3

Petri Net Based Modelling of Communication in Systems on Chip

Holger Blume, Thorsten von Sydow, Jochen Schleifer and Tobias G. Noll
Chair of Electrical Engineering and Computer Systems
RWTH Aachen University
Germany

1. Motivation

Due to the progress of modern microelectronics the complexity of integrated electronic systems is steadily increasing. For example, the number of transistors which can be integrated on a single piece of silicon doubles every 18 months according to Moore's Law (Moore, 1965). At the same time, the costs for manufacturing deep-sub-µ devices with feature sizes down to 45 nm are dramatically increasing.

Due to this progress, today, complete systems are integrated on a single silicon die as so-called Systems on Chip (SoCs). The huge complexity of these SoCs and the very high manufacturing costs demand sophisticated design strategies as it is not possible to simulate a sufficiently large number of implementation alternatives in advance. Furthermore, errors within the design process lead to dramatically increased costs.

Therefore, the field of model based design space exploration (DSE) is of increasing importance. Model based DSE allows a reduction of the number of implementation alternatives in an early stage of the design process by quantitative analysis of possible implementation alternatives (Blume, 2005).

Especially, the design of a sophisticated communication structure on a SoC is of great interest. For SoCs with moderate complexity mainly bus based communication structures are applied, but this is not sufficient for modern high complex SoCs, since bus based communication provides only very limited scalability, reduced bandwidth and no guaranteed latencies. Furthermore, with a high number of system components the need for simultaneous communication between different communicating units increases. All these requirements are already known from multi computer networks. Therefore, for complex on-chip communication requirements also network-like structures are considered. Hence, the concept of multi computer networks is transferred and adapted to on-chip communication problems building so-called Networks on Chip (NoCs) featuring in future the communication infrastructure for many processor cores.

Generally, NoCs consist of

- network-interfaces (NI), where clients like e.g. processor cores can access the NoC,
- routing-switches (RS), which route the data through the NoC and
- links, through which the data is transported (see Fig. 1).

Source: Petri Net,Theory and Applications, Book edited by: Vedran Kordic, ISBN 978-3-902613-12-7, pp. 534, February 2008, I-Tech Education and Publishing, Vienna, Austria

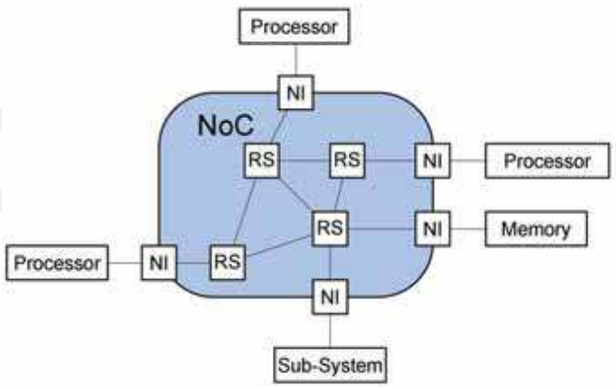


Fig. 1. Network-on-Chip (NI: Network-Interface, RS: Routing-Switch)

These NoCs imply a huge parameter space featuring parameters like network topology, routing strategy, link properties, arbitration mechanisms etc.. Some of these parameters are roughly sketched in the following:
Topology in this case is the way of connecting the various network components to each other, common examples (see Fig. 2) are networks based on mesh, torus and ring topologies (Bjerregaard, 2006), besides further regular topologies also heterogeneous and/or hierarchical topologies as well as completely irregular ones (for example optimized for specialized signal processing tasks) are discussed.

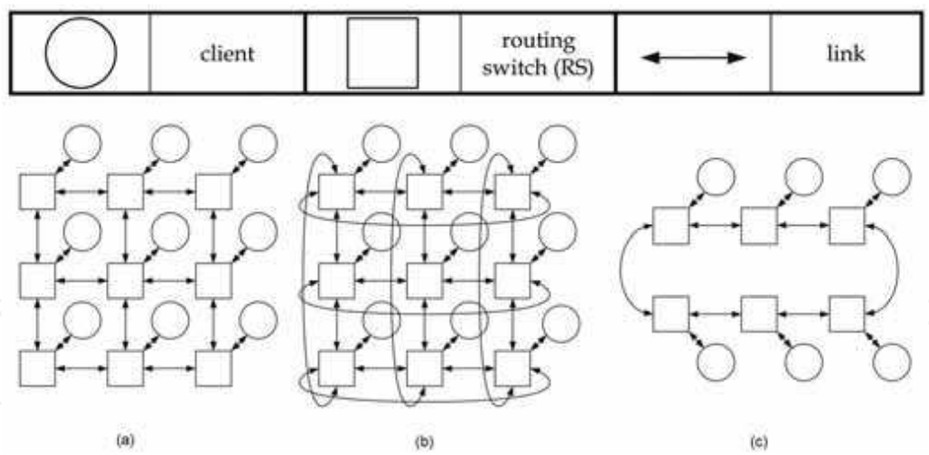


Fig. 2. Common NoC topologies: mesh (a), torus (b), ring (c)

The *switching concept* defines the way information is sent through the network. Concepts for this are line and packet switching. In a line switched network a complete route from source to destination is established before information is passed along this route. In the packet switching approach information is divided into small packets that are delivered

independently of each other. While line switching generates a considerable overhead for route establishment there is no need to send destination information along with each message and vice versa for packet switching. Furthermore, concepts such as wormhole routing combine characteristics of both approaches (Duato, 2003).

In any of the cases described before, the actual route through the network is determined by the *routing algorithm*. In each network node, information is routed from an input to an output port according to the routing algorithm. These algorithms can be divided into static and adaptive algorithms as well as minimal-path and non-minimal-path ones. When using a static routing algorithm there is only a single route for each possible pair of source and destination. Adaptive algorithms allow for different routes dependent on the current network state and generally tend to reduce congestion for the cost of higher complexity. Minimal-path routing algorithms only consider those routes with minimum possible length while non-minimal-path algorithms also regard further routes featuring non-minimal path lengths.

Arbitration mechanisms define the way to resolve resource conflicts; this can range from simple first-come-first-serve or Round Robin schemes to complex schemes including priority access and disruption of routes.

All of these parameters have a significant influence on congestion, latency and network load, thereby affecting and possibly limiting SoC and NoC performance.

It is a key task of modern SoC and NoC design to efficiently explore the design space regarding aspects like performance, flexibility and power consumption presumably in an early stage of the design flow in order to reduce design time and design costs.

Different approaches for exploring the design space concerning performance aspects have been proposed:

- Emulation on FPGA based platforms (Neuenhahn, 2006)
- Simulative approaches, e.g. applying SystemC (Kogel, 2003), (Madsen, 2004), (Sonntag, 2005)
- Combined simulative-analytic approaches (Lahiri, 2001)
- formal communication modelling and refinement systems applying dedicated modelling languages like the Action Systems Formalism (Plosila, 2004)
- stochastic approaches applying Markov Models (Mickle, 1998), Queuing Theory (Kleinrock, 1975) or different forms of Petri Nets incl. deterministic and stochastic Petri Nets (DSPN) (Ciardo, 1995), (Blume, 2006), (Blume, 2007) and Coloured Petri Nets (CPN) (Zaitsev, 2004)

Each of these techniques provides its individual advantages and disadvantages. For example, simulative approaches based on SystemC like (Kogel, 2003) provide highly accurate results but suffer from long simulation times, making them not appropriate for an early stage of communication modelling and evaluation. Emulation of communication architectures and scenarios on FPGA based platforms (Neuenhahn, 2006) provides on one hand the possibility to quickly acquire results for different aspects. If a suitable FPGA based model is available it is much faster to attain results than using a simulation based method. On the other hand the modelling and realization effort of the emulation (incl. the synthesis of the NoC on the FPGA) is very high. The complexity of the modelled scenarios is limited by the capacity of the used FPGA. Recently, communication modelling approaches which are based on so-called deterministic and stochastic Petri Nets (DSPNs) have been presented. In (Blume, 2006), (Blume, 2007) it could be shown that with the application of these DSPN

modelling techniques it is possible to efficiently trade modelling effort and modelling accuracy. Basic but exemplary test scenarios like resource conflicts in state-of-the-art DSP architectures, basic bus based communication test cases and basic NoC structures demonstrate a very good modelling accuracy at low modelling effort.
In this chapter the usability of different Petri Net based modelling techniques like DSPNs and CPNs for modelling complex NoC communication scenarios is investigated and their specific properties are discussed. This chapter is structured as follows: section 2 provides an introduction into the Petri Net variants DSPN and CPN. In section 3 these modelling techniques are applied in order to model different forms of on-chip communication. The corresponding accuracy of the models compared to values which were derived using FPGA and DSP based testbeds are provided and discussed. Furthermore, the related modelling effort is analyzed. Section 4 provides a conclusion and a short outlook to possible future applications of Petri Net based techniques in the domain of design space exploration for NoC-architectures.

2. Introduction to Petri nets

In the following section a short introduction to the Petri Net methods which have been applied in context of this chapter is given. The modelling with these specific methods will be illustrated by means of descriptive basic examples. First of all, the design with so called deterministic and stochastic Petri Nets (DSPNs) is sketched. Afterwards, coloured Petri Nets which extend the possibilities of DSPNs are briefly presented.

2.1 Deterministic and stochastic Petri nets

Deterministic and stochastic Petri Nets have been introduced in 1987 by Ajmone Marslan and Chiola (Ajmone Marslan, 1987) as an extension of classical Petri Nets. DSPNs extend the modelling possibilities of classical Petri Nets by introducing the concept of deterministic transition times. In the following, only a subset of all features provided by DSPNs is discussed. For a thorough overview see e.g. (Lindemann, 1998).
Petri Nets consist of so-called places, arcs and transitions. Places, depicted as circles in the graphical representation, correspond to states of e.g. system components. E.g. a place could be named *copy word* to illustrate that this place represents the state of copying a word. Places can be unmarked or marked with one or even more tokens. This illustrates that the corresponding place is currently allocated. E.g. if a place called *copy word* is marked, the associated component is in the state of copying a word.
In Petri Nets a state change is modelled by means of so called (timed) transitions. Three types are differentiated in DSPNs: immediate transitions, transitions with a probability density function of their delay (e.g.: negative exponential) or deterministic transitions with a fixed delay.
Transitions and places are connected via arcs. There are two types of arcs, regular or inhibitor arcs. Inhibitor arcs are identified by a small inversion circle instead of an arrowhead at the destination end of it (see Fig. 3). If more than one input place is connected to a transition via regular arcs, the transition will only be activated when all connected places are marked. In case of one or more of these arcs being inhibitor arcs the transition will not fire if the corresponding place is marked. Furthermore, a numeric weight can be

assigned to each arc. A weighted arc is only activated if the number of tokens, located in the place the arc is originating from, is greater or equal than the assigned weight.
The form of graphical representation is often used to build DSPNs. The underlying mathematical representation of DSPNs can be specified as a nine-tuple

$$DSPN = (P, T, I(\cdot), O(\cdot), H(\cdot), \Pi(\cdot), M_0, D(\cdot), W(\cdot))$$

with

- P, a finite number of places,
- T, a finite number of transitions,
- $I(\cdot), O(\cdot), H(\cdot)$ denote the input-, output- and inhibitor-functions, which connect transitions and places,
- $\Pi(\cdot)$ denotes the firing-priority-function (specifying firing-priority-levels) for all immediate transitions,
- M_0 denotes the initial marking of the DSPNs,
- $D(\cdot)$ denotes the firing-delay-function (specifying the average delay) for timed transitions,
- $W(\cdot)$ denotes the firing-weight-function, which specifies the weights which are associated to each transition.

When a Petri Net model has been implemented by use of a graphical design tool or by directly defining the characteristic DSPN nine-tuple, the belonging mathematical models of the implemented Petri Net can be analyzed. Then, this analysis yields for example

- the static expectation value of the marking of places (occurrence of tokens at a given place),
- the stationary probability for the occurrence of a specific marking of a place,
- the average number of tokens passing a transition per unit of time, i.e. the throughput of a transition.

For the acquisition of results three different approaches exist:

- **mathematical analysis**, within which a closed equation system is deduced from the Petri Net and this equation system can be solved in order to acquire the desired results,
- **mathematical approximation**, which is based on numeric methods of calculation being suited for Petri Nets, which cannot be solved in the form of closed equation systems,
- **simulation**, within which the flow of tokens through the net is simulated. This simulation is carried out until the desired results can be computed according to a specified confidence bound. Therefore, the relative occurrences of tokens within the single places are acquired.

Each alternative provides its specific advantages and disadvantages regarding required computational effort, achievable accuracy etc.. In case of two or more concurrently enabled deterministic transitions, mathematical analysis is not possible and simulative or approximative methods have to be applied (Lindemann, 1998).
A further advantage of Petri Nets is the availability of comfortable mathematical methods in order to determine features of Petri Nets such as the so-called liveness or the absence of deadlocks (non-resolvable blockades) (Lindemann, 1998). The associated mathematical methods are often included in the modelling tools and therefore allow a fast verification of features like absence of deadlocks.

In order to demonstrate the application of DSPNs to model communication structures a basic DSPN is depicted in Fig. 3. A simplified arbitration scheme which handles the competition of a DMA controller and a CPU for the critical resource memory interface is modelled here. The DSPN consists of two components: a section of a CPU and a DMA-controller.

In the following, two aspects of the current state and their implications for the following state of this simple net will be explained. As can be seen in Fig. 3 the *memory request* place of the CPU and the *memory access granted* place are connected to the immediate transition ② via regular arcs. These two places are the only places which are connected to this transition and both are marked with tokens. Thus, the transition is going to fire immediately. The mentioned places are going to be cleared and the *memory access* place of the CPU is marked. This transition example describes the situation where the CPU requests the memory at a time where the *memory access* is available. The CPU accesses the memory and transfers data from or to the memory. The resource memory is therefore busy until the deterministic transition ③ fires and the place *memory access granted* is marked again. Thus, access to the memory by another device (here the DMA-controller) cannot be granted.

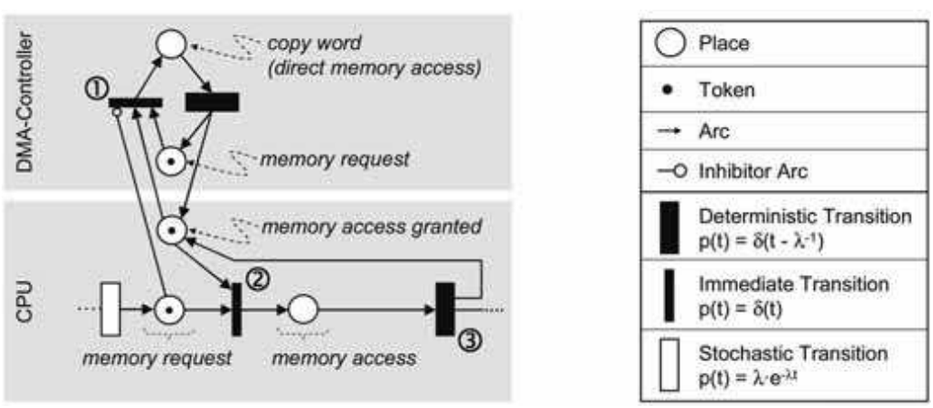


Fig. 3. Basic DSPN example (depicted here for a specific state)

The upper immediate transition ① of the DSPN depicted in Fig. 3 behaves differently compared to transition ② as one of the three places connected to transition ① is connected via an inhibitor arc. Therefore, this transition is not going to fire as long as all three connected places are marked. In case that the *memory request* place of the CPU is not marked and the other ones are marked, transition ① will fire immediately. Thus, the DMA-controller only gets access to copy a word if the CPU is not having or requesting memory access. Therefore, in this arbitration scheme the CPU has higher priority than the DMA.

The described DSPN requires input parameters such as the memory access delay time $T_③$ etc. to determine probabilities and expectations of previously defined places as mentioned above.

A variety of DSPN modelling environments is available today (Petri Nets World, 2007). For the DSPN modelling experiments described in this chapter, the modelling environment DSPNexpress (DSPNexpress, 2003) has been applied. DSPNexpress provides a graphical editor for DSPN models, as well as a solver backend for analysis of DSPNs. Experiments can

be performed for a fixed parameter set and for a parameter sweep across a user-defined range of values. The package supports the computation of the transient response e.g. the distribution of tokens (using Picards Iteration Algorithm) as well as computation of the steady state behaviour of the DSPN model. The latter can be determined by iteratively using the Generalized Minimal Residual Method, by employing the direct quadrature method or by utilizing the discrete event simulator backend (Lindemann, 1998). These methods correspond to the DSPN computation methods mentioned in the beginning of this section.

2.2 Coloured Petri nets

In this section a short introduction to Coloured Petri Nets (CPN) and to the software CPNtools (Ratzer, 2006) that has been used for the modelling examples discussed here, is given. First, the basic features of this modelling approach are presented then it is explained using a basic application example.

Coloured Petri Nets have been developed by K. Jensen in course of his PhD thesis (Jensen, 1980) to expand the modelling possibilities of classical Petri Nets. Like other forms of Petri Nets a CPN consists of places, tokens, transitions and arcs. The primary feature unique to CPNs is the inclusion of data structures into tokens. These data structures are called coloursets and resemble data structures in high level programming languages; they can range from simple data types such as integers to complex structures like structs or unions in C/C++. Similar to programming languages it is possible to define variables associated with these coloursets. Some examples of colourset and variable definitions are shown in Fig 4.a. Tokens as well as places of a CPN are always associated with a colourset and a place may only contain tokens of the same colourset as its own. Places in a CPN are depicted as ellipses (Fig 4. b) with the name of the place written into it and the associated colourset (word) below. A token in a CPN is represented by a circle (Fig 4. b). Its value (the data stored in the token) is shown in a rectangle attached to the circle. A number in the circle denotes the number of tokens with the same value. Fig 4. b for example shows a place called link associated with the colourset word and holding three tokens, two storing the value (ack, 5) one with a value of (req, 13). Tokens associated with the predefined colourset unit do not store any data and thus resemble tokens in an ordinary Petri Net or a DSPN.

Fig. 4. Colourset and variable definitions (a) and graphical representation of a place in a CPN (b)

Transitions in a CPN are represented by rectangles (Fig. 5) and can access the data stored in tokens by mapping tokens to variables. There are two possibilities to access this data:

- **Guard conditions**: The transition is enabled only if a specific condition – called a guard condition – regarding one or more variables is met. Guard conditions are encased in brackets and written above the transition (Fig. 5a).
- **Transfer function**: The transition reads and writes variables according to a specified function that can range from simple addition of values to complex conditional commands. Transfer functions consist of the definition of *input()* variables, *output()* variables and the commands to be carried out (*action()*) and are attached below the transition (Fig. 5b).

The examples depicted in Fig. 5 show a transition that only fires if the variable ctrl has the value req (Fig. 5a) and a transition that generates an output variable dest without taking any input variables (Fig. 5b), the variable dest is filled with the return value of the function defined in the action part which in this case is a uniformly distributed random number between 0 and 15.

Fig. 5. Transitions with guard condition (a) and transfer function (b)

Places and transitions in a CPN are linked by arcs. Arcs in a CPN can be unidirectional like in a DSPN or bidirectional. Unidirectional arcs transfer tokens from a place to a transition or vice versa (Fig. 6 a), bidirectional arcs transfer the same token from a place to a transition and back (Fig. 6 b). Arc inscriptions define the mapping of tokens to variables. An inscription can either be a constant value (Fig. 6 a) or a variable of the colourset that is associated to the place the arc is connected to (Fig. 6 b). In case of complex colousets an inscription can also contain a set of variables. The *word* colourset defined in Fig. 4 a for example consists of two parts, a *control* and an *address* part. A token of the colourset *word* can be either mapped to a single variable of *word* or to a set *(var1, var2)* with *var1* having the colourset *control* and *var2* being of the colourset *address*.

If all places connected to a transition by unidirectional input arcs or by bidirectional arcs hold tokens and its (optional) guard condition is met, the transition is said to be enabled. In case of more than one enabled transition in a CPN the one to fire is chosen randomly. Upon firing a transition deletes the appropriate tokens from input places and generates tokens in its output places. Places linked to the transition by bidirectional arcs are treated as both input and output places.

Fig. 6. Unidirectional arc with mapping to value *3* (a), bidirectional arc with mapping to variable *dest* (b)

For an analysis of clocked systems it is possible to define *timed* coloursets, defined by the keyword *timed* (Fig. 7a) and transition delays marked by the characters @+ (Fig. 7b). If a colourset is defined as *timed*, a timestamp is added to the tokens of this colourset. The timestamp cannot be accessed by guard conditions or transfer functions. When using *timed* coloursets the firing of transitions depends on a global clock counter. Transitions can only fire if the clock value is the same as the largest timestamp of its input tokens. When a *timed* transition fires, the timestamp of its output tokens is the sum of the current clock value and the transition delay, in the example in Fig. 7b this delay is 100 clock cycles.

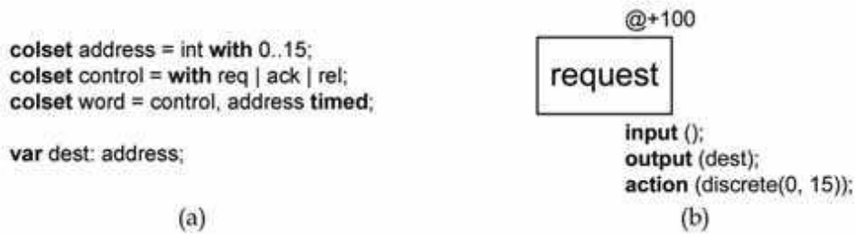


Fig. 7. Timed colourset definition (a) and transition with associated delay (b)

As an introductory example to CPN modelling a basic model of NoC communication is presented in the following paragraph. Clients in the NoC are identified by their addresses (here, integers ranging from 0 to 15). Since the communication in this NoC is based on line switching a route from source to destination has to be established before starting data transmission. The coloursets and variables used in this example are those shown in Fig. 7 a as well as the colourset unit. Messages sent through the NoC are represented by tokens of the colourset word. This colourset contains a part with the colourset control that designates how the message is to be handled and a destination address. Possible values for the control colourset are req (request route), ack (acknowledge route) and rel (release route).

In the beginning, the data source in the modelling example depicted in Fig. 8 is idle – no data is to be sent. The global clock (clock counter) is supposed to be 0. The place idle is marked, thus the transition request is enabled. This transition then fires and generates a token in the place wait – the source is now waiting for establishing of the route. At the same time the transition generates a token (req, dest) @ 100 in the place link, with @ 100 denoting the timestamp. This is a request to the network to make a route available from the source to the client with the address dest. The value of dest is a random number between 0 and 15 generated by the transfer function of the transition request (input (); … discrete(0, 15));) (see Fig. 8). With a token (req, dest) in link the transition routing becomes enabled. It fires as soon as the clock reaches 100 and generates an acknowledgement to notify the source of successful routing. Supposing the routing takes Troute = 30 clock cycles the token generated in link is (ack, empty) @ 130. Transition ack is now enabled and fires at a clock value of 130 generating a token in the place send. This means that the source switches from wait to the send mode (data transmission). Because the colourset associated with send is unit timed rather than unit like for idle and wait the token generated in send receives a time stamp of 130+Tburst, where Tburst describes the duration of a data burst. The transition release therefore cannot fire until the clock value is 130+Tburst. Transmission of a data burst is modelled only by setting the source to send mode for Tburst clock cycles. After sending the data burst (global clock at 130+Tburst) the transition release fires. Firing of this transition

resets the source state to idle and generates a token (rel, empty) in the place link, signalling the network to release the route as it is no longer needed. The rel token enables transition relNet that handles the actual release of the route, which is not modelled explicitly.

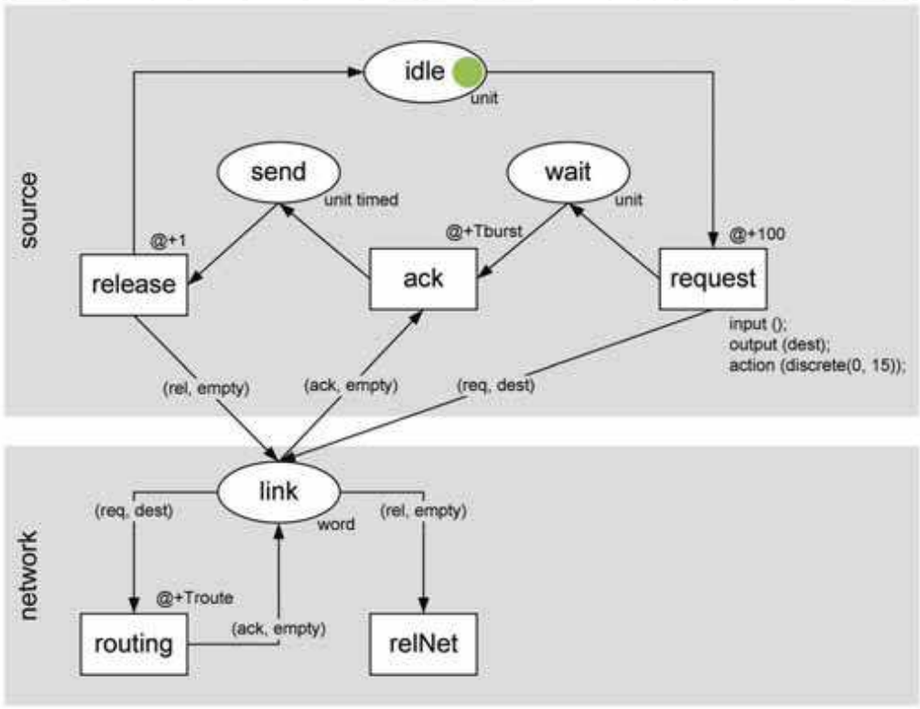


Fig. 8. CPN model of communication between a network and an attached data source

This example shows that the inclusion of data structures into CPN modelling increases the modelling capabilities compared to DSPNs. Both, the inclusion of data structures and the related use of transfer functions allow for greater functionality and smaller models that are easier to handle. With a DSPN model for example it would not be possible to store destination address information in a token or generate random addresses. In a DSPN it would be necessary to store the address in a binary format in a number of places while random generation of an address needs a sophisticated DSPN for modelling this process.

The software tool CPNtools (Ratzer, 2006), which has been used for NoC performance analysis, is a package for modelling and simulation with CPN. It consists of a graphical user interface for composition of CPN models and a simulator. CPN models are described in a format derived from Standard Markup Language (SML) called CPNml. Furthermore, CPNtools allows hierarchical definition of CPNs to facilitate reuse and simplify handling of large models. Parts of a model that are used multiple times can be encapsulated in a submodel. These submodels are included in higher hierarchy levels as substitute transitions with a defined mapping of input and output places of the transition to places in the submodel. In contrast to DSPNexpress CPNtools does not provide a means of analytical or

iterative solution but is centred on simulation. In principle it is possible to generate an ordinary Petri Net with the same functionality as a CPN that can then in turn be solved analytically. Due to the complex data structures (coloursets) and transfer functions included in a CPN the equation system describing such an underlying Petri Net would be very large. Model parameters can be measured by definition of monitors that collect data relating to different parts of the CPN such as occupation of places or the number of times a specific transition fires. The markup language used for model description also allows to use more complex monitors, including for example conditional data collection.

3. Petri net modelling of exemplary communication scenarios

In this section the exemplary application of Petri Nets for modelling communication scenarios is presented. The modelling possibilities span from simple bus based processor communication scenarios to complex NoC examples.

3.1 DSPN based processor communication model

The TMS320C6416 (Texas Instruments, 2007) (see
Fig. 9) is a high performance digital signal processor (DSP) based on a VLIW-architecture. This DSP features a couple of interfaces, an Enhanced DMA-controller (EDMA) handling data transfers and two dedicated coprocessors (Viterbi and Turbo decoder coprocessor). Exemplary communication scenarios on this DSP have been modelled. The C6416 TEB (Test Evaluation Board) platform including the C6416 DSP has been utilized to measure parameters of these modelled communication scenarios described in the following. Thus, modelling results have been proved and verified by comparison with measured values.

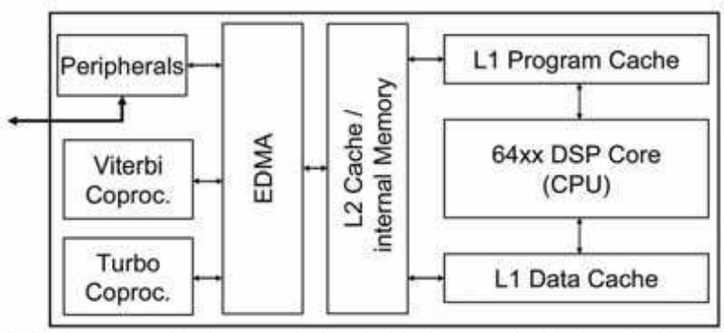


Fig. 9. Basic block diagram of the TMS320C6416 DSP

In Fig. 10 a block diagram of the C6416 and different communication paths of basic communication processes (①,② and ③) are depicted.

In the first scenario two operators compete for one critical resource, the external memory interface (EMIF). Requests for the external memory and with it the memory interface are handled and arbitrated by the enhanced direct memory access controller (EDMA) applying an arbitration scheme which is based on priority queues including four different priorities.

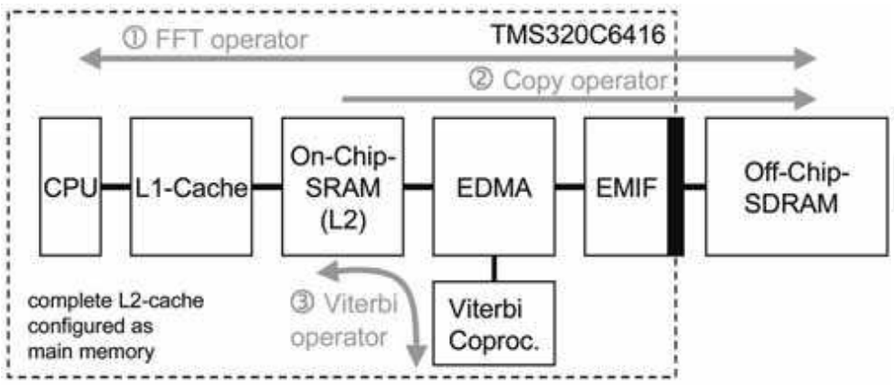


Fig. 10. Communication paths on the C6416 of different analysis scenarios

An FFT (Fast Fourier Transformation) operator runs on the CPU and reads and stores data from the external memory (e.g. for a 64-point FFT, 1107 read and 924 write operations are required which can be determined by analysis of the corresponding C-code). The corresponding communication path ① of this operator is illustrated on top of the simplified schematic of the C6416. The communication path of the copy operator ② is also depicted in Fig. 10. This operator utilizes the so called Quick Direct Memory Access mechanism (QDMA) which is a part of the EDMA. It copies data from the internal to the external memory section. Here, it requests a copy operation every CPU cycle. Since both operators run concurrently, both aim to access the critical external memory interface resource. Requests are queued in the assigned transfer request queue according to their priority. If the CPU and the QDMA both simultaneously request the memory with the same priority, the CPU request will be handled at first. In all modelled communication scenarios the priority of request initiated by the CPU and the QDMA were both assigned to the same priority which means that a competition situation for this waiting queue has been forced. The maximal number of waiting requests of this queue is 16.

The DSPN depicted in Fig. 11 represents the concurring operators and the arbitration of these two operators for the memory resource. It is separable into three subnets (see dashed boxes: Arbitration, FFT on CPU and QDMA-copy operator). The QDMA-copy operator works similar to the DMA-controller device depicted in Fig. 3.

The proprietary transfer request queue is modelled by the place *TransferRequestQueue*. The depth of the queue is modelled by inhibiting arcs with the weight 16 (the queue capacity) originating from this place. This means that these arcs inhibit the firing of transitions they are connected to if the corresponding place (*TransferRequestQueue*) is marked with 16 tokens. These inhibiting arcs are linked to subnets representing components of the system which apply for the transfer request queue. The deterministic transition T6 repetitively removes a token with a delay which corresponds to the duration of an external memory access (see parameterization in the following).

The QDMA copy operator is modelled by a subnet which produces a memory request to the EDMA every CPU cycle. The delay of deterministic transition T5 corresponds to the CPU cycle time. The places belonging to this subnet are *COPY_Start* and *COPY_Submitted*. The token of the place *COPY_Start* is removed after the deterministic delay assigned to

transition T5. The places *COPY_Submitted* and *TransferRequestQueue* are then both marked with a token. If no FFT request initiated by the CPU is pending this process recurs.

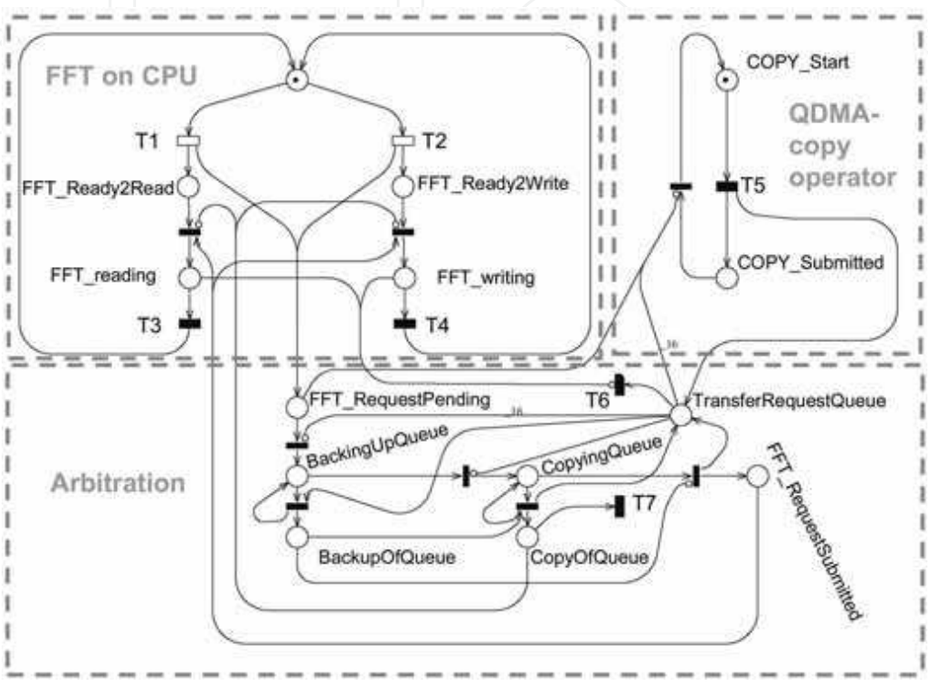


Fig. 11. DSPN of FFT / copy operator resource conflict scenario

The subnet representing the FFT operator executed on the CPU (FFT on CPU) is depicted in the upper left of Fig. 11. If one of the places *FFT_Ready2Read* (connected to stochastic transition T1) or *FFT_Ready2Write* (connected to stochastic transition T2) is marked the place *FFT_RequestPending* is also marked by a token. Hereby, a part of the model is activated which represents the queuing of the CPU requests and the assignment of the associated memory access. Places belonging to this part are: *FFT_RequestPending*, *BackingUpQueue*, *BackupOfQueue*, *CopyingQueue*, *CopyOfQueue* and *FFT_RequestSubmitted*. The place *CopyOfQueue* is a copy of the place *TransferRequestQueue*. That means that these places are marked identically. This copy proceeds by firstly removing every token in *TranferRequestQueue* and transferring it via an immediate transition to the place *BackUpQueue*. This procedure is controlled by the place *BackingUpQueue*. As soon as every token is transferred the place *CopyingQueue* is marked. Now every token in the *BackUpQueue* place is transferred simultaneously to *TransferRequestQueue* as well as to *CopyOfQueue*. Thus, the original marking of *TransferRequestQueue* is restored and also copied in the *CopyOfQueue* place. Now the *FFT_RequestSubmitted* is marked and an additional token is added to the *TransferRequestQueue* representing a further CPU request. The transitions between *FFT_RequestSubmitted* and *FFT_Reading* as well as *FFT_Writing* remove the token from the first mentioned place as soon as the CPU request is granted. The deterministic transition T7

detracts tokens from *CopyOfQueue* in the same way T6 does in context with *TransferRequestQueue*. The external memory access requested by the CPU is granted when the *CopyOfQueue* is not marked by any token. The inhibiting arcs between *CopyOfQueue* and the transitions connected to *FFT_Reading* and *FFT_Writing* ensure that only then the duration of a read and respectively a write access is modelled with the aid of deterministic transitions T3 and T4. During memory access initiated by the CPU no further request to the memory is processed. This is modelled by the inhibiting arcs originating in *FFT_Reading* and *FFT_Writing* (connected to T6). Thus, no further token from the *TransferRequestQueue* is removed.
The required parameters of the deterministic and stochastic transitions T1-T7 of this DSPN model are given in Table 1.
Here, it holds:

T_{FFT} : duration of a single block FFT operation (dependent on FFT length, without parallel copy operation)
$N_{Read/Write}$: number of memory read/write accesses per FFT operation
$T_{Read/Write,ext.mem}$: time required to read/write a word from/to the external memory
$p_i(t)$: probability density function of the delay time of a specific transition

Transition	Transition type	Formula and parameters
T1	stochastic (negative exponential distributed)	$p_1(t) = \lambda_1 \cdot e^{-\lambda_1 t}$ for $t > 0$ with $\lambda_1 = \frac{N_{Read}}{T_{FFT} - N_{Read} \cdot T_{Read,\,ext.mem} - N_{Write} \cdot T_{Write,\,ext.mem}}$
T2	stochastic (negative exponential distributed)	$p_2(t) = \lambda_2 \cdot e^{-\lambda_2 t}$ for $t > 0$ with $\lambda_2 = \frac{N_{Write}}{T_{FFT} - N_{Read} \cdot T_{Read,\,ext.mem} - N_{Write} \cdot T_{Write,\,ext.mem}}$
T3	deterministic	$\Delta t_3 = T_{Read,ext.\,mem} = N_{Read} \cdot 0.188\,\mu s$
T4	deterministic	$\Delta t_4 = T_{Write,ext.\,mem} = N_{Write} \cdot 0.088\ \mu s$
T5	deterministic	$\Delta t_5 = 1/f_{\mathrm{Pr}\,oc} = 1/500\,\mathrm{MHz} = 2\ \mathrm{ns}$
T6	deterministic	$\Delta t_6 = 1/f_{ext.mem} = 1/133\,\mathrm{MHz} = 7.5\ \mathrm{ns}$
T7	deterministic	$\Delta t_7 = 1/f_{ext.mem} = 1/133\,\mathrm{MHz} = 7.5\ \mathrm{ns}$

Table 1. Transition type and transition parameters of the DSPN model of Fig. 11

The required input parameters for the DSPN model like the duration of a single block FFT without running the concurrent copy operator (T_{FFT}) have been determined by

measurements performed on a DSP board. In order to verify the assumptions e.g. for $T_{Read,ext.mem}$ and $T_{Write,ext.mem}$, several experiments with a variation of external factors have been performed. For example, the influence of the refresh frequency has been studied. By modification of the value within the so-called EMIF-SDTIM register the refresh frequency of the external SDRAM could be set. Through different measurements it could be verified that the resulting influence on the read and write times is below 0.3 % and therefore negligible. For the final measurements a refresh frequency of 86.6 kHz (what is equal to a refresh period of 1536 memory cycles and therefore an EMIF-SDTIM register value of 1536) has been applied.

The influence of the parameter N_{Read} will be explained exemplarily in the following. The probability density function $p_1(t)$ which is a function of N_{Read} characterizes the probability for each possible delay of the stochastic transition T1. N_{Read} directly influences the expected delay respectively the firing probability of T1. Here, high values for N_{Read} correspond to a low firing probability respectively a large expected delay and vice versa.

The modelling results of the DSPN for the duration of the FFT are depicted in Fig. 12. Here, the calculation time of the FFT operator determined by simulation with the DSPN model has been plotted against different FFT lengths. In order to attain a quantitative evaluation of the computed FFT's duration, reference measurements have been made again on a DSP board. As can be seen from Fig. 12 the model yields a good estimation of the duration for the FFT operator. The maximum error is less than 10 % (occurring in case of an FFT length of 1024 points).

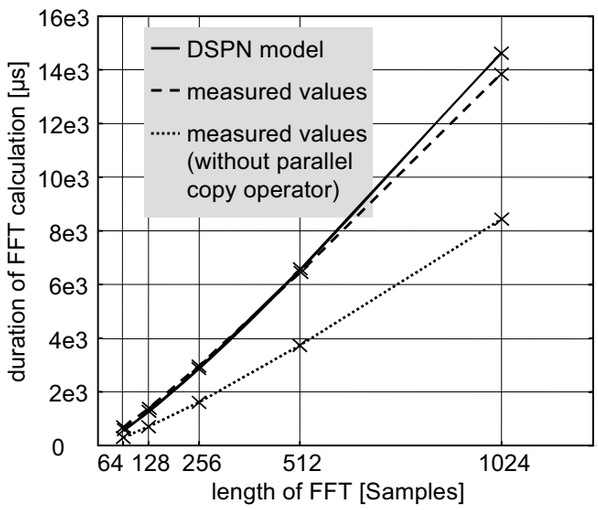


Fig. 12. Comparison of measured values with DSPN (FFT vs. copy operator)

Another example based on this DSP was analyzed in order to consolidate the suitability of using DSPNs for modelling in terms of on-chip communication: Now, the Viterbi Coprocessor (VCP) and the copy operator compete for the critical external memory interface resource. The VCP also communicates with the internal memory via the EDMA (commu-

nication path ③ in Fig. 10). Arbitration is handled by a queuing mechanism configured here in that way that only a single queue is utilized. This is accomplished by assigning the same priority to all EDMA requestors i.e. memory access is granted to the VCP and the copy operator according to a first-come-first-serve policy.

For this experiment the VCP has been configured in the following way. The constraint length of the Viterbi decoder is 5, the number of states is 16 and the rate is 1/2. In the VCP configuration inspected here, the VCP communicates with the memory by getting 16 data packages of 32x32 bit in order to perform the decoding. Both, EDMA and VCP are clocked with a quarter of the CPU clock frequency (fCPU = 500 MHz). The results are transferred back to the memory with a package size of 32x32 bit. Performing two parallel operations (Viterbi decoding and copy operation), the two operators have to wait for their corresponding memory transfers. The EDMA mechanism of the C6416 always completes one memory block transfer before starting a new one. Hence, there is a dependency of the Viterbi decoding duration on the EDMA frame length. This situation has been modelled and the results have been compared to the measured values as depicted in Fig. 13.

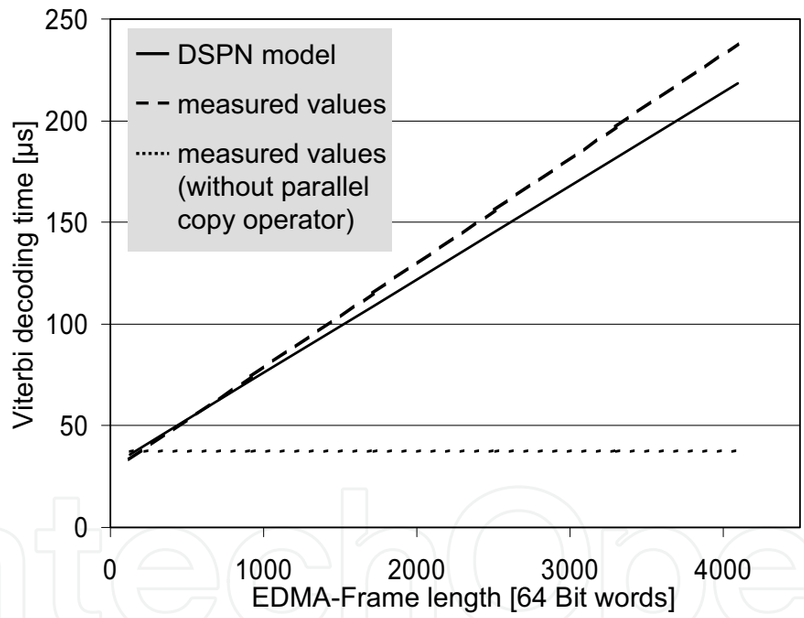


Fig. 13. Comparison of measured values with DSPN (Viterbi vs. copy operator)

Performing only the Viterbi decoding, there is of course no dependency on the EDMA frame length. If a copy operation is carried out, the Viterbi decoding time significantly increases. In detail not the decoding process itself is concerned but the duration of data package transfers between VCP and internal memory. Again the maximum error is less than 10 %.

3.2 DSPN based switch fabric communication model

The second DSPN modelling example deals with communication via a switch fabric based structure. The modelled scenario is a resource sharing conflict. This scenario has been evaluated on an APEX based FPGA development board (Altera, 2007).

A multi processor network has been implemented on this development board by instantiating Nios soft core processors on the corresponding FPGA. The synthesizable Nios embedded processor is a general-purpose load/store RISC CPU that can be combined with a number of peripherals, custom instructions, and hardware acceleration units to create custom system-on-a-programmable-chip solutions. The processor can be configured to provide either 16 or 32 bit wide registers and data paths to match given application requirements. Both data width versions use 16 bit wide instruction words. Version 3.2 of the Nios core typically features about 1100 logic elements (LEs) in 16 bit mode and up to 1700 LEs in 32 bit mode including hardware accelerators like hardware multipliers.

More detailed descriptions can be found in (Altera, 2001). A processor network consisting of a general communication structure that interfaces various peripherals and devices to various Nios cores can be constructed. The Avalon (Avalon, 2007) communication structure is used to connect devices to the Nios cores. Avalon is a dynamic sizing communication structure based on a switch fabric that allows devices with different data widths to be connected with a minimal amount of interfacing logic. The corresponding interfaces of the Avalon communication structure are based on a proprietary specification provided by Altera (Avalon, 2007). In order to realize a processor network on this platform the so-called SOPC (system on a programmable chip) Builder (SOPC, 2007) has been applied. SOPC is a tool for composing heterogeneous architectures including the communication structure out of library components such as CPUs, memory interfaces, peripherals and user-defined blocks of logic. The SOPC Builder generates a single system module that instantiates a list of user-specified components and interfaces incl. an automatically generated interconnect logic. It allows to modify the design components, to add custom instructions and peripherals to the Nios embedded processor and to configure the connection network.

The analyzed system is composed of two Nios soft cores which compete for access to an external shared memory (SRAM) interface. Each core is also connected to a private memory region containing the program code and to a serial interface which is used to ensure communication with the host PC. The proprietary communication structure used to interconnect all components of a Nios based system is called Avalon which is based on a flexible crossbar architecture. The block diagram of this resource sharing experiment is depicted in Fig. 14. Whenever multiple masters can access a slave resource, SOPC Builder automatically inserts the required arbitration logic. In each cycle when contention for a particular slave occurs, access is granted to one of the competing masters according to a Round Robin arbitration scheme. For each slave, a share is assigned to all competing masters. This share represents the fraction of contention cycles in which access is granted to this corresponding master. Masters incur no arbitration delay for uncontested or acquired cycles. Any masters that were denied access to the slave automatically retry during the next cycle, possibly leading to subsequent contention cycles.

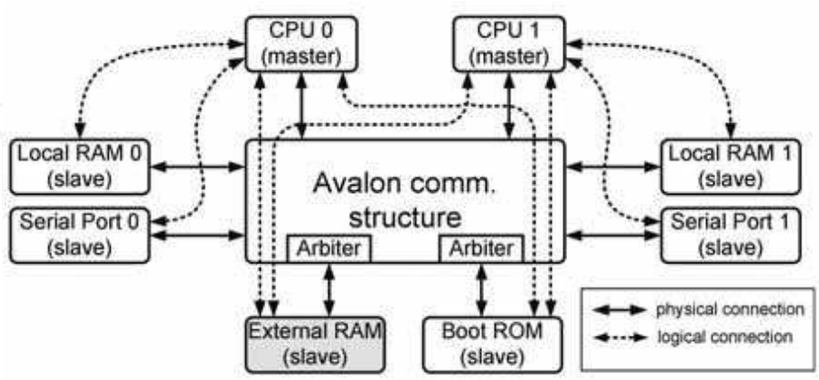


Fig. 14. Block diagram of the resource sharing experiment using the Avalon communication structure

In the modelled scenario the common slave resource for which contention occurs is a shared external memory unit (shaded in gray in Fig. 14) containing data to be processed by the CPUs. Within the scope of this fundamental resource sharing scenario several experiments with different parameter setups have been performed to prove the validity of the DSPN modelling approach. Adjustable parameters include:

- the priority shares assigned to each processor,
- the ratio of write and read accesses,
- the mean delay between memory accesses.

These parameters have been used to model typical communication requirements of basic operators like digital filters or block read and write operations running on these processor cores. In addition, an experiment simulating a more generic, stochastic load pattern, with exponentially distributed times between two attempts of a processor to access the memory has been performed. Here, each memory access is randomly chosen to be either a read or a write operation according to user-defined probabilities. The distinction between load and store operations is important here because the memory interface can only sustain one write access every two cycles. Whereas no such limitation exists for read accesses. The various load profiles were implemented in C, compiled on the host PC and the resulting object code has been transferred to the Nios cores via the serial interface for execution. In the case of the generic load scenario, the random values for the stochastic load patterns were generated in a MATLAB routine. The determined parameters have been used to generate C code sequences corresponding to this load profile. The time between two attempts of a processor to access the memory has been realized by inserting explicit NOPs (No Operation instruction) into the code via inline assembly instructions. Performance measurements for all scenarios have been achieved by using a custom cycle-counter instruction added to the instruction set of the Nios cores. The insertion of NOPs does not lead to an accuracy loss related to pipeline stalls, cache effects or other unintended effects. The discussed example is constructed in such a way that these effects do not occur. In a first step, a basic DSPN model has been implemented (see Fig. 15) in less than two hours. Implementation times of the DSPN models are related to the effort a trained student (non-expert) has to spend to realize the corresponding model. The training time for a student to become acquainted with DSPN modelling lasts a couple of days. Distinction between read and write accesses was explicitly

neglected to achieve a minimum modelling complexity. The DSPN consists of four sub-structures:

- two parts represent the load generated by the Nios cores (CPU #1 and #2)
- a basic cycle process subnet providing a clock signal (Clock-Generation)
- the more complex arbitration subnet

Altogether, this basic model includes 19 places and 20 transitions. The immediate transitions T1, T2 and T3 and the associated places P1, P2 and P3 (see Fig. 15) are an essential part of the Round Robin arbitration mechanism implemented in this DSPN. The marked place P2 denotes that the memory is ready and memory access is possible. P1 and P3 belong to the CPU load processes and indicate that the corresponding CPU (#1, #2) tries to access the memory. If P1 and P2 or P3 and P2 are tagged the transition T1 or accordingly transition T3 will fire and remove the tokens from the connected places (P1, P2 or P2, P3). CPU #1 or CPU #2 has been assigned the memory access in this cycle. A collision occurs if P1, P2 and P3 are tagged with a token. Both CPUs try to access the memory in the same cycle (P1 and P3 marked). Furthermore, the memory is ready to be accessed (P2 marked). A higher priority has been assigned to transition T2 during the design process. This means that if the conditions for all places are equal the transition with the highest priority will fire first. Therefore, T2 will fire and remove the tokens from the places. Thus, the transitions T1, T2 and T3 and the places P1, P2 and P3 handle the occurrence of a collision.

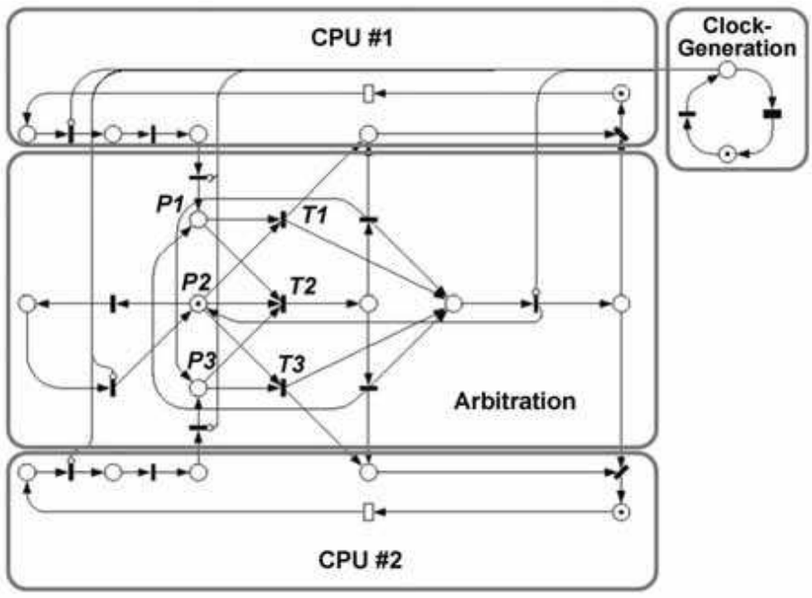


Fig. 15. Basic DSPN for Avalon-Nios example

The modelling results discussed in the following have been acquired by application of the iterative evaluation method. Though the modelling results applying this basic DSPN model are quite accurate (relative error less than 10 % compared to the physically measured values, see Fig. 18), it is possible to increase the accuracy even more by extending the modelling

effort for the arbitration subnet. For example it is possible to design a DSPN model of the arbitration subnet which properly reflects the differences between read and write cycles. Thus, the arbitration of write and read accesses has been modelled in different processes resulting in different DSPN subnets. This results in a second and enhanced DSPN model depicted in Fig. 16. The implementation of this enhanced model has taken about three times the effort in terms of implementation time (approximately five hours) than the basic model described before.

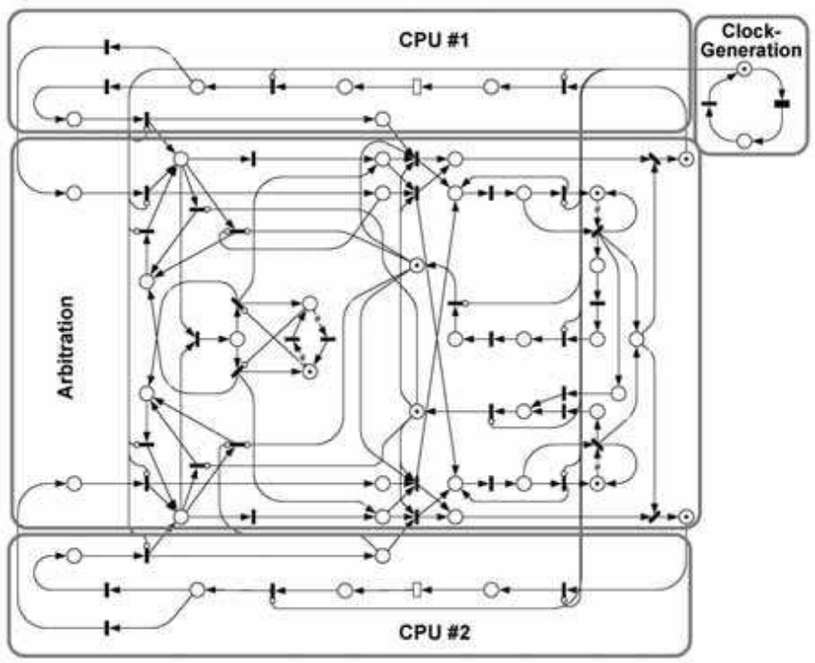


Fig. 16. Enhanced DSPN for Avalon-Nios example

The DSPN model now consists of 48 transitions and 45 places. Compared to the basic model the maximum error has been further reduced (see Fig. 17 and Fig. 18). The enhanced model also properly captures border cases caused e. g. by block read and write operations.
The throughput measured for a code sequence containing 200 memory access instructions has been compared to the results of the basic and enhanced DSPN model. Fig. 18 shows the relative error for the throughput (results of the DSPN model compared to measured results of an FPGA based testbed) which is achieved by varying the mean number of computation cycles between two attempts of a processor to access the memory. On average the relative error of calculated memory throughput is reduced by 4-6 % with the transition from the basic to the enhanced model. Using the enhanced DSPN model the maximum estimation error is below 6 %. As mentioned before, the evaluation of DSPNs can be performed by different methods (see Fig. 19). The effort in terms of computation time has been compared for a couple of experiments. Generally, the time consumed when applying the simulation

method is about two orders of magnitude longer than the time consumed by the analysis methods. The simulation parameters have been chosen in such a way that the simulation results match the results of the analytic approaches. DSPNexpress provides an iterative method (Picard's iteration method) and a direct solution method (general minimal residual method). Fig. 19 illustrates a comparison of the required computational time for the analysis and the simulation of the introduced basic and enhanced DSPN models. For the example of the enhanced model the computation time of the DSPN analysis method only amounts to 0.3 sec. and the DSPN simulation time (10^7 memory accesses) amounts to 20 sec. on a Linux based PC (2.4 GHz, 1 GByte of RAM). The difference between the iterative and direct analysis method is hardly noticeable.

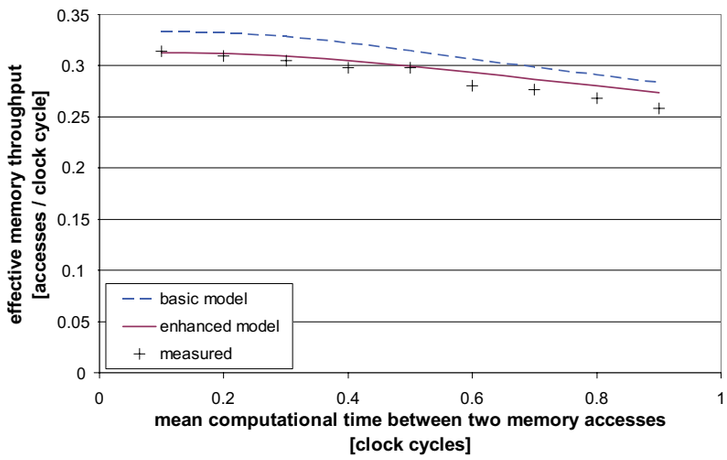


Fig. 17. Effective memory throughput comparison

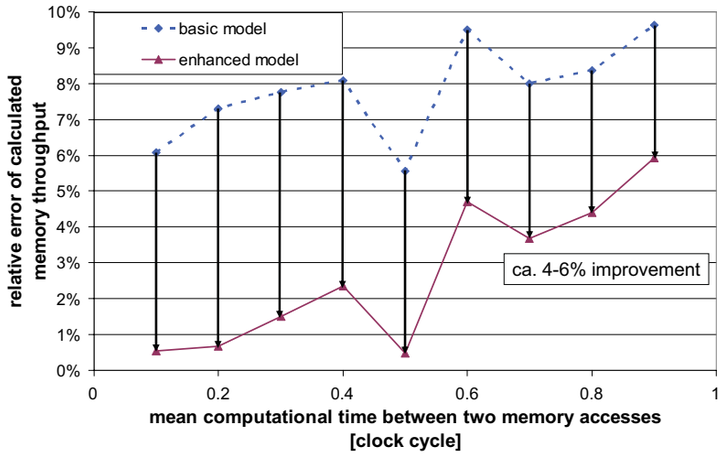


Fig. 18. Relative error of memory throughput for basic and enhanced DSPN model

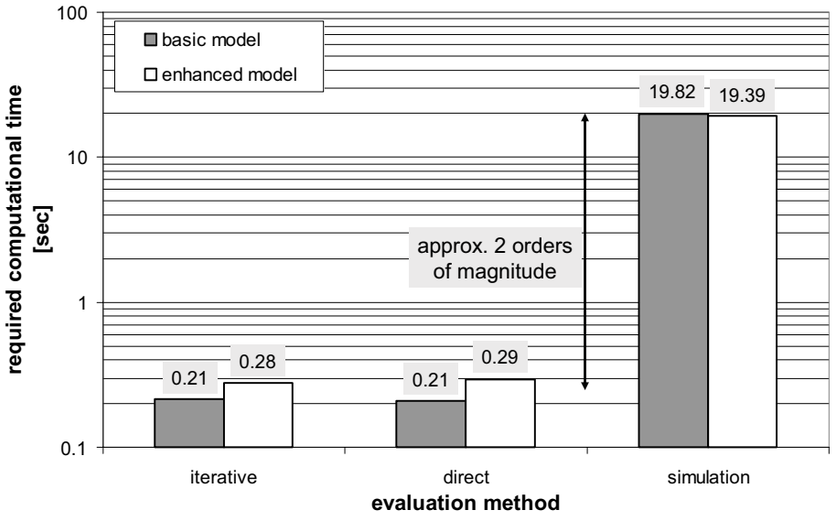


Fig. 19. Required computational effort for the different evaluation methods

3.3 CPN based NoC model

The NoC model presented in this section consists of 25 network nodes arranged in a 5x5 square mesh as depicted in Fig 20. Each network node consists of a routing switch and a client. Clients are any data sources connected to the NoC, for example embedded processors. They are identified by a unique address containing their x and y coordinates. The routing switches and the links connecting them form the actual communication infrastructure facilitating communication between clients. The switching scheme chosen in the model is line switching. Hence, communication between any two clients can be divided into three stages:

- establishing of a route from originating client (source) to the receiving one (destination),
- data transmission and
- releasing the route.

The communication protocol is described in more detail below. The implemented routing algorithm is xy-routing. This is a minimal-path routing algorithm trying first to route horizontally until the destination column is reached (matching of x-coordinates) and then completes the route vertically (matching of y-coordinates). The arbitration scheme used is first-come-first-serve.

For analyzing different traffic experiments the clients are configurable by

- a list of possible destinations for communication attempts,
- the duration of data bursts (measured in clock cycles) to be transmitted (*Lburst*) and
- the average delay between the end of a transmission and the request for routing the next one (*Ldelay*).

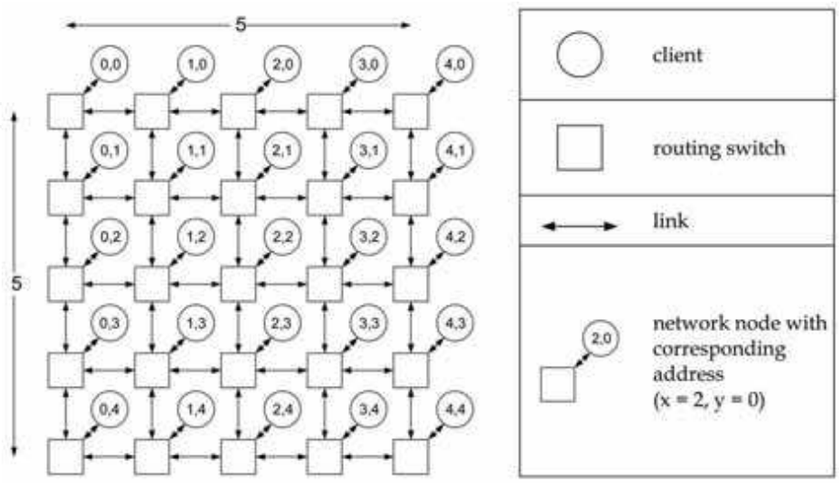


Fig. 20. NoC setup for experiments

The routing switches are not configurable. Performance is measured by latency and source load. Latency in this case corresponds with the time needed for establishing a route and is chosen as a performance measure because it is critical for applications that need fast data transmission, for example real time applications. For applications generating a lot of data, throughput is important. Therefore, the source load is selected as another performance characteristic. The source load is defined as the relative time a data source is transmitting; the requested source load is defined as

$$source\_load_{req} = \frac{Lburst}{Lburst + Ldelay} \quad , \tag{1}$$

while the achieved source load takes into account the latency (*Lat*) caused by establishing a route:

$$source\_load_{ach} = \frac{Lburst}{Lburst + Ldelay + Lat} \, . \tag{2}$$

Since the requested load does not include latency which always occurs in a network, the achieved source load is always smaller than the requested one.

The essential parts of the model are briefly explained in the following. The NoC model consists of two main submodels, the routing switch model and the client model. Each network node consists of a routing switch and an attached client. Messages sent through the net are represented by tokens of the colourset *word* (Fig. 21). Each message consists of a header (colourset *control*) defining how it is to be processed and the *content* that is used according to the header specification. Possible headers are *req* (request route), *rel* (release route), *relb* (acknowledge release route), *kill* (routing failure) and *ack* (acknowledge route), *content* can be either a destination address (*de*) or *emp*ty.

```
colset coordinate = int with 0..15;                  // x and y coordinates of nodes
colset address = coordinate, coordinate;             // node addresses (x, y)
colset control = with req | rel | relb | kill | ack; // control sequence to identify message type
colset content = de:address | empty;                 // contains either an address type variable de or is empty
colset word = control, content;                      // messages to be sent through the network
```

Fig. 21. Essential coloursets for the CPN based NoC model

Since in this example line switching is used, communication between two clients is made up of several stages as stated above. When a data source tries to send data, a *req* message is generated that is then routed through the network according to the routing algorithm. The content of a *req* message contains the destination address of the route. In each network node the *req* message is processed by the routing switch which reserves the appropriate connection of two of its ports for the requested route. This is done by comparing the local and destination addresses and then adding a member to the list of current routes stored in a token of the colourset *routelist*. Upon arrival of the *req* signal at the destination the client generates an *ack* message to travel back along the route. Reception of an *ack* at the source triggers data transmission. Data is not represented by any tokens because network performance does not depend on the actual data sent across the NoC but on the time the route is occupied. After completing data transmission the source client sends a *rel* signal which is returned by the destination as *relb*. Processing of a *relb* message initiates release of the partial routes stored in the routing switches. When routing fails because an output port in any node cannot be reserved since it is already occupied by another route, a *kill* signal is generated and sent backwards along the route. If employing a static routing algorithm this signal is handled like *relb*. If an adaptive algorithm is used processing of a *kill* message can lead to the attempt to select another route. When the source client receives a *kill* signal it will issue a new routing request (*req*) after a configurable time.

The client model is comprised of two submodels, *source* and *sink*, of which only the *source* model will be discussed here in detail as the functionality of the *sink* model is elementary. As the data content is not important for network performance this basic model is sufficient to model a wide range of possible clients that can be attached to a NoC.

The *source* submodel shown in Fig 22. includes transitions for handling incoming and outgoing messages. The place *out* is the interface of the data source to the network, similar to the place *link* in the introductory example (

Fig. 8). The current state of the source is stored in the *status* place. The source sequentially passes through the states *idle*, *wait* and *send*. These states are defined as:

- *idle*: There is currently no data to be sent.
- *wait*: A route was requested, the source is waiting for it to be established.
- *send*: A route was established, the source is transmitting data.

Switching from *idle* to *wait* occurs when the transition *request* fires. This transition generates tokens in the places *wait_for_route* and *out*. The token in *wait_for_route* is later used to measure latency, the token *(req, addr)* in the place *out* signals the network, that a route to the client with address *addr* is requested. The timestamp of the token generated in out is the current global clock value increased by a random number between one and *Lpause* due to the processing delay associated with the *request* transition. The network then replies by either signalling successful routing (*ack*) or an aborted routing attempt (*kill*) by generating a token in the place *out*. If a *kill* token is generated the transition *killreq* becomes enabled. The timestamp is increased by *Tkill* to ensure that there is a delay before the new attempt. If an *ack* token is in the place *out*, the transition *ackdata* fires. The source state is thereby set to *send*.

Firing of *ackdata* removes the token from *wait_for_route* – the route was successfully established – and generates a token into the place *sending*. This token receives a time stamp according to the configured burst length (*Tburst*). The source stays in the *send* state for *Lburst* clock cycles before the transition *release* fires and sets the status back to *idle*. The token *(rel, emp)* generated in the place *out* by this transition signals the network to release the route originating from this source. Successful release of the route is then acknowledged by generation of a *relb* token in the place out. This token enables the transition *release_back* by which it is then removed.

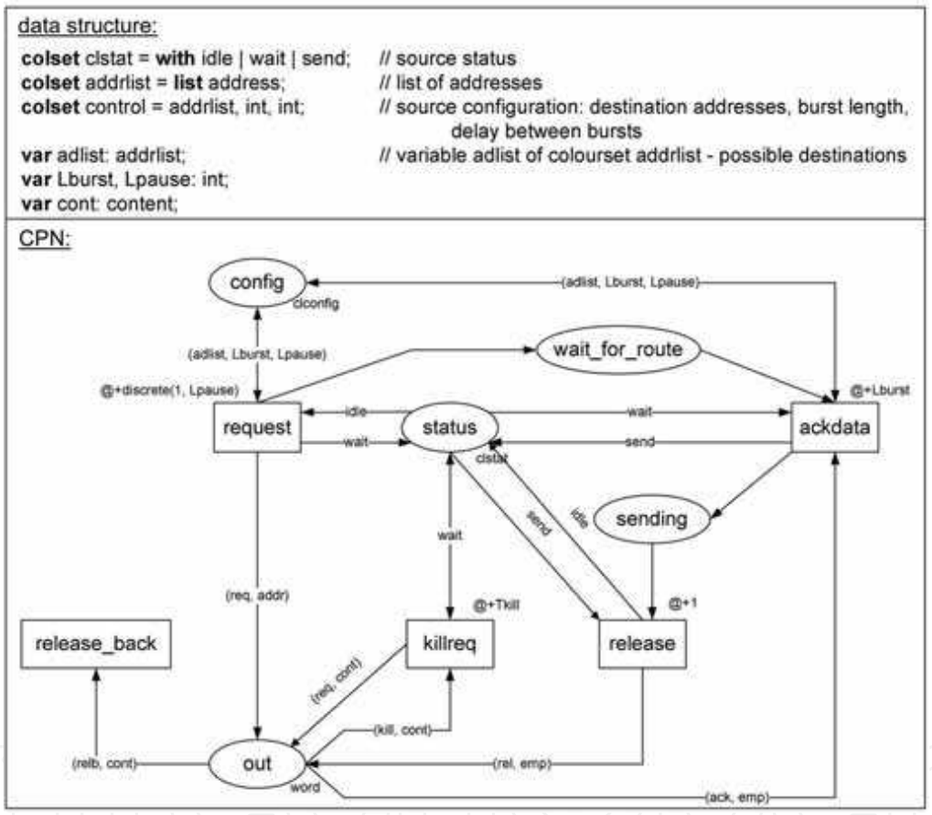


Fig. 22. CPN submodel of a data source and required data structure

The place config is used to configure the source. Variables that can be configured are:

- *adlist*: A list of destination addresses that the source can request routes to. If an address is contained in this list multiple times the probability that a route to the corresponding client is requested increases accordingly.
- *Lburst*: The length of a data burst, measured in clock cycles.
- *Lpause*: The maximum delay between the end of a transmission and the subsequent routing request.

Performance measures obtained in this submodel are the number of routing requests sent, source load and latency. The number of routing requests sent is measured by counting the times the transition *request* fires. Source load is measured by the average occupation of the place *sending*. Latency in this case corresponds to the time used for route establishment. It is measured by computing the total time spent for route establishment (product of the inspected time period and the average occupation of the place *wait_for_route*). This value is then divided by the number of routing requests sent.
The *sink* submodel is comprised of transitions to respond to incoming *req* and *rel* messages with the appropriate signals of its own, these being *ack* and *relb*, without gaining any performance measures.
The routing switch model (Fig. 23) consists of four submodels, *req*, *forward*, *ack* and *kill* as well as the places *addr*, *routes* and the interface places *inWest* to *inClient* and *outWest* to *outClient*. The *addr* place contains the address of the node in the network while the current switching table containing the connections of input and output ports is stored in *routes*. The places *inWest* to *inSouth* and *outWest* to *outSouth* are interfaces to neighbouring routing switches, the places *inClient* and *outClient* are interfaces to the attached client (*inClient* is mapped to the place *out* of the *source* model shown in Fig. 22). The *req* submodel itself contains two further submodels, *router* and *arbiter*, and is used to process routing requests. The *forward* and *ack* submodels transmit tokens according to the switching table the former processing *rel* the latter *ack* messages. The *kill* submodel resembles the *ack* model but also includes deletion of routes from the switching table for handling *relb* and *kill* signals. The *router* and *arbiter* submodels contained in the *req* model are used to separate the actual routing from the arbitration.

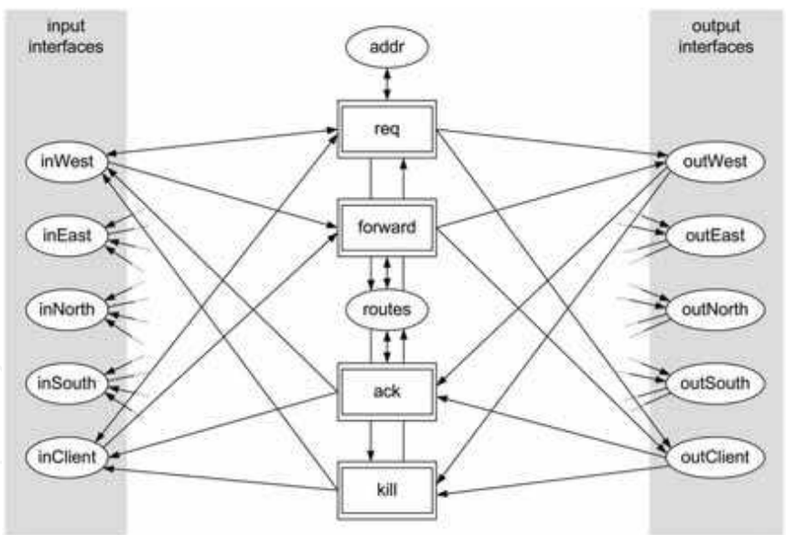


Fig. 23. CPN model of a routing switch for mesh networks

The places *inWest* and *interest* in the router model depicted in Fig. 24 are interfaces, *inW* is mapped to the equivalent place in the routing switch model while *interW* is the connection to the *arbiter* model. The section of the router shown in Fig. 24 is the part responsible for the

western port of the routing switch, the sections for the other ports resemble this section and only differ in the names of places and transitions. The place *addr* contains the address of the network node which the routing switch belongs to. In the *router* the destination address contained in an incoming *req* token (*x1*, *y1*) is compared to the address of the current network node (*x0*, *y0*) before generating a request to the *arbiter* according to the routing algorithm. In case of the router shown in Fig. 24 this is a static xy-routing scheme. The request generated by the *router* is used by the *arbiter* to make the appropriate entry in the switching table (*routes*), resource conflicts are resolved in a first come first serve manner. If an output port is occupied, a kill signal is generated by the *arbiter*. The single performance characteristic gained in the routing switch is its load, which is monitored by occupation of the place *routes*.

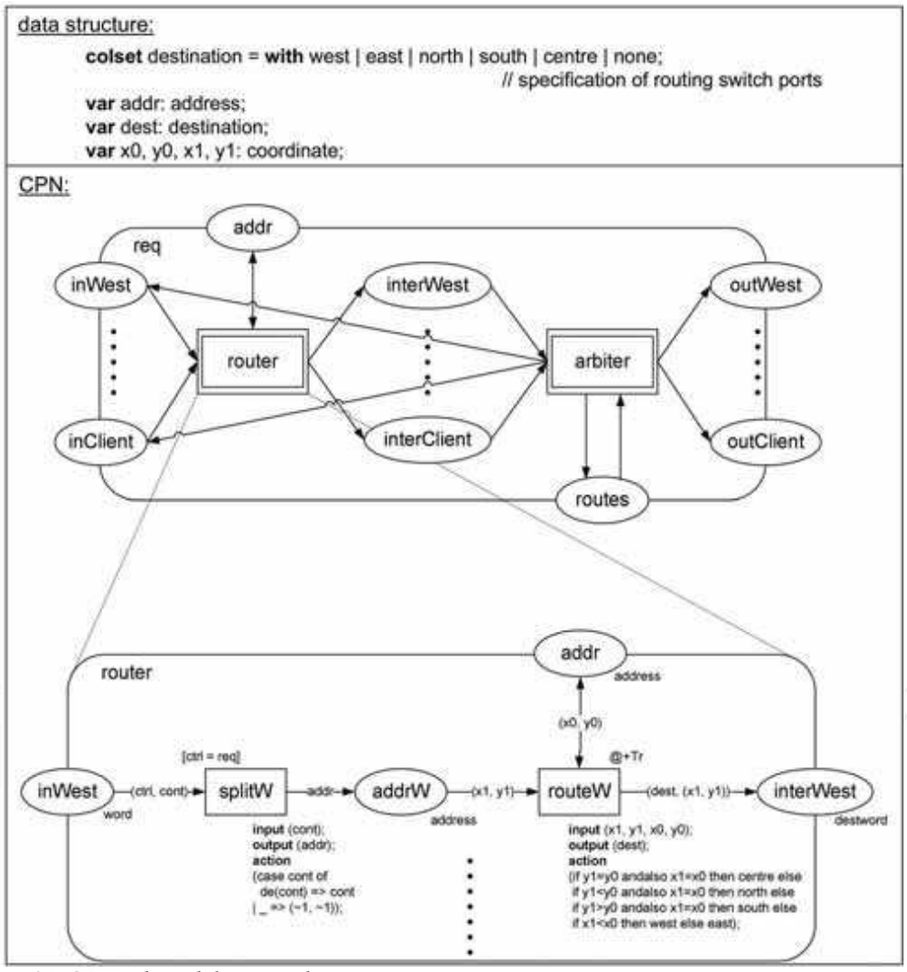


Fig. 24. CPN submodels req and router

In the context of network performance analysis three different experiments have been conducted with the model. The results have afterwards been compared to those obtained with an FPGA based NoC emulator (Neuenhahn, 2006). The experiments are defined as follows:

- **Experiment 1**: Each client is configured to send data to all other clients with equal probability. Requested source load is the same at all clients and set to values from 10 % to 80 % with steps of 10 %.
- **Experiment 2**: Same as experiment 1, but static xy-routing is exchanged for an adaptive variant. This variant tries to route vertically, if a resource conflict prohibits horizontal routing. Source load is set to the same values as in Experiment 1.
- **Experiment 3**: All clients are configured to send data to the nodes (1, 0), (3, 0) and (2, 2) with a probability of 14 %. Here, a testcase is modelled, where some I/O interfaces are accessed more often than other clients. Probability to send to another node is 3 %. The requested source load is 50 % at all clients.

The size of the complete model is 725 places and 1050 transitions for experiments 1 and 3 and 800 places and 1075 transitions in case of experiment 2 because the adaptive router is more complex than the static one. Due to the complexity contained in coloursets and transfer functions these numbers are not easily comparable to those of DSPN models.
Traffic is generated in form of data burst with a duration of *Lburst* = *100* clock cycles, source load is set by adjusting the delay between transmissions (*Lpause*).
Source load in the first two experiments is set to values from 10 % to 80 % with steps of 10 %. All simulations are stopped and repeated after a total of 10,000 transmissions; all results presented below are averaged over 50 repetitions. Model components are the same for all experiments with exception of the adaptive router for the second one.
All simulations were conducted on a computer with a dual Intel Pentium processor running at 3.0 GHz, 1 GB RAM and Microsoft Windows XP as operating system.
Fig. 25 shows a comparison between the first two experimental setups (experiment 1, experiment 2) with static and adaptive versions of the xy-routing algorithm. It is obvious that the achieved source load is close to the requested one for small loads as there are only few resource conflicts. With higher load and thus shorter pauses between individual transmissions the number of conflicts increases. This leads to a decline in network performance. The use of adaptive xy-routing slightly increases the achieved load because

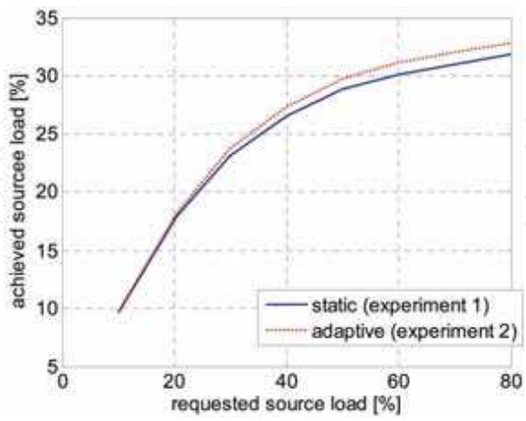


Fig. 25. Average achieved load using static and adaptive xy-routing (5x5 mesh)

some resource conflicts can be resolved without releasing a partial route and reattempting from its source. An adaptive variant that would also include rerouting attempts if a network node receives a *kill* signal would further increase performance at the cost of more complex routing switches.

By analyzing the load of the routing switches it is furthermore possible to locate hotspots that are generated by certain traffic patterns. An example of such a pattern is the one used in experiment 3. The results obtained from simulating experiment 1 and experiment 3 are shown in Fig. 26. In experiment 1 the load distribution among the routing switch shows a flat profile with its maximum in the middle of the NoC (address 2, 2). This hotspot is caused by the fact, that the majority of possible routes contains the central node because the NoC is symmetric to its centre. As all possible pairs of source and destination of routes are equally probable the load of the routing switches is determined only by the number of possible routes including the corresponding node. The modification of the traffic pattern in experiment 3 results in the forming of three distinct hotspots easily identified as peaks in the shown load diagram. Besides high switch loads at those network nodes that are accessed with a higher probability (addresses 2, 2; 1, 0; 3, 0) hotspots also form at the addresses (1, 1) and (3, 1) because most routes ending in (1, 0) and (3, 0) run through these nodes.

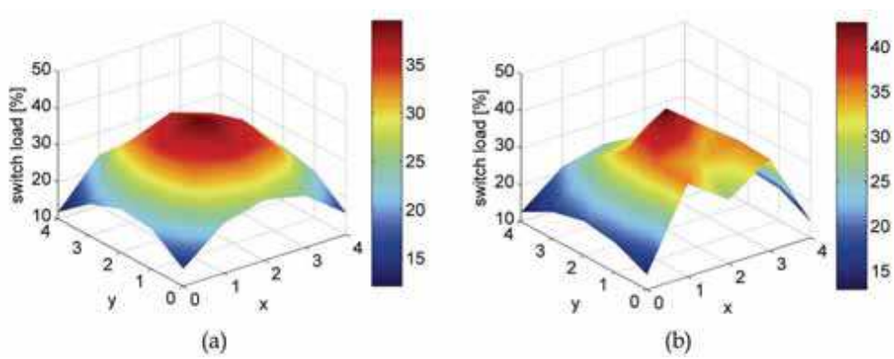


Fig. 26. Switch load for uniform (a, experiment 1) and irregular traffic distribution leading to hotspots (b, experiment 3)

A comparison of these results with an FPGA based emulator (Neuenhahn, 2006) shows that the error of the results obtained with CPN modelling is approximately 2 % for both performance measures. As an example, Fig. 27 shows the relative deviation of the CPN modelling results from those of the emulation for experiment 1. The results shown in Fig. 27 are obtained with a requested load of 20 %. For any single load setting the error is below 5 % within less than five minutes of simulation. This corresponds to approximately eight simulation runs that are needed to gain acceptably accurate result. Complete syntax checking takes approximately 15 minutes and is only needed before the first simulation run. Since syntax checking in CPNtools is incremental this time is reduced to one minute if only the load setting is changed between simulations.

Since the intended use for CPN modelling, like it is presented here, is analysis of NoC performance in an early stage of the design flow, the modelling effort is important to evaluate the usefulness of this method. This effort can be divided into two separate parts, the initial modelling of the NoC components and the combination of these component models to form a

complete NoC model. An overview of the time needs for CPN based modelling of a NoC is given in Table 2. Like the times needed for DSPN modelling these are related to the effort an experienced student has to spend. The time needed for modelling of the components varies from five minutes needed for the data *sink* model which only contains a single place and two transitions to six hours for more complex components like the *source* model or the *router* model.

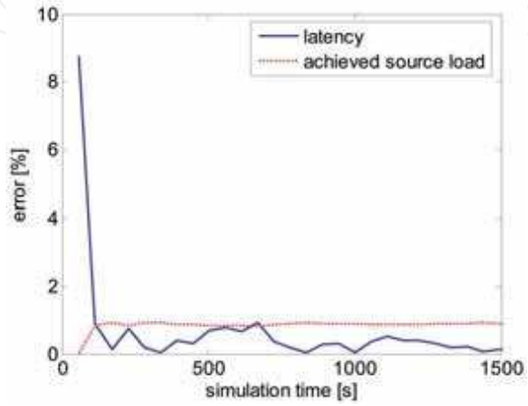


Fig. 27. Error of load and latency compared to emulation (requested load 20 %, 5x5 mesh)

Combining these components to the NoC model used in the experiments described in this section is done in approximately one hour. Exchanging single components, for example the *router* model when switching from experiment 1 to experiment 2, takes one minute but initiates a new syntax check that lasts for five minutes. Altering parameters such as the requested load setting is a matter of seconds but also requires a new partial syntax check of approximately two minutes.

modelling step	**effort**
modelling components	5 to 300 minutes
assembly of NoC model	60 minutes
complete syntax check	15 minutes
partial syntax check	< 5 minutes
simulation	5 minutes

Table 2. Time needs for CPN based modelling of a NoC

The results obtained and the precision achieved show that CPN based modelling of NoCs is an adequate approach for use in design space exploration of communication architectures. After initial modelling of NoC components only little time is needed to construct and modify a NoC model. The error of the results obtained by simulation of this model is small after a reasonably short simulation time. Furthermore, due to the modelling possibilities of CPNs complex components can be modelled with only a few places and transitions thus enabling the user to handle models of large NoCs with relative ease. These properties make CPN based modelling an attractive tool for pruning the design space by early elimination of NoC variants that do not provide the required performance with acceptable costs.

4. Conclusion and outlook

It is a key task of modern System-on-Chip (SoC) and Network-on-Chip (NoC) design to efficiently explore this design space regarding aspects like performance, flexibility and power consumption presumably in an early stage of the design flow in order to reduce design time and design costs.

In this chapter several examples for modelling of on-chip communication using Petri Net based modelling techniques have been presented. These examples include modelling of internal processor communication and modelling of inter-processor communication using a crossbar switch fabric. For these examples deterministic and stochastic Petri Nets have been applied as modelling technique. More complex NoC communication has been modelled applying Coloured Petri Nets. The results obtained with all of these models were compared to those calculated on an FPGA based emulator. In all presented experiments the performance measures derived using these models showed a good precision compared to the results acquired using the FPGA based emulator. Furthermore, the Petri Net based results could be derived in attractively short modelling times with only moderate effort.

Therefore, Petri Net based modelling of on-chip communication appears to be a very attractive approach to explore the design space of communication architectures in an early stage of the design process. DSPN based and CPN based modelling both provide specific advantages. DSPN models are suited for systems with moderate complexity such as communication systems with a small number of clients or bus based communication. The ease of modelling combined with the possibility of an analytical solution of the equations underlying the DSPN model provides a way to quickly obtain results. For more complex systems including a lot of data and complex functionalities, for example the addressing scheme and the routing algorithm in a NoC, CPN models are more adequate. DSPN based modelling of such systems is not as efficient since DSPNs do not provide a means of modelling data structures. As CPNs include data structures and allow to model complex behaviour in form of colourseets and transfer functions, CPN based modelling is well suited to analyze complex on-chip communication systems.

Current topics in the field of NoC communication modelling to be addressed with Petri Net based methods are locating hotspots, analyzing quality-of-service aspects (data integrity, guaranteed service, etc.) and complex adaptive routing algorithms (incl. the checking of absence of deadlocks).

5. References

Ajmone Marslan, M.; Chiola, G. (1987). On Petri Nets with Deterministic and Exponentially Distributed Firing Times, in G. Rozenberg (Ed.) *Advances in Petri Nets* 1986, Lecture Notes in Computer Science, Vol. 266, Springer, pp. 146-161

Altera (2001). Nios Embedded Processor Software Development Reference Manual.

Altera (2007). http://www.altera.com

Avalon (2007). http://www.altera.com/literature/manual/mnl_avalon_spec.pdf Bus specification manual.

Benini, L. & de Micheli, G. (2002). Networks on Chips: A New SoC Paradigm, Computer, Vol. 35, Iss. 1, pp. 70-78, January 2002, ISSN 0018-9162

Bjerregaard, T. & Mahadevan, S. (2006). A Survey of Research and Practices of Network-on-Chip, ACM Computing Surveys, Vol. 38, article 1, March 2006, ISSN 0360-0300

Blume, H.; Feldkämper, H.; Noll, T. G. (2005). Model-based Exploration of the Design Space for Heterogeneous Systems-on-Chip, *Journal of VLSI-Signal Processing*, Vol. 40, Nr. 1, May 2005, pp. 19-34

Blume, H.; von Sydow, T.; Becker, D. Noll, T. G. (2007). Application of Deterministic and Stochastic Petri Nets for Performance Modelling of NoC Architectures, *Journal of Systems Architecture*, Vol. 53, Issue 8, 2007, pp. 466-476

Blume, H.; von Sydow, T.; Noll, T. G. (2006). A Case Study for the Application of Deterministic and Stochastic Petri Nets in the SoC Communication Domain, *Journal of VLSI Signal Processing 2006*, Vol. 43, Nr. 2-3, June 2006, pp. 223-233

Ciardo, G.; Cherkasova, L.; Kotov, V.; Rokicki, T. (1995). Modelling a scalable high-speed interconnect with stochastic Petri Nets, in: *Proceedings of the Sixth International Workshop on Petri Nets and Performance Models PNPM'95* October 03–06, Durham, North Carolina, USA, pp. 83–94.

DSPNexpress (2003). http://www.dspnexpress.de

Duato, J.; Yalamanchili, S. & Ni, L. (2003). *Interconnection Networks – An Engineering Approach*, Morgan Kaufmann, ISBN 0818678003 , San Francisco

Jensen, K. (1980). Net Models in System Development, PhD thesis, Aarhus University

Kleinrock, L. (1975). *Queueing Systems* – Vol. 1: Theory, John Wiley and sons

Kogel, T.; Doerper, M. et al. (2003). A modular simulation framework for architectural exploration of On-Chip interconnection networks, *CODES + ISSS*, October 2003.

Lahiri, K.; Raghunathan, A.; Dey, S. (2001). System-level performance analysis for designing On-Chip communication architectures, *IEEE Transactions on CAD of Integrated Circuits and Systems*, June 2001.

Lindemann, C. (1998). *Performance Modelling with Deterministic and Stochastic Petri Nets*, John Wiley and sons, ISBN 0471976466, Berlin

Madsen, J.; Mahadevan, S.; Virk, K. (2004). Network-centric systemlevel model for multiprocessor SoC simulation, in: J. Nurmi et al. (Eds.), *Interconnect Centric Design for Advanced SoC and NoC*, Kluwer Academic Publishers

Mickle, M. H. (1998). Transient and steady-state performance modelling of parallel processors, *Applied Mathematical Modelling* 22 (7) (1998) 533–543

Moore, G. (1965). Cramming more components onto integrated circuits, *Electronics*, Volume 38, Number 8, April 19, 1965

Neuenhahn, M.; Blume, H.; Noll, T. G. (2006). Quantitative analysis of network topologies for NoC-architectures on an FPGA-based emulator, .*Proceedings of the URSI Advances in Radio Science - Kleinheubacher Berichte*, Miltenberg, September 2006

Petri Nets World (2007). http://www.informatik.uni-hamburg.de/TGI/PetriNets/

Plosila, J.; Seceleanu, T.; Sere, K. (2004). Formal communication modelling and refinement, in: J. Nurmi, H. Tenhunen, J. Isoaho, A. Jantsch (Eds.), *Interconnect Centric Design for Advanced SoC and NoC*, Kluwer Academic Publishers

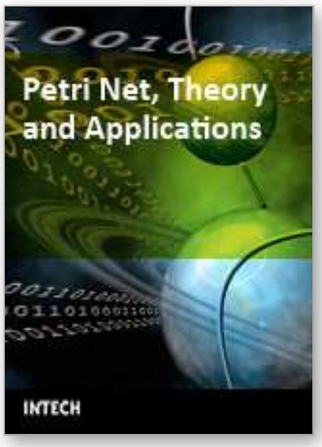
Ratzer, A. V.; Wells, L.; Lassen, H. M.; Laursen, M.; Qvortrup, J. F.; Stissing, M. S.; Westergaard, M.; Christensen, S. & Jensen, K. (2006). CPN Tools for Editing, Simulating and Analysing Coloured Petri Nets, *Proceedings of the 24th International Conference on Applications and Theory of Petri Nets (ICATPN) 2003*, pp. 450-462, ISSN 0302-9743, Eindhoven, June 2003, Springer Verlag, Berlin

Sonntag, S.; Gries, M.; Sauer, C. (2005). SystemQ: A Queuing-Based Approach to Architecture Performance Evaluation with SystemC, *Proceedings of the SAMOS V Workshop*, Samos, Greece, July 18-20 2005, LNCS 3553, ISBN 354026969, pp. 434-444

SOPC (2007). http://www.altera.com/products/software/products/sopc/sop-index.html

Texas Instruments (2007). http://www.ti.com

Zaitsev, D. A. (2004). An Evaluation of Network Response Time using a Coloured Petri Net Model of Switched LAN In: K. Jensen (ed.): *Proceedings of the Fifth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, October 2004, Department of Computer Science, University of Aarhus, PB-570, 157-166.



Petri Net, Theory and Applications

Edited by Vedran Kordic

ISBN 978-3-902613-12-7
Hard cover, 534 pages
Publisher I-Tech Education and Publishing
Published online 01, February, 2008
Published in print edition February, 2008

Although many other models of concurrent and distributed systems have been de- veloped since the introduction in 1964 Petri nets are still an essential model for concurrent systems with respect to both the theory and the applications. The main attraction of Petri nets is the way in which the basic aspects of concurrent systems are captured both conceptually and mathematically. The intuitively appealing graphical notation makes Petri nets the model of choice in many applications. The natural way in which Petri nets allow one to formally capture many of the basic notions and issues of concurrent systems has contributed greatly to the development of a rich theory of concurrent systems based on Petri nets. This book brings together reputable researchers from all over the world in order to provide a comprehensive coverage of advanced and modern topics not yet reflected by other books. The book consists of 23 chapters written by 53 authors from 12 different countries.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Holger Blume, Thorsten von Sydow, Jochen Schleifer and Tobias G. Noll (2008). Petri Net Based Modelling of Communication in Systems on Chip, Petri Net, Theory and Applications, Vedran Kordic (Ed.), ISBN: 978-3-902613-12-7, InTech, Available from: http://www.intechopen.com/books/petri_net_theory_and_applications/petri_net_based_modelling_of_communication_in_systems_on_chip

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen