Estonian Journal of Engineering, 2010, **16**, 1, 23–38 doi: 10.3176/eng.2010.1.05

System-level communication synthesis and dependability improvements for Network-on-Chip based systems

Mihkel Tagel, Peeter Ellervee and Gert Jervan

Department of Computer Engineering, Tallinn University of Technology, Raja 15, 12618 Tallinn, Estonia; {mihkel.tagel, peeter.ellervee, gert.jervan}@ati.ttu.ee

Received 28 October 2009, in revised form 2 February 2010

Abstract. Technology scaling into subnanometer range will create process variations that have impact on the overall manufacturing yield and quality. Smaller feature sizes permit to pack more functionality into a single chip. Increasing variability, complexity and communication bandwidth requirements will make the System-on-Chip designer's goal, to design a fault-free system, a very difficult task. Shift from traditional bus-based systems to networked systems solves several design problems but requires more focus on communication modelling. In this work we propose a system-level approach for communication modelling and synthesis. It makes possible to calculate precise communication delays that can be taken into account during application scheduling to avoid network congestions. We present a possible application of the proposed framework for scheduling fault-tolerant applications on non-reliable network.

Key words: communication synthesis, Network-on-Chip, dependability, system-level design.

1. INTRODUCTION

The trends in previous and current decades in automotive industry show increasing number of system components (so-called electronic control units, ECUs) and also flexibility with regard to topology and transmission support redundancy [[1]]. One luxury car includes more than 100 ECUs that form a distributed system and have different safety and dependability requirements. In the environment, where a lot of safety-critical actions are made by the system in real time without human intervention, dependability plays an important role.

Similar trends can be seen also in the consumer electronics area. The future of chip systems (System-on-Chip, SoC) will resemble more computer networks than traditional chips. At the same time there is an increasing demand for higher computational power and complexity of the systems. Take, for example,

telecommunication devices. The 3G mobile phones are not used only for voice calls and SMS – they are more versatile providing a large range of diverse functionalities. Limited throughput and bus lengths set limitations to the system design and to the number of system modules. To bridge the design and technology gap, the Network-on-Chip (NoC) paradigm has been proposed. NoC looks similar to computer networks and offers to a SoC designer a flexible, scalable and unified layered communication platform. NoC provides intellectual property (IP) reuse and decouples computation from the communication. In addition, new integration methodologies have enabled new 3D architectures, where the dies are stacked into 3-dimensional structures, thus providing even higher densities and complexity. 3D NoC architectures are emerging as a promising solution for high-performance multi-core systems on chip. They have clear advantages over more conventional planar counterparts in terms of system level performance metrics like throughput and latency [2].

Such SoCs and NoCs are centrepieces of many modern embedded systems, which can be found everywhere, thus having major influence on our way of life. Unfortunately, as technologies advance and semiconductor process dimensions shrink into the nanometer and subnanometer range, a high degree of sensitivity to defects begins to impact overall yield and quality [3]. The 2007 International Technology Roadmap for Semiconductors (ITRS) [4] states that relaxing the requirement of 100% correctness for devices and interconnects may dramatically reduce costs of manufacturing, verification and test. Such a paradigm shift is likely forced by technology scaling that leads to more transient and permanent failures of signals, logic values, devices and interconnects. It means that in consumer electronics, for example, where the reliability has not been a major concern so far, the design process has to be changed. Otherwise, there is a high loss in terms of faulty devices due to problems stemming from the nanometer and subnanometer manufacturing process.

NoC paradigm requires a shift in the design methodology – instead of a conventional vertical design, focusing mainly on optimizations at register-transfer-level (RTL), the NoC designer concentrates on the system level. At higher abstraction levels, the NoC designer has a much wider selection of design alternatives that will affect the final design. Independently from design methodology, the system-level design consists of several major tasks, as depicted in Fig. 1 [5]. System-level design (marked with gray background) starts with system modelling and architecture selection. Once the hardware platform is fixed, the software functions need to be extracted from the specification and mapped to the available hardware. Scheduling produces a valid execution order for the software functions [5].

The paper is structured as follows. First, motivation and the NoC design paradigm are described in Section 2. Next, system-level design flow is introduced in Section 3. Thereafter, communication synthesis is explained in Section 4, followed by task graph scheduling with dependability requirements in Section 5. Finally, we conclude our work in Section 6.

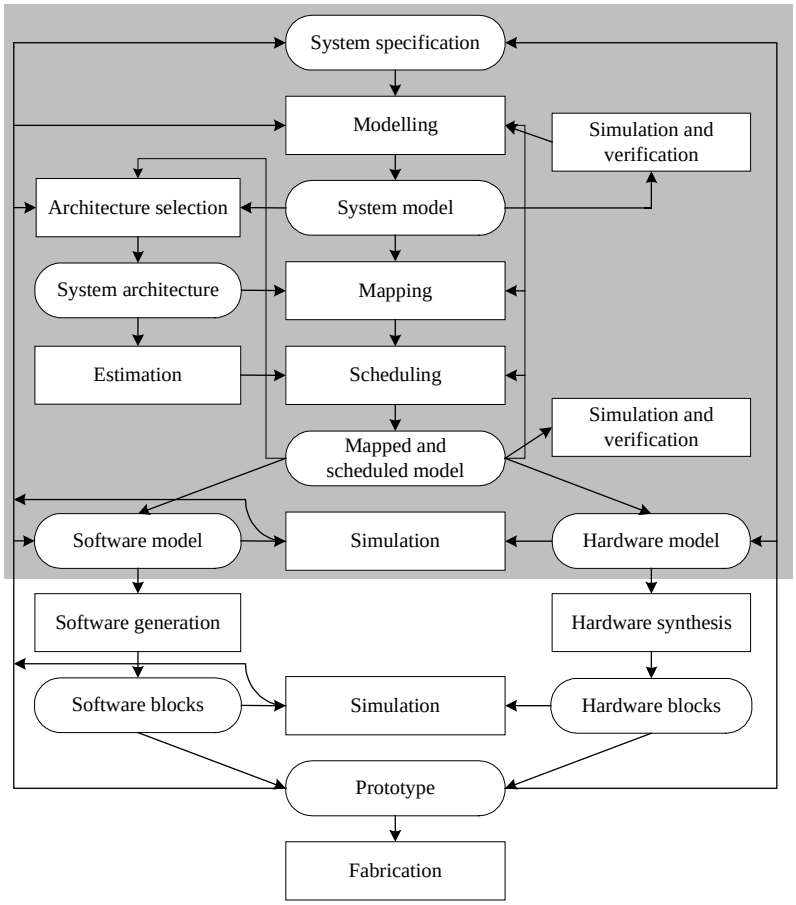

Fig. 1. System-level design flow.

2. MOTIVATION AND THE NoC DESIGN PARADIGM

There has been a lot of research made on system reliability in different computing domains by employing data encoding (Hamming, Berger code, cyclic codes), duplicating system components (triple modular redundancy), software-based fault tolerance techniques (signatures, watchdogs, checkpointing, memory protection codes). The research areas mostly either have had focus on low-level hardware reliability or have covered macro-distributed systems. Due to future design complexities and technology scaling, it is infeasible to concentrate only to low-level reliability analysis and improvement in embedded system design. ITRS states that the future holds new requirements for the SoC design flow, notably tool support for higher abstraction levels in both digital and analogue flows [4]. We should fill the gap by looking at the application level reliability analysis and improvement. We have to assume that the manufactured devices might contain faults and the application, running on the system, must be aware that the

underlying hardware is not perfect. NoC platform provides flexibility to tolerate faults and guarantee system reliability.

SoC is traditionally based on a bus architecture, where system modules (central processing unit, CPU), digital signal processor (DSP), etc exchange data via a central bus. When the number of components increases rapidly, we have a situation where the clock signal can not be distributed over the entire SoC during one clock cycle. To overcome the clock distribution problem, the NoC paradigm has been proposed that essentially is a distributed system. Point-to-point connections (*circuit switching*), common to SoC, is replaced in NoC by dividing the messages into packets (*packet switching*). Each component stores its state and exchanges data autonomously with others. Such systems are called in literature globally asynchronous locally synchronous (GALS) systems. Having multiple different network routes available for data transmission makes NoC adaptive – to balance the network load, for instance.

The NoC design paradigm has two good properties – predictability and reusability. Throughput, electrical properties, design and verification time are easier to predict due to the regular structure of the NoC. We can connect to the network any IP component that has the appropriate network interface. The NoC paradigm does not set any limits to the number of components. The components and also the communication platform are reusable – designer needs to design, optimize and verify them once. The layered network architecture provides the needed communication and network services enabling the functionality reuse [6].

The NoC paradigm has many advantages for designers:

- it enables the separation of communication structure and computation resources,
- NoC can deal with a large amount of computation at higher speed,
- NoC architecture is more flexible and scalable than other architectures [7].

An example NoC design flow, based on Philips Athereal platform [8], is shown in Fig. 2. The design flow input consists of the NoC topology specification, network interface constraints and communication platform constraints (latency, throughput). First, the topology is selected and the mapping of IP ports on the network interface ports is determined. Based on the results, the corresponding hardware platform (VHDL code) is generated. One intermediate step is throughput and latency calculation and verification. The results from previous steps can be used for SystemC simulation or for generating the NoC configuration code [8]. The Athereal design flow contains necessary steps for configuring the NoC platform parameters but it does not contain dependability analysis and design.

The communication platform limitations, data throughput, reliability and Quality-of-Services (QoS) are more difficult to address in NoC architectures than in computer networks. The NoC components (memory, resources) are relatively more expensive, whereas the number of point-to-point links is larger on chip than the off-chip. On-chip wires are also relatively shorter than off-chip ones, thus allowing a much tighter synchronization than the off-chip. On one hand, only a

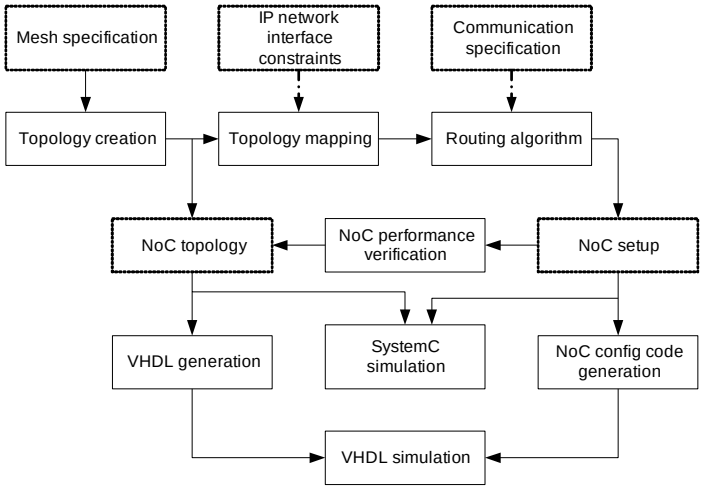

Fig. 2. Philips Athereal NoC design flow.

minimum design overhead is allowed that is needed to guarantee reliable data transfer. On the other hand, the on-chip network must handle data ordering and flow control issues [9]. The packets might appear at the destination resource out of order – they need to be buffered and put into correct order.

The properties of a traditional NoC platform are the following.

Topology refers to the physical structure of the network (how the resources and switches are connected to each other). Our NoC topology is a $m \times n$ (2D) mesh with bidirectional links between the switches (Fig. 3). The regular topology is not the most efficient in terms of manufacturing, but allows easier routing algorithms and better predictability. Each switch is connected to 4 switches and to one resource. Resources can be heterogeneous. A resource can be memory,

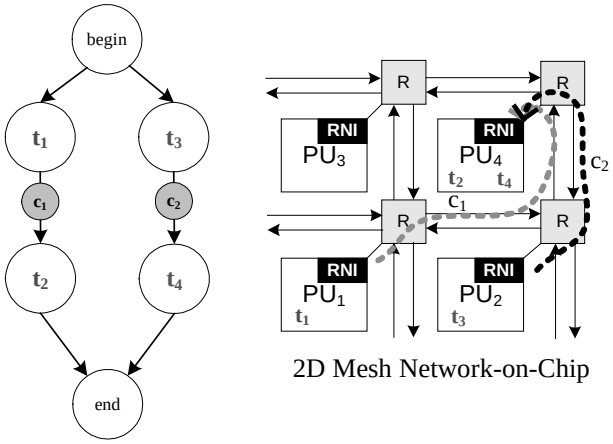

Fig. 3. Example task graph.

processor core, DSP, reconfigurable block or any IP block that conforms to the network interface (NI). Every resource is connected to switch via resource network interface (RNI). The resources have unique addresses.

Switching strategy determines how a message traverses its route. There are two main switching strategies: circuit switching and packet switching. Packet switching techniques include store-and-forward, virtual cut-through and worm-hole switching. The techniques are different in message splitting to smaller units and buffering/forwarding schemes.

Routing algorithm determines the routing paths the packets may follow through the network. The routing algorithms can be divided into deterministic and adaptive routing. In our work we are using the deterministic source-ordered XY routing algorithm. In XY routing the processing cores are numbered by their geographical coordinates. Packets are routed first via X and then via Y axis by comparing the source and destination coordinate.

Flow control deals with network load monitoring and congestion resolution. Due to the limited buffers and throughput the packets may be blocked and flow control decides how to resolve this situation. The most common techniques are credit-based, on/off and ack/nack flow control.

3. SYSTEM-LEVEL DESIGN FLOW

We are employing a traditional system-level design flow that we have extended to include NoC communication modelling and dependability issues (Fig. 4). In our approach the input application model for the system-level design flow is the extended task graph (ETG). The ETG describes application tasks, their dependencies and the worst case execution time (WCET). NoC platform introduces communication latency that depends not only on message size but also on resource mapping and needs to be taken into account. Therefore the task graph captures, in addition to control dependencies, also communication details between the tasks.

The architecture model, describing various NoC parameters (topology, size of NoC, switching/routing algorithm, channels and their parameters), is also given. The third part of the input information contains the application dependability requirements – number of faults to be tolerated on processing cores during one execution and number of faults to be tolerated on communication links during each data transmission.

Once the tasks have been mapped to the architecture, the constructive task scheduling starts. It consists of communication synthesis and task scheduling that are described in greater detail below. If dependability and other design requirements are met, the lower levels of HW/SW co-design processes continue. Otherwise changes are needed in the architecture or in the mapping.

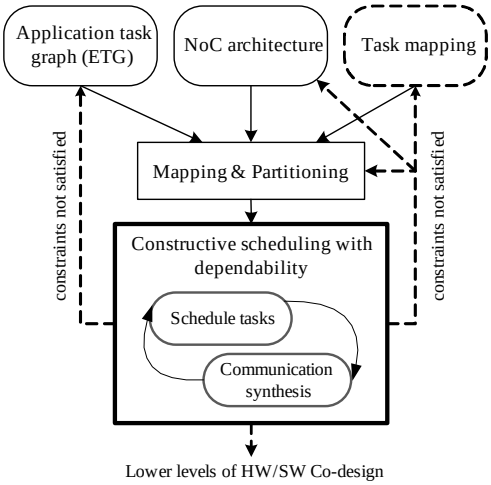

Fig. 4. Our system-level design environment.

4. COMMUNICATION SYNTHESIS AND SCHEDULING

To meet the required level of dependability, the designed system must be predictable. One of the main bottlenecks in application performance estimation for NoCs with hard real-time requirements is the unpredictability of communication latencies when using traditional task graph based models [10]. As introduced in Section 3, the communication latency depends not only on message size but also on the resource mapping and routing algorithm being used. In several research papers the average or worst case communication delay has been considered. Papers [11] and [12] approximate communication delay without considering the congestions. Authors of [13] use the term *compatible communication* and try to avoid congestions during task graph scheduling. However, they have customized the scheduling process, not the communication model. In [14] the worst case communication delay is calculated. The approach proposed in [15] divides communication events into scenarios. In [16] upper bounds (blocking included) are calculated for network delay. Authors of [17], similarly to [16], are trying to estimate the worst case communication delay. One of the key components of our dependable scheduling framework is communication synthesis; its main purpose is to calculate hard communication deadlines that are represented by communication delay (CD) and guide the scheduling process. In our approach we keep the communication network design margin as low as possible and avoid overdimensioning. Another design aspect is the ratio of modelling speed and accuracy. A communication schedule could be extracted by simulating the application on a NoC simulator, but the simulation speed will be the limiting factor.

The main reason why the simplified communication model does not work is rather simple – it does not take into account that different flow control units

(packages, flits) may travel using different routes and therefore their latencies may differ. In Fig. 3, an example task graph and its mapping onto three processing units are presented. Task t_1 is mapped onto PU_1, t_3 onto PU_2, and tasks t_2 and t_4 are mapped onto PU_4. It can be seen that communication c_1 (from t_1 to t_2) takes two links but c_2 (from t_3 to t_4) takes only one link. Moreover, the physical links that the communication traverses are shared resources. It means that in addition to calculating the communication delays, we need to avoid network congestions. It should be noted that the actual routes will depend on how the tasks are mapped and which routing approach is used.

A traditional task graph does not allow describing communication latencies that depend on various parameters such as topology, routing, switching algorithms, etc., and need to be calculated after task mapping and before the task graph scheduling. In [18] the authors proposed a task graph extension with detailed communication dependencies employing virtual cut-through switching with deterministic source-ordered XY routing. The basic idea is to cover with the task graph not only the tasks but also the flow control units. That is, all communication edges between tasks are transformed into sequences of nodes representing flow control units on the communication links. Edges represent dependencies between tasks and/or flow control units. Such an approach assumes that both tasks and communication are already mapped, i.e., it is known which tasks are mapped onto which resources and which data-transfers are mapped onto which links. Of course, different routing strategies will give different communication mapping but all information needed for scheduling is captured in the extended task graph. We have generalized the approach presented in [18] and made it compatible with wormhole switching and wormhole switching with virtual channels.

Input for scheduling is the extended task graph, where the tasks are mapped onto resources. Our proposed approach can be used with arbitrary scheduling algorithm, although the schedules in this paper are produced by using list scheduling. First, we will calculate the priorities of the tasks based on mobility. Mobility is defined as the difference between task ASAP (as-soon-as-possible) and ALAP (as-late-as-possible) schedules. Once a communication task is ready to be scheduled we start the communication synthesis sub-process. Depending on the selected switching method, some of the flow control units must be scheduled strictly to the subsequent time slots. In wormhole switching, the header flits contains the routing information and builds up the communication path, meaning that when the header flit goes through a communication link, the body flits must follow the same path. Also, when a header flit is temporarily halted, e.g., because of the traffic congestion, the following flits in downstream routers must be halted too. This sets additional constraints for the communication synthesis. The constraints – fixed order and delay between some of the nodes – are similar to the restrictions, used in pipelined scheduling [19].

One way to solve this is to have a specialized scheduler that takes into account that (communication) nodes must be scheduled in a certain order. For instance,

the list based scheduling (see, e.g., [19]) can take this into account with fine-tuned priorities – in addition to the traditional priorities (distance from the sink, mobility, etc.), these priorities should be fine tuned in the way that guarantees that flits of a package are transferred one after another. The main problem with such a model is that it is not general and each scheduler must be tuned, i.e., essentially pipelined. Another way is to perform communication synthesis and introduce additional nodes and edges in the original task graph. The nodes represent flow control units, while edges represent the dependencies between them. Figure 5 depicts the communication synthesis sub-process for communication (messages) c_1 between tasks t_1, t_2 and c_2 between tasks t_3, t_4 using wormhole switching. The variable size message is divided into bounded size packets. As a result, we can see that communication c_1 consists of one and c_2 of two packets. A packet is further divided into three type of control flow units (flits) – header (H), body (B) and tail (T). Typically there is only one H and T flit, but many B flits. The flit pipeline is built for all links the communication traverses. An important difference between wormhole switching compared to virtual cut-through is the contention handling. Once we have started the communication synthesis sub-process and find out that header flit of the package will conflict with some already scheduled flit on the corresponding link we need to halt the package submission. The body flit B_1 of communication c_1 on link 1 (Fig. 5) depends after

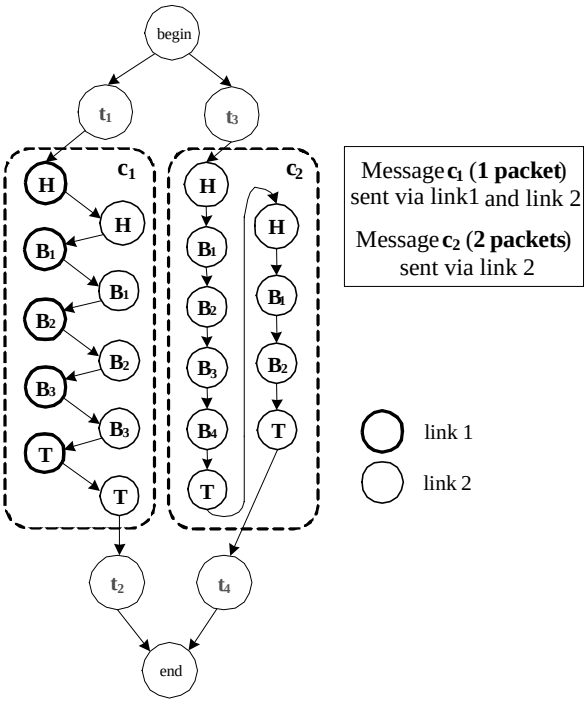

Fig. 5. Communication synthesis.

communication synthesis explicitly on the header flit (H) on the link 2. Combined with traditional priority scheduling to handle network resource conflicts (e.g., list scheduling), the body flit will be scheduled after the header flit has been sent and requirements of the wormhole switching are met.

The main benefit of the proposed model is that when the task graph has been transformed, we can calculate the communication delay for each communication task. Currently we take into account only the transmission time between the network links. The start-up latency (time required for packetization, copying data between buffers) and inter-router delay are static components that for sake of simplicity are considered as 0 delay. Another benefit is the generalization of the communication modelling – the communication is explicitly embedded in natural way into the task graph. The communication task graph model can be easily extended also for wormhole switching with virtual channels. The virtual channels are implemented in terms of separate input/output buffers in routers for each virtual channel and a Time Division Multiple Access (TDMA) method for shared channel access. The proposed approach does not suffer also from the destination contention problem, thus eliminating the need for buffering at the destination.

We have developed a software application that supports our system-level design flow and scheduling framework. We have performed several tests to evaluate different aspects of the approach described in this paper. We have used synthetic task graphs with 500, 750 and 1000 tasks mapped on different NoC architectures. Random task mapping to processing units have been used. Table 1 describes the NoC architecture parameters of the experiments. Figures 6 and 7 show scaling of our approach in terms of application and NoC size. When the number of tasks or NoC cores increases linearly, complexity increase of the extended task graph is near to linear. The increase is slighter for virtual cut-through switching than for wormhole. It is because in virtual cut-through, the flow control unit is a packet but in wormhole switching it is a flit. Figure 8 depicts how NoC size has impact on the schedule length. When the NoC size (available processing cores) increases, the schedule length decreases.

Table 1. NoC architecture parameters of the experiments

Parameter name	Value
NoC operating frequency	500 MHz
Link bit-width	32 bit
Flit size, packet size	64 bit, 512 bit
Packet header size	20 bit
Link bandwidth	16 Gbit/s
Topology and routing algorithm	2D Mesh, XY routing
Mapping	Random
NoC size (if not noted differently)	5×5

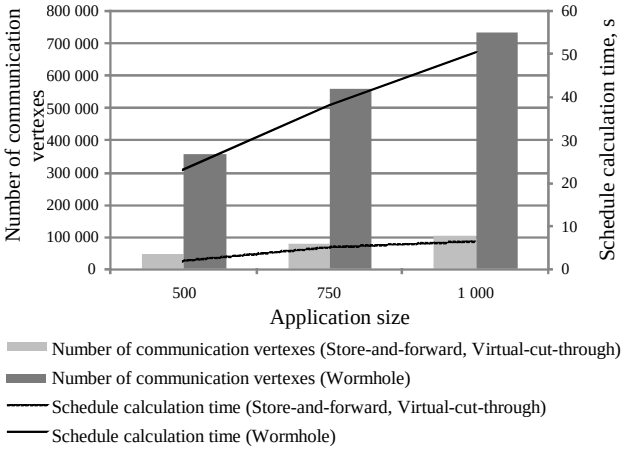

Fig. 6. Application size impact on ETG complexity (mapped on 5×5 NoC).

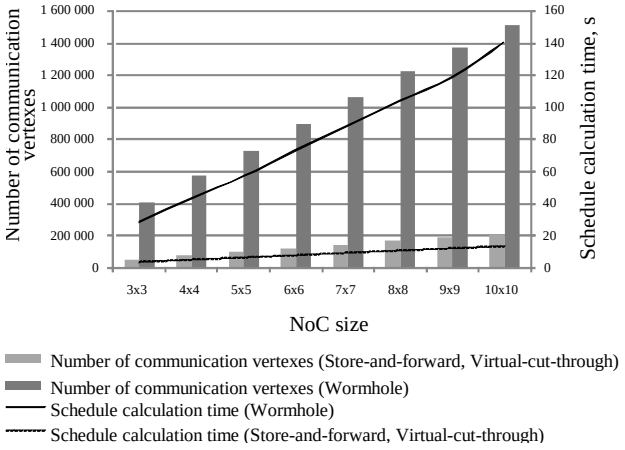

Fig. 7. NoC size impact on ETG complexity (source application with 1000 tasks).

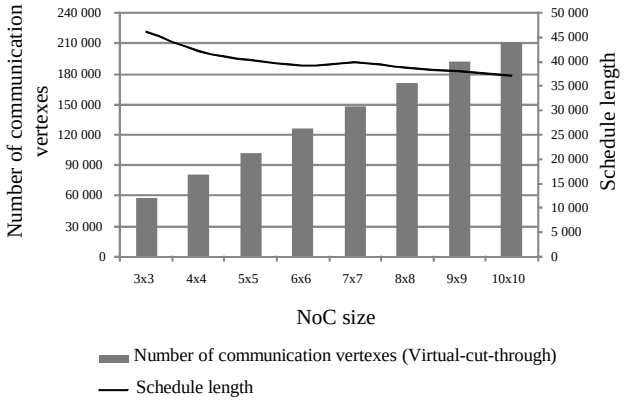

Fig. 8. NoC size impact on schedule length (source application with 1000 tasks).

5. TASK GRAPH SCHEDULING WITH DEPENDABILITY REQUIREMENTS

In [20] the impact of the transient faults in a microprocessor system is described. Three different error detection mechanisms are used – signature, Watchdog Timer and Error Capturing Instruction (ECI) mechanism. Signature is a technique where each or set of operations are assigned with a pre-computed "checksum" that indicate whether a fault has occurred during those operations. Watchdog Timer is a technique where the program flow is periodically checked for presence of faults. Watchdog Timer can monitor for example execution time of the processes or to calculate periodically "checksums" (signatures). In case of the ECI mechanism, redundant machine instructions are inserted into main memory to detect control flow errors (CFE).

Once a fault is detected with one of the techniques above, it can be handled by a fault tolerance mechanism. The author of [21] describes the following software-based fault tolerance mechanisms: re-execution, rollback recovery with check-pointing and active/passive replication. Re-execution restores the initial inputs of the task and executes it again. Time penalty depends on the task length. Rollback recovery with checkpointing mechanism reduces the time overhead – the last non-faulty state (so called checkpoint) of a task has to be saved in advance and will be restored if the task fails [21]. It requires checkpoints to be designed into application that is not a deterministic task. Active and passive replication utilizes spare capacity of other computational nodes [21]. Paper [22] describes fault-tolerant routing schemes in macro-distributed networks.

Network-on-Chip fault tolerance has been covered by several authors. The authors of [23] propose a fast and computationally lightweight paradigm for the on-chip communication, based on an error-detection and multiple-transmissions scheme. The key observation behind the strategy is that, at the chip level, the bandwidth is less expensive than in traditional networks because of existing high-speed buses and interconnection fabrics that can be used for the implementation of a NoC. Therefore, we can afford to have more packet transmissions than in the previous protocols in order to simplify the communication scheme and to guarantee low latencies. Similar approach is proposed in [24]. According to paper [25], reliability problems can be avoided with physical autonomy, i.e., by constructing the system from simple, physically autonomous cells. The electrical properties and logical correctness of each cell should be subject to verification by other autonomous cells that could isolate the cell if deemed erroneous (note that self-diagnosis is insufficient, because the entire cell, including the diagnostic unit, may be defect) [25]. Papers [26–28] concentrate on NoC interconnect, topology and routing algorithms. These are some example techniques for improving reliability in dependable systems. They have been widely used in either bus-based embedded, macro distributed systems or cover lower layers of NoC. Our objective is to extend these techniques to the system level, to provide design support at early stages of the design flow. The application should be able to

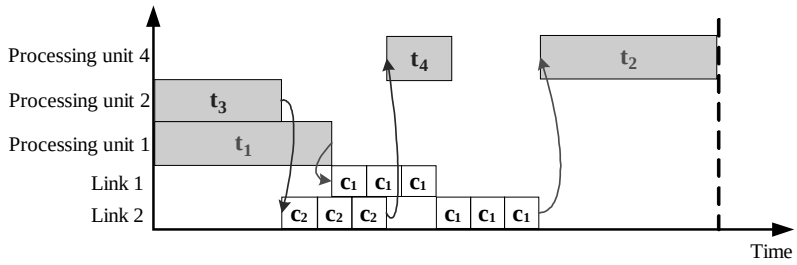

Fig. 9. Application schedule in NoC.

tolerate transient or intermittent faults. We are not currently considering permanent faults that need a somewhat different approach and can be handled by re-scheduling and re-mapping of the application on a NoC.

Paper [21] describes scheduling and optimization of fault-tolerant embedded systems in bus-based systems. The work considers faults only in computational tasks. The communication fault tolerance is not taken into account. Additionally, in bus-based systems the task mapping does not have such influence on communication delays as in NoC.

The application schedule without recovery slacks can be seen in Fig. 9. Once the task t_3 has finished, it initiates the data transfer c_2 that sends one packet over link 2. Communication between task t_1 and t_2 is denoted by c_1 that passes link 1 and link 2. As mapping of the tasks influences communication latencies, it is important to note that although both tasks send one packet to the destination node, the transfer from task t_1 to t_2 takes more time to complete. In hard real-time dependable systems, the predictable communication delays are crucial. Once a fault occurs, the system will apply a recovery method that might finally require re-scheduling of the application. To analyse the fault impact on the system, we need to have information how a fault affects the task execution and communication delays.

In our approach we assume that each NoC processing and communication node is capable of detecting faults (signatures) and executing corrective actions (re-execution, re-submission). We assign the recovery slacks and schedule the application using shifting-based scheduling. Shifting-based scheduling is an extension of the transparent recovery against single faults [21]. A fault, occurring on one computation node, is masked to other computation nodes. It has impact only on the same computation node. In shifting-based scheduling, the start time of communication is fixed. It means that we do not need a global real-time scheduler. The local scheduler controls the task execution on processing node and will switch to a contingency schedule in case of fault occurrence. An example is depicted in Fig. 10. We schedule the application tasks until we reach a communication task. Before freezing the communication start time, we will calculate the recovery slack as

$$\max(\mathrm{WCET}_{\text{tasks mapped to the same node}}) \cdot k_n, \tag{1}$$

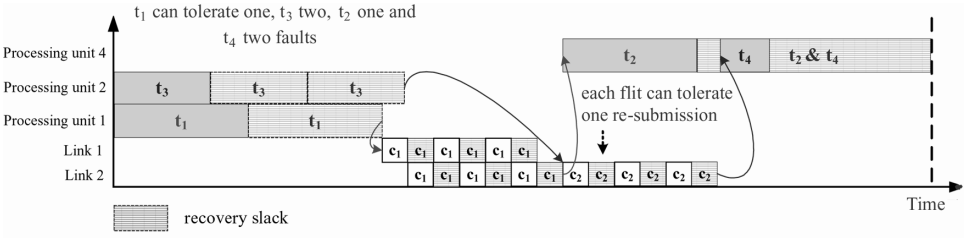

Fig. 10. Shifting-based scheduling.

where k_n is the number of transient faults to be tolerated by given task n. For example t_1 is assigned to tolerate one fault, adding

$$\max(\mathrm{WCET}_{t_1}) \cdot 1, \tag{2}$$

recovery slack into the schedule. In similar way we can calculate the recovery slack for t_3

$$\max(\mathrm{WCET}_{t_3}) \cdot 2. \tag{3}$$

For each flit transmission over a communication link we will assign a re-submission slack time as

$$CD \cdot t, \tag{4}$$

where t is the number of faults to be tolerated during transmission.

Comparing the two schedules in Figs. 9 and 10, we can see that the schedule with recovery slacks is longer than without. When fault tolerance would not have been taken into account, we would have went for a shorter schedule that would have lead to a deadline miss in case of fault occurrence. Schedule, which was produced having dependability requirements, is longer but will tolerate the specified amount of faults and the calculated deadline is satisfied.

The advantage of our approach is that we can take into account communication-induced latencies and fault effects already at very early stages of the design flow. The objective is to develop the proposed approach further and to produce an environment that can be used successfully for system level synthesis. It includes optimization of the communication synthesis and more efficient strategies for handling faults.

6. CONCLUSIONS

We have developed a framework for predictable communication synthesis in NoCs with hard real-time constraints. The framework models communication at the link level, using traditional task graph based modelling technique and supports various switching techniques. We have also demonstrated how our communication synthesis approach can be used in scheduling of dependable

NoC-based systems using, for example, shifting-based scheduling principles. In the future this work can be extended by developing more efficient scheduling heuristics, taking into account the specifics of on-chip networks. As our modelling approach provides detailed information about the communication then it is also possible to use different deterministic routing algorithms during the communication synthesis, in addition to the XY-routing algorithm, used in this paper. It is also important to address the routers buffer size optimization problem.

REFERENCES

1. Navet, N., Song, Y., Simonot-Lion, F. and Wilwert, C. Trends in automotive communication systems. *Proc. IEEE*, 2005, **93**, 1204–1223.
2. Pande, P., Ganguly, A., Feero, B. and Grecu, C. Applicability of energy efficient coding methodology to address signal integrity in 3D NoC fabrics. In *Proc. 13th IEEE International On-Line Testing Symposium (IOLTS 2007)*. Crete, Greece, 2007, 161–166.
3. Chandramouli, R. Infrastructure IP design for repair in nanometer technologies. *IEEE Design & Test Computers*, 2005, **22**, 17.
4. International Technology Roadmap for Semiconductors, 2007. URL: http://www.itrs.net/Links/2007ITRS/2007_Chapters/2007_Design.pdf (29 Sept. 2009).
5. Pop, P. *Analysis and Synthesis of Communication-intensive Heterogeneous Real-time Systems*. Ph.D thesis, Linköping University, Sweden, 2003.
6. Jantsch, A. and Tenhunen, H. *Networks on Chip*. Kluwer Academic Publishers, Boston, 2003, 9–15.
7. Guang, L. *Design of Frequency Controller for Minimizing Power Consumption in Network-on-Chip*. M.A. thesis, Royal Institute of Technology, Sweden, 2005.
8. Bartels, C., Huisken, J., Goossens, K., Groeneveld, P. and Meerbergen, J. Comparison of an aethereal network on chip and a traditional interconnect for a multi-processor DVB-T system on chip. In *Proc. IFIP International Conference on Very Large Scale Integration*. Nice, France, 2006, 80–85.
9. Radulescu, A. and Goossens, K. Communication services for networks on chip. In *Proc. Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. 2002, vol. 2, 275–299.
10. Marculescu, R., Ogras, U. Y. Li-Shiuan Peh, Jerger, N. E. and Hoskote, Y. Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives. *IEEE Trans. Computer-Aided Design of Integrated Circuits Systems*, 2009, **28**, 3–21.
11. Lei, T. and Kumar, S. A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In *Proc. Euromicro Symposium on Digital System Design (DSD'03)*. Belek – Antalya, Turkey, 2003, 180–187.
12. Marcon, C., Kreutz, M., Susin, A. and Calazans, N. Models for embedded application mapping onto NoCs: timing analysis. In *Proc. 16th IEEE International Workshop on Rapid System Prototyping (RSP 2005)*. Montreal, Canada, 2005, 17–23.
13. Hu, J. and Marculescu, R. Communication and task scheduling of application-specific networks-on-chip. *Proc. IEEE*, 2005, **152**, 643–651.
14. Shin, D. and Kim, J. Power-aware communication optimization for networks-on-chips with voltage scalable links. *CODES + ISSS 2004*. Stockholm, Sweden, 2004, 170–175.
15. Stuijk, S., Basten, T., Geilen, M., Ghamarian, A. H. and Theelen, B. Resource-efficient routing and scheduling of time-constrained streaming communication on networks-on-chip. In *Proc. 9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools (DSD 2006)*. Dubrovnik, Croatia, 2006, 45–52.
16. Shi, Z. and Burns, A. Real-time communication analysis for on-chip networks with wormhole switching networks-on-chip. In *Proc. Second ACM/IEEE International Symposium on Networks-on-Chip (NoCS 2008)*. Newcastle, UK, 2008, 161–170.

17. Shin, D. and Kim, J. Communication power optimization for network-on-chip architectures. *J. Low Power Electronics*, 2006, **2**, 165–176.
18. Manolache, S. *Analysis and Optimisation of Real-time Systems with Stochastic Behaviour*. Ph.D thesis, Linköping University, Sweden, 2005.
19. De Micheli, G. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, New York, 1994.
20. Miremadi, G. and Torin, J. Evaluating processor behaviour and three error-detection mechanisms using physical fault-injection. *IEEE Trans. Reliability*, 1995, **44**, 441–454.
21. Izosimov, V. *Scheduling and Optimization of Fault-tolerant Distributed Embedded Systems*. Tech. Lic. thesis, Linköping University, Sweden, 2006.
22. Koren, I. and Krishna, C. *Fault-Tolerant Systems*. Morgan Kaufmann, San Francisco, 2007.
23. Dumitras, T. and Marculescu, R. On-chip stochastic communication. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE 2003)*. Munich, Germany, 2003, 790–795.
24. Pirretti, M., Link, G. M., Brooks, R. R., Vijaykrishnan, N., Kandemir, M. and Irwin, M. J. Fault tolerant algorithms for network-on-chip interconnect. In *Proc. IEEE Computer Society Annual Symposium on VLSI*. Tampa, FL, USA, 2004, 46–51.
25. Valtonen, T., Nurmi, T., Isoaho, J. and Tenhunen, H. An autonomous error-tolerant cell for scalable network-on-chip architectures. In *Proc. 19th IEEE Nordic Event in ASIC Design (NorChip 2001)*. Stockholm, Sweden, 2001, 198–203.
26. Yang Yu, Mei Yang, Yulu Yang and Yingtao Jiang. A RDT-based interconnection network for scalable network-on-chip designs. In *Proc. International Conference on Coding and Computing (ITCC 2005)*. Las Vegas, NV, USA, 2005, vol. 2, 723–728.
27. Kariniemi, K. and Nurmi, J. Fault tolerant XGFT network on chip for multi processor system on chip circuits. In *Proc. International Conference on Field Programmable Logic and Applications (FPLA 2005)*. Tampere, Finland, 2005, 203–210.
28. Murali, S., Atienza, D., Benini, L. and De Micheli, G. A multi-path routing strategy with guaranteed in-order packet delivery and fault-tolerance for networks on chip. In *Proc. 43rd ACM/IEEE Design Automation Conference (DAC 2006)*. San Francisco, CA, USA, 2006, 845–848.

Süsteemitaseme kommunikatsiooni süntees ja usaldusväärsuse parendamine kiipvõrkudel põhinevates süsteemides

Mihkel Tagel, Peeter Ellervee ja Gert Jervan

Pooljuhttehnoloogia areng nanostruktuuride suunas on kaasa toonud olukorra, kus tehnoloogiliste protsesside varieerumisel on toodete kvaliteedile ja tootlusele märgatav mõju. Samas muutuvad komponendid väiksemaks ja ühele kiibile on võimalik pakkida üha rohkem funktsionaalsust. Eelmainitud põhjustel on veavabade kiipsüsteemide loomine muutunud väga keerukaks ülesandeks. Liikumine traditsiooniliselt arvutuspõhiselt mudelilt kommunikatsioonipõhisele mudelile aitab mõningaid probleeme lahendada. Käesolevas töös on vaadeldud süsteemitasemel lähenemist kommunikatsiooni modelleerimisele ja sünteesile keerukatel kiipvõrkudel põhinevates kiipsüsteemides. Kirjeldatud lähenemine võimaldab täpselt välja arvutada kommunikatsioonile kuluvat aega, mis on ülesannete planeerimisel esmaoluline, et vältida kiipvõrgu ülekoormatust. Samuti on demonstreeritud esitatud lähenemise üht võimalikku rakendust, mis lubab mitteusaldusväärset kiipvõrku kasutades veakindlaid rakendusi planeerida.