Worst-Case Latency Analysis for the Versal NoC Network Packet Switch

Ian Lang
Electrical and Computer Engineering
University of Waterloo
Waterloo, Canada
ielmorla@uwaterloo.ca

Nachiket Kapre
Electrical and Computer Engineering
University of Waterloo
Waterloo, Canada
nachiket@uwaterloo.ca

Rodolfo Pellizzoni
Electrical and Computer Engineering
University of Waterloo
Waterloo, Canada
rpellizz@uwaterloo.ca

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 present a formal description of the NPS switches that compose the Versal NoC from a flit (or packet) scheduling perspective, based on the available cycle-accurate switch simulation code. We then analyze a scenario where network clients transfer data periodically over a single switch, and propose a method for calculating worst-case communication times in this scenario.

KEYWORDS

Network-on-chip, FPGA, Real-time

ACM Reference Format:
Ian Lang, Nachiket Kapre, and Rodolfo Pellizzoni. 2021. Worst-Case Latency Analysis for the Versal NoC Network Packet Switch. In *International Symposium on Networks-on-Chip (NOCS '21), October 14–15, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3479876.3481593

1 INTRODUCTION

Announced by FPGA vendor Xilinx Inc. in 2019, the Versal ACAP line of devices includes a hard Network-on-Chip (NoC) [5]. The NoC is intended as a backbone for system-level communication, connecting the heterogeneous elements in the device (e.g. the programmable logic (PL), DDR Memory Controllers (DDRMC), ARM cores, I/O interfaces, AI cores [14, 15]).

The target markets for Versal devices include applications with hard real-time constraints, such as automotive driver assist, airspace and defense [13]. For these applications, designers should be able to determine upper bounds on the Worst-Case Execution Times (WCET) of various tasks performed by the system, in order to ensure that mission-critical requirements are respected. Additionally, such upper bounds should be as "tight"as possible, to avoid over-provisioning of resources.

As the time spent exchanging data over the Versal NoC can be part of a task's execution time, it would be useful to have access to Worst-Case Communication Time (WCCT) upper-bounds for such exchanges. However, the development tools associated with the Versal platform only provide figures for the "structural latency" of single data flits, which correspond to an otherwise idle network where the flit in question is the only one travelling. In general, this information is insufficient, as the interference between different flows of data can force packet contents to wait in buffers inside NoC switches, increasing communication latency. In this paper, we review the architecture of the Versal NoC, and propose a WCCT method for data transferred periodically over a single switch, as a first step towards a full WCCT method for packets exchanged over the Versal NoC.

The Versal platform can be seen as a relevant case study for real-time communication analysis for NoC, as it combines multiple features that are not usually considered together in timing analysis works: wormhole routing, multiple Virtual Channels (VCs - each shared by multiple flows of data), Least-Recently Used (LRU) arbitration, and additional Quality-of-Service (QoS) mechanisms. This works's contributions include:

- A formal description of the arbitration policy implemented by the NoC.
- A method for calculating WCCTs for the case of network clients exchanging data over a single NoC switch, and a corresponding framework for simulating this configuration.

2 BACKGROUND

2.1 NoC for FPGA

Overlay/"soft" NoC, built with resources from the reconfigurable fabric are a traditional approach for implementing the NoC paradigm in FPGAs [6, 11]. As these devices have grown to integrate more specialized components embedded in the PL (e.g. DSP modules, block RAM and even processor cores), embedded "hard" NoCs have become a concrete possibility, with the initial motivations being gains in area and power consumption [1].

Alongside the Versal line from Xilinx Inc., the Speedster7t line from Achronix Semiconductor is among the first commercial FPGA products to include an embedded NoC [3]. Compared to the Versal

NOCS '21, October 14–15, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-9083-5/21/10…$15.00
https://doi.org/10.1145/3479876.3481593

NoC, the Speedster7t NoC has a simpler architecture (e.g. it lacks multiple VCs), but runs at a higher frequency (2 versus 1 GHz) and has wider data links (256 versus 128 bits). Besides the gains in performance, other possible advantages of embedded NoCs in commercial FPGAs include simplifying timing closure and helping with support for partial reconfiguration and hardware virtualization of portions of the device [2].

Finally, Intel Spiderweb [8], proposed by another major FPGA vendor, is described as "firm", pre-floorplanned overlay NoC that attempts to combine the performance gains of hard embedded NoC with the flexibility of soft implementations.

2.2 Real-Time Analysis for NoC

The approach for computing WCCT bounds for an NoC depends on the model of operation followed by the network. A Time-Division Multiplexed NoC, where communications obey a pre-calculated schedule, is an example of a network that make it simple to guarantee worst-case bounds for data exchanges.

Among networks without a fixed transmission schedule, the prioritized VCs model has received attention as an alternative for guaranteeing real-time deadlines. This model assumes that flows of data travel in VCs associated with priority levels, and the transmission of a lower-priority VC through an output port can be preempted by a higher-priority VC. Such networks can be analyzed with techniques adapted from scheduling of prioritized tasks in processors, as done in [12]. The identification of the phenomenon of multi-point progressive blocking [18] has necessitated modifications to this approach when backpressure can emerge in the NoC, increasing the pessimism of results [10].

Many practical NoC architectures do not support multiple prioritized VCs, and instead perform Round-Robin Arbitration (RRA) between input ports. Techniques for analyzing this type of network include Recursive Calculus [4] and Compositional Performance Analysis [16].

As discussed in Section 3, the Versal NoC architecture includes both multiple VCs and round-robin arbitration, as well as an additional mechanism of *token counters* that affects the contention for output ports, necessitating a custom approach.

3 SYSTEM MODEL

In this section we describe the operation of the Versal NoC, focusing on the arbitration mechanism implemented by NPS modules. We base our description on existing documentation for the Versal NoC [13, 14, 17], complemented with study of the NPS bus functional model simulation code available with Vivado 2020.3 [7].

3.1 Versal NoC Architecture

Figure 1a shows a high-level block diagram of a Versal NoC in the context of the ACAP device. NoC clients, such as the programmable logic (PL), access the network via Network Master Units (NMU) and Network Slave Units (NSU). To start a data exchange over the NoC, a client can access the AXI interface of an NMU, which will convert the transaction to the NoC's internal Network Packet Protocol (NPP). The network is organized as two Horizontal NoCs (HNoCs), connected by multiple Vertical NoCs (VNoCs). The VNoCs are embedded in the PL, while the HNoCs provide network access to the hard components in the system.

The supported traffic classes are ISOC, LL and BE, with the ISOC class being designed for communication tasks with real-time deadlines. At run-time, ISOC and LL packets are treated in the same way by the network, having a bit set in each of their flits indicating high-priority, while BE flits are low-priority. Thus, we simply differentiate between high and low-priority packets for the rest of the text. The user guide [17] mentions a mechanism for increasing the priority of ISOC packets after a timeout expires, but this feature is not covered by the current version of the tools and simulation code.

The NoC uses *wormhole routing* [9]: each packet can contain up to 16 data flits, each with a 128-bit payload, as well as a request flit at the head that contains information on the associated AXI transaction. We identify the first flit of a packet as the Start-of-Packet (SOP) flit, and the last flit as the End-of-Packet flit (EOP), with both being the same for a single-flit packet. Packets have a size limit of 256 bytes per packet, with larger AXI transactions are broken into multiple packets. The flits traverse the NoC in a pipelined manner over multiple NPS units until they reach their destination NMU or NSU.

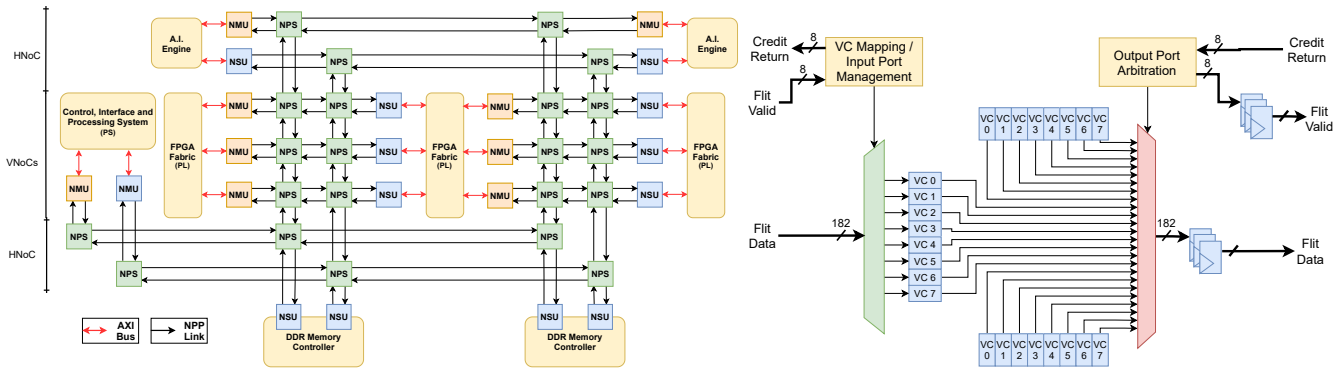
Figure 1b shows the internal structure of an NPS, with four bidirectional ports, numbered 0 to 3, which can send and receive one flit per cycle. While each flit only holds 128 bits of payload data, each NPP data connection is actually 182 bits wide, with the additional 54 bits holding metadata for the communication.

The minimum latency through the NPS is 2 cycles (one to copy the flit into a buffer, another to copy the flit to the appropriate output after arbitration). The NoC supports 8 VCs: each NPS input port has one VC Buffer (VCB) for each VC, where incoming flits associated with said VC are stored. For a given NPS, we use the notation $V_{ip,vc}$) for the VCB of input port ip associated with VC number vc. VCBs have a depth of 5 in VNoCs, and of 7 in HNoCs.

3.2 Arbitration Policy

Each output port can be targeted by 24 VCBs (the 8 VCBs of each of the other 3 ports). We use $op.\mathcal{V}$ to denote the set of VCBs that can target output port op. Every cycle, each output port op performs Least Recently Used (LRU) arbitration among eligible VCBs $V \in op.\mathcal{V}$ to determine which flit, if any, to send downstream. The user guide describes five rules (here **Rule 1** to **Rule 5**) that determine the VCBs that can contend for the output each cycle. We restate said rules with additional details identified from the NPS simulation code. **Rule 1** to **Rule 3** are used to simply disqualify VCBs:

- **Rule 1:** A VCB cannot participate in arbitration for an output if it is empty, or if the flit at its head should be routed to a different output.
- **Rule 2:** If a VCB with VC number vc has previously won the arbitration for an output port with a SOP flit, and has not yet sent the corresponding EOP flit through, other VCBs associated with VC number vc cannot participate in arbitration for op. This prevents flits from two different packets from ending up interleaved in VCBs.
- **Rule 3:** Each output port has 8 credit counters to keep track of the available places in each of the VCBs downstream from



(a) Versal NoC (b) NPS

Figure 1: Block diagrams of a Versal NoC and of an NPS (with details of an input port and an output port).

it. A VCB cannot participate in arbitration for an output port if the corresponding downstream VCB is full. There is one cycle of delay from a space for a flit being freed in the downstream VCB to the credit being received.

VCBs that pass Rules 1 to 3 are submitted to Rule 4, which can disqualify VCBs and also sorts them into high and low-priority requests.

Rule 4: As an additional QoS mechanism, each output port maintains a *token counter* for each of the 24 VCs that can target it (for a total of $4 \times 24 = 96$ token counters in each NPS). Each token counter is decremented whenever the corresponding buffer wins arbitration, and is associated with a token register, which contains the counter's maximum value. For a given output port, we refer to the token counter of a VCB V as $V.c$, and to the corresponding register as $V.r$.
A VCB is considered a high-priority request if it has a high-priority flit at the head and either has a token counter > 0 or the flit is not a SOP. If the token counter is ≥ 0 or the flit at the head is not a SOP, but the VCB does not qualify as a high-priority request, it counts as a low-priority one. Finally, a VCB that has a SOP flit at the head and a counter < 0 cannot be included in either group and is excluded from arbitration this cycle.

Finally, Rule 5 determines how the high and low-priority requests are considered by the arbitration:

Rule 5 If there are any high-priority requests, the LRU arbitration is performed only with said high-priority requests.

Algorithm 1 shows the sequence in which the rules are applied every cycle by each output port, as well as the condition for reloading the token counters (Line 2). When a token counter $V.c$ is reloaded, it receives $V.r$ if $V.c \geq 0$, and $V.r - 1$ otherwise.

Algorithm 1: Arbitration policy performed every clock cycle by each NPS output port

1 Find VCBs that pass Rules 1-3;
2 If at least one buffer is selected by **Line 1**, and no buffer selected by **Line 1** has a token counter > 0, record that all token counters for this port have to be reloaded;
3 Determine low and high-priority requests according to Rule 4, among buffers selected by **Line 1**;
4 Perform LRU arbitration among low or high-priority requests according to Rule 5;
5 If a VCB won the LRU arbitration, decrement its token counter and move the flit to the output port;
6 Reload all token counters if **Line 2** determined they should be reloaded;

4 ANALYSIS

As discussed in section 3, the Versal NoC switches contain multiple mechanisms that can interact to delay packets contending for an output port. We focus on modeling a scenario that makes no simplifying assumptions on the internal NPS structure, but assumes that packets travel over a single switch.

4.1 Problem Statement

Consider a situation with an NPS connected to three NMU modules and one NSU module, which in turn communicate with network clients through AXI buses. The NMU modules periodically send packets of write data to the NSU, over different VCs. We model this situation as a set of Communication Tasks (CTs) $\Gamma = \{\tau_0, \tau_1, ...\}$, each associated with an input port and VC. We can focus on AXI writes without loss of generality, as the NPS treats read and write data in the same manner.

Each CT $\tau_i \in \Gamma$ has the following parameters, which we access through dot notation (e.g. $\tau_i.T$): $T, J, D, L, ip, op, vc, V$. Their meaning is as follows:

- $\tau_i.T$: minimum interarrival time (in cycles) between the generation of subsequent data packets of CT τ.
- $\tau_i.J$: jitter (in cycles) in the release of each packet. Formally, this means that if a packet is generated at time t, it is sent to the NPS at time $t' \in [t, t + \tau_i.J]$.

- $\tau_i.D$: deadline (in cycles) for the reception of each packet, relative to the packet's generation. We assume $\tau_i.D \leq \tau_i.T$.
- $\tau_i.L$: length of each packet in flits.
- $\tau_i.BP$: the maximum number of cycles of backpressure (blocking due to Rule 3) that each packet of τ can suffer.
- $\tau_i.ip, \tau_i.op, \tau.vc, \tau_i.V$: input port, output port, VC and VCB the data travels through. To avoid a low-priority packet blocking a high-priority packet via Rule 2, we assume that vc numbers are reserved for either high-priority or low-priority CTs. We have observed the Versal NoC tools to follow this assignment policy.

We wish to find a bound for $\tau_i.R$, the worst-case latency for a packet of a high-priority CT under analysis τ_i to cross the switch, after reaching the head of $\tau_i.V$. For simplicity, we do not consider the head-of-line blocking incurred while a packet of τ_i waits to arrive at the front of the VCB queue. In the absence of backpressure or other CTs to contend for the output port to the NSU, the worst-case latency of the packet will simply be $\tau_i.L$, which is the time to copy all flits to the output. Otherwise, we define the blocking term $\tau_i.B$ to be the number of cycles, after reaching the head of $\tau_i.V$, that the packet fails to send a flit (due to losing the LRU arbitration, or being blocked by rules 2, 3, 4 or 5). The actual latency to cross the switch after reaching the head of the buffer will then be:

$$\tau_i.R = \tau_i.L + \tau_i.B, \tag{1}$$

and we say that the packet is schedulable (that is, meets the deadline) if:

$$\tau_i.J + \tau_i.R + 1 \leq \tau_i.D. \tag{2}$$

Note that the +1 term in the equation is needed to count the extra cycle required to copy each flit into the buffer.

4.2 Contending Buffers

Let $\tau_i.op.\mathcal{V}$ be the set of 24 VCBs that can contend for the output port crossed by CT under analysis τ_i. We can partition $\tau_i.op.\mathcal{V} \setminus \tau_i.V$ into 3 subsets that interact in different manners with the buffer $\tau_i.V$:

- The set $\mathcal{V}^{SV}$ containing the two other buffers associated with $\tau_i.vc$.
- The set $\mathcal{V}^{DVH}$ of high-priority VCBs associated with a different VC from $\tau_i.vc$.
- The set $\mathcal{V}^{DVL}$ of low-priority VCBs associated with a different VC from $\tau_i.vc$.

Figure 2 shows an example of this partition, assuming that $\tau_i.ip = 3, \tau_i.vc = 0$ and that VCs $0-3$ are transporting high-priority CTs, while VCs $4-7$ are assigned low-priority CTs.

Each VCB $V_j \in \tau_i.op.\mathcal{V} \setminus \tau_i.V$ has a maximum number $V_j.n$ of flits it can use to win arbitration for the output port while the packet crosses the NPS, given by:

$$V_j.n = \sum_{\tau_k \in V_j.\Gamma} \left\lceil \frac{\tau_i.R + \tau_k.J}{\tau_k.T} \right\rceil \times \tau_k.L, \tag{3}$$

where $V.\Gamma$ is the set of CTs that cross the VCB V.

4.3 Stages of Transmission

Let $t = 0$ be the cycle when the SOP flit for the packet under analysis from τ_i first reaches the head of $\tau_i.V$. To help us think about how a packet travels through the NPS in the worst case, we also define the following additional time values:

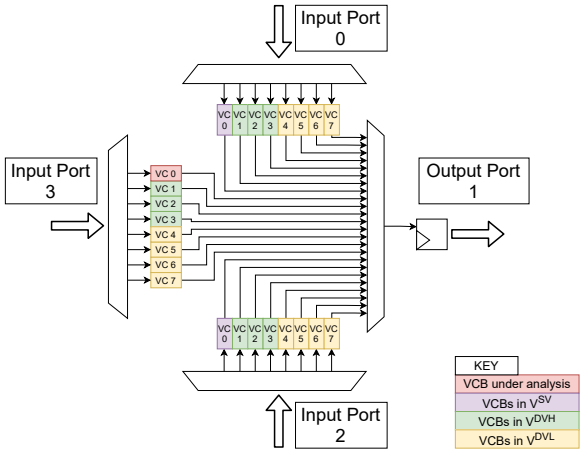


Figure 2: Example of the subsets of VCBs that can interfere with the VCB containing the packet under analysis.

- Time $t = t_R \geq 0$, after which the packet under analysis can never be blocked again by **Rule 4** or **Rule 5**. This will be the time $t \geq 0$ when we first have $\tau_i.V.c > 0$, or when the SOP flit of the packet wins arbitration with $\tau_i.V.c = 0$, if that happens first.
- $t = t_S \geq t_R$ when the SOP flit wins arbitration.
- $t = t_E \geq t_S$ when the EOP flit wins arbitration.

We note that:

- For $0 \leq t < t_R$ the packet is assumed to be blocked by **Rule 4** or **Rule 5**. Any other cause of blocking, such as being blocked by **Rule 2**, is redundant.
- For $t_R \leq t < t_S$ the packet cannot be blocked by **Rule 4** or **Rule 5** anymore. However, it can still be blocked by **Rule 2** while a packet from a VCB from V^{SV} is in progress, **Rule 3** or by losing the LRU arbitration.
- For $t_S \leq t < t_E$ the packet cannot be blocked by any of **Rules 2, 4** or **5**. However, it can still be blocked by **Rule 3** or by losing the LRU arbitration.

The following lemma limits how many flits each buffer can send in the interval $0 \leq t < t_R$.

Lemma 4.1. *A buffer $V_j \in \tau_i.op.\mathcal{V} \setminus \tau_i.V$ can only send up to $V_j.r + V_j.L$ flits to $\tau_i.op$ before t_R, where $V_j.L$ is the maximum length of a packet that crosses V_j and leaves through $\tau_i.op$.*

Proof. Starting with $V_j.c = V_j.r$, the buffer can send $V_j.r + 1$ flits, with the last one being a SOP flit that leaves $V_j.c = -1$. Rule 4 only blocks buffers with a negative token counter from sending SOP flits, so V_j can send the remaining $V_j.L - 1$ flits of the packet it started without necessarily requiring a token reload, for a total of $V_j.r + V_j.L$ 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 $V_j.r < 0$, due to Rule 4. □

The result below allows us to start describing how a buffer from $\mathcal{V}^{SV}$ can interfere with the packet under analysis.

Lemma 4.2. *A buffer $V_j \in \mathcal{V}^{SV}$ can only send one SOP flit in the $t_R \leq T < t_S$ interval, and only if it sent no flits during the $0 \leq T < t_R$ interval.*

Proof. For $t \geq t_R$, $\tau_i.V$ is not blocked by Rule 4, so it can only lose to a SOP from a buffer in $\mathcal{V}^{SV}$ if it loses the LRU arbitration. If V_j sent anything during the $0 \leq T < t_R$ interval, it necessarily has been used more recently than $\tau_i.V$. Otherwise it can win the LRU arbitration with a SOP once, but for the next SOP it will again have been used more recently than $\tau_i.V$. □

This leaves three options for $V_j \in \mathcal{V}^{SV}$, with respect to when it sends its final SOP and EOP flits (if any) during the $0 \leq T < t_E$ interval:

- Option 1: Send the final SOP and EOP flits before t_R.
- Option 2: Send the final SOP flit before t_R, but not the final EOP flit. Only one $V_j \in \mathcal{V}^{SV}$ can use this option.
- Option 3: Send the final SOP flit after t_R.

We can consider the eight possible assignments of the options above to the two buffers in V_j, and take as the worst case the assignment that results in the maximum value of $\tau_i.R$. If Option 1 is assigned to V_j, it can send up to $V_j.r + V_j.L$ flits before t_R. For each packet sent completely or partially in the interval, the parameter $f.BP$ of the corresponding CT can also be added to the interference. The backpressure does not necessarily cause a reset, as while a packet from V_j is in progress $\tau_i.V$ is not valid, and thus if no other buffer requests to transmit without tokens a reset will not happen.

Option 2 is similar to Option 1, but additional interference can be caused by the (up to $V.L - 1$) flits of the final packet sent after t_R losing the LRU arbitration to the buffers in the set $\mathcal{V}^{DVH}$.

If Option 3 is assigned to V_j, it can send a single packet after t_R. The parameter $f.BP$ of the corresponding CT can also be added to the interference, and the flits can cause additional interference by losing the LRU arbitration to the buffers in the set $\mathcal{V}^{DVH}$.

Regarding the set $\mathcal{V}^{DVH}$, each $V_j \in \mathcal{V}^{DVH}$ can send $V_j.r + V_j.L$ flits before t_R, as given by Lemma 4.1. After t_R, it can add interference by winning arbitration with one flit for every flit transmitted by a buffer from the set $\{\tau_i.V\} \cup \mathcal{V}^{SV}$, due to the LRU arbitration.

Finally, each $V_j \in \mathcal{V}^{DVL}$ can send $V_j.r + V_j.L$ flits before t_R. After t_R, it cannot win against a high-priority CT.

4.4 Optimization approach

We state the following optimization problem to determine the maximum interference a packet of τ_i can suffer. Maximize:

$$\tau_i.B = 1 + \tau_i.BP + \sum_{V_j \in \mathcal{V} - \tau_i.V} \tau_i.B(V_j), \tag{4}$$

where $\tau_i.B(V_j)$ is the interference caused by buffer V_j, whose calculation depends on the subset of $\mathcal{V}$ that V_j belongs to. The additional one cycle added covers the situation where $\tau_i.V.c < 0$ and no other $V \in \tau_i.op.\mathcal{V}$ passes Rules 1-3, in which case we lose a cycle waiting for the token reload without any interference from backpressure or other VCBs.

We begin with $V_j \in \mathcal{V}^{SV}$. For each $\tau_k \in V_j.\Gamma$, we define three variables $b(\tau_k), c(\tau_k), a(\tau_k)$, which count the number of packets of τ_k that are completely sent before t_R, that are in progress on t_R, and that are completely sent after t_R. The sum of the three variables must not be over the max number of packets from τ_k that we can have:

$$\begin{aligned} &\forall V_j \in \mathcal{V}^{SV}, \forall \tau_k \in V_j.\Gamma, \\ &b(\tau_k) + c(\tau_k) + a(\tau_k) \leq \left\lceil \frac{\tau_i.R + \tau_k.J}{\tau_k.T} \right\rceil. \end{aligned} \tag{5}$$

Depending on the option (1, 2 or 3) we have assigned to V_j, we also need additional constraints on $b(\tau_k), c(\tau_k), a(\tau_k)$ so the definition of the option is respected. For option 1, we only send before t_R:

$$\sum_{\tau_k \in V_j.\Gamma} c(\tau_k) = 0; \sum_{\tau_k \in V_j.\Gamma} a(\tau_k) = 0; \tag{6}$$

for option 2, a single packet from V_j can be in progress during t_R, and none are sent after:

$$\sum_{\tau_k \in V_j.\Gamma} c(\tau_k) = 1; \sum_{\tau_k \in V_j.\Gamma} a(\tau_k) = 0; \tag{7}$$

and finally for option 3, we start no packets before t_R and send a single one after:

$$\sum_{\tau_k \in V_j.\Gamma} b(\tau_k) = 0; \sum_{\tau_k \in V_j.\Gamma} c(\tau_k) = 0; \sum_{\tau_k \in V_j.\Gamma} a(\tau_k) = 1. \tag{8}$$

The interference $\tau_i.B(V_j)$ due to $V_j \in \mathcal{V}^{SV}$ will then be:

$$\begin{aligned} &\forall V_j \in \mathcal{V}^{SV}, \\ &\tau_i.B(V_j) = \sum_{\tau_k \in V_j.\Gamma} (\tau_k.L + \tau_k.BP) \times (b(\tau_k) + c(\tau_k) + a(\tau_k)), \end{aligned} \tag{9}$$

while the number of flits n^T sent by buffers from the set $\{\tau_i.V\} \cup \mathcal{V}^{SV}$ after t_R will be:

$$n^T = \tau_i.L + \sum_{V_j \in \mathcal{V}^{SV}} \sum_{\tau_k \in V_j.\Gamma} c(\tau_k) \times (\tau_k.L - 1) + a(\tau_k) \times \tau_k.L. \tag{10}$$

The interference $\tau_i.B(V_j)$ due to $V_j \in \mathcal{V}^{DVH}$ will then be at most:

$$\forall V_j \in \mathcal{V}^{DVH}, \tau_i.B(V_j) = \min(V_j.n, V_j.L + V_j.r + n^T), \tag{11}$$

where $V_j.L + V_j.r$ corresponds to the flits sent before t_R, while n^T corresponds to the flits sent after t_R by winning the LRU arbitration against the buffers from $\{\tau_i.V\} \cup \mathcal{V}^{SV}$. Note that the minimum operation with $V_j.n$ corresponds to the fact that each buffer from $\mathcal{V}^{DVH}$ can only interfere with as many flits as the CTs that could have been released in the time frame. Also note that we are using $V_j.n$, a value defined in function of $\tau_i.R$, as an input for calculating $\tau_i.R$ itself. In order to use the expression for $\tau_i.R$ as a fixed point equation, we start with an initial guess of $\tau_i.R = \tau_i.C$ and iterate until a fixed point is reached.

Finally, the interference $\tau_i.B(V_j)$ due to $V_j \in \mathcal{V}^{DVL}$ will be:

$$\forall V_j \in \mathcal{V}^{DVL}, \tau_i.B(V_j) = \min(V_j.n, V_j.L + V_j.r). \tag{12}$$

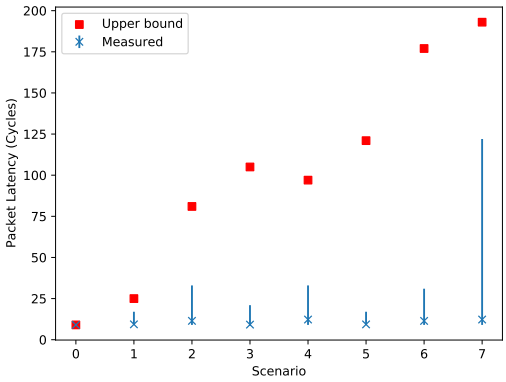


Figure 3: Results for the eight simulation scenarios.

5 EVALUATION

We develop a simulation framework for the configuration in Figure 2, that allows for configuring CT parameters and token registers. We connect a single NPS to custom NMU and NSU modules, which can inject packets programmatically and print timestamps for events. We use Modelsim to simulate the system, and a Python script to post-process simulation logs and calculate statistics.

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 $V_{3,0}$, and one CT for each VCB in the set $\mathcal{V}^{scen}$, defined depending on the scenario:

Scenario 0: $\mathcal{V}^{scen} = \emptyset$.
Scenario 1: $\mathcal{V}^{scen} = \mathcal{V}^{SV}$.
Scenario 2: $\mathcal{V}^{scen} = \mathcal{V}^{DVH}$.
Scenario 3: $\mathcal{V}^{scen} = \mathcal{V}^{DVL}$.
Scenario 4: $\mathcal{V}^{scen} = \mathcal{V}^{SV} \cup \mathcal{V}^{DVH}$.
Scenario 5: $\mathcal{V}^{scen} = \mathcal{V}^{SV} \cup \mathcal{V}^{DVL}$.
Scenario 6: $\mathcal{V}^{scen} = \mathcal{V}^{DVH} \cup \mathcal{V}^{DVL}$.
Scenario 7: $\mathcal{V}^{scen} = \mathcal{V}^{SV} \cup \mathcal{V}^{DVH} \cup \mathcal{V}^{DVL}$.

For simplicity, we assume a value of 16 for each token register, and the following parameters for all CTs: $T = D = 200$, $J = 20$, $L = 8$, $BP = 0$. To model a minimum interarrival time, we sample an exponential distribution with mean T and add to T to get the time between arrivals of packets from the same CT. Each scenario was simulated for 10^7 cycles.

The plot in Figure 3 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 V^{DVL} (scenarios 3, 5 and 6). As the VCBs from 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.

6 CONCLUSION

In this work, we present a formal description of the behaviour of the NPS components of the Versal NoC, and propose a model for computing upper bounds for communication times of high-priority packets, for the case of network clients exchanging data over a single NPS.

The natural next step for this work is to extend both the mathematical model and the simulation framework to cover the more realistic scenarios, with packets traversing multiple switches.

REFERENCES

[1] Mohamed S. Abdelfattah and Vaughn Betz. 2014. The Case for Embedded Networks on Chip on Field-Programmable Gate Arrays. *IEEE Micro* 34, 1 (2014), 80–89. https://doi.org/10.1109/MM.2013.131
[2] Achronix Semiconductor 2019. *Eight Benefits of Using an FPGA with an On-chip High-Speed Network (WP020).* Achronix Semiconductor.
[3] Achronix Semiconductor 2019. *Speedster7t Network on Chip User Guide (UG089).* Achronix Semiconductor.
[4] T. Ferrandiz, F. Frances, and C. Fraboul. 2009. A method of computation for worst-case delay analysis on SpaceWire networks. In *2009 IEEE International Symposium on Industrial Embedded Systems.* IEEE, USA, 19–27. https://doi.org/10.1109/SIES.2009.5196187
[5] B. Gaide, D. Gaitonde, C. Ravishankar, and T. Bauer. 2019. Xilinx Adaptive Compute Acceleration Platform: VersalTM Architecture. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Seaside, CA, USA) *(FPGA '19).* Association for Computing Machinery, New York, NY, USA, 84–93. https://doi.org/10.1145/3289602.3293906
[6] Y. Huan and A. DeHon. 2012. FPGA optimized packet-switched NoC using split and merge primitives. In *2012 International Conference on Field-Programmable Technology.* IEEE, USA, 47–52. https://doi.org/10.1109/FPT.2012.6412110
[7] Xilinx Inc. 2021. *Vivado Design Suite.* Xilinx Inc. https://www.xilinx.com/products/design-tools/vivado.html
[8] M. Langhammer, G. Baeckler, and S. Gribok. 2020. SpiderWeb - High Performance FPGA NoC. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW).* IEEE, USA, 115–118. https://doi.org/10.1109/IPDPSW50202.2020.00025
[9] L. M. Ni and P. K. McKinley. 1993. A survey of wormhole routing techniques in direct networks. *Computer* 26, 2 (1993), 62–76. https://doi.org/10.1109/2.191995
[10] B. Nikolić, S. Tobuschat, L. Soares Indrusiak, R. Ernst, and A. Burns. 2019. Real-Time Analysis of Priority-Preemptive NoCs with Arbitrary Buffer Sizes and Router Delays. *Real-Time Syst.* 55, 1 (Jan. 2019), 63–105. https://doi.org/10.1007/s11241-018-9312-0
[11] M. K. Papamichael and J. C. Hoe. 2012. 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* (Monterey, California, USA) *(FPGA '12).* Association for Computing Machinery, New York, NY, USA, 37–46. https://doi.org/10.1145/2145694.2145703
[12] Z. Shi and A. Burns. 2008. 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).* IEEE Computer Society, USA, 161–170.
[13] Burkhard Stiller, Thomas Bocek, Fabio Hecht, Guilherme Machado, Peter Racz, and Martin Waldburger. 2010. *Mobile Systems IV.* Technical Report. University of Zurich, Department of Informatics.
[14] I. Swarbrick, D. Gaitonde, S. Ahmad, B. Gaide, and Y. Arbel. 2019. Network-on-Chip Programmable Platform in Versal (TM) ACAP Architecture. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Seaside, CA, USA) *(FPGA '19).* Association for Computing Machinery, New York, NY, USA, 212–221. https://doi.org/10.1145/3289602.3293908
[15] I. Swarbrick, D. Gaitonde, S. Ahmad, B. Jayadev, J. Cuppett, A. Morshed, B. Gaide, and Y. Arbel. 2019. Versal Network-on-Chip (NoC). In *2019 IEEE Symposium on High-Performance Interconnects (HOTI).* IEEE, USA, 13–17. https://doi.org/10.1109/HOTI.2019.00016
[16] S. Tobuschat and R. Ernst. 2017. Real-time communication analysis for Networks-on-Chip with backpressure. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017.* IEEE, USA, 590–595. https://doi.org/10.23919/DATE.2017.7927055
[17] Xilinx Inc. 2020. *Versal ACAP Programmable Network on Chip and Integrated Memory Controller (PG 313).* Xilinx Inc.
[18] Q. Xiong, Z. Lu, F. Wu, and C. Xie. 2016. Real-Time Analysis for Wormhole NoC. In *Proceedings of the 26th edition on Great Lakes Symposium on VLSI.* ACM, USA. https://doi.org/10.1145/2902961.2903023