Channel Trees: Reducing Latency by Sharing Time Slots in Time-Multiplexed Networks on Chip

Andreas Hansson
Electronic Systems Group
Eindhoven University of Technology
Eindhoven, The Netherlands
m.a.hansson@tue.nl

Martijn Coenen
Corporate Research Department
NXP Semiconductors
Eindhoven, The Netherlands
martijn.coenen@nxp.com

Kees Goossens
Corporate Research Department
NXP Semiconductors
Eindhoven, The Netherlands
kees.goossens@nxp.com

ABSTRACT

Networks on Chip (NoC) have emerged as the design paradigm for scalable System on Chip communication infrastructure. A growing number of applications, often with firm (FRT) or soft real-time (SRT) requirements, are integrated on the same chip. To provide time-related guarantees, NoC resources are reserved, e.g. by non-work-conserving time-division multiplexing (TDM). Traditionally, reservations are made on a per-communication-channel basis, thus providing FRT guarantees to individual channels. For SRT applications, this strategy is overly restrictive, as slack bandwidth is not used to improve performance.

In this paper we introduce the concept of *channel trees*, where time slots are reserved for sets of communication channels. By employing work-conserving arbitration within a tree, we exploit the inherent single-threaded behaviour of the resource at the root of the tree, resulting in a drastic reduction in both average-case latency and TDM-table size. We show how channel trees enable us to halve the latter in a car entertainment SoC, and reduce the average latency by as much as 52% in a mobile phone SoC. By applying channel trees to an H264 decoder SoC, we increase processor utilisation by 25%.

Categories and Subject Descriptors: B.4.3 [Input/Output and Data Communications]: Interconnections

General Terms: Algorithms, Design, Performance

Keywords: System-on-Chip, Network-on-Chip, Quality-of-Service, Time-Division-Multiplexing

1. INTRODUCTION

Systems on Chip (SoC) grow in complexity with an increasing number of processors, memories and accelerators integrated on a single chip. These heterogeneous high-complexity chips are programmable and integrate a rich set of applications [6], e.g. PDA phones with MP3 players, cameras, radios and gaming.

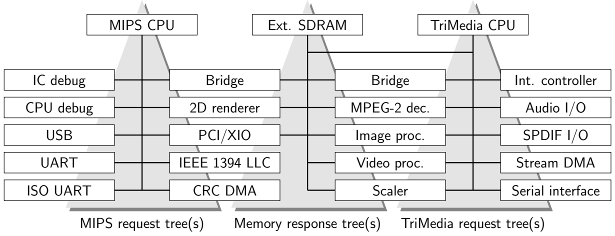


Figure 1: The Viper set-top box SoC [6].

The individual *applications* have different Quality of Service (QoS) requirements, e.g. bandwidth and latency constraints that the communication infrastructure must accommodate efficiently. Firm real-time (FRT) applications are not allowed to miss any deadlines, due to e.g. standardisation requirements or steep quality reduction in the case of misses. For this type of applications, designers use FRT analysis techniques [12] and worst-case dimensioning. Soft real-time (SRT) applications, e.g. an MPEG decoder, can tolerate deadline misses with only a modest quality degradation and SRT analysis techniques are used for average-case performance analysis [2]. For both FRT and SRT techniques, it is crucial that the properties established at the application level are maintained during system integration, which is only possible with a *composable* architecture [12].

Networks on Chip (NoC) have emerged as the design paradigm for scalable on-chip communication architectures [5] and can provide composability by means of QoS guarantees [8, 14], bounding the maximum latency and the minimum throughput. The guarantees are traditionally implemented by reserving resources, e.g. links and buffers, to individual *channels* [3, 13, 16] or *connections* [14], where a connection comprises both a request and a response channel, from master to slave and back.

One common strategy to provide QoS is *time-division multiplexing* (TDM) [1, 14, 16, 20]. While offering FRT guarantees, this scheme is non work-conserving, resulting in average-case latencies that are close to the worst case even in a lightly loaded system [3, 21]. Average and worst case are even closer due to a potentially large difference in communication granularity between the NoC and the Intellectual Property (IP) cores. The primary example is an SDRAM, as exemplified in Figure 1, where one burst (64-256 bytes) requires multiple TDM revolutions since the NoC granularity is an order of magnitude smaller (8-16 bytes). This is illustrated in Figure 2, where a burst, produced for the channel c_B, takes four iterations of the TDM schedule to finish. As a

CODES+ISSS'07, September 30–October 3, 2007, Salzburg, Austria.
Copyright 2007 ACM 978-1-59593-824-4/07/0009 ...$5.00.

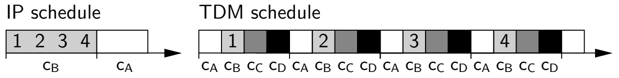

IP schedule
TDM schedule
1 2 3 4
c_B
c_A
1
2
3
4
c_A c_B c_C c_D c_A c_B c_C c_D c_A c_B c_C c_D c_A c_B c_C c_D

Figure 2: Service granularity mismatch.

result of the finer granularity, the burst is spread out in time, giving rise to an average latency close to the worst case.

As we shall see, the effect on average latency can be mitigated when channels form a tree around a shared source or destination. This occurs naturally at a shared slave, exemplified by the external SDRAM in Figure 1, but also around the CPUs, responsible for the peripherals and accelerators belonging to their respective task domain.

The main contribution of this paper is the *channel tree* link-reservation scheme for channels that form tree structures in a time-multiplexed NoC. By: 1) sharing time slots between multiple SRT channels, and 2) employing work-conserving scheduling within the tree, we exploit the inherent serialisation of channels diverging from a single-threaded source. With only a few percents addition to the NoC area, we show how channel trees: 1) mitigate discretisation effects on bandwidth requirements and reduce the TDM-table size (important for both SRT and FRT) by a factor two in a car entertainment SoC, and 2) reduce average latency (important for SRT) by 52% for a mobile phone SoC. By applying channel trees to an H264 decoder SoC, we halve the amount of stall cycles and increase CPU utilisation by 25%.

The remainder of the paper is structured as follows. We start by introducing related work in Section 2. Next, the problem domain is described in Section 3. The concept of channel trees is presented in Section 4, followed by a description of how to apply the technique in an existing NoC architecture in Section 5. Finally, experimental results are shown in Section 6 and conclusions are drawn in Section 7.

2. RELATED WORK

Silicon BackPlane reserves time-slots for masters rather than communication channels [20]. Thus, no real-time guarantees can be given to the individual channels with potentially different target destinations.

Connection-oriented guarantees, implemented by globally scheduling all connections, is provided by NuMesh [18] (for parallel processing), Nostrum's looped containers [14], aSOC's global scheduling [13], and Æthereal's contention-free routing [16]. All these works reserve network time-slots on a per-connection, or even per-channel, basis and use fixed schedules with non-work-conserving arbitration. As a consequence, slack is not distributed across channels and the TDM table grows at least with the number of channels.

In [7, 17], the granularity issue of frame-based arbitration is solved by multiplexing several channels on a single slot. A non-work-conserving multi-rate control mechanism, based on a network processor, is outlined in [17]. In our context, the arbitration speed is higher and buffering is more constrained, demanding solutions that exploit the tighter coupling between NoC components. The latter is identified in [7], but the work does not detail how to couple the concept to time-slot allocation and does not apply the channel multiplexing to any real-world systems.

Multiplexing of independent channels in a TDM-based interconnect is introduced in [9] by scheduling individual messages. A similar concept is used in [19] where mutually exclusive channels are allowed to use the same time-slots through short-term reconfiguration. While this approach reaches high utilisation, it requires a detailed knowledge of message injection times and sizes, which is not typically known at design time, in particular for SRT applications.

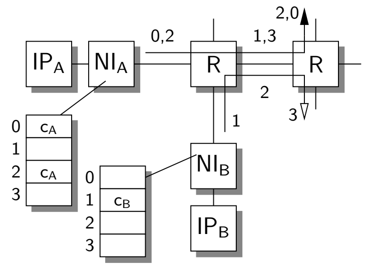


Figure 3: Contention-free routing.

In this paper we propose a scheme where time slots are reserved for sets of communication channels, for which only the same source or destination is required. No prior knowledge is needed about packet injection times as arbitration is done at *run time* while preserving performance guarantees. Different *applications are still isolated* by non-work-conserving TDM arbitration, whereas average-case performance is significantly improved for the individual SRT applications by using a work-conserving arbitration within the trees.

3. PROBLEM DESCRIPTION

There are several possible hardware solutions that implement the regulation of traffic. In this work we look at the particular implementation of Æthereal, where throughput and latency guarantees are provided to the channels by using TDM-based *contention-free routing* [16]. The injection of *flow control units* (flits) is regulated by a table in the NI such that no flits contend, as illustrated in Figure 3. In the figure, there are two IP cores, IP_A and IP_B, that communicate over the network of routers (R). It is assumed that IP_A needs one communication channel, c_A, that has two slots reserved in the TDM table, and that IP_B needs one channel, c_B, with only one slot reserved. The paths of channel c_A and c_B are indicated by a solid and open-headed arrow, respectively. Channel c_A has slots 0 and 2 reserved in the table, and channel c_B has slot 1 reserved in the table. The TDM table has the same period throughout the NoC, in this case 4, and works on a fixed slot size.

Contention-free routing is non work-conserving, resulting in an average latency that is relatively high. Although the time in the router network is minimal (no contention), flits have to wait for their slots in the NIs, even if there are no other flits in the network. Clearly, the larger the TDM table and the fewer slots are reserved, the higher the waiting latency as the distance between reserved slots increases.

In many cases, latency is an important characteristic of on-chip communication. Especially for processors, which stall until a read request to the memory is completed, the average latency to memory has a large effect on the cycles they effectively compute, and hence on total system performance.

The issue becomes even more apparent when a big burst of data is sent, which, due to the mismatch between the granularity of the TDM table and the granularity of the burst, requires multiple iterations of the table to complete. Figure 4 shows the impact of different burst sizes for a 166

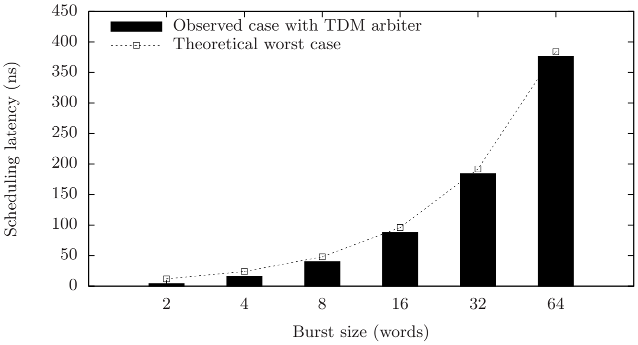


Figure 4: Latency added due to network arbitration.

MHz DDR SDRAM memory controller, in theory capable of producing a word every 3 ns. This is matched by the NoC that runs on a 500 MHz clock frequency and has one third of the total bandwidth assigned to the channel in question. The line shows the theoretical maximum latency contribution of the NoC arbitration, whereas the bars indicate the average latency observed in simulation with a cycle-accurate NoC model employing TDM. In SRT applications, that are able to use slack, the small difference between average and worst-case latency may lead to unacceptable utilisation of e.g. the processors.

The effect can be mitigated by increasing the NoC slot size. This, however, leads to larger NoC buffers, greatly increasing the NoC area [4]. Decreasing the SDRAM granularity, on the other hand, leads to an unacceptable memory efficiency (as low as 40%), as the command overhead grows [11].

For frame-based schedulers, such as TDM, low bandwidth channels, in combination with limited allocation granularity, are a source of over-dimensioning [15] and can lead to under-utilisation of resources [17, 21]. Figure 5(a) illustrates an example in which the IP core IP_R communicates with IP_A, IP_B, IP_C and IP_D through four channels. The channels c_A, c_B, c_C, and c_D require $\frac{1}{40}$, $\frac{2}{40}$, $\frac{3}{40}$, and $\frac{4}{40}$ of the link capacity respectively. Due to the granularity, the TDM table requires at least 40 slots. It is possible to use fewer slots at the expense of over allocation, e.g. use a table with only four slots and assign a single slot to each channel.

4. CHANNEL TREES

Traditionally slot allocation is only correct if every slot of a link is allocated to at most one channel. Channel trees exploit that with an additional level of scheduling in the source NI, it is possible to multiplex several channels on a single slot, as depicted in Figure 5(b). In the example, the slots that were previously reserved for c_A, c_B, c_C and c_D are now collectively reserved for the set that comprises all four channels. Hence, each of the channels may access the network in slots 0-9.

Note that the aggregation of channels into trees is orthogonal to the concept of connections as the latter always comprise one request and one response channel [16]. The only requirement on the constituent channels is that their usage of the network links is contention free, either by construction, as in a diverging tree, or externally enforced, as must be done for a converging tree.

Table 1: Latency (ns) for the channels in Figure 5.

Channel	With trees			Without trees		
	Min.	Avg.	Max.	Min.	Avg.	Max.
c_A	38	40	42	120	128	134
c_B	40	42	46	122	130	138
c_C	44	46	48	124	132	140
c_D	50	52	54	130	152	164

4.1 Advantages

When many channels diverge from a single-threaded resource that does not interleave transactions, channel trees significantly reduce the average latency by allowing more freedom in the scheduling between the constituent channels. This is especially important for shared-memory-centric architectures, with processors that need a low average latency to achieve a good performance. Table 1 shows the observed latency, with and without channel trees, for the system depicted in Figure 5(b). Four response channels, with bandwidth requirements of 20, 40, 80 and 160 Mbyte/s respectively, diverge from a slave IP_R and as seen in the table, the observed latency is effectively reduced by two thirds.

It is important to note, that while enabling performance improvements for SRT applications, the channel trees are isolated from each other and other individual channels. Thereby, the trees cover the entire spectrum from reservations per channel [13,14,16,18] to reservations per port [20].

In addition, by using channel trees, the size of the TDM table can be reduced. With the introduction of $\{c_A, c_B, c_C, c_D\}$ in Figure 5(b), it is possible to redistribute the ten slots equidistant over the TDM table. This distribution not only minimises the worst-case waiting time for a slot but also enables a reduction of the table size by a factor of ten. The reduced table has only four slots with one allotted to the newly introduced channel tree.

4.2 Implications

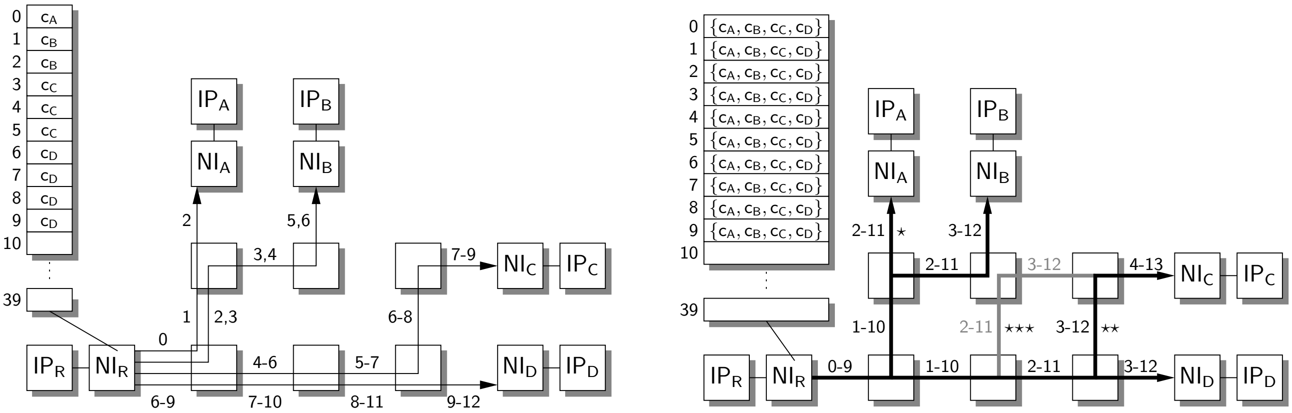
An additional level of arbitration is required to ensure that the constituent channels of the tree are scheduled so that real-time guarantees can be given to them [17]. This is elaborated on and exemplified in Section 5.1.

Channel trees trade average latency for over-allocation in the branches. Consider for example the ingress link of NI_A (marked by a $\star$) in Figure 5. Without the channel tree, one slot is reserved while ten slots are required after the aggregation. Clearly, the amount of over-allocation grows with the distance from the root of the tree. Let $\pi(c)$ denote the set of links that constitute the path of a channel c, and $s(c)$ the set of slots reserved to c on the ingress link of the source NI. Without channel trees, a total of $\sum_{\forall c} |\pi(c)|\,|s(c)|$ slots are reserved. Should a tree t be used, then a maximum of $\left|\bigcup_{\forall c \in t} \pi(c)\right| \times s(t)$ slots are reserved for the constituent channels. In Figure 5, this corresponds to $3 \times 1 + 4 \times 2 + 5 \times 3 + 4 \times 4 = 42$ and $10 \times 10 = 100$ slots respectively.

The potential over-allocation can be mitigated by combining the channels into more than one tree, e.g. $\{c_A, c_B\}$ and $\{c_C, c_D\}$. Thus, there is less over-allocation in the network, but the average latencies increase.

5. ADAPTATIONS

In order to apply the channel trees to a NoC design, adaptations, although minor, are needed both in the architecture and in the allocation of time slots.



(a) No sharing

(b) Full sharing

Figure 5: Diverging channel tree on a 2 × 3 mesh.

5.1 Architecture

When channels are combined into trees, a mechanism is required to schedule the data sequentially onto the network. A tree of *converging* channels requires explicit contention resolution. In the simplest case, with a master at the root of the tree, the order can be decided by only allowing one outstanding request. This is, however, at the cost of pipelining and parallelism, and in the general case a second arbitration scheme must be added on top of the TDM. The latter can be implemented by for example letting IP_R provide extra control information, embedded in the requests, that IP_A, IP_B, IP_C, and IP_D use to determine when to inject data in the converging tree, i.e. a hand-shake protocol. Such a mechanism lies outside the scope of this paper.

For a *diverging* tree, which is the focus of this paper, the contention resolution requires only minor adaptations of the source NI. Figure 6 shows the relevant parts of the NI architecture [16] for an instance with six channels, c_A to c_F. There are two components of the NI that are affected by the introduction of the trees. First, the TDM table contains both individual channels and *tree identifiers*. Second, in addition to the TDM table, an additional level of arbitration is required to choose between the channels within each tree.

Any channel that is to be scheduled must: 1) belong to the set of channels having reserved the TDM slot 2) have data, and 3) have end-to-end flow control credits available [16]. If the time slot belongs to a tree, the tree arbiter selects one of the eligible channels.

A tree arbiter is a stand-alone module with a general interface that is oblivious of the actual arbitration mechanism. At run time, the arbiter is configured with the set of channels that belong to the tree and any settings specific for the arbiter, e.g. the length of time windows and budgets for the individual channels. Typically, the arbiter is composed of a rate regulator, that shapes the traffic such that *no channel can corrupt the guarantee of another* [17], and a work-conserving scheduler.

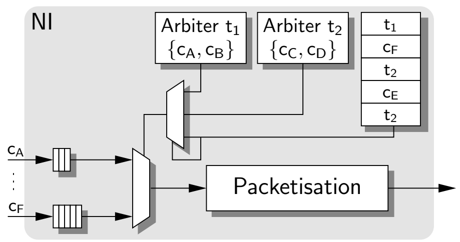


Figure 6: Adapted NI architecture.

Depending on the goals of the design, any scheduling algorithm, e.g. first-come first-served (FCFS) or round-robin [21], can be used, as long as it is combined with a rate regulator. In our experiments we use a simple round-robin arbiter with an enforced maximum packet size. This arbiter serves as both rate regulator and scheduler, while occupying less than 10% of the NI area. Note that under these assumptions, arbitration within the tree is *starvation free* and *guarantees can still be given to the individual channels.*

Irrespective of whether channel trees are used or not, each channel has a separate NI queue. Thus, there is no possibility of inter-channel blocking, and the trees can be programmed at run-time to adapt to specific application needs.

5.2 Allocation algorithm

We base this work on an existing algorithm [10] by adapting the order in which channels are traversed, the conditions on which paths and time slots are selected, and how reservations are made. The goal is still to find paths and time slots such that the constraints of all channels are satisfied.

The outmost loop of our allocation algorithm is iterating over all trees (also including single-channel trees), ordered on their total bandwidth requirements to reduce bandwidth fragmentation, conserve resources and give precedence to channels with a more limited set of paths [10]. Together with the appropriate path-selection cost function, this traversal order also asserts that the overlap between the individual channels is maximised, e.g. by choosing the path $\star\star$ instead of $\star\star\star$ for channel c_C in Figure 5(b).

For every channel, a constrained least-cost path is selected such that throughput and latency guarantees are fulfilled. When paths are evaluated, for every hop, the set of available time slots is used to prune the search space and determine path cost. The path selection algorithm therefore starts by determining which time slots are available on the first link, between the source NI and the router. This is done by pruning all slots occupied by other channels in *any* of the already

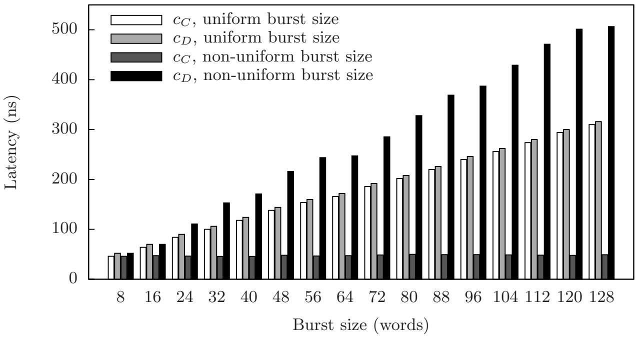


Figure 7: Latency scaling with burst size.

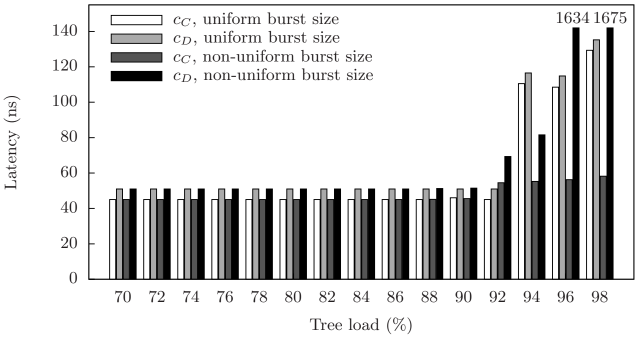


Figure 8: Latency scaling with bandwidth.

allocated branches. Consider for example the allocation of c_C in Figure 5(b), with c_A and c_B already allocated. Only the slots that are available on both paths are considered for c_A. Then, when deciding on a path for c_A, the link cost is based on how much a hop reduces the set of available slots. Thereby, links that are already occupied by the tree are preferred. Finally, resources are reserved for the whole tree, spanning all the branches.

6. EXPERIMENTAL RESULTS

To evaluate the performance of our methodology, we apply it to three SoC designs and a range of synthetic benchmarks.

6.1 Synthetic benchmarks

We start by evaluating how the burst size and bandwidth requirement of the channels in a tree affect the performance benefits. We evaluate this on a system similar to what is depicted in Figure 5, with a 500 MHz NoC operating frequency and 32 bit word width. The average latencies are measured during a simulation time of 3×10^6 clock cycles in a cycle-accurate SystemC simulator of Æthereal, using traffic generators to mimic IP behaviour. Unless indicated otherwise, the maximum packet size is 4 flits, and the four channels have the bandwidth requirements 100, 200, 300 and 400 Mbyte/s. We apply the channel tree concepts by allocating all four channels to a single tree.

Figure 7 shows how an increase in burst size affects the average latency for two different scenarios. First, the burst size is increased *uniformly* across all the channels, starting from 8 words and going up to 128 words. Second, referred to as *non-uniform*, only the burst size of c_D is increased. With a uniform burst size, the latency grows linearly for both c_C and c_D. The channels c_A and c_B are not shown, but have a similar behaviour. The latency increase is due to the increased amount of data being delivered in a burst (one word at a time), and not due to the TDM waiting time. This is to be compared with the scheduling latency introduced in the absence of channel trees, shown in Figure 4. In the non-uniform case, only c_D is affected by the increase in burst size, showing the isolation provided by the arbiter. The maximum packet size, here tuned for the constant burst size, can be increased to improve the performance of c_D at the expense of a slightly higher latency for the other channels.

Next, we investigate how the availability of slack bandwidth within the tree affects the average latency. In contrast to the relatively low load in Table 1, Figure 8 shows how the behaviour changes when the load within the tree approaches 100%, both when scaling the bandwidth uniformly, increasing all four channels equally much, and non-uniformly, increasing only the bandwidth of c_D. As seen, the availability of slack is not affecting average latency until the load goes above 90%. Hence, the tree's ability to serialise whole bursts from the IP is the main contributor to the improved performance, and the tree is very much beneficial also when the load is high. Note that in the non-uniform case, when only c_D is increased, c_C is hardly affected. Again, the channels c_A and c_B are not shown, but have a similar behaviour. When c_D goes above one fourth of the available bandwidth, the round-robin arbiter breaks down, leading to a severe increase in latency for this channel. Remedying this situation requires a more sophisticated arbiter [21].

6.2 Car entertainment

To evaluate the performance of our methodology on a real SoC design, we apply it to a multi-core architecture for in-car digital entertainment [15]. The system does not have a shared memory, but instead employs streaming communication. A large number of low bandwidth (less than 1 Mbyte/s) channels connect two DSP tiles with an I/O tile containing e.g. SPDIF in/out interfaces, DACs as well as ADCs, and USB connectivity. The I/O-related channels clearly dictate the minimum TDM-table size and we therefore create two trees to the involved DSPs. By introducing these trees, the TDM table is reduced from 19 to 8 slots. As a direct consequence, the register file containing the table is halved in each NI, and the average latency observed by the channels in the system diminishes by 24%.

6.3 Mobile phone SoC

Our next example is a mobile phone SoC design, featuring applications like telephony, storage, audio/video decoding, camera image encoding, image preview and 3D gaming. The architecture is similar to what is shown in Figure 1 with a total of 13 cores (27 ports distributed across an ARM, a TriMedia, two DSPs, a rendering engine etc.), one off-chip DDR SDRAM memory, one on-chip SRAM plus a number of peripherals. Communication is done via memory, running at 117 MHz with a word width of 64 bits. As the native word width of our NoC is 32 bits we choose to let the NoC run at double the memory frequency, 235 MHz, thus offering the same gross bandwidth.

We define four channel trees: 1) between the peripherals and the ARM, 2) from the ARM to the external memory, 3) from the TriMedia to the external memory and additionally,

Table 2: Cache-miss latency (TriMedia cycles).

Component		With trees	Without trees
Request	TDM arbitration	4	4
	NoC traversal	4	4
Memory	arbitration	19	19
	service time	32	32
Response	TDM arbitration	2	102
	NoC traversal	4	4
Total		65	165

4) a diverging tree for the remaining response channels from the external memory.

The reduction in time-slots is only minor, going from 16 without trees to 14 with trees. This is due to the many request channels that still converge at the shared memory, each requiring at least one slot. More noticeable, however, is a 52% reduction of the average latency for the channels in the four trees. In particular, the average read latency (round-trip) for the ARM and TriMedia is reduced from 565 and 470 ns to 230 and 215 ns respectively.

6.4 H264 decoding

In order to show the impact of average latency on processor utilisation, we evaluate a H264 decoder SoC. The SoC consists of a 166 MHz DDR SDRAM memory controller, a CPU that reads a H264 file from a storage subsystem and writes it to memory, a TriMedia TM3270 video processor (clocked at 350 MHz) that reads the encoded bitstream from memory and writes the decoded frames to memory, and finally a display controller that reads each of the frames from memory. In contrast with the previous examples, this SoC is evaluated using cycle-accurate models of the IPs.

The cache misses generated by the TriMedia, which are 128 byte requests, are sent over the NoC to the SDRAM. Since the memory controller is the sole point of contention, the responses from it form a diverging tree to which the channel tree concept is applied. Table 2 details the components of the total round-trip latency of a cache-miss, measured in TriMedia cycles.

Without trees the largest fraction of time is spent in the memory controller and waiting for a TDM slot on the response path, where only one out of four slots is allocated to the TriMedia. On the request path TDM arbitration does not add a significant latency, because a read request is very small (typically two words) and fits perfectly with the NoC granularity. When channel trees are used, the response messages are fully pipelined, as the NoC consumes the data produced by the memory immediately, without having to wait for TDM arbitration. The NoC introduces only the packetisation (the 2 cycles shown under TDM arbitration) and hop-delay latency on the response path. This improves the overall round-trip latency by almost 60%. Effectively the amount of stall cycles is halved, increasing the TriMedia's utilisation by 25%.

7. CONCLUSIONS AND FUTURE WORK

Networks on Chip must provide time-related guarantees in order to allow a composable design methodology, fitting both firm (FRT) and soft real-time (SRT) applications. Time-division multiplexing offers these guarantees, but its non-work-conserving nature conduces to high average latencies, which results in bad processor utilisation and possibly unacceptable performance for SRT applications.

In this paper we introduce the channel tree concept, where time slots are shared between a selected set of channels. When applied to a number of SoC designs, the average latency of the constituent channels decreases significantly, up to 60%, and the processor utilisation of an H264 decoder improves by as much as 25%. Since TDM is used between the channel trees, FRT and SRT applications can still be combined in the same system, providing full isolation and composability between the applications.

It remains an open issue to automatically identify suitable channels to join in trees. Similarly, the many optimisation problems that arise when allocating resources to trees rather than single channels are yet to be fully explored.

8. REFERENCES

[1] E. Beigne *et al.* An asynchronous NOC architecture providing low latency service and its multi-level design framework. In *Proc. ASYNC*, pages 54–63, 2005.

[2] M. Bekooij *et al.* Efficient buffer capacity and scheduler setting computation for soft real-time stream processing applications. In *Proc. SCOPES*, 2007.

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

[4] M. Coenen *et al.* A buffer-sizing algorithm for networks on chip using TDMA and credit-based end-to-end flow control. In *Proc. CODES+ISSS*, 2006.

[5] W. J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Proc. DAC*, 2001.

[6] S. Dutta *et al.* Viper: A multiprocessor SOC for advanced set-top box and digital TV systems. *IEEE Design and Test of Computers*, 2001.

[7] O. P. Gangwal. Id 693979 – network-on-chip environment and method for reduction of latency. Technical report, Royal Philips Electronics, 2005.

[8] K. Goossens *et al.* A design flow for application-specific networks on chip with guaranteed performance to accelerate SOC design and verification. In *Proc. DATE*, 2005.

[9] R. Guérin and A. Orda. Networks with advance reservations: The routing perspective. In *Proc. INFOCOM*, 2000.

[10] A. Hansson *et al.* A unified approach to constrained mapping and routing on network-on-chip architectures. In *Proc. CODES+ISSS*, 2005.

[11] JEDEC Solid State Technology Association. *DDR2 SDRAM Specification*, JESD79-2C edition, 2006.

[12] H. Kopetz. *Real-Time Systems: Design Principles for Distibuted Embedded Applications.* Kluwer Academic Publishers, 1997.

[13] A. Laffely. *An interconnect-centric approach for adapting voltage and frequency in heterogeneous system-on-a-chip.* PhD thesis, University of Massachusetts Amherst, 2003.

[14] M. Millberg *et al.* Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. In *Proc. DATE*, 2004.

[15] A. Moonen *et al.* A multi-core architecture for in-car digital entertainment. In *Proc. GSPx*, 2005.

[16] A. Rădulescu *et al.* An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network programming. *IEEE Trans. on CAD of Int. Circ. and Syst.*, 2005.

[17] D. Saha *et al.* Multi-rate traffic shaping and end-to-end performance guarantees in ATM networks. In *Proc. ICNP*, 1994.

[18] D. Shoemaker. *An Optimized Hardware Architecture and Communication Protocol for Scheduled Communication.* PhD thesis, Massachusetts Institute of Technology, 1997.

[19] S. Stuijk *et al.* Resource-efficient routing and scheduling of time-constrained network-on-chip communication. In *Proc. DSD*, 2006.

[20] D. Wingard and A. Kurosawa. Integration architecture for system-on-a-chip design. In *Proc. CICC*, 1998.

[21] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proc. IEEE*, 83(10), 1995.