Timing Organization of a Real-Time Multicore Processor

Martin Schoeberl, Jens Sparsø

Department of Applied Mathematics and Computer Science
Technical University of Denmark
Email: masca@dtu.dk, jsp@dtu.dk

***Abstract*—Real-time systems need a time-predictable computing platform. Computation, communication, and access to shared resources needs to be time-predictable. We use time division multiplexing to statically schedule all computation and communication resources, such as access to main memory or message passing over a network-on-chip. We use time-driven communication over an asynchronous network-on-chip to enable time division multiplexing even in a globally asynchronous, locally synchronous multicore architecture. Using time division multiplexing at all levels of the architecture yields in a time-predictable multicore processor where we can statically analyze the worst-case execution time of tasks.**

I. Introduction

A real-time system needs to deliver a result in time, that means it must deliver it before a given deadline. The deadline is usually constrained by the environment with which the system interacts. For example, a control loop operating at a certain frequency, demands one computation per iteration. Therefore, the deadline is the period of that control loop.

If the real-time system is part of a safety-critical system, it needs to be certified. Part of the certification is to show that all deadlines are met. As "testing shows the presence, not the absence of bugs" (Dijkstra in [1]), testing alone is not a valid approach to certify that all deadlines are met in a safety-critical system. Instead, formal methods are needed to statically determine the worst-case execution time (WCET) of tasks. To enable static WCET analysis the program needs to be analyzable (e.g., all loops need to be bounded) and the executing platform needs to be time-predictable [2].

This paper presents the timing organization of a time-predictable multicore processor that enables static WCET analysis. Figure 1 shows the T-CREST multicore processor [3]. It consists of several processing cores connected to two networks-on-chip (NoCs): (1) one core-to-core NoC [4] for message passing between processing cores and (2) a memory NoC [5], in a tree structure, to access a shared memory controller and main memory.

This paper is organized in 6 sections: The following section presents related work. Section III provides background on time-division multiplexing (TDM) and the use in the multicore processor. Section IV explores time-driven communication in a globally asynchronous, locally synchronous multicore processor. Section V shows how WCET can be bounded on the multicore platform. Section VI concludes.

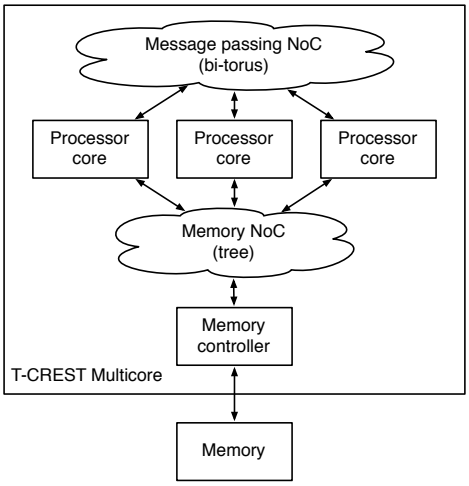

Fig. 1. The T-CREST multicore architecture with several processor cores connected to two NoCs: one for core-to-core message passing and one for access to the shared, external memory

II. Related Work

The Tilera processor TILE-Gx (now part of Mellanox Technologies Ltd.) is available with 36 or 72 processor cores (called tiles). Those cores are connected by the iMesh NoC consisting of five mesh networks [6]. The choice to have physically separate NoCs is intended to separate traffic to avoid interference. With T-CREST we also split between memory and core-to-core traffic to avoid interference. In contrast to the Tilera processor our focus is in time predictability instead of maximum bandwidth.

Epiphany is a high-performance energy-efficient multicore processor [7]. Epiphany is intended as an accelerator processor for real-time embedded systems. The multicore processor Epiphany is a distributed memory architecture. Each core contains 32 KB of local memory that is mapped into a global address space. The processors contain no caches. Accesses to memory of a remote core is performed over a NoC. The NoC is organized as a mesh and favors writes over reads, as writes are posted writes where the processor does not need to wait for the write to finish. Packets are single word long and routing is performed in a single cycle per hop. A second NoC is dedicated for read responses and a third NoC supports

off-chip traffic, e.g., external shared memory. In contrast to Epiphany our multicore processor contains a NoC with explicit support for message passing. Processor-local memories are only accessible from the local processor.

Æthereal [8] is a NoC that uses TDM where slots are reserved to allow a block of data to pass through the NoC router without waiting or blocking traffic. A credit-based flow control is applied for end-to-end control. Guaranteed services are combined with best effort routing to utilize unreserved resources. aelite, a light version of Æthereal, only offers guaranteed services, resulting in a simpler router design [9]. In the latest version of aelite, called dAElite [10], the static routing tables are in the routers to support multicast routing. In contrast to the Æthereal family of NoCs our NoC implements TDM arbitration from end-to-end. I.e., access to the scratchpad memory (SPM) with a DMA controller is scheduled with the NoC TDM schedule.

Paukovits and Kopetz use a time-triggered NoC for the time-triggered system-on-chip (TTSoC) architecture [11], which has been used in the GENESYS multiprocessor system-on-a-chip [12]. The aim of GENESYS is to provide higher level of abstractions by core services such as global time, NoC based communication, configuration, and execution control. The main difference to other NoC designs is the absolute time format, which is *not* directly related to the clock frequency. The macro tick is a power of two fraction of a second and the basis for the TDM slotting. The idea behind this time format is a good integration with off-chip versions of time-triggered networks. In contrast to the TTSoC, our TDM schedule uses a common time base established by mesonchronous clocking of the network interfaces. Therefore, we can schedule communication at clock cycle granularity, even without the need of a global system clock.

III. Time-Division Multiplexing

Access to shared resource, such as the shared NoC or shared memory, needs arbitration. The result of this arbitration influences the execution time. If this arbitration is dynamic, different tasks can interfere. When using time for arbitration, called time-division multiplexing (TDM), there is no interference between tasks executing on different cores. Using time as arbitration mechanism is known as TDM or as time-triggered architecture [13]. The main benefit of time based arbitration is that the arbitration decision is performed offline, which results in a static schedule. This static schedule has two main advantages: (1) it is time predictable, as the timing and the time to wait for an access slot can be bounded and (2) the hardware for enforcing the static schedule is simpler and scales better. The later allows, under the assumption that all clients have a common notion of time, to have distributed arbitration.

TDM arbitration is simple. Simple systems are good for safety-critical systems as it is easier to analyze the WCET and it is easier to show that the analysis is correct. TDM arbitration results in a static schedule. No dynamic scheduling decisions need to be done and in the case of multiple arbitration points, such in a NoC, when those TDM arbitration points are coordinated, no buffering and no credit based flow control is needed. This considerably simplifies the hardware and reduces the hardware resources.

In our implementation of the message passing NoC and the memory NoC we use distributed arbitration. The decision when a time slot is available and a request can be submitted to the NoC is a local decision.

One known downside of TDM based arbitration is that this scheme is not what is called "work conserving". That means when a client has no need to access the shared resource in its slot, this slot is not used. This is considered by some a "waste of resources". However, implementing work conserving, but still time-predictable arbitration, does not scale very well and it does not help for tight WCET bounds. For example, implementing a true round-robin arbiter for a memory arbiter needs to look at all clients' requests and perform the arbitration decision in the same cycle as the request shall be issued. For hard real-time systems, where WCET is of utmost importance, there is no benefit in trying to use unused slots.

Furthermore, when the resources are plenty, as the bandwidth in the NoC, having empty slots is not an issue. On a highly congested resource, such as the single shared memory, clients are usually memory bound and most of the time there is an outstanding request for an upcoming access slot.

TDM is an overarching theme for the timing organization in our architecture as we use it at several levels:

- At the message passing NoC
- In the network interface (NI) for the NoC
- At the memory NoC
- For DRAM refresh

IV. Timing Organization

Using TDM, the Argo NoC [4] provides virtual end-to-end circuits supported with functionality that can transfer a block of data from the local SPM in the source processor to the local SPM in the target processor. This functionality can be used to implement message passing or other higher level communication primitives.

Argo uses source routing and wormhole switching and due to the static TDM schedule packets never collide. As a result, a router is a simple pipelined 5-ported crossbar that implements the switching. Such a data-flow style circuit is easily and efficiently implemented using asynchronous techniques [14].

With the choice of source routing and wormhole switching follows that the NIs alone are responsible for implementing the TDM mechanism. This include storing and executing the TDM schedule. Details about the NI can be found in [15]. Briefly, the source end of every virtual circuit is supported by a DMA controller that is activated according to the TDM schedule. As the payload of a packet is only a few words, the transfer of larger blocks of data (i.e., messages) requires a sequence of TDM schedule periods. As a result, traffic on different virtual circuits originating from the same processor core is transmitted in an interleaved fashion.

The combination of clockless asynchronous routers and clocked NIs (and processor cores) result in an attractive and

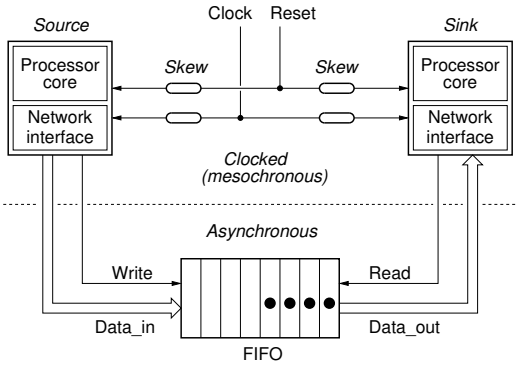

Fig. 2. A model of a virtual circuit connecting two processors. The asynchronous ripple FIFO represents the path through the NoC that packets sent across the virtual circuit follow.

efficient globally asynchronous, locally synchronous timing architecture that can tolerate some (clock)skew among the processor cores, while still offering sufficient global synchrony to implement TDM.

Figure 2 illustrate the principle showing two processors connected by an asynchronous ripple FIFO. The two processors (source and a sink) operate using the same clock. If the FIFO is initialized to be half full then the structure allows the two processors to operate with some skew. At a first glance this is like a conventional FIFO-based clock-domain crossing circuit. However, the implementation is fundamentally different as there are no explicit FIFOs in the design. The FIFO in Figure 2 is a very coarse abstraction representing the entire network of pipelined asynchronous routers and links, and it is the self-timed ripple-behavior of these that provide time elasticity equivalent to that of a FIFO.

The skew between source and sink may be unknown and varying over time. The design rests on the following fundamental assumptions: (i) the FIFO must be initialized half full, (2) the clock skew must be bounded, and (3) the network of pipelined asynchronous routers and links must be able to operate faster than the clocked source and sink. With these assumptions, the flow control signals (empty on the read port of the FIFO and full on the write port) can be ignored. This scenario is identical to the STARI principle introduced in [16].

To fully understand and analyze the operation of the Argo NoC it can be modeled as a mesh of small ripple-FIFOs connected by crossbar switches, where each crossbar switch operate like a transition in a Petri net: collecting one token from every input (place) and emitting one token to every output (place). The interesting point is that from this local synchronization among the ports in the individual routers, and from the flow of tokens among routers, emerges sufficient global synchrony to implement TDM. The details are beyond the scope of this paper. The interested reader is referred to the following papers for in-depth details [17], [18]. Briefly, clock skew has the effect of filling or draining a FIFO relative to their initial states, and this results in a slowdown of the FIFOs. For a sufficiently large clock skew the assumption that a FIFO is capable of operating faster than the clock no longer holds. In [18] we show that a skew of several clock-cycles can easily be absorbed.

A token in the FIFO, shown as a dot in Figure 2, can be understood as a container that may contain a word of a packet. Whether it contains a flit/word is indicated by a valid-bit in much the same way as a valid bit would be needed in clocked design. Therefore, a token in the Argo NoC is equivalent to a clock cycle in a clocked circuit.

V. Worst-Case Execution Time Analysis

To derive save upper bounds on the execution time of tasks, the WCET, we need to analyze the program on a concrete processor. In the T-CREST multicore processor we use as processing core a processor called Patmos [19], which is optimized to support WCET analysis. Patmos is supported by the standard industrial WCET analysis tool aiT [20] the research WCET analysis tool platin [21]. To provide time-predictable access time of main memory accesses in a multicore architecture, we use TDM for the main memory arbitration. Furthermore, for providing end-to-end WCET we need to determine the maximum latency of a message traveling on the message passing NoC.

For both, the message passing NoC and the memory NoC, we can derive the worst-case time for a single transaction as

$$T_{trans} = T_{wait} + T_{rw} + T_{NoC}$$

where T_{wait} is the worst case waiting time for the TDM slot to arrive, T_{rw} is the time for a read or write action at the source, and T_{NoC} is the time spent to traverse the NoC. T_{wait} depends on the phasing of the request relative to the TDM schedule and is a variable time, whereas T_{rw} and T_{NoC} are constant. The worst case of T_{wait} is when a request just missed its own slot by a single clock cycle. For a TDM schedule with N slots, a slot length of s clock cycles, and a clock period of t_{clk} the worst-case waiting time is $T_{wait} = (N \cdot s - 1) \cdot t_{clk}$.

The worst-case memory access time is

$$T_{mem} = ((N \cdot s - 1) + s + L_{NoC}) \cdot t_{clk}$$

where N is the number of TDM slots, s the slot length in clock cycles, L_{NoC} the pipeline latency, and t_{clk} the clock period.

The worst case end-to-end latency for sending a message is the WCET of the software function implementing the send primitive (setting up the DMA transfer), which can be computed by the WCET analysis tool aiT, plus the time it takes to transfer the message across the Argo NoC. The latter is the time it takes the DMA controller to inject the necessary sequence of packets into the network plus the time it takes a packet to traverse the NoC

$$T_{msg} = ((N \cdot s - 1) + ((\left\lceil \frac{S_{msg}}{S_{chan}} \right\rceil - 1) \cdot N \cdot s + s) + L_{\mathrm{NoC}}) \cdot t_{clk}$$

where N is the number of TDM slots, s the slot length in clock cycles, S_{msg} is the size of the message (in bytes), S_{chan} is the number of bytes sent across the channel in one TDM

period, L_{NoC} is the number of clock cycles it takes to traverse the NoC and t_{clk} the clock period. L_{NoC} is the number of tokens that the asynchronous network has been initialized. In our implementation, each router is equivalent of three buffers, initialized with two tokens. Therefore, L_{NoC} is two times the number of hops.

For 9 cores an all-to-all schedule results in $9 \cdot (9-1) = 72$ channels. The static schedule to support those 72 channels is $N = 10$ slots long [22]. Each of the 9 processor cores needs 8 out of the 10 slots to send a message to each other core. With a default packet size of 3 words, each slot being $s = 3$ clock cycles long resulting in a packet data load of $S_{chan} = 8$ bytes. The resulting TDM period P is 30 clock cycles. During this interval, each of the 72 channels can transmit 8 bytes.

VI. Conclusion

For real-time systems, we need time-predictable processors. With the T-CREST architecture we present a multicore processor that is optimized for the worst-case execution time. All arbitrations of shared resources are implemented with time-division multiplexing to provide static arbitration. Static arbitration is easy to analyze for the worst case and avoids interference between different processors or tasks.

By having a constant injection rate and drain rate in the network-on-ship we can even apply time-division multiplexing on an asynchronous implementation of that network, leading to a time-predictable globally asynchronous, locally synchronous processor.

Acknowledgment

The work presented in this paper was partially funded by the Danish Council for Independent Research | Technology and Production Sciences under the project PREDICT (http://predict.compute.dtu.dk/), contract no. 4184-00127A.

Source Access

The T-CREST project is open-source and the README[1] of the Patmos repository provides a brief introduction how to setup Ubuntu for T-CREST and how to build T-CREST from the source. More detailed installation instructions, including setup on Mac OS X, are available in the Patmos handbook [23]. To simplify the exploration of T-CREST, we also provide a VM[2] with all packages and tools preinstalled.

References

[1] J. N. Buxton and B. Randell, "Software engineering techniques," in *Report on a Conference sponsored by the NATO Science Committee, Rome, October 1969*, NATO Science Committee, 1970.

[2] M. Schoeberl, "Time-predictable computer architecture," *EURASIP Journal on Embedded Systems*, vol. vol. 2009, Article ID 758480, p. 17 pages, 2009.

[3] M. Schoeberl, S. Abbaspour, B. Akesson, N. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann, S. Hepp, B. Huber, A. Jordan, E. Kasapaki, J. Knoop, Y. Li, D. Prokesch, W. Puffitsch, P. Puschner, A. Rocha, C. Silva, J. Sparsø, and A. Tocchi, "T-CREST: Time-predictable multi-core architecture for embedded systems," *Journal of Systems Architecture*, vol. 61, no. 9, pp. 449–471, 2015.

[4] E. Kasapaki, M. Schoeberl, R. B. Sørensen, C. T. Müller, K. Goossens, and J. Sparsø, "Argo: A real-time network-on-chip architecture with an efficient GALS implementation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, pp. 479–492, 2016.

[5] M. Schoeberl, D. V. Chong, W. Puffitsch, and J. Sparsø, "A time-predictable memory network-on-chip," in *Proceedings of the 14th International Workshop on Worst-Case Execution Time Analysis (WCET 2014)*, (Madrid, Spain), pp. 53–62, July 2014.

[6] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. Brown, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *Micro, IEEE*, vol. 27, pp. 15 –31, sept.-oct. 2007.

[7] A. Olofsson, T. Nordström, and Z. ul Abdin, "Kickstarting high-performance energy-efficient manycore architectures with Epiphany," in *in Proc. Asilomar Conference on Signals, Systems and Computers* (M. B. Matthews, ed.), pp. 1719–1726, IEEE, 2014.

[8] K. Goossens and A. Hansson, "The AEthereal network on chip after ten years: Goals, evolution, lessons, and future," in *Proceedings of the 47th ACM/IEEE Design Automation Conference (DAC 2010)*, pp. 306 –311, 2010.

[9] A. Hansson, M. Subburaman, and K. Goossens, "aelite: a flit-synchronous network on chip with composable and predictable services," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2009)*, (Leuven, Belgium), pp. 250–255, 2009.

[10] R. Stefan, A. Molnos, and K. Goossens, "dAElite: A tdm noc supporting qos, multicast, and fast connection set-up," *Computers, IEEE Transactions on*, vol. 63, no. 3, pp. 583–594, 2014.

[11] C. Paukovits and H. Kopetz, "Concepts of switching in the time-triggered network-on-chip," in *Proceedings of the 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2008)*, pp. 120 –129, August 2008.

[12] R. Obermaisser, H. Kopetz, and C. Paukovits, "A cross-domain multiprocessor system-on-a-chip for embedded real-time systems," *Industrial Informatics, IEEE Transactions on*, vol. 6, pp. 548 –567, nov. 2010.

[13] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[14] J. Sparsø, "Asynchronous circuit design – a tutorial," in *Principles of asynchronous circuit design – A systems perspective* (J. Sparsø and S. Furber, eds.), ch. 1-8, pp. 1–152, Kluwer Academic Publishers, 2001.

[15] J. Sparsø, E. Kasapaki, and M. Schoeberl, "An Area-efficient Network Interface for a TDM-based Network-on-Chip," in *Proc. Design, Automation and Test in Europe (DATE)*, pp. 1044–1047, 2013.

[16] M. Greenstreet, "Implementing a STARI chip," in *Proc. Int'l. Conf. Computer Design (ICCD)*, pp. 38–43, 1995.

[17] E. Kasapaki and J. Sparsø, "Argo: A Time-Elastic Time-Division-Multiplexed NOC using Asynchronous Routers," in *Proc. IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pp. 45–52, IEEE Computer Society Press, 2014.

[18] E. Kasapaki and J. Sparsø, "The Argo NOC: Combining TDM and GALS," in *European conference on circuit theory and design (ECCTD)*, pp. 1–4, 2015.

[19] M. Schoeberl, P. Schleuniger, W. Puffitsch, F. Brandner, C. W. Probst, S. Karlsson, and T. Thorn, "Towards a time-predictable dual-issue microprocessor: The Patmos approach," in *First Workshop on Bringing Theory to Practice: Predictability and Performance in Embedded Systems (PPES 2011)*, (Grenoble, France), pp. 11–20, March 2011.

[20] R. Heckmann and C. Ferdinand, "Worst-case execution time prediction by static program analysis," tech. rep., AbsInt Angewandte Informatik GmbH. [Online, last accessed November 2013].

[21] S. Hepp, B. Huber, J. Knoop, D. Prokesch, and P. P. Puschner, "The platin tool kit - the T-CREST approach for compiler and WCET integration," in *Proceedings 18th Kolloquium Programmiersprachen und Grundlagen der Programmierung, KPS 2015, Pörtschach, Austria, October 5-7, 2015*, 2015.

[22] M. Schoeberl, F. Brandner, J. Sparsø, and E. Kasapaki, "A statically scheduled time-division-multiplexed network-on-chip for real-time systems," in *Proceedings of the 6th International Symposium on Networks-on-Chip (NOCS)*, (Lyngby, Denmark), pp. 152–160, IEEE, May 2012.

[23] M. Schoeberl, F. Brandner, S. Hepp, W. Puffitsch, and D. Prokesch, "Patmos reference handbook," tech. rep., Technical University of Denmark, 2014.

[1] https://github.com/t-crest/patmos

[2] http://patmos.compute.dtu.dk/