dAElite: A TDM NoC supporting QoS, multicast, and fast connection set-up

Radu Stefan, Anca Molnos, Kees Goossens

Abstract—Networks-on-Chip are seen as promising interconnect solutions, offering the advantages of scalability and high frequency operation which the traditional bus interconnects lack. Several NoC implementations have been presented in the literature, some of them having mature tool-flows. The main differentiating factor between the various implementations are the services and communication patters they offer to the end-user. In this paper we present dAElite, a TDM Network-on-Chip that offers a unique combinations of features, namely guaranteed bandwidth and latency per connection, built-in support for multicast, and a short connection set-up time. While our NoC was designed from the ground up, we leverage on existing tools for network dimensioning, analysis and instantiation. We have implemented and tested our proposal in hardware and we compared it to Æthereal, a state-of-the-art NoC with similar features, but no multicast. We find that the connection set-up time is reduced by a factor of 10 and the network traversal latency is decreased by 33%. Moreover, considering realistic values of the network parameters dAElite has a lower hardware area when synthesized in 65 nm technology.

Index Terms—Hardware Implementation, Network Architecture and Design, Circuit-Switching Networks

1 Introduction

As the complexity of Systems-on-Chip (SoC) increases, traditional bus-based interconnects become limited in terms of efficiency and performance. Networks-on-Chip [12], [5] (NoC) were proposed as a scalable replacement that can cope with the increasing number of on-chip IPs.

SoCs typically execute various, real-time or non real-time, applications which may have diverse requirements from the interconnect, e.g., high throughput for video, low latency for serving cache misses, etc.. These applications run concurrently in different combinations denoted as *use-cases*. Providing NoC service guarantees, e.g., minimum bandwidth, bounded latency, is crucial for the timing analysis, verification, and execution of real-time applications [17]. At the same time, special communication patterns like multicast or broadcast may be required, for example for implementing cache coherence or synchronization primitives, and in certain applications that involve distributed decision making algorithms. Besides the service guarantees and various communication patterns, the NoC implementation should have low cost, and ideally provide fast (re)configuration to adapt to dynamic use case switches. Existing NoC approaches either (i) offer service guarantees but do not support multicast [19], [20], or (ii) provide multicast but at the expense of a high cost [13], [29] or of compromising the guarantees of service [24].

- *Radu Stefan is with the Technical University of Eindhoven, email: R.Stefan@tue.nl*
- *Anca Molnos is with the Technical University of Delft, email: A.M.Molnos@tudelft.nl*
- *Kees Goossens is with the Technical University of Eindhoven, email: K.G.W.Goossens@tue.nl*

In this study, we have chosen a Circuit Switching technique to implement a network offering hard guarantees in terms of bandwidth and latency per connection. Circuit Switching was proven to be a cost-effective technique for implementing real-time communication [14]. Our proposal supports multicast which is increasingly a requirement in multi-core systems. Our network uses a time-division-multiplexing (TDM), contention-free scheme and a distributed routing model similar to one of the Æthereal [13] flavors. However, we propose a new configuration infrastructure, that is one order of magnitude faster than Æthereal, and has efficient encoding of the configuration data, thus increasing the speed of setting up and tearing down connections. The configuration infrastructure also features a novel method for synchronizing the TDM slot counters. Our proposal compares favorably in terms of hardware cost and performance to the most cost-effective of the Æthereal models. We support only guaranteed-services (GS) because, as suggested in [14], GS offers a better performance-cost ratio and are in fact the more likely to be required by applications in the embedded domain. We refer to our network as distributed-aelite, dAElite, as for network dimensioning and hardware instantiation we use the standard Æthereal tools, with a modified back-end to generate the new architecture.

The rest of this paper is organized as follows. Section 2 describes similar network implementations and other related work. In Section 3 we describe the contention-free routing model which is used by our current proposal. We present overall system view and the building blocks of our network in Section 4. The various configuration tasks, including setting up multicast trees are presented in Section 5. Experimental results are presented in Section 6 while the last section presents conclusions.

TABLE 1
Comparison with network implementations using similar concepts

Network	Link sharing	Routing	Connection Setup	End-to-End Flow Cont	Connection types
dAElite	TDM	distributed	dedicated	separate wire,TDM	1-1,multicast
aelite [19]	contention-free TDM	source	GS	headers	1-1,channel trees
TTNoC [31]	contention-free, pulsed	source	GS,over data network	none	1-1,multicast
TTNoC [34]	contention-free, pulsed	distributed	unspecified	none	1-1,broadcast
Æthereal w/BE [13]	contention-free TDM	source/distributed	GS/BE,guaranteed	headers	1-1,multicast[1]
Kavaldjiev [20]	VCs	source	packet,BE[2]	none	1-1
Wolkotte [40]	SDM	distributed	separate BE	separate wire	1-1
Nostrum [29]	TDM,looped	unspecified[3]	container[4]	none	1-1,multicast
SoCBUS [23]	none	distributed	packet,BE	none	1-1

2 Related work

Many NoC implementations, either connectionless or connection-oriented, have been proposed in the literature. These networks may offer both Best-Effort (BE) and Guaranteed Services (GS). Networks-on-chip as SPIN [1], xPipes [6], qNoC [11], SoCIN [42], artNoC [35], Quarc [30] and [33], implement a connectionless packet switching approach. QNoC implements quality-of-service through the means of traffic classes, but the guarantees offered are at best statistical. ArtNoC has support for multicast but only from one node at a time.

An overview and analysis of several multicast methods is presented in [2]. Support for multicast is provided in NoCs presented in [30] and [33]. Another approach is BENoC [27], which uses a bus to complement the services of the NoC. While the NoC would provide high data throughput, the bus would provide low latency messaging, multicast and broadcast. Compared to BENoC the advantage of our approach is that we can provide high-troughput multicast and more multicast connections operating in parallel. A different approach that can provide multicast is the stochastic communication proposed in [10]. While this is a significant departure from the established practices of NoC it may become useful in future technologies.

Connectionless packet switching NoCs typically do not offer latency and bandwidth guarantees, thus we do not discuss them further. In the following we comment on the connection-oriented, circuit-switching NoCs, as they are similar to dAElite. Among these we give special attention to [13], as it is the closest approach to ours.

Æthereal [13] is a hybrid network offering both Best-Effort and Guaranteed Services. Æthereal supports three routing models, distributed routing with BE configuration, source routing with BE configuration and source routing with GS configuration. More recent studies [14] suggest that the BE versions of Æthereal are not very cost-effective. For guaranteed services, Æthereal makes use of a routing model called contention-free routing in which each connection may use a link in a given timeslot. Channel trees [15] enhance the performance of this basic scheme, by allowing sharing of timeslots between channels, i.e., connections. This sharing may render invalid the service guarantees per connection, thus are not discussed further.

aelite [19] inherits the GS-only model from Æthereal, and introduces the possibility of using asynchronous and mesochronous links. Although we have not currently investigated this possibility, we believe that the same techniques can be used in dAElite. From here onwards, we will refer to the GS-only version of Æthereal as aelite, without any implications to a particular asynchronous or mesochronous link implementation scheme.

The implementation of multicast in Æthereal was proposed in [32] using separate connections. dAElite uses instead a broadcast/multicast tree to achieve the same result. Our solution is more efficient since it avoids both using separate channels inside the NI and using the link bandwidth n times, one for each of n destinations. Compared to Æthereal, we also use a more efficient, low-cost connection set-up mechanism. The connection state is stored inside all network elements in a distributed manner and the network configuration mechanism is centralized. Moreover, aelite requires a separate data connection over the network to configure the buses around the NoC. dAElite programs these busses through a broadcast mechanism, leading to faster configuration.

A network very similar to aelite is TTNoC [31] which also uses contention-free routing but claims to offer more freedom than a fixed, periodic TDM table. The network supports multicast but because source routing is used we expect a significant overhead in encoding the multicast trees in the packet headers. An earlier version of TTNoC [34] used distributed routing and supported a broadcast operation. However, a ring topology was intrinsic to this implementation while our implementation supports any topology. TTNoC does not include flow control.

Æthereal and dAElite use a TDM scheme to share the link bandwidth between connections, the model is described in more depth in the following section. One of its advantages is low buffer requirements at router level. Another network that uses a TDM scheme to provide guaranteed bandwidth is Nostrum [29]. Nostrum does

1. The distributed version of Æthereal could in theory support multicast at network level, although a solution for configuring the nework for this scenario was not proposed; multicast was proposed using separate connections for each target

2. Guaranteed connections have preallocated VCs and setup is assumed to always succeed

3. The paper only mentions that routes are decided at run-time, possibly they are stored in a distributed fashion inside the routers

4. No explicit connection setup is required, containers can be added and removed at will at runtime by any of the nodes on the route but lack of conflicts must be ensured

not have a fixed TDM wheel size, but instead, the TDM period is linked to the length of looping connections. Multicast is supported by adding more receiver nodes to a closed loop. Nostrum also offers BE communication using deflection routing. One disadvantage of Nostrum is that routing paths, and consequently multicast node sets, must be decided at design time.

The network proposed in [20] uses per-connection virtual channels (VCs) and round-robin arbitration to provide communication guarantees. VCs are in general expensive as they require buffers, multiplexers, demultiplexers and separate flow control. The number of VCs per router suggested by the authors due to cost concerns is of only 4 which may limit the number of simultaneously supported connections.

aSOC [22], [21] implements the same type of static TDM schedule found in Æthereal, but it does not implement the actual end-to-end connections, leaving this task to the IPs.

MANGO [8] is an asynchronous network implementation that uses, as [20], per-connection virtual channels. Since the network is clockless, there is no actual TDM table. Like Æthereal, connection setup is provided by using a Best-Effort network.

Another possibility for link sharing is SDM, used by [40]. Like our implementation, it makes use of an external network for route configuration, but it does not explicitly specify how this network is implemented. Reported configuration times are higher than those of dAElite.

Some implementations like SoCBUS [23] do not share the link between connections. This approach has a very low cost but it may result in excessive blocking.

Table 1 summarizes the related approaches to several aspects of the NoC implementation. One key differentiator is the type of routing employed which also has implications on the location where the connection state is stored. Source routing encodes the packet path in the header of the packet while distributed routing relies on separate routing decisions at each hop. We consider source routing to be too expensive for multicast and broadcast especially if small packets are considered, thus dAElite utilizes distributed routing.

Our proposal provides a unique set of features, namely multicast, multi-path routing, a low cost contention-free routing model and distributed routing, along with an improved performance/cost ration compared to the state-of-the-art.

3 Background: Contention-Free Routing

In this section we present contention-free routing, which is the routing model that we used to provide guaranteed services. Under this model, the bandwidth of each link is split, in the time domain, into a predefined number of timeslots. Each connection receives exclusive use of some of these timeslots from the moment it is set up until torn down, in a typical circuit-switching scheme. A network-wide schedule guarantees that packets never collide and never have to wait for each other (Figure 1), hence the name of contention-free routing. This reduces buffer requirements as well as network traversal time.

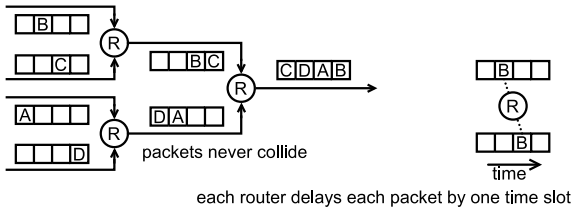


Fig. 1. Contention-free routing

The problem of computing the schedule which guarantees the collision-free movement of packets inside a network has been addressed in several studies. When the communication pattern is known beforehand it can be computed at design-time [18], [25], but if that is not the case, computation at run-time is also possible [28], [38].

Contention-free routing may be used in combination with either source or distributed routing to implement the global communication schedule. In source routing (Figure 2a) the path corresponding to each connection is stored inside the Network Interface (NI) and is sent inside the header of each packet. Slot tables inside the NI control the exact time when packets are allowed to be inserted into the network. No provisions are made for sending data to multiple destinations.

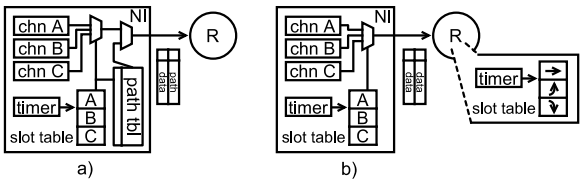


Fig. 2. Source routing (a) and distributed routing (b)

Distributed routing (Figure 2b) uses slot tables inside each router in addition to the ones inside the NIs. Packets are routed based on their time of insertion into the network and their time of arrival at each router. No header is thus necessary and the payload efficiency is higher. Broadcast and multicast can be easily achieved by setting up the router slot tables to forward the data packet to multiple destinations simultaneously. Existing distributed models [13] rely on the Best-Effort (BE) infrastructure for connection set-up which is both expensive and does not deliver guarantees regarding the set-up time [14]. In dAElite we use the contention-free model with distributed slot tables but we rely on a different configuration mechanism, based on a dedicated broadcast configuration network.

4 dAElite

In this section we present the principles behind and the hardware implementation of our proposed NoC. At the core of our proposal is the broadcast based configuration layer, which is capable of (1) setting up and tearing down connections quickly, (2) ensuring synchronization of the router and NI slot counters, (3) setting and reading connection state and (4) configuring adjacent buses. We start by presenting an overview of a typical system based on dAElite, after which we will give the details of the configuration infrastructure, the router and NI architecture. dAElite was implemented and tested in FPGA.

4.1 System overview

A typical SoC platform based on dAElite is exemplified in Figure 3. dAElite is a connection based network. For a master IP to communicate to a slave IP over the network, a connection is set up between two network interface shells, one connected to the master, the other connected to the slave. The network interface shells have the role of translating the request between the bus protocol spoken by the IPs and the packet format used by data while traversing the network.

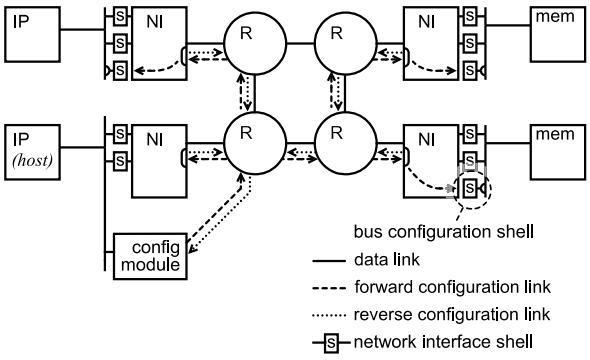


Fig. 3. Example dAElite platform.

IPs are connected to the NI shells by lightweight local buses which have the role of multiplexing or demultiplexing requests to and from multiple network shells (and thus network connections). This is because the network connections are long-lived and an IP can have several connections simultaneously open to multiple other IPs.

The buses may be configurable, on the master side to select address ranges corresponding to each connection, on the slave side to select arbitration schemes and priorities.

A typical usage scenario is that the required connections are set up before starting an application or an execution phase of an application. The application can use the configured connections during that execution phase without further intervention to the network configuration. The connections are torn down once they are no longer needed. Setting up and tearing down connection can be done dynamically without affecting the normal operation of the system, i.e., an application can use existing open connections while others are being set up and torn down.

4.2 Proposed channel set-up, configuration, and reset logic

The configuration infrastructure is used to update to set-up and tear-down network connections by updating the contents of the slot tables inside routers and NIs, to set and read back flow control information for each connection, to perform the synchronization of the slot counters inside NIs and routers and to configure buses adjacent to the network.

We implement the configuration logic as a dedicated broadcast network with a tree topology, with links running in parallel to a subset of the normal data network links. This subset is chosen in such a way as to minimize the distance from the configuration node to any of the network nodes.

One IP, by convention called *host*, has exclusive access to the configuration logic. The *host* performs write operations to a configuration module. These writes typically have a wide data width, e.g., 32-bit, compared to the width of the configuration links. The configuration module thus serializes the data words received from the *host* into several, smaller bit-width *configuration words* which are inserted at the root of the broadcast tree.

If the *host* does not need to send at one time as many *configuration words* as are contained in a data word, it can perform "0-padding". The configuration module will also send "0" values into the configuration broadcast tree when it has no data to send.

The configuration tree provides of forward and a reverse paths. The forward configuration path is of broadcast type. Each non-leaf node (always a router), forwards all the configuration data it receives to all its downstream neighbors. The NIs, being leaves in the tree, do no forward the incoming configuration messages. They do however produce messages for the configuration of buses using a different type of link. The configuration payload is deserialized into wider 37-bit words which are then translated by an NI shell into the DTL protocol, used by the configuration ports of the buses.

On the reverse path in the configuration tree, messages converge toward the configuration module. To avoid arbitration on the response path, the *host* only issues one message requiring response at a time. In our case, the requests requiring a response are read operations directed at the state tables of the NIs.

The requests and responses traveling through the configuration network take the form of packets, the format of which will be presented in Section 5.

4.3 Network Routers

The structure of network routers is presented in Figure 4. Because we are using a distributed routing mechanism

each router contains a slot table to store the TDM schedule. Incoming packets are "blindly" routed based on this schedule. A high operating frequency can be achieved because no arbitration is required and the router does not need to examine the packet contents.

The schedule for packet destinations for each of the inputs is contained in a slot table. A counter iterates circularly through this slot table and the selected row is used to control the router crossbar. A configuration submodule, implemented as a state machine is used for setting the initial value of the counter, as we will discuss in Section 5.4 and to update the contents of the slot table.

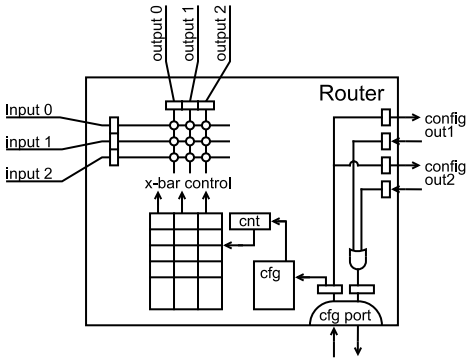


Fig. 4. dAElite Network Router

On the configuration connections, the router simply copies the input value to all of the outputs on the forward path and performs an "or" operation between all inputs writing the result to the output on the reverse path.

Data is thus buffered twice inside the router: once after link traversal, and once after crossbar traversal. The latency per hop is thus fixed to two cycles. In order to simplify design data is also buffered twice at each hop in the configuration tree. This allows the configuration links to be treated in the same way as the data links when dealing with timing constraints.

4.4 Network Interfaces

Network interfaces have the role of providing end-to-end connections over the network. A network interface (NI) is connected using a network data link to a network router, and one or more links to network interface shells. Each link to an NI shell supports an end-to-end connection to another NI shell at the other side of the network. NIs thus multiplex several communication channels having NI shells as endpoints to a single network data link.

As specified by the contention-free routing model, the packets belonging to different connections are inserted into the network only at specific times. The arriving packets are also forwarded to the proper NI shell based on their arrival time, according to a strict schedule. The departure and arrivals schedule is stored inside a slot table which is part of the NI. The slot table controls the multiplexer and demultiplexer in the same way the router slot table controls the router crossbar.

Figure 5 presents a diagram of the network interfaces. The network slot table, same as the one of the router, is indexed with the value of a circular counter and is programmed by a configuration submodule. For each of the data connections there are input and output FIFOs, credit counters for end-to-end flow control and decision logic for enabling or disabling the sending of data from an input FIFO to the network.

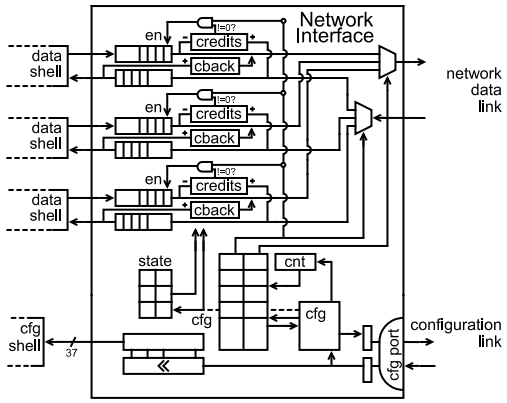


Fig. 5. dAElite Network Interface

Credit-based flow control is provided by the NI for each of the one-to-one connections going over the network (optionally flow control can also be disabled on a per-connection basis). Two credit counters are used for each connection. One credit counter keeps track of how many words of storage space are available in the output FIFO at the other end of the network. An input FIFO is only allowed to send data into the network if the value of this credit counter is different than 0 and the slot table indicates it is the connections' turn to send data. A second credit counter accumulates the number of words that were delivered to the destination from the output FIFO. The value of this counter is periodically sent back to the other end of the connection and the counter is reset.

As connections are bidirectional, credits for one direction are sent on separate bit-lines alongside data in the opposite direction. The separate credit lines and data obey the same TDM scheme and there is actually no distinction between the two at the router level. Other networks, like aelite [19] send the credits inside packet headers, but that approach is not viable here as dAElite does not employ packet headers. The number of bit-lines transporting credit information is configurable. To make better use of these lines, the value of the credit counters is sent serialized, over the 2 cycles of a TDM slot. In our test design, 3 wires dedicated to sending credit data are sufficient for sending the value of a 6-bit credit counter during each slot cycle.

The configuration submodule is responsible for updating other network state information like enabling

or disabling connections or reading or writing flow control information to credit counters. It also identifies special messages which are destined to the buses and deserializes them to the bus configuration shell.

5 NETWORK CONFIGURATION PROCEDURE

This section describes in detail the network configuration tasks. Special attention is given to the connection set-up and tear-down which is the most complex operation. We describe as well as packet format, explain how multicast is achieved and present our proposed method for synchronizing the slot counters inside routers and network interfaces.

5.1 Configuration packet format

The configuration is performed using configuration packets, consisting of one or more words, transmitted one per cycle over the configuration links.

The format of the configuration packet is illustrated in Figure 6. An end-of-packet is implicitly marked by the beginning of a new packet, but can also be marked explicitly. The configuration mechanism supports the writing of slot tables, reading and writing credit information, writing status flags governing NI behavior, and resetting internal TDM counters.

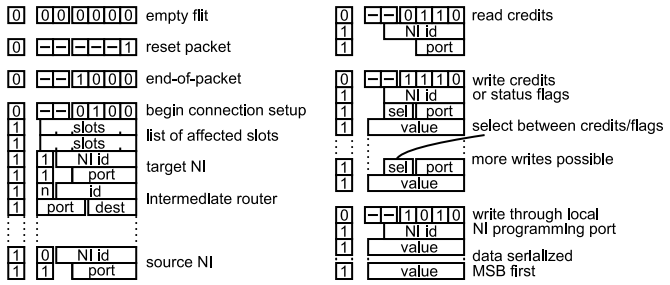


Fig. 6. dAElite configuration packet format

The configuration links have small bit-width, that is equal to the size of the configuration words. In order to optimize the logic of the state machines governing the configuration process, this width is selected in such a way that any of the parameters listed below can be encoded in a single configuration word in addition to one bit marking the command headers.

- a value uniquely identifying a specific router or NI, of size $(\log_2 N)$ where N is the number of network elements;
- two values identifying one input and one output port of a router or a null entry, of size $(2\times\log_2(p+1))$ where p is the number of ports;
- an NI channel id or null value, and an additional bit to indicate direction (note that in general an NI is expected to have more connections that a router has ports, but it has only one link connecting it to a router), of size $(1+\log_2(c+1))$ where c is the number of channels supported by the NI; and
- the value of a credit counter plus one state bit, of size $(1+\log_2(b+1))$ where b is the size of the largest buffer the counter must keep track of.

A configuration word size (including a one-bit marker for beginning-of-packet and empty flits) of 7 bits is sufficient for a network with 64 network elements (routers and NIs), routers with an arity of 7, network interfaces supporting up to 31 channels and buffer sizes of up to 31. From here onwards, we denote as *configuration word* a 7-bit word transmitted on a configuration link.

The first word in each packet indicates the type of configuration command. The first bit is always zero for a command word. The packet formats for the various operations are as follows (Figure 6):

For set-up and tear-down operations, the command word is followed by a list of slots used at the destination NI, represented as a bit-mask with one bit per slot. A list of the traversed Routers and NIs follows, each with a corresponding input and output descriptor. Each affected NI/Router recognizes its own id (a constant parameter defined at circuit design time) from the list and modifies its internal slot table. The path set-up procedure will be described in detail in the following section.

Comparatively, the other configuration operations are simpler. For the reading of credits, a header, the NI id, the port (connection) number are broadcast into the network and the addressed NI will recognize its own id from the packet and will reply with the counter value over the reverse configuration link.

Writing of credits is similar, except after the port selector the new value of the counter is sent. The addressed NI, once selected will keep watching for port/value pairs. In this way, writing multiple credit counters of a single NI can be done in a single configuration packet.

Writing to a bus programming port is again performed with a packet marked by a distinctive header. The addressed NI will deserialize the received data to the shell connected to the configuration port of the bus. We used this approach for compatibility reasons with existing configurable bus implementations.

5.2 A connection set-up example

Setting up a network connection consists of:

1) setting up network paths;
2) initializing credit counters; and
3) initializing bus address decoders.

Steps (1) and (2) have to be performed for the both the request and response channels. Step (3) is the last one performed as it signals to the bus that it can start using the connection for transferring data.

We illustrate in this section step-by-step how a path set-up is performed. Consider the system in Figure 7. We analyze the operation of setting up a path from port 0 of NI10 to port 0 of NI11. The IP which has access to the network configuration, which we call the *Host IP*, writes data words to the configuration module. As the

bit width of the dedicated configuration links is typically lower than the data width of the bus, a single write from the Host may contain multiple configuration words (for this system 4), which are sent over the configuration link by the configuration module.

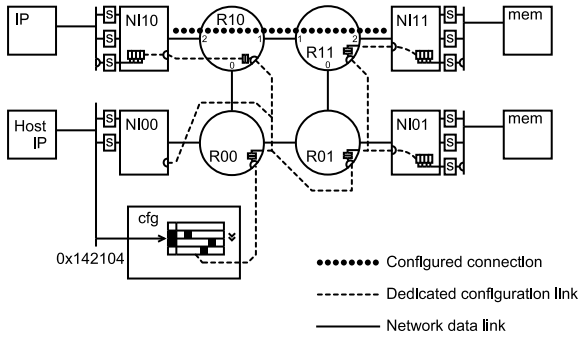


Fig. 7. Path set-up example

The first configuration word is a header that informs all the network elements that a path setup sequence will follow. It is uniquely identified by a most significant bit with a value of 0 and the value of 4 in the last 4 bit positions. The next two configuration words contain a table of slots affected by this path set-up operation. We assume here a slot table size of 8. The two bits set to one in this example identify slots 7 and 4.

Three cycles after starting the path set-up process (Figure 8a), allowance being made for whatever pipeline stages the configuration logic may contain, the complete slot table has been registered at router R00, while routers R01 and R10 have just recognized the path set-up header entered set-up mode. This is because each router has two pipeline stages on the configuration path, thus delaying the configuration by two cycles for the next routers in the broadcast tree. The configuration module, having emptied its serialization register is able to accept one more data word.

The configuration words after this point are organized in pairs. The first word in a pair identifies a network element while the second describes modifications to the slot table of the identified element. The MSB of each word is set to identify a valid configuration word. For the first word, a value of 1 in the MSB-1 position identifies an NI while a value of 0 identifies a router. The ID is stored in the least significant bits. For NIs the least significant bits identify the port (channel) number and the following bit identifies the input or output slot table. For routers, the least significant bits identify the input port and the immediately higher bits the output port.

The meaning of the configuration words in Figure 8a is the following: for the defined slots, the input of the Network Interface NI11 should be forwarded to channel 0. On the previous path segment, thus with the same slot table rotated by 1, router R11 should forward data from input port 1 to output port 2.

After two more clock cycles (Figure 8b), the path set-up header reaches R11 and NI10, the slot table is

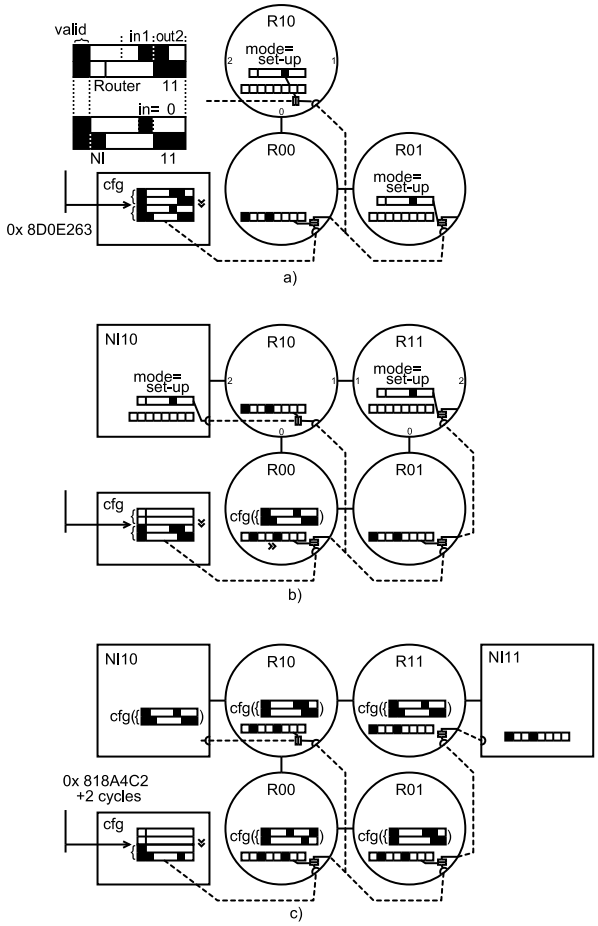


Fig. 8. Path set-up, slot table registered at the first router.

registered in R10 and R01, and the first network element identifier pair reaches R00. Because R00 does not recognize its own ID in this pair, it ignores the pair but at the same time it rotates its received slot table by 1 position.

Four cycles later (Figure 8b), after another 4 configuration words have been transmitted, the slot table finally arrived at all network elements and is stored, rotated by a different number of positions. So far, the network element IDs in the configuration packets never matched the element IDs of the traversed routers. The number of rotations in the slot table seems to depend so far on the distance from the root of the configuration tree, but this is misleading. The number of rotations actually depends on the number of path elements in the received list that did not match the local network element ID.

Eventually a configuration word carrying the proper ID reaches the network elements that need to be configured. In our case that happens simultaneously for R10, R11, NI11, as shown in Figure 9.

In order to avoid configuring upstream nodes before downstream nodes, we had to take an additional precaution: we need to verify that no path for data exists that is shorter than the difference in configuration path length from the root of the configuration tree.

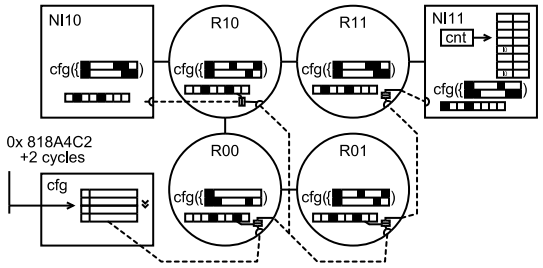


Fig. 9. Path set-up, slot table update takes place in R10, R11, NI11.

The update of the slot table is not performed instantaneously but through the use of a counter, one slot table element is initialized each cycle. We preferred this approach to reduce the cost of FPGA implementation. This implies that a cool down period is necessary before another path set-up or teardown may take place. A hardware cool down timer inside the configuration module enforces this policy by blocking channel set-up packet headers as long as it is different from 0. Other configuration packets are allowed during this time so the initialization of credit counters of bus address decoders can overlap with the actual initialization of slot tables.

The list of traversed routers/NIs begins at destination to ensure that downstream routers are initialized before the upstream NI and routers start sending packets. For each item in the list, the slot table which was sent in the beginning of the packet is rotated by one slot so that all routers along the path, when they recognize their id in the list, already have the properly aligned table. It is not mandatory that a packet contains a complete source-to-destination NI path, independent path segments can be initialized as well. This can be used to set up broadcast trees, for example.

Connection tear-down can be performed in the following way:

1) the application program must first make sure that no more requests are delivered to the given connection. This can be enforced by resetting the address range associated with that port on the bus;
2) the credit values are read back from the credit counters to check if all data items have reached their destination; and
3) the route is torn down using the same mechanism as setting up. For additional safety this can be done in two stages: first, the source NI to make sure no more flits are sent into the network, then the rest of the path.

5.3 Multicast

dAElite offers a mechanism to achieve multicast that is both simple and efficient. The TDM schedule in a dAElite router is implemented as a table that specifies for each output port which input port should the data be taken from during each cycle. Two (or more) output ports are allowed to use the same input port as a source (Figure 10).

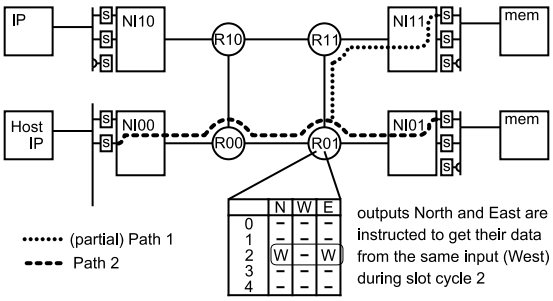


Fig. 10. Multicast in dAElite

The multiple paths to the different destinations form a tree, rooted at the source NI. This is more efficient and offers higher performance than having separate connections from the source NI to all destinations because in the latter case the bandwidth on output link of the source NI would need to be divided between all the connections.

The initialization of multicast trees is made possible by allowing a configuration packet to set up partial paths; i.e., paths that start at a router instead of a source NI. In the tree, partial branches should be set-up first, before the "main branch" which goes to the root of the tree. This is done to avoid the source NI starting to transmit before the entire multicast tree has been set up. In the example in Figure 10, the partial path R01-R11-NI11 should be set up first and the path NI00-R00-R01-NI01 should be set up second.

All multicast destination shells will receive the same stream of messages and will translate them into the same write commands on the destination buses (issuing read commands would be pointless since there are no semantics defined for a simultaneous read from multiple destination).

One potential problem when using multicast is that the default flow-control mechanism cannot be used (the source NI only has one credit counter for each communication channel). The least expensive solution to this problem is to guarantee that the destination bus can process the received memory transaction at the same rate that they are transmitted. In our scheme guaranteeing this is made easier by the fact that the connection bandwidth can be set to a desired value (with a certain granularity) by allocating more or fewer TDM slots to it.

5.4 Slot counter synchronization

The dAElite network is at the logical level a synchronous network implementation, which furthermore relies on a notion of global time. This may in practice be difficult to achieve [26] due to clock skew issues in large designs. Nevertheless, studies have shown [7] that clock skew values in the order of tens of picoseconds are achievable. Other approaches exist [19] that avoid the problem by offering synchronous behavior at the logical level while

relying on a mesochronous or asynchronous implementation at the physical level.

In this section, making abstraction of how the synchronization issue is solved at the physical level, we show how to solve the issue of synchronizing the slot counters at the logical level. Consider the following scenario, illustrated in Figure 11. We use the following simplification which allows us to explain the synchronization mechanism: assume that the clock skew between neighboring nodes (nodes that are connected by a network link) is sufficiently small to allow correct data transmission between the nodes and allow the node to agree on a common value of the current time slot when they exchange data. Over large distances between nodes that are not connected using a link that is not required.

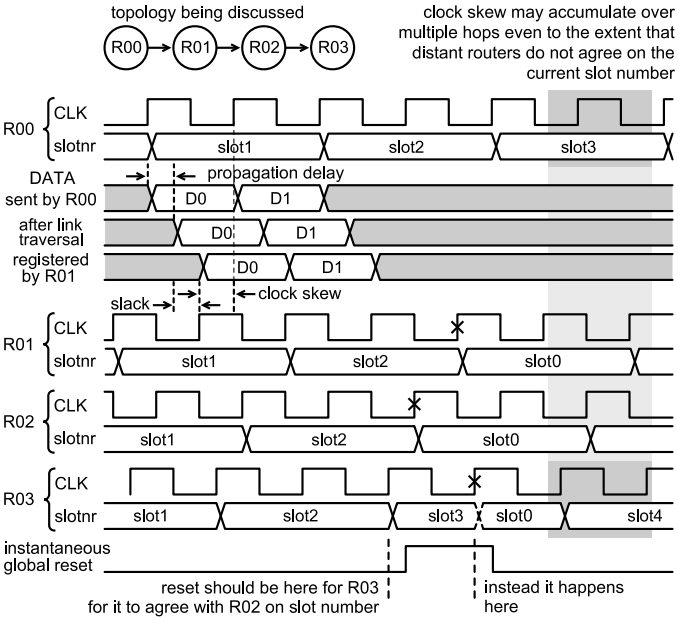


Fig. 11. dAElite relaxed synchronous model.

This may be a reasonable assumption for a mesh network where links are short and connect only physically neighboring nodes. The important aspect here is that each router agrees on the value of the slot counter and is able to transfer data to its neighbors. It is sufficient if this happens only from the logical point of view, regardless of the physical implementation.

If the slot counter values were reset using a global reset signal, and that reset signal did not follow the same skew pattern as the clock, the reset signal will inevitably be sampled on the wrong clock edge in one of the various clock skew domains. (Figure 11 shows how an instantaneous reset signal which does not follow the pattern of clock skews causes R02 and R03 to disagree on the current slot number).

To solve this problem we provide a reset mechanism which follows the logical view of network time. The reset signal is transmitted as a special packet through the configuration network. This packet will arrive at the different nodes in different clock cycles, but the logical delay is known (two cycles per hop) and the nodes compensate for it by initializing their slot counters to the distance to the root of the configuration tree.

6 Experimental results

Comparing NoC implementations is not straightforward as the services provided by them may be different. Furthermore many of the solutions presented in the literature only give details of the network routers which makes it difficult to asses the cost of an entire network able to deliver the service level demanded by an application.

The obvious target for comparison is represented by the Æthereal network, with which dAElite shares the connection-free-routing based TDM model. We dedicate therefore the largest part of the cost and performance evaluation to the comparison with Æthereal.

We have implemented a fully-fledged dAElite NoC supporting network and remote buses configuration, link set-up and tear-down, setting status flags, setting and reading credit information, normal reads and writes from the IP to the memories, and broadcast from an IP to multiple memories was prototyped in FPGA. Our experimental platform, tested in FPGA, is the same one presented in Figure 3 and uses two MicroBlaze processors as IPs. For network dimensioning and hardware instantiation we use the standard Æthereal tools, with a modified back-end to generate the new dAElite architecture.

In the rest of this section we will present a comparison of hardware cost to the cost of the aelite network together with a hardware cost breakdown per component, a comparison of the connection set-up time and performance, as well against aelite, and a general hardware cost comparison against other network implementations presented in the literature.

6.1 Hardware cost

The hardware cost of dAElite compared aelite (a lightweight version of Æthereal) is presented in Table 2. Both implementations consist of 2x2 mesh networks, with 4 NIs and 4 connections. The sizes of the FIFOs implementing the queues inside NIs have been set to 16 words except for the aelite configuration channels which use the minimum required size of 3 words on the forward path with 8 words on the return paths. Our proposed implementation uses the dedicated configuration infrastructure. We believe the FPGA implementation can be further improved by taking advantage of FPGA specific structures for implementing slot tables and FIFOs.

For the FPGA implementation using the Xilinx tools, we performed runs with both area and speed optimizations. For the ASIC implementation we did not perform time-constrained synthesis as timing is more likely to be dictated by floorplaning (in particular the length of the long router-to-router links) which we did not address in this paper.

TABLE 2
Hardware cost comparison

	Xilinx ISE Virtex 6 - Area	Xilinx ISE Virtex 6 Constrained 200MHz	Synopsys TSMC 65nm flatten, high effort
aelite (8 slots)	5,500 slices 7,665 LUTs 15,444 registers 8.379 ns	5,393 slices 9,403 LUTs 15,840 registers 4.986 ns	171,820 total 39,563 combinational 132,261 noncomb. 1.13 ns
dAElite (8 slots)	3,098 slices 10,026 LUTs 12,323 registers 8.19 ns	3,655 slices 12,470 LUTs 12,973 registers 4.968 ns	141,233 total 29,541 combinational 111,693 noncomb. 1.08 ns
dAElite (32 slots)	3,483 slices 10,903 LUTs 13,533 registers 8.84 ns	4,425 slices 14,335 LUTs 14,191 registers 4.85 ns	156,639 total 33,410 combinational 123,230 noncomb. 1.08 ns

For both aelite and dAElite it is possible to arbitrarily introduce pipeline stages (in a number multiple of the slot size) on the links in order to more easily meet the timing requirements. These will behave at the logical level as routers with only one input and one output, that do not require configuration. dAElite also has an advantage here, as it allows a finer granularity in choosing the number of pipeline stages due to the smaller slot size.

The cost of each hardware component, based on ASIC synthesis, is presented in Figure 12. "NI-aelite" and "router-aelite" denote the corresponding component cost of aelite. The bulk of the cost is in the network interfaces due to the relatively large FIFOs.

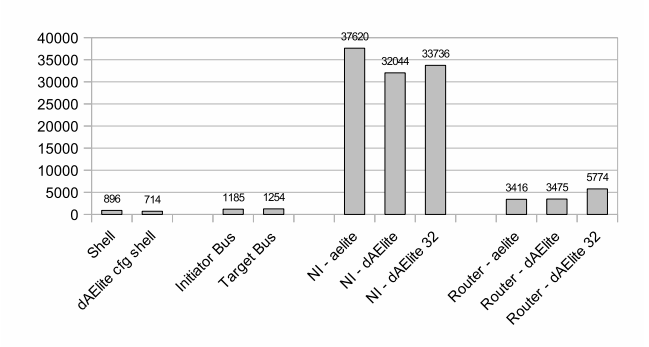


Fig. 12. Hardware cost breakdown

For the routers, the gain of our implementation is on account of the reduction in flit size to 2 which eliminates one register per router channel. A multiplexer per channel per router is also eliminated as we do not need to shift routing information in packet headers. We add instead more complex configuration logic and a slot table. The cost of a dAElite router with a slot table containing 8 entries is roughly equal to the cost of a aelite router, however, with a slot table of 32 entries the cost of the dAElite router is 68% higher.

Compared to aelite, our NIs benefit from a simpler configuration mechanism which uses a state machine connected directly to the configuration infrastructure, instead of interpreting requests on a DTL bus (aelite uses a regular network channel for configuration, followed by a shell translating the configuration messages into DTL bus transactions.) Further area gains originate from the removal of the table of paths inside the NI, and some of the configuration buses. The cost of NIs in dAElite is 14.8% lower than that of aelite when the slot table has 8 entries. The dAElite NI with a slot table size of 32 is still 10.3% less expensive than the aelite NI with a slot table of size 8.

The cost of dAElite is more sensitive to the size of the TDM wheel. The slot tables inside the NIs need to contain entries for both departures and arrivals whereas in aelite they only contain entries for departures. Additionally, slot tables are present in each router. We expect, and we confirm experimentally that the cost increases linearly with the number of slots. It is expected therefore that as the number of slots is increased the hardware cost of dAElite will increase relatively to aelite. We performed additional synthesis runs to determine the point where the cost of dAElite becomes higher than that of aelite. We found that point to correspond to a TDM table size of 70 (Figure 13). In practice we have found TDM table sizes of 32 slots (or less) to be sufficient for supporting most types of traffic [36].

On the other hand, the cost of dAElite increases less than that of aelite with the number of connections, because a path per connection is not stored inside the NIs (the path is stored in a distributed manner inside the router slot tables). Our setup, which uses a relatively low number of connections provides a conservative estimate of the hardware area benefit of dAElite.

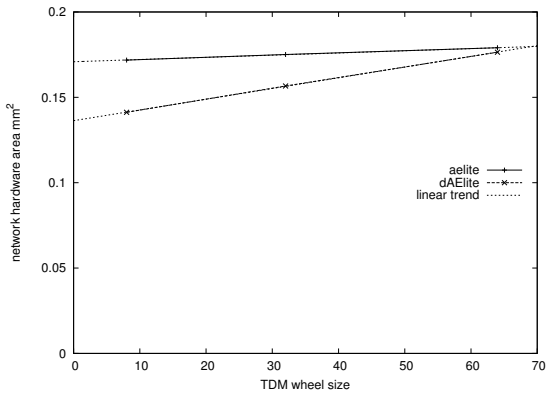


Fig. 13. Dependence of the hardware cost on the number of slots

dAElite has one disadvantage, namely an increase in the number of link wires, in part due to the configuration network, and in part because of the separate wires for end-to-end credit communication. In our test setup this increase amounts to 20.8%. This value is analytically derived from the width of the link w, the number of lines for credit transmission d, the size of the configuration links c and the ratio between the number of configuration links and the number of data links r. We used the formula $(w-1+d+r*c)/w-1$. The first -1 term is due to the fact that one of the wires in aelite signaling the

beginning of a new packet is not needed in dAElite. In our test setup w was 39. Denser topologies will present lower overhead because the configuration network is only a subtree of the network topology and thus r is lower. For the mesh topology the r asymptotically decreases to 0.75 as the size of the network grows. Larger networks on the other hand will see a slow increase due to c increasing with $log(n)$ where n is the number of network elements. These effects cancel each other to some extent. For example an 8x8 network having the same link and credit line width would have an overhead of 19.9%. Increasing the link width obviously decreases the relative overhead.

6.2 Configuration time

In our proposal, the path set-up time is only dependent on path length as all slots reserved for a connection are configured at once, for each one of the routers on the path.

We present the results of experiments in Table 3. All results are expressed in cycles. In both cases the configuration code is written in C and compiled with maximum optimization. The ideal value reported for Æthereal is taken from [16] and represents the configuration delay without taking into account processor execution time of the configuration code, but only the configuration read and write operations. In aelite, the path set-up time has is dependent on the distance between the host and the source node of the path, the distance between the host and destination node, and is roughly proportional to the number of reserved slots, as the slots in the source and destination NIs are reserved separately.

The ideal value for our proposal is computed analytically from the number of configuration words, which are sent one per cycle through the configuration network. This number includes two cycles for each hop on both the request and response path, six cycles for transmitting the slot tables (three for the request and three for the response path), header words and the updating of credits as indicated in Figure 6. This value also includes the a cooldown period of 20 cycles (one cycle for each slot plus a margin of 4 cycles for allowing the configuration logic to return to the idle state). A second cooldown period (one period is necessary for both the request and response path) was not included because it is allowed to overlap with the writing of credits and it may continue in background after the connection setup ended. The times measured on FPGA include both cooldown periods but the second is completely hidden by the function calling overhead.

A concern may arise here that for large slot table sizes the configuration time would be significantly increased. However this is not an issue in practice, for the following reasons. Configuring only one slot per cycle is not an intrinsic characteristic of our proposed architecture, but rather an implementation detail. This choice was made to allow packing the slot tables into distributed RAM in the FPGA implementation. The distributed RAM blocks have a size of 16 or 32 locations (for Xilinx FPGAs, depending on the model) and only one location can be updated per cycle. If the slot table is larger, it will use multiple distributed RAM blocks and slots in all blocks could be configured in parallel. The cooldown period would be limited to 16 or 32 and thus independent on the number of slots in the TDM period.

TABLE 3
Configuration times

	Æthereal		dAElite	
	ideal	measured	ideal	measured
6 hops	246	822-1555	60	81
8 hops			68	94
10 hops			76	119
12 hops			84	133

The improvement in set-up time compared with aelite is due to the following:

- overheads are reduced: the original implementation used DTL transactions encoded into packets;
- we avoid transmitting redundant information, the slot tables at the intermediate nodes are generated by rotating the slot table at destination; and
- the configuration bandwidth is not dependent on the slot table size and the number of slots allocated to configuration. A dedicated infrastructure is less expensive than implementing the same functionality over a generic network.

Our FPGA experiments indicate that dAElite configuration is roughly one order of magnitude faster than aelite.

6.3 Additional performance benefits

Compared to aelite, dAElite presents several other performance benefits:

Firstly, in dAElite, the router (and link) traversal delay is 2 cycles. This is lower than the 3 cycles used by aelite. We are able to achieve this without a negative impact on the clock frequency, because dAElite does not need to look at packet contents before making a routing decision. Routing is performed solely based on the packet arrival time and the contents of router's own slot table. This results in a reduction in the network traversal latency of 33%.

Secondly, the dAElite TDM slot is 2 words, and could be further decreased to a single word if necessary. In aelite it is not possible to arbitrarily decrease the slot table size because packets contain a header and the header overhead would become higher in shorter packets. A small TDM slot size is useful for improving the scheduling latency (packets need to wait for their turn before they can be inserted into the network).

Thirdly, dAElite does not suffer from header overhead. In aelite, the header overhead is between 11% and 33% because one header is required at least every 3

slots (possibly every slot when slots belong to different connections) and the header represents one third of the slot size.

Fourthly, dAElite allows routing one connection over multiple paths at no additional cost. In [37] it was shown that multipath routing can provide bandwidth gains of 24% on average. Multipath routing is also possible in aelite [37], but at an increased hardware cost.

Finally, aelite reserves at least one slot on each of the NI-router and router-NI links for configuration traffic. For a slot wheel size of 16 this represents 6.25% of the link bandwidth which cannot be used for data. This is not the case for dAElite.

All these improvements stack up, making dAElite a more attractive solution.

6.4 dAElite compared to other NoCs

We compare here the cost of the dAElite to the cost of other Networks-on-Chip reported in the literature. A conclusive comparison is difficult to perform because the different NoCs support different features and furthermore most publications only report the cost of the routers while our proposal consists of an entire NoC, including the interface to IPs. dAElite compares favorably in terms of router hardware area cost but the bulk of the cost is concentrated in the NIs. Nevertheless we present here a comparison using the available data.

We have synthesized our design in TSMC 65nm, 90nm and 130nm. We report the area after synthesis in Table 4, compared to other values of the router area reported in the literature. We do not report timing as we expect it to be more affected by other factors like length of the links. We expect dAElite to perform well in terms of operating frequency as it allows arbitrary pipelining of links and it lacks complex decision logic like arbiters.

A solution similar in concepts and functionality to dAElite is the one proposed in [4]. Same as dAElite, it makes use of a separate configuration network but it is based on a SDM scheme instead of TDM. The result is roughly 6.7 times more expensive but it offers more routing flexibility (in SDM any of the 4 lanes of an input port can be forwarded to any of the 4 lanes of an output port, but in our TDM scheme, one TDM time-slot can be forwarded only to the immediate next time-slot).

Compared to each of the other solutions dAElite shows an advantage in terms of hardware cost, an advantage ranging from 18% to more than one order of magnitude.

TABLE 4
Cost of a dAElite router compared to other implementations

16-bit 5-port router, 130 nm technology:	
artnoc [35] 2-flit buffers, 4 Virtual Channels	$0.060\ mm^2$
Wolkotte [41] circuit switched	$0.050\ mm^2$
Wolkotte [41] packet switched	$0.180\ mm^2$
dAElite router	$0.016\ mm^2$
16-bit 4-port router:	
Mango [9] *120 nm*, 8 Virtual Channels	$0.188\ mm^2$
dAElite router *130 nm*	$0.020\ mm^2$
32-bit 8-port router, 130 nm technology:	
Quarc [30] (not full 8x8 crossbar)	$0.063\ mm^2$
dAElite router (full 8x8 crossbar)	$0.053\ mm^2$
36-bit 8-port router, 130 nm technology:	
SPIN [3] (not full 8x8 crossbar)	$0.240\ mm^2$
dAElite router	$0.057\ mm^2$
5-port router, 90 nm technology:	
Banerjee and Wolkotte [4], 4 SDM lanes 16 bit/lane	$0.108\ mm^2$
dAElite router 64 bit links divided into 4 TDM slots	$0.016\ mm^2$
32-bit 4-port router, 130 nm technology:	
xpipes lite [39], 4 stages output buffer	$0.091\ mm^2$
dAElite router	$0.020\ mm^2$

7 Conclusions

In this paper we have proposed, implemented and evaluated a hardware prototype of a TDM NoC using contention-free, distributed routing, that has the following distinctive features: (i) support for QoS; (ii) support for multicast/broadcast; (iii) lower area cost than previously proposed implementations and no header overhead; and (iv) configuration and path set-up times significantly shorter than the closest approach. We are currently looking into the possibility of making this implementation open source.

References

[1] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. A Zeferino. SPIN: a scalable, packet switched, on-chip micro-network. In *DATE*, 2003.

[2] Ahmed Al-Dubai, Mohamed Ould-Khaoua, and Imed Romdhani. Design and analysis of multicast communication in multidimensional mesh networks. In *Parallel and Distributed Processing and Applications*, volume 4742 of *Lecture Notes in Computer Science*. 2007.

[3] A. Andriahantenaina and A. Greiner. Micro-Network for SoC: implementation of a 32-Port SPIN network. In *DATE*, 2003.

[4] A. Banerjee, Pascal T. Wolkotte, Robert D. Mullins, Simon W. Moore, and G. J. M. Smit. An energy and performance exploration of Network-on-Chip architectures. *IEEE Transactions on VLSI Systems*, 2009.

[5] L. Benini and G. De Micheli. Networks on chips: a new SoC paradigm. *Journal in Computer*, 2002.

[6] D. Bertozzi and L. Benini. Xpipes: a network-on-chip architecture for gigascale systems-on-chip. *IEEE Circuits and Systems Magazine*, 2004.

[7] N. Bindal, T. Kelly, N. Velastegui, and K.L. Wong. Scalable sub-10ps skew global clock distribution for a 90nm multi-ghz ia microprocessor. In *International Solid-State Circuits Conference*, volume 1, pages 346–498, 2003.

[8] T. Bjerregaard. The MANGO clockless network-on-chip: Concepts and implementation. *PhD Thesis*, 2005.

[9] T. Bjerregaard and J. Sparso. A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip. In *DATE*, 2005.

[10] Paul Bogdan, Tudor Dumitras, and Radu Marculescu. Stochastic communication: A new paradigm for fault-tolerant networks-on-chip. *VLSI Design*, 2007.

[11] E. Bolotin, I. Cidon, Ran Ginosar, and A. Kolodny. QNoC: QoS architecture and design process for network on chip. *Journal of Systems Architecture*, 2004.

[12] William J. Dally and Brian Towles. Route packets, not wires: On-chip inteconnection networks. In *DAC*, 2001.

[13] K. Goossens, J. Dielissen, and A. Radulescu. Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Design & Test of Computers*, 2005.

[14] Kees Goossens and Andreas Hansson. The aethereal network on chip after ten years: Goals, evolution, lessons, and future. In *DAC*, 2010.

[15] A. Hansson, M. Coenen, and K. Goossens. Channel trees: reducing latency by sharing time slots in time-multiplexed networks on chip. In *CODES+ISSS*, 2007.

[16] A. Hansson and K. Goossens. Trade-offs in the configuration of a network on chip for multiple use-cases. In *NOCS*, 2007.

[17] A. Hansson, K. Goossens, M. Bekooij, and Jos Huisken. CoMP-SoC: A template for composable and predictable multi-processor system on chips. *ACM Trans. on Design Automation of Electronic Systems*, 2009.

[18] A. Hansson, K. Goossens, and A. Rădulescu. A unified approach to mapping and routing on a network-on-chip for both best-effort and guaranteed service traffic. *VLSI Design*, 2007.

[19] A. Hansson, M. Subburaman, and K. Goossens. aelite: A flit-synchronous network on chip with composable and predictable services. In *DATE*, 2009.

[20] N. Kavaldjiev, G.J.M. Smit, P.G. Jansen, and P.T. Wolkotte. A virtual channel Network-on-Chip for GT and BE traffic. In *ISVLSI*, 2006.

[21] A. Laffely, Jian Liang, R. Tesseir, and W. Burleson. Adaptive system on a chip (ASOC): a backbone for power-aware signal processing cores. In *ICIP*, 2003.

[22] Jian Liang, Sriram Swaminathan, and Russell Tessier. aSOC: A scalable, single-chip communications architecture. In *PACT*, 2000.

[23] D. Liu, D. Wiklund, E. Svensson, O. Seger, and S. Sathe. SoCBUS: the solution of high communication bandwidth on chip and short TTM. In *RTECC*, 2002.

[24] Zhonghai Lu, Bei Yin, and Axel Jantsch. Connection-oriented multicasting in wormhole-switched networks on chip. In *Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*. IEEE Computer Society, 2006.

[25] Zhonghai Lu *et al.* Slot allocation using logical networks for TDM virtual-circuit configuration for network-on-chip. In *ICCAD*, 2007.

[26] D. Ludovici, A. Strano, G. N. Gaydadjiev, and D. Bertozzi. Mesochronous noc technology for power-efficient gals mpsocs. In *Proceedings of the Fifth ACM Interconnection Network Architecture, On-Chip Multi-Chip Workshop (INA-OCMC)*, pages 27–30, January 2011.

[27] R. Manevich *et al.* Benoc: A bus-enhanced network on-chip for a power efficient cmp. *Computer Archchitecture Letters*, 2008.

[28] T. Marescaux, B. Bricke, P. Debacker, V. Nollet, and H. Corporaal. Dynamic time-slot allocation for QoS enabled networks on chip. In *ESTIMedia*, 2005.

[29] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip. In *DATE*, 2004.

[30] M. Moadeli, P. Maji, and W. Vanderbauwhede. Quarc: A High-Efficiency network on-Chip architecture. In *AINA*, 2009.

[31] Christian Paukovits and Hermann Kopetz. Concepts of switching in the Time-Triggered Network-on-Chip. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 120–129, 2008.

[32] A. Radulescu, J. Dielissen, S. Gonzalez Pestana, Om Prakash Gangwal, E. Rijpkema, P. Wielage, and K. Goossens. An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network programming. *Trans. on CAD of Integrated Circuits and Systems*, 2005.

[33] F.A. Samman *et al.* Adaptive and deadlock-free tree-based multicast routing for networks-on-chip. *Transactions on VLSI*, 2010.

[34] M. Schoeberl. A Time-Triggered Network-on-Chip. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pages 377 –382, August 2007.

[35] C. Schuck, S. Lamparth, and J. Becker. artNoC - a novel Multi-Functional router architecture for organic computing. In *FPL*, 2007.

[36] Radu Stefan. *Resource Allocation in Time-Division-Multiplexed Networks on Chip*. PhD thesis, Delft University of Technology, 2012.

[37] Radu Stefan and Kees Goossens. A TDM slot allocation flow based on multipath routing in NoCs. *Microprocessors and Microsystems (MICPRO)*, 2011. Elsevier.

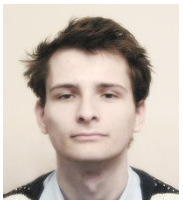
[38] Radu Stefan, Ashkan Beyranvand Nejad, and Kees Goossens. Online allocation for contention-free-routing nocs. In *Proceedings of the 2012 Interconnection Network Architecture: On-Chip, Multi-Chip Workshop (INA-OCMC)*, 2012.

[39] S. Stergiou, F. Angiolini, S. Carta, L. Raffo, D. Bertozzi, and G. D Micheli. Xpipes lite: a synthesis oriented design library for networks on chips. In *DATE*, 2005.

[40] P.T. Wolkotte, G.J.M. Smit, G.K. Rauwerda, and L.T. Smit. An Energy-Efficient reconfigurable Circuit-Switched Network-on-Chip. In *IPDPS*, 2005.

[41] P.T. Wolkotte, G.J.M. Smit, G.K. Rauwerda, and L.T. Smit. An energy-efficient reconfigurable circuit-switched network-on-chip. In *IPDPS*, 2005.

[42] C.A. Zeferino and A.A. Susin. SoCIN: a parametric and scalable network-on-chip. In *SBCCI*, 2003.



Radu Stefan was born in Braşov, Romania on the 6^{th} of May 1981. He obtained his PhD degree from the Delft University of Technology in 2012 with a thesis on resource allocation in TDM NoCs and his Master of Science and Bachelor of Engineering degrees from the Transilvania University of Braşov in 2006 and 2005 respectively. He has worked at Splash Software Braşov on high-end video codecs performing development of rate-control algorithms and codec parallelization. He participated in the Scientific Committee of the Romanian National Olympiad in Informatics in 2001 and 2004, and the Balkan Olympiad in Informatics in 2011. In 2011-2012 he worked as a researcher at the Eindhoven University of Technology.

Anca Molnos received the MSc degree in computer science from the "Politehnica" University of Bucharest, Romania and the PhD degree in computer engineering from the Delft University of Technology, The Netherlands, in 2001 and 2009, respectively. The topic of her PhD thesis was cache management for embedded multi-processor systems executing multimedia applications. Between 2006 and 2009 she worked at NXP Semiconductors, The Netherlands, on low-power multi-processors and distributed real-time systems. Currently she is a Post-Doctoral researcher with the Delft University of Technology, The Netherlands. Her research interests include composable, predictable multi-core embedded platforms and embedded multi-core resource management for low-power, quality of service, and dependability.

Kees Goossens received his PhD in Computer Science from the University of Edinburgh in 1993 on hardware verification using embeddings of formal semantics of hardware description languages in proof systems. He worked for Philips/NXP Research from 1995 to 2010 on networks on chip for consumer electronics, where real-time performance, predictability, and costs are major constraints. He was part-time professor at Delft university from 2007 to 2010, and is now professor at the Eindhoven university of technology, where his research focusses on composable (virtualised), predictable (real-time), low-power embedded systems. He published 2 books, 100+ papers, 9 patents, and 22 patent applications.