Journal of Systems Architecture 74 (2017) 1–13

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

A resource-efficient network interface supporting low latency reconfiguration of virtual circuits in time-division multiplexing networks-on-chip

Rasmus Bo Sørensen*, Luca Pezzarossa, Martin Schoeberl, Jens Sparsø

Department of Applied Mathematics and Computer Science, Technical University of Denmark, Kgs. Lyngby, Denmark

ARTICLE INFO

Article history:
Received 4 July 2016
Revised 21 December 2016
Accepted 10 February 2017
Available online 13 February 2017

2010 MSC:
90B18
68M10

Keywords:
Network-on-chip
Real-time systems
Reconfiguration
Time-division multiplexing

ABSTRACT

This paper presents a resource-efficient time-division multiplexing network interface of a network-on-chip intended for use in a multicore platform for hard real-time systems. The network-on-chip provides virtual circuits to move data between core-local on-chip memories. In such a platform, a change of the application's operating mode may require reconfiguration of virtual circuits that are setup by the network-on-chip. A unique feature of our network interface is the instantaneous reconfiguration between different time-division multiplexing schedules, containing sets of virtual circuits, without affecting virtual circuits that persist across the reconfiguration. The results show that the worst-case latency from triggering a reconfiguration until the new schedule is executing, is in the range of 300 clock cycles. Experiments show that new schedules can be transmitted from a single master to all slave nodes for a 16-core platform in between 500 and 3500 clock cycles. The results also show that the hardware cost for an FPGA implementation of our architecture is considerably smaller than other network-on-chips with similar reconfiguration functionalities, and that the worst-case time for a reconfiguration is smaller than that seen in functionally equivalent architectures.

1. Introduction

Packet-switched networks-on-chip (NoCs) have become the preferred paradigm for interconnecting the many cores (processors, hardware accelerators etc.) found in today's complex application-specific multi-processor systems-on-chip [1,2] and general-purpose chip multi-processors [3,4].

In the multi-processor systems-on-chip domain, a significant amount of previous research has targeted the generation of application-specific NoC platforms e.g., [5,6]. With the growing cost of developing and fabricating complex VLSI chips, application-specific platforms are only feasible for very few ultra-high-volume products. In all other cases, a cost-efficient platform must support a range of applications with related functionality. This implies that the hardware resources and the functionality they implement should be as general-purpose and generic as possible, targeting a complete application domain instead of a single application. This view is expressed in the principle *provide primitives not solutions* that is well-known and accepted in the field of computer architecture. We adopt this view, striving to avoid hardware resources for dedicated and specific functionality.

The application domain we target is real-time systems. In real-time systems, the whole architecture needs to be time-predictable to support worst-case execution time (WCET) analysis. A NoC for real-time systems needs to support guaranteed-service (GS) channels. Furthermore, many hard real-time applications have multiple modes of operation. To support applications that change between operating modes, the NoC must be able to reconfigure the virtual circuits (VCs) at run-time.

This paper proposes and evaluates a flexible and resource-efficient network interface (NI) for hard real-time systems. Our NoC implements VCs using static scheduling and time-division multiplexing (TDM). A VC provides GS channels in the form of a guaranteed minimum bandwidth and a maximum latency. Furthermore, transfer of data between an on-chip memory and the NoC is coupled with the TDM schedule so that we can give end-to-end guarantees for the movement of data from one core-local memory to another core-local memory. This architecture avoids both *physical* VC buffers in the NIs and credit-based flow control among the NIs that are found in most other NoC designs [7–9]. Moreover, the usage of TDM schedules leads to a reduced hardware complexity

* Corresponding author.
E-mail address: rboso@dtu.dk (R.B. Sørensen).

http://dx.doi.org/10.1016/j.sysarc.2017.02.001
1383-7621/© 2017 Elsevier B.V. All rights reserved.

due to the lack of buffering in the routers and due to a static traffic arbitration.

The main contribution of the paper and a key feature of this NI is its very efficient support for mode changes. The active schedule can be switched from one TDM period to the next, without breaking the communication flow of VCs that persist across the switch. This contrasts to the Æthereal family of NoCs [9,10], which provides similar functionality at a higher hardware cost and longer reconfiguration time.

Our NI can store multiple TDM schedules and it supports instant switching from one schedule to another, synchronously across all NIs. The last TDM period of one schedule can be followed immediately by the first TDM period of a new schedule. This allows VCs that persist across a schedule switch to be mapped to different paths, without any interference to their data flow. This again avoids the fragmentation of resources seen in the previously published solutions [9,10], in which no changes can be made to circuits that persist across a mode change and where the set-up of a new circuit is limited to using free resources.

If the schedule tables are too small to store all necessary schedules, our NoC can transparently transmit new schedules via the standard VCs. In this way, we avoid fixed allocation of resources for schedule transmission.

The NI presented here is an extension of [11], which is part of the Argo NoC [12]. A preliminary version of the new NI was published in [13]. In the rest of the paper, we refer to the NoC that uses the new NI as the Argo 2.0 NoC. The main contributions of this paper are:

- support of instant reconfiguration of VCs;
- a more elaborate analysis of the TDM schedule distribution through the NoC;
- variable-length packets to reduce the packet header overhead, resulting in shorter schedules and/or higher bandwidth on the VCs;
- interrupt packets to support multicore operating systems;
- a more compact TDM schedule representation in the NIs, reducing the schedule memory requirements;
- analysis of the effect on the TDM period length of using GS communication for reconfiguration;
- a discussion on the scalability of the architecture.

This paper is organized into seven sections. Section 2 presents related work. Section 3 provides background on mode changes, TDM scheduling, and the Argo NoC. Section 4 presents the Argo 2.0 architecture in detail. Section 5 describes the reconfiguration method and its utilization. Section 6 evaluates the presented architecture. Section 7 concludes the paper.

2. Related work

This section presents a selection of NoCs that offer GS connections and that support run-time reconfiguration of the GS provided. One approach to implementing GS connections is to use non-blocking routers in combination with mechanisms that constrain packet injection rates. These NoCs are reconfigured by resetting the parameters that regulate the packet injection rates to the new requirements.

Mango [14] uses non-blocking routers and rate-control, but only links are shared between VCs. Each end-to-end connection is allocated to a unique buffer in the output port of every router that the connection traverses and these buffers use credit-based flow control between them. The bandwidth and latency of the different connections are configured by setting priorities in the output port arbiters of the router and by bounding the injection rate at the source NI. Connections are set up and torn down by programming the crossbar switches, which is done using best effort (BE) traffic. In Mango, we can observe that the reconfiguration directly interacts with the rate control mechanism in the NIs, the crossbars, and the arbiters in the routers. In addition, the fact that GS connections are programmed using BE packets may compromise the time-predictability of performing a reconfiguration.

The NoC used in the Kalray MPPA-256 processor [15] uses flow regulation, output-buffered routers with round-robin arbitration, and no flow control. Network calculus [16] is used to determine the flow regulation parameters that constrain the packet injection rates such that buffer overflows are avoided and GS requirements are fulfilled. The Kalray NoC is configured by initializing the routing tables and injection rate limits in the NIs.

IDAMC [17] is a source-routed NoC using credit-based flow control and virtual channel input buffers together to provide GS. IDAMC provides GS connections by implementing the back suction scheme [18], which prioritizes non-critical traffic while the critical traffic progresses to meet the deadline.

To our knowledge, details on how reconfiguration is handled in Kalray, Mango and IDAMC have not been published. However, we can safely assume that setting up a new connection must involve the initialization and modification of flow regulation parameters, and tearing down a connection must involve draining in-flight packets from the VC buffers in the NoC.

An alternative to the usage of non-blocking routers in combination with constrained packet injection rates is VC switching implemented using static scheduling and TDM. These NoCs can be reconfigured by modifying the schedule and routing tables in the NIs and/or in the routers.

The Æthereal family of NoCs [9,10] uses TDM and static scheduling to provide GS. The original Æthereal NoC [19] supports both GS and BE traffic. The scheduling tables are in the NIs and the routing tables are in the routers. Reconfiguration is performed by writing into these tables using BE traffic. Analogously to the Mango NoC approach, using BE traffic may compromise the time-predictability of a (re)configuration. The dAElite NoC [9] focuses on multicast and overcomes this problem by introducing a separate dedicated NoC with a tree topology for the distribution of the schedule and routing information during run-time reconfiguration.

The aelite NoC [10] only supports GS traffic and it is based on source routing. This reduces significantly the high hardware cost of distributed routing and combined support for BE and GS traffic of the original Æthereal NoC. For this NoC, the routers are simple pipelined switches and both schedule tables and routing tables are in the NIs. Reconfiguration involves sending messages across the NoC using GS connections from a reconfiguration master to the schedule and routing tables that are required to change; these GS connections are reserved for this purpose only.

The original version of the Argo NoC [12] has some functional similarity with aelite. It only supports GS traffic and it also uses a TDM router with source routing. The Argo design avoids VC buffers and the credit-based flow control that account for most of the area of the NIs of the Æthereal, aelite, and dAElite range of NoCs. The Argo NoC uses a more efficient NI [11] in which the direct memory access (DMA) controllers are integrated with the TDM scheduling. The original version of the Argo NoC does not support reconfiguration.

In all the presented NoCs that uses VC switching and TDM static scheduling, the re-mapping of VCs that persist across the reconfiguration is not supported, since the reconfiguration is done incrementally (tearing down unused circuit and setting up new ones). This can lead to sub-optimal usage of resources due to fragmentation. If re-mapping of VCs is needed, the entire application must be suspended during the reconfiguration.

This paper presents a new version of the Argo architecture that implements the same functionality as the first version of Argo,