Worst-Case Latency Analysis for the Versal Network-on-Chip

by

Ian Elmor Lang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2021

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contribution

The work presented in this thesis is partially based upon and extends the work presented in the following published short paper:

[26]: Ian Lang, Nachiket Kapre, and Rodolfo Pellizzoni. 2021. Worst-case latency analysis for the versal NoC network packet switch. In Proceedings of the 15th IEEE/ACM International Symposium on Networks-on-Chip (NOCS '21). Association for Computing Machinery, New York, NY, USA, 55–60. DOI:https://doi.org/10.1145/3479876.3481593

Due to the relation with the published work, parts of this thesis contain significant material from [26], including Chapters 1 - 3 and Sections 4.3 and 5.2.

Abstract

The recent line of Versal FPGA devices from Xilinx Inc. includes a hard Network-On-Chip (NoC) embedded in the programmable logic, designed to be a high-performance system-level interconnect. While the target markets for Versal devices include applications with real-time constraints, such as automotive driver assist, the associated development tools only provide figures for "structural latencies" of data packets, which assume that the network is otherwise idle. In a realistic setting, this information is not enough to ensure deadlines are met, as different packets can contend for NoC switch outputs, which causes packet contents to be buffered while in transit, increasing their latency. In this work, we develop an approach for calculating upper bounds for such worst-case latencies (WCLs), assuming a model where system tasks release packets into the NoC periodically. In order to develop an accurate model for latencies in the network, we review the architecture and operation of the Versal NoC. We focus on a formal description of the NPS switches that compose the NoC from a flit arbitration perspective, based on study the available cycle-accurate switch simulation code. Working with the presented model, we propose an adaptation to an existing approach for WCL analysis in NoC, Recursive Calculus (RC), in order to apply it to the arbitration policy implemented in the Versal NoC. To evaluate the proposed approach, we implement a simulation experiment for the Versal NoC, with custom endpoints that allow for injecting packets programatically and measuring their latencies over the NoC. We simulate both a single NPS module and a complete NoC routing periodic workloads, in order to compare with the values given by the WCL approach and identify sources of pessimism.

Acknowledgements

I would like to thank my supervisors Prof. Nachiket Kapre and Prof. Rodolfo Pellizzoni, and the members of my thesis committee Prof. Sebastian Fischmeister and Prof. Ziqiang Huang.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ACAP Adaptive Compute Acceleration Platform 1

AI Artificial Intelligence 1

AXI Advanced eXtensible Interconnect 5

BE Best Effort 14

CPA Compositional Performance Analysis 9

CT Communication Task 2

DDRMC DDR Memory Controller 1, 16

DOR Direction-Ordered Routing 34

DSP Digital Signal Processing 6

EoP End-of-Packet 13

FPGA Field-Programmable Gate Array 1

HDL Hardware Description Language 6

HNoC Horizontal Network-on-Chip 1

ILP Integer Linear Programming 34, 40, 45

IP Intellectual Property 1, 5

ISOC Isochronous 14

JSON Javascript Object Notation 14

LL Low-Latency 14

LRU Least Recently Used 3

MPB Multi-Point Progressive Blocking 10

NCRB NoC Clock-Reconvergent Buffer 13

NIDB NoC Inter-Die Bridge 16

NMU NoC Master Unit 13

NPI NoC Programming Interface 13, 21

NPP NoC Packet Protocol 13

NPS NoC Packet Switch 3, 12

NSU NoC Slave Unit 13

NoC Network-on-Chip 1

QoS Quality of Service 3

RC Recursive Calculus 1, 3, 11

RRA Round-Robin Arbitration 10

SAT Binary Satisfiability 14

SLA Stage-Level Analysis 10

SLR Super Logic Region 16

SoC System-on-Chip 2

SoP Start-of-Packet 13

TDMA Time Division Multiple Access 10

VCB Virtual Channel Buffer 16

VC Virtual Channel 3, 7

VNoC Vertical Network-on-Chip 1

WCET Worst-Case Execution Time 2

WCL Worst-Case Latency 2

List of Symbols

$C(\tau_i)$: Structural latency to deliver packet of real time CT τ_i 27

Γ^L : Set of best-effort (low-priority) CTs traversing the Versal NoC 27

Γ : Set of real-time (high-priority) CTs traversing the Versal NoC 25

$R(\tau_i)$: WCL to deliver packet of real time CT τ_i 28

$V.\Gamma(l)$: Set of CTs that are stored in VCB V and leave the NPS through link l 27

$V.\Gamma$: Set of CTs that are stored in VCB V 27

$V.L(l)$: Maximum size for a packet that is stored in VCB V and leaves through link l 27

$V.S$: Buffer size for the VCB V 27

$V.c(l)$: Token counter maintained by output link l for VCB V 18, 27

$V.r(l)$: Token register for the counter maintained by output link l for VCB V 18, 27

$d_{\boldsymbol{R2}}(\tau_i, l)$: Number of cycles that a packet of τ_i may block another packet associated with the same VC while being transmitted through output link l, without counting nested blocking cycles. 41

$d_{\boldsymbol{bubbles}}(\tau_i, l)$: Number of cycles, after the SoP flit of a packet of τ_i wins arbitration for l, that the subsequent flits of the packet are not supplied to the arbitration process for l due to interference suffered upstream in the packet's route. 39

$d_{\boldsymbol{buf}}(\tau_i, l)$: WCL for the buffer downstream from l where τ_i is stored to be emptied. 34

$d_{\boldsymbol{local}}(\tau_i, l)$: Worst-case number of cycles that a packet of τ_i is prevented from accessing l due to mechanisms for contention with other CTs. 36

$d(\tau_i, l)$: WCL to deliver a packet of τ_i, assuming that the SoP flit of the packet of τ_i is at the head of $VCB(\tau_i, l)$ and flits of the packet are continuously supplied for the arbitration for l, up until the EoP flit. 30, 31

$VCB(\tau_i, l)$: The NPS VCB that CT τ_i accesses the link l from. 26

firstLink(τ_i) : Returns the first link in the route $\tau_i.\delta$ 26

lastLink(τ_i) : Returns the last link in the route $\tau_i.\delta$ 26

nextLink(τ_i, l) : Returns the link following l in the route $\tau_i.\delta$ 26

packetCount(τ_i, τ_k) : Maximum number of packets of τ_k that can interfere with τ_i, for the current estimate of WCL values. 39

prefixPath(τ_i, l) : Returns the last link in the route $\tau_i.\delta$ 26

prevLink(τ_i, l) : Returns the link preceding l in the route $\tau_i.\delta$ 26

$l.F$: Credit feedback delay for the NPP link l 27

$l.b$: Basic latency for the NPP link l 27

$\mathcal{V}^{DVH}(\tau_i, l)$: Set of other buffers that can contend with $VCB(\tau_i, l)$ for l, store flits from real-time CTs, and are associated with a VC different from $\tau_i.VC$ 35

$\mathcal{V}^{DVL}(\tau_i, l)$: Set of other buffers that can contend with $VCB(\tau_i, l)$ for l, and store flits from best-effort CTs 35

$\mathcal{V}^{SV}(\tau_i, l)$: Set of other buffers that can contend with $VCB(\tau_i, l)$ for l and are also associated with $\tau_i.VC$ 35

$\mathcal{V}(\tau_i, l)$: Set of other buffers that can contend with $VCB(\tau_i, l)$ for l 34

$\tau_i.D$: Relative deadline for packets of real-time CT τ_i 26

$\tau_i.J^R$: Release Jitter for packets of real-time CT τ_i 26

$\tau_i.L$: Length in flits for packets of real-time CT τ_i 26

$\tau_i.T$: Minimum inter-arrival time for packets of real-time CT τ_i 26

$\tau_i.VC$: Relative deadline for packets of real-time CT τ_i 26

$\tau_i.\delta$: Route followed by packets 26

Chapter 1

Introduction

The Versal ACAP line of re-configurable computing devices from Xilinx Inc. was announced in 2019 [50], with the first evaluation boards having been released in 2021 [46]. The Versal architecture includes the new generation of the company's FPGA fabric, as well as a dual-core ARM Cortex A57 processor and specialized accelerators for AI operations [51], a focus on heterogeneous computing that is reflected in the ACAP denomination.

The backbone for communications between the different computing resources in the platform, and between computing resources and DDRMC modules or I/O, is a hardened NoC embedded in the FPGA fabric [45].

Figure 1.1 shows an example floorplan of a Versal device, including the embedded NoC. The Versal NoC consists of VNoC and HNoC segments: multiple VNoCs are embedded in the FPGA fabric, offering connectivity to instantiated IP modules, while two HNoCs on the extremities of the device connect all VNoCs and provide network access to hard blocks in the system.

Internally, VNoCs and HNoCs implement a high-performance, pipelined interconnect based on 128-bit, full-duplex data links operating at 1GHz, with VNoCs having a bisection bandwidth of 512 Gbps, and HNoCs a bandwidth of 512 or 1024 Gbps. Besides representing an opportunity for performance gains in data movement-intensive applications, the NoC offers other advantages for designers (further discussed in Section 2.1.3), such as simplifying timing closure for modules in the FPGA fabric [43].

In this work, we explore an issue that can make it more difficult for developers working with real-time deadlines to take advantage of this new component (the uncertainty on the latencies that can be experienced by packets traversing the network) and discuss how to adapt an existing approach, RC [14, 29], to calculate upper bounds for said latency values.

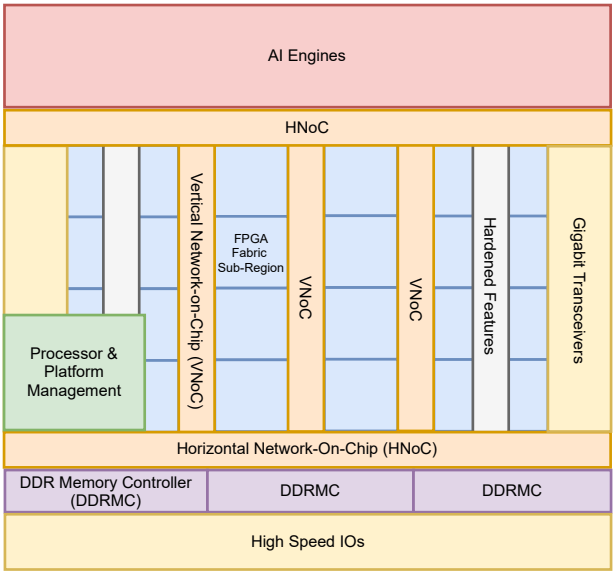


Figure 1.1: Example floorplan of a Versal device (adapted from Figure 1 of [43])

1.1 Motivation

The target markets for Versal devices include automotive, airspace and defense applications [50]. Such applications usually encompass a number of timing constraints, which must be respected in order to ensure the safety of users. A common constraint is that the WCET of a system task should be inferior to a given relative deadline.

In the context of a SoC deployed on a Versal device, the WCL experienced by packets traversing the NoC will factor into the WCETs of tasks that involve communications between different components. For example, an AI inference task for obstacle detection in an automotive driver assist application may involve the processor using the NoC to send input data to AI engines or kernels in the FPGA, and to read results subsequently.

If the latency for sending data or reading results is too large, the processor will fail to complete the sequence of steps before the deadline, violating a timing constraint, which makes evident the need to determine upper bounds for packet WCLs.

In this work, we model a timing constraints on NoC communications as a CT that releases packets into the network periodically, with associated deadlines for reception. The Versal NoC user guide differentiates between two latency values for packets [49]: the *structural latency* which corresponds to the best-case latency experienced by a packet

traversing an otherwise vacant NoC, and the *queuing latency* which also includes the time packet contents spend buffered inside NoC switches due to contention with other CTs (the WCL corresponds then to the *worst-case queuing latency*).

Only the structural latency for packets of each CT is provided by the development tools for the Versal NoC, as it is a simple function of the static route the packet follows, while calculating an upper bound for the queuing latency also depends on the behaviour of the other CTs using the network. Our goal is then to receive the NoC configuration and CT parameters as inputs, and determine an upper bound for the WCL experienced by packets of each CT.

In the context of previous works in WCL analysis for NoCs, we also identify the Versal NoC as a relevant case study because it supports a diverse set of features that can interact to affect packet latency, compared to NoC models adopted previously – including more than one VC, LRU arbitration and additional mechanisms for QoS [49].

Finally, the NPS modules, the main routing mechanism in the NoC, are well documented, and have an available functional simulation model, which facilitates the experimental evaluation of the WCL analysis approach, as performed in Chapter 5.

1.2 Contributions and Outline

This work presents the following contributions:

- We present a detailed review of the architecture and operation of the Versal NoC, including low-level details on the NoC arbitration policy and communication protocols based on our study of the NPS simulation code.
- We propose a modification to an existing approach for WCL analysis in NoCs, RC, in order to adapt it to the particular features of the Versal NoC architecture.
- We implement the proposed method as a Python script, and compare the WCL bounds it provides with results from simulating a subset of the NoC in the xcvc1902 Versal device routing the same workload.

The remainder of this text is organized as follows:

- Chapter 2 reviews the two main background topics for this work, which are embedded NoCs for FPGA, and WCL analysis for NoCs.

- Chapter 3 reviews relevant aspects of the architecture and operation of the Versal NoC.
- Chapter 4 presents the model we adopt for CTs using the NoC, and our approach for bounding the WCL experience by packets under said model.
- Chapter 5 presents an experimental evaluation of the proposed WCL analysis approach, based on simulating the NoC present in a Versal device.
- Chapter 6 includes a conclusion and considerations on future work.

Chapter 2

Background and Related Work

In this chapter, we review two important background topics for this text:

- Section 2.1 discusses embedded NoC architectures for FPGAs, including a review of NoC flow control concepts important for the remainder of this text.
- Section 2.2 reviews WCL analysis approaches for NoCs based on wormhole routing, the mechanism employed by the Versal NoC.

2.1 Embedded NoCs for FPGA

2.1.1 Context for Embedded NoCs

The Versal NoC replaces multiple system-level interconnects that can be present in a SoC deployed in an Ultrascale+ device, the preceding line of FPGA devices from Xilinx [47]. In an Ultrascale+-based SoC, an IP module instantiated in the FPGA fabric that needs, for instance, to access DDR memory would need to be connected with fabric resources to one of the limited number of direct AXI links to a DDRMC.

Additionally, if it is necessary to share the DDRMC links with other modules, this connection will not be direct, instead happening through an interconnect also created with fabric resources. In the Xilinx IP library, the standard solution for creating such interconnects are multiplexer-based crossbars, such as the SmartConnect IP [48], which is limited to up to 16 clients.

A more scalable solution uses fabric resources to construct NoC structures, which transport data packets over multiple interconnected router modules. Following the terminology in [1, 2], we call such fabric-based implementations *overlay* (or "soft") NoCs, in opposition to *embedded* (or "hard") NoCs like the Versal NoC, which are a fixed part of the device die. Some important trends in the development of overlay NoCs include:

- Configuration options for different topologies and feature sets (e.g. CMU CONNECT [35] offers a web interface for generating HDL code for NoC variants with different topologies and features, [32] presents variants of a NoC based on butterfly fat-tree topology with different degrees of connectivity as expressed by the Rent parameter [6]).
- Architectural optimization taking characteristics of the FPGA fabric into account, such as fracturable LUTs (e.g. Penn Split-Merge [18], Hoplite [22]).
- Employing hard blocks included in more recent FPGA architectures (e.g. DSP blocks [9], memory cascades [21], high-speed wires [31], wide multiplexers [25]).

While the presence of a hard embedded NoC, such as the Versal NoC, mitigates the need for FPGA fabric-based interconnects, a complementary role for such interconnects may still be envisioned for projects that instantiate a large number of IP modules in the fabric (e.g. the GRVI Phalanx accelerator [16], which includes 400 soft RISC-V processors compared to the 54 NoC endpoints that the Versal NoC offers to the fabric).

2.1.2 Contemporary Embedded NoCs

NoC Architecture

In a Versal device overlay interconnects can be completely avoided if there are enough access links to the embedded NoC, which provides memory-mapped connectivity to any other client, including DDRMCs. The same applies for Speedster S7 line from FPGA developer Achronix Semiconductor [5], which also features an embedded NoC, with a different architecture and operating characteristics.

The idea of hardening a NoC as a global interconnect for the FPGA fabric has been explored in academia [15, 1] before the release of commercial solutions in the form of the Versal and Speedster S7 lines [5]. A prototype that uses *FabricLink* modules to interface

the FPGA fabric with the NoC [2], in particular, anticipated several aspects of these commercial NoC designs.

The three mentioned NoC architectures (Versal, Speedster S7 and FabricLink) share the strategy of implementing the "core" of the network as a high-frequency (1-2GHZ) clock domain, associated with interfacing modules to the FPGA fabric and to hard clients. The interfacing modules are responsible for the clock-domain crossing between the clock domains of the client and the NoC, as well as for translating between the communication protocols used by the client (e.g. AXI) and the internal protocol of the NoC. In Table 2.1, we see a comparison of the NoC models in terms of connectivity offered to the FPGA fabric.

Table 2.1: Connectivity offered to FPGA in each embedded NoC model

NoC Model	No. of Interfaces	Frequency	Data Width (user data)
FabricPort [2]	32	1.2 GHz	128
Versal (xcvc1902) [45]	54	1 GHz	182 (128)
Speedster S7 [4]	162	2 GHz	256 (192)

Another difference between the mentioned embedded NoC architectures is the topology, where each adopts a different approach: while the FabricLink-based NoC uses a simple square mesh, the Speedster S7 NoC is based on a peripheral ring NoC connected to multiple bidirectional links embedded in the fabric, and the Versal NoC uses a custom topology discussed in Section 3.3. These differences in topology are reflected in the approach to routing of each NoC, where the FabricPort and Speedster S7 NoCs employ straightforward routing algorithms based on destination address, while the less regular topology of the Versal NoC implements routing tables loaded at boot time.

Flow Control and Virtual Channels

An important aspect of NoC architecture, with implications to WCL analysis, is the flow control strategy adopted by the NoC, or how the transmission of flits is dynamically managed by the network to ensure NoC buffers never overflow. The Versal and FabricPort NoCs support multiple VCs, independent sets of buffers in the network with which packets are associated. The presence of multiple VCs helps avoid the phenomenon of transitive blocking [10], and, in the Versal NoC, provides support for differential QoS between CTs associated with different VCs [45].

The FabricPort, Speedster and Versal NoCs employ the *credit feedback* mechanism for flow-control, in which each output link maintains a counter of *credits* for each buffer

downstream from it (multiple buffers being possible when there are multiple VCs), with each credit corresponding to a free buffer space. Credits are returned as buffers are emptied via an auxiliary *credit feedback* signal with associated latency.

The term backpressure is employed for the situation in which flits of a packet are prevented from proceeding through a NoC link due to insufficient credits for the relevant buffer. In the context of the Versal NoC arbitration policy, described in Section 3.7, the presence of backpressure corresponds to a buffer blocked by arbitration Rule 3.

2.1.3 Trade-offs of Embedded NoCs

We may summarize the main advantages of the embedded hard NoC approach as follows [3, 43]:

- Improves area and power by reusing the same link between router for different CTs, and sacrificing bit-level control of the wires composing the NoC links (compared to using the FPGA fabric to implement an Overlay NoC) [43]. Savings in area also facilitate support for advanced features, such as multiple VCs.
- Improves frequency by implementing the NoC as a packetized, pipelined interconnect [43].
- Simplifies timing closure [45], by limiting the issue to the level of individual modules in the FPGA, while the timing correctness of the system-level NoC is insured by design.
- Beyond timing closure, the isolation between modules afforded by the NoC also simplifies support for modern FPGA trends such as partial reconfiguration and multi-tenant systems [3].

We may also consider the following disadvantages:

- The latency of a packet can vary in complex ways as a result of contention with other packets over multiple NoC switches (the issue explored in this text) [45].
- In order to use the NoC to transport data between two IP modules in the FPGA fabric, both modules are constrained to using a protocol supported by the NoC (in the case of the Versal NoC, one of the supported variants of the AXI protocol). This

issue is diminished by the current trend of structuring complex FPGA designs by integrating smaller IP modules, where the use of standard communication protocols is expected independently from the presence of the hard NoC [45].

The Intel Spiderweb NoC [27], proposed by another FPGA manufacturer, is presented as a "firm NoC" that is built with fabric resources, but is pre-floorplanned to improve performance and timing closure in a similar manner to embedded NoCs, while maintaining the flexibility of Overlay NoCs. In this sense, it represents a different approach to the trade-offs discussed above.

2.2 WCL Analysis for NoCs with Wormhole Routing

A number of different approaches for WCL analysis have been proposed for wormhole NoCs. We identify two important factors leading to this diversity:

- The different models adopted for the operation of NoC, particularly in terms of how different CTs contend for outputs.
- The different mathematical tools employed (e.g. Network Calculus [28], CPA [17]).

In this Section, we review WCL analysis approaches for *wormhole routing* NoCs [33], in which packets of data traverse the network as a sequence of flow-control digits (flits) in a pipelined manner, as opposed to *store-and-forward* mechanisms that store complete packets in each network switch before transmitting to the next one. As it allows for switches with significantly smaller buffer sizes, this mechanism is widely employed in contemporary NoCs, including the Versal NoC and the two other architectures discussed in Section 2.1.2.

Adopting the classification from [34], we can further divide Wormhole NoCs between *contention-less* and *contention-aware* models. We briefly review non-contention models in Section 2.2.1, and the two most widely employed contention-aware models in Sections 2.2.2 and 2.2.3.

2.2.1 Contention-less Models

Architectures where contention for NoC router outputs is avoided by design are particularly amenable to WCL analysis. Two common strategies for are:

- TDMA approaches, which determine at design time a schedule for the transmission of packets of each CT that avoids contention for switch outputs [38, 42].
- Virtual Circuit approaches, where all the links in the route of a packet are reserved before packet starts travelling [38, 42].

2.2.2 Prioritized VC Model

Among NoC models that do include contention between packets for router outputs, the *prioritized VC model* has received significant attention. In this model, each CT travels in its own VC (set of buffers in the network), associated with a priority level, and a higher priority CT can preempt a lower priority CT at any moment (flit-level preemption).

With the prioritized VC model, the WCL analysis can be performed with similar techniques to WCET analysis in uni-processors, treating the whole path followed by a packet as an atomic resource that has to be shared with other CTs [39]. A subsequent approach, known SLA, considers the individual links as resources instead [24].

The CPA approach, a different formalism that calculates latencies iteratively by propagating events from the outputs to the inputs of system elements, has also been applied to the prioritized VC model [36].

The identification of the phenomenon of MPB [53] showed that existing analysis approaches for the prioritized VC model were optimistic when *backpressure* can emerge due to limited buffer spaces, requiring corrections that increase pessimism [34].

Another solution proposed to address the MPB phenomenon is to extend the prioritized VC model to temporarily store packets that cannot proceed in NoC clients, which are assumed to have enough buffer space for complete packets. This eliminates backpressure, while keeping the VC model [8].

2.2.3 Round-Robin Model

While the prioritized VC model allows for applying existing WCET techniques in a natural manner, it is difficult to realize in practice, especially in designs with many CTs contending for an output.

Another widespread model has network routers performing RRA among input links: in this each input link has a single input buffer, and output link performs RRA between

said buffers. The winning buffer transmits a complete packet through the output before the RRA process is performed again.

The RRA-based model reflects the operation of NoCs present in commercial devices, for instance by Tilera [7] and Kalray [12]. Techniques that have been applied for WCL analysis in RRA-based NoC include RC [11, 30], which we review in detail in Section 4.2.1, Network Calculus [13] and CPA [37].

Chapter 3

Versal NoC System Model

In this chapter, we review details of the Versal NoC architecture and operation, in order to support the WCL analysis in Chapter 4. In Sections 3.1-3.3, we present the general architecture of the NoC (Sec. 3.1) and how a designer may configure it as part of a project (Sec. 3.2). These sections correspond to an "user" view of the NoC, as provided by the official documentation [49, 50] and tutorial [19] for the Versal NoC.

In Sections 3.3- 3.7 we describe additional details of the 1 GHz clock domain that corresponds to the "core" of the network, including the necessary information to develop a precise timing model. To provide this description, we complement our survey of the documentation by studying the NPS simulation source code available with Xilinx Vivado version 2020.3 [20], and simulating the complete system including NoC endpoints, for which simulation source code is not available.

While Section 3.3 (topology) and Section 3.6 (routing) are included to provide a better understanding of the Versal NoC, their content does not factor into the WCL analysis developed in Chapter 4.

3.1 Versal NoC Architecture

Figure 3.1 shows a block diagram of a Versal NoC, including the components of the network and different types of client that access it.

Internally, the vertical and horizontal segments of the NoC are composed of NPS modules, and packet flits traverse a sequence of NPS modules along their route in a pipelined

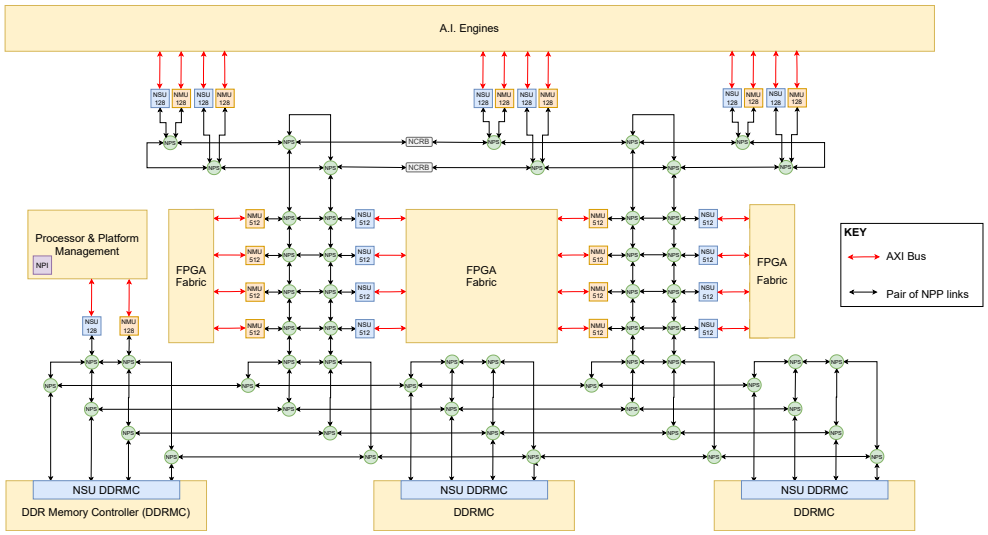


Figure 3.1: Block diagram of a Versal NoC.

manner according to *wormhole routing* [33]. We call the first and last flits of a packet the SoP and EoP flits, respectively. An HNoC may also include NCRB modules, which simply act as pipelining registers that ensure the timing closure of the NoC [52].

Network interface modules clients interact with the Versal NoC through NMU and NSU modules, which they access as AXI Slave and Master interfaces, respectively. Following the general embedded NoC paradigm discussed in Section 2.1.2, the NMU and NSU endpoints perform the clock domain crossing between the AXI interface and the internal 1GHz clock domain of the NoC, as well as translate the communication protocol between AXI and the internal NPP of the Versal NoC [49].

Clients in the FPGA fabric access the NoC via NMU 512/NSU 512 endpoints, which offer AXI interfaces with width of up to 512b, while hard blocks (such as AI engines and processors) access the NoC through NMU 128/NSU 128 endpoints. Finally, the Versal platform includes a number of DDRMC NSUs directly integrated in DDRMC modules, bypassing the need for an AXI interface.

The NoC is configured on boot time via a NPI module, present in the platform management system. The NPI programs the various NoC components through an auxiliary, low-throughput interconnect [49, 52].

3.2 Design Entry

Starting from version 2020.2, the Xilinx Vivado and Vitis design software support targeting Versal devices. In order to incorporate the Versal NoC as part of a design, a user must supply:

- The connectivity matrix of which NMUs communicate with which NSUs.
- The expected write and read bandwidth for each NMU/NSU pair.
- *Traffic classes* for AXI writes and reads, for each NMU/NSU pair.
- The AXI address address range of each NSU in the SoC memory map.

Three traffic classes are supported, with corresponding use cases described in the user guide [52]:

- ISOC traffic class, for reads or writes that exchange data with real-time constraints.
- LL traffic class, for reads that service cache misses.
- BE traffic class, for all other cases.

The traffic classes influence the solution produced by the NoC compiler. At runtime, they are currently only differentiated in that flits of ISOC and LL packets travel with a high-priority bit set (discussed in Sections 3.5 and 3.7), while flits of BE packets travel with the bit unset.

With the NoC parameters as inputs, a *NoC compiler* attempts to find a valid network configuration using a SAT solver on an appropriately constructed problem [45]. If a solution is found, the NoC compiler generates the configuration information for all NoC components. The compiler produces a solution report in a JSON file with `.ncr` extension. Below, we reproduce an entry from the report with the VC assignment and the route followed by packets of write data transmitted from an NMU to an NSU:

Listing 3.1: Example path data in a .ncr report produced by the NoC compiler

```
"PhyInstanceStart": "NOC_NMU512_X0Y0",
"PhyInstanceEnd": "NOC_NSU512_X0Y1",
"VC": 5,
"CommType": "WRITE",
"Connections": [
    "NOC_NMU512_X0Y0",
    "req_out",
    "NOC_NPS_VNOC_X0Y1",
    "port3_in",
    "NOC_NPS_VNOC_X0Y1",
    "port1_out",
    "NOC_NPS_VNOC_X0Y0",
    "port1_in",
    "NOC_NPS_VNOC_X0Y0",
    "port2_out",
    "NOC_NPS_VNOC_X0Y2",
    "port0_in",
    "NOC_NPS_VNOC_X0Y2",
    "port3_out",
    "NOC_NSU512_X0Y1",
    "req"
],
```

3.3 NoC Topology

Figure 3.2 shows the complete topology of the NoC embedded in the xcvc1902 Versal device, which is included in the VCK 190 evaluation board [46].

The HNoC and VNoC segments of the network are structured around lanes that encompass the whole segment, each capable of transmitting 128 Gbps of user data in each direction. Each VNoC has two lanes, one connected to fabric NMUs, other to fabric NSUs, for a total bandwidth of 512 Gbps. The number of lanes of an HNoC, on the other hand, is determined by whether it is connected to DDRMC ports (four lanes / 1024 Gbps), or to hard accelerators (two lanes / 512 Gbps) [43].

The Versal NoC topology is constructed with repeatable elements, to allow for adjusting

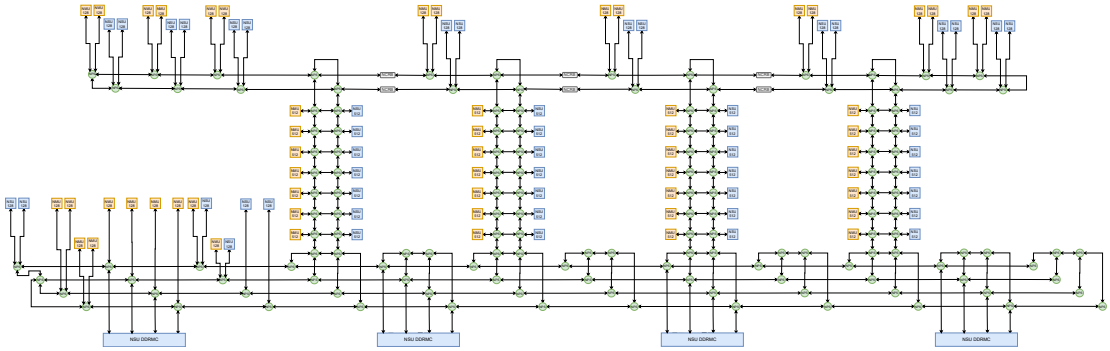


Figure 3.2: Complete topology of the NoC in the xcvc1902 Versal device.

the network dimension to different FPGA devices in the Versal line. A device with more DDRMC ports will be wider, while larger FPGA devices will use multiple silicon dies (each named a SLR), with NIDB components allowing VNoCs to cross between SLRs [45].

We may compare this custom topology with existing options by applying the concept of the rent parameter [6], which is determined by bisecting the NoC graph recursively, and modelling the number IO of NoC links cut by each bisection as:

$$IO = c \times N^p \tag{3.1}$$

where $c = 8$ is the number of ports in each switch, N is is the number of vertices in the graph to be bisected, and p is the rent parameter, with a higher parameter corresponding to a more richly connected network.

Figure 3.3 shows the result of the recursive bisections of the NoC graph in Figure 3.2, performed with the METIS graph partitioning library [23]. The rent parameter for the NoC is the slope of the trend-line in the plot, which is equal to 0.3. This place the rent parameter of the Versal NoC above that of a ring or binary tree NoC ($p = 0$), but below that of a mesh ($p = 0.5$) or a graph clique/crossbar ($p = 1.0$).

3.4 NPS Architecture

3.4.1 Buffers and Arbitration

Figure 3.4 shows the internal structure of an NPS. The NPS module is input queued and supports eight VCs: each input link has one VCB per VC, for a total of 32 VCBs in each

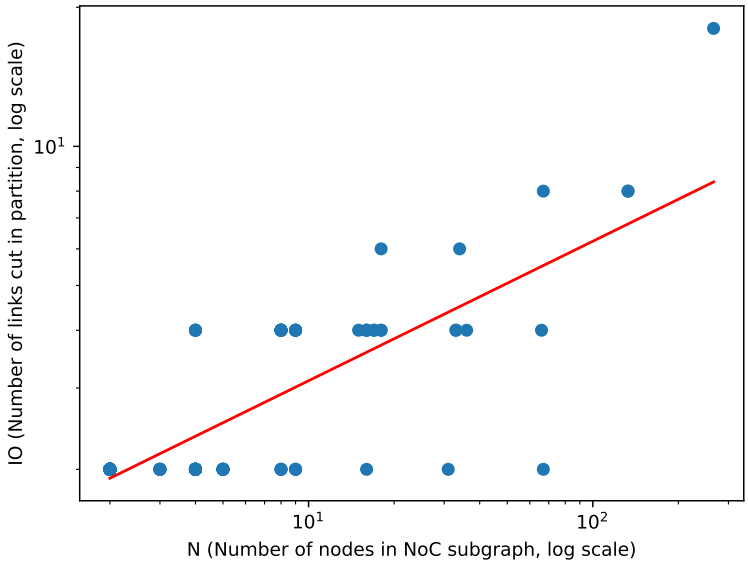


Figure 3.3: Plot for Rent Parameter calculation for the Versal NoC.

NPS – Figure 3.4 shows the VCBs of the top, left and bottom links. Each VCB has a depth of five in NPS modules belonging to VNoCs, and of seven in modules belonging to HNoCs.

Each packet traversing the network is associated with one of the eight VCs, determined by the NoC compiler, and its flits are stored in a corresponding VCB in each NPS they traverse. Each flit and its corresponding VC number are transmitted via the flit data and valid signals in Figure 3.4.

Routing is performed by the indexing a three-level segmented routing table with the input link, VC and destination address. Tables are indexed by VC, input link and destination address segment, and return the 2-bit ID of the output link [44].

Configuration data for each NPS, such as routing tables and token registers (discussed below in Section 3.4.2), is programmed by the NPI module on boot, and may be reprogrammed at run-time when the network is quiescent [49].

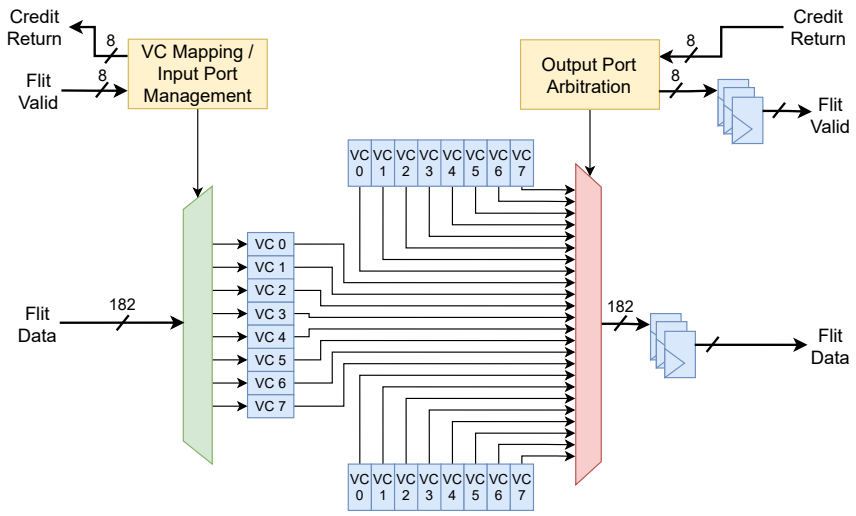


Figure 3.4: Block diagram of an NPS, including VCBs of three input link and details of input link and an output link.

3.4.2 Token Counters

As an additional QoS mechanism, each of the four output links of an NPS maintains an 8-bit token counter for each of the 24 VCBs that can target it, for a total of 96 token counters per NPS [49]. The token counters are used to implement an adapted version of the Deficit Round-Robin approach [44], which was originally developed for store-and-forward networks [41].

We refer to the token counter maintained by output link l for a VCB V as $V.c(l)$, and to the corresponding token register as $V.r(l)$. An SoP flit at the head of VCB V can only be transmitted through output link l if $V.c(l) \geq 0$, and if $V.c(l) = 0$ it is treated as low-priority even if its priority bit is high (with the other flits in the packet will still be treated as high priority). Each counter is associated with a configurable token register that provides its reset value.

We contextualize the token counters with respect to the complete arbitration policy implemented by NPS modules in Section 3.7.

3.5 NoC Packet Protocol (NPP)

Figure 3.4 includes the signals used in NPP to transmit flit data (*Flit Valid* and *Flit Data*) and to implement credit-based flow control (*Credit Return*).

The *Flit Valid* signal acts as a one-hot bitmask indicating which VC the flit belongs to, while the flit data includes 128 bits of payload data (either application data or encapsulated data about the AXI transaction) and a 54-bit NPP header. Figure 3.5 shows the generic format for flits transmitted over the NoC.

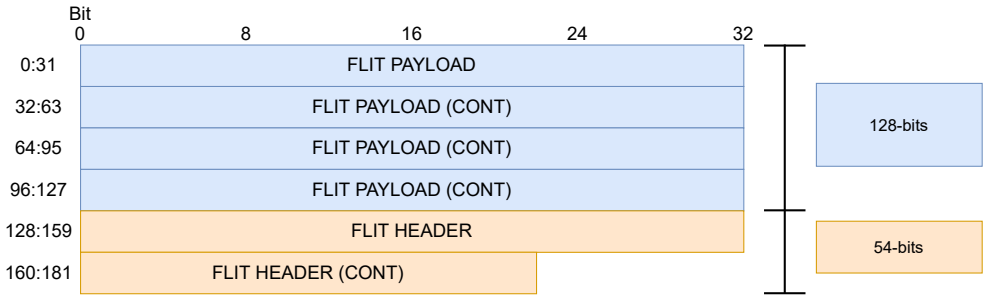


Figure 3.5: Generic format for flits traversing the Versal NoC.

The relevant fields of the header for understanding the arbitration policy described in Section 3.7 approach are:

- A one-bit field indicating high-priority: by simulating the Versal NoC (including NMU/NSU endpoints) with Vivado 2020.3, we have observed that LL and ISO flits always travel with this bit set, while BE packets travel with this bit unset.
- A one-bit field indicating if the flit is at the end its packet (EoP flit). SoP flits are implicitly detected when receiving a new flit after a reset or after an EoP flit.

Other fields in the flit header include:

- Two 12-bit fields, one with internal NoC addresses of the source endpoint, and one with the address of the destination endpoint (24 bits total). NPS modules use the destination NoC address for routing, as discussed in Section 3.6.
- A four-bit code identifying the type of AXI transaction.

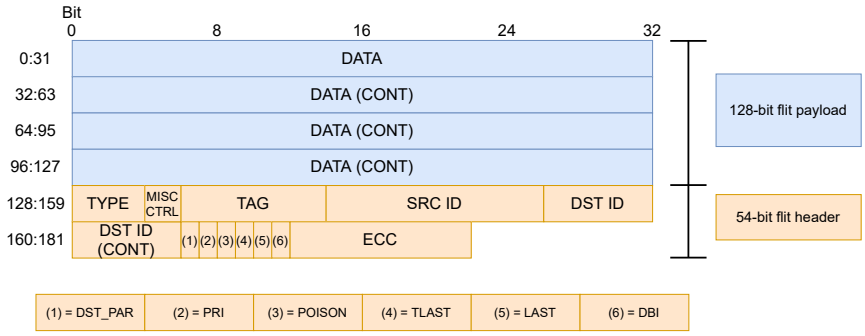


Figure 3.6: Format for data flits in the Versal NoC.

- A 10-bit Error Correction Code (ECC), which can be optionally enabled to detect errors in packet transmission on endpoints.

Regarding the payload-data field, data flits (Figure 3.6) use it to store user data in the payload field, while Control flits (Figure 3.7) use it to store encapsulated information about the AXI transaction, including address and AXI ID.

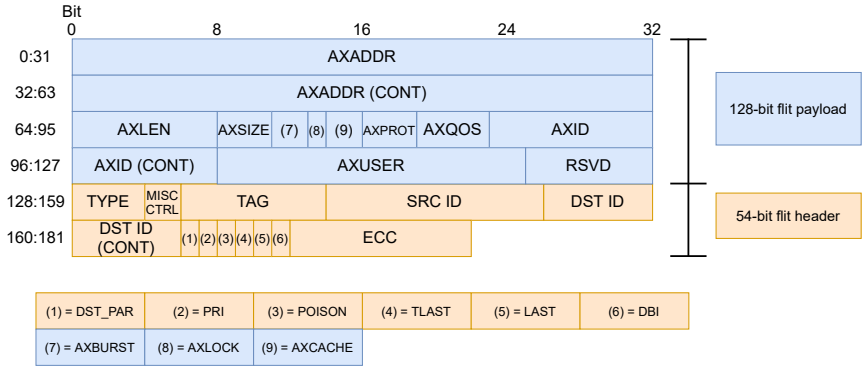


Figure 3.7: Format for control flits in the Versal NoC.

A packet of write data, as considered in Chapter 4, traverses the Versal NoC from NMU to NSU as a sequence of data flits with a control flit at the head, as seen in Figure 3.8.

As discussed in Section 2.1.2, the Versal NoC implements a credit-based flow control mechanism: each output link maintains one counter for each VCB downstream from it, keeping track of available buffer spaces (*credits*). Every clock cycle, the eight-bit *Credit Return* signal is read as a bitmask, with each set bit indicating that a credit is being returned to the corresponding VC.

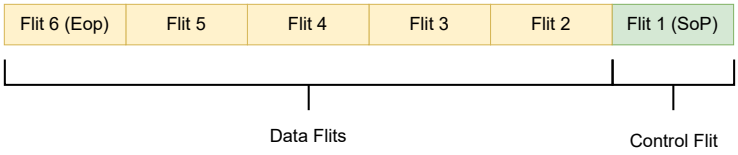


Figure 3.8: Packet with control and data flits.

3.6 Packet Routing

Each NPS is configured with three routing tables programmed via the NPI, organized hierarchically in order to reduce storage needs. In order to consult the routing tables, the 12-bit destination addresses of flits are split into three fields, each associated with one of the tables:

- A 6-bit HIGH ID (ID bits 11 to 6).
- A 2-bit MID ID (ID bits 5 to 4).
- A 4-bit LOW ID (ID bits 3 to 0).

NPS modules have their own MID and HIGH IDs programmed via the NPI. Algorithm 1 shows the policy followed by NPS modules to determine which table to consult. The routing tables are indexed with the corresponding ID field, VC number and link port [44].

Algorithm 1: Routing policy followed to determine the output link of each flit

```
1  if NPS HIGH ID ≠ flit HIGH ID then
2  |  output link ← HIGH ID table[input link][VC][flit HIGH ID];
3  end if
4  else
5  |  if NPS MID ID ≠ flit MID ID then
6  |  |  output link ← MID ID table[input link][VC][flit MID ID];
7  |  end if
8  |  else
9  |  |  output link ← LOW ID table[input link][VC][flit LOW ID];
10 |  end if
11 end if
```

We may quantify the space saved by the hierarchical organization by comparing it with what a single table with one entry for each of the 2^7 IDs required to cover all 120 NoC endpoints in the xcvc1902 Versal NoC. If said table were otherwise indexed in the same way (i.e. with ID, VC and ingress port), the required storage would be:

$$2^7 \text{ IDs} \times 8 \text{ VCs} \times 4 \text{ ports} \times 2 \text{ bits/entry} = 1024 \text{ bytes} \tag{3.2}$$

While the hierarchical organization with three tables used by the NPS modules requires a storage space of:

$$(2^6 + 2^4 + 2^2) \text{ IDs} \times 8 \text{ VCs} \times 4 \text{ ports} \times 2 \text{ bits/entry} = 627 \text{ bytes} \tag{3.3}$$

Which is 61.2% of the value obtained for the non-hierarchical routing table organization in Eq. 3.2.

We note that the NPS simulation code code defines a previous version of the names of the three fields, controlled by the macro `RT_2_43`:

- REGION ID (previous name for HIGH ID).
- LOCAL ID (previous name for MID ID).
- CHIP ID (previous name for LOW ID).

We hypothesize that this direct semantic meaning for the three address fields was abandoned to afford more flexibility to the SAT solver employed by the NoC compiler to generate all the NoC configuration parameters for a design.

3.7 Output Arbitration

At every cycle, each output link of an NPS performs LRU arbitration among a subset of VCBs that can send flits through it, possibly performing the arbitration among all 24 buffers from the other three links. For instance, the output link on the right in Figure 3.1 can be requested by any of the 24 VCBs from the three other links depicted.

The Versal NoC user guide presents a high-level description of five rules that determine the VCBs chosen to participate in arbitration each cycle [49]. We restate these rules with additional details in order to make a precise model possible. The first three rules are used to validate that the VCB has a flit that can be sent downstream:

- **Rule 1:** A VCB V must not be empty, and the flit at its head should be leave the NPS through link l.
- **Rule 2:** If another VCB V', associated with the same VC number as V, has a packet in progress leaving the switch through l, V cannot participate in arbitration for l until after the EoP flit of said packet wins arbitration for l. This prevents the flits of different packets from interleaving in the VCB downstream from l associated with the same VC number.
- **Rule 3:** A VCB V cannot participate in arbitration for the output link if the corresponding VCB downstream is full, as determined by the credit-based flow control mechanism.

The two remaining rules operate on the set of valid VCBs produced by the first three, to implement additional QoS mechanisms.

- **Rule 4:** This rule produces sets of high-priority and low-priority requests from the set of valid VCB requests produced by Rules 1 to 3, according to the following procedure:
  - High-priority requests correspond to VCBs that have a high-priority flit at the head and either have a $V.c(l) > 0$ or do not have a SoP flit at the head.
  - Low-priority requests, on the other hand, only require that VCBs either have a token counter $V.c(l) \geq 0$ or do not have a SoP flit at the head.
  - A VCB that has a SoP flit at the head and a token counter $V.c(l) < 0$ cannot be included in either group and is excluded from LRU arbitration in the given cycle.
- **Rule 5:** If the sets of high-priority and low-priority requests produced by Rule 4 are both non-empty, the low-priority requests are excluded from LRU arbitration in the cycle, including those corresponding to high-priority flits in a VCB with a token counter of 0 for the relevant output. Otherwise, if there are only high-priority priority requests or only low-priority priority requests, the LRU arbitration is performed among all the requests in the non-empty group.

In Algorithm 2, we can see the sequence in which these operations are performed every cycle by each output link. When a token counter $V.c(l)$ is reset at the end of a cycle on Line 6, it receives $V.r(l)$ if $V.c(l) \geq 0$, and $V.r(l) - 1$ otherwise.

Algorithm 2: Arbitration policy performed every cycle by each NPS output link

1 Determine valid requests (VCBs that pass Rules 1 to 3) ;
2 If there are any valid requests, and no valid request corresponds to a VCB with a token counter > 0, record that all token counters for this output link have to be reloaded ;
3 Determine low and high-priority requests according to Rule 4, among VCBs selected by **Line 1** ;
4 Perform LRU arbitration among high-priority requests, or among low-priority requests if there are no high-priority ones ;
5 If a VCB won LRU arbitration, decrement its token counter and move the corresponding flit to the output link ;
6 Reload all token counters if **Line 2** determined they should be reloaded ;

Chapter 4

WCL Analysis for Versal NoC

As discussed in Chapter 3, the Versal NoC switches contain multiple mechanisms that can interact to delay packets contending for an output link, requiring a custom WCL approach. Section 4.1 presents the problem statement we address, including the model adopted for CTs and other assumptions.

Section 4.2 reviews the RC WCL analysis approach and considers how to apply it to Versal NoC, by changing the optimization problem solved to determine the maximum delay a packet can experience locally in each NPS in its path. Sections 4.3, 4.4 and 4.5 detail how to calculate elements of the proposed WCL approach. Section 4.6 summarizes the complete WCL formulation.

Section 4.7 shows an illustrative example, comparing the bound given by the WCL approach for a packet traversing an NPS with a constructed trace of the switch status.

Finally, Section 4.8 discusses two alternative versions of the NPS arbitration policy that simplify the WCL analysis.

4.1 Problem Statement

Consider a set of real-time CTs $\Gamma = \{\tau_1, \ldots, \tau_N\}$ that transmit data over a Versal NoC that conforms to the architecture presented in Chapter 3 (in particular, we refer to the five arbitration rules presented in Section 3.7 simply as Rules 1 to 5).

Figure 4.1 shows an example with five real-time CTs that traverse a VNoC. According to the traffic class use cases discussed in Section 3.2, we model the real-time CTs with the

ISOC traffic class, which means that packet flits travel with the header priority bit set, as discussed in Sections 3.5 and 3.7.

Each real-time CT $\tau_i \in \Gamma$ is associated with the following parameters, accessed through dot notation (e.g. $\tau_i.D$ for the relative deadline of τ_i):

- $\tau_i.T$ is the minimum inter-arrival time, in NoC clock cycles, between releases of subsequent packets by the real-time CT τ_i.
- $\tau_i.J^R$ is the jitter in clock cycles on the release of a packet of the CT. This means that if a packet is generated at time t, the NMU can start sending it to the NPS at time $t' \in [t, t + \tau_i.J^R]$.
- $\tau_i.D$ is the relative deadline by which the packet of τ_i must be completely received. We assume constrained deadlines for all real-time CTs (i.e. $\forall \tau_i \in \Gamma, \tau_i.D \leq \tau_i.T - \tau_i.J^R$).
- $\tau_i.VC$ is the VC that packets of τ_i are associated with.
- $\tau_i.L$ is the length of a packet of τ_i in flits (including the SoP and EoP flits).
- $\tau_i.\delta$ is a sequence of NPP links corresponding to the route that a packet of τ_i travels through before being received in its destination.

In Figure 4.1, the route $\tau_1.\delta$ of CT τ_1 consists of three NPP links: from its origin NMU client to NPS 2, from NPS 2 to NPS 3, and from NPS 3 to its destination NSU client. We employ the following auxiliary functions for accessing a packet's route:

- $VCB(\tau_i, l)$ returns the NPS VCB that CT τ_i accesses the link l from.
- $firstLink(\tau_i)$/$lastLink(\tau_i)$ return the first/last link in $\tau_i.\delta$, respectively.
- $prevLink(\tau_i, l)$/$nextLink(\tau_i, l)$ return the link preceding/succeeding l in $\tau_i.\delta$, respectively.
- $prefixPath(\tau_i, l)$ returns the sub-sequence of $\tau_i.\delta$ up to but not including l, returning an empty sequence if $l = firstLink(\tau_i)$ (used in Section 4.5).

Besides real-time CT parameters, in order to account for NCRBs, which act as pipeline registers passing NPP signals along transparently while adding latency, we also parameterize each output link as:

- $l.b$ is the basic latency for sending a flit through link l: two cycles for a direct connection between NPS modules or between an NPS and an NoC endpoint; four cycles for a connection going through a NCRB.
- $l.F$ is the delay for receiving a credit through the credit feedback mechanism: one cycle for a direct connection between NPS modules or between an NPS and an NoC endpoint; two cycles for a connection going through a NCRB.

Finally, we must define the following parameters for VCBs:

- As in Section 3.7, we access the token counter and register maintained by output link l for VCB V as $V.c(l)$ and $V.r(l)$, respectively.
- $V.S$ is the VCB size in flits ($V.S = 5$ for NPS modules in VNoCs, $= 7$ for modules in HNoCs).

For convenience, we define additional properties of VCBs derived from which CTs are stored in each VCB:

- $V.\Gamma$ is the set of CTs that are stored in the VCB V.
- $V.\Gamma(l)$ is the set of CTs that are stored in the VCB V and leave the NPS through link l.
- $V.L(l)$ is the maximum packet size for a CT in $V.\Gamma(l)$.

We also consider a set of best-effort CTs $\Gamma^L = \{\tau_1^L, \ldots, \tau_{N^L}^L\}$ that traverse the network with the priority bit unset, modelled with the BE traffic class. As these are not real-time CTs, they do not have associated T, D, and J^R parameters, only the L, VC and δ parameters necessary for specifying a NMU/NSU connection in the Versal NoC. We do not calculate WCLs for these best-effort flows, but consider how they may contribute to delaying the real-time CTs via the token counter mechanism discussed in Chapter 3.

We may calculate the best-case (or structural) latency $C(\tau_i)$ for packets of each real-time CT $\tau_i \in \Gamma$ by summing the basic latencies over its route to get the time for the SoP flit to be received, and adding $\tau_i.L - 1$ cycles corresponding to the reception of the other flits in the packet:

$$C(\tau_i) = \left(\sum_{l \in \tau_i.\delta} l.b \right) + \tau_i.L - 1 \tag{4.1}$$

Our goal is to find the value of $R(\tau_i)$ for packets of each high-priority CT $\tau_i \in \Gamma$, which is the WCL between the arrival of a packet and its complete reception by the corresponding client, considering the possible interference from other CTs.

Further assumptions adopted for this chapter are listed below:

- For simplicity, we focus on writes from NMUs to NSUs, as packets of write and read data are treated in the same way by NPS arbitration, so all routes considered in this chapter begin in an NMU.
- We assume that each VC is associated either only with high-priority traffic, or only with low-priority traffic, which we observed to be the case when using the NoC compiler.
- We focus only on the 1GHz "core" network composed by the NPS modules, as the NMU/NSU are not documented in enough detail, and also because the NPP level of the NoC is independent of the AXI protocol layer, which can be seen with the NSU DDRMC interfaces directly embedded in the DDRMC modules. Figure 4.1 shows the abstract model of the NoC that we adopt for the analysis, with NMU/NSU clients playing the role of the corresponding NoC units and the clients connected to them.
- We adopt the following simple model for injection into the NoC: each NMU client can only have one high-priority packet in progress at a time, sending its flits as soon as possible (i.e. when the credit flow-control mechanism indicates there is free space in the appropriate VCB). The NMU performs RRA among real-time CTs with the same origin to decide the next real-time packet to send. Any low-priority CTs present in the system have their flits sent by the NMU only when a high-priority CT is not in progress, or when high-priority flits cannot be sent due to lack of buffer space (i.e. there is flit-level preemption of low priority packets by high-priority packets in the NMU client).
- We assume that the NSU clients are able to consume data and return credits fast enough that they never block any VCBs due to Rule 3.

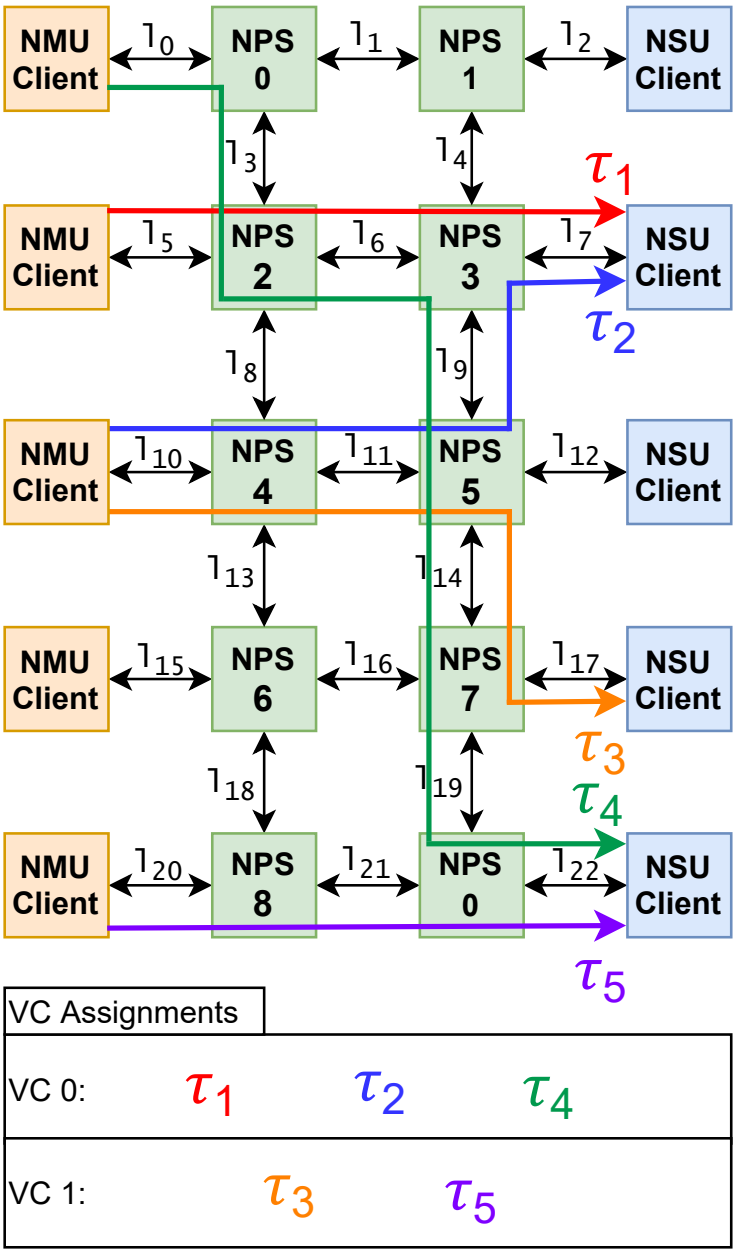


Figure 4.1: Example of five real-time CTs traversing a VNoC in a Versal NoC

4.2 RC Approach for Versal NoC

4.2.1 RC Review

We review the RC approach for WCL analysis in NoCs [14, 29], employing the notation from the problem statement in Section 4.1 in order to facilitate the adaptation of this approach to the Versal NoC.

As discussed in Section 2.2.3, the original RC is applied to an NoC model without multiple VCs, that employs wormhole routing and performs packet-level RRA among input buffers. Figure 4.2 shows a diagram of a switch that conforms to this model. As this NoC model does not support multiple VCs, real-time CTs do not have an associated VC parameter, but may still be defined with the other parameters described Section 4.1.

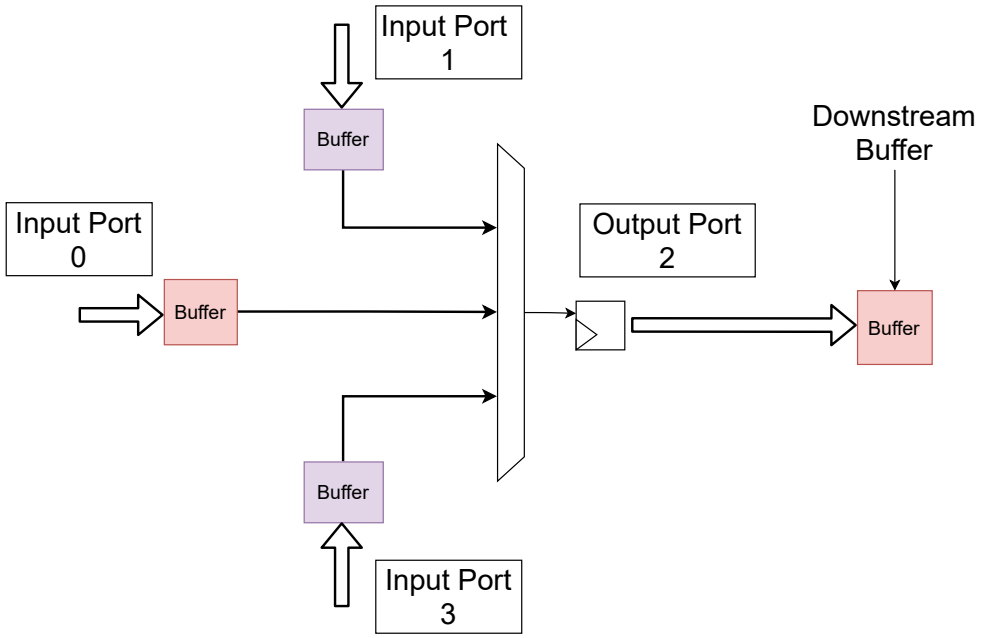


Figure 4.2: Diagram of a NoC switch that performs RRA among input port buffers, the object of the original RC approach

The core of the RC approach consists in evaluating a *delay function* $d(\tau_i, l)$. We employ the following definition for this function, in order for it to be applicable it for both the RRA and Versal NoC architectures:

Definition 1. *$d(\tau_i, l)$, for any $l \in \tau_i.\delta$, is the WCL for a packet of τ_i to be completely received at its destination after the two following conditions are verified:*

1. *The packet has won RRA at its origin to be transmitted (if $l = firstLink(\tau_i)$) or its SoP flit has reached the head of the buffer from which it accesses l (for any $l \neq firstLink(\tau_i)$). In the case of the Versal NoC, said buffer is $VCB(\tau_i, l)$.*
2. *After the SoP flit, the subsequent flits of the packet of τ_i are supplied continuously to the arbitration process for l as they are transmitted, until the EoP flit of the packet wins arbitration.*

We note that the second condition from Definition 1 is always true for the RRA NoC model as soon as the first condition is verified, as the non-SoP flits of τ will never lose arbitration against another CT for a link that the SoP has already traversed, only being prevented from progressing in a pipelined manner if the SoP flit itself is prevented from progressing due to losing RRA [14]. Thus the definition reduces to the first condition for that model, but not for the Versal NoC, as discussed in Section 4.2.2.

For the overall WCL $R(\tau_i)$ of a packet under analysis of τ_i, RC sums the terms $d(\tau_k, firstLink(\tau_i))$ over all other real-time CTs τ_k with the same origin as τ_i, including τ_i itself, as shown in Algorithm 3. This correspond to each real-time CT with the same origin as τ_i sending a complete packet before τ_i sends the packet under analysis.

Algorithm 3: RC Algorithm for computing $R(\tau_i)$, adapted from [29]

```
1 R(τ_i) ← 0;
2 forall τ_k such that firstLink(τ_i) = firstLink(τ_k), including τ_i do
3 |   R(τ_i) ← R(τ_i) + d(τ_k, firstLink(τ_i))
4 end forall
5 return R(τ_i);
```

To evaluate the function $d(\tau_i, l)$, RC follows the steps shown in Algorithm 4, which we comment on below.

In Figure 4.2, consider that a packet under analysis of the real time CT τ_i is stored in the buffer of input port 0 (in red) and leaves through port 2. In the worst case, each of the other two buffers (in purple) that can target output port 2 will win the RRA process once before the port 0 buffer, and send a complete packet through output port 2 before the SoP flit of the packet of τ_i may be sent through the same port.

Algorithm 4: RC Algorithm for computing $d(\tau_i, l)$ for a NoC that performs RRA between input ports, adapted from [29]

```
1  d(τ_i, l) ← 0 ;
2  if l ≠ firstLink(τ_i) then
3  |  forall input links l' ≠ prevLink(τ_i, l) do
4  |  |  delayFromLink ← 0 ;
5  |  |  forall τ_k that enter the switch through l' and leave through l do
6  |  |  |  delayFromCT ← 0 ;
7  |  |  |  if l = lastLink(τ_k) then
8  |  |  |  |  delayFromCT ← τ_k.L ;
9  |  |  |  else
10 |  |  |  |  delayFromCT ← l.b + d(τ_k, NextLink(τ_k, l)) ;
11 |  |  |  end if
12 |  |  |  delayFromLink ← max(delayFromCT, delayFromLink) ;
13 |  |  end forall
14 |  |  d(τ_i, l) ← d(τ_i, l) + delayFromLink ;
15 |  end forall
16 end if
17 if l = lastLink(τ_i) then
18 |  d(τ_i, l) ← d(τ_i, l) + l.b + τ_i.L − 1 ;
19 else
20 |  d(τ_i, l) ← d(τ_i, l) + l.b + d(τ, nextLink(τ_i, l)) + d_buf(τ_i, l) ;
21 end if
22 return d(τ_i, l) ;
```

This interaction with other NoC input buffers only applies if the link l is accessed from an NoC switch, meaning that $l \neq \mathit{firstLink}(\tau_i)$, as tested for in Line 2. If l is indeed accessed from an NoC switch, the nested loops in Line 3-15 determines the contribution of the delay due to contention with other CTs in the switch to the overall value of $d(\tau_i, l)$. Each iteration of the outer loop in Line 3-15 considers one of the input buffers that can interfere with τ_i, while the inner loop in Line 5-13 chooses the CT associated with said buffer that maximizes the contribution to the overall delay. We note that that the *If* condition in Line 2 is left implicit in the original pseudo-code for the delay function in [29].

The contribution of a packet of a specific CT τ_k being completely transmitted through the output link l before the packet under analysis is evaluated in Line 7-11. If l is the last link in the interfering packet's route, said contribution is calculated in Line 8 as the $\tau_k.L$ cycles necessary for the all the flits of the packet to be received at the NoC endpoint.

If l is not the last link, on the other hand, the delay calculation must account for the possibility that the packet of τ_k itself loses arbitration for an output further along in its route, and has to wait for another packet to be transmitted before proceeding. As the NoC works with a wormhole routing mechanism, this may happen while flits of τ_k are still waiting to be transmitted through l, in which case the packet of τ_i will be delayed further.

To account for this scenario, RC calculates the contribution of the interfering packet to the overall delay in Line 10 by calling the delay function recursively, with the interfering CT and the next link in its path as arguments. The result of this recursive call is added to the basic latency $l.b$ required for the SoP flit to reach the head of the buffer downstream from l (depicted on the right of Figure 4.2), thus establishing the first condition from Definition 1.

After the if-condition in Line 2-16 is exited, and the delay due to local contention with other CTs has been accumulated, Line 17-21 add the remaining delay terms necessary to bound the WCL for delivery of the packet under analysis of τ_i, depending on whether l is the last link in the route of the packet.

If l is the last link, Line 18 adds the basic latency $l.b$ for the SoP flit to traverse the last link and reach the NSU client, and $\tau_i.L - 1$ additional cycles for the reception of the non-SoP flits in a pipelined manner. We note that, while the original version of the algorithm [29] adds $\tau_i.L$ cycles rather than $\tau_i.L - 1$, the term $l.b$ already corresponds to the reception of the SoP flit of the packet, so it is valid to add only $\tau_i.L - 1$ cycles for the remaining flits of the packet.

On the other hand, if l is not the last link in the route, Line 20 adds the basic latency for l and a recursive call to the delay function with τ_i and the next link in the route as

arguments, similar to Line 10, as well as a *buffer delay* term d_{buf} corresponding to the worst-case delay for the buffer downstream from l to be emptied:

Definition 2. *For $l \neq lastLink(\tau_i)$, the buffer delay $d_{buf}(\tau_i, l)$ is an upper bound on the worst-case delay for the buffer $VCB(\tau_i, nextLink(\tau_i, l))$ that is downstream from l and holds τ_i to be emptied.*

Adding the buffer delay term $d_{buf}(\tau_i, l)$ is necessary because the first packet to be transmitted through l in the worst case scenario will need to wait until the buffer downstream from l is emptied to reach its head, in order for the delay function to be applicable.

The buffer delay term may be calculated as an ILP problem on variables counting the packets of each CT stored in the buffer downstream from l, as formulated for the RRA NoC model in [29]. In Section 4.4 we calculate the term $d_{buf}(\tau_i, l)$ for the Versal NoC following an analogous procedure.

Finally, we note that the RC approach assumes that no cyclical dependencies exist between flow routes, otherwise it would not be always possible to evaluate $d(\tau_i, l)$ by calling the function recursively. In a NoC with a conventional rectangular mesh topology, for instance, this property can be ensured by adopting an appropriate routing algorithm, such as DOR.

In the Versal NoC, the NoC Compiler ensures that the set of routing tables and addresses that the NPS modules are programmed with does not produce any cyclical dependencies [45].

4.2.2 RC on Versal NoC

To motivate adapting RC for the Versal NoC, and identify issues that must be addressed for doing so, we may compare the diagram of a Versal NPS module in Figure 4.3 with the switch model in Figure 4.2.

In a Versal NPS, arbitration for an output link is performed among VCBs instead of a single buffer per port. Given a CT under analysis τ_i, we define $\mathcal{V}(\tau_i, l)$ as the set of 23 other VCBs in the NPS, besides the buffer $VCB(\tau_i, l)$, that may target the output link l. In Figure 4.3 if VCB 0 of input port 3 (in red) contains a packet under analysis of τ_i that leaves through output port 2, the set $\mathcal{V}(\tau_i, l)$ consists in the 23 other VCBs depicted inside the switch (in purple).

We may partition $\mathcal{V}(\tau_i, l)$ into three subsets that interact with $VCB(\tau_i, l)$ in different manners as the flits of the packet of τ_i participate in arbitration:

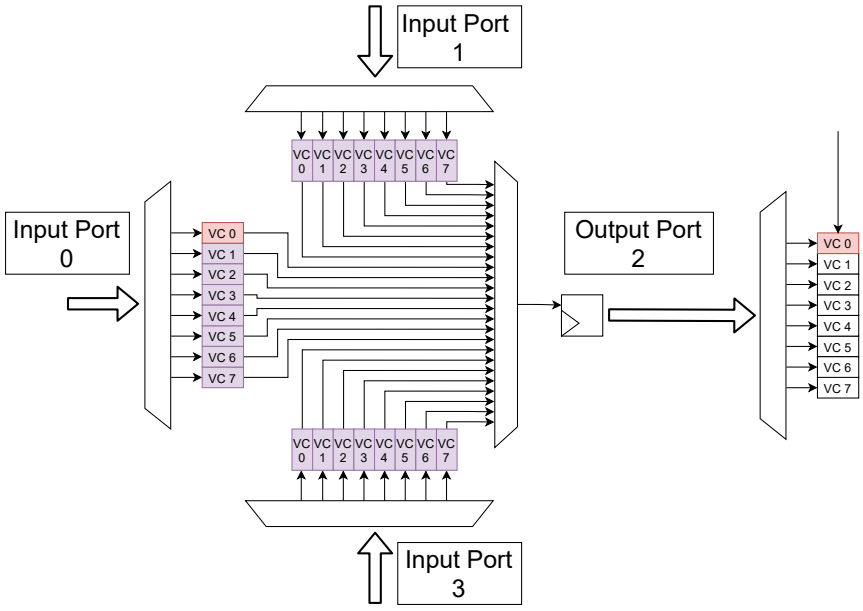


Figure 4.3: Diagram of an NPS module, for which we adapt the RC approach initially developed for the switch model depicted in Figure 4.2

- The set $\mathcal{V}^{SV}(\tau_i, l)$ containing the two other VCBs associated with $\tau_i.VC$ that can target l.
- The set $\mathcal{V}^{DVH}(\tau_i, l)$ of VCBs associated with a VC $\neq \tau_i.VC$ and high-priority traffic, that can target l.
- The set $\mathcal{V}^{DVL}(\tau_i, l)$ of VCBs associated with a VC $\neq \tau_i.VC$ and low-priority traffic, that can target l.

We motivate adapting RC to the Versal NoC by observing that arbitration Rule 2 enforces a similar packet-level ordering for packets stored in $\mathcal{V}^{SV}(\tau_i, l)$ with respect to the packet under analysis: if a packet from a CT τ_k stored in a VCB $V \in \mathcal{V}^{SV}(\tau_i, l)$ starts being transmitted before the SoP flit of the packet of τ_i is sent through l, the SoP flit of τ_i will be prevented from participating in arbitration at all until the packet of τ_k has been completely transmitted through l.

We may refer to Figure 4.1 to illustrate this phenomenon: consider CTs τ_1 and τ_4, which contend for an output link of NPS 2 and are both associated with VC 0. Because the CTs are associated with the same VC, if the SoP flit of τ_4 is sent before τ_1, τ_1 will be blocked

from participating in arbitration until the packet of τ_4 has been completely transmitted through the link they share. If τ_4 in turn is blocked further along in its path, in this case by contention with τ_5 for an output link of NPS 9, the packet of τ_i under analysis may be further delayed. We adopt an RC approach for the Versal NoC in order to capture this interaction by calculating the delay function recursively, similar to the procedure adopted in Line 10 of Algorithm 4.

In order to adapt the RC approach to an architecture that implements a more complex arbitration policy, we abstract the steps from Algorithm 4 as Algorithm 5, which shows a generic pseudo-code for evaluating the delay function:

Algorithm 5: Generic Algorithm for computing $d(\tau_i, l)$

```
1  d(τ_i, l) ← 0 ;
2  if l ≠ firstLink(τ_i) then
3  |  Evaluate d_local(τ_i, l) via local delay analysis;
4  |  d(τ_i, l) ← d_local(τ_i, l);
5  end if
6  if l = lastLink(τ_i) then
7  |  d(τ_i, l) ← d(τ_i, l) + l.b + τ_i.L − 1;
8  else
9  |  d(τ_i, l) ← d(τ_i, l) + l.b + d(τ_i, NextLink(τ_i, l)) + d_buf(τ_i, l);
10 end if
11 return d(τ_i, l);
```

For a RRA NoC, Line 3-4 in Algorithm 5 corresponds to the nested loops in Line 3-15 in Algorithm 4, which determine an upper bound on how many additional cycles the packet under analysis may be delayed due to losing arbitration to other CTs in the switch. This step of the analysis may be generalized as a *local delay analysis* step that calculates a *local delay term* $d_{local}(\tau_i, l)$:

Definition 3. *The local delay term $d_{local}(\tau_i, l)$ is an upper bound on how many cycles the flits of packet of τ_i under analysis are prevented from winning arbitration and accessing l due to the arbitration mechanisms for contention with other CTs:*

- *In the RRA NoC considered in the original RC, this corresponds to the number of additional cycles that the SoP flit of the packet of τ_i under analysis is prevented from accessing l due to another packet winning the RRA arbitration for the output first.*

- *In the Versal NoC, this corresponds to any cycle in which flits of the packet of τ_i are prevented from accessing l due to any of: Rule 2, Rule 4, Rule 5, or losing LRU arbitration. We may exclude Rule 1 from this definition because it necessarily only applies if a flit of τ_i is not at the head of the VCB, and Rule 3 because the cycles that flits of the packet are blocked from progressing only due to the flow-control mechanism are accounted for in the recursive call $d(\tau_i, nextLink(\tau_i, l))$.*

For $l \neq lastLink(\tau_i)$ the local delay term is calculated assuming that the VCB downstream from l that receives the flits of τ_i (which is $VCB(\tau_i, nextLink(\tau_i, l))$) starts empty. The worst-case delay for this said buffer to be emptied is added separately to the total delay as $d_{buf}(\tau_i, l)$ in Line 9 of the generic approach for calculating the delay function in Algorithm 5.

Line 3-15 in Algorithm 4 may then be understood as calculating the local delay term $d_{local}(\tau_i, l)$ as a simple optimization problem that consists in choosing one CT per contending buffer that has the maximum contribution to the total delay.

Line 6-10 of Algorithm 5 correspond to Line 17-21 of Algorithm 4, adding the necessary delay terms in function of whether $l = lastLink(\tau_i)$. To apply the delay function recursively in Line 9, we must consider presence of the credit feedback delay $l.F$. The credit feedback delay causes the condition of backpressure to be removed only after the returned credit for an open space in the buffer downstream from l is received, with latency $l.F$, and not immediately after the buffer space is freed. This could cause the continuous supply of flits to the arbitration process for the link $nextLink(\tau_i, l)$ to be interrupted, necessitating an additional correction factor to establish the second condition of Definition 1.

However, after SoP flit of the packet under analysis of τ_i reaches the head of the buffer $VCB(\tau_i, nextLink(\tau_i, l))$ that is downstream from l, the subsequent flits of the same packet will only suffer backpressure when accessing l if the buffer downstream from l is completely filled with other flits of τ_i. If the size of the buffer downstream from l is larger than the sum $l.F + l.b$, there is enough time for a credit to be returned, and for a flit of τ_i that was previously blocked by backpressure to reach the buffer, before the buffer becomes empty and the supply of flits for the arbitration process for $nextLink(\tau_i, l)$ is interrupted. In Lemma 4.2.1, we show that this condition is always the case in the Versal NoC.

Lemma 4.2.1. *In the Versal NoC, for all NPS VCBs V where flits received from a link l are stored, the condition in Eq. 4.2 is always verified.*

$$V.S \geq l.F + l.b \tag{4.2}$$

Proof. We show that the condition in Eq. 4.2 applies to all VCBs in the Versal NoC by considering the three possible cases for a link leading to an NPS:

- For a link l ending in an VNoC NPS, the sum $l.F + l.b = 1 + 2 = 3$ is smaller than the buffer size of 5 of VCBs in an VNoC NPS.
- For a link l ending in an HNoC NPS, that does not traverse an NCRB module, the sum $l.F + l.b = 1 + 2 = 3$ is smaller than the buffer size of 7 of VCBs in an HNoC NPS.
- For a link l ending in an HNoC NPS, that traverses an NCRB module, the sum $l.F + l.b = 2 + 4 = 6$ is again smaller than the buffer size of 7 of VCBs in an HNoC NPS.

□

Thus, in the Versal NoC the presence of the credit feedback delay $l.F$ never impedes the continuous supply of flits of the packet of τ_i to the next link in its route, and it is not necessary to add a corresponding correction factor before calculating $d(\tau_i, nextLink(\tau_i, l))$ in Line 9 of Algorithm 5.

Finally, we identify the following issues that must be addressed for a local delay analysis adapted for the Versal NoC, alongside the Section(s) of this Chapter where they are discussed:

Issue 1: Due to arbitration Rules 4 and 5, it is not the case that each VCB in $\mathcal{V}^{SV}(\tau_i, l)$ can only send a single packet before $VCB(\tau_i, l)$, as was the case for the buffers in the RRA-based model. This requires formulating the local delay analysis as a more complex ILP optimization problem (Section 4.3.3).

Issue 2: The interference from the VCBs in the subsets $\mathcal{V}^{DVH}(\tau_i, l)$ and $\mathcal{V}^{DVL}(\tau_i, l)$, which do not have analogues in the RRA NoC model, must be incorporated into the local delay analysis. As there is no packet-level ordering between the packet under analysis and packets stored in VCBs from these subsets, unlike with VCBs in $\mathcal{V}^{SV}(\tau_i, l)$, their interference may be accounted for without calling the delay function recursively (Section 4.3.3).

Issue 3: The original RC approach does not include the credit feedback delay $l.F$ in the model, which is necessary for the Versal NoC (this section and Section 4.3.1).

Issue 4: A correction factor must be added to the delay caused by packets stored in VCBs belonging to $\mathcal{V}^{SV}(\tau_i, l)$, because the second condition in Definition 1 is not verified immediately in the Versal NoC when the first one is, unlike the RRA NoC model.

We may again refer to Figure 4.1 to illustrate this phenomenon: consider flows τ_1 and τ_2, which contend for an output link of NPS 3 and are both associated with VC 0. The packet of CT τ_1 may be blocked by a complete packet of τ_2 due to Rule 2, only being allowed to proceed through the output link of NPS 3 after the EoP flit of the packet of τ_2 wins arbitration for said link.

Earlier (upstream) in its route, the CT τ_2 also contends with the CT τ_3 for an output link of NPS 4: because the two CTs are not associated with the same VC, there is no packet-level ordering between τ_2 and τ_3 enforced by Rule 2, so flits of τ_3 can interfere with non-SoP flits of τ_2 by winning the LRU arbitration. This interference from τ_3 may delay the arrival of the EoP flit of the packet of τ_2 at NPS 3, where the CT contends with τ_1, further delaying the progress of the packet of τ_1.

From the perspective of CT τ_1 at the NPS 3, this delay is experienced as a number of cycles where the VCB containing τ_2 fails to supply the subsequent flits of the interfering packet. We term this additional amount of delay cycles due to upstream interference experienced by CTs associated with $\mathcal{V}^{SV}(\tau_i, l)$ as "bubbles", as an analogy with bubbles in processor pipelines.

Thus, for the Versal NoC, a *bubbles delay term* $d_{bubbles}(\tau_k, l)$ must then be added to the equivalents of Line 8 and Line 10 in Algorithm 4 (Section 4.5):

Definition 4. *$d_{bubbles}(\tau_i, l)$ is an upper bound on how many cycles the VCB $VCB(\tau_k, l)$ may fail to supply a non-SoP flit to the arbitration process for the link l after the SoP flit of a packet of τ_k is transmitted, due to upstream interference from real-time CTs associated with other VCs.*

We note that it is not necessary to add a bubbles term $d_{bubbles}(\tau_i, l)$ to the equivalents of Line 18 and Line 20 in Algorithm 4 when evaluating $d(\tau_i, l)$, as, by the second condition of Definition 1, the flits of τ_i are supplied continuously for the arbitration for l.

Issue 5: If l is not the last link in the route of the packet under analysis, an appropriate d_{buf} term must be calculated, corresponding to the worst-case delay for the buffer $VCB(\tau_i, nextLink(\tau_i, l))$ that is downstream from l to be emptied (Section 4.4).

Issue 6: Given that multiple packets of each CT may interfere with the packet under analysis in each NPS, unlike in the RRA NoC, it is interesting to take the timing parameters of other real-time CTs into account in order to produce tighter bounds. In this Chapter, we use the function $packetCount(\tau_i, \tau_k)$ defined in Eq. 4.3 below for an upper bound on how many packets of a real-time CT τ_k can interfere with a packet of a CT under analysis τ_i in any given switch.

$$packetCount(\tau_i, \tau_k) = \left\lceil \frac{R(\tau_i) + \tau_k.J^R + R(\tau_k) - C(\tau_k)}{\tau_k.T} \right\rceil \tag{4.3}$$

As the formulation for *packetCount* function includes the WCL $R(\tau_i)$ of the packet under analysis itself, it will be necessary to apply the WCL method developed in this Chapter iteratively, starting from an estimation $R(\tau_i) = C(\tau_i)$. When we summarize the complete method in Section 4.6, we review the instances where the *packetCount* is employed and demonstrate that the iterative approach converges.

4.3 Local delay Analysis for Versal NoC

In this Section, we present an approach for the local delay analysis step that calculates $d_{local}(\tau_i, l)$ (according to Definition 3) for the Versal NoC as an ILP optimization problem [26].

We first discuss how to quantify the cycles of delay due to Rule 2 a specific interfering packet from a VCB $V \in \mathcal{V}^{SV}(\tau_i, l)$ can add to the total local delay, in order to get an expression analogous to Line 7-11 of Algorithm 4 (Section 4.3.1). We refer to the resulting expression for this per-packet delay due to Rule 2 over the rest of the Chapter.

We then describe how the transmission of a packet under analysis over an NPS may be broken into three time intervals (stages), with different interactions between the packet under analysis and the subsets of $\mathcal{V}(\tau_i, l)$ in each stage (Section 4.3.2).

Finally, we formulate the local delay analysis as a ILP problem on variables counting how many packets from each CT that is stored in a VCB in $\mathcal{V}^{SV}(\tau_i, l)$ are transmitted before the packet under analysis during each stage (Section 4.3.3).

4.3.1 Per-Packet Rule 2 delay

In Algorithm 4, Line 8-10 follow the procedure shown in Eq. 4.4 to determine an upper bound on how many cycles the packet under analysis must wait before a packet from CT τ_k is completely transmitted through l:

$$\begin{cases} \tau_k.L, & \text{if } l = lastLink(\tau_k) \\ d(\tau_k, nextLink(\tau_k, l)) + l.b, & \text{otherwise} \end{cases} \tag{4.4}$$

To formulate a similar expression for the Versal NoC, we must first address the possibility of *nested blocking*, by which we mean the situation in which:

1. The packet under analysis from τ_i is blocked via Rule 2 by a complete packet of τ_k, associated with a buffer in $\mathcal{V}^{SV}(\tau_i, l)$, while trying to access a link l.
2. Flits from VCBs in $\mathcal{V}^{DVH}(\tau_i, l)$, in turn, win LRU arbitration for l against the VCB containing the packet of τ_k.

While the situation of nested blocking does increase the number of cycles of blocking due to Rule 2, it is more convenient to calculate the contribution of nested delay cycles to the total local delay separately, in function of the total number of flits associated with $\tau_i.VC$ that leave through link l while the packet of τ_i is trying to access said link (which we do in Section 4.3.3).

Lemma 4.3.1 provides an upper bound $d_{R2}(\tau_i, l)$ for per-packet Rule 2 blocking in the Versal NoC, without considering eventual nested blocking cycles:

Lemma 4.3.1. *Without considering nested blocking cycles, the expression in Eq. 4.5 provides an upper bound to the total number of cycles that a packet under analysis τ_i that leaves through l is delayed by Rule 2 if a packet of CT τ_k stored in a VCB in $\mathcal{V}^{SV}(\tau_i, l)$ must be completely transmitted through l first:*

$$d_{R2}(\tau_k, l) = \begin{cases} \tau_k.L + d_{bubbles}(\tau_k, l), & \text{if } l = lastLink(\tau_k) \\ d(\tau_k, nextLink(\tau_k, l)) + d_{bubbles}(\tau_k, l) + l.b, & \text{otherwise} \end{cases} \tag{4.5}$$

Proof. If l is the last link in the route of τ_k, the packet of τ_k will never be prevented from progressing due to insufficient buffer space in the NSU client. Thus, the time for the packet to be completely received is the number of flits in the packet $\tau_k.L$, and the number of cycles the EoP flit of the packet is delayed by interference upstream in its route $d_{bubbles}(\tau_k, l)$.

On the other hand, if l is not the last link in the route, it is necessary to add the following values so the two conditions of Definition 1 are established for τ_k and the link $nextLink(\tau_k, l)$:

- A term $l.b$ corresponding to the delay for the SoP flit of τ_k to reach the head of the buffer downstream from l, which is $VCB(\tau_k, nextLink(\tau_k, l))$ (first condition).

- A term $d_{bubbles}(\tau_k, l)$, the delay until flits of τ_k supplied may continuously supplied to the arbitration process for $nextLink(\tau_k, l)$, even in the presence of bubbles (second condition).

We must also account for the presence of the credit feedback delay $l.F$, in a similar manner to the discussion in Section 4.2.2 regarding Line 9 of Algorithm 5. The presence of the credit feedback delay causes the condition of backpressure to be removed only after the returned credit for the open space is received, and not immediately after space opens up for it in the relevant buffer. This could cause the continuous supply of flits to the arbitration process for $nextLink(\tau_k, l)$ to be interrupted, necessitating an additional correction factor to establish the second condition of Definition 1.

However, after τ_k reaches the head of the buffer $VCB(\tau_k, nextLink(\tau_k, l))$ downstream from l, the subsequent flits of the packet of τ_k will only suffer backpressure when accessing l if said buffer is completely filled with other flits of τ_k. If the size of the buffer is larger than the sum $l.F + l.b$, there is enough time for a credit to be returned, and for a flit of τ_k that previously blocked by backpressure to reach the buffer, before the buffer empties and the supply of flits for $nextLink(\tau_k, l)$ is interrupted.

As shown in Lemma 4.2.1, the size of a buffer downstream from a link l is always greater than or equal to $l.F + l.b$ in the Versal NoC. Thus, it is not necessary to add a correction factor for the credit feedback delay in the Versal NoC.

With the two conditions of Definition 1 established, value $d(\tau_k, nextLink(\tau_k, l))$ is then the delay for the packet of τ_k to be received at its destination. □

4.3.2 Stages of Packet Transmission

Stage Limits

Let $t = 0$ be the clock cycle when the SoP flit of the packet under analysis τ_i reaches the head of $VCB(\tau_i, l)$. We define the following auxiliary points in time, which help us reason about the interaction with the VCBs in $\mathcal{V}(\tau_i, l)$:

- The cycle $t = t_R$, the first cycle after which Rules 4 and 5 cannot exclude the VCB containing the packet under analysis from participating in arbitration anymore. This correspond to either the cycle when a token reset causes $VCB(\tau_i, l).c(l)$ to become > 0 for the first time, or to the SoP of the packet winning arbitration with $VCB(\tau_i, l).c(l) = 0$.

- The cycle $t = t_S$, the last cycle when a flit from a VCB in $\mathcal{V}^{SV}(\tau_i, l)$ wins arbitration for l. The next flit associated with $\tau.VC$ to win arbitration for l will then be the SoP flit of the packet under analysis, after which and VCBs in $\mathcal{V}^{SV}(\tau_i, l)$ attempting to access l will be blocked by Rule 2 until the EoP flit of the packet under analysis is transmitted.
- The cycle $t = t_E$, the cycle when the EoP flit of the packet under analysis wins arbitration for l.

The four points in time $t = 0$, t_R, t_S and t_E delimit three time intervals (stages) for the transmission of the packet of τ_i over the NPS:

- A first stage for $0 \leq t \leq t_R$.
- A second stage for $t_R < t \leq t_S$.
- A third stage for $t_S < t < t_E$.

We may now consider how the VCBs belonging to each subset of $\mathcal{V}(\tau_i, l)$ may interfere with the packet under analysis of τ_i, during each of the three time intervals defined above. Figure 4.4, shows a schematic representation of the interactions we describe below in Sections 4.3.2-4.3.2.

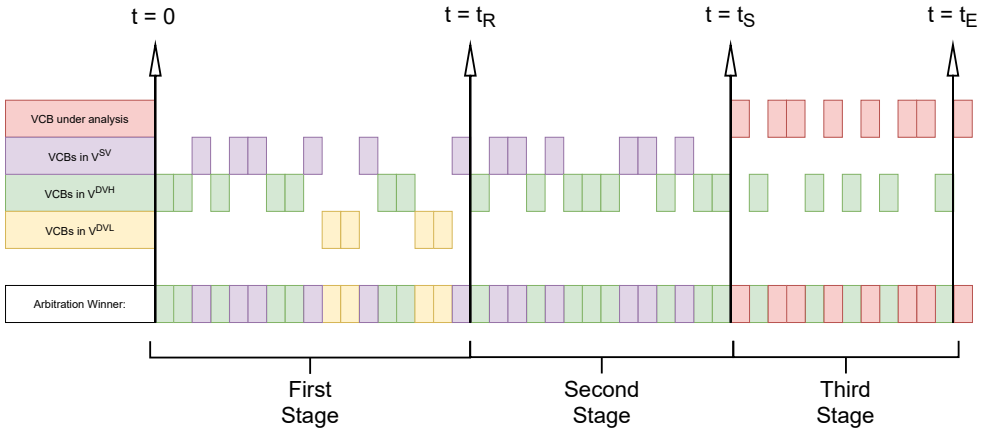


Figure 4.4: Diagram of the three stages of transmission through an NPS, showing which sets of VCBs can delay the packet under analysis by winning arbitration in each stage.

First Stage

During the interval $0 \leq t \leq t_R$, we assume that one of the following applies every cycle:

- The SoP flit of τ_i is blocked by one of Rule 4 or 5.
- The SoP flit of τ_i is blocked by Rule 2 or Rule 3, but would be blocked by Rule 4 or 5 anyway if it were not.

Any VCB $V \in \mathcal{V}(\tau_i, l)$ may extend this interval by sending a flit through l with a token counter > 0. The following lemma provides a limit on how many flits each VCB can send through l during this first stage:

Lemma 4.3.2. *A VCB $V \in \mathcal{V}(\tau_i, l)$ may only send up to $V.r(l) + V.L(l)$ flits through l before during the first stage, where $V_j.L(l)$ is the maximum length of a packet that crosses V_j and leaves through l.*

Proof. Starting with $V.c(l) = V.r(l)$, the VCB V may send $V.r(l) + 1$ flits with a non-negative counter, with the last one being a SoP flit that leaves $V.c = -1$.

Rule 4 only blocks buffers with a negative token counter from sending a flit if it is an SoP flit, so V can send the remaining $V.L(l) - 1$ flits of the packet it started without necessarily requiring a token reload, for a total of $V.r(l) + V.L(l)$ flits.

Sending any additional flits before the token reload, however, is not possible, as the next flit will necessarily have to be a SoP, which cannot be sent while $V.c(l) < 0$, due to Rule 4. □

Besides delaying t_R by sending flits, each packet from a CT τ_k associated with a VCB in $\mathcal{V}^{SV}(\tau_i, l)$ that is transmitted during the first stage can delay t_R for a total of $d_{R2}(\tau_k, l)$ cycles:

Lemma 4.3.3. *A packet of a flow τ_k stored in $\mathcal{V}^{SV}(\tau_i, l)$ that is transmitted through l during the first stage can delay t_R for up to $d_{R2}(\tau_k, l)$ cycles.*

Proof. If a VCB from $\mathcal{V}^{SV}(\tau_i, l)$ has a packet in progress during the first stage, and is blocked during a cycle by Rule 1 (due to bubbles) or Rule 3 (due to the flow-control mechanism), the token reload may also be delayed without any flits being transmitted or tokens being spent, as the VCB $VCB(\tau_i, l)$ is blocked by Rule 2 and thus cannot cause a token reload by itself. Thus, t_R may be delayed by the total number of cycles that the packet is blocked by Rule 2, which is given by $d_{R2}(\tau_k, l)$. □

Finally, this first interval is the only one during which the low-priority CTs stored in $\mathcal{V}^{DVL}(\tau_i, l)$ can delay the packet under analysis, while the high-priority packet under analysis is prevented from participating in arbitration, or is considered low-priority, by Rule 4 as a result of a non-positive token counter. From the definition of t_R, for $t \geq t_R$, Rule 4 cannot block the packet of τ_i under analysis or cause it to be considered as low-priority, so flits of τ_i will always be sent instead of low-priority flits if possible.

Second Stage

During the interval $t_R < t \leq t_S$, additional flits are sent by VCBs in $\mathcal{V}^{SV}(\tau_i, l)$, with the possibility of *nested blocking* (while this phenomenon may also happen during the first stage, during said stage the packet under analysis is also blocked by Rule 4 or Rule 5, so the nested delay does not contribute to delay the packet overall).

Without considering nested delay, the contribution to the local delay from each packet from a CT τ_k associated with a VCB in $\mathcal{V}^{SV}(\tau_i, l)$ is given by $d_{R2}(\tau_k, l)$ by definition.

After the second interval, VCBs belonging to $\mathcal{V}^{SV}(\tau_i, l)$ cannot interfere with the packet under analysis anymore, due to Rule 2.

Third Stage

During the interval $t_S < t < t_E$, the packet under analysis can only be blocked by VCBs belonging to $\mathcal{V}^{DVH}(\tau_i, l)$, which can win the LRU arbitration for the link l.

4.3.3 ILP Formulation

Problem Variables

We formulate the local delay analysis step for the Versal NoC as an ILP optimization problem on variables $x(\tau_k)$, $y(\tau_k)$, $z(\tau_k)$ and $w(\tau_k)$ defined for each CT τ_k that is stored in a VCB $V \in \mathcal{V}^{SV}(\tau_i, l)$ and contends with τ_i for l. The variables have the following meanings:

- $0 \leq x(\tau_k) \leq 1$ counts the number of packets of τ_k in progress at $t = 0$.
- $0 \leq y(\tau_k)$ counts the number of packets of τ_k completely sent in the first stage of transmission ($0 < t \leq t_R$).

- $0 \leq z(\tau_k) \leq 1$ counts the number of packets of τ_k in progress when $t = t_R$.
- $0 \leq w(\tau_k) \leq 1$ counts the number of packets of τ_k completely sent in the second stage of transmission ($t_R < t \leq t_S$). This variable is necessarily ≤ 1 because the LRU arbitration would not let a second packet win against the packet under analysis after the latter is not blocked by Rule 4 anymore.

Constraints on Problem Variables

We may add the following constraints on the values of the variables $x(\tau_k)$, $y(\tau_k)$, $z(\tau_k)$ and $w(\tau_k)$, derived from the NoC architecture and the problem statement:

Constraint 1: The sum of the variables counting the packets of each CT belonging to a VCB in $\mathcal{V}^{SV}(\tau_i, l)$ cannot go over the number of packets of the flow that can effectively arrive while the packet of τ_i is traversing the NoC:

$$\forall V \in \mathcal{V}^{SV}(\tau_i, l), \forall \tau_k \in V.\Gamma(l), x(\tau_k) + y(\tau_k) + z(\tau_k) + w(\tau_k) \leq \mathit{packetCount}(\tau_i, \tau_k) \quad (4.6)$$

Constraint 2: Due to Rule 2, there can be at most one packet in progress from a VCB in $\mathcal{V}^{SV}(\tau_i, l)$ through l at $t = 0$:

$$\sum_{V \in \mathcal{V}^{SV}(\tau_i, l)} \sum_{\tau_k \in V.\Gamma(l)} x(\tau_k) \leq 1 \quad (4.7)$$

Constraint 3: Likewise, there can be at most one packet in progress from a VCB in $\mathcal{V}^{SV}(\tau_i, l)$ through l at $t = t_R$:

$$\sum_{V \in \mathcal{V}^{SV}(\tau_i, l)} \sum_{\tau_k \in V.\Gamma(l)} z(\tau_k) \leq 1 \quad (4.8)$$

Constraint 4: Due to LRU arbitration, each VCB in $\mathcal{V}^{SV}(\tau_i, l)$ can send at most one packet after t_R, as the SoP flit of an eventual second packet would lose arbitration to the SoP of the τ_i packet:

$$\forall V \in \mathcal{V}^{SV}(\tau_i, l), \sum_{\tau_k \in V.\Gamma(l)} w(\tau_k) \leq 1 \quad (4.9)$$

Constraint 5: Each VCB in $\mathcal{V}^{SV}(\tau_i, l)$ can either send anything in the interval $0 \leq t < t_R$, or start a packet after t_r, but not both, as sending anything before t_R would cause it to lose LRU arbitration to the SoP of τ_i.

$$\forall V \in \mathcal{V}^{SV}(\tau_i, l), \left(1 - \sum_{\tau_k \in V.\Gamma(l)} w(\tau_k)\right) * M \geq \sum_{\tau_j \in V.\Gamma(l)} (x(\tau_j) + y(\tau_j) + z(\tau_j)) \tag{4.10}$$

Where:

$$M = \sum_{\tau_k \in V.\Gamma(l)} packetCount(\tau_i, \tau_k) \geq \sum_{\tau_j \in V.\Gamma(l)} (x(\tau_j) + y(\tau_j) + z(\tau_j)) \tag{4.11}$$

Constraint 6: The sum of flits sent through l during the first stage (i.e. $0 \leq t < t_R$) must be less than or equal to $V.r(l) + V.L(l)$, for each VCB in $\mathcal{V}^{SV}(\tau_i, l)$, otherwise a token reload would happen before t_R.

We assume the extreme case where the packets that are only partially transmitted during $0 \leq t < t_R$ (counted by $x(\tau_k)$ and $z(\tau_k)$) only consume one token each.

$$\forall V \in \mathcal{V}^{SV}(\tau_i, l), \sum_{\tau_k \in V.\Gamma(l)} (x(\tau_k) \times 1) + (y(\tau_k) \times \tau_k.L) + (z(\tau_k) \times 1) \leq V.r(l) + V.L(l) \tag{4.12}$$

Objective Function

Finally, we must define the local delay term $d_{local}(\tau_i, l)$ as the objective function to be maximized in the ILP optimization problem. We express this objective function as a sum:

$$d_{local}(\tau_i, l) = 1 + d^{SV}_{local}(\tau_i, l) + d^{DVH}_{local}(\tau_i, l) + d^{DVL}_{local}(\tau_i, l) \tag{4.13}$$

Where:

- The one cycle added as the first term corresponds to the cycle needed for a token reset if the token counter $VCB(\tau_i, l).c(l)$ is initially negative, which may happen independently of any other CTs.

- $d_{local}^{SV}(\tau_i, l)$ is the number of cycles that VCBs from $\mathcal{V}^{SV}(\tau_i, l)$ can delay t_R during the first stage (Lemma 4.3.3) or delay t_S via Rule 2 during the second stage, without considering nested delay ($d_{R2}(\tau_k, l)$ by definition).
- $d_{local}^{DVH}(\tau_i, l)$ is the number of cycles that VCBs from $\mathcal{V}^{DVH}(\tau_i, l)$ can delay t_R by sending flits (during the first stage), delay t_S via nested blocking (during the second stage) or win LRU arbitration against the packet under analysis (during the third stage).
- $d_{local}^{DVL}(\tau_i, l)$ is the number of cycles that VCBs from $\mathcal{V}^{DVL}(\tau_i, l)$ can delay t_R by sending flits (during the first stage).

The term $d_{local}^{SV}(\tau_i, l)$ may be calculated in function of the ILP problem variables, by applying Lemma 4.3.1 and Lemma 4.3.3:

$$d_{local}^{SV}(\tau_i, l) = \sum_{V \in \mathcal{V}^{SV}(\tau_i, l)} \sum_{\tau_k \in V.\Gamma(l)} (x(\tau_k) + y(\tau_k) + z(\tau_k) + w(\tau_k)) \times d_{R2}(\tau_k, l) \tag{4.14}$$

For the term $d_{local}^{DVH}(\tau_i, l)$, we sum the individual contribution $d_{local}^{DVH,V}(\tau_i, l)$ of each $V \in \mathcal{V}^{DVH}(\tau_i, l)$:

$$d_{local}^{DVH}(\tau_i, l) = \sum_{V \in \mathcal{V}^{DVH}(\tau_i, l)} d_{local}^{DVH,V}(\tau_i, l) \tag{4.15}$$

Each $V \in \mathcal{V}^{DVH}(\tau_i, l)$, in each of the three stages, necessarily must send a flit through l in order to interfere with the packet under analysis (either to delay t_R in the first stage or to win LRU arbitration in the second and third stages). Thus, each term $d_{local}^{DVH,V}(\tau_i, l)$ is upper-bounded by the number of flits the VCB V can send through link l while the packet of τ_i is contending for said link:

$$\forall V \in \mathcal{V}^{DVH}(\tau_i, l), d_{local}^{DVH,V}(\tau_i, l) \leq \sum_{\tau_k \in V.\Gamma(l)} packetCount(\tau_i, \tau_k) \times \tau_k.L \tag{4.16}$$

On the other hand, the number of flits each $V \in \mathcal{V}^{DVH}(\tau_i, l)$ can send during the first stage is limited by Lemma 4.3.2. Additionally, during the second and third stages, each $V \in \mathcal{V}^{DVH}(\tau_i, l)$ can only delay the packet under analysis by winning LRU arbitration

against another flit associated with the VC $\tau_i.VC$ (either flits of τ_i itself during the third stage, or flits stored in a buffer in $\mathcal{V}^{DVH}(\tau_i, l)$ during the second stage). We may add these per-stage bounds to arrive at another bound for each term $d_{local}^{DVH,V}(\tau_i, l)$:

$$\begin{aligned}&\forall V \in \mathcal{V}^{DVH}(\tau_i, l), d_{local}^{DVH,V}(\tau_i, l) \leq \\ &V.r(l) + V.L(l) + \tau_i.L + \sum_{V_j \in \mathcal{V}^{SV}(\tau_i, l)} \sum_{\tau_k \in V_j.\Gamma l} x(\tau_k) + y(\tau_k) + z(\tau_k) + w(\tau_k)\end{aligned} \tag{4.17}$$

Each term $d_{local}^{DVH,V}(\tau_i, l)$ is then obtained as the minimum of the two bounds:

$$\begin{aligned}&\forall V \in \mathcal{V}^{DVH}(\tau_i, l), d_{local}^{DVH,V}(\tau_i, l) = \min(\\ &V.r(l) + V.L(l) + \tau_i.L + \sum_{V_a \in \mathcal{V}^{SV}(\tau_i, l)} \sum_{\tau_k \in V_a.\Gamma l} x(\tau_k) + y(\tau_k) + z(\tau_k) + w(\tau_k), \\ &\sum_{\tau_k \in V.\Gamma(l)} packetCount(\tau_i, \tau_k) \times \tau_k.L \\ &)\end{aligned} \tag{4.18}$$

Finally, the term $d_{local}^{DVL}(\tau_i, l)$ may be calculated by applying Lemma 4.3.2 to find upper bounds for how many flits each VCB in $\mathcal{V}^{DVL}(\tau_i, l)$ can send during the first stage, and summing over all such VCBs:

$$d_{local}^{DVL}(\tau_i, l) = \sum_{V \in \mathcal{V}^{DVL}(\tau_i, l)} V.r(l) + V.L(l) \tag{4.19}$$

4.4 Buffer delay Term

In this Section, we derive an upper bound for the buffer delay term $d_{buf}(\tau_i, l)$ defined in Section 4.2.1, the worst-case delay for the buffer downstream from l where the packet of τ_i is stored. We adopt the procedure from [29], using, for each packet of a CT τ_k completely of partially stored in the buffer downstream from l, the value $d(\tau_k, nextLink(\tau_k, l))$ as its contribution to the buffer delay.

The following Lemma, adapted from the analogous result for RRA NoCs in [29] (Theorem 1) provides a description of the initial state of the buffer downstream from l that produces the worst-case buffer delay:

Lemma 4.4.1. *The downstream VCB $d(\tau_k, nextLink(\tau_k, l))$ can only hold up to two partial packets, one at head and other at the tail of the buffer. The buffer delay $d_{buf}(\tau_i, l)$ is maximized when there is only one partial packet, at the head of the VCB.*

Proof. The possible presence or absence of the two partial packets results in four cases to be considered:

1. **Case 1:** No partial packets present in the VCB.
2. **Case 2:** Only one partial packet present in the VCB, at the head of the buffer.
3. **Case 3:** Only one partial packet present in the VCB, at the tail of the buffer.
4. **Case 4:** Two partial packets present in the VCB.

We consider the other cases in turn to show that Case 2 maximizes the delay:

- For **Case 1**, if the complete packet of a CT τ_k at the head of the VCB is moved forward, until only its EoP remains (resulting in **Case 2**), there will be more space at the tail of the buffer for other flits, while the contribution of the packet at the head to the buffer delay remains $d(\tau_k, nextLink(\tau_k, l))$. Even if the space that is freed does not allow for a complete packet to be stored, the total delay will not decrease, so Case 1 may not produce a larger delay than Case 2.
- For **Case 3**, if the packet at the tail of the VCB is moved forward until it is completely stored in the buffer, resulting in Case 1 or Case 2, the VCB from $\mathcal{V}^{SV}(\tau_i, l)$ sending the partial packet will be able to send another complete packet of the same CT before the packet under analysis, resulting in a larger delay.
- For **Case 4**, demonstrated in the same manner as case 3.

□

We may then calculate as an ILP problem in terms of the packets stored in the buffer. Given Lemma 4.4.1, we define the following variables for each CT in that is stored in the downstream VCB $VCB(\tau_i, nextLink(\tau_k, l_i))$ (except τ_i itself):

- $0 \leq A(\tau_k) \leq 1$ counts the number of packets of τ_k completely stored in the buffer $VCB(\tau_i, nextLink(\tau_k, l_i))$ that is downstream from l.

- $0 \leq B(\tau_k) \leq 1$ counts the number of packets of τ_k partially stored at the head of the buffer $VCB(\tau_i, nextLink(\tau_k, l_i))$.

We may apply the following constraints to the problem variables:

Constraint 1: At most one packet of each CT may be stored in the VCB, as we have constrained deadlines.

$$\forall \tau_k \in VCB(\tau_i, nextLink(\tau_i, l)).\Gamma \setminus \tau_i, A(\tau_k) + B(\tau_k) \leq 1 \tag{4.20}$$

Constraint 2: The buffer space occupied by all packets stored in the buffer downstream from l must not be larger than the size $VCB(\tau_i, nextLink(\tau_i, l)).S$ of the buffer.

$$\sum_{\tau_k \in VCB(\tau_i, nextLink(\tau_i, l)).\Gamma \setminus \tau_i} A(\tau_k) \times \tau_k.L + B(\tau_k) \times 1 \leq VCB(\tau_i, nextLink(\tau_i, l)).S \tag{4.21}$$

We maximize $d_{buf}(\tau_i, l)$ as the objective function, adding the credit feedback delay for the transmission of the last flit.

$$d_{buf}(\tau_i, l) = \sum_{\tau_k \in VCB(\tau_i, nextLink(\tau_k, l_i)).\Gamma \setminus \tau_i} (A(\tau_k) + B(\tau_k)) \times d(\tau_k, l_{j+1}) + l.F + 1 \tag{4.22}$$

4.5 Bubbles delay Term

In this Section, we derive an upper bound for the bubbles delay term $d_{bubbles}(\tau_i, l)$ defined in Section 4.2.2. The Lemma below provides an upper bound for this term:

Lemma 4.5.1. *The expression in Eq. 4.23 below provides an upper bound on how many cycles the VCB $VCB(\tau_i, l)$ may fail to supply a non-SoP flit to the arbitration process for the link l after the SoP flit of a packet of τ_i is transmitted, due to upstream interference from real-time CTs associated with other VCs.*

$$d_{bubbles}(\tau_i, l) = \sum_{l' \in prefixPath(\tau_i, l)} \sum_{V \in \mathcal{V}^{DVH}(\tau_i, l')} \min \left(\tau_k.L - 1, \sum_{\tau_k \in V.\Gamma(l')} packetCount(\tau_i, \tau_k) \times \tau_k.L \right) \tag{4.23}$$

Proof. The bubbles delay term results from non-SoP flits of the packet of τ_i losing arbitration for links along its path $prefixPath(\tau_i, l)$ to the NPS with l as an output link.

For a link $l' \in prefixPath(\tau_i, l)$, the non-SoP flits of the packet may only lose arbitration to VCBs in $\mathcal{V}^{DVH}(\tau_i, l')$, as VCBs in $\mathcal{V}^{SV}(\tau_i, l')$ are blocked by Rule 2 and Rule 4 does not apply to non-SoP flits. Each VCB in $\mathcal{V}^{DVH}(\tau_i, l')$ may win LRU arbitration once against each of the $\tau_i.L - 1$ non-SoP flits of the packet of τ_i.

Additionally, the number of times each VCB in $\mathcal{V}^{DVH}(\tau_i, l')$ may win LRU arbitration is limited by the number of flits $\sum_{\tau_k \in V.\Gamma(l)} packetCount(\tau_i, \tau_k) \times \tau_k.L$ it has available for doing so. For the per-VCB contribution to the bubbles delay, we take the minimum of the two bounds.

For the total bubbles delay term, we sum this per-VCB contribution over all VCBs in $\mathcal{V}^{DVH}(\tau_k, l')$, for each $l' \in prefixPath(\tau_k, l)$. □

4.6 WCL Approach Summary

We summarize how to apply Sections 4.2 - 4.5 to calculate WCL bounds for each real-time CT. To evaluate the WCL function $R(\tau_i)$, we may simply use the same Algorithm 3 employed for the RRA NoC model, as the same assumptions apply for how packets are injected. For the delay function $d(\tau_i, l)$, we employ the generic approach in Algorithm 5, formulating the local analysis step according to Sections 4.3-4.5.

We must address the use of the function *packetCount*, which calls the WCL function $R(\tau_i)$, producing a recurrence relation. We proceed by calculating WCL values iteratively, in similar manner to existing WCL approaches based on WCET techniques for processors [39, 24], producing a sequence of vectors of WCL values $\mathbf{R_0}, \mathbf{R_1}, \ldots$.

Each vector of WCL values has N elements, with the i-th entry corresponding to the WCL value for τ_i. For iteration number $j \geq 1$, when calculating the elements of the vector $\mathbf{R_j}$, any time the function *packetCount* is evaluated according to the definition in Eq. 4.3 we apply the following procedure:

- the i-th entry $\mathbf{R_{j-1}}$ is used for the value $R(\tau_i)$.
- the k-th entry $\mathbf{R_{j-1}}$ is used for the value $R(\tau_k)$.

For the initial vector $\mathbf{R_0}$, we use the values provided by the best-case delay function $C(\tau_i)$, which is guaranteed to be inferior or equal to the actual WCL values for each CT.

The iterative process is completed when the WCL values are unchanged by an iteration, or when a WCL value is obtained that does not respect the deadline for a real-time CT. In Lemma 4.6.1, we show that this approach completes after a finite number of iterations:

Lemma 4.6.1. *The series of iterations producing the vectors of WCL values*

$$\mathbf{R_0}, \mathbf{R_1}, \ldots$$

ends after a finite number of steps to either converge to a vector WCL values or show that a deadline is not respected.

Proof. We employ the function *packetCount*, defined in Eq. 4.3, in the following three instances: Eq. 4.6, Eq. 4.18 and Eq. 4.23. We first show that the entries of vectors $\mathbf{R_0}, \mathbf{R_1}, \ldots$ can only increase or remain constant in each iteration:

- In Eq. 4.6 the function *packetCount* is used for a bound on the sum of ILP problem variables representing packets of CTs from $\mathcal{V}^{SV}(\tau_i, l)$ that can interfere with the packet under analysis. If the value returned by *packetCount* increases, due to $R(\tau_i)$ or $R(\tau_k)$ increasing, the number of packets that can interfere also increases, subsequently increasing the local delay.
- In Eq. 4.18, the *packetCount* function is used for a bound on how many flits from a buffer in $\mathcal{V}^{SV}(\tau_i, l)$ can delay the packet under analysis. If the value returned by *packetCount* increases, due to $R(\tau_i)$ or $R(\tau_k)$ increasing, the number of flits that can interfere also increases, subsequently increasing the local delay.
- In Eq. 4.23, the *packetCount* function is used for a bound on how many flits from a buffer in $\mathcal{V}^{SV}(\tau_k, l)$ can delay the non-SoP flits of the packet for which the bubbles term is being calculated. If the value returned by *packetCount* increases, due to $R(\tau_i)$ or $R(\tau_k)$ increasing, the number of flits that can interfere also increases, subsequently increasing the bubbles delay term, and the local delay experienced by the packet under analysis.

As all values in question are integers, the WCL values are bounded by their deadlines and only increase or remain constant, the iterative process will complete after a finite number of steps.

□

4.7 Illustrative Example

In this Section, we present an example WCL calculation for a simplified scenario where packets traverse a single NPS module (Section 4.7.1), and compare the WCL value obtained for the packet under analysis with a cycle-by-cycle trace of the NPS module state corresponding to Worst-Case interference (Section 4.7.2).

4.7.1 Example Upper Bound

Example Configuration

Figure 4.5 shows a simplified configuration with three NMU clients connected to a NSU client over a single NPS module. The NMU clients execute a total of five real-time CTs τ_1-τ_5 and two best-effort CTs τ_1^L and τ_2^L, each associated with a different VCB in the switch. We focus on τ_1 as the CT under analysis.

Table 4.1 shows the parameters for the different CTs. Additionally, we assume that all token registers in the NPS are configured with a value of 3.

Table 4.1: Example CT parameters

CT	VCB	L	T	D	J^R
τ_1^H	Port 0, VC 0	6	200	200	0
τ_2^H	Port 1, VC 0	3	100	100	0
τ_3^H	Port 3, VC 0	3	100	100	0
τ_4^H	Port 1, VC 1	3	100	100	0
τ_5^H	Port 3, VC 3	3	100	100	0
τ_1^L	Port 1, VC 4	3	-	-	-
τ_2^L	Port 3, VC 6	3	-	-	-

With τ_1 as the packet under analysis, the other CTs are associated with VCBs from all three subsets of $\mathcal{V}(\tau_1, l_2)$ as defined in Section 4.2.2.

The basic latencies for each real-time CT, given that each packet traverses two links, are calculated with Eq. 4.1 as:

$$\begin{aligned} &C(\tau_1) = 9,\ C(\tau_2) = 6,\ C(\tau_3) = 6, \\ &C(\tau_4) = 6,\ C(\tau_5) = 6 \end{aligned} \tag{4.24}$$

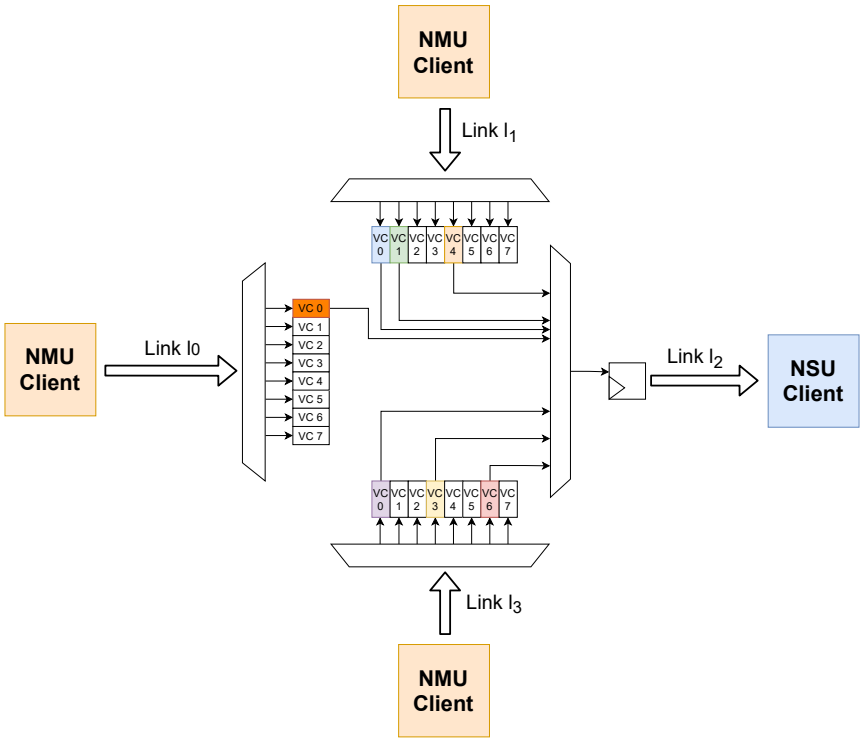


Figure 4.5: Example configuration with a single NPS.

First iteration

For the first iteration, following the procedure delineated in Section 4.6, we use the best-case values in Eq. 4.24 for the vector $\mathbf{R_0}$:

$$\mathbf{R_0} = [9, 6, 6, 6, 6]^T \tag{4.25}$$

We may then evaluate how many packets of each real-time CT may interfere with the packet of τ_1:

$$\begin{aligned} &packetCount(\tau_1, \tau_2) = 1,\ packetCount(\tau_1, \tau_2) = 1, \\ &packetCount(\tau_1, \tau_2) = 1,\ packetCount(\tau_1, \tau_2) = 1 \end{aligned} \tag{4.26}$$

The CTs associated with VCBs in $\mathcal{V}^{SV}(\tau_1, l_2)$, for which the ILP problem variables are defined, are τ_2 and τ_3. A solution to the ILP problem is:

$$\begin{aligned} &x(\tau_2) = 0,\ y(\tau_2) = 0,\ z(\tau_2) = 0,\ w(\tau_2) = 1, \\ &x(\tau_3) = 0,\ y(\tau_3) = 0,\ z(\tau_3) = 0,\ w(\tau_3) = 1 \end{aligned} \tag{4.27}$$

We evaluate the local delay based on the ILP problem variables with Section 4.3.3. The term $d_{local}^{DVL}(\tau_1, l_2)$ corresponding to interference from best-effort CTs does not vary with subsequent iterations and is calculated with Eq. 4.19:

$$d_{local}^{DVL}(\tau_1, l_2) = \sum_{V \in \mathcal{V}^{DVL}(\tau_1, l_2)} V.r(l_2) + V.L(l_2) = (3 + \tau_1^L.L) + (3 + \tau_2^L.L) = 12 \tag{4.28}$$

The term $d_{local}^{SV}(\tau_1, l_2)$ is calculated in function of the ILP variables with Eq. 4.14:

$$\begin{aligned} &d_{local}^{SV}(\tau_1, l_2) = \sum_{V \in \mathcal{V}^{SV}(\tau_1, l_2)} \sum_{\tau_k \in V.\Gamma(l_2)} (x(\tau_k) + y(\tau_k) + z(\tau_k) + w(\tau_k)) \times d_{R2}(\tau_k, l_2) = \\ &d_{R2}(\tau_2, l_2) + d_{R2}(\tau_3, l_2) = 10 \end{aligned} \tag{4.29}$$

The term $d_{local}^{DVH}(\tau_1, l_2)$ is calculated in function of the ILP variables with Eq. 4.15:

$$d_{local}^{DVH}(\tau_1, l_2) = \sum_{V \in \mathcal{V}^{DVH}(\tau_1, l_2)} d_{local}^{DVH,V}(\tau_1, l_2) = \min(12, 3) + \min(12, 3) = 6 \tag{4.30}$$

Where 12 is the number of flits associated with $\tau_1.VC = 0$ that leave through l_2 (two packets with $L = 3$ and the packet under analysis with $L = 6$).

The overall local delay term $d_{local}(\tau_1, l_2)$ is then:

$$d_{local}(\tau_1, l_2) = 1 + d_{local}^{SV}(\tau_i, l_2) + d_{local}^{DVH}(\tau_i, l_j) + d_{local}^{DVL}(\tau_i, l_2) = 1 + 10 + 6 + 12 = 29 \tag{4.31}$$

And the overall $R(\tau_1)$ in this simple NoC configuration is:

$$R(\tau_1) = d_{local}(\tau_1, l_2) + \tau_1.L + l_0.C + l_2.C - 1 = 38 \tag{4.32}$$

By applying the a similar process for CTs τ_2 to τ_5, we arrive at the vector $\mathbf{R_1}$:

$$\mathbf{R_1} = [38, 73, 73, 73, 73]^T \tag{4.33}$$

Where the other CTs have larger WCL due to sharing an NMU client output link.

Second Iteration

With the vector $\mathbf{R_1}$, we may evaluate the function packetCount for the second iteration:

$$\begin{aligned} &packetCount(\tau_1, \tau_2) = 2,\ packetCount(\tau_1, \tau_2) = 2, \\ &packetCount(\tau_1, \tau_2) = 2,\ packetCount(\tau_1, \tau_2) = 2 \end{aligned} \tag{4.34}$$

With the updated packet counts, a solution to the local delay analysis ILP problem is:

$$\begin{aligned} &x(\tau_2) = 0,\ y(\tau_2) = 1,\ z(\tau_2) = 1,\ w(\tau_2) = 0, \\ &x(\tau_3) = 0,\ y(\tau_3) = 2,\ z(\tau_3) = 0,\ w(\tau_3) = 0 \end{aligned} \tag{4.35}$$

The term $d^{SV}_{local}(\tau_1, l_2)$ may again be calculated in function of the ILP variables with Eq. 4.14:

$$d^{SV}_{local}(\tau_1, l_2) = \sum_{V \in \mathcal{V}^{SV}(\tau_1, l_2)} \sum_{\tau_k \in V.\Gamma(l_2)} (x(\tau_k) + y(\tau_k) + z(\tau_k) + w(\tau_k)) \times d_{R2}(\tau_k, l_2) = \\ 2 \times d_{R2}(\tau_2, l_2) + 2 \times d_{R2}(\tau_3, l_2) = 20 \tag{4.36}$$

The term $d^{DVH}_{local}(\tau_1, l_2)$ is also again calculated in function of the ILP variables with Eq. 4.15:

$$d^{DVH}_{local}(\tau_1, l_2) = \sum_{V \in \mathcal{V}^{DVH}(\tau_1, l_2)} d^{DVH,V}_{local}(\tau_1, l_2) = \min(15, 6) + \min(15, 6) = 12 \tag{4.37}$$

The overall local delay term $d_{local}(\tau_1, l_2)$ for the second iteration is then:

$$d_{local}(\tau_1, l_2) = 1 + d^{SV}_{local}(\tau_i, l_2) + d^{DVH}_{local}(\tau_i, l_j) + d^{DVL}_{local}(\tau_i, l_2) = 1 + 20 + 12 + 12 = 45 \tag{4.38}$$

And the complete vector $\mathbf{R_2}$ is:

$$\mathbf{R_2} = [51, 91, 91, 91, 91]^T \tag{4.39}$$

Third Iteration

With the vector $\mathbf{R_2}$, we may evaluate the function *packetCount* for the third iteration:

$$\begin{aligned} &packetCount(\tau_1, \tau_2) = 2, \ packetCount(\tau_1, \tau_2) = 2, \\ &packetCount(\tau_1, \tau_2) = 2, \ packetCount(\tau_1, \tau_2) = 2 \end{aligned} \tag{4.40}$$

The packet counts are the same as in the second iteration, so the value of $R_1(\tau_1)$ will remain the same. The other WCL values also remain the same in this iteration:

$$\mathbf{R_3} = [51, 91, 91, 91, 91]^T \tag{4.41}$$

Thus, no more iterations are necessary.

4.7.2 Example Trace

In this section, we show a trace corresponding to the worst case interference suffered by a packet of CT τ_1, and compare with the WCL bound obtained in Section 4.7.1. The scenario depicted corresponds to the solution for the ILP problem:

$$\begin{aligned} x(\tau_2) = 0,\ y(\tau_2) = 1,\ z(\tau_2) = 1,\ w(\tau_2) = 0, \\ x(\tau_3) = 0,\ y(\tau_3) = 2,\ z(\tau_3) = 0,\ w(\tau_3) = 0 \end{aligned} \tag{4.42}$$

And to the packet counts:

$$\begin{aligned} \mathit{packetCount}(\tau_1, \tau_2) = 2,\ \mathit{packetCount}(\tau_1, \tau_2) = 2, \\ \mathit{packetCount}(\tau_1, \tau_2) = 2,\ \mathit{packetCount}(\tau_1, \tau_2) = 2 \end{aligned} \tag{4.43}$$

Figure 4.6 shows how we represent the state of the NPS module at every clock cycle, including all information necessary to determine the arbitration result each cycle.

The meaning of each element labeled in Figure 4.6 is as follows:

1. Current clock cycle, with $t = 0$ being the first cycle with the SoP flit of the packet of τ_1 at the head of the VCB.
2. VCB whose state is represented by the three rows to its right.
3. Token Counter Value for the VCB during the clock cycle.
4. Clock cycle when the VCB last won arbitration, relative to $t = 0$ (necessary to determine the winner of LRU arbitration).
5. Flit at the head of the VCB.
6. VCB that wins arbitration in the cycle.
7. Flit at the head of the winning VCB.
8. Rules determining why the VCB containing τ_1 does not win arbitration in the cycle, or `None` if it wins arbitration.

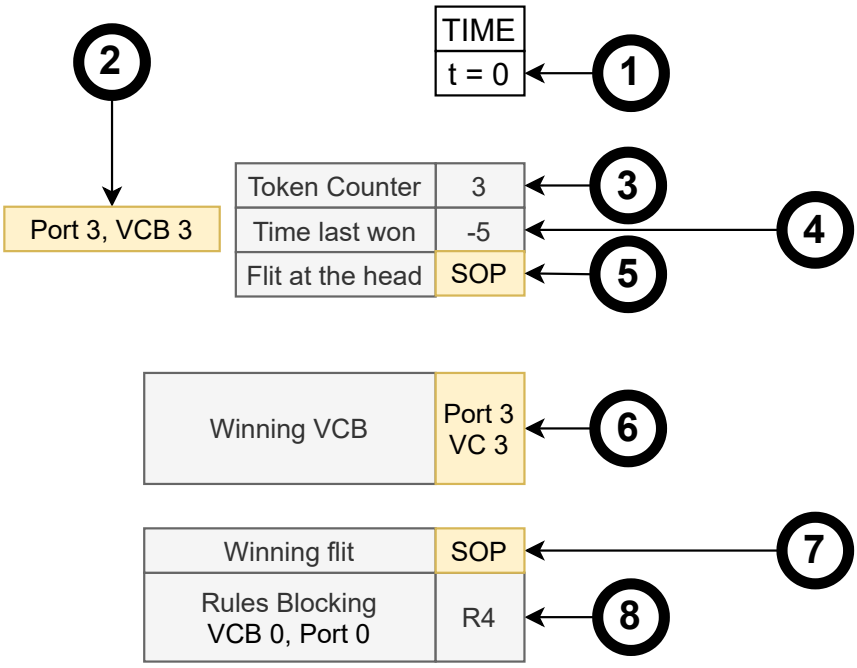


Figure 4.6: Key for diagram elements in the example trace.

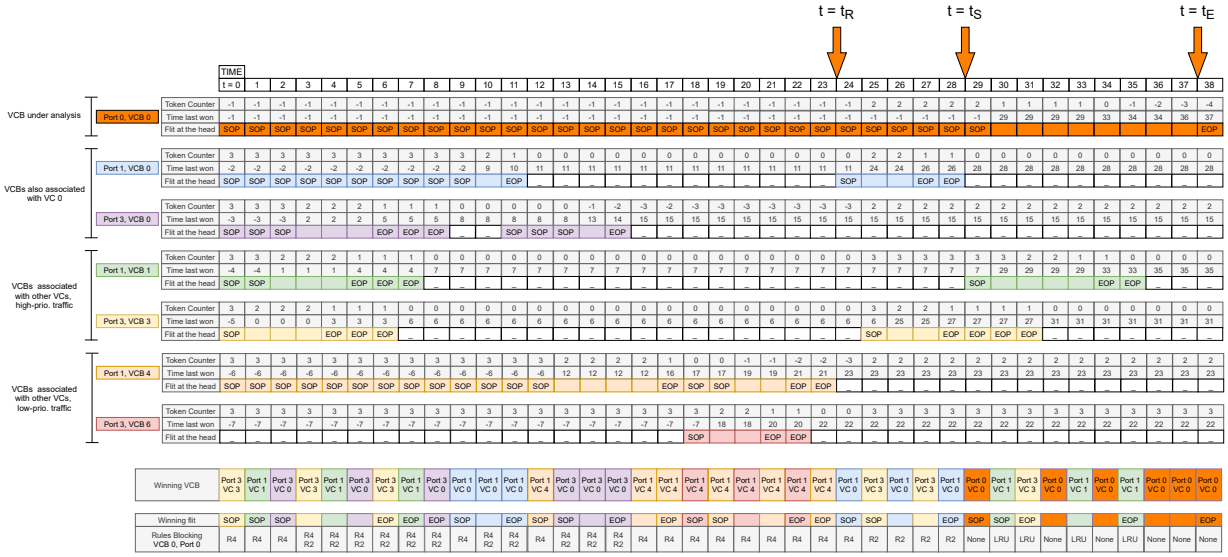


Figure 4.7: Example Trace.

Figure 4.7 shows the complete trace for a packet of CT τ_1 traversing the NPS. We comment below on the events depicted in the Figure by referring to the time values.

At $t = 0$, all SoP flits of packet of τ_1 under analysis is at the head of VCB 0 of port 0. The token counter for this VCB maintained by output link l_2 is negative, so the SoP flit of the packet under analysis is blocked by Rule 4 (R4). Additionally, the VCBs in $\mathcal{V}(\tau_1, l_2)$ with associated CTs all have a packet at the head leaving through link l_2, except for VCB 6 of port 3. Among VCBs with high-priority flits, VCB 3 of Port 3 wins LRU arbitration.

At $t = 2$, VCB 0 of Port 3 wins arbitration for l_2 with the SoP flit of a packet, so VCB 0 of Port 1 will be blocked from participating in arbitration until the EoP of the packet is transmitted.

At $t = 11$, a second packet arrives at the head of VCB 0 of Port 3, but it cannot participate in arbitration until the packet of VCB 0 of Port 1 in progress is completely transmitted.

At $t = 12$, the low-priority flit in VCB 4 of Port 1 wins LRU arbitration because the token counter for VCB 0 of Port 3 is equal to 0, so the flit at the head of the VCB is treated as low-priority.

At $t = t_R = 24$, the conditions for a token reset are fulfilled, because VCB 0 of Port 0 and VCB 0 of port 1 are requesting the link l_2 without being blocked by Rule 2 or Rule 3, and neither has tokens. From this point on, the packet under analysis is never blocked by Rule 4 anymore.

At $t = t_S = 28$, the EoP flit of the second packet from VCB 0 of Port 1 is transmitted, the last flit from $\mathcal{V}SV(\tau_1, l_2)$ to win arbitration before the SoP flit of the packet under analysis of τ_1.

In the following cycle $t = 29$, VCB 0 of Port 0 is not blocked by Rule 2 anymore and wins LRU arbitration with the SoP flit of the packet under analysis. From this point on, the packet under analysis is never blocked by Rule 2 anymore.

At $t = t_E = 38$, the EoP flit of the packet under analysis wins arbitration for the output, completing the third stage of transmission through the NPS.

To the 39 cycles depicted in the image, we must add 4 cycles for the two links traversed by the packet, for a total of 43. If we compare this result with the bound of 51 cycles found in Section 4.7.1, the bound is 18.6% larger than the WCL value in this trace.

Sources of pessimism in the analysis include it not being possible for all low-priority packets to actually send $V.r(l_2) + V.L(l_2)$ flits during the first stage of transmission.

4.8 Changes to NPS Arbitration

We consider two alternative versions of the NPS arbitration policy:

Alternative Version 1 operates with the following modified Rule 4, Rule 4-ALT, which ignores token counters:

- **Rule 4-ALT:** This rule produces sets of high-priority and low-priority requests from the set of valid VCB requests produced by Rules 1 to 3, according to the following procedure:
  - High-priority requests correspond to VCBs that have a high-priority flit at the head.
  - Low-priority requests, on the other hand, only require that VCBs do not have a SoP flit at the head.

Alternative Version 2 operates with Rule 4-ALT and with the following modified Rule 2, Rule 2-ALT:

- **Rule 2-ALT:** If another VCB V', that satisfies any of the following:
  - is associated with the same VC number as V
  - stores high-priority flits.

  has a packet in progress leaving the switch through l, V cannot participate in arbitration for l until after the EoP flit of said packet wins arbitration for l. This prevents the flits of different packets from interleaving in the VCB downstream from l associated with the same VC number.

In Sections 4.8.1-4.8.2, we discuss the changes to the WCL analysis implied by each alternative version of NPS arbitration.

4.8.1 Alternative Version 1

As Rule 4-ALT ignores the value of token counters, the first stage of transmission defined in Section 4.3.2 is eliminated (i.e. $t_R = 0$). This has the following the effects on the constraints:

- All the ILP problem variables $x(\tau_k)$, $y(\tau_k)$ and $z(\tau_k)$ defined in Section 4.3.3 are constrained to be equal to 0.
- Eq. 4.18 is updated to:

$$\begin{aligned}&\forall V \in \mathcal{V}^{DVH}(\tau_i, l), d^{DVH,V}_{local}(\tau_i, l) = \min(\\ &\tau_i.L + \sum_{V_a \in \mathcal{V}^{SV}(\tau_i,l)} \sum_{\tau_k \in V_a.\Gamma l} w(\tau_k),\\ &\sum_{\tau_k \in V.\Gamma(l)} packetCount(\tau_i, \tau_k) \times \tau_k.L\\ &)\end{aligned} \tag{4.44}$$

- Eq. 4.19 is updated to:

$$d^{DVL}_{local}(\tau_i, l) = 0 \tag{4.45}$$

4.8.2 Alternative Version 2

With Rule 2-ALT, VCBs in $\mathcal{V}^{SV}(\tau_i, l)$ and VCBs in $\mathcal{V}^{DVH}(\tau_i, l)$ interact with $VCB(\tau_i, l)$ in the same manner. Additionally, there is no need for the correction term $d_{bubbles}(\tau_i, l)$, as the non-SoP flits of a real-time packet cannot lose arbitration to another VCB, which are all blocked by Rule 2-ALT.

The number of cycles that the packet under analysis may be blocked by a packet from a VCB in $\mathcal{V}^{SV}(\tau_i, l) \cup \mathcal{V}^{DVH}(\tau_i, l)$ is then:

$$d_{R2A}(\tau_k, l) = \begin{cases} \tau_k.L, & \text{if } l = lastLink(\tau_k) \\ d(\tau_k, nextLink(\tau_k, l)) + l.b, & \text{otherwise} \end{cases} \tag{4.46}$$

which is equivalent to basic RC, and the local delay term $d_{local}(\tau_i, l)$ may be calculated simply as:

$$d_{local}(\tau_i, l) = \sum_{V \in \mathcal{V}^{SV}(\tau_i,l) \cup \mathcal{V}^{DVH}(\tau_i,l)} \max_{\tau_k \in V.\Gamma(l)} d_{R2A}(\tau_k, l) \tag{4.47}$$

Chapter 5

Evaluation

In this section, we evaluate the local analysis step and overall RC approach developed in Chapter 4 by simulating the 1 GHz "core" of the Versal NoC. Section 5.1 details how the NoC is simulated, and the general workflow. Section 5.2 describes an experiment with a single NPS, while Section 5.3 presents a case study with multiple CTs in a complete NoC.

Finally, Section 5.4 repeats the case study for the two alternative versions of the NPS arbitration policy described in Section 4.8.

5.1 Experimental Setup

Figure 5.1 shows a subset of the NoC embedded in the xcvc1902 Versal AI Core Device, including 16 NMU/NSU pairs connected to four VNoCs, which in turn are connected by the two HNoCs at their extremities. We choose the xcvc1902 because it is present in the VCK 190 board [46], and is the model used by the official Versal NoC tutorials [19]. The subset of the NoC with 16 clients is representative, in the sense that each individual NMU can only target 16 NSUs.

To simulate the NoC subset, we configure an NoC project with Vivado 2020.3 [20], and generate simulation files for the NoC following the steps in the official documentation [49]. The NoC configuration used in simulation (including topology, routing, token registers, etc.) may be modified by replacing the `xlnoc.v` topology file and NPS instantiation files with a design generated with Vivado.

We develop custom Verilog NMU/NSU modules to play the role of the NMU/NSU clients from Section 4, injecting packets via the NPP interface, and simulate the complete

system with Modelsim, through a script interface that allows for configuring the behaviour of the NMU/NSU clients via a CT specification in a `.csv` file, including the T, J^R, D, T and L parameters.

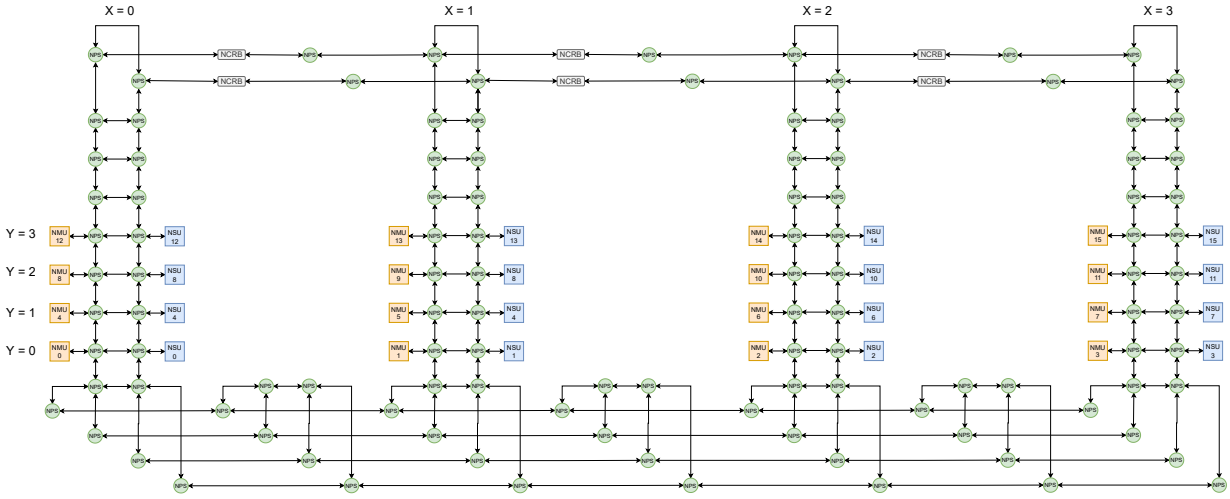


Figure 5.1: Subset of the NoC in the xcvc1902 Versal device that was simulated

Finally, we develop Python 3 scripts to post-process simulation logs to determine packet latencies, and also to implement the WCL approach described in Section 4. The WCL analysis script processes the `.ncr` file produced by the NoC compiler, as well as the Verilog simulation files for info on routing, VC assignments and token registers, which serve as inputs to the analysis.

Figure 5.2 shows the different components of the experiment, for instance for generating the plot in Figure 5.4 discussed later in this section.

The simulation takes the real-time CT parameters into account in the following manner:

- Arrival time for next packet sampled from $\tau_i.T + exp(1/T)$, where $exp(\lambda)$ is the exponential distribution with average value $\frac{1}{\lambda}$.
- Extra delay on the release of the packet uniformly selected from $[0, \tau_i.J^R]$.
- One packet injected at a time by the NMU.

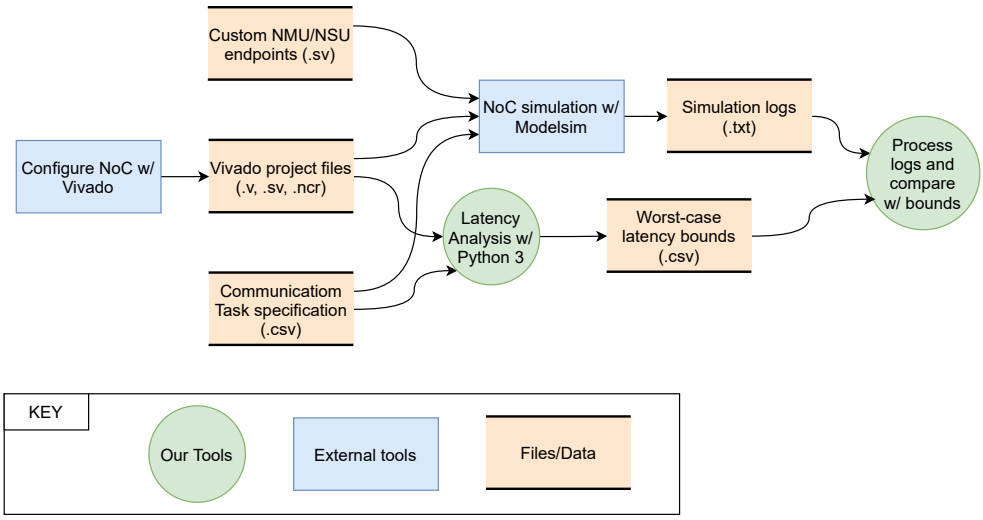


Figure 5.2: Data flow diagram of the evaluation experiment.

5.2 Single NPS

In order to capture the different types of interaction inside an NPS, we simulate the 8 following scenarios. In each scenario, we include the CT under analysis going through Port 3, VC 0 and one CT for each VCB in the set $\mathcal{V}^{scen}$, defined depending on the scenario as show in Table 5.1. We reserved VCs 0-3 for real-time CTs, and VCs 4-7 for best-effort CTs.

Table 5.1: Single NPS scenarios.

Scenario No.	$\mathcal{V}^{scen}$
0	$\varnothing$
1	$\mathcal{V}^{SV}$
2	$\mathcal{V}^{DVH}$
3	$\mathcal{V}^{SV} \cup \mathcal{V}^{DVH}$
4	$\mathcal{V}^{DVL}$
5	$\mathcal{V}^{SV} \cup \mathcal{V}^{DVL}$
6	$\mathcal{V}^{DVH} \cup \mathcal{V}^{DVL}$
7	$\mathcal{V}^{SV} \cup \mathcal{V}^{DVH} \cup \mathcal{V}^{DVL}$

For simplicity, we assume a value of 16 for each token register, and the following parameters for all CTs: $T = D = 200, J = 20, L = 8$. Each scenario was simulated for 10^7

cycles.

The plot in Figure 5.3a shows the distribution of latencies observed for the CT under analysis in each scenario, while Figure 5.3b shows a comparison with the WCL bound for each scenario. Figure 5.3b shows the range of latencies observed for packets of the CT under analysis, with a marker on the average value. The plot also shows the upper bound on packet latency for each scenario, which ranges from 1× to 7.11× the worst observed latency, depending on the scenario.

We note that the most significant gaps between the bounds and observed values correspond to configurations that include the VCBs from $\mathcal{V}^{DVL}$ (scenarios 3, 5 and 6). As the VCBs from $\mathcal{V}^{DVL}$ can only interfere with the CT under analysis before the time t_R, this indicates that, during the limited time-span of experiment, the unlikely event where other VCBs are all able to send up $V.r + V.L$ flits before t_R did not occur.

5.3 Multi-NPS Case Study

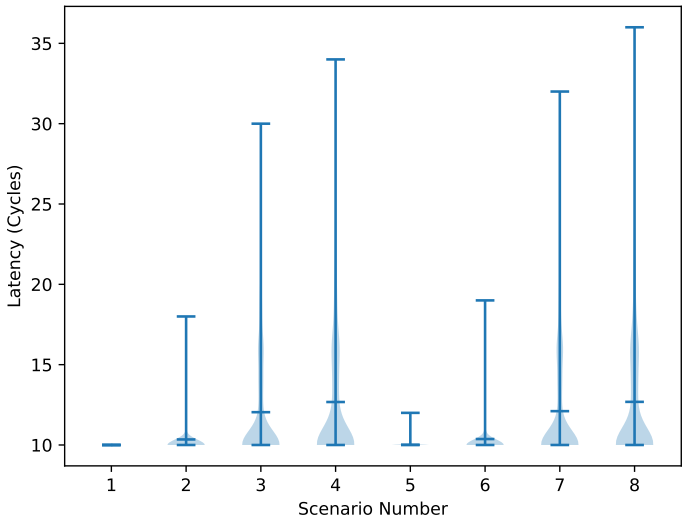
We adapt the 16-client workload used in [40], maintaining the connectivity matrix and relative bandwidths between CTs but increasing the bandwidth requested by each CT by 1000× to stress the simulated Versal NoC, which operates at 1GHz. The workload contains 37 CTs that exchange data between applications partitioned among the 16 clients, corresponding to sensing and computation tasks in an autonomous robot application. Table 5.2 shows the parameters of all CTs, with the VC assignments being determined by the Versal NoC compiler and $L = 8$ for each CT.

We calculate bounds for and simulate the system for 10^7 cycles. The violin plot in Figure 5.4a shows the distribution of latencies for packets of each CT that is simulated.

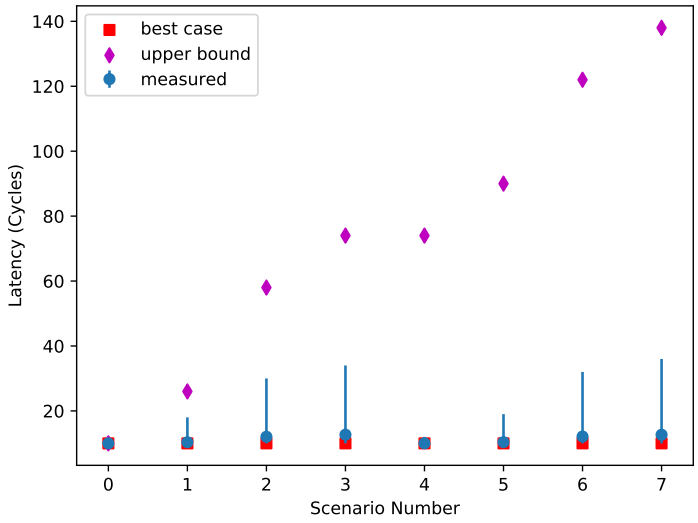
We compare simulation results with WCL bounds in the log-scale plot in Figure 5.4b. CTs from the same source are numbered sequentially, so we may observe multiple CTs sharing the same WCL bound due to how the WCL function $R(\tau)$ is calculated.

For each CT in the workload, the plot shows the range of packet latencies observed in simulation, with a marker on the average value, as well as the best-case (structural) latency and the WCL upper bound. While the markers at the extremities show the structural latency and the calculated WCL upper bound.

Excluding the trivial case of CT number 37, which does not contend with any others, the upper bound is 6× to 12× the maximum observed latency. This likely emerges from



(a) Distribution of latencies

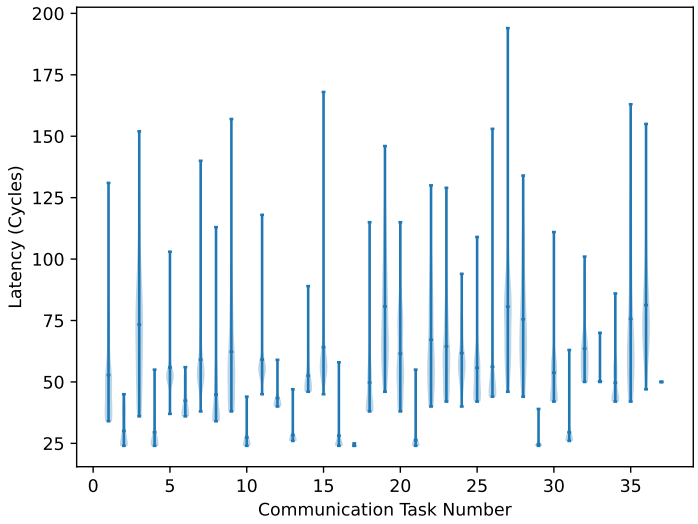


(b) Comparison with Bounds

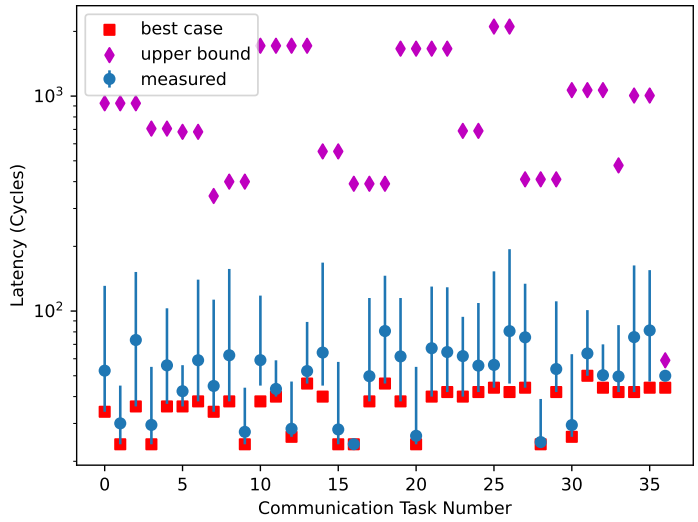
Figure 5.3: Results for experiment with a single NPS.

Table 5.2: Flow parameters for the case study in Figure 5.4, adapted from [40]

CT	**NMU**	**NSU**	**T = D**	**J**	**CT**	**NMU**	**NSU**	**T = D**	**J**
1	0	1	1000	100	21	8	4	500	50
2	0	4	2000	200	22	8	5	500	50
3	0	5	2000	200	23	8	9	500	50
4	1	5	500	50	24	9	6	500	50
5	1	6	1000	100	25	9	10	500	50
6	2	7	1000	100	26	10	4	500	50
7	2	9	500	50	27	10	9	500	50
8	3	2	500	50	28	11	5	1000	100
9	4	5	500	50	29	11	7	500	50
10	4	8	1000	100	30	11	10	500	50
11	5	6	500	50	31	12	4	500	50
12	5	8	1000	100	32	12	7	500	50
13	5	13	500	50	33	12	13	500	50
14	5	15	1000	100	34	13	6	1000	100
15	6	9	500	50	35	14	5	500	50
16	6	10	500	50	36	14	9	500	50
17	7	3	2000	200	37	15	14	500	50
18	7	6	500	50					
19	7	13	500	50					
20	8	1	500	50					



(a) Distribution of latencies



(b) Comparison with Bounds

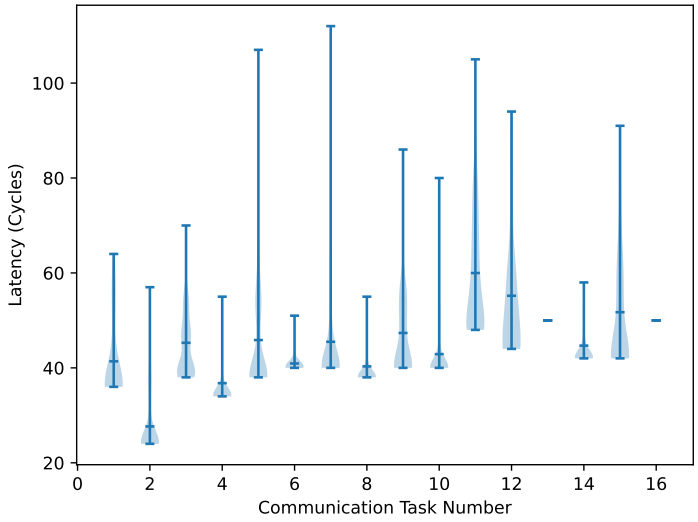
Figure 5.4: Distribution of latencies and comparison with WCL bounds for the case study variation in Table 5.2.

a combination of a degree of pessimism in the analysis and the worst-case scenario being difficult to capture in simulation, especially the first stage of transmission defined in Section 4.3.2 where multiple flits are sent without a token reload happening.

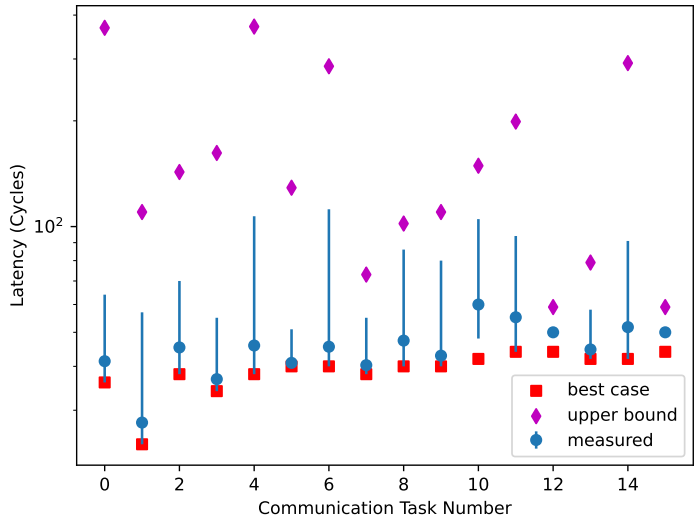
Table 5.3: Flow parameters for the case study in Figure 5.5, adapted from [40]

CT	**NMU**	**NSU**	**T = D**	**J**	**CT**	**NMU**	**NSU**	**T = D**	**J**
1	0	1	1000	100	9	8	1	500	50
2	1	5	500	50	10	9	6	500	50
3	2	7	1000	100	11	10	4	500	50
4	3	2	500	50	12	11	5	1000	100
5	4	5	500	50	13	12	4	500	50
6	5	6	500	50	14	13	6	1000	100
7	6	9	500	50	15	14	5	500	50
8	7	3	2000	200	16	15	14	1000	100

We also consider a subset of the case study with 16 CTs shown in Table 5.3, such that each NMU client injects a single CT in the NoC. The plot in Figure 5.5a shows the distribution of latencies for this version of the case study, while the log-scale plot in Figure 5.5b shows the overall results, with bounds $2\times$ to $5\times$ the maximum observed latency, suggesting that the approach to evaluating the WCL function $R(\tau_i)$ by summing the delay function for all CTs with the same origin as τ_i is a source of pessimism.



(a) Distribution of latencies



(b) Comparison with Bounds

Figure 5.5: Distribution of latencies and comparison with WCL bounds for the case study variation in Table 5.3.

5.4 Changes to NPS Arbitration

We modify the arbitration logic followed by the simulated NPS modules (function `arb()` in the file `nps_ipa.sv`) to implement the two alternative versions described in Section 4.8.

5.4.1 Single NPS Module

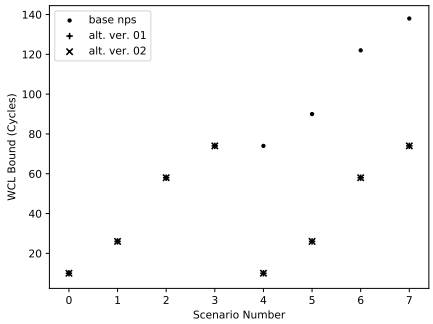
The plots in Figure 5.6 show a comparison between the base NPS policy and the two variants for the single switch experiment from Section 5.2. The most significant effect of the alternative versions in Figure 5.6 is that the WCL bounds for scenarios 4-7 (which include VCBs in $\mathcal{V}^{DVL}$) can ignore the presence of best-effort CTs, which may not interfere with the packet under analysis under Rule 4-ALT, leading to tighter estimates. The WCL bounds for scenarios 0-3 remain the same under the alternative versions, as the number of interfering packets from VCBs in $\mathcal{V}^{SV}$ and $\mathcal{V}^{DVH}$ remain bounded to one for each VCB due to timing parameters.

In the scenarios including VCBs in $\mathcal{V}^{DVH}$ (scenarios 2, 3, 6 and 7), we note that alternative version 2 produces higher maximum latencies, but lower average latencies than the base policy and alternative version 1.

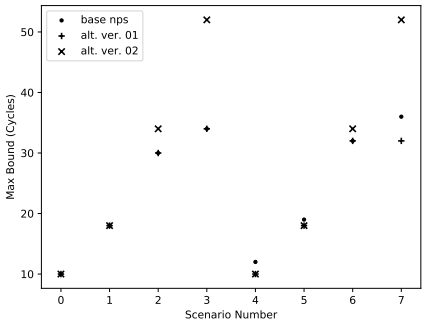
5.4.2 Multiple NPS Modules

The plots in Figure 5.7 show a comparison between the base NPS policy and the two variants for the multi-switch experiment from Section 5.3.

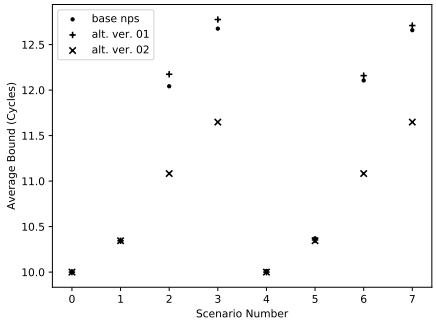
We note that alternative version 1 does not change maximum and average latencies significantly, while slightly reducing the WCL estimates. Alternative version 2, on the other hand, produces small increases or decreases to the maximum latencies observed by each CT, but causes the WCL analysis bounds to increase for all cases, due to the larger number of complete packets blocking the packet under analysis in each switch, resulting in a worse performance for the analysis. However, we again observe that alternative version 2 decreases the average latency experienced by most of the CTs.



(a) WCL Bounds

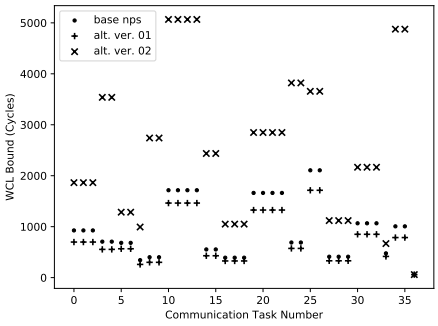


(b) Maximum Latencies

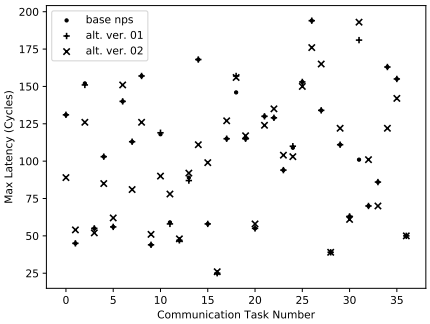


(c) Average Latencies

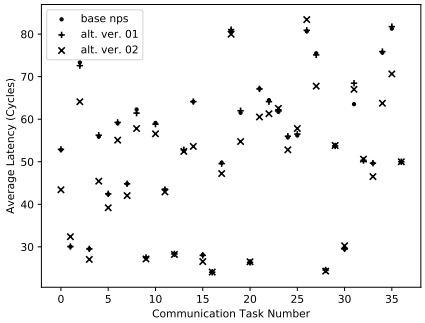
Figure 5.6: Results for the alternative versions of NPS arbitration, for the experiment with a single NPS module.



(a) WCL Bounds



(b) Maximum Latencies



(c) Average Latencies

Figure 5.7: Results for the alternative versions of NPS arbitration, for the experiment with multiple NPS modules.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this work, we present and evaluate an WCL analysis approach for the Versal NoC adapted from RC, by following the sequence of steps:

- Study of the architecture and operation of the Versal NoC, focusing on the arbitration policy implemented by the NPS modules.
- Review of the RC approach for WCL analysis.
- Adaptation of the RC approach to the Versal NoC architecture, by changing the optimization problem solved to determine the local delay suffered by packets in each NPS module.
- Evaluation with a simulated NoC from the xcvc1902 Versal device.

In the evaluation step, we observe that the upper bounds obtained through the WCL analysis dominate the worst-case latencies resulting from simulating the NoC, as the WCL analysis method was designed to provide.

However, the WCL bounds overestimate the worst-case observed values by a factor of 6× to 12× in the first version of the case study in Section 5.3. We identify mechanisms implemented by the NPS arbitration policy, and consequently accounted for in the WCL analysis, that may contribute to the degree of pessimism observed:

- In the worst case, the first stage of transmission through an NPS (defined in Section 4.3.2) corresponds to multiple CTs sending flits in a particular configuration in order to delay the token counter reset as much as possible, which is unlikely to happen in practice.
- The interaction between different VCs may gives rise to bubbles (as defined in Section 4.3.2) that contribute to further delay the packet under analysis, but are also unlikely to emerge in practice (as they are maximized when the SoP flit of the interfering packet is never delayed, while the following non-SoP flits are delayed as much as possible).

6.2 Future Work

We identify the following possible next steps for WCL analysis in the Versal NoC:

- Add the complete behaviour of actual NMU/NSU endpoints to the analysis, pending additional investigation on their behaviour or release of additional information by the designers of the Versal NoC.
- Selection of NoC parameters (e.g. VC assignments, token registers, routing tables) in order to guarantee real-time deadline are met.
- Improvement to the WCL analysis tool developed for Chapter 5, for instance in terms of how the results are displayed to the designer.
- In a low-latency inference task, such as the one discussed in Section 1.1, the timing behaviour of the Versal AI engines [51] will also factor into WCET calculations. The AI engines, structured as a 2-dimensional arrays of scalar cores with vector processing units, are another architecturally innovative block of the Versal platform, to which existing WCET techniques could be adapted. A method for this would be an important complement to the WCL analysis for the NoC component.

References

[1] Mohamed S. Abdelfattah and Vaughn Betz. The case for embedded networks on chip on field-programmable gate arrays. *IEEE Micro*, 34(1):80–89, 2014.

[2] Mohamed S. Abdelfattah, Andrew Bitar, and Vaughn Betz. Design and applications for embedded networks-on-chip on fpgas. *IEEE Transactions on Computers*, 66(6):1008–1021, 2017.

[3] Achronix Semiconductor. *Eight Benefits of Using an FPGA with an On-chip High-Speed Network (WP020)*, 2019.

[4] Achronix Semiconductor. *Speedster7t Network on Chip User Guide (UG089)*, 2019.

[5] Achronix Semiconductor. *FPGAs Enable the Next Generation of Communication and Networking Solutions (WP021)*, 2020.

[6] Fernando Alcalde Cuesta, Pablo González Sequeiros, and Álvaro Lozano Rojo. A method for validating rent's rule for technological and biological networks. *Scientific Reports*, 7(1):5378, Jul 2017.

[7] Shane L. Bell, Bruce Edwards, John Amann, Rich Conlin, Kevin Joyce, Vince Leung, John MacKay, Mike Reif, Liewei Bao, John F. Brown, Matthew Mattina, Chyi-Chang Miao, Carl Ramey, David Wentzlaff, Walker Anderson, Ethan Berger, Nat Fairbanks, Durlov Khan, Froilan Montenegro, Jay Stickney, and Jon C. Zook. Tile64 - processor: A 64-core soc with mesh interconnect. *2008 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, pages 88–598, 2008.

[8] A. Burns, L. S. Indrusiak, N. Smirnov, and J. Harrison. A novel flow control mechanism to avoid multi-point progressive blocking in hard real-time priority-preemptive nocs. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 137–147, 2020.

[9] Kumar H B Chethan and Nachiket Kapre. Hoplite-dsp: Harnessing the xilinx dsp48 multiplexers to efficiently support nocs on fpgas. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–10, 2016.

[10] W.J. Dally. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, 1992.

[11] Dakshina Dasari, Borislav Nikoli'c, Vincent N'elis, and Stefan M. Petters. Noc contention analysis using a branch-and-prune algorithm. *ACM Trans. Embed. Comput. Syst.*, 13(3s), March 2014.

[12] Benoit Dupont de Dinechin. Kalray mppa®: Massively parallel processor array: Revisiting dsp acceleration with the kalray mppa manycore processor. In *2015 IEEE Hot Chips 27 Symposium (HCS)*, pages 1–27, 2015.

[13] Thomas Ferrandiz, F. Frances, and C. Fraboul. A network calculus model for spacewire networks. *2011 IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications*, 1:295–299, 2011.

[14] Thomas Ferrandiz, Fabrice Frances, and Christian Fraboul. A method of computation for worst-case delay analysis on spacewire networks. In *2009 IEEE International Symposium on Industrial Embedded Systems*, pages 19–27, 2009.

[15] Rosemary Francis and Simon Moore. Exploring hard and soft networks-on-chip for fpgas. In *2008 International Conference on Field-Programmable Technology*, pages 261–264, 2008.

[16] Jan Gray. Grvi phalanx: A massively parallel risc-v fpga accelerator accelerator, 2016.

[17] Rafik Henia, Arne Hamann, Marek Jersak, R. Racu, Kai Richter, and Rolf Ernst. System level performance analysis - the symta/s approach. *Computers and Digital Techniques, IEE Proceedings* -, 152:148 – 166, 04 2005.

[18] Yutian Huan and André DeHon. Fpga optimized packet-switched noc using split and merge primitives. In *2012 International Conference on Field-Programmable Technology*, pages 47–52, 2012.

[19] Xilinx Inc. 2021.1 vivado™ - versal architecture-specific tutorials.

[20] Xilinx Inc. Vivado 2020.3, 2021.

[21] Nachiket Kapre. Implementing fpga overlay nocs using the xilinx ultrascale memory cascades. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 40–47, 2017.

[22] Nachiket Kapre and Jan Gray. Hoplite: Building austere overlay nocs for fpgas. In *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8, 2015.

[23] George Karypis and Vipin Kumar. Kumar, v.: A fast and high quality multilevel scheme for partitioning irregular graphs. siam journal on scientific computing 20(1), 359-392. *Siam Journal on Scientific Computing*, 20, 01 1999.

[24] Hany Kashif, Sina Gholamian, and Hiren Patel. Sla: A stage-level latency analysisfor real-time communicationin a pipelined resource model. *IEEE Transactions on Computers*, 64:1177–1190, 2015.

[25] Ian Lang, Ziqiang Huang, and Nachiket Kapre. Exploring the impact of switch arity on butterfly fat tree fpga nocs. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 70–74, 2020.

[26] Ian Lang, Nachiket Kapre, and Rodolfo Pellizzoni. Worst-case latency analysis for the versal noc network packet switch. In *Proceedings of the 15th IEEE/ACM International Symposium on Networks-on-Chip*, NOCS '21, page 55–60, New York, NY, USA, 2021. Association for Computing Machinery.

[27] M. Langhammer, G. Baeckler, and S. Gribok. Spiderweb - high performance fpga noc. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 115–118, USA, 2020. IEEE.

[28] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, Berlin, Heidelberg, 2001.

[29] Meng Liu, Matthias Becker, Moris Behnam, and Thomas Nolte. Buffer-aware analysis for worst-case traversal time of real-time traffic over rra-based nocs. In *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 567–575, 2017.

[30] Meng Liu, Matthias Becker, Moris Behnam, and Thomas Nolte. A tighter recursive calculus to compute the worst case traversal time of real-time traffic over nocs. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 275–282, 2017.

[31] Pongstorn Maidee, Alireza Kaviani, and Kevin Zeng. Linkblaze: Efficient global data movement for fpgas. In *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–8, 2017.

[32] Gurshaant Singh Malik and Nachiket Kapre. Enhancing butterfly fat tree nocs for fpgas with lightweight flow control. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 154–162, 2019.

[33] L.M. Ni and P.K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, 1993.

[34] Borislav Nikolic, Sebastian Tobuschat, L. Indrusiak, R. Ernst, and A. Burns. Real-time analysis of priority-preemptive nocs with arbitrary buffer sizes and router delays. *Real-Time Systems*, 55:63–105, 2018.

[35] Michael K. Papamichael and James C. Hoe. Connect: Re-examining conventional wisdom for designing nocs in the context of fpgas. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '12, page 37–46, New York, NY, USA, 2012. Association for Computing Machinery.

[36] Eberle A. Rambo and Rolf Ernst. Worst-case communication time analysis of networks-on-chip with shared virtual channels. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 537–542, 2015.

[37] Eberle A. Rambo and Rolf Ernst. Worst-case communication time analysis of networks-on-chip with shared virtual channels. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 537–542, 2015.

[38] Martin Schoeberl, Sahar Abbaspour, Benny Akesson, Neil Audsley, Raffaele Capasso, Jamie Garside, Kees Goossens, Sven Goossens, Scott Hansen, Reinhold Heckmann, Stefan Hepp, Benedikt Huber, Alexander Jordan, Evangelia Kasapaki, Jens Knoop, Yonghui Li, Daniel Prokesch, Wolfgang Puffitsch, Peter Puschner, and Alessandro Tocchi. T-crest: Time-predictable multi-core architecture for embedded systems. *Journal of Systems Architecture*, 61, 04 2015.

[39] Zheng Shi and Alan Burns. Real-time communication analysis for on-chip networks with wormhole switching. In *Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '08, page 161–170, USA, 2008. IEEE Computer Society.

[40] Zheng Shi, Alan Burns, and Leandro Indrusiak. Schedulability analysis for real time on-chip communication with wormhole switching. *IJERTCS*, 1:1–22, 04 2010.

[41] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round-robin. *IEEE/ACM Transactions on Networking*, 4(3):375–385, 1996.

[42] Radu Stefan, Anca Molnos, Angelo Ambrose, and Kees Goossens. A tdm noc supporting qos, multicast, and fast connection set-up. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1283–1288, 2012.

[43] Ian Swarbrick, Dinesh Gaitonde, Sagheer Ahmad, Brian Gaide, and Ygal Arbel. Network-on-chip programmable platform in versal™ acap architecture. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '19, page 212–221, New York, NY, USA, 2019. Association for Computing Machinery.

[44] Ian Swarbrick, Dinesh Gaitonde, Sagheer Ahmad, Bala Jayadev, Jeff Cuppett, Abbas Morshed, Brian Gaide, and Ygal Arbel. Versal network-on-chip (noc). In *2019 IEEE Symposium on High-Performance Interconnects (HOTI)*, pages 13–17, 2019.

[45] Kees Vissers. Versal: The xilinx adaptive compute acceleration platform (acap). In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '19, page 83, New York, NY, USA, 2019. Association for Computing Machinery.

[46] Xilinx Inc. *VCK190 Evaluation Board (UG 1366)*.

[47] Xilinx Inc. *Xilinx UltraScale: The Next-Generation Architecture for Your Next-Generation Architecture (WP 435)*, 2014.

[48] Xilinx Inc. *SmartConnect v1.0 LogiCORE IP Product Guide (PG 247)*, 2020.

[49] Xilinx Inc. *Versal ACAP Programmable Network on Chip and Integrated Memory Controller (PG 313)*, 2020.

[50] Xilinx Inc. *Versal: The First Adaptive Compute Acceleration Platform (WP 505)*, 2020.

[51] Xilinx Inc. *Xilinx AI Engines and Their Applications (WP 506)*, 2020.

[52] Xilinx Inc. *Versal ACAP Technical Reference Manual (AM 011)*, 2021.

[53] Qin Xiong, Zhonghai Lu, Fei Wu, and Changsheng Xie. Real-time analysis for wormhole noc: Revisited and revised. In *2016 International Great Lakes Symposium on VLSI (GLSVLSI)*, pages 75–80, 2016.

Glossary

backpressure : Situation where flits are prevented from progressing due to insufficient buffer space downstream, as determined by the flow-control mechanism. In the Versal NoC, correspond to blocking due to Rule 3. 8

flit : Unit of data that may be transferred over an NoC link over a clock cycle (short for flow control digit) 9

flow control : Mechanism for handling buffer occupancy in real-time in the NoC. 5, 7

token counter : Counter maintained by an NPS output link for a VC buffer, decremented for each flit the buffer sends through the output link. Used to implement differential QoS between CTs. 18

token register : Reset value for a corresponding token register. 18

wormhole routing : Strategy for NoC routing where contents of a packet travel as a sequence of flits in a pipelined manner. 5