Formal Development of NoC Systems in B

Tsiopoulos, Leonidas; Walden, Marina

Published: 01/01/2006

Document Version
Final published version

Leonidas Tsiopoulos | Marina Waldén

Formal Development of NoC Systems in B

Turku Centre *for* Computer Science

TUCS Technical Report
No 751, March 2006

Formal Development of NoC Systems in B

Leonidas Tsiopoulos
Åbo Akademi University, Department of Information Technologies
Marina Waldén
Åbo Akademi University, Department of Information Technologies

TUCS Technical Report
No 751, March 2006

Abstract

When developing complex Network-on-Chip (NoC) systems we need to ensure that they satisfy their functional requirements. This can be achieved by developing the systems in a structured way using a formal method with tool support. We use the B Action Systems formalism for this purpose. We give a general formal framework for the development of NoC systems avoiding architectures with complex controllers and complex arbiter modules for deciding the routing path of a data packet. The development is performed in a stepwise manner composing more advanced routing components out of simpler units.

Keywords: B Method, Action Systems, Network-on-Chip, formal verification, parallel composition

TUCS Laboratory
Distributed Systems

1. Introduction

With the advances in the semiconductor technology, we can have thousands of computational resources on a single chip [17]. Network-on-Chip (NoC) systems are useful for connecting this vast amount of resources. These systems are used in many important applications, such as in automotive and airplane control systems. Hence, it is important that they are reliable. Since NoC systems are also often very complex, it is important to build formally verified specifications that are structured and unambiguous. Therefore, efficient methods are needed in order to specify reliable NoC systems, as well as to model the communication and verify their design.

Formal methods for distributed systems can be used for this purpose. In order to facilitate the formal development, tool support is needed. *B Action Systems* [10, 35] is such a formal method. The B Action Systems is a state-based formalism that was created in order to be able to reason about parallel and distributed systems, like Action Systems [7, 8], within the B Method [1]. While Action Systems provides a rigorous framework for interconnect circuit design [22, 26, 27, 30], the B Method is provided with commercial tool support, e.g., Atelier B [12].

In this paper we are mainly interested in the *router*, the most important part of a NoC system, and how it distributes data packets. Since there is one router per resource on a NoC, a single chip could contain thousands of routers. After interconnect wires, routers are the most power consuming parts on a NoC [33]. The more complex the router the bigger the power consumption. Therefore, they need to be small, simple, efficient and reliable.

We focus on the formal specification of communication routers and give a compositional framework for *asynchronous* routing schemes for NoC system design using the B Action Systems formalism. We propose a development method where we create the specification in a component-wise and hierarchical manner. We rely on the method developed by Tsiopoulos et al [32] for System-on-Chip intercommunication design. A structured approach to specifications is essential in particular for complex systems. Within our framework the routers are composed out of simple asynchronous channels relying on the request and acknowledgement phases of the asynchronous communication. In order to provide the flow control for routing data packets efficiently in a network, simple asynchronous system modules are further composed out of these routers.

The structure of the paper is as follows. Section 2 describes briefly the NoC paradigm. In Section 3 an overview of the B Action Systems formalism adapted for the NoC system design is given. Section 4 formally specifies a router within the B Action Systems formalism using the proposed compositional development method. In Section 5 we elaborate on related work and in Section 6 some concluding remarks are drawn.

2. Network-on-Chip systems

Nowadays the technological advances allow for millions of transistors on a single chip. Therefore, bus-based System-on-Chip (SoC) communication architectures are not

sufficient anymore. The NoC paradigm was recently proposed as a solution to the increasing problems of on-chip communication [13, 17]. A regular, two dimensional NoC architecture consists of one router per functional resource [21]. The routers are responsible for the data transactions between the resources. Thus, the performance and reliability of the NoC architecture depends on the performance and reliability of its routers. Therefore, the design of routers is the most important part of the design of a NoC approach and efficient methods are needed for this.

Generally, two different main routing modes are used, the *deterministic* and the *adaptive* mode [23]. Even a combination of the modes has been used in *DyAD* [18]. In deterministic routing the path is determined by the source and the destination address. The main advantages of this routing is the simplicity in terms of routers design and low routing latency when the network is not congested. On the other hand, the disadvantage is that deterministic routers suffer from throughput degradation when the network is congested. In adaptive routing given a source and a destination address, the path taken by a packet depends on dynamic network conditions. This increases the possibilities of packets to avoid congested links by using alternative routing paths, which in turn leads to higher throughput. The disadvantage of adaptive routing is that it has a higher routing latency compared to deterministic routing, at low levels of network congestion. This is due to the extra logic needed in order to decide on a good path. Moreover, the flexibility in routing the data packets can lead to packet cycling, where the same packet arrives to the same router several times. This problem can be avoided by explicit restrictions [11] in the design of the routers. We have chosen the adaptive routing mode in this paper due to its flexibility and take care of the cycling problem via the features of our formalism.

The timing in the NoC systems is also important for the performance of the routers. There are two main schemes for the timing of hardware systems, the *synchronous* and *asynchronous* schemes. In the synchronous timing scheme the whole system is synchronized by a single clock or by several clocks with a synchronization controlling mechanism. In the asynchronous scheme there is no clock and the operation of the whole system relies on the *request* and *acknowledgement* phases of the communication between the subsystems. For NoC system design the main timing approach has been the *Globally Asynchronous Locally Synchronous* system approach [17, 18]. It is a combination of synchronous and asynchronous timing where the operation of the functional resources is controlled by clocks and the inter-resource communication is asynchronous.

The completely asynchronous approach we use in this paper has several advantages compared to any synchronous one. There are no synchronization problems as the complexity of a chip increases and there are no clock related problems, like clock skew and clock delay. Moreover, there is less power consumption on an asynchronous NoC system, because its parts are active only when and where needed [16]. The electromagnetic emission is also low, since the activity is not synchronized by periodic clocks.

Although there is much work presented on NoC systems, there is not much on formal specification and design of such complex systems using tool support. We use the B Action Systems formalism for specification and verification of asynchronous and

adaptive routing schemes for NoC systems and for providing a formal framework for structured and hierarchical NoC system design with commercial tool support.

3. Specifications in B Action Systems

B Action Systems [10, 35] is a state-based formalism based on Action Systems and the B Method. The B Action Systems formalism was created in order to be able to reason about parallel and distributed systems, like Action Systems, within the B Method and is related to Event B [2].

The form of a B action system [1] is given in Fig. 1. The system is identified by a unique name. The variables of the system are given in the **VARIABLES**-clause. The **INVARIANT**-clause defines the types of the local variables and gives their guaranteed behaviour. Initial values are assigned to the local variables in the **INITIALISATION**-clause. The operations (or actions) in the **OPERATIONS**-clause are of the form *Oper* = **SELECT** *P* **THEN** *S* **END**, where *P* is a predicate (later called a guard) on the variables and *S* is a substitution statement. When *P* holds the action *Oper* is said to be enabled. Only enabled actions are considered for execution and if there are several actions enabled simultaneously they are selected for execution in a non-deterministic manner. When there are no enabled actions the system terminates.

MACHINE A
INCLUDES/EXTENDS/SEES
B
VARIABLES
x
INVARIANT
Inv(x)
INITIALISATION
$x := x_0$
OPERATIONS
$Oper_1$ = **SELECT** P_1 **THEN** S_1 **END**;
…
$Oper_n$ = **SELECT** P_n **THEN** S_n **END**
END

Fig. 1. Example of a B action system

The substitution statement in the operation (action) can, for example, be a *skip*-substitution, a simple substitution, a multiple substitution, a preconditioned substitution or a guarded substitution [1]. Each substitution statement S is defined as a predicate transformer that transforms a postcondition Q into the weakest precondition, wp(S,Q) [14], the initial states from which S is guaranteed to terminate. The substitutions above are defined as follows:

wp(*skip*, Q)	$= Q$
wp(x:=e, Q)	$= Q[x:=e]$
wp(x:=e || y:=f, Q)	$= Q[x,y:=e,f]$, where $x \cap y = \varnothing$
wp(**PRE** P **THEN** S **END**, Q)	$= P \wedge$ wp(S,Q)

$$\text{wp}(\textbf{SELECT } P \textbf{ THEN } S \textbf{ END}, Q) = P \Rightarrow \text{wp}(S,Q)$$

where x and y are distinct variables, e and f are expressions, P is a predicate and S is a substitution statement.

Structuring mechanisms as, e.g., **SEES**, **INCLUDES** and **EXTENDS** can be used to express B action systems as a composition of subsidiary systems [1]. The **SEES**-mechanism allows read access to the seeing system, meaning that variables of the seen system can be used in the initialization and actions of the seeing system. By using the **INCLUDES**-mechanism the invariant of the including system can express requirements on the variables of the included system. The variables of the included system are directly visible to the including system, but may only be updated via the included system. Actions of the included system can be made available by promoting them into the including system within a **PROMOTES**-clause. These actions must preserve the invariant of the including system. If all the actions of the included system are to be promoted to the including system, then the included system is an extension which can be modelled with the **EXTENDS**-mechanism.

When modelling a system as composition of subsystems instances are useful. An instance *inst* of a system *A* is referenced in B Action Systems via the renaming function in B, *inst.A*. The above mechanisms and the instantiation of subsystems allow us to have a structured, hierarchical and well defined development method for system design.

3.1. Communication mechanisms

In B Action Systems we can have global variables that can be read and updated by more than one system. The global variable z is declared in a separate machine *GlobalVar_z*. It is then **INCLUDED** in the systems that refer to the global variable as shown in Fig. 2. By including the machine *GlobalVar_z* in system *A* and system *B* the systems are both allowed to assign a new value y to the variable z via the operation *assign_z(y)* [10, 35].

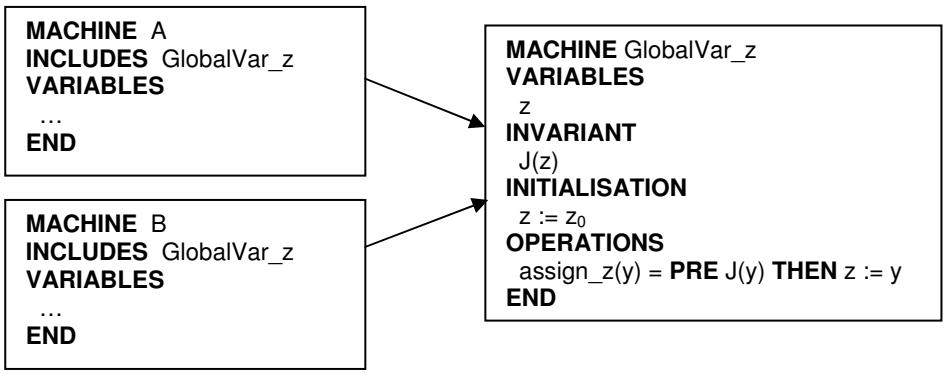


Fig. 2. Global variables in B

For communication purposes we can also declare global procedures in B Action Systems [31]. The procedures are of the same form as the actions. When an action,

A_oper = **SELECT** *P* **THEN** *S* || *Proc* **END**, calls a procedure, *Proc* = **SELECT** *Q* **THEN** *T* **END**, action *A_oper* is enabled only if procedure *Proc* is also enabled ($P \wedge Q$ holds). The action and the procedure are executed as an atomic entity. More details on procedures are given elsewhere [31, 34]. We note that in B Action Systems the calling action and the procedure need to be in separate machines.

3.2. Composing the model

B action systems can be composed into parallel systems [10, 28]. The parallel composition of B action systems can be presented using the **EXTENDS**-clause. This also provides an efficient way to model system hierarchy, i.e., a system can be modelled as a composition of subsystems listed in the **EXTENDS**-clause. Let us consider the B action systems in Fig. 3 where system *A* extends system *B* indicating that *A* is considered to be composed in parallel with *B*.

The parallel composition *AB* of subsystem *A* and subsystem *B* is formed by merging the variables, invariants, procedures and actions of *A* and *B*. The local variables *x* and *y* of the subsystems have to be distinct. This can, however, easily be achieved by renaming before forming the composition. The global variable *z* will be a global variable of the parallel composition. Since the invariant of system *AB* is the conjunction of the invariants of the subsystems, the action *A_oper* should preserve the invariant *Inv_B* and the action *B_oper* should preserve the invariant *Inv_A*. This is mainly a restriction on the assignments to the common global variables *z*.

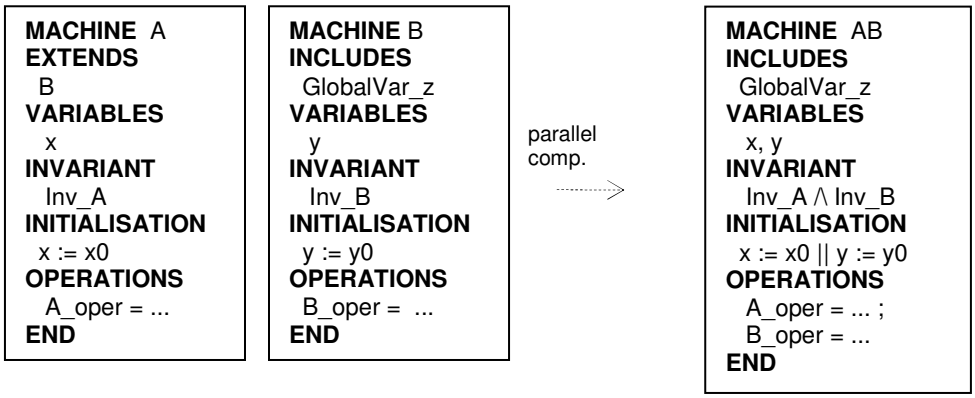


Fig. 3. Parallel composition with **EXTENDS** of B action systems *A* and *B*

Another hierarchical way to model composition of B Action Systems can be achieved by using the **INCLUDES** structuring mechanism, i.e., a system can be modelled as a parallel composition of itself and a number of subsystems or instances of these listed in its **INCLUDES**-clause. This composition is shown in Fig. 4 where system *A* includes system *B*.

The invariant of system *A* is the conjunction of its own invariant and the invariant of subsystem *B*. Hence, the actions in system *A* should preserve the invariant of its subsystems. We assume that a system can contain actions, as well as procedures in its

OPERATIONS-clause. In order to form the composition of the systems *A* and *B*, all the actions and some of the procedures of the included subsystem *B* are promoted to system *A*. Only the procedures of the subsystem that preserve the invariant of system *A* are promoted. These procedures can be considered to form the interface of system *A*.

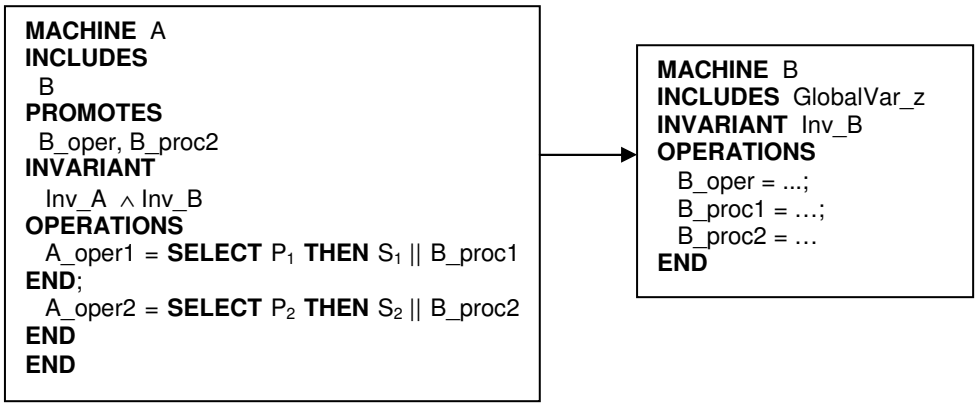


Fig. 4. Parallel composition with **INCLUDES** of B action systems *A* and *B*

The invariant of system *A* is the conjunction of its own invariant and the invariant of subsystem *B*. Hence, the actions in system *A* should preserve the invariant of its subsystems. We assume that a system can contain actions, as well as procedures in its **OPERATIONS**-clause. In order to form the composition of the systems *A* and *B*, all the actions and some of the procedures of the included subsystem *B* are promoted to system *A*. Only the procedures of the subsystem that preserve the invariant of system *A* are promoted. These procedures can be considered to form the interface of system *A*.

For developing complex systems, like NoC systems, we propose a hierarchical design approach using parallel composition with **INCLUDES** and **PROMOTES**. Hence, we compose the specification of the system out of components. In the compositional development we proceed as follows:

1. *Define the global variables of the system*
2. *Specify the basic components (or subsystems) of the system*
3. *Create a system for controlling the subsystems (via inclusion)*
    a. *Add the interface procedures to the subsystems*
    b. *Promote the actions of the subsystems into the controlling system*
    c. *Specify actions in the including system that schedule the execution of the subsystems via calls to their procedures*
4. *Repeat step 3 until the desired level of control has been reached*

Note that the interface procedures can be added to the subsystems in Step (3a) either by creating them or by promoting them. By composing systems in the above way we have

a formal and structured specification of them, which can be verified with the help of the tool support.

3.3. Consistency proofs

The guaranteed behaviour of a system is given in the invariant. In order to show that the variables and the actions conform to the invariant we prove the internal consistency of the system [1]. A system *A* with the state variables *x*, the invariant *Inv_A* and actions A_oper_i = **SELECT** P_i **THEN** S_i **END** is internally consistent, if the following proof obligations hold:

(C1) $\exists\ x.\ Inv\_A$
(C2) $\mathrm{wp}(x:=x0,\ Inv\_A)$
(C3) $(Inv\_A \wedge P_i) \Rightarrow \mathrm{wp}(S_i,\ Inv\_A)$

Intuitively, we have to prove that the invariant is consistent (C1) and that it is established initially (C2). Furthermore, each action A_oper_i of system *A* should preserve the invariant (C3). The proof obligations (C1) – (C3) are generated automatically with the tool support Atelier B for the B Action Systems. With the automatic and interactive proof tools of Atelier B these proof obligations can then be discharged.

For a hierarchical system composition to be internally consistent, the following compositionality theorem [32] should hold. This theorem was inspired by the theorem in [9]. Relying on this theorem we can build consistent systems out of consistent subsystems in a compositional manner.

THEOREM 1. *Let systems A and B form subsystems of system C. Assume that these subsystems are internally consistent. Furthermore assume that*

(a) the variables of the subsystems are disjoint: ***var*** *of* $A \cap$ ***var*** *of* $B = \varnothing$
(b) each subsystem respects the invariant of the other subsystem, and
(c) the system C respects the invariants of the subsystems.

Then system C is internally consistent.

Theorem 1(a) requires that the variables of the subsystems are disjoint. This is easily achieved by renaming their variables. Furthermore, the actions of the subsystems should preserve each other's invariants. This is mainly a restriction on the assignments to the common global variables in the actions of the subsystems. If system *C* includes instances of the subsystems, these subsystems will by definition have disjoint variables. Hence, Theorem 1(a) and 1(b) hold trivially. Theorem 1(c) will be proved with the help of the Atelier B tool for B Action Systems, since the invariants of the subsystems are included in the invariant of the system.

4. Formal specification of a router with distribution control

We show our specification method by modelling a routing scheme in a two dimensional (2D) mesh (see Fig. 10) NoC as a case study. Since the specification of routers leads to complex systems, a compositional development method is needed. Following our development method using B Action Systems, as presented in Section 3.2, we start by specifying a channel as a basic component. Then we create a router that uses instances of this channel to distribute data packets. Finally, instances of this router are composed into a module which controls the packet distribution in the network. We emphasize that our framework can be easily extended for other topologies, like, for example, the honeycomb topology [17] or the octagon topology [19].

4.1. Specifying the channels

The development of the routing scheme of the 2D mesh NoC starts with the specification of a channel giving its variables and properties within the B Action Systems formalism. This involves Steps (1) and (2) of our development method presented in Section 3.2. To exemplify these steps we here specify an abstract point-to-point channel, i.e., an asynchronous push channel transferring data upon request [25].

The push channel, which is here considered to be buffer-less, can propagate data as well as control values. The variables *cin* and *cout* model the communication control at the input and the output of the channel, respectively. They can have a request value (here modelled as *TRUE*) or an acknowledgement value (here *FALSE*). Moreover, the data in the input and output of the channel is given as the variables *din* and *dout* of the generic type *DATA*. The values of *cin* and *cout* can be observed by the modules which will be connected to the input and the output of this channel, respectively. When the input of the channel receives data, this data is propagated to the output of the channel. After the data has been removed from the output of the channel, *cout* is acknowledged. The channel propagates backwards the acknowledgement to its input *cin* and becomes ready to receive new data again. A diagrammatic view of this channel is given below in Fig. 4.

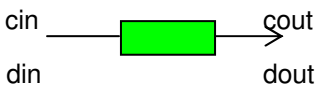


Fig. 4. The asynchronous push channel

When modelling the asynchronous push channel as a B action system we first create a system *ChannelData* for *defining the global variables* (Step (1)) of the channel. For each global variable (*cin*, *cout*, *din* and *dout*) we give a procedure for changing its value. The procedures for changing the data values also change the corresponding control values, since both data and requests are propagated along the channel simultaneously. The B Action System specification of *ChannelData* is given in Fig. 5. System *def* defines the deferred set *DATA*.

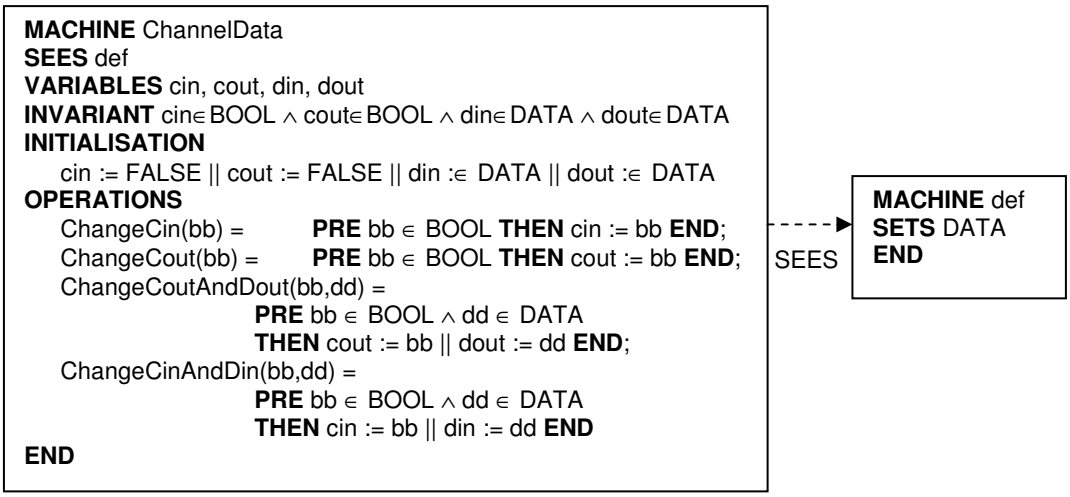


Fig. 5. The B action system for *ChannelData*

After we have defined the global variables of the channel in *ChannelData*, we *create the basic component of a router* (Step (2)), the asynchronous push channel. The channel component is given as the B action system *PushChannel* that includes *ChannelData*. The action *TransferData* propagates the data along the channel upon request on *cin*. Action *AckTransfer* is enabled when *cout* is acknowledged (by another system) and then it acknowledges *cin*. The invariant of the channel guarantees that when *cout* has received the data, then *cin* must also have received this data. Furthermore, if this is the case, then the data at the output of the channel is equal to the data at the input (*dout* = *din*). Moreover, when *cin* has been acknowledged, then *cout* must also have been acknowledged. The B Action System specification of *PushChannel* and its relationship to *ChannelData* is shown in Fig. 6.

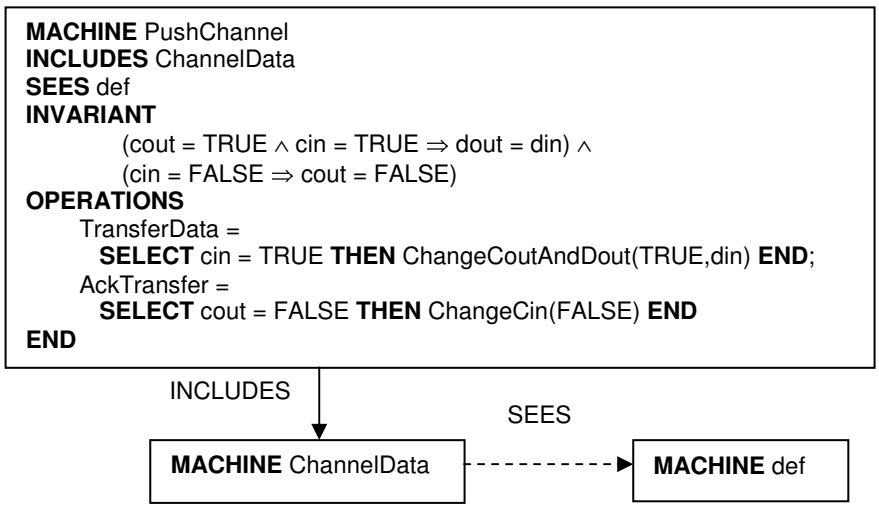


Fig. 6. The B action system for *PushChannel*

4.2. Composing channels to a router

In the 2D mesh network every node in the middle of the network is connected to four other nodes and at each node there are four routers. More specifically, we here focus on the router receiving data packets from its northern router via push channel *p0* and sending them to its eastern, southern and western routers via push channels *p1*, *p2* and *p3*, respectively, as shown in Fig. 7. The other three routers at the same node work similarly receiving data packets from east, south and west, respectively.

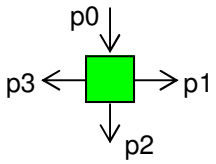


Fig. 7. A router of a two dimensional mesh network

After the basic push channel has been specified, we can *create a router for controlling instances of the channel* (Step (3)). Channel *p0* acts as the input channel of the router and channels *p1*, *p2* and *p3* as its output channels. Before we give the specification of the router as a B action system, we *add new global procedures* (Step (3a)) to system *PushChannel* for changing the values of the global variables that form the interface of the channel. These procedures restrict the changes on the global variables according to the invariant of *PushChannel*. Hence, we allow the router to assign only the request value to *cin* with the data *din* in *ProcChangeCinAndDin* and the acknowledgement value to *cout* in *ProcChangeCoutFalse*, as shown in Fig. 8.

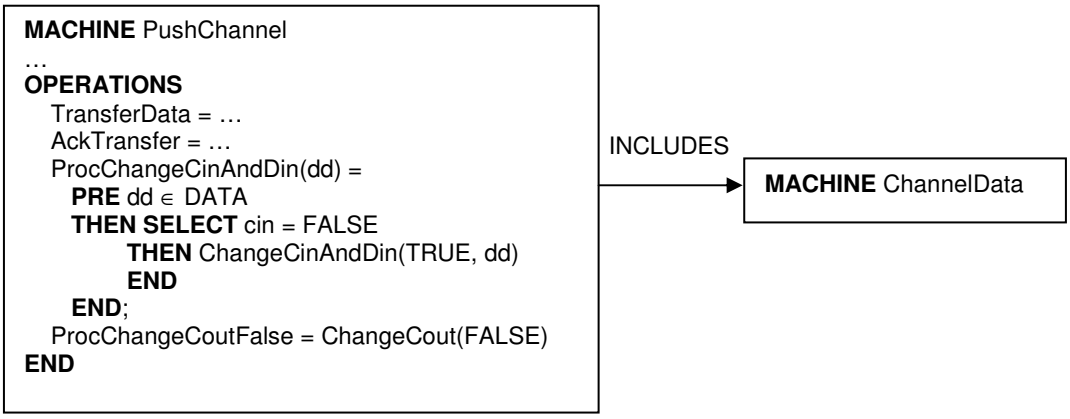


Fig. 8. New global procedures in *PushChannel*

We can now create system *Router* given in Fig. 9. Four instances of *PushChannel* are included in *Router* and *their actions are promoted* (Step (3b)). We specify two new actions, *DistributeData* and *CollectAck*, for controlling the channels and propagating the data along them (Step (3c)).

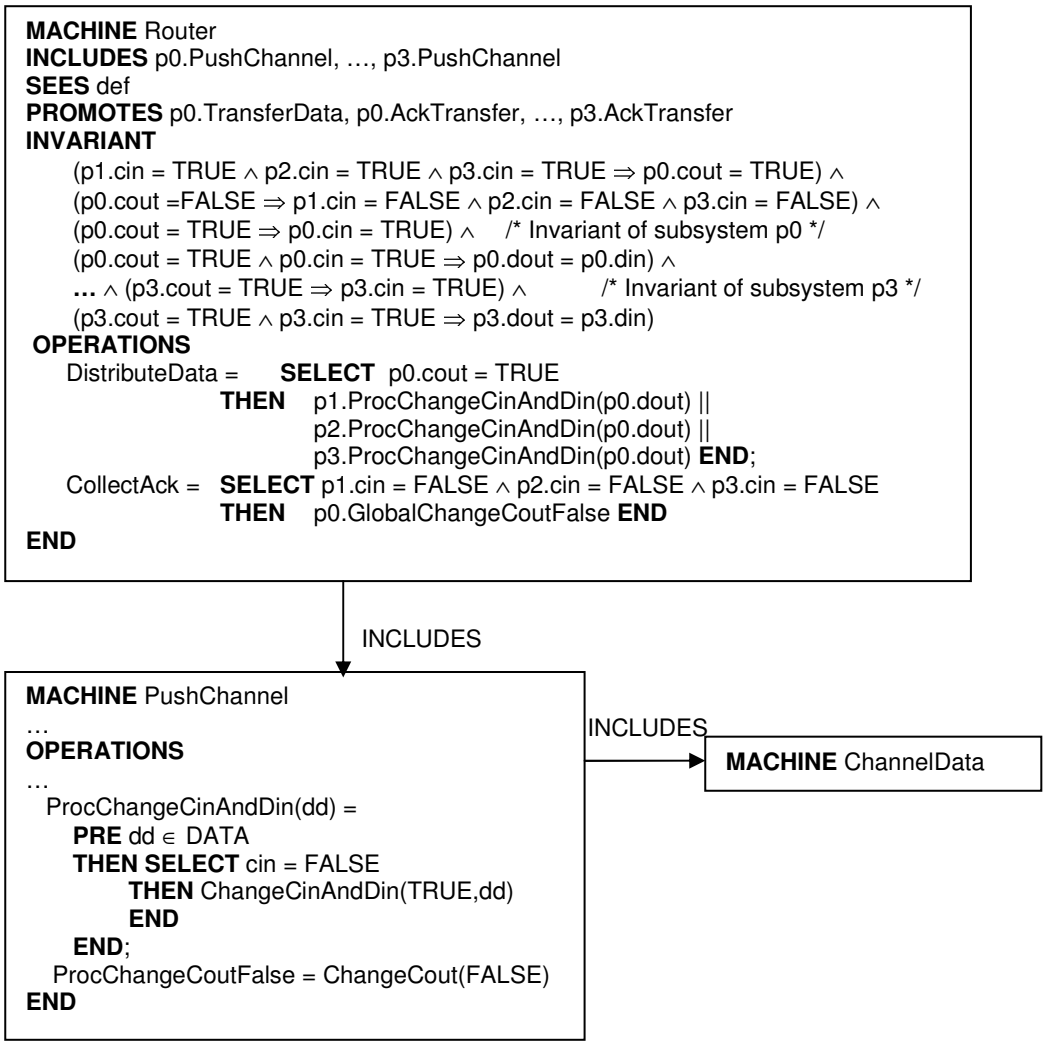


Fig. 9. The B action system for *Router*

Action *DistributeData* copies the request value of the communication and the data from the input channel to the output channels, so that the data can be propagated further to the neighbouring routers. Action *CollectAck* propagates backwards the acknowledgement values of the channels *p1*, *p2* and *p3* to indicate that the router is ready to receive new downward data packets via channel *p0* again. The values of the relevant global variables of the push channels are changed by calling the corresponding new procedures in *PushChannel*, *ProcChangeCinAndDin* and *ProcChangeCoutFalse*. The invariant of *Router* states that when the output channels, *p1*, *p2* and *p3*, have received the request (and the data), then the input channel, *p0*, must also have been requested. Furthermore, when the input channel has been acknowledged, then the output channels must also have been acknowledged.

4.3. Composing routers to a system module

In a network with a set of routers specified in Section 4.2, there is a possibility for packet cycling when data is propagated. This can be solved by *creating a module that controls a number of routers* in the network (Step (3)).

Let us consider the following example. A region of four nodes in a 2D network is shown in Fig. 10, where the routers are named *a*, *b*, *c* and *d*. We assume that a data packet leaves from some router in the network and that the packet arrives after some time at the routers *a* and *d*. The same packet will be at channel *p1* of *a* and at channel *p3* of *d*. In this case the data transfer is not allowed to take place over these channels and the routing control will take care of avoiding packet cycling.

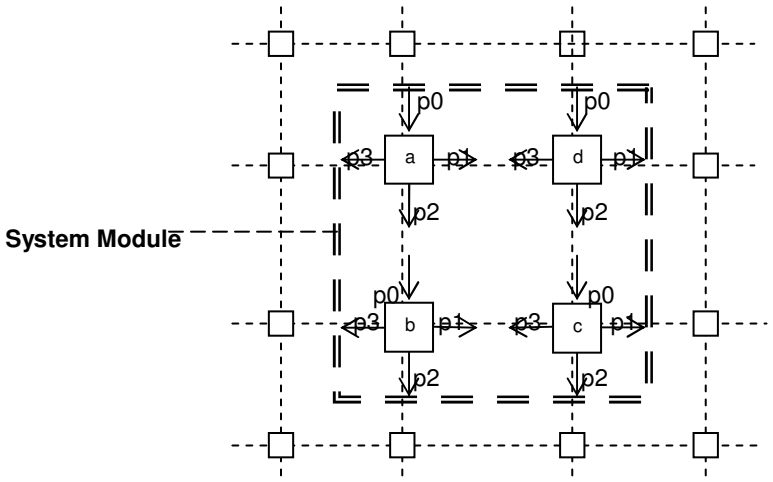


Fig. 10. Part of a two dimensional NoC mesh

For the B Action Systems modelling we first *add the interface procedures* (Step (3a)) to system *Router* by promoting them from system *PushChannel*. Via these procedures we can assign the request value and propagate the data to the input of channel *p0* and acknowledge the outputs of channels *p1*, *p2* and *p3*, as e.g., *p0.ProcChangeCinAndDin* and *p1.ProcChangeCoutFalse* shown in Fig. 11.

Finally, we design system *Module_abcd* to instantiate and include the relevant routers. *The actions of the routers are promoted* in this system (Step (3b)). In order *to control the routers we create new actions* (Step (3c)). The actions *TransferData_ab* and *TransferData_dc* propagate data downwards from routers *a* and *d* to the neighboring routers *b* and *c*, respectively. Except for these actions we need actions that detect the cycling. For example action *CheckCycling_ad* detects cycling between routers *a* and *d* and acknowledges the outputs of channels *a.p1* and *d.p3* canceling the data packets on them. The specification of *Module_abcd* is given in Fig. 11.

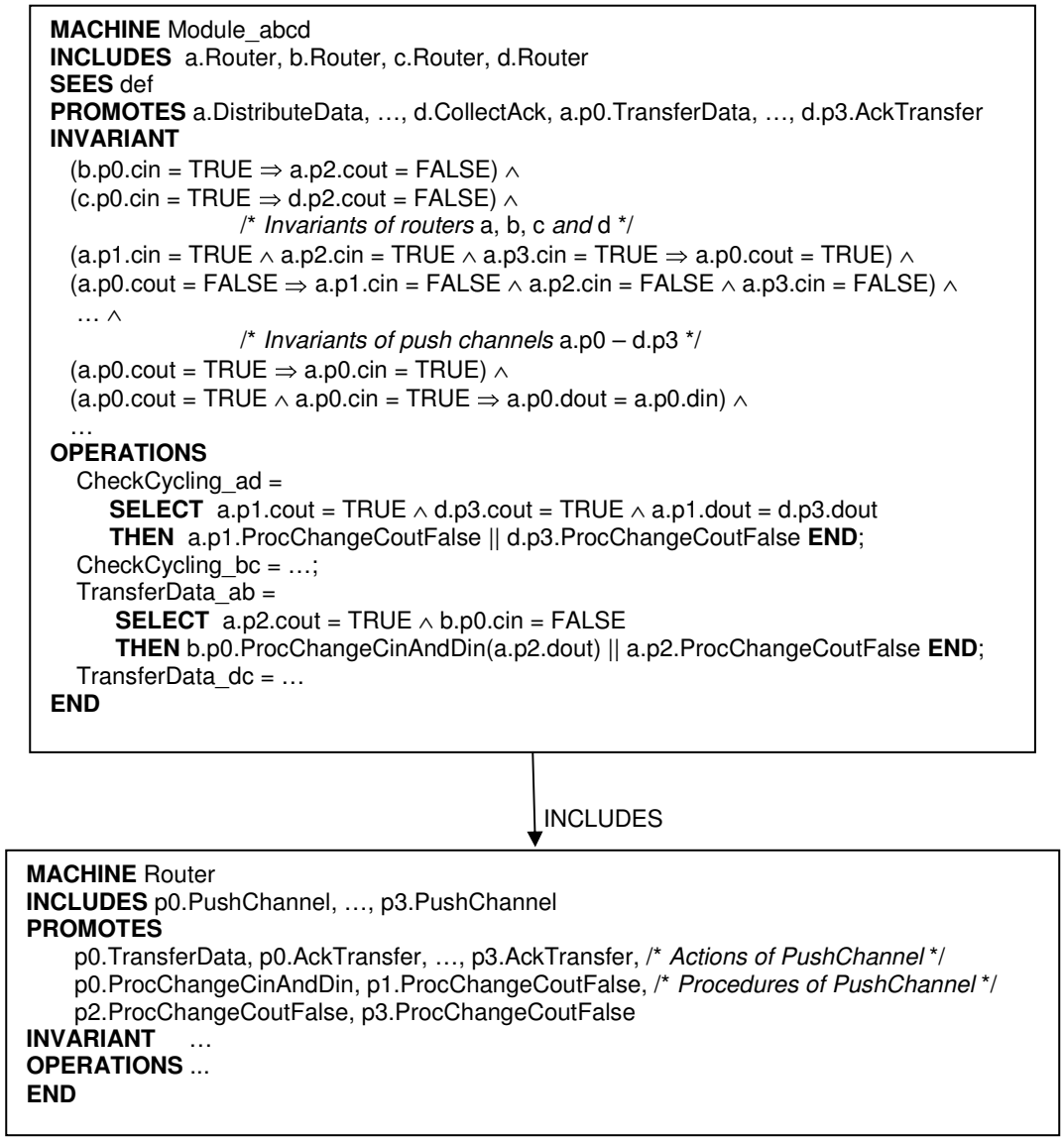


Fig. 11. Part of the B action system for *Module_abcd*

To complete the specification three more modules should be created to control the other routers on the four nodes in Fig. 10. These are the routers receiving data packets from the other three directions. By composing all these modules in parallel we achieve a well structured and efficient routing scheme for NoC systems. Depending on the application extra logic may be added to the routers and the modules controlling them.

4.4. Verification of the development

In order to prove that the system is valuable and satisfies its functional requirements we have to check that it fulfils its guaranteed behaviour, i.e., it is internally consistent.

Hence, to prove that the composed system *Module_abcd* is internally consistent we generate the proof obligations (C1) – (C3) for all its subsystems with the tool Atelier B. The automatic and interactive prover of the tool can then discharge the proof obligations.

Table I below shows that all the proof obligations for the routing scheme were discharged automatically. Most of the proof obligations are obvious due to the simple structure of the modules. Furthermore, there are no quantified expressions in the invariant. Our incremental structure of the invariant can be seen in the increased number of proof obligations in the modules higher up in the hierarchical composition.

Table I. Number of proof obligations for the whole system

Component	**Obvious p.o.**	**Generated p.o.**	**Autom. proved**
ChannelData	24	0	100%
PushChannel	19	0	100%
Router	183	1	100%
Module_abcd	548	0	100%
TOTAL	**774**	**1**	**100%**

5. Related work

A framework which has influenced our approach is *Reo* [4, 5, 6]. It is a recently introduced channel-based coordination model, where complex coordinators, also called connectors, are compositionally built out of simpler ones. The connectors coordinate components and both the connectors and the components are distributed and mobile. However, they only rely on the functionality of the subsystems and their simple interconnection, while our framework allows us to add and verify extra functionality in the composed system to control the subsystems.

In order to avoid the cycling of packets in the network, various routing schemes use the *odd-even* routing together with the adaptive routing mode [11, 18]. The odd-even routing prohibits specific turns in odd and even numbered columns of a network to stop the same data packet from cycling back to routers which have already received and distributed it. In order to achieve the results of this routing scheme we have embedded the required logic via the B Action Systems features into the system modules controlling the routers giving similar specifications independently of where they reside in the network.

Our abstract routing scheme can be compared to the *DyAD* (**Dy**namically switching between **A**daptive and **D**eterministic modes) routing scheme presented by Hu et al [18]. *DyAD* combines the advantages of both deterministic and adaptive routing [23], by switching between the two routing modes depending on the congestion levels on the network. Since *DyAD* combines the advantages of these routing modes, its performance is consistently better than purely adaptive routing schemes. Even if *DyAD* has a good performance, the main disadvantage of the method is its complexity. In order to

achieve the switching between the modes mentioned above, a *DyAD* router consists of many different complex components. Because of all this complexity, there is routing path selection delay and crossbar arbitration delay. Within our approach this complexity is eliminated by composing routers out of simple asynchronous channels relying on the request and acknowledgement phases of a communication. The flow control for providing efficient routing of data packets is taken care of by controlling modules composed out of simple routers.

In order to increase the maximum tolerable load of a NoC, a *proximity congestion awareness* technique is proposed by Nilsson et al [24] where routers use load information of neighboring routers for their own routing decisions. In our approach we keep the routers simple only allowing them to distribute data packets and the routing decisions are taken by the modules controlling them. This goes hand in hand with the requirement that the routers themselves must be as small and efficient as possible in order to limit the overhead in terms of size, power and routing path selection delay introduced by the NoC.

A formal approach for NoC system design using Action Systems is presented by Liljeberg [22] where the routers are asynchronous and use pipelining for the distribution of data. However, this approach lacks tool support for the formal development. Another approach to design NoC systems formally is given by Schmaltz et al [29]. They have tool support for their method, but their specifications contain complex arbitration logic and are not generic as ours. Al Sammane et al [3] also have tool support for their NoC system design. However, they consider only synchronous systems which are not of interest for us due to their disadvantages compared to asynchronous systems. Another formal and synchronous approach to specify protocols for NoC systems is given by Gebremichael et al [15], where PVS is used as a verification tool.

6. Conclusion

Formal methods with commercial tool support are important for the design of complex NoC systems in order to help to eliminate and correct errors in the early design phases. Even though, there has not yet been much work on formal specification of NoC systems. In this paper we have specified an abstract routing scheme for NoC system design in the formal B Action Systems framework having commercial tool support. We have given a new and convenient routing scheme on a NoC.

We proposed a compositional development method where we created the specification in a component-wise and hierarchical manner. We started by creating the basic components and composed controlling components in layers on top of them. For the NoC system development we first specified the channels in our framework. Then out of these channels we composed routers. Finally, we included these routers into modules that take care of the needed arbitration to select the path and to control the congestion levels on the network, as well as to eliminate the data packet cycling. The development steps were performed in the B Action Systems framework which enabled us to prove the consistency of the modules.

The B Action Systems formalism offers the notion of stepwise superposition refinement [7, 20] where abstract specifications can be stepwise refined towards executable implementations proving the correctness of each step. Here we have only given a B Action Systems specification of the routing scheme of a NoC system, but in our future work we will try to investigate how easy it is to generate implementations of these specifications that can be mapped to an asynchronous hardware language, as for example Haste [16]. In this way we will have implementations of verified NoC systems already in an early abstract state which in our opinion will further aid the design process.

Via the features of B Action Systems and the structured modeling we achieve the positive features of different NoC approaches without their level of complexity. Due to our modular approach of the development we can also reuse in different applications earlier defined and internally consistent channels and routers, which may reside in a library. We believe that the incremental compositionality of the routers and the modules controlling them, can help reducing the complexity and also the involved costs of NoC system design and development.

Acknowledgement

We would like to thank Professor Kaisa Sere for fruitful discussions on earlier versions of this paper.

References

[1] J.–R. Abrial. The B-Book: Assigning Programs to Meanings. Cambridge University Press, 1996 .

[2] J.–R. Abrial and L. Mussat. Event B Reference Manual, 2001. http://www.atelierb.societe.com/ressources/evt2b/eventb_reference_manual.pdf / (accessed 24.01.2006)

[3] G. Al Sammane, J. Schmaltz, D. Toma, P. Ostier and D. Borrione. TheoSim: Combining Symbolic Simulation and Theorem Proving for Hardware Verification. In Proc. of the 17th Symposium on Integrated Circuits and System Design. Pernambuco, Brazil, 2004, pp. 60 – 65.

[4] F. Arbab. A channel-based coordination model for component composition. Report SEN-R0203, CWI, 2002. Available at URL www.cwi.nl.

[5] G. F. Arbab and F. Mavaddat. Coordination through channel composition. In Proceedings of Coordination Languages and Models, volume 2315 of Lecture Notes in Computer Science, pp. 22 – 39, Springer, 2002.

[6] F. Arbab and J.J.M.M Rutten. A coinductive calculus of component connectors. Report SEN-R0216, CWI, 2002. Available at URL www.cwi.nl.

[7] R. J. R. Back and R. Kurki-Suonio. Decentralization of Process Nets with Centralized Control. Proc. of the 2nd ACM SIGAST-SIGOPS Symposium on Principles of Distributed Computing, pp. 131 – 142, 1983.

[8] R. J. R. Back and K. Sere. From modular systems to action systems. Software - Concepts and Tools 17, pp. 26 – 39, 1996.

[9] R. J. R. Back and J. von Wright. Compositional Action System Refinement. Formal Aspects of Computing, 15(2-3): 103 – 117, November, 2003.

[10] M. J. Butler and M. Waldén. Distributed system development in B. In H. Habrias (Ed.), Proc. of the First Conference on the B Method, Nantes, France, November, 1996, pp. 155 – 168.

[11] G.-M. Chiu. The odd-even turn model for adaptive routing. IEEE Transactions On Parallel and Distributed Systems, 11(7):729 – 738, July 2000.

[12] ClearSy, Atelier B, http://www.atelierb.societe.com/ (accessed 24.01.2006)

[13] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In Proc. of the 38th annual ACM IEEE Design Automation Conference, pp. 681 – 689, 2001.

[14] E. W. Dijkstra. A Discipline of Programming. Prentice-Hall International, 1976.

[15] B. Gebremichael, F.W. Vaandrager, M. Zhang, K. Goossens, E. Rijpkema and A. Radulescu. Deadlock Prevention in the Aethereal Protocol. In D. Borrione and W. Paul, editors. Proceedings 13th IFIP Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'05), Saarbrucken, Germany, LNCS 3725, pp. 345 – 348. Springer-Verlag, 2005.

[16] Handshake Solutions. http://www.handshakesolutions.com/ (accessed 24.01.2006)

[17] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Öberg, M. Millberg and D. Lindqvist. Network on a Chip: An architecture for billion transistor era. In Proceedings of the IEEE NorChip Conference, November 2000.

[18] J. Hu and R. Marculescu. DyAD – Smart Routing for Networks-on-Chip. In Proc. of the 41st annual ACM IEEE Design Automation Conference, pp. 260 – 263, 2004.

[19] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Örberg, K. Tiensyrjä and A. Hemani. A Network on Chip Architecture and Design Methodology. In Proceedings of IEEE Computer Society Annual Symposium on VLSI, pp. 117 – 124, April 2002.

[20] P. Liljeberg. On Self-Timed Communication Architectures for Network-on-Chip. Ph.D. Thesis, University of Turku, 2005.

[21] L. M. Ni and P. K. McKinley. A Survey of Wormhole Routing Techniques in Direct Networks. IEEE Transactions On Computers, 26: 62 – 76, Feb. 1993.

[24] E. Nilsson, M. Millberg, J. Öberg and A. Jantsch. Load distribution with the proximity congestion awareness in a network on chip. In Proceedings of the Design Automation and Test Europe (DATE), pp. 1126 – 1127, March 2003.

[25] A. Peeters. Single-Rail Handshake Circuits. PhD Thesis, Eindhoven University of Technology, 1996.

[26] J. Plosila, T. Seceleanu and K. Sere. Formal Communication Modeling and Refinement. Chapter 12. In J. Nurmi, H. Tenhunen, J. Isoaho, and A. Jantsch, editors, Interconnect-Centric Design for Advanced SoC and NoC, Kluwer, 2004.

[27] J. Plosila. Self-timed Circuit Design – The Action Systems Approach. Ph.D. Thesis, University of Turku, 1999.

[28] J. Plosila, K. Sere and M. Waldén. Asynchronous System Synthesis. Science of Computer Programming, 55 (2005), pp. 259 – 288. Elsevier.

[29] J. Schmaltz and D. Borrione. A Formal Approach to the Formal Specification of Networks on Chip. In Proc. of the 5th Intl. Conference on Formal Methods in Computer-Aided Design (FMCAD'04), pp. 52 – 66, 2004.

[30] T. Seceleanu. Systematic Design of Synchronous Digital Circuits. Ph.D. Thesis, Åbo Akademi University, 2001.

[31] K. Sere and M. Waldén. Data Refinement of Remote Procedures. Formal Aspects of Computing, 12(4):278-297, 2000.

[32] L. Tsiopoulos, J. Plosila and K. Sere. Modeling SoC Systems with Flow Control. TUCS Technical Report, No 746, available at: www.tucs.fi, February 2006.

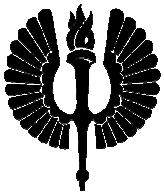
[33] A. Vitkovski, R. Haukilahti, A. Jantsch and E. Nilsson. Low-Power and Error Coding for Network-on-Chip Traffic. In Proceedings of the IEEE NorChip Conference, November 2004.

[34] M. Waldén. Distributed load balancing. Chapter 7 in E. Sekerinski and K. Sere (eds.). Program Development by Refinement - Case Studies Using the B Method. Springer-Verlag, 1998, pp. 255 – 300.

[35] M. Waldén and K. Sere. Reasoning About Action Systems Using the B Method. Formal Methods in Systems Design 13(5 – 35). Kluwer Academic Publishers, 1998.

Turku
Centre *for*
Computer
Science

Lemminkäisenkatu 14 A, 20520 Turku, Finland | www.tucs.fi



University of Turku
- Department of Information Technology
- Department of Mathematics

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration
- Institute of Information Systems Sciences

ISBN 952-12-1695-6
ISSN 1239-1891