz / 1600Mbps, 128MB (1024Mb) Linear BPI Flash for PCIe Configuration, SD Card Slot, 8Kb IIC EEPROM.
- **Communication and Networking**: GigE Ethernet RGMII/GMII,SGMII, SFP+ transceiver connector, GTX port (TX, RX) with four SMA connectors, UART To USB Bridge, PCI Express x8 gen2 Edge Connector (lay out for Gen3).
- **Display**: HDMI Video OUT, 2 x16 LCD display, 8X LEDs.
- **Expansion Connectors**: FMC1 - HPC (8 XCVR, 160 single ended or 80 differential, user-defined pins), FMC2 - HPC (8 XCVR, 116 single ended or 58 differential user-defined pins), Vadj supports 1.8V, IIC.
Clocking: Fixed Oscillator with differential 200MHz output used as the system clock for the FPGA, programmable oscillator with 156.250 MHz as the default output, default frequency targeted for Ethernet applications but oscillator is programmable for many end uses, differential SMA clock input, differential SMA GTX reference clock input, Jitter attenuated clock used to support CPRI/OBSAI applications that perform clock recovery from a user-supplied SFP/SFP+ module.

Control and I/O: 5X Push Buttons, 8X DIP Switches, Rotary Encoder Switch (3 I/O), AMS FAN Header (2 I/O).

Power: 12V wall adapter or ATX, Voltage and Current measurement capability.

Debug and Analog Input: 8 GPIO Header, 9 pin removable LCD, Analog Mixed Signal (AMS) Port.

3 The System Under Test

In order to integrate and demonstrate numerous of the NaNoC developments over the years, an ambitious Virtex 7 FPGA-based platform was conceived for this Deliverable. The high-level view of the design can be found in Figure 2.

The system comprises a large number of components within the FPGA. As can be seen on the left side of the diagram, a relatively standard Xilinx subsystem is instantiated first; this comprises an AXI interconnect linking together a MicroBlaze (to run the supervision software), a small memory and an external
DRAM controller, and several peripheral controllers required to run software on the MicroBlaze and to communicate with a laptop.

The right side of the diagram depicts the components that were designed during the NaNoC project, including some that were specially developed for this Deliverable. This part of the system is the “Device Under Test” (DUT) of the platform, whose functionality is to be verified. It comprises mainly:

- The main NoC, built as a 4x4 mesh of GPNaNoC switches as developed by UNIFE.
- The dual NoC, built as a chain that follows the topology of the main NoC. The dual NoC is in charge of configuring the main NoC and of collecting status information (e.g. fault detections) from the main NoC. This component was developed by UNIFE.
- At each node of the main NoC (see also Figure 5), a MicroBlaze and a memory (by Xilinx) are connected to the switch by means of Network Interfaces designed by INOCS. To provide extra validation of Deliverable D3.2, the MicroBlaze NI has an AMBA AXI NI while the memory is given an AMBA AHB NI.
- Two special blocks, based on INOCS AXI NIs, have been designed to connect the dual NoC to the supervision subsystem. These blocks allow the supervision MicroBlaze to receive notifications by the dual NoC, and to reprogram it.
- A sniffer module, developed by INOCS, monitors traffic along all links of the main NoC mesh, computing link utilization. It is designed so that the supervision subsystem can probe it at regular intervals and transfer its contents towards a user’s laptop.
- A fault injection module has been instantiated along a mesh link. This simple module, connected to a physical button on the FPGA board, provides a method to inject faults on that link to test the platform’s fault tolerance and the NoC reconfiguration capability.

To build this platform, we proceed in steps (Figure 3). First, we instantiate within Xilinx Platform Studio (XPS) a complete design comprising all the supervision subsystem, the 16 additional MicroBlazes, and the corresponding 16 memories (Figure 4). At this stage, no NoC is instantiated yet. Using XPS for this task allows us to efficiently connect and configure all the Xilinx blocks, and facilitates the instantiation of the toplevel HDL files. Additionally, this makes it possible to subsequently load the applications into all 17 MicroBlazes’ memories, and to debug those processor step-by-step, directly through the Xilinx toolchain, which is Eclipse-based. After the first pass of synthesis, however, we remove from the design the Xilinx AXI subsystem which is connecting the 16 additional MicroBlazes and memories, and swap in the NoC (main and dual) in its place. We then proceed to finish the implementation flow within Xilinx ISE by performing mapping, placement and routing, and generating the final bitstream.

We leverage some key features of the Virtex 7 board, apart from the FPGA chip. The on-board DRAM is used to provide sufficient space for the software
Figure 3: Design flow for platform implementation.

running on the supervision MicroBlaze to work. Physical buttons and switches of the board are connected to an on-chip GPIO controller to allow the user to interact with the platform. Finally, a laptop can be connected to the board by means of two cables to monitor the platform’s operation; one cable carries serial port signals (piggybacked onto a USB port) and the other carries JTAG signals (also piggybacked onto a USB port). The former is used to read the board’s outputs, while the latter allows for programming the board and interactively debugging the on-FPGA MicroBlazes. An Ethernet cable had initially been considered instead of the serial interface, in light of its higher throughput, but the Ethernet PHY of the board was found to be defective.

Custom-written software runs in three locations of the system: on the supervision MicroBlaze, on the 16 MicroBlazes connected to the main NoC, and on the external laptop.

- The software on the supervision MicroBlaze is tasked with oversight of the main NoC and data NoC, with regular polling of the Traffic Sniffers, and with interfacing with the external world through the serial interface. The code was co-developed by UPV, UNIFE and INOCS. UPV took care of all aspects related to routing tables and reconfiguration policies; UNIFE contributed diagnostic code and reconfiguration mechanisms to drive the dual NoC; INOCS was in charge of Traffic Sniffer monitoring, software testing and platform integration.

- The 16 MicroBlazes connected to the mesh run micro-benchmarks originally developed by INOCS and upgraded by UNIFE. These micro-benchmarks have the main role of generating traffic on the mesh, so that the various platform features can be tested. Real functional behaviour
was implemented: the nodes perform pipelined matrix multiplications, exchanging data in producer-consumer fashion. More advanced applications could not be implemented due to the lack of I/O interfaces on these nodes and due to lack of memory to instantiate a full C library.

- The user’s laptop is connected to the board through a JTAG-over-USB cable and a serial-over-USB cable. The former can be leveraged mainly by the Xilinx toolchain, allowing for board programming and debugging. The latter is monitored by a UPV-designed GUI that displays in real-time the platform status and link utilization. This GUI allows the user to analyze the impact of running different software on the system and the behaviour upon fault injection or virtualization implementation.

3.1 Basic components: the on-chip network

A 4x4 mesh with one core and one memory per switch has been chosen as target on-chip network of the FPGA platform. In particular, Figure 5 represents the basic components instantiated to realize the 4x4 mesh. A MicroBlaze and a memory are connected to each switch through two Network Interfaces. Finally, a sniffer is placed on each bidirectional network link to monitor the network traffic. The sniffers collect information about the traffic crossing the switch-to-switch and NI-to-switch links and deliver such information to the global manager (i.e., the supervision MicroBlaze).

Both the NIs and the switches have been designed ad-hoc to support the target on-chip network where fault-tolerance, testing capability and reconfigurability features are guaranteed.
Figure 5: Basic components of the on-chip network.

Note that the MicroBlaze also includes a directly-connected BRAM of 128 kB (not shown in the figure) to store its application software; loading the binary image of the application into the AHB memory would be unnecessarily problematic from the toolchain viewpoint. However, we explicitly use the AHB memory as storage and for inter-processor communication in the application (Section 3.5).

3.1.1 The Network Interfaces

The Network Interfaces used by the main NoC are based on the INOCS architecture, and were discussed in Deliverable 3.2 (Figure 6). We instantiate two types of NIs: an AXI initiator NI to interface with the MicroBlaze, and an AHB target NI to interface with the memory. This choice was deliberate (e.g., both could have been AXI) to demonstrate interoperability among the two.

Due to the relatively simple needs of the MicroBlaze core, which does not support multiple transaction IDs, we save area by instantiating a small AXI initiator NI with support for only one such ID. However, the NI is still supporting all AXI features. Both AXI and AHB NIs, and their interoperability, were extensively tested in RTL and on the FPGA.

For integration into the platform, a few tweaks to the INOCS NI were needed:

- INOCS NIs embed routing tables to statically perform source routing. In this platform, routing is distributed and reconfigurable to work around faults or to enforce virtualization. Therefore, the routing tables are modified to instead encode the XY coordinates of the destination core; these will be processed at the switches. The coordinates are expressed as strings of
Figure 6: Generic NI architecture.

- The input and output buffers of the NIs are extended by UNIFE to support the NACK-GO flow control protocol used by their switches, instead of STALL-GO by INOCS.
- The AXI initiator NIs are extended with two extra pins, directly connected to FPGA pads, in turn connected to physical switches of the FPGA board. This means that the user, manually flipping those switches, can change the value of two bits inside each NI. The NI in turn exposes these two bits to the MicroBlaze at the reserved address 0x11000000. The MicroBlaze can poll this location to change among operating modes, e.g. staying idle, or executing one of multiple pre-programmed applications. As can be inferred from Figure 2, note that in the platform, the 16 MicroBlazes attached to the mesh have no way to communicate with the external world except for this facility (although 7 of them are also connected to the Debug Module - see Section 3.2).

3.1.2 The switch

The switch architecture adopted in this work is an extension of the GPNaNoC switch for general purpose NoC architectures described in Deliverable 5.1. The switch implements logic-based distributed routing (LBDR [1]), relies on wormhole switching and implements both input and output buffering. The crossing latency is thus 1 cycle in the link and 1 cycle inside the switch. This section briefly summarizes the key features of the switch together with the extensions introduced to meet the target FPGA platform requirements.
Fault Tolerance. The adopted flow control protocol is called NACK/GO [2], from WP2. NACK/GO combines the best of STALL/GO and ACK/NACK [3]. A stall signal enables to block or resume the communication flow to manage congestion, while an acknowledgment signal enables to notify error detection and to trigger data retransmission with Go-Back-N policy. The NACK/GO advantages come at the cost of extra channel wiring and a more complex control logic inside buffers and switches than STALL/GO. Moreover, the minimum storage requirements on each buffer grow from 2 slots to 3.

Switch extensions for fault tolerance were designed around the new flow control protocol and are pictorially illustrated in Figure 7. The architecture targets single event upsets (SEUs). In the data path, each buffer has both a detector and a corrector. Detectors are used to monitor the upstream data path and to assert the backward propagating NACK signal, requesting for a retransmission. Upon receipt of a NACK, the upstream stage performs error correction of the stored data before actually retransmitting it, since the SEU might have affected logic values of stored data. In this latter case, a simple retransmission without any correction would not solve the problem. Control logic of input and output buffers is triplicated and voted, and so are their state registers to avoid misalignment of FSMs.

By exploiting the retransmission capability provided by NACK/GO, the control path of the switch could be protected with Dual Modular Redundancy (DMR) instead of TMR. The combinational logic of each port arbiter is doubled, and associated outputs are compared by a Dual-Rail Checker (DRC) module. This latter freezes arbiter state registers in case of discrepancies between its inputs. State registers are again triplicated to preserve the consistency of FSMs. The voted current state is then fed to the duplicated output logic of the arbiter and monitored by another DRC stage. In case of discrepancies, the valid signal to the output port is deasserted, thus the incoming information is discarded. Further details can be found in Deliverable 5.1. Finally, in order to protect switch routing logic, two LBDR replicas directly feed the combinational logic.
Figure 8: Reconfiguration steps performed in an OSR-Lite environment.

Figure 8 shows the complete OSR-Lite mechanism, involving a central manager. In particular, we exploit the dual NoC, through which routers can ex-
change information about expected topology changes (e.g., frequent transient faults). The control network collects all the notification events and sends them to a central manager (step 1). If the reconfiguration is instead initiated by a resource manager in the context of virtualization strategies, step 1 can be skipped. The central manager then computes the new configuration (step 2) and disseminates the new routing information to the switches (step 3). Then, every switch starts the OSR-Lite reconfiguration process in step 4.

Figure 9 depicts the proposed OSR-Lite logic at each input port, which is able to allow an arbitrary number of reconfigurations at runtime. A local register stores the current epoch of the input port while an additional register triggered by the control network stores the switch epoch. The switch and input port epoch are compared in an $\text{exor}$ logic block to reveal a reconfiguration request ($\text{New Epoch}$ flag). When the reconfiguration request is asserted then the input port evolves to the new epoch if there are no stored flits from current epoch ($\text{Current Epoch Packet}$ signal) and the token in compliance with current switch epoch has been received from the upstream switch ($\text{Comparison Epoch}$ signal). Finally the $\text{Change Epoch}$ flag enables to set the routing module with the new configuration bits and to switch the local epoch register to new epoch. All the details of the reconfiguration strategy are provided in Deliverable 5.1.

Testing Architecture. This work implements a BIST strategy for the switch, targeting single stuck-at faults. Moreover, pseudo-random test patterns are preferred to deterministic ones because of their flexibility, lower area footprint and lower development time [5], and to functional ones due to their better coverage of control paths [4]. The basic testing framework was developed in WP1.

We envision a unique 39 bit LFSR per switch that feeds pseudo-random test patterns to every switch port in parallel. In turn, all network switches are tested and diagnosed in parallel, thus cutting down on test application time and making it independent of network size. Test responses of a switch sub-block are fed to connected sub-blocks serving as test patterns for them, provided coverage remains reasonable. As a result, we identify groups of switch sub-blocks to which we apply diagnosis. This approach minimizes test pattern generator (TPG) and
test wrapper overhead. Such grouping also reflects the switch structuring into modular input and output ports, thus facilitating diagnosis and reconfiguration in LBDR networks. Link testing is performed in a cooperative way: test patterns are injected by the upstream switch while diagnosis is performed by the downstream one. The BIST architecture is showed in Fig. 10. Test responses are collected by MISRs that are specific of input and of output ports. In fact, one MISR performs signature analysis of test responses from the link, the input buffer and the LBDR block together with the associated OSR-Lite logic. Another one collects responses from a crossbar mux, an output buffer, a port arbiter and the associated OSR-Lite logic.

Routing logic extension. As previously mentioned, the switch implements logic-based distributed routing by means of LBDR logic modules. The LBDR modules natively support a single core per switch. Thus this latter logic had been extended for the target FPGA platform where two cores belong to each switch, a MicroBlaze and a memory.

In practice, LBDR consists of a selection logic of the target switch output port relying on a few switch-specific configuration bits (namely routing $R_{xy}$, connectivity $C_z$ and deroute bits $d_{rt}$). These latter switch configuration bits need to be updated whenever the topology evolves from one connectivity pattern to another. More details can be found in [1], a scientific outcome of WP1.

In principle, the LBDR selection logic computes the destination output port by reading the destination address information contained in the header flit of each packet. In particular, the LBDR logic performs a comparison between the destination address (Dest_ID) and the local switch ID (Local_ID). When the local switch ID matches the destination address, the packet is forwarded to the local port (i.e., to the core). However, the FPGA platform is enhanced with
two nodes per switch thus each switch integrates two local ports. As a result, further information must be added to the incoming destination address of the header flit and the LBDR logic must be extended to determine whether the packet should be routed to the first or the second local port. The destination address information has been extended by 1 bit (Core_Flag). The additional bit is exploited to determine the target core at the destination switch. If Dest_ID matches Local_ID, the Core_Flag bit is used to distinguish between the two local ports. Figure 11 shows the logic gates added to the native LBDR logic.

3.2 Basic components: the supervision subsystem

In order to demonstrate the NoC functionality, a supervision subsystem is required. We choose to instantiate it within Xilinx Platform Studio, and using as many Xilinx IP cores as possible, for convenience; we integrate it with custom NaNoC IP when suitable. The subsystem (see Figure 2 and Figure 4) includes a Xilinx AXI interconnect, with two masters (a Microblaze and the Dual NoC Receiver) and numerous AXI, AXI Lite and AHB slaves.

At the heart of this subsystem is a Microblaze running software by UPV, UNIFE and INOCS. This software is tasked with:

- Probing the status of the NoC, e.g. after BIST and upon fault occurrences.
- In response to the above, configuring or reconfiguring the NoC.
- Awaiting for possible user requests to reconfigure the NoC in a virtualized manner.
- In response to the above, reconfiguring the NoC.
- Pulling the link sniffers periodically to monitor activity on the NoC links.
- Transferring key information about the platform’s functioning outside the FPGA through the serial port (or, potentially, an Ethernet port).

To perform these actions, multiple support controllers and devices are needed. First of all, since the supervision software and the required underlying C library have a non-negligible footprint, incompatible with on-chip resources, a DRAM controller is advisable to be able to store the software. To support the basic functionality of the Xilinx C library, a timer and an interrupt controller must also be present. (Note that, in contrast, the 16 Microblazes connected to the NoC mesh do not have access to external memory, timer or interrupt controller; this limits the capabilities of the software that can be run on those).
In order to monitor the NoC, it is necessary for the Microblaze to be able to access the dual NoC. This is done via three components plugged to the AXI bus: a Dual NoC Driver, a Dual NoC Receiver, and a memory. The first two blocks derive from INOCS AXI NIs, modified by UNIFE to exchange packets in the specific format expected by the dual NoC. The Microblaze can directly write to the Dual NoC Driver, which is a slave on the AXI bus, to program the main NoC. Due to the way the dual NoC was designed, the reverse operation cannot be done with a read to the same device; instead, whenever there is a message requiring attention (e.g., upon BIST completion or fault detection), the dual NoC sends a packet to the Dual NoC Receiver, which converts it into an AXI transaction directed at the on-bus memory (a standard Xilinx core). The Microblaze can periodically poll this memory to check all notifications.

To supervise the NoC activity, the Microblaze can also poll the Traffic Sniffers. These blocks can be connected to up to 16 links of the main NoC on one side, and to the AXI bus on the other. For maximum thoroughness, we choose to monitor as many as 80 links of the NoC (almost all, disregarding just a few whose information is redundant), with five Traffic Sniffers in parallel. The sniffers include a counter that is incremented at the passage of any flit; whenever the counter is read by the MicroBlaze, it automatically resets itself. A simple division yields a utilization metric.

Finally, the FPGA needs external interfaces. First of all, a GPIO controller allows the Microblaze to periodically check the status of a few physical buttons and switches on the FPGA board. This allows the user to change operating modes of the platform; for example, we use this feature to instruct the software on the Microblaze to initiate the reconfiguration to get into virtualized application mode. Two extra blocks are used to communicate with the user’s computer. A UART controller is an output-only interface that allows the platform to transfer information to the laptop, where the GUI by UPV can visualize it. A debug module, relying on a JTAG-over-USB electrical connection, allows for bidirectional communication: the user can program the supervision Microblaze, step through its software, and check the content of certain on-FPGA registers and memories. Since the debug module allows for monitoring of up to 8 Microblazes, we connect it to the supervision Microblazes and to selected 7 other Microblazes out of the 16 attached to the main NoC mesh.

### 3.3 Basic components: the reconfiguration algorithm

The supervisor MicroBlaze is constantly monitoring the status of the NoC through the dual NoC. Whenever a notification is received about a fault on a link, if deemed necessary (e.g. unless it is assumed to be a transient), the supervisor triggers the reconfiguration algorithm. This algorithm computes the required changes in the LBDR bits at specific switches in order to migrate from the current routing algorithm to a new one that avoids the use of the notified link. The algorithm is reported in Deliverable 2.1 and is a simple access to two precomputed tables containing all the changes for every possible link failure. Those bits are encoded and transmitted through the dual NoC together with a triggering notification to switches to launch the reconfiguration process (the generation of tokens and their advance through the network). All the details of the reconfiguration algorithm are shown in Deliverable 2.1.
3.4 The GUI infrastructure

UPV has extended its simulation platform (gMemNoCsim) with a proper Graphical User Interface (GUI) for the FPGA prototype. The GUI can display custom-made NoC topologies from CEF files, thus enabling the interoperation between consortium partner tools. In detail, the GUI is able to take into account all the blocks and links connecting blocks. Figure 12 shows the GUI with the topology of the prototype case loaded and represented.

![Figure 12: GUI topology panel.](image)

The GUI has been equipped with support for analysis of the operation of the topology on the FPGA. In particular, the GUI offers the possibility to display the activity of any monitored link inside the FPGA. This can be performed in two directions:

- **RealTime communication** with the NoC: The link activity is collected and delivered to the computer tethered to the FPGA through the serial port. The output is written to a file and the GUI reads from that file (the file is treated as a pipe).

- **Previously recorded activity file**: The GUI displays the link load already stored in a file. The data can be represented in two different ways:
  - **Automanged**: Link utilization and events are shown and updated automatically on the screen plots. This is the same way as when configuring the GUI for real time communication.
  - **Manual**: In this mode the data is displayed statically and it is possible to navigate backwards or forwards cycle by cycle.

The link activity\(^1\) can be displayed in two different ways:

- **Topology view** The information is shown over the NoC topology representation. The link activity is represented with a coloured bar that gets

\(^1\)All links are updated at the same time.
filled and changes its color as the activity increases. Green is used for an idle link and red for a fully utilized link. An example is shown in Figure 13. The percentage is also displayed in numerical format just above the coloured bar.

- **Charts view** The information is shown in a new panel with independent charts for each link as shown in Figure 14. The number of charts to display can be selected in this panel, as well as the link represented in each chart.

![Figure 13: GUI load at links in topology view.](image1)

![Figure 14: GUI load at links in graph view.](image2)

The GUI has been implemented exclusively for the support of the prototype showcase. Extra effort was required for the proper development and communication with the FPGA platform. However, the developed effort is very useful for the UPV team as the simulator can now be used for teaching purposes on UPV’s master courses related to on-chip interconnects.
3.5 The application

The MicroBlazes have been programmed in order to start their application after the 4x4 mesh is configured, upon flipping a physical switch on the board. The application run by the MicroBlazes is a matrix multiplication consisting of the product of a pair of matrices. The MicroBlazes sequentially forward the results to each other in a pipelined producer-consumer fashion. Each MicroBlaze performs the multiplication of a private matrix and a matrix delivered by the previous MicroBlaze of the sequence. Once the matrix product is computed the resulting matrix is forwarded to the next MicroBlaze. The lack of I/O interfaces and memory does not allow the implementation of more advanced applications.

The private matrix (\textit{mat\textsubscript{private}}) is stored by each MicroBlaze into its local (pertaining to its local switch) AHB scratch memory of 4kB and it is simply initialized as follows:

\begin{verbatim}
for (row = 0; row < LINES; row ++)
    for (column = 0; column < LINES; column ++)
        (*mat_private)[row][column] = row + column;
\end{verbatim}

Notice that the matrix size can be tuned by means of the \textit{LINES} parameter. Moreover the same matrix multiplication can be set to run for a specific number of times or in an endless fashion. The AHB memory is used as storage and for inter-processor communication in the application. Indeed the incoming matrix from the previous MicroBlaze (\textit{mat\textsubscript{input}}) is stored in the AHB memory connected to the local switch. Each MicroBlaze stores its matrix product result (\textit{mat\textsubscript{output}}) into the AHB memory connected to the switch to which the next MicroBlaze of the sequence is connected. The first MicroBlaze of the pipeline initializes its own local AHB memory before performing the matrix product. Each MicroBlaze has local registers storing the address of the local AHB memory, the addresses of the remote AHB memory of the next MicroBlaze, the position within the pipeline, and the pipeline length. Figure 15 depicts the application at work.

The application run by each MicroBlaze is simply the following:

\begin{verbatim}
for (row = 0; row < LINES; row ++)
    for (column = 0; column < LINES; column ++)
        (*mat_private)[row][column] = row + column;
\end{verbatim}
for (i = 0; i < LINES; i++)
for (k = 0; k < LINES; k++) {
    (*mat_output)[i][k] = 0;
    for (j = 0; j < LINES; j++)
        (*mat_output)[i][k] += (*mat_input)[i][j] * (*mat_private)[j][k];
}

In order to guarantee the synchronism between the MicroBlazes, custom semaphores are implemented. Interestingly, these are purely software and do not need dedicated hardware support. Such a solution slightly increases the complexity of the code but clearly simplifies the hardware design effort and the area overhead. Of course this approach is possible only since the application is fixed and known upfront; more sophisticated synchronization capabilities would demand hardware-level atomicity support.

The goal of these semaphores is to avoid reading the same incoming matrix multiple times, and to avoid overwriting output matrices before the next MicroBlaze has been able to process them. Each MicroBlaze has been enhanced with 4 semaphores:

- **Semaphores_read** is stored in the local AHB memory. It is polled (and therefore set) by the local MicroBlaze, while it is reset by the next MicroBlaze of the sequence to notify that it is available to receive new input. As soon as polling reveals that the semaphore is reset, the local MicroBlaze can forward the matrix product result to the next MicroBlaze.

- **Semaphores_write** is stored in the local AHB memory. Similarly to the previous semaphore, it can be polled and set by the local MicroBlaze while it is reset by the previous MicroBlaze of the sequence to notify that a new input matrix is incoming.

- **Semaphores_readprev** is stored in the AHB memory of the previous MicroBlaze. The current MicroBlaze resets it as soon as it has read the matrix incoming from the previous MicroBlaze.

- **Semaphores_writenext** is stored in the AHB memory of the next MicroBlaze. The current MicroBlaze resets it as soon as it has forwarded the matrix product result to the next MicroBlaze.

Notice that semaphores to be polled have been placed in the local AHB memory in order to reduce congestion in the network. Figure 16 shows the semaphores location.

Since hardware support to semaphores has not been implemented, special care was devoted to guarantee the proper semaphore initialization. Indeed **Semaphores_write** and **Semaphores_readprev** needs to be respectively initialized to 1 and 0. Also, semaphore initialization must occur in synchronism otherwise the first MicroBlaze could start operations before the next MicroBlaze is actually ready. Thus, the initialization sequence progresses in reverse order with respect to the pipeline: each MicroBlaze only initializes its own semaphores once the semaphores of the next MicroBlaze have been already initialized. The last MicroBlaze of the pipeline is therefore the first allowed to initialize itself, while the first will be the last. This ensures that the matrix multiplication starts only when all the MicroBlazes are properly initialized.
3.6 The physical platform implementation

Some steps of the implementation flow described in Figure 3 can be parallelized; for example, the initial platform description involves several blocks which can be independently synthesized in parallel. Even after joining all the pieces together, the mapping stage can be run on two threads in the Xilinx toolchain, and the placement and routing in four. Despite this, we measure end-to-end flow runtimes of about 7 hours on a dual-chip Opteron 6378 (16 threads/core) server with 128 GB of RAM. We observe peak memory utilization close to 10 GB during implementation. The layout of the platform implementation can be seen in the screenshot of Figure 17.

The platform fills the FPGA almost completely, as can be seen in Table 1. The left column reports the utilization when the template system generated in XPS (Figure 4) is implemented, the right one is for the same system where the NoC, dual NoC and associated NaNoC IP (e.g. sniffers, dual NoC interface blocks, etc.) have been instantiated to replace the simple AXI interconnect. It can be seen that the NoC represents approximately 17% of the FPGA’s sequential resources and 66% of the combinational resources (or a little bit more, since this is the overhead on top of the default bus); it does not occupy any BRAM nor require external pins.

Due to development timing constraints, no specific optimizations could be taken to reduce the area of the design; given the large number of blocks and the redundancy features (e.g. triplication of some components, BIST, datapath encoding) built into the NoC, we perceive the resource utilization figures as very positive. Note that triplicated logic in the GPNaNoC switch has to be marked with special annotations embedded in the RTL, otherwise the Xilinx synthesis tools would detect it as redundant and prune it away.

The design is not aimed at, and not optimized for, high performance. The very high resource utilization features also impose a significant timing overhead as routing necessarily becomes more convoluted and less efficient. We record a maximum operating frequency of 38 MHz; the critical path is in the BIST logic of the GPNaNoC switch.
Figure 17: Layout of the full FPGA design. Green: data NoC; red: Network Interfaces; yellow: dual NoC; cyan: MicroBlazes and other logic.

Table 1: Resource utilization of the Virtex 7 chip.

<table>
<thead>
<tr>
<th>Resource utilization</th>
<th>Supervisor subsystem only</th>
<th>Full platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice Registers</td>
<td>5%</td>
<td>22%</td>
</tr>
<tr>
<td>Slice LUTs</td>
<td>16%</td>
<td>88%</td>
</tr>
<tr>
<td>IOs</td>
<td>20%</td>
<td>20%</td>
</tr>
<tr>
<td>36-bit BRAMs</td>
<td>61%</td>
<td>61%</td>
</tr>
</tbody>
</table>

4 Validating Built-in Self-Testing and NoC configuration

In order to validate the Built-in Self-Testing implemented in the 4x4 mesh, a permanent failure was forced in the network by hard-wiring to zero a link wire. In this implementation, the failure was injected in the link between switch 11
and 10. However, it could have been freely injected in different locations since the 4x4 mesh has been based on the GPNaNoC switch that guarantees around 97% of stuck-at-fault coverage.

![Diagram](image)

(a) Cooperative BIST Procedure.  
(b) Test result notification.

(c) Supervisor polls the memory and computes configuration bits.  
(d) Network configuration.

Figure 18: Built-in-Self-Testing at work.

As soon as the FPGA board is booted the BIST automatically starts and the switches cooperatively exchange test patterns as shown in Figure 18(a). When the BIST procedure is completed, the BIST managers integrated in each switch send to the dual NoC the diagnosis information related to the switch they belong to. In the FPGA platform under test, the BIST manager of Switch 10 reveals an error on its East input channel where the error has been injected. Thus it notifies the dual NoC, which takes care of delivering all the BIST diagnosis information to the global manager (i.e., the supervision MicroBlaze). In particular, the diagnosis information crosses the dual NoC and the Dual NoC Receiver before being stored in the memory connected to the supervision subsystem (Figure 18(b)).

The supervision MicroBlaze has been programmed to periodically check for dual NoC notifications by polling the control bus memory (Figure 18(c)). It recognizes when the BIST notification information has been stored in the memory (i.e., the BIST procedure is completed) and it runs the configuration algorithm.
described in Section 3.3. Thus, it computes configuration bits able to guarantee deadlock-free routes despite the failed link. The configuration bits are sent to the Dual NoC Driver through the AXI bus. They cross the dual NoC and configure the routing mechanism of each switch (Figure 18(d)).

4.1 Protocol for BIST notification and configuration

The 4x4 mesh is properly configured only once the configuration bits are sent twice by the supervision MicroBlaze. The exchanged information follows a sophisticated protocol envisaged to meet multiple requirements:

- The dual NoC is a highly simplified version of the main NoC. The information travels in small packets composed of 2 or 3 flits as a function of the notification type. Only head and tail flits are required when a notification related to transient errors is delivered.

- The information must be exhaustive in order to allow the supervision MicroBlaze to take a wide range of diagnosis and recovery actions. As an example, it should be able to identify the kind of fault (fault_type) that has occurred (permanent or transient) and the address of the failed switch (address_sender).

- The notification information should be able to pinpoint the exact location of the error inside the switch (bist_result) to enable an effective network configuration.

- The protocol must be intrinsically fault tolerant to guarantee a reliable communication. Thus, the information is sent multiple times or encoded within the packets.

- The configuration bits generated by the supervision MicroBlaze must match the specifications of the switch routing mechanism (Section 3.1.2).

In the case of BIST notification and configuration the protocol consists of 4 phases. In the first phase, the switches notify the BIST results by means of three 22-bit flits with the following format:

Head:
\[ flit_type(2), priority(2), address_sender(8), unused(10); \]

Body:
\[ flit_type(2), fault_type(2), ctrl(2), bist_result(10), unused(6); \]

Tail:
\[ flit_type(2), ctrl_neg(2), fault_type_neg(2), bist_result_neg(10), unused(6); \]

Note that the tail flit contains the body information in its negated version for fault tolerance reasons.

During the second phase, the supervision MicroBlazes sends the configuration bits in compliance with the LBDR specification by means of three flits per switch:

Head:
\[ flit_type(2), priority(2), address_receiver(8), unused(10); \]

Body:
In the third phase, the switches forward back to the MicroBlaze the configuration information just received following the same above reported format.

In the last phase, the MicroBlaze checks the incoming information to ensure that the configuration information delivered in the second phase has been correctly received by the switches. If the delivery is confirmed as safe, the MicroBlaze notifies the switches by sending for the second time a replica of the second phase information.

It was established that this 4-phase protocol provides the best tradeoff between robustness, performance and complexity. It is however conceivable, and relatively straightforward, to extend it further for additional resilience to faults, for example with additional request/response exchanges or additional encoding of the configuration strings. This is entirely a degree of freedom that can be customized in the platform depending on the envisaged final application.

Once the 4 phases have been executed, the 4x4 mesh is configured and the matrix multiplication application described in Section 3.5 can run properly.

5 Validating Fault Detection and NoC Reconfiguration

Once the network has been tested and permanent faults have been detected and tackled by the off-line configuration, the system can be still affected by run-time transient and intermittent faults. Such faults cannot be handled by off-line strategies as they appear and disappear unpredictably. As a result, the network has been designed as fault tolerant to satisfy the high reliability constraints imposed by modern systems (see Section 3.1.2). In particular, the fault-tolerant flow control protocol (NACK/GO) is used on the data path to notify error detection and trigger data retransmissions. Although this protocol has been primarily designed to tackle SEUs (Single Event Upset), the system is also able to tackle physical effects such as wear-out. Indeed wear-out effects end up in permanent faults but they are known to have a gradual onset. In practice, frequent transient faults affecting the same circuitry denote the possible onset of a permanent fault. Before this happens, the network routing function could be modified to exclude the affected circuit from communication traffic. NACK/GO lends itself to such a policy, since its retransmission and/or voting events may be notified to the supervision MicroBlaze which may monitor the distribution and frequency of transient faults over time and eventually take the proper course of recovery action. This exact policy is supported and validated by the FPGA platform.

Physical buttons and switches of the board are connected to an on-chip GPIO controller to allow the user to interact with the platform. The physical buttons have been leveraged to inject transient faults in the network and validate the above mentioned fault tolerance policy. For this purpose, a fault injection module has been instantiated along the link routed from switch 4 to 5. This module is connected to a physical button on the FPGA board and provides a
method to inject transient faults on that link (see Figure 19(a)). Since the link may be idle when the button is pressed, the fault injection module integrates a simple FSM that waits until a valid flit is crossing the link to inject the fault, ensuring that actual important information is corrupted. Therefore, every time the button is pushed, the error is revealed and notified to the supervision MicroBlaze (Figure 19(b)).

Similarly to the procedure followed by the BIST notification described in Section 4, the transient notification crosses the dual NoC and it is stored into the control bus memory. The supervision MicroBlaze periodically polls the memory also during run-time operations. Thus it reads the transient notification and updates its register with distribution and frequency of transient faults over time. Only when the number of transient notifications from the same link reaches a threshold is recovery action taken. For the sake of the demonstration, the MicroBlaze’s software is set to run its reconfiguration procedure after three
notifications (i.e., after the button has been pushed three times). Note that the 4x4 mesh at this stage is irregular since a link has been already disabled due to a previously detected permanent failure. Thus, the algorithm computes the reconfiguration bits for this irregular network and delivers them to the dual NoC (Figure 19(c) and Section 3.3).

The new reconfiguration bits coming from the dual NoC cannot directly update the existing routing strategy, as during off-line operations, since applications are now running. Thus, the network implements the OSR-Lite reconfiguration mechanism which avoids stopping or draining network traffic during the transition from one network configuration to another. As described in Section 3.1.2, the switches of the FPGA platform start to inject tokens into the network. The tokens follow the channel dependency graph of the old routing function and progressively drain the network from old packets, as represented in Figure 19(d).

5.1 Protocol for transient notification and reconfiguration

As described in the case of BIST notification, also transient notification and reconfiguration follow a sophisticated protocol able to meet the platform requirements underlined in Section 4.1. The protocol consists again of 4 phases. The protocol moves to the second phase only when the switches notify multiple times a transient fault on the same link/switch port. Otherwise, the first phase is repeated until the notification threshold value is reached. The transient notification is forwarded to the supervision MicroBlaze by means of two flits with the following format:

Head:
- flit_type(2), priority(2), address_sender(8), time_info(10);
- Tail:
  - flit_type(2), fault_type(2), ctrl(2), transient_position(10), unused(6);

Differently from BIST notification, the head flit contains information about the error time occurrence (time_info), which can be useful to compute the frequency of the transient. The transient_position field pinpoints the location of the fault within the switch.

When the number of transient notifications reaches the threshold the second phase starts. The MicroBlaze sends reconfiguration bits following the same format of the off-line configuration, previously described:

Head:
- flit_type(2), priority(2), address_receiver(8), unused(10);

Body:
- flit_type(2), ctrl(2), lbdr_configuration(18);

Tail:
- flit_type(2), lbdr_configuration(20);

As during configuration, the switches forward back the reconfiguration information to the MicroBlaze (third phase) and finally the latter sends for a second time a replica of the reconfiguration information (fourth phase). Once the 4 phases have been executed, the 4x4 mesh starts to inject the OSR-Lite
tokens to drain the network and migrate to the new routing strategy.

6 Validating NoC Virtualization

![Network regions before (a) and after virtualization (b). Note that the arrows indicate the logical application flow, not necessarily the route followed by packets. For case (b), the arrows are only indicative of partitioning, but the pipeline sequence is in fact shuffled for verification purposes.](image)

Figure 20: Network regions before (a) and after virtualization (b). Note that the arrows indicate the logical application flow, not necessarily the route followed by packets. For case (b), the arrows are only indicative of partitioning, but the pipeline sequence is in fact shuffled for verification purposes.

The 4x4 mesh of the FPGA platform is also able to support virtualization. In order to validate such a feature, the platform is virtualized at run-time. In particular, the network has a single region when the operations start after the boot procedure. Thus, the same matrix multiplication application is run by all the 16 MicroBlazes and it propagates from Switch 0 to Switch 15 in a zig-zag fashion, as illustrated in Figure 20(a). Since virtualization is not in place, packets follow the best deadlock-free path available. Note that packets do not strictly respect the arrow of Figure 20(a) but they are routed in compliance with the actual routing strategy which takes into account also the faults of the considered scenario. If the user requires a virtualization, the network is split in three regions, as represented in Figure 20(b). In this new mode, a dedicated matrix multiplication application runs into each region and packets are constrained to respect virtual boundaries.

In order to enforce virtualization at the user’s request, external interfaces are required. Dedicated physical switches on the board serve this purpose. These are directly connected to the initiator NIs of the 16 data NoC MicroBlazes (Section 3.1.1). All 16 MicroBlazes periodically check the status of the physical switches. When they detect a change, they transition operating mode, interrupting the current application, reading the address map for the new matrix multiplication sequence of Figure 20(b), and finally running the new application.

The status of the physical switches can be polled by the supervision MicroBlaze too, through the GPIO controller. Therefore, in parallel, the software on
the supervision MicroBlaze detects the request to get into virtualized application mode (Figure 21(a)). The reconfiguration bits required to virtualize the network are then computed and sent through the dual NoC (Figure 21(b)). In particular, the routing algorithm modifies the connectivity bits of the switch routing functions to shape the network in three virtual regions. The token propagation of the OSR-Lite reconfiguration mechanism is triggered (Figure 21(c)) as described in the previous section. The packets are eventually forced to propagate within the region boundaries.

Note that the sequence of the application propagation of Figure 20(b) is only indicative; if the pipeline were as depicted, communication would be strictly point-to-point across a single hop, and virtualization would be unnecessary. To verify its effectiveness, each of the three pipelines has been shuffled, so that producer-consumer traffic would tend to stray out of the geometric shape of

(a) Notification of virtualization request. (b) Notification of network reconfiguration.

(c) Overlapped static reconfiguration (OSR-Lite) at work.

Figure 21: Virtualization request and reconfiguration.
each region, if not for virtualization enforcement. As a consequence, it has been possible to prove that the packets do not violate the region constraints even when the theoretical best deadlock-free path goes through a neighboring region.

The protocol adopted to communicate the reconfiguration bits for virtualization matches that for transient reconfiguration (Section 5.1). However the protocol is 3-phase only; the first phase is missing since the reconfiguration is triggered by external physical switches instead of by a network notification.

7 Conclusions

This deliverable reports on the prototyping of a 16-core homogeneous multicore processor with a fault-tolerant, runtime reconfigurable and dynamically virtualizable on-chip network. The prototyped system has been successfully validated in its capability of boot-time testing and configuration, transient or intermittent fault detection, runtime reconfiguration of the routing function, and dynamic partitioning and isolation. The FPGA prototype is the outcome of a tight cooperation between key NaNoC partners: UNIFE, INOCS, UPV, and SIMULA, to such an extent that their contributions is currently indivisible. The validated NoC prototype is a key enabler for the future evolution of embedded systems. First, it enables the integration of multiple software functions on a single multi- and many-core processor (multifunction integration). This is the most efficient way of utilizing the available computing power. Ultimately, the NaNoC project is a milestone in the direction of mixed-criticality multifunction integration. Second, NaNoC design methods enable advanced forms of platform control, especially power management and thermal control. In fact, parts of the network can be easily powered off, while preserving its global functionality and guaranteeing safe transitions between network configurations. Third, NaNoC technology paves the way for an effective graceful degradation of the system in the presence of permanent and intermittent faults. The routing function can be reconfigured at runtime to avoid faulty links and switches. Finally, a comprehensive reliability framework has been set into place, from switch-level to network-level, while covering all design aspects (e.g., reliable control signaling) and effectively co-optimizing different architectural features together (fault-tolerance, testing, runtime reconfiguration, control signaling).

References


