CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Journal Paper

HopliteRT*: Real-Time NoC for FPGA

This article was presented in part at the International Conference on Embedded Software 2020 and appears as part of the ESWEEK-TCAD special issue.

Yilian Ribot; Geoffrey Nelissen

CISTER-TR-201102

HopliteRT*: Real-Time NoC for FPGA

Yilian Ribot; Geoffrey Nelissen

CISTER Research Centre

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail:

https://www.cister-labs.pt

Abstract

With the increasing number of computation nodes integrated in multi and many-core platforms, network-on-chips (NoCs) emerged as a new communication medium in systems-on-chips (SoCs). HopliteRT is a new NoC design that was recently proposed to address the needs of real-time systems whilst respecting the constraints of field-programmable gate array (FPGA) platforms. In this article, we: 1) introduce priority-based routing in HopliteRT; 2) change the network topology in order to improve the packets 19 worst-case traversal time (WCTT); 3) identify a flaw in the existing timing analysis of HopliteRT; and 4) develop a new timing analysis that is proven correct. We also show by means of experiments that the modifications of HopliteRT proposed in this article allows for at least 2× improvement on the worst and average case traversal time of high priority packets, without impacting the quality of service of low-priority packets. The timing properties of high priority flows are greatly improved for negligible additional hardware costs. The proposed NoC has been implemented in Verilog and synthesized for a Xilinx Virtex-7 FPGA platform.

HopliteRT*: Real-Time NoC for FPGA

Yilian Ribot González and Geoffrey Nelissen

***Abstract*—With the increasing number of computation nodes integrated in multi and many-core platforms, network-on-chips (NoCs) emerged as a new communication medium in systems-on-chips (SoCs). HopliteRT is a new NoC design that was recently proposed to address the needs of real-time systems whilst respecting the constraints of field-programmable gate array (FPGA) platforms. In this article, we: 1) introduce priority-based routing in HopliteRT; 2) change the network topology in order to improve the packets' worst-case traversal time (WCTT); 3) identify a flaw in the existing timing analysis of HopliteRT; and 4) develop a new timing analysis that is proven correct. We also show by means of experiments that the modifications of HopliteRT proposed in this article allows for at least 2× improvement on the worst and average case traversal time of high priority packets, without impacting the quality of service of low-priority packets. The timing properties of high priority flows are greatly improved for negligible additional hardware costs. The proposed NoC has been implemented in Verilog and synthesized for a Xilinx Virtex-7 FPGA platform.**

***Index Terms*—Field programmable gate array, network-on-chips, real-time embedded systems, systems-on-chips, timing analysis.**

I. Introduction

SYSTEMS-ON-CHIPS (SoCs) are usually composed of several, possibly heterogeneous, processing elements (PEs). In order to communicate, PEs used to rely on shared busses. However, due to the large increase of on-chip elements during the last decade, communication through shared busses is not an appropriate solution for such platforms anymore. Indeed, at most one node can take control of a bus and transmit data at each cycle. This causes a bottleneck for the overall system. Network-on-chips (NoCs) have been identified as a good alternative to palliate this issue. NoCs are router-based packet switching networks and hence allow several PEs to transmit messages in parallel. As discussed in [1] and [2], NoCs have remarkable scalability, parallelism, and reusability properties, and help meet system-wide power requirements. Nevertheless, the use of NoCs in real-time systems requires also that their transmissions respect timing constraints.

Manuscript received April 17, 2020; revised June 17, 2020; accepted July 6, 2020. Date of publication October 2, 2020; date of current version October 27, 2020. This work was supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit under Grant UIDB/04234/2020. This article was presented in part at the International Conference on Embedded Software 2020 and appears as part of the ESWEEK-TCAD special issue. *(Corresponding author: Yilian Ribot González.)*

Yilian Ribot González is with the CISTER Research Centre, ISEP, Polytechnic Institute of Porto, 4200-465 Porto, Portugal (e-mail: ribot@isep.ipp.pt).

Geoffrey Nelissen is with the Department of Mathematics and Computer Science, Eindhoven University of Technology, 5612 AZ Eindhoven, The Netherlands.

Digital Object Identifier 10.1109/TCAD.2020.3012748

Concurrently to the growing complexity of SoCs, the capability improvements of field-programmable gate array (FPGA) platforms, and their flexibility to implement any digital functionality by programming reconfigurable elements, have promoted FPGAs as a valid alternative to application-specific integrated circuits (ASICs) for the development of custom-made SoCs. FPGAs allow designing systems with a high degree of parallelism and high data processing rate at a relatively low cost. A complete SoC composed of multiple soft-core processors [i.e., multiprocessor SoC (MPSoC)] may be implemented on an advanced FPGA (e.g., [3]). However, the number of soft-core processors that may be embedded is limited by the capacity of such platforms (i.e., limited by the number of FPGA' reconfigurable elements, called LCs). Most FPGAs do not supply enough resources to embed complex NoCs together with a large number of PEs.

The literature on NoCs is extensive. NoCs can differ considerably depending on their design features. Most of the proposed solutions that present suitable properties for real-time systems (i.e., those with deterministic behaviors and bounded worst-case timing properties) with dynamic traffic rely on wormhole switching [4] with virtual channels (VCs), and often implement some sort of priority-driven routing arbitration. VCs are buffers located in the input or output ports of each router. They allow storing flits (identically sized elements into which packets are divided) coming from different ports in a parallel fashion and then decide which one should be sent based on their priority. VCs are the backbone of the most common real-time NoCs arbitration policies that have been studied in the real-time systems, e.g., [5] and [6]. These strategies develop powerful NoC infrastructures with bounded WCTT but: 1) they are expensive to implement in terms of hardware surface requirements (especially in FPGA platforms); 2) their buffers and VCs increase the overall power consumption of the platform; and 3) their complexity renders their analysis complex as evidenced by the number of issues that were recently discovered and exposed in [7]–[10].

Most prominently, NoCs based on VCs such as those mentioned above require around 100 000 LCs to implement an 8×8 NoC, that is, between 20% and 150% of the total number of LCs that can be found in typical mid-range FPGAs. Therefore, VC-based NoCs are not suitable for systems implemented over FPGAs. In complete opposition, Hoplite is a newly proposed NoC infrastructure [11]. It is bufferless, does not use VCs, and each router is composed of only a few LCs. An 8×8 NoC requires 1%–8.5% of the total number of LCs offered in a mid-range FPGA. Nevertheless, Hoplite does not provide WCTT bounds and hence is not suitable for real-time systems.

Introduced by Wasly *et al.* [12], HopliteRT is a variation of Hoplite that makes it compatible with real-time systems'

requirements. However, even though it bounds the packets WCTT, in the worst-case scenario it may still require a flit to travel through every router in the network before reaching its destination. HopliteRT also treats all packets identically, i.e., it does not allow to associate different priorities, and thus the quality of services to different packets.

Contribution: In this article, we propose a new NoC design called HopliteRT* that keeps the advantages of HopliteRT while improving its real-time capabilities. The main contributions of this article are as follows.

1) To propose a solution to reduce the WCTT of a packet.
2) To introduce a notion of quality of service in the routing policy.
3) We identified a flaw in the timing analysis of HopliteRT and present a counter example.
4) We propose a worst-case communication time (WCCT) analysis of HopliteRT*.
5) We implemented our NoC in Verilog (a hardware description language) that can be instantiated on a real FPGA platform.
6) We present evaluation results of our new design against related work in terms of hardware requirements and computed WCCT bounds.

As discussed later in Section VI, contribution 4) shows that the changes introduced by contributions 1) and 2) allow the transmission of high priority packets to be, in the worst-case, twice as fast as in the original HopliteRT design without impacting the quality of service of low-priority packets, and in the average case, to be up to four times faster.

II. Related Work

Several NoC designs based on time-triggered routing arbitration protocols have been proposed over the years to address real-time systems requirements (e.g., [13] and [14]). It allows to isolate the timing properties of different flows by allocating precalculated transmission slots to them. This approach is extremely reliable and especially suited to critical systems. However, they require to know the complete system specification at configuration time and does not allow to adapt to changes in the system workload at runtime.

Several works [15], [16] have been published on the analysis of wormhole switching NoC with shared VCs as found in many COTS multi/many-core platforms, e.g., Kalray MPPA [17] and Tilera Tile [18].

Shi and Burns [19] proposed a WCTT analysis for a real-time NoC adopting a fixed priority preemptive routing protocol in which each priority level is assigned its own VC. Several variations of that NoC and its analysis were proposed over the years, for instance, handling the case where several flows share the same priority [20], changing the routing policy to EDF [21], or supporting communication flows with different criticality levels [22], [23]. However, the complexity of the NoC design and its routing policy led to several issues in their analysis [7]–[10]. To try to avoid the problematic cases mentioned in those publications, Nikolic *et al.* [24] recently proposed a new type of NoC relying on a global arbitration protocol centered around a CAN bus shared between all routers. Theoretical results are promising but one must still implement such NoC in a real platform. Alternatively, Giroudot and Mifdaoui [25], [26] addressed most of the limitations of the previous work by proposing a worst-case timing analysis of wormhole NoCs using network calculus. IDAMC [5] is another wormhole-based NoC designed specifically for mixed-criticality systems that use the back suction flow-control to implement service guarantees.

Hoplite is an inexpensive NoC design first proposed in [11] and [27]. Its routing policy is built upon the concept of deflection to avoid the cost of packet buffering, which makes it compact but does not allow to provide a bounded WCCT. Thus, it is not suited to real-time systems. Introduced in [12], HopliteRT is a variant of Hoplite that introduces: 1) a new routing protocol to prevent unbounded traversal times and 2) implements a traffic injection regulation protocol at each PE in order to avoid resource starvation. Thus, HopliteRT keeps the simplicity of Hoplite while ensuring bounded communication times for all communication flows. Finally, HopliteBuf is an evolution of HopliteRT that completely eliminates the notion of deflections and provides in-order packet delivery [28]. However, it requires to add large buffers in each router and hence increases the resource and power requirements of the NoC. The WCTT guaranteed by HopliteBuf is identical to that of HopliteRT.

III. System Model

In this article, we assume a system composed of m PEs $\{\pi_1, \ldots, \pi_m\}$ organized in a torus of size $S_x \times S_y$. Each PE π_k is connected to a different router R_k. The coordinates of the PE π_k (and its router R_k) in the torus are (x_k, y_k) with $0 \leq x_k < S_x$ and $0 \leq y_k < S_y$.

Each PE π_k injects a set of n_k communication flows $F_k = \{f_1, f_2, \ldots, f_{n_k}\}$ into the network. A communication flow f_i is defined by the tuple $\{x_o^i, y_o^i, x_d^i, y_d^i, \text{prio}_i, C_i, T_i\}$. A communication flow f_i generates a potentially infinite number of packets that are injected at coordinates (x_o^i, y_o^i) of the NoC and must reach the PE at coordinates (x_d^i, y_d^i). f_i respects a minimum interarrival time T_i between the generation of every two packets. Each packet sent by flow f_i is divided in C_i flits that are sequentially injected in the network. Each flit has a size S_{flit} (in bits). We assume that all the routing information is encoded in each flit of the packet, i.e., there is no distinction between header, body, or tail flits. The routing information is composed of the coordinates of the destination PE, and 1 bit encoding the priority prio_i (equal to high or low) of the associated flow. We denote the sets of high and low priority flows as $\text{hp} = \{f_i \mid \text{prio}_i = \text{high}\}$ and $\text{lp} = \{f_i \mid \text{prio}_i = \text{low}\}$.

IV. Background

In this section, we recall useful properties of HopliteRT and prove that its published timing analysis is incorrect.

A. HopliteRT Routing Protocol

Introduced in [12], HopliteRT is a variant of Hoplite that provides an upper bound on the WCTT of packets. HopliteRT implements a modified version of *X-Y* routing [a type of dimension ordered routing (DOR)]. Packets first travel east along the x-axis until they reach a router with the same X

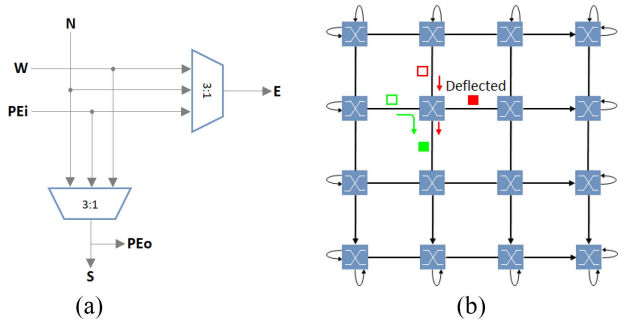

Fig. 1. Hoplite and HopliteRT designs. (a) HopliteRT router. (b) HopliteRT's routing policy.

TABLE I
ROUTING TABLE OF HOPLITERT

Input requests	Routing decisions	Explanation
W → E + N → S	W → E + N → S	No contention.
W → S + N → S	W → S + N → E	Conflict over the S port. W → S requests wins. N packet is deflected.
N → S + PE → E	N → S + PE → E	No contention.
W → E + PE → S	W → E + PE → S	No contention.
W → S + PE → E	W → S + ...	Not allowed. PE cannot inject its packet.

coordinate than their destination. The packet then turns south to travel along the y-axis until their destination. HopliteRT's routing policy differs from X-Y routing in that it allows packets to be "deflected" east while traveling south. A deflected packet must then travel along the x-axis again until reaching the same router where it was deflected and resume its journey south. Specifically, a packet may enter in a router by its N, W, or PE port [see Fig. 1(a)]. Packets entering by the W or PE port may request to go to the S or E output port. Packets entering by the N port may only request the S port. The packets injected by a programming element through the PE port always have the lowest priority and must wait for the requested port to be free. If both a packet entering by W and another entering by the N port request the S port at the same time, HopliteRT always gives the highest priority to the packet entering by the W port and deflects the packet entering by the N port toward the E port instead (see Table I).

Example: In Fig. 1(b), both the red packet (entering by the N port) and the green packet (entering by the W port) request to go south. Since the W port is given higher priority in the routing policy, the green packet pursues its route to the S port while the red packet is deflected toward the E port.

Note that because deflected packets travel along the x-axis, they will always enter by the W port, and hence will have the highest priority the next time they will require to go south. Therefore, the maximum number of deflections suffered by a packet can be upper bounded as discussed in Section IV-C. Nonetheless, a packet may be deflected after each and every hop on the y-axis, thereby leading to possibly large WCTTs.

Additionally, HopliteRT implements a traffic injection regulation protocol at each PE port in order to avoid programming elements to limit the number of packets that it may send in bursts when the output ports are available. This avoids some PEs to be indefinitely blocked because of unfair use of the bandwidth by other PEs.

B. HopliteRT Router Architecture

In HopliteRT, a router is implemented using two multiplexers of three inputs [see Fig. 1(a)]. HopliteRT takes advantage of the possibility of *fracturing* the lookup tables (LUTs) of modern FPGAs (i.e., the possibility to use a single LUT to implement two functions that would normally require two different LUTs) to reduce the implementation cost of the expensive crossbar multiplexers. The modern families of Xilinx FPGAs present 6-inputs LUTs that can be fractured in two 5-inputs LUTs sharing the same five input signals. Since each 3:1 multiplexer can be implemented with a 5-inputs LUT, the two multiplexers of the router can be implemented with a single 6-inputs LUT.

C. HopliteRT Worst-Case Traversal Time

The WCTT $wctt$ of a flit transmitted with HopliteRT between two nodes with coordinates (x_o, y_o) and (x_d, y_d) in a torus of size $S_x \times S_y$ is given by (1) (in clock cycles) [12]

$$wctt = h_x + h_y + \left(h_y \times S_x\right) + 2 \tag{1}$$

where h_x and h_y are the distances traveled by the packet on the x- and y-axes, respectively, when it does not contend with any other packet (i.e., without any deflection). Then

$$h_x = (x_d - x_o + S_x) \mod S_x \tag{2}$$

$$h_y = \left(y_d - y_o + S_y\right) \mod S_y \tag{3}$$

where mod is the modulo operator.

The term $(h_y \times S_x)$ in (1) accounts for the total cost of potential deflections; according to HopliteRT's routing policy, a packet can be deflected at most h_y times and each such deflection increases the packet's traversal time by S_x hops. The two additional hops in (1) represent the injection of the flit into the network by the PE and its exit at its destination.

A bound on the worst-case injection time (i.e., worst-case delay before a PE may be able to inject a packet into the NoC) is also proposed in [12]. However, as shown below by means of a counterexample that bound is incorrect as it may return optimistic and hence unsafe results.

D. Counterexample to the WCIT Bound of [12]

The traffic injection regulation protocol of HopliteRT is implemented using a leaky bucket implemented with two cascaded counter at each PE π_k: 1) a rate counter and 2) a token counter. The first counter overflows every θ_k cycles, where θ_k is the flit injection period of the programming element π_k. Then, the second counter is incremented on each overflow until it reaches a maximal value σ_k. The value of the token counters determines the maximum number of flits that the programming element can inject consecutively. The value of the token counter is decremented whenever a flit is injected in the network by π_k.

Let Γ_p^C denote the set of flows that conflict with the injection of a packet p of flow f_i by π_k. Then, $\sigma(\Gamma_p^C)$ and $\rho(\Gamma_p^C)$

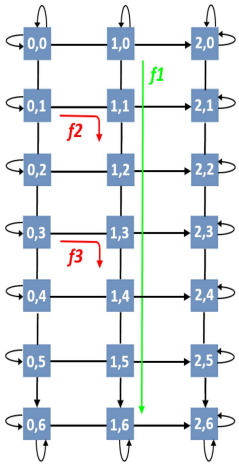

Fig. 2. Counter-example to the WCIT bound of [12] and [29].

define the cumulative burst length and injection rate of the conflicting flows, respectively. Wasly *et al.* [12], [28], [29] proved that if $\rho(\Gamma_p^C) < 1$ (i.e., the cumulative injection rate of conflicting flows is less than 1 packet/cycle) eventually there will be available clock cycles, hence the injection of p will not be infinitely blocked. Then, (4) is presented as a bound on the WCIT of a flow f_i injected by π_k, assuming that the condition $\rho(\Gamma_p^C) < 1$ is satisfied

$$wcit = (\theta_k - 1) + B(p) \tag{4}$$

with

$$B(p) = \left\lceil \frac{\sigma\left(\Gamma_p^C\right)}{1 - \rho\left(\Gamma_p^C\right)} \right\rceil \tag{5}$$

where $\sigma(\Gamma_p^C) = \sum_{f_l \in \Gamma_p^C} \sigma_l$ and $\rho(\Gamma_p^C) = \sum_{f_l \in \Gamma_p^C} (1/\theta_l)$.

Note (5) above assumes that a conflicting flow inherits the burst length and injection period of its origin router, i.e., σ_l and θ_l refer to the burst length and regulation period of the origin router of flow f_l.

Now, consider the system presented in Fig. 2. It consists of three flows f_1, f_2, and f_3. f_1 is injected in router $(1, 0)$ and has for destination $(1, 6)$. f_2 is injected at $(0, 1)$ and has for destination $(1, 2)$. f_3 is injected at $(0, 3)$ and has for destination router $(1, 4)$.

We are interested about the maximum blocking delay $B(p)$ suffered by a packet p injected to the S port of the router $(1, 5)$. The set of conflicting flows $\Gamma_p^C = \{f_1\}$, i.e., only f_1 may pass through the S port of router $(1, 5)$ and block packet p.

Assume that the burst length $\sigma_1 = 1$ and the regulation period $\theta_1 = 4$ at f_1's origin router. Then, according to (5), the maximum amount of time the S port of the router $(1, 5)$ may be kept busy by conflicting flows is given by $B(p) = \lceil (\sigma(\Gamma_p^C))/(1 - \rho(\Gamma_p^C)) \rceil = \lceil 1/(1 - (1/4)) \rceil = 2$.

However, let us now assume that f_1 injects three packets at times 0, 4, and 8 (note that those times respect the regulation period of 4 at f_1's origin router). Then, assume that f_2 injects packets at times 0 and 4. This means that the two first packets of f_1 will be deflected by packets of f_2 in router $(1, 1)$ while the third packet of f_1 will not be deflected. Therefore, the three packets of f_1 will reach router $(1, 2)$ at times 5, 9, and 10, respectively. Now, suppose that flow f_3 injects a packet at time 5. Then, the first packet of f_1 will be deflected for a second time in router $(1, 3)$. That is, the three packets of f_1 will reach router $(1, 5)$ at times 11, 12, and 13, respectively. Since they all arrive at one clock cycle of the interval, this means that the S port of router $(1, 5)$ will be kept busy by the conflicting flow f_1 during three clock cycles, thereby contradicting (5).

We conclude that the analysis in [12], [28], and [29] is incorrect because it forgot to account for the fact that different packets of the same flow may suffer different numbers of deflections on their route to their destination.

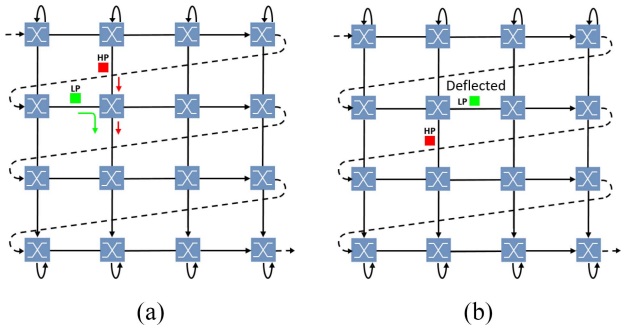

Fig. 3. HopliteRT* priority-based routing example. (a) Packet route requests. (b) Situation after routing arbitration.

V. Improving HopliteRT's Real-Time Capabilities

In this section, we present HopliteRT*, a variant of HopliteRT designed to: 1) introduce priorities in the routing policy and 2) decrease the WCTT of high-priority packets while keeping the WCTT of low-priority packets bounded. We prove a new WCTT and WCIT analysis in Section VI.

A. Introducing Priority Levels

A requirement of many real-time embedded systems is to provide different quality of service to different classes of traffic. It is classically done by assigning different priorities to those classes. Thus, we add a notion of packet priority in the arbitration mechanism of HopliteRT*. It is based on two priority levels (low and high) routing scheme. Our main objective is to ensure that low priority packets cannot interfere with the WCTT of high priority packets.

In HopliteRT, the packets coming from the W port always have the highest priority. Instead, in HopliteRT*, low priority packets coming from the W port will never be permitted to deflect high priority packets coming from the N port (see Fig. 3 for an example). That is, if a high priority packet coming from the N port [red packet in Fig. 3(a)] and a low priority packet coming from the W port [green packet in Fig. 3(a)] conflict for the S port, then the N packet wins the right to use the S port, and the W packet is deflected toward the E port [see Fig. 3(b)]. In any other case, the routing policy is the same as in HopliteRT (refer to Section IV-A and Table I). To implement this new routing policy, the packet priority is encoded in its most significant bit. Table II summarizes the routing policy of HopliteRT*.

TABLE II
ROUTING TABLE OF HOPLITERT*

Input requests	Routing decisions	Explanation
W → E + N → S	W → E + N → S	No contention.
W^{HP}→ S + N^{LP}→S	W → S + N → E	High priority W → S always wins;
W^{HP}→ S + N^{HP}→S	W → S + N → E	High priority W → S always wins;
W^{LP}→ S + N^{LP}→S	W → S + N → E	W → S wins over N → S of same priority;
W^{LP}→ S + N^{HP}→S	W → E + N → S	A low priority request never wins over a high priority one.
N → S + PE → E	N → S + PE → E	No contention.
W → E + PE → S	W → E + PE → S	No contention.
W → S + PE → E	W → S + …	Not allowed. PE cannot inject its packet;

B. Ensuring Progress

Even though it looks beneficial, the new priority-based routing policy described in Section V-A is in fact extremely inefficient; the WCTT of high priority packets remains unchanged (only their average-case traversal time is potentially reduced), but more importantly, the WCTT of low priority packets is not bounded anymore.

Consider the red packet in Fig. 1(b). Since that packet was deflected, it will hop through S_x routers (where S_x is the number of routers on the x-axis) before entering again in the same router in which it was initially deflected. That is, the packet did not progress at all toward its destination after those S_x additional hops. If the red packet has a low priority, it may again be deflected in the same router, and again, it will not experience any progress during the next S_x hops.

We prevent the lack of progress discussed above by changing the network topology. We connect the routers together considering a *circulant topology* as shown in Fig. 4(a). In that topology, all routers are connected by a single unidirectional ring (red links in Fig. 4). Then, every pair of routers that are S_x positions apart on the ring are connected by a bypass (green and black links in Fig. 4). Equivalently, if we look at the network as a grid [see Fig. 4(b)], the main unidirectional ring (in red) corresponds to the rows of the torus where the E port of the last router in row number y is connected to the W port of the first router in row number $(y + 1) \mod S_y$. Similarly, the bypasses (green and black in the figure) correspond to the links on the columns of the torus, where the last router in a column is connected to the first router in that same column. That is, in Fig. 4, the green links in the inset (a) correspond to the green links in the inset (b).

Thanks to this new topology, when a packet is deflected, then S_x hops later, it reaches the same router as it would have if it was not deflected. That is, the packet always progresses toward its destination even when deflected. Consequently, the WCTT of all packets is: 1) bounded and 2) decreases in comparison to HopliteRT (see Section VI).

Note that the circulant topology does not suffer from the same limitations as the traditional ring topology as it is fully scalable. Indeed, the bypasses allow the NoC to have as much bandwidth as with a torus topology.

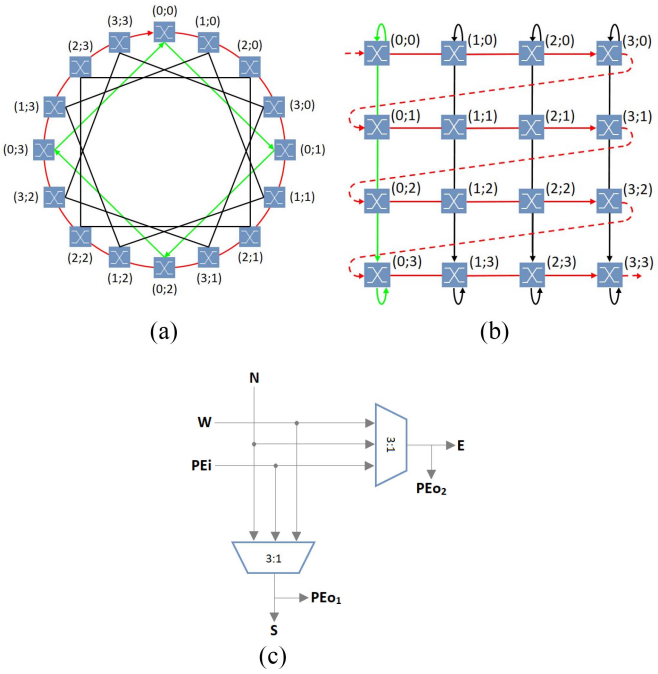

Fig. 4. Circulant topology of HopliteRT* and its router architecture. (a) Circulant topology $C(16;1,4)$. (b) Equiv. grid representation. (c) Router microarchitecture.

C. Modification to the Router Architecture

The new topology of HopliteRT* requires to slightly modify the router design. Indeed, originally, in the particular case where two packets arrive at the same instant in the same router (via the W and N ports) and that router is their destination, HopliteRT would send one packet to the programming element and would deflect the other to the E port. Then, the deflected packet would travel around the whole row in order to reach its destination router once more and finally be sent to the PE. If the same approach was adopted in HopliteRT*, because of the change of topology, the deflected packet may have had to travel around the entire network (instead of just the current row) before reaching its destination router for the second time. This situation causes a remarkable and unacceptable increase in the WCTT of the deflected packet. We aim at solving this issue by allowing the programming element connected to the router to read both packets simultaneously. For this reason, we connect inputs of the programming element to both the E and S ports of the router (instead of just the S port as in HopliteRT). We call those PEo_1 and PEo_2 in Fig. 4(c). That is, PEo_1 shares its wires with the S port and PEo_2 share its wires with the E port. Additionally, a new wire is used to signal the availability of a message to the PE on the E port. Sharing output ports with the inputs of the PE slightly increases the complexity of the logic in the router, but it avoids the implementation of expensive multiplexers.

To accommodate the arrival of packets on a PE's output ports, we consider that each PE has two FIFO queues, one per output port. By connecting PEo_1 and PEo_2 to these buffers, we ensure that each PE will be able to serve two petitions per cycle (i.e., one from PEo_1 and PEo_2) and, therefore, no backpressure is created in the network.

We also consider that each PE has two FIFO queues (which may be implemented in software or hardware), one per priority level, for flits that are pending to be injected into the network. High priority packets are injected first in the network. Flits of low priority flows are injected only when the high priority FIFO queue is empty. That is, high priority packets do not suffer delay due to low priority packets sent by the same PE.

Additionally, in this article, we assume that there is no traffic injection regulator at PEs, that is, PEs can inject flits as fast as possible. However, we assume that each flow may have at most one packet in the FIFO queue pending to be injected in the network at any time instant. After injecting a packet, a new packet from the same flow can be stored in the FIFO to be injected in the network. In other words, we assume that $\forall f_i,\ T_i \geq wcit_i$.

VI. Bound on the Worst-Case Communication Time

In the previous section, we described HopliteRT*. In this section, we propose an analysis of the worst-case communication time (WCCT) between two PEs connected with HopliteRT*. The WCCT of a packet is defined as the sum of the maximum amount of time *wcit* during which the last flit of the packet must wait in the PE before to be injected in the network, and the maximum amount of time *wctt* taken by any flit of the packet to traverse the network and reach its destination. Thus, the WCCT of a packet pertaining to flow f_i is defined as

$$wcct_i = wcit_i + wctt_i \tag{6}$$

where $wcit_i$ is the worst-case injection time and $wctt_i$ is the WCTT of flow f_i. To avoid notation clutter, we omit to specify the index of the flow when referring to their WCTT and WCIT when there is no ambiguity.

A. Worst-Case Traversal Time

We start by deriving a bound on the WCTT of any flit of a packet p. A bound on the WCIT will be proven in Section VI-C.

We decompose the WCTT of a flit of flow f_i in two terms

$$wctt_i = n^i_{\text{hops}}(x_d, y_d) + n^i_{\text{def}}(x_d, y_d) \times c_{\text{def}} \tag{7}$$

where $n^i_{\text{hops}}(x_d, y_d)$ is the number of hops in a network with zero load (i.e., when the flit does not suffer any deflection) until its destination (x_d, y_d), $n^i_{\text{def}}(x_d, y_d)$ is the maximum number of deflections suffered by the flit on its route until its destination, and c_{def} is the cost of a deflection. As for WCTT and WCIT, in the following, we omit the superscript i of $n^i_{\text{hops}}(x_d, y_d)$ and $n^i_{\text{def}}(x_d, y_d)$ when there is no ambiguity on the flow to which it refers.

The term n_{hops} is defined as

$$n_{\text{hops}}(x_d, y_d) = h_r(x_d, y_d) + h_b(x_d, y_d) + 2 \tag{8}$$

where $h_r(x_d, y_d)$ and $h_b(x_d, y_d)$ are the number of hops on the ring and bypasses, respectively, until the destination (x_d, y_d). The additional two hops account for the injection (at the source node) and exit (at the destination node) of the flit into and from the network. In the following, we prove upper bounds for $h_r(x_d, y_d)$ (Lemma 1), $h_b(x_d, y_d)$ (Lemma 2), $n_{\text{def}}(x_d, y_d)$ (Lemmas 3 and 5), and $c_{\text{def}}(x_d, y_d)$ (Lemma 6).

Lemma 1: The number of hops on the ring by a flit of flow f_i to reach a destination (x, y) in a zero-load network is given by $h^i_r(x, y) = (x - x^i_o + S_x) \mod S_x$.

Proof: According to our routing policy, each flit travels first through the ring from the origin router at coordinate (x^i_o, y^i_o) until it reaches a router with the same X coordinate x as the destination. According to the topology presented in Fig. 4(b), the number of hops $h^i_r(x, y)$ on the ring is thus, $(x - x^i_o)$ when $x \geq x_o$ and $(x - x^i_o + S_x)$ when $x < x_o$. That is, $h^i_r(x, y) = (x - x^i_o + S_x) \mod S_x$. ■

Lemma 2: The number of hops on bypasses by a flit of flow f_i to reach a destination (x, y) in a zero-load network is given by $h^i_b(x, y) = (y - y^{i'}_o + S_y) \mod S_y$, where

$$y^{i'}_o = \begin{cases} y^i_o, & \text{when } x \geq x^i_o \\ y^i_o + 1, & \text{when } x < x^i_o. \end{cases} \tag{9}$$

Proof: Remember that a bypass in Fig. 4(a) corresponds to a link of a column of the modified torus in Fig. 4(b). Let S_y be the number of routers in a column, and $y^{i'}_o$ be the Y coordinate of the router at which the packet stops traveling on the ring and starts using bypasses (i.e., the first router with the same X coordinate as the destination). Then, according to the router numbering shown in Fig. 4(b), $y^{i'}_o = y^i_o$ when $x \geq x^i_o$ and $y^{i'}_o = y^i_o + 1$ when $x < x^i_o$, and the number of hops $h^i_b(x, y)$ on the y-axis of the torus is $y - y^{i'}_o$ when $y \geq y^{i'}_o$ and $y - y^{i'}_o + S_y$ otherwise. That is, $h^i_b(x, y) = (y - y^{i'}_o + S_y) \mod S_y$. ■

The maximum number of deflections $n_{\text{def}}(x, y)$ that a flit may suffer until its destination (x, y) differ for high and low priority packets. We analyze both cases in Lemmas 3 and 5.

Lemma 3: The maximum number of deflections suffered by a flit of a low priority packet with destination (x, y) is bounded by $n_{\text{def}}(x, y) \leq h_b(x, y)$.

Proof: According to HopliteRT*'s routing policy, a low priority flit entering from the W or N port may always be deflected. Therefore, a low-priority packet may be deflected as many times as it may try to use a bypass, i.e., h_b times. ■

Lemma 4: A flit of a high priority packet cannot be deflected in two successive routers on the same column (i.e., two routers directly connected to one another by their S and N ports).

Proof: We prove this lemma by contradiction. Consider that the high priority flit is deflected in two successive routers R_k and R_l on the same column of the modified torus (i.e., the S port of R_k is connected to the N port of R_l). Because the flit is deflected in R_l, then, according to the routing policy of HopliteRT*, it must have entered by the N port of R_l. That is, it must have exited R_k by the S port. Since the flit left R_k by the S port, it means that it was not deflected in R_k, thereby leading to a contradiction. ■

Lemma 5: The maximum number of deflections suffered by a flit of a high priority packet with destination (x, y) is bounded by $n_{\text{def}}(x, y) \leq \lfloor h_b(x, y)/2 \rfloor$.

Proof: According to the routing policy of HopliteRT*, a high priority flit can only be deflected when it enters by the N port of a router. That is, it can only be deflected when it

travels along bypasses. Moreover, a flit can only use bypasses that belong to the same column (i.e., same Y-coordinate).

By definition of $h_b(x, y)$, the flit under analysis does at most $h_b(x, y)$ hops through bypasses, all of which are consecutive links of the same column of the modified torus, and at most $(h_b(x, y) - 1)$ of those hops can be made by entering by the N port of a router. Furthermore, by Lemma 4, a high priority flit cannot be deflected in two successive routers on the same column of the modified torus. Therefore, the flit under analysis may be deflected in at most half of the routers, i.e., in $\lceil (h_b(x, y) - 1)/2 \rceil = \lfloor h_b(x, y)/2 \rfloor$ routers, which proves the lemma. Note that the last equality holds because $h_b(x, y) \in \mathbb{N}$. ■

The additional cost in terms of hops introduced by each deflection is analyzed in Lemma 6.

Lemma 6: The cost of a deflection is $c_{\text{def}} = S_x - 1$.

Proof: When a flit is deflected, it must hop through S_x routers on the ring to reach the same router as it would have if it could have used the bypass instead, i.e., though S_x routers instead of 1, thus leading to an additional cost of $S_x - 1$. ■

Injecting all the bounds proven in Lemmas 3, 5, and 6 into (7), we can compute the WCTT of any flit of any flow f_i.

B. Improved Analysis for the WCTT

The analysis proposed in Section VI-A only uses information on the packet under analysis and does not rely on any information related to other communication flows that may be transmitted in the system. That analysis is thus useful for dynamic systems where the set of communication flows may vary over time. However, it may also be pessimistic if more information on the system is known. Indeed, the analysis of Section VI-A always assumes that the packet under analysis will suffer the maximum number of possible deflections. This may never happen in the real network if, for instance, there are no other flows using the same route than p. In this section, we derive more precise bounds on n_{def} based on the knowledge of the actual set of flows that can interfere with the transmission of the packet under analysis.

To derive the sets of flows that may interfere with a packet p under analysis, we first define the sets $\Gamma_k^{N\to S}$, $\Gamma_k^{W\to S}$, and $\Gamma_k^{E\to S}$ as the sets of flows that traverse a router R_k from the N to S ports, from W to S, and from W to E, respectively, assuming that no deflection ever happens in the network.

Lemma 7: $\Gamma_k^{N\to S} = \{f_i \mid (x_d^i = x_k) \wedge h_b^i(x_k, y_k) > 0 \wedge h_b^i(x_d^i, y_d^i) \geq h_b^i(x_k, y_k)\}$.

Proof: According to the modified DOR routing policy adopted by HopliteRT*, a flow f_i may request to leave by the S port of R_k only if its destination is on the same column of the modified torus than R_k (i.e., $x_d^i = x_k$) and its destination is either R_k or further south than R_k [i.e., $h_b^i(x_d^i, y_d^i) \geq h_b^i(x_k, y_k)$]. Furthermore, a flow cannot enter by the N port of R_k if it must not perform at least one hop on a bypass to reach R_k, i.e., we must have $h_b^i(x_k, y_k) > 0$. ■

Lemma 8: $\Gamma_k^{W\to S} = \{f_i : (x_d^i = x_k) \wedge h_b^i(x_k, y_k) = 0\}$.

Proof: According to the modified DOR routing policy adopted by HopliteRT*, a *nondeflected* flow f_i may enter by the W port only if it does not need to use any bypass to reach R_k [i.e., $h_b^i(x_k, y_k) = 0$]. Furthermore, it may request leaving by the S port of R_k only if its destination is on the same column of the modified torus than R_k (i.e., $x_d^i = x_k$). ■

Lemma 9: $\Gamma_k^{W\to E} = \{f_i : h_b^i(x_k, y_k) = 0 \wedge h_r^i(x_d^i, y_d^i) > h_r^i(x_k, y_k)\}$.

Proof: According to the modified DOR routing policy adopted by HopliteRT*, a *nondeflected* flow f_i may enter by the W port only if it does not need to use any bypass to reach R_k [i.e., $h_b^i(x_k, y_k) = 0$]. Furthermore, it may request to exit by the E port only if the number of hops it must do on the ring to reach its destination is larger than the number of hops it must do on the ring to reach R_k [i.e., $h_r^i(x_d^i, y_d^i) > h_r^i(x_k, y_k)$]. ■

We define def_k^{hp} and def_k^{lp} as binary functions that return 1 if high and low priority flows may be deflected in the router R_k, respectively.

Lemma 10: Let R_n be the router directly north to R_k, then

$$\text{def}_k^{hp} = \begin{cases} 1, & \Gamma_k^{N\to S} \cap \text{hp} \neq \emptyset \\ & \wedge \left(\Gamma_k^{W\to S} \cap \text{hp} \neq \emptyset \vee \text{def}_n^{hp} = 1\right) \\ 0, & \text{otherwise.} \end{cases} \quad (10)$$

Proof: According to HopliteRT*'s routing policy, a high priority flow can be deflected to the E port only if it is a flow incoming by the N port that conflicts for the S port (i.e., there must be $\Gamma_k^{N\to S} \cap \text{hp} \neq \emptyset$). Furthermore, conflicting flows must be of high priority too and must be incoming by the W port. Since the flows that were deflected in R_n are the only *deflected* flows that may enter by the W port and request the S port of R_k, there must either be $\text{def}_n^{hp} = 1$ or there is at least one *nondeflected flow* entering by the W port and requesting the S port (i.e., $\Gamma_k^{W\to S} \cap \text{hp} \neq \emptyset$). ■

Lemma 11: Let R_n be the router directly north to R_k, then

$$\text{def}_k^{lp} = \begin{cases} 1, & \Gamma_k^{N\to S} \cap \text{hp} \neq \emptyset \\ & \wedge \left(\Gamma_k^{W\to S} \cap \text{lp} \neq \emptyset \vee \text{def}_n^{lp} = 1\right) \\ 1, & \Gamma_k^{N\to S} \cap \text{lp} \neq \emptyset \\ & \wedge \left(\Gamma_k^{W\to S} \neq \emptyset \vee \text{def}_n^{lp} = 1 \vee \text{def}_n^{hp} = 1\right) \\ 0, & \text{otherwise.} \end{cases} \quad (11)$$

Proof: According to HopliteRT*'s routing policy, a low priority flow can be deflected to the E port under two possible scenarios: 1) it is a flow entering by the W port that conflicts for the S port (i.e., $\Gamma_k^{W\to S} \cap \text{lp} \neq \emptyset$) with a high priority flow coming from the N port (i.e., $\Gamma_k^{N\to S} \cap \text{hp} \neq \emptyset$); or 2) if it is a flow coming from the N port that conflicts for the S port (i.e., $\Gamma_k^{N\to S} \cap \text{lp} \neq \emptyset$) with any flow coming from the W port. We remind that the flows that were deflected in R_n are the only *deflected* flows that may enter by the W port and request the S port of R_k. Therefore, for scenario 2) to happen, there must be $\text{def}_n^{lp} = 1$ or $\text{def}_n^{hp} = 1$, or there is at least one *nondeflected* flow entering by the W port and requesting the S port (i.e., $\Gamma_k^{W\to S} \neq \emptyset$). Scenarios 1) and 2) directly correspond to the first and second case in (11), respectively. ■

The functions def_k^{hp} and def_k^{lp} allows us to identify the set of routers in which a flit may be deflected. Therefore, thanks to them, we can now compute the maximum number of deflections a flit may suffer on its route to its destination. We first derive such bound for low priority flits in Lemma 12.

Lemma 12: The maximum number of deflections suffered by a flit of a low priority packet p of flow f_i with destination (x, y) is

$$n_{\text{def}}(x, y) = \sum_{R_k \in \mathcal{R}^b(p)} \text{def}_k^{lp} \tag{12}$$

where $\mathcal{R}^b(p) = \{R_k \mid x_k = x \wedge y_k = (y_o^{i'} + j) \mod S_y \forall j = 0, \ldots, h_b(x, y) - 1\}$.

Proof: The set $\mathcal{R}^b(p)$ contains all the routers on the same column x than the destination of flow f_i and that are on its route to its destination. That is, those routers between the Y coordinate $y_o^{i'}$ and the destination y of the flow. Therefore, it contains all the routers in which a packet of f_i may be deflected (since a packet can only be deflected in a router on the same column of its destination and that is on its route to its destination). Since, by Lemma 11, def_k^{lp} returns 1 if a low priority packet may be deflected in router R_k, and 0 otherwise, the sum $\sum_{R_k \in \mathcal{R}^b(p)} \text{def}_k^{lp}$ returns the total number of routers in which a packet of f_i may be deflected. ■

The procedure to compute the maximum number of deflections for a flit of a high priority packet p is a bit more complex. Let us first define the set $\mathcal{R}^{\text{def}}(p)$ as the set of routers in which a flit of the high priority packet p under analysis may be deflected. That is, $\mathcal{R}^{\text{def}}(p) = \{R_k \mid R_k \in \mathcal{R}^b(p) \wedge \text{def}_k^{hp} = 1\}$, where $\mathcal{R}^b(p)$ is defined as in Lemma 12. The size of that set is obviously an upper bound on the number of deflections that may be suffered by a flit of p. However, that value would be very pessimistic. Indeed, according to Lemma 4, the same high priority flit cannot be deflected in two successive routers in the same column of the modified torus. Lemma 13 integrates that information to compute a tighter bound on the maximum number of deflections suffered by a flit of p.

Lemma 13: Let $\mathbb{G}^{\text{def}}(p)$ be a graph that contains one vertex per router in $\mathcal{R}^{\text{def}}(p)$, and such that any two vertices V_i and V_j of $\mathbb{G}^{\text{def}}(p)$ are connected by an edge if the routers R_i and R_j corresponding to those vertices are direct neighbors (i.e., they are connected by a $S \rightarrow N$ link). The maximum number of deflections suffered by a flit of the high priority packet p is the size of the maximum independent set of $\mathbb{G}^{\text{def}}(p)$.

Proof: The maximum independent set of a graph $\mathbb{G}^{\text{def}}(p)$ is the largest subset $\mathcal{S}$ of vertices of $\mathbb{G}^{\text{def}}(p)$ such that any two vertices in $\mathcal{S}$ is not connected by an edge in $\mathbb{G}^{\text{def}}(p)$.

Since vertices in $\mathbb{G}^{\text{def}}(p)$ are connected by an edge if and only if they are neighbors in the NoC (i.e., they are connected by a $N \rightarrow S$ link), the maximum independent set $\mathcal{S}$ of $\mathbb{G}^{\text{def}}(p)$ contains the largest possible number of routers from $\mathcal{R}^{\text{def}}(p)$ that are not connected by a $S \rightarrow N$ link. That is, it contains the largest number of routers in which a flit of packet p may be deflected and that are not successive routers in the same column of the network. Therefore, it contains the maximum number of routers in which a flit of p may be deflected while respecting the constraint set by Lemma 4. ■

The new bounds on the number of deflections provided in Lemmas 12 and 13 can then be used instead of those in Lemmas 3 and 5 to compute the WCTT with (7).

C. Worst-Case Injection Time

In the previous sections, we derived upper bounds on the maximum traversal time of any flit of a packet p. In this section, we provide an analysis for the worst-case injection time of p.

We first discuss the best case injection scenario (Lemma 14) for flow f_i. The worst-case injection scenario is then discussed in Lemma 15.

Lemma 14: In any time interval of length t, the flow f_i can transmit at most $\lambda_i(t) = \min\{t, \lceil (t + wcit_i)/T_i \rceil C_i\}$ flits.

Proof: Since at most one flit can be sent by f_i every clock cycle, it holds that

$$\lambda_i(t) \leq t. \tag{13}$$

Moreover, let $wcit_i$ be the WCIT suffered by any packet of flow f_i. Then, f_i injects the most flits in an interval of length t when one of its packet was kept from being injected during $wcit_i$ cycles before the beginning of the interval that packet starts to be injected right at the start of the interval, new packets are generated with their minimum interarrival time T_i and no flit of f_i suffers blocking during the interval of length t. Under such conditions, we have

$$\lambda_i(t) \leq \left\lceil \frac{t + wcit_i}{T_i} \right\rceil C_i. \tag{14}$$

Since the minimum of two upper bounds is an upper bound, the minimum of (13) and (14) is an upper bound on $\lambda_i(t)$. ■

Now that we discussed the best case scenario, we consider the worst-case delay that a flow may experience to inject a packet in the network. To ease the discussion, we denote by Γ_k^C the set of flows that conflict with the injection of a packet p at router R_k. Let us assume that the set of conflicting flows Γ_k^C is known. We can upper bound $wcit_i$ as in Lemma 15.

Lemma 15: Let p be a packet of flow f_i injected at router R_k with coordinates (x_k, y_k). The WCIT $wcit_i$ caused by flows conflicting with packet p is given by the smallest positive solution to the recursive equation

$$wcit_i \geq \sum_{\forall f_l \in \mathcal{I}_k} C_l + \sum_{\forall f_j \in \Gamma_k^C} \lambda_j\big(wcit_i + J_j + 1\big) \tag{15}$$

where $J_j = n_{\text{def}}^j(x_k, y_k) \times c_{\text{def}}$ and

$$\mathcal{I}_k = \begin{cases} F_k, & \text{if } \text{prio}_i = \text{low} \\ F_k \cap \text{hp}, & \text{if } \text{prio}_i = \text{high}. \end{cases}$$

Proof: According to HopliteRT*'s routing policy, PE π_k will be able to inject the last flit of packet p as soon as: 1) all flits previously pending in the FIFO queues of π_k have been injected and 2) there is one clock cycle where no packet from other PEs conflicts for the same output port than p. This happens as soon as the length of the interval is larger than the maximum number of pending flits that must be injected by π_k and the maximum number of flits generated by conflicting flows injected by other PEs that conflict to access the same output port during the interval.

The term referred to in point 1) is given by the maximum number of flits that may be generated by flows on π_k that are ahead of p in the FIFO queues of π_k. Since each flow

may have at most one packet in the FIFO queue at a time (see Section V-C), we have that the maximum number of flits ahead of p is strictly smaller than $\sum_{\forall f_l \in F_k} C_l$. Furthermore, according to Section V-C, low priority flows injected by π_k cannot block high priority ones. Therefore, if the priority prio_i of f_i is high, then strictly less than $\sum_{\forall f_l \in F_k \cap \text{hp}} C_l$ flits may be sent ahead of p in π_k. Summarizing, at most

$$\left(\sum_{\forall f_l \in \mathcal{I}_k} C_l \right) - 1 \tag{16}$$

flits generated by π_k (with $\mathcal{I}_k$ defined as in the claim) may interfere with the transmission of the last flit of the packet p under analysis.

To compute the term referred by point 2) in the explanation above, consider the flow f_j that may conflict with the packet p under analysis (i.e., $f_j \in \Gamma_k^C$). By definition of $n^j_{\text{hops}}(x_k, y_k)$, flits from f_j will reach the router (x_k, y_k) in no less than $n^j_{\text{hops}}(x_k, y_k)$ clock cycles. That is, the *last flit* generated by f_j that may conflict with the injection of p must have been injected in the NoC *no later* than $n^j_{\text{hops}}(x_k, y_k)$ clock cycles *before the end* of the period during which p is interfered with. Similarly, according to (7), flits from flow f_j will reach the router (x_k, y_k) in no more than $(n^j_{\text{hops}}(x_k, y_k) + n^j_{\text{def}}(x_k, y_k) \times c_{\text{def}})$ clock cycles. Thus, the *first flit* generated by f_j that may conflict with the injection of p must have been injected *no earlier* than $n^j_{\text{hops}}(x_k, y_k) + n^j_{\text{def}}(x_k, y_k) \times c_{\text{def}}$ clock cycles *before the start* of the interference with p. Therefore, the length of the interval during which f_j may inject flits that conflict with the packet p under analysis is given by $\Delta t = wcit_i + 1 - n_{\text{hops}}(x_k, y_k) + n_{\text{hops}}(x_k, y_k) + n^j_{\text{def}}(x_k, y_k) \times c_{\text{def}} = wcit_i + J_j$, where $J_j = n^j_{\text{def}}(x_k, y_k) \times c_{\text{def}}$ and $(wcit_i + 1)$ is the duration of the time interval starting when p is inserted in R_k's FIFO queue and finishing when the last flit of p is injected in the network.

As proven in Lemma 14, a flow f_j can inject at most $\lambda_j(\Delta t)$ packets in the network in any time interval of length Δt. Therefore, all the flows that may interfere with the injection of packet p can inject at most

$$\sum_{\forall f_j \in \Gamma_k^C} \lambda_j\big(wcit_i + J_j + 1\big) \tag{17}$$

flits that may conflict with p.

Combining (16) and (17), we get that p may inject its last flit as soon as

$$wcit_i + 1 \geq \left(\sum_{\forall f_l \in \mathcal{I}_k} C_l \right) - 1 + \sum_{\forall f_j \in \Gamma_k^C} \lambda_j\big(wcit_i + J_j + 1\big).$$

Hence proving the lemma. ∎

The set of flows Γ_k^C conflicting with the injection of a packet p at a router R_k is composed of all the flows injected by other PEs that may request the E or S port of R_k. That is

$$\Gamma_k^C = \Gamma_k^S \cup \Gamma_k^E \tag{18}$$

where Γ_k^S and Γ_k^E are the set of flows that may request the S and E port of R_k, respectively. We define Γ_k^S and Γ_k^E in Lemmas 18 and 19. However, to compute Γ_k^S and Γ_k^E, we must first define the set Γ_k^{def} of flows that may be deflected in router R_k. We further divide that set in $\Gamma_k^{\text{def\_hp}}$ and $\Gamma_k^{\text{def\_lp}}$ such that $\Gamma_k^{\text{def\_hp}}$ is the set of high priority flows that may be deflected in R_k, and $\Gamma_k^{\text{def\_lp}}$ is the set of low priority flows that may be deflected in R_k. By definition, $\Gamma_k^{\text{def}} = \Gamma_k^{\text{def\_hp}} \cup \Gamma_k^{\text{def\_lp}}$.

Lemma 16:

$$\Gamma_k^{\text{def\_hp}} = \begin{cases} \Gamma_k^{N \to S} \cap hp, & \text{if } \text{def}_k^{hp} = 1 \\ \emptyset, & \text{otherwise.} \end{cases}$$

Proof: According to HopliteRT*'s routing policy, only high priority flows entering by the N port and contending for the S port can be deflected (i.e., all flows in $\Gamma_k^{N \to S} \cap hp$). Therefore, $\Gamma_k^{\text{def\_hp}} = \Gamma_k^{N \to S} \cap hp$ when deflections may happen in router R_k (i.e., when $\text{def}_k^{hp} = 1$). If no deflection may happen in R_k (i.e., $\text{def}_k^{hp} = 0$), then the set of deflected flows in R_k is obviously empty. ∎

Lemma 17:

$$\Gamma_k^{\text{def\_lp}} = \begin{cases} \left\{ \Gamma_k^{W \to S} \cup \Gamma_k^{N \to S} \cup \Gamma_n^{\text{def}} \right\} \cap lp, & \text{if } \text{def}_k^{lp} = 1 \\ \emptyset, & \text{otherwise} \end{cases}$$

with R_n being the router directly north to R_k.

Proof: First, remember that according to HopliteRT*'s routing policy, any flow that is deflected in R_n (i.e., flows in Γ_n^{def}) will enter in R_k by the W port and compete for the S port. Furthermore, any low priority flow entering by the N or W port and contending for the S port can be deflected (i.e., all flows in $\{\Gamma_k^{W \to S} \cup \Gamma_k^{N \to S} \cup \Gamma_n^{\text{def}}\} \cap lp$). Therefore, $\Gamma_k^{\text{def\_hp}} = \{\Gamma_k^{W \to S} \cup \Gamma_k^{N \to S} \cup \Gamma_n^{\text{def}}\} \cap lp$ when deflections may happen in the router R_k (i.e., when $\text{def}_k^{lp} = 1$). If no deflection may happen in R_k (i.e., $\text{def}_k^{lp} = 0$), then the set of deflected flows in R_k is obviously empty. ∎

Now, we can define Γ_k^S and Γ_k^E.

Lemma 18: The set of flows coming from other routers that conflict on the S port of router R_k is given by $\Gamma_k^S = \Gamma_k^{N \to S} \cup \Gamma_k^{W \to S} \cup \Gamma_n^{\text{def}}$ where R_n is the router directly north to R_k.

Proof: The S port of the router R_k may be kept busy by any flow entering by the N or W port of R_k and requesting the S port. Since according to HopliteRT*'s routing policy, the only deflected flows that may enter by the W port of R_k and request the S port are those deflected in R_n (i.e., flows in Γ_n^{def}), the set of all flows that may request the S port of R_k is $\Gamma_k^{N \to S} \cup \Gamma_k^{W \to S} \cup \Gamma_n^{\text{def}}$ (where $\Gamma_k^{N \to S} \cup \Gamma_k^{W \to S}$ is the set of all flows that were not deflected that may enter R_k and request the S port). ∎

Lemma 19: The set of flows coming from other routers that conflict on the E port of the router R_k is given by $\Gamma_k^E = \Gamma_k^{W \to E} \cup \Gamma_{\text{ring}}^{\text{def}} \cup \Gamma_k^{W \to S} \cup \Gamma_n^{\text{def}}$, where R_n is the router directly north to R_k and

$$\Gamma_{\text{ring}}^{\text{def}} = \bigcup_{\forall R_l:\ (x_l > x_k \wedge y_l = y_k - 1) \vee (x_l < x_k \wedge y_l = y_k)} \left\{ \Gamma_l^{\text{def}} \right\}. \tag{19}$$

Proof: Let R_n be the router that is directly north to R_k [i.e., router at coordinates $(x_k, y_k - 1)$]. Then, according to HopliteRT*'s routing policy, the only deflected flows that may enter by the W port of R_k and request the E port are

TABLE III
RESOURCES UTILIZATION OF 8×8 NOCS IN MID-RANGE FPGA

NoC	LUTs	% Resource utilization of the platform
ProNoC	100000	20%-150%
IDAMC	83000	18%-127%
CONNECT	96000	20%-147%
HopliteRT*	5632	1.1%-8.5%

those that were deflected in routers located between R_n and R_k on the ring of the NoC [i.e., any router R_l such that $(x_l > x_k \wedge y_l = y_k - 1) \vee (x_l < x_k \wedge y_l = y_k)$]. The set of all those flows is provided by $\Gamma^{\text{def}}_{\text{ring}}$. Therefore, the E port of R_k may only be requested by the flows in $\Gamma^{\text{def}}_{\text{ring}} \cup \Gamma^{W \to E}_k$, where $\Gamma^{W \to E}_k$ is the set of flows that are not deflected in any router and request the E port of R_k (note that there is no flow that may enter by the N port and request the E port). Finally, the routing table of HopliteRT* (see Table II) does not allow a packet p to be injected by the PE toward the E port whenever there is packet entering R_k by the W port and requesting the S port. Thus, the set of all flows entering by the W port and requesting the S port (i.e., $\Gamma^{W \to S}_k \cup \Gamma^{\text{def}}_n$ as proven in Lemma 18) must be added to the set of conflicting flows, proving the lemma. ■

The results of Lemmas 18 and 19 can then be used to compute the WCIT using Lemma 15 and (18).

VII. EXPERIMENTAL RESULTS

A. Implementation of HopliteRT*

We implemented HopliteRT* with the hardware description language Verilog taking advantage of the possibility of fracturing the look-up tables in recent FPGA platforms. A 64-bits HopliteRT* router synthesized for a Xilinx Virtex-7 485T FPGA requires 88 LUTs and 139 Flip-Flops (FFs). It is only three additional LUTs (after fracturation) in comparison to HopliteRT. Note that a single HopliteRT* router requires only 0.03% and 0.02% of the total number of LUTs and FFs available in the Xilinx Virtex-7, respectively.

Next, we connected the router to a Microblaze soft core and synthesized a 4×4 network for a Virtex-7 485T using Xilinx Vivado. We computed the maximum operating frequency and obtained ≈275 MHz for both HopliteRT and HopliteRT*, thereby showing no degradation when adopting HopliteRT*.

Table III shows an approximation of the resource utilization for the implementation of HopliteRT*, ProNoC [30], IDAMC [5], and CONNECT [31] in a Xilinx Kirtex-7. Contrary to the Virtex-7 that is targeting rather high-end applications, the Xilinx Kirtex-7 is a mid-range product that exposes approximately between 65 600 and 477 760 LUTs. When we synthesized a single ProNoC router with two VCs (equivalent to two priorities), it required 1574 LUTs, and according to [31] and [32], a router of IDAMC requires ≈1300 LUTs, and one of CONNECT approximately 1500 LUTs. Thus, as reported in Table III, to implement an 8×8 ProNoC NoC, we need ≈100 000 LUTs (IDAMC: ≈83 000 LUTs and CONNECT: ≈96 000 LUTs), leaving very little space, if any, for the computation logic. Those NoCs are thus too expensive for such platforms. Conversely, an 8×8 HopliteRT* NoC consumes only 5632 LUTs, i.e., between 1.1% and 8.5% of the Xilinx Kirtex-7 resources. Therefore, HopliteRT* is a suitable solution for such FPGA platforms unlike virtual-channel-based NoCs.

B. Analyses Results

In this section, we provide experimental results by computing the WCTT and WCCT of sets of communication flows generated under different system configurations, (i.e., distinct NoC sizes, number of flows, and traffic patterns).

NoC Latency Bounds: We generated sets of flows according to two traffic patterns: 1) `random`: the origin and destination coordinates are randomly generated using a uniform probability distribution of a flow to originate and target any router in the network and 2) `all2one`: origins are randomly generated but the same destination coordinates are assigned to all the flows. A priority level (low or high) is randomly assigned to each flow. The number of high priority flows was roughly kept at 50% of the total number of flows in the network. The number of flits of packets released by a communication flow was randomly chosen between 1 and 5, and their interarrival times were generated as in [33].

In Fig. 5(a) and (b), we provide the results computed by using the analysis of HopliteRT [12], [29], the analysis presented in Section VI-A, and the improved analysis of Section VI-B. We show the evolution of the maximum and average packets WCTT for an increasing number of flows in a 16×16 network considering a `random` traffic pattern. Each point in the plot is the result of 100 experiments. We varied the number of generated flows from 10 to 300 by steps of 10.

The maximum WCTT observed over all flows for both HopliteRT and HopliteRT* is roughly the same. This can easily be explained by the fact that HopliteRT* has the same WCTT bound than HopliteRT for low priority flows. However, high priority packets see their WCTT drastically reduced in HopliteRT*. It is even more visible when looking at the results returned by the improved WCTT analysis, which reduces considerably the analysis pessimism by considering the actual set of flows that may interfere with each other. We also see that the improved analysis gain reduces as the number of flows increases. This is expected since the number of interfering flows and hence the number of deflections that flows may suffer increases when more packets are injected into the network. Even though, we show that the average WCTTs of high priority flows remain considerably lower than that of other flows. We also show that the modifications of HopliteRT introduced in this article allow at least a $2\times$ improvement on the worst and average case traversal time of high priority packets, without impacting the quality of service of low-priority packets, and up to $5\times$ improvement on the average traversal time when there are few flows. With a `all2one` traffic pattern (see [34, Fig. 7]), the improved WCTT bound of HopliteRT* returns the same result as the simple bound of Section VI-A. That can be explained by the fact that most packets compete for the same resources (i.e., links and routers output ports) since their destination is the same. Hence, the number of conflicting flows and the number of deflections increase considerably for all the packets in comparison to the random traffic pattern.

In Fig. 5(d) and (c), we show the average and maximum WCCT of a set of flows with the origin and destination

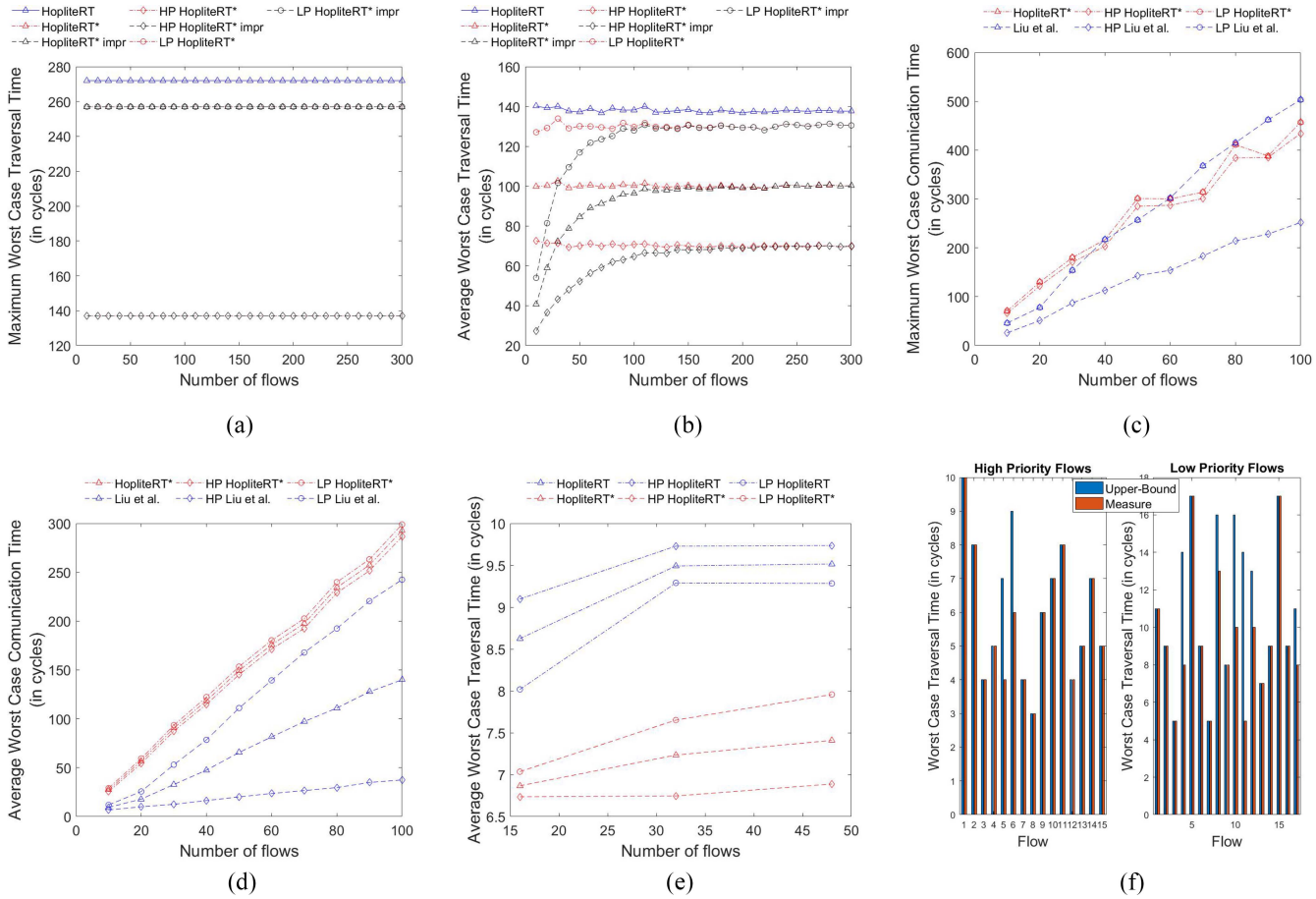

Fig. 5. Experimental results for a random traffic pattern. (a) Max WCTT 16 × 16 NoC. (b) Average WCTT 16 × 16 NoC. (c) Max WCCT 4 × 4 NoC. (d) Average WCCT 4 × 4 NoC. (e) Avg. measured WCTT 4 × 4 NoC. (f) Packets WCCT 4 × 4 NoC.

routers randomly chosen in a 4 × 4 NoC. Those results were obtained by using the improved analysis of HopliteRT* and that presented in [20] by Liu *et al.*, which is an improved analysis of that proposed in [33] and [35] by Shi and Burns. To establish a fair comparison, we assume two VCs (i.e., two priority levels) for the analysis presented in [20]. We observe that the analysis by Liu *et al.* performs better than that of HopliteRT* in terms of average and maximum worst-case communication time in most cases. We assume that it happens because [20] considers that each flow can only have one packet traversing the network at the same time, while HopliteRT* supports the transmission of multiple packets from the same flow simultaneously, leading to more possible contentions, as well as, more pessimism in the HopliteRT*'s WCIT analysis. However, we recall that a router similar to that assumed by [20] is likely 10–20 times more costly from a hardware viewpoint than a HopliteRT* router (see Table III).

We also use an autonomous vehicle application that has been studied in [6] and [20], to compare the improved analysis of HopliteRT* and that by Liu *et al.* The application is composed of multiple tasks that rely on 38 traffic flows to communicate. The application is mapped on 16 PE connected to a 4 × 4 NoC. The parameters of traffic flows and their origin and destination routers are kept the same as those used in [6]. The priority was assigned according to their period and deadline, that is, flows $f_8 - f_{30}$ were given the highest priority

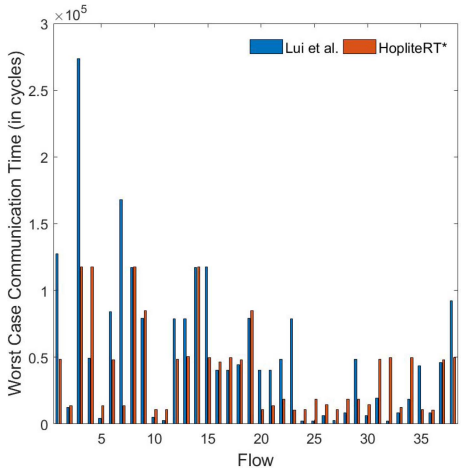

Fig. 6. Experimental results for the case study.

since their periods/deadlines are the shortest. In Fig. 6, we show that the WCCT of 14 flows is noticeably better with HopliteRT* in comparison to Liu *et al.*'s analysis [20]. The WCCT of 12 flows is noticeably better with Liu *et al.*'s analysis, and the last 12 flows have comparable results with both analyses. Therefore, we conclude that on a real use case, there is no analysis that dominates the other.

RTL Simulations: We performed cycle-accurate simulations of HopliteRT and HopliteRT* using HDL Verilog

implementations of 4×4 NoCs. We configured each PE to inject one, two, or three flows in the network. We generated flows with a random traffic pattern and random priorities. The interarrival times of flows were randomly chosen within the set $\{100, 200, \ldots, 1000\}$. The utilization of each PE was set at 20%, then the utilization of each flow from the same PE was generated using [36]. The number of flits in a flow is given by multiplying its utilization by its interarrival time. The flit size was set to 64 bits.

In Fig. 5(e), we provide the *measured* average WCTT for HopliteRT and HopliteRT*. We observe that packets reach their destination considerably faster with HopliteRT*. Note that the WCTT of high priority packets is higher than that measured for low priority ones in HopliteRT. That is due to HopliteRT not making any distinction between high and low flows. However, we show that by using HopliteRT*, the quality of service is guaranteed to flows of high priority, and hence their WCTT decreases. We provide the measured average WCIT and WCCT in [34, Fig. 8].

In Fig. 5(f), we present the measured average WCTT in HopliteRT* for a set of 32 flows against the improved version of the bound introduced in this article. In this experiment, each PE can inject packets from two different flows in the network. We observe that our approach provides safe and mostly tight upper bounds on the WCTT for high and low priority flows.

VIII. Conclusion

We presented HopliteRT*, a new NoC design with improved timing performances at a marginal increase of the hardware resource utilization in comparison to HopliteRT. The circulant topology adopted by HopliteRT* reduces the number of deflections and therefore, the WCTT of high priority packets. We identified an issue in the analysis of HopliteRT and proposed a new timing analysis for HopliteRT*. We also provided a complete implementation of HopliteRT* in HDL Verilog. Both cycle-accurate emulation of the NoC on a Xilinx7 FPGA and synthetic experiments show important performance improvements in comparison to the related work.

References

[1] L. Benini and G. De Micheli, "Networks on chip: A new paradigm for systems on chip design," in *Proc. Design Autom. Test Europe Conf. Exhibit.*, Mar. 2002, pp. 418–419.

[2] J. Henkel, W. Wolf, and S. Chakradhar, "On-chip networks: A scalable, communication-centric embedded system design paradigm," in *Proc. IEEE 17th Int. Conf. VLSI Design*, 2004, pp. 845–851.

[3] G.-G. Mplemenos and I. Papaefstathiou, "MPLEM: An 80-processor FPGA based multiprocessor system," in *Proc. IEEE 16th Int. Symp. Field Program. Custom Comput. Mach.*, 2008, pp. 273–274.

[4] D. M. Holman and D. C. S. Lee, "A survey of routing techniques in store-and-forward and wormhole interconnects," Sandia Nat. Lab., U.S. Dept. Energy, Washington, DC, USA, Rep. SAND2008-0068, 2008.

[5] S. Tobuschat, P. Axer, R. Ernst, and J. Diemer, "IDAMC: A NoC for mixed criticality systems," in *Proc. IEEE 19th Int. Conf. Embedded Real Time Comput. Syst. Appl.*, 2013, pp. 149–156.

[6] Z. Shi, A. Burns, and L. Indrusiak, "Schedulability analysis for real time on-chip communication with wormhole switching," in *Proc. IJERTCS*, 2010, pp. 1–22.

[7] Q. Xiong, Z. Lu, F. Wu, and C. Xie, "Real-time analysis for wormhole NoC: Revisited and revised," in *Proc. IEEE Int. Great Lakes Symp. VLSI*, 2016, pp. 75–80.

[8] Q. Xiong, F. Wu, Z. Lu, and C. Xie, "Extending real-time analysis for wormhole NoCs," *IEEE Trans. Comput.*, vol. 66, no. 9, pp. 1532–1546, Sep. 2017.

[9] L. S. Indrusiak, A. Burns, and B. Nikolić, "Buffer-aware bounds to multi-point progressive blocking in priority-preemptive NoCs," in *Proc. Design Autom. Test Europe Conf. Exhibit.*, 2018, pp. 219–224.

[10] B. Nikolić, S. Tobuschat, L. Soares, R. Ernst, and A. Burns, "Real-time analysis of priority-preemptive nocs with arbitrary buffer sizes and router delays," *Real Time Syst.*, vol. 55, no. 1, pp. 63–105, Jan. 2019.

[11] N. Kapre and J. Gray, "Hoplite: Building austere overlay NoCs for FPGAs," in *Proc. 25th Int. Conf. Field Program. Logic Appl.*, Sep. 2015, pp. 1–8.

[12] S. Wasly, R. Pellizzoni, and N. Kapre, "HopliteRT: An efficient FPGA NoC for real-time applications," in *Proc. Int. Conf. Field Program. Technol.*, Dec. 2017, pp. 64–71.

[13] T. Picornell, J. Flich, C. Hernández, and J. Duato, "DCFNoC: A delayed conflict-free time division multiplexing network on chip," in *Proc. 56th Annu. Design Autom. Conf.*, 2019, pp. 1–6.

[14] M. G. Alonso, J. Flich, M. Turki, and D. Bertozzi, "A low-latency and flexible TDM NoC for strong isolation in security-critical systems," in *Proc. IEEE 13th Int. Symp. Embedded Multicore Many Core Systems-on-Chip (MCSoC)*, 2019, pp. 149–156.

[15] J. Diemer, J. Rox, M. Negrean, S. Stein, and R. Ernst, "Real-time communication analysis for networks with two-stage arbitration," in *Proc. IEEE 9th ACM Int. Conf. Embedded Softw.*, 2011, pp. 243–252.

[16] E. A. Rambo and R. Ernst, "Worst-case communication time analysis of networks-on-chip with shared virtual channels," in *Proc. IEEE Design Autom. Test Europe Conf. Exhibit.*, 2015, pp. 537–542.

[17] B. D. De Dinechin, D. Van Amstel, M. Poulhiès, and G. Lager, "Time-critical computing on a single-chip massively parallel processor," in *Proc. IEEE Design Autom. Test Europe Conf. Exhibit.*, 2014, pp. 1–6.

[18] D. Wentzlaff *et al.*, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, Sep./Oct. 2007.

[19] Z. Shi and A. Burns, "Real-time communication analysis for on-chip networks with wormhole switching," in *Proc. 2nd ACM/IEEE Int. Symp. Netw. Chip*, 2008, pp. 161–170.

[20] M. Liu, M. Becker, M. Behnam, and T. Nolte, "Tighter time analysis for real-time traffic in on-chip networks with shared priorities," in *Proc. 10th IEEE/ACM Int. Symp. Netw. Chip*, 2016, pp. 1–8.

[21] B. Nikolić and S. M. Petters, "EDF as an arbitration policy for wormhole-switched priority-preemptive nocs-myth or fact?" in *Proc. Int. Conf. Embedded Softw.*, Oct 2014, pp. 1–10.

[22] A. Burns, J. Harbin, and L. S. Indrusiak, "A wormhole NoC protocol for mixed criticality systems," in *Proc. IEEE Real Time Syst. Symp.*, 2014, pp. 184–195.

[23] L. S. Indrusiak, J. Harbin, and A. Burns, "Average and worst-case latency improvements in mixed-criticality wormhole networks-on-chip," in *Proc. IEEE 27th Euromicro Conf. Real Time Syst.*, 2015, pp. 1–84.

[24] B. Nikolic, R. Hofmann, and R. Ernst, "Slot-based transmission protocol for real-time NoCs-SBT-NoC," in *Proc. 31st Euromicro Conf. Real Time Syst.*, 2019, pp. 1–22.

[25] F. Giroudot and A. Mifdaoui, "Buffer-aware worst-case timing analysis of wormhole NoCs using network calculus," in *Proc. IEEE Real Time Embedded Technol. Appl. Symp.*, 2018, pp. 37–48.

[26] F. Giroudot and A. Mifdaoui, "Tightness and computation assessment of worst-case delay bounds in wormhole networks-on-chip," in *Proc. 27th Int. Conf. Real Time Netw. Syst.*, 2019, pp. 19–29.

[27] N. Kapre and J. Gray, "Hoplite: A deflection-routed directional torus NoC for FPGAS," *ACM Trans. Reconfig. Technol. Syst.*, vol. 10, no. 2, pp. 1–24, 2017.

[28] T. Garg, S. Wasly, R. Pellizzoni, and N. Kapre, "HopliteBuf: FPGA NoCs with provably stall-free FIFOs," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, 2019, pp. 222–231.

[29] S. Wasly, R. Pellizzoni, and N. Kapre. (2017). *Worst Case Latency Analysis for Hoplite FPGA-Based NoC. UWSpace*. [Online]. Available: http://hdl.handle.net/10012/12600

[30] A. Monemi, J. Tang, M. Palesi, and M. N. Marsono, "ProNoC: A low latency network-on-chip based many-core system-on-chip prototyping platform," *Microprocessors Microsyst.*, vol. 54, pp. 60–74, Oct. 2017.

[31] M. K. Papamichael and J. C. Hoe, "CONNECT: Re-examining conventional wisdom for designing Nocs in the context of FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, 2012, pp. 37–46.

[32] S. Tobuschat, "Predictable and runtime-adaptable network-on-chip for mixed-critical real-time systems," Ph.D. dissertation, Faculty Elect. Eng. Inf. Technol. Phys., Techn. Univ. Braunschweig, Braunschweig, Germany, 2019.

[33] Z. Shi and A. Burns, "Real-time communication analysis with a priority share policy in on-chip networks," in *Proc. IEEE 21st Euromicro Conf. Real Time Syst.*, 2009, pp. 3–12.

[34] Y. Ribot González and G. Nelissen, "Hoplitert*: Real-time NoC for FPGA," CISTER Res. Centre, ISEP, Polytechnic Inst. Porto, Porto, Portugal, Rep. CISTER-TR-200702, 2020.

[35] Z. Shi and A. Burns, "Improvement of schedulability analysis with a priority share policy in on-chip networks," in *Proc. 17th Int. Conf. Real Time Netw. Syst. (RTNS)*, 2009, pp. 75–84.

[36] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real Time Syst.*, vol. 30, pp. 129–154, May 2005.