Towards Dependable Placement of NoC Resources

Leonidas Tsiopoulos
Department of Information Technologies,
Åbo Akademi University,
Turku, Finland
leonidas.tsiopoulos@abo.fi

Abstract. In this paper we present an approach on how executable formal specifications of Network-on-Chip routing schemes can help on deciding efficient placement of processing resources on 3D-integrated systems. We use a routing scheme specified with the B Action Systems formalism and we execute it with the model checking and animating tool ProB in order to obtain traces of operation executions based on different data flow scenarios.

1 Introduction

Advances in technology allow us to have thousands of computational resources on a single electronic chip. Recently the Network–on–Chip (NoC) communication paradigm [6, 8] has been proposed as the alternative to bus–based Systems–on–Chip (SoC) communication paradigms. Bus–based communications offer insufficient bandwidth for concurrent on–chip data transactions whereas regular NoC interconnects can scale incrementally and data is routed in a distributed and flexible manner.

The number of resources which can be placed on a single chip increases rapidly with the continuous technology scaling resulting in growing wire delay and increased power consumption while meeting the constraints on interconnect latency is an issue. To overcome these challenges, 3D–integrated systems [7, 10] have been proposed during the last years with the main advantage of reduced length of the global interconnect which in turn reduces power consumption considerably. These complex systems are being used in many important applications, such as in automotive and advanced control systems, hence, it is important that they are reliable. Appropriate methods are needed in order to specify reliable 3D–integrated systems, as well as to model the communication and verify their design.

Formal methods of concurrent programming with adequate tool support are important for the development of complex 3D–integrated systems in order to avoid costly errors in the later design phases and contribute to the *dependability* of such systems. The B Action Systems [4, 13] is a state–based formalism

which was created in order to be able to reason about parallel and distributed systems, like Action Systems [3], within the B Method [1]. We use a hierarchical formal specification of an asynchronous NoC routing scheme [12] in order to show how executable traces of this formal model can help on deciding an efficient placement of communicating resources on a chip for reduced power consumption which in turn aids the reliability of the system.

The organization of this paper is as follows. In Section 2, we present a short introduction to the B Action Systems. In Section 3 we discuss the model of the NoC routing scheme considered for this paper. Section 4 discusses the trace generation procedure within ProB [9] which is a model checking and animating tool for B specifications. Section 5 discusses the related work. Finally, Section 6 concludes this paper.

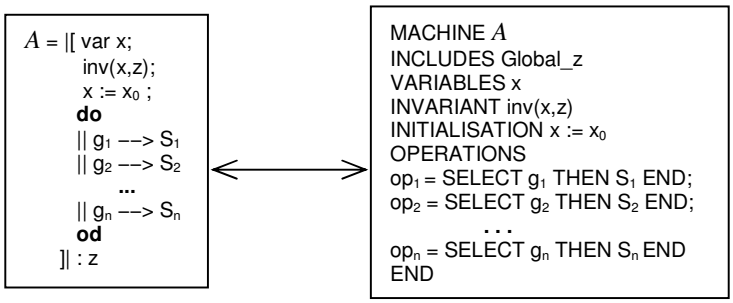
2 B Action Systems

B Action Systems [4, 13] is a state–based formalism based on Action Systems [ref] and the B Method [1] and was created in order to be able to reason about parallel and distributed systems, like Action Systems, within the B Method. Note that a model in B Action Systems is a valid model within the B Method; therefore, tool support for the B Method [5, 9] can be used to analyze models in B Action Systems.

The structure of an action system *A* is shown in Figure 1(a) in which z and x are respectively the global and local state variables. The action system interacts with the environment through the global variables. State variables have types and the set of possible assignments to them constitutes the state space. The statement $x := x_0$ assigns initial values to the local variables. Each action has the form g_i --> S_i, where $i = 1..n$, in which g_i is the guard and S_i is a statement on the state variables. An action is enabled only if its guard evaluates to true. As regards to the behaviour of an Action System [3], the initialization statement is executed first, and thereafter as long as there are enabled actions, one of the enabled actions is selected non–deterministically for execution. When there are no enabled actions, the system terminates.

An action system can be translated to a B abstract machine – an abstract machine is the basic unit of specification in B – as shown in Figure 1(b). This is an action system within B, and is called a B Action System. The system is identified by a unique name. The local 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 op = SELECT g THEN S END, where g is

said to be the guard and S the body. The guard g is a predicate on the variables, and when g holds the action op 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, analogous to the behaviour of an action system. B Action Systems can be composed to model parallel systems.



(a) (b)

Fig. 1. Structure of a B Action System.

Structuring mechanisms such as SEES, INCLUDES and PROMOTES can be used to express B Action Systems as a composition of subsystems [1]. The SEES-mechanism allows read access to the seeing system. The INCLUDES-mechanism allows write access to the variables of the included system. Actions of the included system can also be made available by promoting them into the including system within a PROMOTES-clause. The structuring mechanisms provide an efficient way to model system hierarchy.

Communication can occur between two B Action Systems in two ways: by use of global variables, or by global procedures; the example of this paper uses global procedures for communication purposes. Operations do not take any parameter and they execute autonomously; procedures may or may not have parameters, and they are to be invoked in a certain context. Consider the operation op_p = SELECT g_p THEN S_p || Proc END; this operation is enabled only if the procedure Proc is also enabled. The action and the procedure in its body are executed as an atomic entity.

3 The Executed Formal Specification

Tsiopoulos and Waldén [12] have specified a formal routing scheme relying on the request and acknowledge phases of the asynchronous communication. Within this asynchronous NoC routing scheme, data packets are received at the

input channels of the routers and they are distributed to their output channels for subsequent propagation to the neighboring routers. The routers acknowledge their input channels so that new data packets can be received. Controlling components of routers propagate the data packets to routers that have not received them yet, and prohibit the cycling of data packets to occur. This is because cycling of data packets back to routers which have received them already, reduces on-chip interconnect efficiency and increases power consumption. A hierarchical and compositional development method was used [12] for this purpose. Figure 2 outlines part of the hierarchical structure of the specification of the NoC routing scheme and presents one of the communications between the subsystems using global procedures.

Starting with the simplest subsystem of the routing framework, an asynchronous channel component called PushChannel (data propagation upon request) was specified as a B action system. The channel's inputs and outputs were modeled as global variables. Of these global variables, the necessary input and output asynchronous control handshakes were modeled with boolean variables named cstart and cend respectively, and the input and output data were captured with variables named dstart and dend of the generic type DATA. These variables together with simple procedures to update their state were defined in a separate machine named ChannelData which was included in PushChannel. The latter had two operations: one to propagate data and the request from its input to its output and the other to acknowledge its input after the output data was taken, so that new data and request could be received. The interface of PushChannel consists of two global procedures, ProcChangeCstartAndDstart and ProcChangeCendFalse; the first to update its input with a new request and data and the second to acknowledge its output after the transfer of the output data.

A B action system named Router including eight instances of PushChannel was then created. We note that twelve instances of PushChannel are needed to model propagation of data along the third dimension too. Four actions were specified for controlling the channels and propagating the data along them. For example, action TransferData_N copies the request value of the communication and the data from the output of the input northern channel inN to the inputs of the output channels, outE, outS, and outW so that the data can be propagated further towards the neighboring routers. Simultaneously it sends acknowledgment to the output of channel inN to indicate that Router is ready to receive new data via this channel. Note that six actions are needed to model distribution of data along the third dimension.

Finally, some instances of this router (000.Router, 010.Router, 020.Router, 030.Router, …; the digits correspond to the *xyz* coordinates on the network) were composed into a controlling system named NoC_Region1 (given a new name for

the purposes of this paper) which controlled the data distribution between the routers in a region of the NoC. In order to do so, some of the global interface procedures of PushChannel were promoted within Router to form its interface.

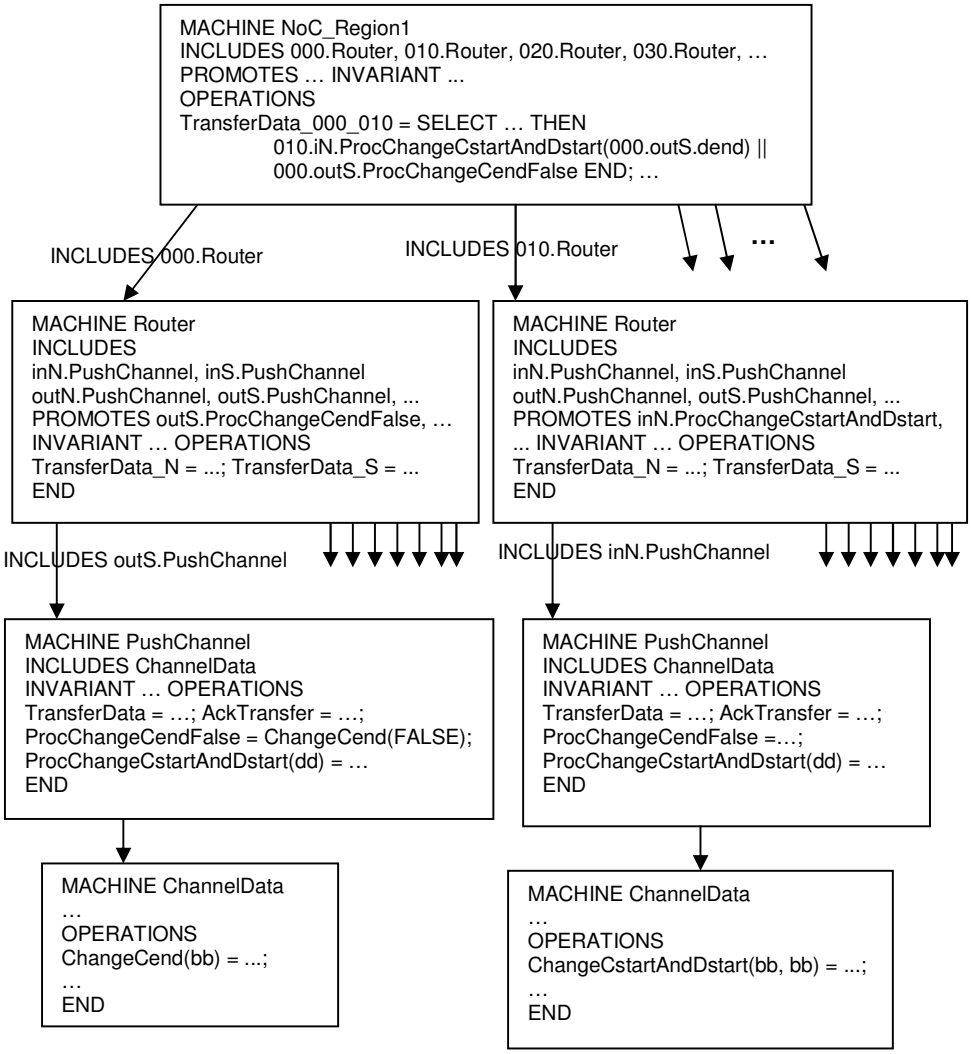


Fig. 2. Machine hierarchy in the NoC specification.

Refer to action TransferData_000_010 of machine NoC_Region1 in Figure 2. This action transfers data from the southern output channel of the router with

coordinates 000 to the northern input channel of router with coordinates 010. The guard checks whether channel 000.outS is ready to transmit and channel 010.inN is ready to receive. If so, the data value of channel 010.inN gets the data value of channel 000.outS. And the flags of both the channels are modified to indicate that the transmission has taken place. The global procedures ProcChangeCstartAndDstart and ProcChangeCendFalse defined in machine PushChannel do this modification. See in the figure, how these global procedures have been promoted to the interface of the router.

4 Trace Generation Procedure

ProB is a model checking and animation tool for B machines [9]. ProB includes a fully automatic animator written in SICStus prolog. ProB takes an *instantiated* model in B, in which any generic set has been instantiated with some concrete values to avoid *state explosion* and generates a finite coverage graph. Within this finite coverage graph, several traces of sequenced operation executions can be obtained. In other words, several different scenarios of operation execution paths for data propagation between resources on a chip can be created by the user.

Figure 3 shows part of one of the planes of a 3D–integrated system with a mesh topology corresponding to the formal routing specification presented in the previous section. Let us assume that a processing element at position 000 wants to send data to another element at position 750. One possible path for this communication is shown in Figure 3.

Each trace of operation executions in ProB originates from some initial state. Table 1 shows the trace of operation executions within the hierarchical B action systems specification corresponding to the communication path shown in Figure 3, including a scenario for possible delays. These delays occur because at the same points the needed operations for the data propagation are being executed for another trace of data communication intersecting the trace shown in Table 1. When these operations are enabled again, the execution of the operations in the path continuous until the data arrives at the destination.

Modelling such data propagation scenarios of formal NoC routing specifications can aid the process of finding a suitable placement of frequently communicating resources on a chip already at an early design level.

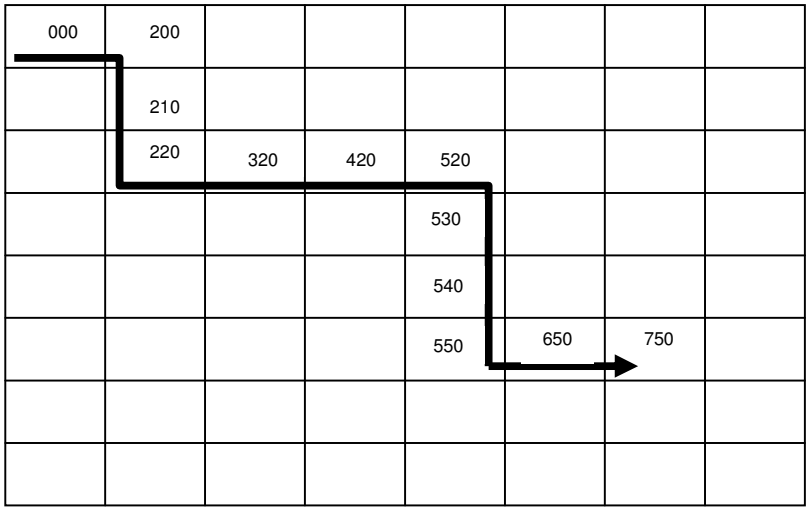


Fig. 3. A possible data propagation path from source 000 to destination 750.

000.localin.TransferData; 000.TransferData_local;
000.outE.TransferData; TransferData_000_200;
200.inW.TransferData; …**delay**; 200.TransferData_W;
200.outS.TransferData; …**delay**; TransferData_200_210;
210.inN.TransferData; …**delay**; 210.TransferData_N;
210.outS.TransferData; TransferData_210_220;
220.inN.TransferData; 220.TransferData_N; …**delay**;
220.outE.TransferData; TransferData_220_320;
320.inW.TransferData; 320.TransferData_W;
320.outE.TransferData; TransferData_320_420;
420.inW.TransferData; 420.TransferData_W;
420.outE.TransferData; …**delay**; TransferData_420_520;
520.inW.TransferData; 520.TransferData_W;
520.outS.TransferData; TransferData_520_530;
530.inN.TransferData; 530.TransferData_N;
530.outS.TransferData; TransferData_530_540;
540.inN.TransferData; 540.TransferData_N; …**delay**;
540.outS.TransferData; TransferData_540_550;
550.inN.TransferData; 550.TransferData_N; …**delay**;
550.outE.TransferData; TransferData_550_650;
650.inW.TransferData; 650.TransferData_W;
650.outE.TransferData; TransferData_650_750;
750.inW.TransferData; 750.localout.TransferData

Table 1. A trace of B operation invocations for data propagation.

5 Related Work

Several works have been presented during the last years on the mapping and placement of resources on 2D and 3D NoC designs. For example, Addo-Quaye [2] presented an approach for thermal and communication–aware mapping and placement for 3D NoC architectures. *Genetic algorithms* were used in order to model solutions to mapping problems. The approach was shown to reduce communication and peak temperature when compared to purely random placements.

Murali et al. [11] presented a method to determine a power efficient topology of a 3D System–on–Chip for a given application and for finding paths for the traffic flows that meet *Through Silicon Vias* constraints. The method accounts for power and delay of both switches and links. The assignment of resources to different 3D layers and the floorplan of the resources in each layer are taken as inputs to the synthesis process, while the output of the process is the optimal positions of NoC switches in each layer and between the layers.

To the best of our knowledge there has not been presented another approach to execute formal specifications of NoC routing schemes in order to aid the process of finding efficient placements of resources on a chip. We believe that our approach can enhance the design of complex 3D–integrated systems and work hand in hand with mapping and placement approaches such as the ones presented in this section.

6 Conclusion

Formal methods with adequate tool support are important for the design of complex NoC systems in order to correct errors in the early design phases and reduce the involved costs of NoC system design and development.

We proposed an approach for finding an efficient placement of resources on a NoC after *data flow* executions at the formal specification level within ProB. To the best of our knowledge it is the first approach on executing several different data flow scenarios within a formal NoC framework in order to aid the process of finding an efficient placement of resources on a chip. This development step is important in order to design reliable and dependable multidimensional integrated systems.

References

[1] Abrial J.–R. (1996). The B–Book, Cambridge University Press.
[2] Addo–Quaye, C. (2005). Thermal-aware mapping and placement for 3-D NoC designs. In Proc. IEEE Int. Syst.-on-Chip Conf., pp. 25–28.
[3] Back, R.J.R., Kurki–Suonio, R. (1983). Decentralization of Process Nets with Centralized Control, Proc. of the 2nd Symposium on Principles on Distributed Computing, pp. 131–142.
[4] Butler, M., Waldén, M. (1996). Distributed System Development in B, Proc. of the 1st conference on the B Method, Nantes, France, pp. 155–168.
[5] ClearSy, Atelier B, http://www.atelierb.societe.com/
[6] Dally, W. J. and Towles, B. (2001). Route packets, not wires: On–chip interconnection networks. In Proc. of the DAC'01, pp. 681–689.
[7] Gutmann, R. J., Lu, J. Q., Kwon, Y., McDonald, J. F., Cale, T. S. (2001). Three-dimensional (3D) ICs: a technology platform for integrated systems and opportunities for new polymeric adhesives. In Proc. Conf. Polymers Adhesives Microelectron. Photon. Pp. 173–180.
[8] Hemani, A., Jantch, A., Kumar, S., Postula, A., Öberg, J., Millberg, M., and Lindqvist, D. (2000). Network on a Chip: An architecture for billion transistor era. In Proc. of the IEEE NorChip Conference.
[9] Leuschel, M., Butler M. (2005). ProB: A Model Checker for B, Proc. FME'03, LNCS Volume 2805, Springer, pp. 855–874.
[10] Li, F., Nicopoulos, C., Richardson, T., Xie, Y., Narayanan, V., Kandemir, M. (2006). Design and Management of 3D Chip Multiprocessors Using Network-in-Memory. ACM SIGARCH Computer Architecture News. Volume 34 , Issue 2. pp. 130 – 141.
[11] Murali, S., Seiculescu, C., Benini, L., De Micheli, G. (2009). Synthesis of Networks on Chips for 3D Systems on Chips. Proceedings of the 2009 Conference on Asia and South Pacific Design Automation. pp. 242-247.
[12] Tsiopoulos, L., Waldén, M. (2006). Formal Development of NoC Systems in B, Nordic Journal of Computing, Vol. 13 (2006). pp. 127–145.
[13] Waldén, M., Sere, K. (1998). Reasoning about Action Systems using the B Method, Formal methods in System Design, Vol. 13(1), pp. 5–35.