OPEN ACCESS

University of BRISTOL

Kerrison, S., May, D., & Eder, K. (2016). A Benes Based NoC Switching Architecture for Mixed Criticality Embedded Systems. In 2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC 2016). (pp. 125-132). Institute of Electrical and Electronics Engineers (IEEE). DOI: 10.1109/MCSoC.2016.50

Peer reviewed version

License (if available):
Unspecified

Link to published version (if available):
10.1109/MCSoC.2016.50

Link to publication record in Explore Bristol Research
PDF-document

University of Bristol - Explore Bristol Research
General rights

A Beneš Based NoC Switching Architecture for Mixed Criticality Embedded Systems

Steve Kerrison, David May and Kerstin Eder
University of Bristol, Department of Computer Science
Merchant Venturers Building, Woodland Road
Bristol, BS8 1UB, United Kingdom.
Email: {firstname.lastname}@bristol.ac.uk

Abstract—**Multi-core, Mixed Criticality Embedded (MCE) real-time systems require high timing precision and predictability to guarantee there will be no interference between tasks. These guarantees are necessary in application areas such as avionics and automotive, where task interference or missed deadlines could be catastrophic, and safety requirements are strict. In modern multi-core systems, the interconnect becomes a potential point of uncertainty, introducing major challenges in proving behaviour is always within specified constraints, limiting the means of growing system performance to add more tasks, or provide more computational resources to existing tasks.**

We present MCENoC, a Network-on-Chip (NoC) switching architecture that provides innovations to overcome this with predictable, formally verifiable timing behaviour that is consistent across the whole NoC. We show how the fundamental properties of Beneš networks benefit MCE applications and meet our architecture requirements. Using SystemVerilog Assertions (SVA), formal properties are defined that aid the refinement of the specification of the design as well as enabling the implementation to be exhaustively formally verified. We demonstrate the performance of the design in terms of size, throughput and predictability, and discuss the application level considerations needed to exploit this architecture.

I. INTRODUCTION

Real-time embedded systems perform a broad range of processing tasks, many of which must take place within hard deadlines in order to avoid loss of functionality or risks to safety. Modern embedded systems are prolific in many safety-critical areas, including automotive, industrial and avionics. Alongside safety, these sectors must deliver increasingly sophisticated features, such as visual processing and automation, requiring increases in computational capabilities and task count, without compromising safety.

This feature growth has necessitated multi-core solutions in-line with processor scaling trends [1]. However, the need to preserve hard-real time and safety requirements in critical tasks creates a unique problem set that must be handled at all levels, from architecture, through the software development process and up to certification. This is further confounded in large systems, where tasks with differing levels of criticality must co-exist [2].

This paper addresses the interconnect challenges of multi-core, MCE systems. We propose a NoC architecture that is designed to be both highly predictable in its routing and timing behaviour, as well as formally verifiable. In doing so, the architecture provides strong behavioural guarantees that allow

mixed-criticality tasks to be scheduled into a multi-core system. This aligns with efforts to provide determinism at higher levels, such as Time Triggered Ethernet [3] for distributed systems. We show that, despite aggressive timing precision requirements, the architecture is scalable and continues to be verifiable due to the simple structures from which it is built. The main contributions of this work are:

- Specification of a NoC that meets multi-core MCE needs.
- A novel, non-blocking and timing-predictable implementation addressing these requirements.
- Demonstration of scaling properties and performance.
- Formal verification to prove correct behaviour.

This paper is structured as follows. In Section II we discuss related work. Section III defines the requirements for the switching architecture. The implementation is explained in Section IV, then its performance evaluated in Section V. Software-level scheduling considerations are discussed in Section VI. Formal verification is presented in Section VII, including proof scaling results. Finally, Section VIII states our conclusions and proposes future work.

II. RELATED WORK

Addressing mixed-criticality multi-core communication builds upon several areas of research: multi-core systems; mixed criticality hardware and software along with its certification challenges; and network architectures, in particular NoC, to facilitate communication between large numbers of nodes in a single device.

A. Multi-core MCE systems

Traditional multi-tasking requires time-slicing of tasks onto a single processor. However, as processor operating frequencies tend to no longer increase, further performance is now more readily achieved through the addition of more processors [1]. The exchange of data between tasks is typically achieved using a shared-memory model, where locations in memory are accessed by multiple tasks. Alternatively, message passing may be used, adopting a model such as CSP-style communication. In either case, the movement of data between processing elements and memory has increasing complexity to ensure high performance and data consistency, although there are claims that this can continue to scale [4].

In a critical systems context, it is insufficient to provide increased performance through multi-core. Predictability must be preserved along with other protective measures to ensure deadlines are met and tasks do not interfere. For example, adding a task to an otherwise unused core in a system may not intuitively affect other tasks, but its network and memory access patterns may in fact do so. To guarantee safety, it must therefore be proven that this is not the case. Minimizing the effort required to do this, by ensuring the underlying architecture can provide the necessary behavioural guarantees, is then clearly desirable.

B. Mixed criticality hardware and software

A critical task has some function that either cannot be interrupted or has a hard deadline to maintain. In a less critical task, interruption to service or a deadline miss may be less important. For example, in an automotive context, the braking system must respond to any sensor reading that indicates a loss of traction under braking. If the response is delayed, or sensor data missed, safety of the vehicle may be negatively affected. However, an in-car entertainment system can tolerate occasionally dropped or corrupted video frames, to within some defined Quality-of-Service (QoS), and safety may be less of a concern.

Despite the clear differences in criticality in the above examples, there may be some interaction between the tasks, as the entertainment system may be integrated with the driver's controls and visual feedback. Therefore, these activities are not necessarily completely isolated, but the most critical task must clearly not be negatively affected by behaviour of the less critical one. This presents a challenge in meeting certification requirements, of which there may be several, whilst ensuring that resource allocation and scheduling allows all tasks to operate correctly in the broadest set of conditions [5].

Hardware must support mixed-criticality, first by providing sufficient resources that guarantee hard real-time behaviour. In this case, predictable timing is essential. Real-time processors often sacrifice performance enhancements such as cache-hierarchies in order to deliver such predictability. Secondly, hardware must prevent tasks from interfering with each other, for example by writing to incorrect memory regions or by creating resource starvation.

The software must be suitably predictable to ensure that underlying hardware guarantees can be provably exploited. Worst Case Execution Time (WCET) [6] analysis may be performed on a task to ensure that it completes within a desired deadline. In a system of tasks, scheduling must take place, either online or offline. Multi-core MCE systems pose additional problems over single-core solutions. In a shared memory system, there may be contention for access to the memory hierarchy or other interconnects [2]. These may reduce predictability and require more complex scheduling efforts or over-provisioning of resources to guarantee safe behaviour. This exposes a need to provide more predictability within the interconnect, to tighten these bounds, aiding analysis and certification.

C. NoC architectures

A Network-on-Chip (NoC) is a collection of resources on a single chip, typically including processing elements (cores), memory elements (caches, DRAMs) and peripheral components (external interfaces, timers, etc). These are all interconnected by one or more networks, departing from traditional bus architectures in favour of a more scalable, routed arrangement. The intent of NoC is to deliver a chip with processing elements numbering in the tens, hundreds or more, thus achieving continued performance growth through many-core scaling. In a multi-core MCE context, NoC is therefore highly desirable in emerging chip designs, but one that must meet the constraints outlined earlier in this section.

A variety of NoC topologies exist. Common approaches include mesh structures, rings and higher-dimension extensions of these. The Xeon Phi [7] uses a ring network to connect its 63 processing elements together along with tag caches and memory controllers. Progress into multi-layer 3D stacked processors [8] is extending NoC requirements into three physical dimensions.

Multiple mesh networks can be used to handle the different traffic patterns and access requirements, for example physically separating network memory accesses, core-to-core exchange and peripheral communication. Such approaches are seen in the five-network TILE64 [9] and three-network Adapteva Epiphany [10]. Alternatively, the network may be segmented through Time-Division Multiplexing (TDM), such as in the picoChip [11] processor, where bandwidth is guaranteed at pre-defined times, but may be over-provisioned.

Due to the size of NoC systems, congestion handling and fault tolerance must also be considered. Groups of cores may have more active communication, and thus benefit from using additional nearby routes. Failures may require message re-transmission [12], re-routing, or be mitigated through redundant hardware [13]. Dynamic mechanisms for routing, particularly unconstrained or non-deterministic, can be undesirable in a critical systems context because they reduce the certainty of timing in message delivery. Further, any network where messages may be blocked by other messages can dramatically increase the upper bound for communication time, resulting in an infeasible system specification or conditions that are unsafe for critical tasks.

D. Clos and Beneš networks

Large scale interconnectivity has previously been addressed in telecommunications, where circuits must be formed between telephone endpoints, focused through potentially several telephone exchanges locally, nationally or internationally. It was required to develop interconnects that did not place infeasible wiring overheads or switch complexity upon the exchanges. Clos [14] described a multi-stage network of smaller crossbar switches, layered in order to provide $N : N$ connectivity with lower wiring complexity than a monolithic solution. Timing changes or re-routing artefacts such as clicking are easily detected by the human ear, hence such networks must minimize
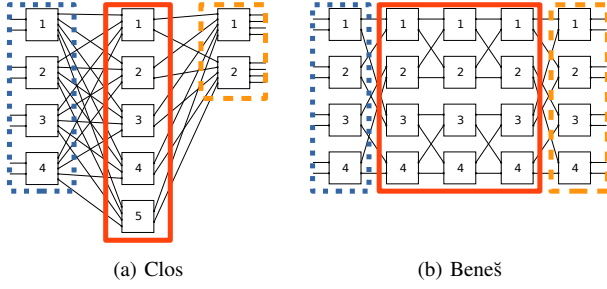
Fig. 1. Example of eight-port Clos and Beneš network structures with equivalent sections highlighted.

Fig. 2. Top-level signalling of the design, valid as a per-switch or whole-network abstraction.

or completely remove such occurrences. This strictness bears many similarities to the requirements of modern MCE systems.

Figure 1a depicts a Clos topology. It comprises three switching stages where the degree of the switches different at each stage. A Beneš [15] network refines the Clos concept into a topology of 2x2 switching elements. It retains the same properties, but is constructed from the smallest possible switch size, shown in Fig. 1b. Any larger switch in a Clos or Beneš network can be realised using sub-networks of Beneš switches.

For our Mixed Critcality Embedded Network-on-Chip (MCENoC), particular properties provided by these networks are desirable for several reasons:

1) The number of routing stages for any communication is consistent.
2) It is proven that any $N : N$ permutation of two-party communications is always routable in Beneš structures [15], [16].
3) The number of stages and switching elements used scales logarithmically with the number of nodes.
4) The switching elements can be built upon simple crossbars, which are straightforward to create in VLSI [17] and are then replicated many times.

Many variations on these networks exist, as well as strategies for routing on them [16]. However, we focus on preserving predictability, whilst keeping the core design simple and scalable, therefore the MCENoC design choices reflect this.

III. REQUIREMENTS

In this section we define high-level requirements for the MCENoC design. These are refined into a specification that guides the implementation. Both the specification and implementation are subjected to scrutiny through the formal verification process detailed in Section VII.

The aim of this work is to provide a communication architecture that enables real-time embedded software to have precise behavioural guarantees in a multi-core context. With this, the behaviour of the entire system can be reasoned about statically, thus allowing verification and certification of applications using the system. To focus this effort, several requirements are specified.

All nodes in the network can be considered equidistant: Predictability is simplified significantly when communication
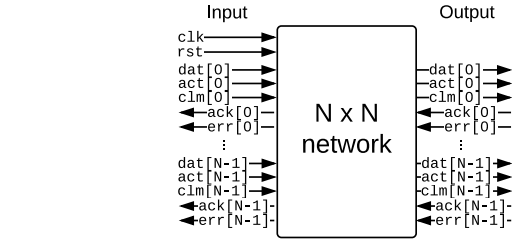
between nodes is a uniform distance, as placement of computation and data is less constrained.

Latency through the network is tightly bounded: Both worst- and best-cases should be known to provide guarantees of expected performance and satisfy task deadlines.

Communication between two nodes is non-blocking: Refining the previous requirement, by removing blocking, the flow of data through the network is simpler to model, analyse and produce bounds for. Buffering and flow control may still be present in network endpoints, however.

Invalid or erroneous communications cannot interfere with more critical communications: Where multiple flows exist within the network, those attached to critical tasks must have priority in the network. Less critical tasks, which may have less strict certification and verification requirements, must not be able to steal routes required by higher criticality tasks, and a failure condition should be asserted if this is attempted.

The switching architecture should scale into many cores: The target is a design that can ultimately scale up to hundreds of cores or more. Thus, the logic utilisation and internal connectivity scaling should support this.

Routing decisions must be statically resolvable with minimal overhead at large scales: All online control should be conducted in-band, without a central arbiter of other similar global control mechanism. A large amount of uncertainty can then be removed from the system's communication profile.

IV. MCENoC ARCHITECTURE

The MCENoC architecture is constructed from a replicated set of relatively simple switches, forming a Beneš network. The edges of the network then provide an interface to nodes, which can be processing elements, memory, or other peripherals. Interface bridging is expected to take place at he edge, with certain system level protocol constraints imposed by the network. The design is implemented in SystemVerilog 2009 and includes a number of SystemVerilog Assertion (SVA) [18] properties for validation of the specification and verification of the design. In this section we first address the switch implementation, then the network construction, followed by system level considerations.

A. Switching element

Each switching element is configurable to have 2^p ports, where p is the number of bits required to define a route

through the switch. A block-level diagram of signals is shown in Fig. 2, with an arbitrary number of ports with signals denoted as $\mathtt{sig}_0 \ldots \mathtt{sig}_{N-1}$. Each port has both forward- and backward-propagated signals, allowing the transfer of data, flow control and error condition between network stages, as well as permitting an idle state.

With no allocated ports, all outputs are set to their defaults and there is scope for fine-grained clock gating. Allocation of a port q is achieved by clocking in p configuration bits into $\mathtt{dat}_q$ whilst asserting the $\mathtt{clm}_q$ and $\mathtt{act}_q$. The configuration bits specify the target port number, r. Following this, input q is connected to output r and propagation of future signals takes place, through a buffer of depth one.

1) Conflict identification and resolution: An output r can only be successfully allocated if no other input port is already using it. If the requested output port is not in an idle state, then a conflict is present, where the port is either already claimed, or multiple inputs are simultaneously attempting to claim it. In the former case, $\mathtt{err}_q$ is asserted until $\mathtt{clm}_q$ is de-asserted, after which a new attempt at routing can begin. In the latter case, the input port with the lowest value of q obtains the route, with all other claims being rejected.

2) Port states: The switch input ports each have four possible states.

Wait: The input is not connected to any output and fewer than p configuration bits have been provided.

Accept: The input is connected to the requested output and is propagating data.

Reject: There were p configuration bits received, but a conflict or protocol violation was observed and the port must be unclaimed before attempting configuration again. No output connection was made.

Abort: A previously accepted connection is being destroyed due to an incoming error condition from the forward switching stage.

If all input ports are in the *wait* state and no $\mathtt{clm}$ signals are asserted, the switch can be considered idle and may be clock-gated. Successful connections can be destroyed either from the initiator, by de-asserting the relevant $\mathtt{clm}$ signal, or from the destination, by asserting $\mathtt{err}$.

B. Network construction

To determine the network construction for an N node network, a switching element size, B, must be selected. If raising B to a positive integer value (in the set of natural numbers, $\mathbb{N}$) yields N, then the number of stages is:

$$S = 2\log_B(N) - 1, \quad \text{if} \quad \exists x \in \mathbb{N} \mid B^x = N. \tag{1}$$

Otherwise, any $N = 2^x$ port network, where $B \geq 2$, has a middle stage with m directional bits (2^m ports per switch), where $2^m \leq B$, such that:

$$X = \lceil \log_B(N) \rceil, \tag{2a}$$
$$S = 2\log_B(X) - 1, \tag{2b}$$
$$m = \log_2\left(\frac{N}{B^{X-1}}\right). \tag{2c}$$

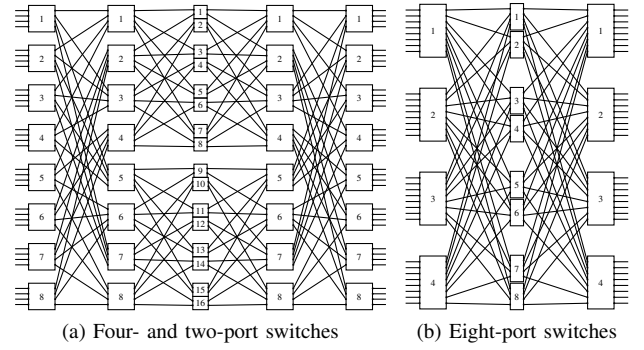(a) Four- and two-port switches   (b) Eight-port switches

Fig. 3. Two 32-port networks using four- and two-port switches vs. eight- and four-port switches.

For example, if the desired switch element size is four ports, then a 32-port network will contain four 4-port stages and a middle stage of twice as many 2-port switches, whereas a 64-port network is simply five 2-port stages. Two 32-port network examples are shown in Fig. 3. All network diagrams are generated in TikZ that is emitted from SystemVerilog during the $\mathtt{initial}$ phase of simulation, giving a direct representation of the structure that is in use.

The connectivity of ports in the network, c_n^i, of index i and at each stage n is shown in Eq. (3). Due to the symmetry of the network, it can be considered as two halves connected in a similar progression, where $n = 0$ is the middle stage. The ith port in the inner stage connects to the jth port in the next stage, wrapped in accordance to the block size at that stage. The block size, b_n, is determined by the stage, n, where the total number of connection stages, s, for half of the network, is obtained using S. The connectivity formula accepts any power-of-two size of switching element, B.

$$c_{n+1}^j \longleftrightarrow c_n^i, \quad \text{where:} \tag{3a}$$
$$j = \left(k + \left\lfloor \frac{k}{b_n} \right\rfloor \right) \bmod b_n + o, \tag{3b}$$
$$k = (i - o) \cdot B, \quad o = \left\lfloor \frac{i}{b_n} \cdot b_n \right\rfloor, \tag{3c}$$
$$b_n = \min\left(B^{2+s-(s-n)}, N\right), \quad s = \frac{S-1}{2}. \tag{3d}$$

The network can be considered folded horizontally, where input port q and output port r belong to the same node.

The topology of the network guarantees that on an s stage network requiring p routing bits, a route can be established $p + s$ cycles. Following this, an input can propagate through the network in p cycles. In the case where a route conflict occurs, the worst-case latency in the input receiving an $\mathtt{err}$ signal is $2p + s$, where the conflict occurs in the final stage and the error must propagate back through all prior stages.

C. System implementation

There is redundancy present in the routing configuration due to the in-band control that is used. If a two-port switching element receives a route configuration on one input, the other input could be implicitly routed to the other available output. However, we choose to require the redundant configuration, to make route setup time consistent across all inputs, as well
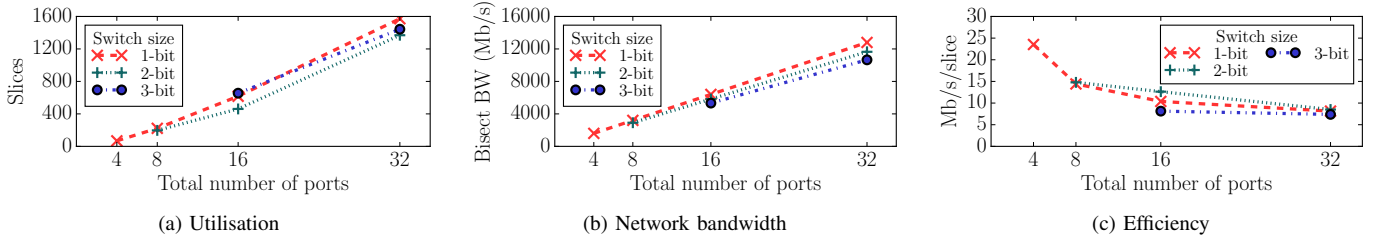
Fig. 4. Scaling behaviour of MCENoC when synthesized to FPGA target.

as to allow early identification of an erroneous route request. If implicit routing was possible, an unexpected route will be identified in *at least* $2s$ cycles, whereas with redundant route configuration, it is *at most* $2p + s$ cycles.

Although no blocking takes place within the network, it can be imposed by the connected nodes. The `cts` signal signifies that the source is clear to send, and is asserted by default during route setup, allowing the route configuration bits to be loaded into the network. Upon completing the route setup, `cts` is then controlled by the destination node. If the destination cannot receive more data, it must de-assert `cts`, and the source must not transmit more data until it is re-asserted.

To accommodate transmission latency, the receiver should have a buffer of at least $2s$ bits in size, and de-assert `cts` if less than this is available. This prevents loss of data. If data production and consumption rates can be determined statically, as may be the case for certain tasks in a real-time context, then it may be possible to ignore the flow control signals, as the switching element behaviour is not affected by `cts`.

V. SCALING AND PERFORMANCE

In this section we illustrate the scaling properties of the MCENoC, examine logic utilisation when various configurations are synthesised to FPGA, and provide a comparison and evaluation against other designs.

The current design can operate at up to $400 \,\mathrm{MHz}$ according to the Xilinx Vivado timing reporting tools, with the worst-case being $333 \,\mathrm{MHz}$. The size of the switching element governs the achievable clock speed, rather than the network size, although a fully integrated system may have other effects upon timing.

The architecture's performance and resource utilisation scaling are shown in Fig. 4. The FPGA slice utilisation (Fig. 4a) scales superlinearly, but to a manageable degree for the network sizes testing. Bisection bandwidth (Fig. 4b) scales linearly with the number of nodes, which is expected given that the operating frequency is fixed for each switching element size. Consolidating these two metrics into a measure of efficiency — $\mathrm{Mbit/s}$ per slice, shown in Fig. 4c — indicates that the two-bit switching element is marginally preferable over others for the network sizes that were synthesised.

To reason about integration with computational IP, we consider the Microblaze core. Its device utilisation is dependent upon configuration, however an example small configuration can use less than 800 slices[1] in the Xilinx v7 architecture.

[1]Evident through public discussion: http://tinyurl.com/j35p7ld

This indicates that, conservatively speaking, a sixteen-core Microblaze system and MCENoC network is possible on the Kintex-7K160T that we are using as our current target, which has $25 \,\mathrm{k}$-slices available.

The bisection bandwidth, B of the MCENoC can be calculated through the simple equation Eq. (4), where w is the bit-width of data transfer, f is the operating frequency and n is the number of ports. The current design assumes each port uses 1-bit serialised data. Therefore, at $364 \,\mathrm{MHz}$ (the operating frequency of our 2-bit switch implementation), an eight-node MCENoC provides a bisection bandwidth of $2.9 \,\mathrm{Gbit/s}$. A 32-node system achieves in excess of $11.6 \,\mathrm{Gbit/s}$.

$$B = f \times w \times n \quad \mathrm{bit/s}. \tag{4}$$

As an example comparison, the Epiphany E64G401 [10] has $102 \,\mathrm{GB/s}$ bisection bandwidth in an ASIC that features double the node count, 8-bit data width and three separate networks compared to our MCENoC example. On a per-node, per-bit-width basis, the E64G401's network achieves $199 \,\mathrm{Mbit/s}$ vs. the 32-node MCENoC's $725 \,\mathrm{Mbit/s}$. More in-depth evaluation would be better served by application-specific case studies.

VI. ROUTING AND SCHEDULING

It is proven in [15] that a route exists for any permutation of $1:1$ communication between all nodes in a Beneš network. However, calculating the routes for a permutation without global knowledge is non-trivial [16]. For the MCENoC, we assume that for pre-defined tasks, routes are determined statically, thus online methods are not required. There are still design decisions to make, in the form of switch size selection, as well as scheduling strategies. This section investigates the implications of the network design with respect to these issues.

A. Network equivalence

Routing algorithms such as [19] assume two-port switches as per a traditional Beneš network, where our header-based configuration requires one bit per switch. However, networks with higher degree switching elements can be equivalent, simply consuming a larger number of header bits. Figure 5 shows an example 8-port permutation in two network variants. Taking a single route from this — connecting port 0 to port 1 — the header bits are `10001`. In Fig. 5a, a single bit is used, left to right, to configure each stage. In Fig. 5b, two bits are used in the outer stages and a single bit in the middle stage, interpreted as `10-0-01`, giving the same connection.
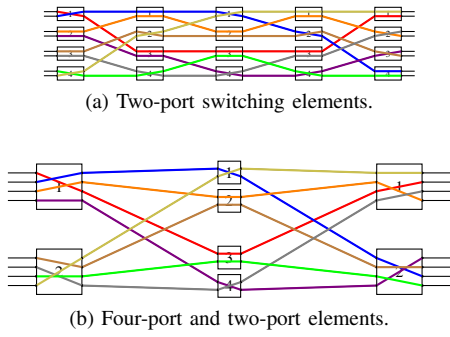
(a) Two-port switching elements.

(b) Four-port and two-port elements.

Fig. 5. Route equivalence in two eight-node MCENoC implementations.

Fig. 6. Time taken to cycle through N communications in an N-node network at $364\,\mathrm{Mbit/s/port}$, considering a desired payload efficiency.

B. Scheduling

In a practical system, a single routing permutation will be insufficient, as each node may have several other nodes with which it needs to communicate. This can be resolved through Time Division Multiplexing (TDM) of permutations. We assume that the nodes are responsible for ensuring the time divisions are adhered to, keeping the switching elements and network control simple, but requiring tightly synchronised timing between nodes. Such synchronisation is tractable due to the predictable latency of the network allowing precise time exchange between nodes. This can take place periodically, or be embedded into each permutation.

A Beneš style network combined with TDM abstracts other topologies well. For example, to emulate a 2D mesh, the MCENoC network can be divided into four permutations, with each permutation representing one of four directions. Adding dimensions requires two additional permutations. A broadcast from a single node can reach all other nodes in $\log_2(n+1)$ permutations as the number of nodes that can forward the message doubles with each permutation.

Applying TDM to the network increases the route calculation time linear to the number of permutations required. The low overhead of the in-band route setup enables low latencies, even between instances of a particular permutation where a large number of permutations are used. The duration of each permutation can be adjusted depending on the payloads that need to be transmitted. An example performance profile of a worst-case TDM schedule in which all nodes talk directly to all other nodes is shown in Fig. 6.

This assumes N TDM phases are needed, are of equal duration, and have a set efficiency achieve by sizing the payload appropriately in relation to the route setup and tear-down latency. Applying this conservative model gives a worst case time between cycles of the TDM of $914\,\mu\mathrm{s}$ for 128 cores with $99\,\%$ payload efficiency. In a $65\,\mathrm{k}$-node system this reaches $1.12\,\mathrm{s}$. However, the time per message is $17.03\,\mu\mathrm{s}$. Therefore, with repetition of routes within the TDM cycle, the latency between critical communications can be kept very low.

The underlying switching architecture is sufficiently flexible and predictable that scheduling even complex communication patterns is tractable. Low latency and predictability is favour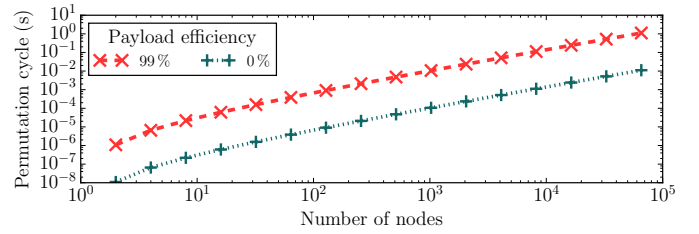ed over high bandwidth, although the network utilisation can be kept close to $100\,\%$ if the number of permutations and arrangement of communications is carefully selected.

To avoid the need for all traffic to be statically scheduled, a dynamic system can be used during pre-allocated slack phases in the network. In [20], a resource controller grants network time to requesting nodes, managing available slack in the system's mesh network. For strict and unimpeded control, separate control and data networks can be used. A similar approach can also be applied to the MCENoC, using TDM to provide this separation.

VII. FORMAL VERIFICATION

Formal methods of verification [21] allow properties of a design to be specified and proven exhaustively, covering all parts of the design's state-space that could be affected by that particular property. This is achieved with the use of formal verification tools (in the case of this paper, JasperGold), which can prove or disprove these properties, providing counterexample traces in the case of failure. This has benefits over traditional, test-driven verification, in that it does not rely on random or pre-defined test vectors to expose design flaws or implementation faults.

Complete verification coverage requires that properties describe all aspects of the design. Further, the exhaustive nature of formal verification means that complex properties and behaviours can lead to infeasibly long proof times.

The construction of formal properties also serves as a test of the specification, requiring it to be formalised in a syntax more rigid than natural language. Thus, properties can expose deficiencies in an implementation, the way in which the specification is defined, as well as its interpretation.

In formal verification terminology, a *property* is a formal description of the behaviour of a device over a period of time. This is typically formulated as a sequence of preconditions, which if met, a postcondition sequence must occur. If a property is *asserted*, then a verification environment (formal or simulation-based) will trigger a failure if the property is ever found not to hold. If a property is *assumed*, then it constrains the verification to consider only states where it holds true.

The MCENoC is designed with formal verification as the intended verification method. Given that is is constructed from simple, replicated elements, a large portion of the proof process is possible within seconds. In the rest of this section
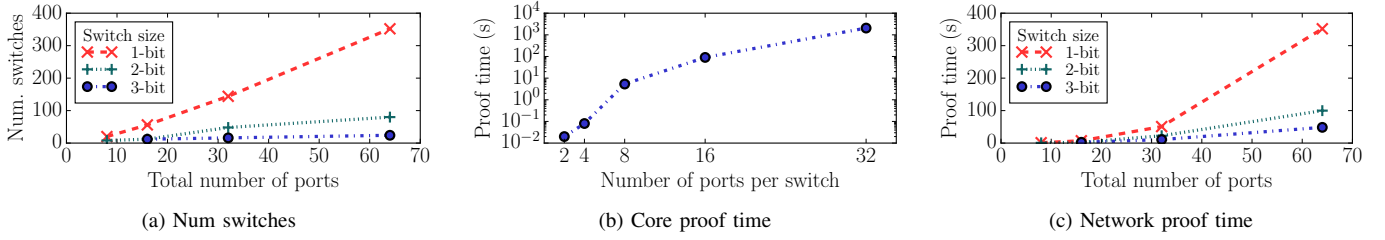
Fig. 7. Performance and scaling of the MCENoC formal verification

TABLE I
EXAMPLES OF SPECIFICATION CRITERIA AT EACH DESIGN LEVEL.

ID	Description	Property
C1	No two active inputs can share the same output channel.	no_shared_direction
C15	A port rejects further data in the event of an inbound error signal and propagates the error signal.	reject_on_err
N4	All routes through the network connect to the expected destination if no routing conflict arises and the target port does not assert an error.	route_correct
S4	Higher priority tasks must create their routes before lower priority tasks.	N/A

we examine the steps necessary for formal proof of MCENoC, as well as the performance scaling of the proof process at various system levels and sizes.

A. Specification

To form the specification of the design, we take the general requirements from Section III and produce more specific criteria. These are classified at several levels:

C: Core switching element level. The behaviour of a single switch and its interfaces.

N: Network level. Connectivity to the edge of the Beneš network, signal propagation and routing behaviour within.

S: System level. Behaviours when interacting with nodes.

Example specification criteria are given in Table I. At the core and network level, all specification items need to have one or more properties that fully describe the relevant behaviour of the MCENoC. However, the system level includes considerations outside of the scope of the MCENoC itself. It is still possible to define properties at the system level, but it may not be possible to verify them formally when solely examining the MCENoC implementation. However, these properties can still be used as assumptions that assist in the proof of other, lower level specification items.

B. Property definition

Properties are defined in SVA [18], and then asserted for each of the ports or instances defined in any configuration of the system. Checking of these properties is performed using the Cadence JasperGold formal verification tool.

Listing 1 gives an example property for a core switching element, seeking to prove that specification criteria C15 is met. This criteria refers to the rejection of further data if an error signal is received, and that the error signal propagates. Thus, C15 ensures that on error, a switch stops forwarding

on the connected port, but communicates the error backwards through the established route.

The property is defined as `reject_on_err(i)` where `i` is the port number. The portion before the `|=>` symbol signifies the pre-condition, stating that the port is connected and that the connected error signal has risen in the current clock cycle. Checking of the post-condition begins one cycle later. This is a two-cycle sequence, where first the port is expected to enter the ABORT state, which should include the error output being asserted downstream. After another cycle (signified by `##1`), the port's forward connections should all be de-asserted, releasing the forward connection.

Listing 1 Property example to check criterion C15

```
property reject_on_err(i);
    portstate[i] == ACCEPT and
    $rose(ports.err_in[direction[i]])
  |=>
    portstate[i] == ABORT and ports.err_out[i]
    ##1 !(ports.clm_out[direction[i]] |
      ports.act_out[direction[i]] |
      ports.dat_out[direction[i]] );
endproperty
```

This property can only fail if the pre-condition is met and then the post-condition sequence contradicts the definition. A formal verification tool can provide coverage in both respects, first demonstrating that the pre-condition is reachable, and then proving exhaustively that, under any condition where the pre-condition is reached, the post-condition sequence holds true.

C. Proof performance

The performance of the verification process was conducted on a dual-socket Intel Xeon X5460 server. Default JasperGold prover settings were used. Two levels of verification were tested: a single switching element and a full network. Results are shown in Fig. 7.

Figure 7a demonstrates the scaling of the number of switches with network size, depending on the size of each switching element. Figure 7b shows the proof time for core switching element properties. The proof time grows exponentially with the port count, which is an expected outcome as the number of generated assertions increases, as does the internal state of the switch. This demonstrates that the proof effort is tractable for sizes that are desirable for synthesis.

In Fig. 7c, the full network proof time is demonstrated. This includes network level properties as well as re-proof of switch-level properties. The majority of the proof burden is dependent upon the number of switches, hence using higher

order switches reduces the overall proof time. However, these results omit the checking of `route_correct` (specification N4 in Table I), which on its own requires approximately 90 minutes to prove on an 8-port network of 1-bit switches. This is due to the length of the pre-condition sequence it uses, which in many cases does not reach completion before some other correct behaviour (such as a routing conflict) takes place. To overcome this, one can rely on the switch-level properties combined with checking of the connectivity, which is a static property, to achieve the same guarantee. Alternatively, properties defining specific routing schedules could be defined, for example constrained to a particular application definition, in order to provide a smaller search space to the prover.

In summary, our performance evaluation demonstrates that formal proof of the MCENoC design that is at a scale practical for realisation in hardware, is tractable and timely using current tools. Potential performance issues have been exposed, and mitigation methods recommended.

VIII. Conclusions

A novel NoC implementation has been presented for multi-core MCE systems that addresses the problem of predictability in systems of growing scale and complexity. In answer to the requirements we define for such systems, this work uses Beneš-type network structures to achieve equidistant communication between nodes that remains scalable, providing an implementation that can be in-band controlled and has clearly defined route creation priorities. The network structure and switch behaviour can be combined with static scheduling and time division multiplexing to provide predictable low-latency communication that guarantees deadlines will not be missed and that critical communications are not interfered with, aiding software-level timing analysis and system certification.

The design is formally verified, providing exhaustive proof that specified behaviours hold, ensuring both that the specification is well defined and that the implementation is correct. Current proof tools can verify designs of sizes that are practical for synthesis to FPGA, whereupon current synthesis results indicate a typical $364\,\mathrm{MHz}$ operation and a bisection bandwidth of $11.6\,\mathrm{Gbit/s}$ for an 32×32 size network.

Future work

Network nodes, in the form of memories, processing and peripherals, may be integrated with the design to demonstrate it at the system level. This can include synthesis to FPGA as well as fully-simulated, potentially relying on the proofs made to simplify the network simulation process. The data-path of the design can also be expanded from 1-bit in order to explore the trade-off between bandwidth and resource utilisation, including FPGA slices and I/O.

Proof of properties at the system level, where node and software behaviour must be considered, is an important next step in exploiting the predictable nature of the MCENoC. This may extend beyond the SystemVerilog of the MCENoC, however, requiring innovative proof techniques.

At the software and tool-chain levels, the implementation of routing algorithms for required communication patterns is the subject of ongoing work. Routing must be reconciled with task priorities and network utilisation through time division multiplexing as discussed in Section VI, which can be evaluated through MCE application case studies.

References

[1] A. B. Kahng, "The ITRS design technology and system drivers roadmap," in *Proceedings of the 50th Annual Design Automation Conference on - DAC '13*. New York, New York, USA: ACM Press, 2013.

[2] J. Nowotsch and M. Paulitsch, "Leveraging Multi-core Computing Architectures in Avionics," in *2012 Ninth European Dependable Computing Conference*. IEEE, may 2012, pp. 132–143.

[3] H. Kopetz, "The Rationale for Time-Triggered Ethernet," in *2008 Real-Time Systems Symposium*. IEEE, nov 2008, pp. 3–11.

[4] M. M. K. Martin, M. D. Hill *et al.*, "Why on-chip cache coherence is here to stay," *Communications of the ACM*, vol. 55, p. 78, jul 2012.

[5] S. Baruah, H. Li *et al.*, "Towards the Design of Certifiable Mixed-criticality Systems," in *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, apr 2010, pp. 13–22.

[6] R. Wilhelm, J. Engblom *et al.*, "The worst-case execution-time problem - overview of methods and survey of tools," *ACM Trans. Embedded Comput. Syst.*, vol. 7, 2008.

[7] Intel Corporation, "Intel Xeon Phi Coprocessor," Intel Corporation, Tech. Rep., 2013.

[8] M. M. Sabry Aly, M. Gao *et al.*, "Energy-Efficient Abundant-Data Computing: The N3XT 1,000x," *Computer*, vol. 48, pp. 24–33, dec 2015.

[9] S. Bell, B. Edwards *et al.*, "TILE64 processor: A 64-core SoC with mesh interconnect," in *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*, vol. 51, 2008.

[10] Adapteva, "E64G401 EPIPHANY 64-core microprocessor datasheet."

[11] A. Duller, G. Panesar *et al.*, "Parallel Processing the picoChip way," *Communicating Processing Architectures*, pp. 299–312, 2003.

[12] R. Abdel-khalek and V. Bertacco, "Correct Runtime Operation for NoCs through Adaptive-Region Protection," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2016*, 2016.

[13] M. Ebrahimi, M. Daneshtalab *et al.*, "Fault-tolerant method with distributed monitoring and management technique for 3D stacked meshes," in *The 17th CSI International Symposium on Computer Architecture & Digital Systems (CADS 2013)*. IEEE, oct 2013, pp. 93–98.

[14] C. Clos, "A Study of Non-Blocking Switching Networks," *Bell System Technical Journal*, pp. 406–424, 1952.

[15] V. E. Beneš, "On Rearrangeable Three-Stage Connecting Networks," *Bell System Technical Journal*, vol. 41, pp. 1481–1492, sep 1962.

[16] C. Scheideler, *Universal Routing Strategies for Interconnection Networks*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, vol. 1390.

[17] Y. Jiang and M. Yang, "On circuit design of on-chip non-blocking interconnection networks," in *2014 27th IEEE International System-on-Chip Conference (SOCC)*, vol. 40, no. 8. IEEE, sep 2014, pp. 192–197.

[18] A. B. Mehta, *SystemVerilog Assertions and Functional Coverage: Guide to Language, Methodology and Applications*. New York, NY: Springer New York, 2014, ch. System Verilog Assertions, pp. 9–28.

[19] A. Waksman, "A Permutation Network," *Journal of the ACM*, vol. 15, pp. 159–163, jan 1968.

[20] A. Kostrzewa, S. Saidi *et al.*, "Slack-Based Resource Arbitration for Real-Time," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2016*, 2016, pp. 1012–1017.

[21] E. Seligman, T. Schubert *et al.*, *Formal Verification: An Essential Toolkit for Modern VLSI Design*. Morgan Kaufmann, 2015.