Modeling and Verification of NoC Communication Interfaces

Vinitha Palaniveloo[1] Arcot Sowmya[1] Sridevan Parameswaran[1]

[1] University of New South Wales, Australia
{vinithaap,sowmya,sridevan}@cse.unsw.edu.au

Technical Report
UNSW-CSE-TR-0903
October 2009

THE UNIVERSITY OF
NEW SOUTH WALES

School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

Abstract

The projected demands of a 21st-century road map in semiconductor industry for processing power, performance, power consumption reflects the need to integrate millions of transistors on a single chip. The major markets for semiconductors are networking and communication, medical, defense, automotive and consumer electronics. To satisfy the processing power required by those segments, the industry is proposing techniques such as parallel chip architecture, multiple data processing engines, more memory interface and chip-to-chip interface. Although recent technological advances allow integration of billions of transistors into a single chip the existing electronic design automation tools are not advanced enough to handle complex chip designs. Hence, the focus has shifted to providing a communication infrastructure for massive real time parallel processing, where communication and computation can be designed independently. The concept of Network on Chip (NoC) addresses the communication requirements on chip and decouples it from computation.

One of the challenges faced by designers of NoC integrated circuits is verifying the correctness of the communication scheme for an NoC Architecture. NoCs are on-chip communication networks that borrow the networking concept from computer networks to interconnect complex Intellectual Property (IP) on chip. Therefore, the applications on IP cores communicate with peer applications through communication architecture that consists of layered communication protocol, routers and switches. The absence of an integrated architectural model poses the challenge of performing end-to-end verification of communication scheme.

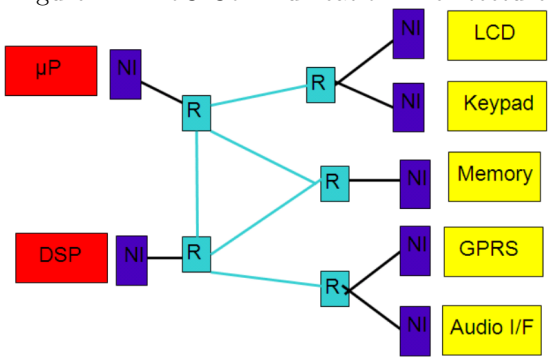
The formal models of NoC proposed so far in the literature focus on modeling parts of communication architecture such as the specific layers of communication protocol or routers or network topology but not as a integrated architectural model. This is attributed to the absence of expressive modeling language to model all the modules of the NoC communication architecture. The NoC communication architecture is heterogeneous as it consists of synchronous and asynchronous IP cores communicating through heterogeneous communication pipelines. We propose a heterogeneous modeling language for modeling and verification of NoC communication architecture. The proposed modeling language is based on formal methods as they provide precise semantics, mathematics based tools and techniques for specification and verification.

1 Introduction

The aim of this research is to design formalism for modeling Network-on-chip(NoC), a concept of interconnecting components on a chip using routers similar to computer networks[1, 2, 3, 4, 5, 6]. The basic concept idea NoC architecture is to separate the computation and communication modules of a system on chip. The basic differences between on-chip and computer networks such as influence of wiring delay, retransmission in communication protocol and guaranteed throughput inhibit us from using the design and validation techniques already used for in computer networks.

The NoC, on-chip communication architecture interfaces to IP cores through Network Interface(NI) as shown in Figure 1. The network interface can be part of IP cores or NoC or router. The IP core implements the application layer and physical layer to the network interface. The Network interface implements packetization, routing algorithms specific to the network layer and data layer and physical layer to the router. The router implements switching scheme and data layer, physical interface to router and Network interface. The Network Interface implements clock crossover algorithms to match clock between IP core and the clock of NoC communication architecture.

Figure 1.1: NoC Communication Architecture



2 Research Problem

NoCs incorporate numerous design variables such as topology, number of routers, router architecture, buffers at the router, switching schemes, routing schemes and clocking schemes. These design variables can be altered to obtain NoCs with different quality of service. NoCs with synchronous clock schemes are SPIN[7], Aethereal[8], xPIPES[9], NOSTRUM[10], HERMES[11]. NoCs with asynchronous clock schemes are MANGO[12], QNOC[13], ANOC[14] and HERMES-GLP [15].

Synchronous NoCs are in theory, implemented with isochronous clock (frequency and phase locked) throughout the chip, but practically the implementation of synchronous clocking scheme is limited by factors such as clock skew, delays, synchronization issues and synchronous latency insensitivity circuits.Asynchronous NoCs are based on the concept of confining the clocks to cores and allowing the

network to be clock-less and is called Globally Asynchronous and Local Synchronous (GALS) models [16]. Moreover, the clock scheme in futures NoC is still unpredictable [17]. However, GALS is a very powerful concept that can enable integration of more components as it is not restricted by wiring delays and synchronization issues and therefore is a prime area of research.

The modeling of synchronous NoC communication architecture can be done using synchronous formalism, since the on-chip communication network from the output of Network interface through router to another network interface is synchronous. This would enable verification of only the communication protocol at the physical interface and not end to end communication, since end-to end communication requires modeling clock cross over. Moreover, the switching at the router can have arbitrary delays depending on the traffic, hence modeling it based on synchronous clock would not be appropriate. While performing end to end verification the synchronous NoC architecture also represents an heterogeneous architecture. Since both the synchronous and asynchronous noc represent an heterogeneous architecture we need an heterogeneous modeling language.

3 Related Work

There are many simulation tools for verification of NoC. But the simulation tools using System C and VHDL require RTL-level implementation and details for verification and these tools do not provide techniques to analyze the reason for failure. The communication scheme can be modeled with different levels of abstraction can be done using formal methods. The need for formal methods at various phases of NoC design and development is emphasized in the literature[18].This research is aimed at identifying a suitable formal method to model and validate NoC. Simulation based design exploration frameworks such as OCCN [19] and ProtoNoC [20] are suitable for a specific communication scheme and NoC architecture. Secondly, they do not provide techniques to analyze the reason for failure since simulation-based techniques are semi-informal.

There are a number of existing works on formal modeling and verification of NoCs [21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32]. However, the formalisms proposed so far, are those already developed for other applications that are control-flow or data-flow based and that are not network based. NoC is a networked data flow application. The formalisms proposed for modeling NoCs in the literature are graphs, finite state machines (FSM) or Petri-Nets (PN). The NoCs are modeled and verfied using verification systems such as PVS Theorem prover [32], ACL2 Theorem prover[33], Finite State Process (FSP) [9], Communicating Hardware Process (CHP) [13], B-Method [12] and Specification Description Language (SDL)[10] that has a specification language, integrated support tools such as theorem prover for verification. Some verification systems such as FSP and CHP does not provide integrated verification tools therefore separate verification tools based on automated theorem proving and model checking are used.

The graph and FSM formalisms were proposed for modeling control applications. An extended graph based formalisms called data flow graphs was proposed to model data flow applications. Hence, graph and FSM models can be used to model simple data flow applications but they are not suitable for

modeling a network of data-flow application such as NoC. Due to this limitation in the selected formalism only an highly abstract model of NoC with a subset of communication or NoC architecture design variables has been modeled. The abstractions depend on the tools provided by formal methods for modeling and secondly on the criteria used to select design variables for modeling. Since NoC design variables are interrelated, these highly abstract models created with limited design variables do not guarantee the results of validation when the parameters change.

The formal models developed so far are for a specific NoC architecture to verify a specific communication scheme or a communication interface. The NoC communication scheme layered similar to the computer networks. The layered concept of networking was developed to accommodate changes in technology. Each layer of a specific network model may be responsible for a different function of the network. Each layer will pass information up and down to the next subsequent layer as data is processed. The application layer allows decoupling functional application from target hardware. The verification of NoC communication scheme is usually done as peer-to-peer communication at different levels of abstraction for each layer and is not interconnected to another layer as modeling each layer requires different formal methods and interactiob between different formalisms. The verification done for specific layers do not hold for a integrated system. Since, each communication scheme requires different formalism, the formal models cannot be reused for the same NoC when different communication scheme is used. Therefore, the focus is on system level modeling and verification of NoC.

In order to validate a new architecture or communication scheme or for changing a topology,a completely new model becomes necessary. Although, an attempt to define generic models exists [24], it does not consider translation of a generic model to a specific NoC. The proposed generic model in ACL2 was used to verify message ordering in the network layer of the protocol stack. The generic model does not support any notion of time, multiple active networked nodes, irregular network topology or bounded buffer size at the nodes and the granularity of switching is limited to a packet.

NoC communication schemes are packet based, the indeterminism in routing and arbitration makes synchronous network as good as asynchronous networks except for the physical layer switching [28]. Therefore, we propose to use a formalism with a notion of time and allow representation of indeterminism at at switching layer besides representation of synchronous and asynchonous communication interfaces at the physical layer of the network. The formal methods proposed so far are for either synchronous NoCs or asynchronous NoC. Graphs, finites state machines were used for modeling synchronous NoC[21, 22, 23, 24, 25, 26]. CHP and ASC an extension of System C, for modeling asynchronous circuits in asynchronous modes of NoC [30, 31]. Little research is targeted at modeling GALS based NoCs; Haskell based ForSyde [32], proposes an extension of synchronous formalism by refinements to incorporate multiple clock domains, channel delays, jitters and channel mapping.

4 Proposed work

Heterogeneous Protocol Automata (HPA) is a formalism recently designed by the group for modeling communication interfaces of heterogeneous modules in Network on chip. HPA is an extension of Synchronous Protocol Automata [30] a synchronous formalism that was proposed to model hardware bus architectures in system on chip. SPA formalism models communication on buses at a low level of abstraction, describing behavior of signals for every clock tick. SPA is a proven formalism that is used to synthesise protocol converters between incompatible bus protocols in [31, 32].

Modeling network of heterogeneous modules at the lower level system layers requires modeling the communication bus, wrapper interface between the heterogeneous modules, bus protocol and system. SPA formalism is proposed for modeling synchronous bus interfaces only. In HPA we will use the formal notation and semantics of SPA to model synchronous buses; in addition, we will extend the formal semantics to model asynchronous communication buses. Since, the formal semantics of both synchronous and asynchronous interfaces use the same formal notation, problem of modeling wrapper interface which is essential for interaction between heterogeneous modules is simplified. Thus, HPA is extended to support modeling both synchronous and asynchronous communication between heterogeneous processes using a single formalism.

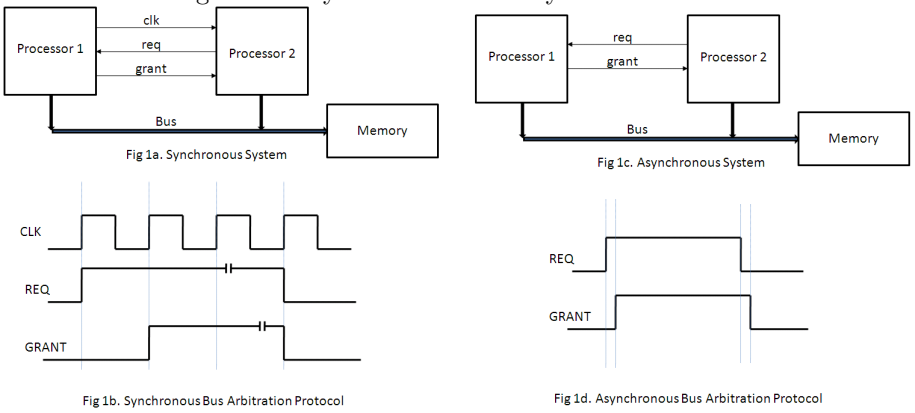
In HPA the processes are represented as finite state machines (FSM) that can transition based on clock tick or events. The clocked communication processes are modeled as FSMs that transition at every clock tick based on the process's clock; at the transition there are guards and communication action actions that are performed before transiting to the next state. The un-clocked communication processes are modeled as FSMs that makes a transition for events; at the transition there are pre-guards and post-guards that must be satisfied before and after the communication action is performed. The status of the signals broadcast to all the FSMs and the signals themselves can be modeled to be read instantaneously when written, as in synchronized message passing and CSP style rendezvous message passing or the signals can be read after they are written using operations such as polling the status of the signal. The formalism has semantics to check the status of the signal in the previous clock cycle, to check if there was any edge transition in case of event triggered FSM.

5 Formal Definition

The GALS communication interfaces of NoC are modeled as as a combination of Synchronous Finite State Machines (SFSM) and Asynchronous Finite State Machines (AFSM) . The transitions in SFSM are triggered by the clock tick. The transitions in AFSM are triggered by events. The modeling language used to model both the asynchronous and synchronous functions is known as Heterogeneous Protocol Automata (HPA).The interaction at the communication interfaces are through an electrical link, which acts as a buffer to store status of signal. The control signals are single length buffered communication channels.

The following example demonstrates handshake between microprocessor and slave devices to request bus control. In the synchronous model, the actions are performed at the clock tick. In asynchronous model, the action are performed

Figure 5.1: Synchronous and Asynchronous FSMs



on edge transitions of the signal. The difference between the synchronous and asynchronous model is determinism in time.

5.1 Definition Adapted from SPA

The HPA retains some of the the formal definitions of SPA formalism such as types of automata communication channel, synchronized read and write on control channels. The HPA has two types of channels: control and data channels. These channels are further classified into input and output channels based on the read or write actions performed on the channel. Therefore, the automata has input control channel, output control channel, input data channel and output data channel.

The actions on an input control channel are: reading presence of a signal a? , reading absence of a signal #a . The read action on the transition acts as a guard takes the transitions immediately when the guard becomes true.

The actions on the output control channel are: write presence of a signal a!. The write action on output channel permits delayed write if there is a silent transition (Tau) in the state or synchronous write where the write happens in the next clock tick and it is not permitted to stay in the state for more than one clock tick unless it has an explicit self-loop.

The write actions on output data lines is denoted by $d!$. The read actions on input data lines is denoted by $d?$ reading for electrical lines at the same instant or some time later. so its like reading from a single size memory. The data lines are non-blocking. The write actions are non-blocking, the read channel act as guards on the transition and hence they are blocking.

5.2 Definition proposed in HPA

New definitions are introduced in HPA to enable the same modeling of both synchronous and asynchronous interfaces in the same formal language. The difference between synchronous and asynchronous model is specified in automata definition. Similar to SPA automata, the synchronous automata is defined to executed in locked step at every clock tick. The aynchronous automata is defined to execute independent of clock based on signal transitions. The semantics

differentiates synchronous and asynchronous model with the definitions of automata.

In the output control channel, an action is defined for writing absence of a signal#a!. This action is added as our definition assumes that signal presence is sustained until the absence is written explicitly incontrast to definition of SPA where assertion has to be written every cycle to sustain a signal or it automatically de-asserts in the next cycle. Similarly in Esterel, sustain(S) is defined to sustain presence of signal until deasserted.

In the input control channel additional actions are defined to enable delayed reading of signals. The new definitions are: delayed reading of signal presence a?? and delayed reading of signal absence #a??. The delayed reading takes the transition if the signal is present or awaits till the signal is present. The read can happen either before the channel is written or after the channel is written but either way it takes the transition after it is true. The signal written once can be read multiple times, but every delayed read should have a write in it is path since the initial state. This is definition performs action similar await(S) and present(S) in Esterel.

All the control signals in the input control channel has a signal register. The register stores the status of control signal in the previous clock cycle in SFSM and previous transition in AFSM. The register updates every clock tick in SFSM and it is updated during state transitions in AFSM. The register associated with the control channel can be read using $\$channel\_name$. Esterel language uses $pre(?S)$ to store signal values in the previous clock cycle.

In an FSM the states indicate processing and transition indicate input or output actions required to go to next state. The processing at the state can take n clock cycles, for $n >= 1$ clock cycle. If the processing takes more than 1 clock cycle it is indicated by a self-loop on that state with a Tau action in SPA, but in HPA we denote states having explicit self-loop with a suspend action denoted by $s_{suspend}$. In SPA, the states with Tau action comes out of the self-loop if any of the other outgoing transition is true; whereas in HPA the outgoing transition is taken only after the computations in the state are complete therefore they are called suspend action.

The complementary actions on data channels $d!$ and $d?$ need not happen at the same instant, they can be synchronized or asynchronous. Each data channel has a type and length associated with it. $channel\_type(d)$ can be serial or parallel. $size(d)$ is integer which denotes width of d in parallel type, $size(d)$ is not used for serial data channel. $access\_type(d)$ can be virtual channel or time multiplexed or simple circuit switched. The $slot(d)$ and $slotid(d)$ are properties of multiplexed access type only; $slot(d)$ denotes maximum slot for multiplexed access, $slotid(d)$ current access slot. The $channel(d)$ and $priorityid(d)$ are properties of virtual channel types only; $channel(d)$ denotes maximum number of virtual channels permitted for the channel, $channelid(d)$ denotes the virtual circuit id of the current item.

The data channels with Multiplexed, Pipelined data interfaces require FIFOs at source and destination. The example of multiplexed data channel is TDMA interface and pipelined data interface is AMBA processor bus interface. The data channel with virtual channel interfaces require FIFO only at the source. Simple point to point data channels that transmit and receive one data requires no FIFOs. The data channels that are associated with a FIFO are modeled as counters in FSM with actions to increment or decrement counter. When the

data is read from the input data channel into FIFO the counter increments, when the data is read from FIFO the counter decrements. When the data is written on the output data channel the counter decrements and when data is written in FIFO the counter increments.

Finally, the synchronous and asynchronous FSM have different formats of signal transition. Generically, the actions on a transition are defined as

$$s \xrightarrow{B1;C;B2} s'$$

where, B1 is a pre-guard and C is the communication action and B2 is a post-guard. In SFMS B2 is not used, the transition are of the format.

$$s \xrightarrow{B1;C} s'$$

The blocking read actions on input control channel appears as pre-guard in B1 and non-blocking write actions on output control and data channels belong to communication action in C. In AFMS B1, B2 are used for modeling asynchronous systems, where B1 and B2 are optional. The system waits in 's' till it receives B2 to transition to s' . The execution of actions happen in the order they appears.

Table 5.1: Channel Operations

Operation	Input Channel	Output Channel
Instananeous Write (control signal)		$a!$
		$\#a!$
Instantaneous Read (control signal)	$a?$	
	$\#a?$	
Delayed Read (control signal)	$a??$	
	$\#a??$	
Read Previous (control signal)	$\$a$	
	$\$\#a$	
Self-loop on the state		$s_{suspend}$
Write Data (data signal)		d!
Read Data (data signal)	d?	

5.3 Assumptions on Channel Properties

Control Channel

The control channels represent the hardware electrical wires. The control channels are not pipelined and hence are one word bounded buffers. Since it is a one word bounded buffer, it does not need any buffer management. The signal can be overwritten by the sender. The receiver must be designed to read instaneously if the message is not to be lost. There are no centralized processes such as semaphores in hardware to co-ordinate message transfer. Hence the sender is always non-blocking. This means that the sender writes the control

signal whenever it is ready to write. The receiver process reads it instantaneously (equivalent to synchronized handshake in formal language) if it has been waiting to read. The receiver can perform a delayed read (equivalent to asynchronous communication in formal language).

Data Channel

The data channels represent the hardware electrical wires. The data channels can be timed-division multiplexed (TDM) or re-used as virtual channels. The TDM data-channels are used in synchronous pipeline. Virtual channels are used in asynchronous pipeline. The receiver can perform a delayed or instantaneous read. Hence it is represented as read on data channel.

6 Definition of Heterogeneous Protocol Automata

A Heterogeneous protocol Automaton (HPA) is a finite state machine with bounded counters to model clocked and clockless systems. The definition allows modeling of event based transitions and clock based transitions.

HPA is a tuple A=(Q,clk,C, D,V, T,q_i,q_f), where

- Q is a set of protocol states
- clk defines if the automaton works on clock ticks or not
- C is a set of input and output control channels ($C_I \cup C_O$)
- V is a set of internal counters that can be associated data channels, buffers and fifos.During initialization V(d) and Capacity(V) is defined. In V(d), d defines the data channel or buffer or fifo associated with the counter and Capacity (V) defines the capacity of counter. The counter can be incremented or decremented using $V++$ or $V--$ respectively. The increment and decrement action on the counters happen at the states not at the transition.
- D is a set of input and output Data channels ($D_I \cup D_O$)
- T is the transition relation $T \subseteq Q \times A(C) \times A(D) \times A(D_c) \times Q$, where
- $A(C)$ is the set of actions on the control channels, $A(D)$ is the set of actions on the data channels, $A(D_c)$ is the set of actions of the counters (if any) of data channels.
- $A(C) = \{s!, \#s!, \#s, s?, s??, \#s??, s_{suspend}, \$s\}$ for $s \in C$
- $A(D) = \{d!, d?\}$ for $d \in D$.During initialization $width(d), slot(d), type(d), v(d)$ are defined.
- q_i is the initial state
- q_f is the final state

Definition of Path

A path in a HPA automaton is a sequence of alternating states and transitions. Hence a path of A is given as $\pi_{A_n} = q_0 \xrightarrow{Q_1} q_1 \xrightarrow{Q_2} q_2 \cdots \xrightarrow{Q_k} q_k$ such that for all $0 \leq m < k, q_m \xrightarrow{Q_n} q_{m+1}$ is a transition in the automaton and n is the number of paths in automaton.

- $\mid \pi \mid$ denotes the number of transitions in π, also known as length of π
- $Paths(A, q_j, q_k)$ denotes the (possibly infinite) set of paths in A from q_j to q_k
- $Writepresences(\pi, s) = i \in \mid 0 < i \leq \mid \pi \mid \wedge s! \in Q_k$ is the set of indices on the path π where there is a write action $s!$ on control channel s
- $Writeabsences(\pi, s) = i \in \mid 0 < i \leq \mid \pi \mid \wedge \#s! \in Q_k$ is the set of indices on the path π where there is a write action $\#s!$ on control channel s
- For a path $\pi = (q_0, s_0) \xrightarrow{Q_0,S_0} \cdots \xrightarrow{Q_k,S_k} (q_{k+1}, s_{k+1}) \in A \| B$, where $q_j \in A$ and $s_j \in B$, the projection of π on A, $\pi \| A = q_0 \xrightarrow{Q_0} q_1 \cdots \xrightarrow{Q_k} q_{k+1}$. The projection $\pi \| B$ is defined similarly.

7 Rules for communication on control channels

7.1 Rules for synchronous transition

The basic rules for correct communication between synchrononous FSMs on control and data channels are:

- The write action of control signal in output control channel communicate only with the read actions of the same control signal on input control channel. That is if control signal $a \in C$, C being control channel. The write actions $a!$ of signal a can be read only using read actions $\#a!, a?, a??, \#a??$ of control signal a.
- The write actions of data signals in data channel communicate only with the read actions of then same data signal on data channel. The interpretation of this rule is similar to the above rule.
- Cyclic dependancy between the control signals is not permitted, if $q_1 \xrightarrow{B_1;C_1} q_2$ and $s_1 \xrightarrow{B_2;C_2} s_2$ then B_1 must not be dependant on C_2 and B_2 must not be dependant on C_1
- The FSMs operate in locked steps, at each clock tick both the FSMs check the transition that can be taken and move to the next state.
- The order of appearance is preserved or checked as a union at the end of transition - to be finalized.

7.2 Rule for instantaneous read and write

The write ($a!$) and read ($a?$) must happen in the same transition. If the write has not occured the automaton waits in the previous state to make a synchronized transition after write occurs.

$$q_1 \xrightarrow{Q_1} q_2 \in A \tag{7.1}$$

$$s_1 \xrightarrow{S_1} s_2 \in B \tag{7.2}$$

$$s! \in Q_1 \wedge s? \in S_1 \tag{7.3}$$

$$(q_1, s_1) \xrightarrow{Q_1, S_1} (q_2, s_2) \in A \| B \tag{7.4}$$

If the instantaneous read is ocurring without a instantaneous write it mus be excluded in the correct communication subset.

$$q_1 \xrightarrow{Q_1} q_2 \in A \tag{7.5}$$

$$s_1 \xrightarrow{S_1} s_2 \in B \tag{7.6}$$

$$s! \notin Q_1 \wedge s? \in S_1 \tag{7.7}$$

$$(q_1, s_1) \xrightarrow{Q_1, S_1} (q_2, s_2) \notin A \| B \tag{7.8}$$

7.3 Rule for Suspended/Delayed Write

The number of cycles the automaton waits in the previous cycle depends on the time writing is suspended to complete the operation in the current state.

$$q_1 \xrightarrow{Q_1} q_1 \wedge q_1 \xrightarrow{Q_2} q_2 \in A \tag{7.9}$$

$$s_1 \xrightarrow{S_1} s_2 \in B \tag{7.10}$$

$$s_{suspend} \in Q_1 \wedge s! \in Q_2 \wedge s? \in S_1 \tag{7.11}$$

$$(q_1, s_1) \xrightarrow{Q_1} (q_1, s_1) \in A \| B \tag{7.12}$$

$$(q_1, s_1) \xrightarrow{Q_2, S_2} (q_2, s_2) \in A \| B \tag{7.13}$$

7.4 Rule for delayed read on control channel

The read ($a??$) requires a write ($a!$) in the any of the previous transitions in the path, without being overwritten by another write. The value on signal a is read after the operation is previous cycle is complete and does not need to take a transition when the status is true. Therefore, the number cycles the automata remains in the current state depends on the operation complexity in the state. Similar to the instantaneous read, the automaton remains in the state checking status of a if it has not be written previously. Therefore, the automaton remains in the state after the computation is complete until the guard on the outgoing transition becomes true. The rule for delayed read consists of three sub-rules depending on when read is happening.

Delayed read happening after write

First, it is checked if there is any valid write in the path to the present state. The valid write one where the value of the signal is not being written over by

other action. If there is a presence of valid write in the previous path the state transitions to next state.

Delayed read happening before write

First, it is checked if there is any valid write in the path to the present state. If there is no valid write, it wails for write to occur in the future and the delayed read synchronizes with future write.

Postponed Read

Here, the checking of the value is postponed even after a valid write is on the path. This option is given to enable process to remain in the present state for n clock cycles to complete its task. The process then checks the status of postponed read to relative state of the other process.

Delayed read happening after write

$$q_i \xrightarrow{Q_i} q_k \cdots q_{k+n} \xrightarrow{Q_{k+n}} q_f \in A \text{ where i is initial state, f is final state} \tag{7.14}$$

$$s_m \xrightarrow{S_m} s_{m+1} \in B \wedge s?? \in S_m \tag{7.15}$$

$$\pi_n = Paths(A\|B, (q_i, s_i), (q_k, s_m)), \text{paths from } (q_i, s_i) \text{ to } (q_k, s_m) \tag{7.16}$$

$\forall$ paths π_n ,

$$Writepresences(\pi_n, s) = \{x \in N | 0 < x \leq |k| \wedge s! \in Q_k\} \tag{7.17}$$

$$Writeabsences(\pi_n, s) = \{y \in N | 0 < y \leq |m| \wedge \#s! \in Q_k\} \tag{7.18}$$

Check if the last action on control channel s was a write action in in atleast one path of π_n. $\forall n$, check if write present is on channel s at the last index l

$$Writepresences(\pi_j, s)[l] > Writepresences(\pi_j, s)[l], j \in n \tag{7.19}$$

$$\text{if so,} (q_k, s_m) \xrightarrow{Q_k, S_m} (q_{k+1}, s_{m+1}) \in A\|B \tag{7.20}$$

The order of appearance of actions on transition is not considered, it is sampled just before making a transition to next state. If there was a write action in atleast one path of in the previous states as well as a write in the present state. The status of signal written in the present cycle overrides the past status as the signals are sampled at the end of clock cycle not in micro ticks.

$$Writepresences(\pi_j, s)[l] > Writepresences(\pi_j, s)[l], j \in n \tag{7.21}$$

$$\text{if} \#a! \in Q_k \tag{7.22}$$

$$\text{then} (q_k, s_m) \xrightarrow{Q_k, S_m} (q_{k+1}, s_{m+1}) \notin A\|B \tag{7.23}$$

$$\tag{7.24}$$

exception, if there is a presence of write absence and write presence in more than one path the delayed read is preserved to be decide during verification.

$$Writepresences(\pi_a, s)[l] > Writepresences(\pi_j, s)[l], forpatha \tag{7.25}$$

$$Writeabsences(\pi_b, s)[l] > Writeabsences(\pi_j, s)[l], forpathb \tag{7.26}$$

$$\text{if so,} (q_k, s_m) \xrightarrow{Q_k, S_m} (q_{k+1}, s_{m+1}) \in A\|B \tag{7.27}$$

Postponed Read

Even if the previous write presences are true, the state can decide not to take the transition and postpone taking a transition. The number of postpone cycle is decide by the state.

$$Writepresences(\pi_j, s)[l] > Writepresences(\pi_j, s)[l], j \in n \tag{7.28}$$

$$\text{if so,}(q_k, s_m) \xrightarrow{Q_k, S_m} (q_k, s_{m+1}) \cdots \xrightarrow{Q_k, S_{m+r}} (q_{k+1}, s_{m+r}) \in A \| B \tag{7.29}$$

Delayed read happening before write

If the above is not true then check for future write presence on channel s

$$\pi_{n1} = Paths(A \| B, (q_k, s_m), (q_k, s_f)), \text{paths from } (q_k, s_m) \text{ to } (q_k, s_f) \tag{7.30}$$

$\forall$ paths π_{n1} ,

$$Writepresences(\pi_n 1, s) = \{x \in N | 0 < x \leq |k| \wedge s! \in Q_k\} \tag{7.31}$$

$$Writeabsences(\pi_n 1, s) = \{y \in N | 0 < y \leq |m| \wedge \#s! \in Q_k\} \tag{7.32}$$

Check if the first action on control channel s in future is a write action in atleast one path of π_{n1}. $\forall n1$, check if write present is on channel s at the first index 1

$$Writepresences(\pi_j, s)[1] > Writepresences(\pi_j, s)[1], j \in n \tag{7.33}$$

$$\text{if so,}(q_k, s_m) \xrightarrow{Q_k, S_m} (q_{k+1}, s_{m+1}) \in A \| B \tag{7.34}$$

if there is no past or future writes then,

$$(q_k, s_m) \xrightarrow{Q_k, S_m} (q_{k+1}, s_{m+1}) \notin A \| B \tag{7.35}$$

7.5 Rule for read previous on control channel

The read previous ($\$a$) action updates the status of the signal in the previous clock cycle is constantly. It is therefore not dependent on the present value of the signal. The value of a is updated in a virtual register $reg(s)$ constantly in the background by a FSM before (or) at the end of transitioning to next state.

$$q_i \xrightarrow{Q_i} q_{i+1}(reg(a_i)) \xrightarrow{Q_i+1} q_{i+2}(updatereg(a_{i+1})) \tag{7.36}$$

$$q_{i+2} \xrightarrow{Q_i+2} q_{i+3}(updatereg(a_{i+2})) \cdots \tag{7.37}$$

$$q_k \xrightarrow{Q_k} q_{k+1}(updatereg(a_k)) \cdots \tag{7.38}$$

$$q_{k+n} \xrightarrow{Q_{k+n}} q_f \in A \text{ where i is initial state, f is final state} \tag{7.39}$$

$$s_m \xrightarrow{S_m} s_{m+1} \in B \wedge \$s \in S_m \tag{7.40}$$

$$(q_k, s_m) \xrightarrow{Q_k, S_m} (q_{k+1}, s_{m+1}) \in A \| B \text{iff} reg(a_{k-1}) = \text{ presence of } a \text{ even if } \#a \in Q_k \tag{7.41}$$

Difference between delayed read and read previous

The delayed read ($a??$) reads previously written present status of the signal.Therefore delayed read cannot be used instead of read previous. The use of read previous ensures that it is in that state for one clock cycle, but using delayed read the transition takes times which is dependent on the state that does delayed read. The delayed read can be used for loosely synchronous models, that can resynchronize after read operation.

7.6 Additional rules for asynchronous transition

The rule for actions are same as the above sections. The transition takes place on clock tick in synchronous FSM but in asynchronous FSM the transition is based on actions, not on time. The rules for asynchronous transition are:

$$q_1 \xrightarrow{B_{q1};C_q;B_{q2}} q_2 \tag{7.42}$$

$$s_1 \xrightarrow{B_{s1};C_s;B_{s2}} s_2 \tag{7.43}$$

then B_1 must not be dependant on C_2 and B_2 must not be dependant on C_1

- $B_{q1}; C_q; B_{q2}$, where B_{q1} and B_{q2} are guards (optional) and C_q can be data or control channel non-blocking actions
- $B_{q1}; C_q$ communicates with $C_s; B_{s2}$ or $C_q; B_{q2}$ communicates with $B_{s1}; C_s$
- Cyclic redundancy between guards and communication channels are permitted
- $B1; C; B2$ communication with $C; B1; C$ is not permitted now.

7.7 Rule for counters

The increment $(v++)$ and decrement $(v--)$ actions that happen on counter (v) are preserved at the states during parallel composition.

8 Synchronous Product

Given two HPA automata,A =(Q,clk_i,C_i, D_i, T_i,q_0,q_f) and B =(S,clk_i,C_i, D_i, T_i,s_0,s_f), i = 1,2.

The synchronous product is derived when two automata operate synchronously, that is clk1 and clk2 are frequencyand phase locked (isochronous) derived from the same global source. That is $clk_1 = clk_2$.

There is absence of cyclic redundancy on control lines. $C_{I1} \cap C_{I2} = \phi$ and $C_{O1} \cap C_{O2} = \phi$ and $D_{I1} \cap D_{I2} = \phi$ and $D_{O1} \cap D_{O2} = \phi$.

We define synchronous product automaton as $A\|_{sync}B = (QXS, C_1 \in C_2, D_1 \in D_2, \rightarrow, (q_0, s_0), (q_f, s_f))$, where $(q1, s1) \rightarrow Q_1, S_1(q_2, s_2)$ is a transition of $A\|_{sync}B$ iff $q_1 \rightarrow Q_1 q_2$ and $s_1 \rightarrow S_1 s_2$, such that $may(Q_1, S_1)$ is true and set of actions is $Q1 \cup S1$.

The pruning rules can be used to derive the actual system from complete product composition using $may(Q_1, S_1)$ predicates. The synchronous transitions Q_1 and S_1 are of the form B_1, C, where B_1 is guard and C is communication. The guard operations $B_1 = \{s?, \#s, s??, \#s??, \$s, s_{suspend}\}$ and communication actions are $C = \{s!, d!, d?\}$. The guards are blocking actions, where as communication actions are non-blocking.

9 Asynchronous Product

Given two HPA automata,A =(Q,clk_i,C_i, D_i, T_i,q_0,q_f) and B =(S,clk_i,C_i, D_i, T_i,s_0,s_f), i = 1,2.

The asynchronous product is obtained two automata operate asynchronously, that is, if clk_1 and clk_2 are absent (or) clk_1 is present but clk_2 is absent (or) clk_2 is present but clk_1 is absent (or) clk_1 and clk_2 are present but they are not frequency and phase locked (mesochronous) although derived from the same global source (or) clk_1 and clk_2 are present but they are not frequency and phase locked derived from independent source

There is absence of cyclic redundancy on control lines. $C_{I1} \cap C_{I2} = \phi$ and $C_{O1} \cap C_{O2} = \phi$ and $D_{I1} \cap D_{I2} = \phi$ and $D_{O1} \cap D_{O2} = \phi$.

We define asynchronous product automaton as $A \|_{async} B = (QXS, C_1 \in C_2, D_1 \in D_2, \rightarrow, (q_0, s_0), (q_f, s_f))$, where $(q1, s1) \rightarrow Q_1(q_2, s_1)$,$(q1, s1) \rightarrow S_1(q_1, s_2)$,$(q1, s1) \rightarrow Q_1, S_1(q_2, s_2)$ is a transition of $A \|_{async} B$ iff $q_1 \rightarrow Q_1 q_2$ and $s_1 \rightarrow S_1 s_2$, such that $may(Q_1, S_1)$ is true and set of actions is $Q1 \cup S1$.

The pruning rules can be used to derive the actual system from complete product composition using $may_a(Q_1, S_1)$ predicates. The asynchronous transitions Q_1 and S_1 are of the form B_1, C, B_2, where B_1,B_2 are guards and C is communication. The guard operations $B_1, B_2 = \{s?, \#s, s??, \#s??, \$s, s_{suspend}\}$ and communication actions are $C = \{s!, d!, d?\}$. The guards are blocking actions, where as communication actions are non-blocking.

Definition of $may(Q_1, S_1)$ perdicates

The predicate $may(Q_1, S_1)$ is true for two actions in Q_1 and S_1. The transitions Q_1 and S_1 are of the form $B_1; C$ for synchronous product and $B_1; C; B_2$ for asynchronous product. The cyclic dependency is not permitted between actions on control channel in Q_1 and S_1 for synchronous product. But, the cyclic dependency between Q_1 and S_1 actions are permitted for asynchronous product as they are not based on clock tick but edge transitions.

The predicate is true iff for every control channel for $s \in C_1, C_2$, the actions in B_1 and B_2 have communication action on C :

- if $s? \in Q_1$, then $s! \in S_1$
- if $\#s \in Q_1$, then $\#s! \in S_1$
- if $s?? \in Q_1$, then $s!$ in the path of S_1 and $\#s!$ is not in the path after $s!$
- if $\#s?? \in Q_1$, then $\#s!$ in the path of S_1 and $s!$ is not in the path after $\#s!$
- if $\$s \in Q_1$, then $s!$ in the previous cycle of S_1 and $\#s!$ is not in the path after $s!$

10 Correct communication subset

The algorithm to generate communication subset from complete parallel product or complete synchronous product will be derived in this section. From the complete product the rules for various actions on control and data channel can be implemented succesively to obtain correct.

If two synchronous FSMs are combined, the complete synchronous product is used to obtain a synchronous FSM based on clock tick. If two asynchronous FSMs or one synchronous and one asynchronous FSM is combined the desired complete parallel product will be used to obtain an asynchronous FSM based on signal transition. The communication rules for control and data channel action will be used to obtain correct communication subset, which will be used for modelchecking or simulation based verification.

The series of algorithms required to extract correct communication are:

- Correct instantaneous read and instantaneous write - keep the transitions where $a!$ and $a?$ appear in pair. Remove the transitions where $a?$ appears without pair. Keep the transitions with $a!$ as they will be required for delayed read.
- Correct instantaneous read and suspended write - Keep the self-loops on the state as they preserve the time delay.
- Correct delayed read - check if there is write in the previous paths without overwritten value, if so keep the $a??$ transition else remove the transition with $a??$
- Correct previous read
- Give extraction rule when one is sync FSM and other is async FSM or both are async FSM where cyclic dependency is allowed on the transition.
- For different types of data channels and counters

11 Complete Parallel Product

Given two HPA automata,A =(Q,clk_i,C_i, D_i, T_i,q_0,q_f) and B =(S,clk_i,C_i, D_i, T_i,s_0,s_f), i = 1,2.

The complete parallel product is nothing but gross product. It is defined as $A \mid B = (Q \times S, C_1 \cup C_2, D_1 \cup D_2, \rightarrow, (q_0, s_0), (q_f, s_f))$, where $(q_1, s_1) \xrightarrow{Q_1,S_1} (q_2, s_2)$ is a transition of $A \mid B$ iff $q_1 \xrightarrow{Q_1} q_2$ or $s_1 \xrightarrow{S_1} s_2$. Here the where the FSMs progress asynchronously and they will be pruned further based on rules on control channel.

The path in a parallel composition of two HPA automata is $A \mid_B$ is defined as $\pi_{A,B} = (q_0, s_0) \xrightarrow{Q_1,S_1} (q_1, s_1) \xrightarrow{Q_2,S_2} (q_2, s_2) \cdots \xrightarrow{Q_k,S_k} (q_k, s_k)$ such that there exists matching paths π_{A_n} and π_{B_m} in A and B. such that

$$\pi_{A_n} = q_0 \xrightarrow{Q_1} q_1 \xrightarrow{Q_2} q_2 \cdots \xrightarrow{Q_k} q_k$$
$$\pi_{B_n} = s_0 \xrightarrow{S_1} s_1 \xrightarrow{S_2} s_2 \cdots \xrightarrow{S_k} s_k$$

Algorithm for obtaining gross product is shown below:

Algorithm 1 CompleteParallelProduct(A,B)

Input:Two FSMs.This is used if both or one of the two FSMs are asynchronous FSM
Output: $C = A \mid B$, complete parallel composition of two FSMS with all the states and transitions from initial state.
$P_c = (q_i, s_j)$ // list of state in the complete product starting from initial states of A and B
for all states $(q_1, s_2) \in P_c$ **do**
  for all transitions $q_1 \xrightarrow{Q_1} q_2 \in A$ **do**
    for all transitions $s_1 \xrightarrow{S_1} s_2 \in B$ **do**
      Add transitions $(q_1, s_1) \xrightarrow{Q_1 \cup S_1} (q_2, s_2)$
      if $(q_2, s_2) \notin P_c$ **then**
        Add (q_2, s_2) to P_c
      end if
    end for
  end for
end for

12 Complete Synchronous Product

Given two HPA automata,A =(Q,clk_i,C_i, D_i, T_i,q_0,q_f) and B =(S,clk_i,C_i, D_i, T_i,s_0,s_f), i = 1,2.

The complete synchronous product is defined as $A \parallel B = (Q \times S, C_1 \cup C_2, D_1 \cup D_2, \rightarrow, (q_0, s_0), (q_f, s_f))$, where the FSMs progress in locked step. The two automata operate in locked step when clk_1 and clk_2 are frequency and phase locked (isochronous) derived from the same global source. For this rule, $clk_1 = clk_2$. There must be absence of cyclic redundancy on control lines. $C_{I1} \cap C_{I2} = \emptyset$ and $C_{O1} \cap C_{O2} = \emptyset$ and $D_{I1} \cap D_{I2} = \emptyset$ and $D_{O1} \cap D_{O2} = \emptyset$.

If $q_1 \xrightarrow{Q_1} q_2$ and $s_1 \xrightarrow{S_1} s_2$ then $(q_1, s_1) \xrightarrow{Q_1, S_1} (q_2, s_2) \in A \parallel B$ if $Rel(Q_1, S_1)$ is true. The $Rel(Q_1, S_1)$ are pruning rules based on rules for communication on control channel. The synchronous transitions $S1$ and $S2$ are of the form $B1, C$, where $B1$ is guard and C is communication. The guard operations $B1 \subset \{s?, \#s, s??, \#s??, \$s, s_{suspend}\}$ and communication actions are $C \subset \{s!, d!, d?\}$. The guards are blocking actions, where as communication actions are non-blocking.

The following $Rel(Q_1, S_1)$ rules are used to obtain the communication subset of the system.

Instantaneous Write and Instantaneous Read

$Rel_a(Q_1, S_1)$ is true for sets of actions on control channels of Q_1, S_1, when one action is blocking and other is non-blocking.

$$\text{if} a! \in Q_1 \text{and} a? \in S_1 \tag{12.1}$$
$$\text{if} \#a! \in Q_1 \text{and} \#a \in S_1 \tag{12.2}$$
$$\text{if} a! \in S_1 \text{and} a? \in Q_1 \tag{12.3}$$
$$\text{if} \#a! \in S_1 \text{and} \#a \in Q_1 \tag{12.4}$$
$$\text{then}(q_1, s_1) \xrightarrow{Q_1, S_1} (q_2, s_2) \in A \parallel B \tag{12.5}$$
$$\tag{12.6}$$

Instantaneous write and Delayed read

$Rel_b(Q_1, S_1)$ is true for sets of actions on control channels of Q_1, S_1, when one action is blocking and other is non-blocking.

$$\text{if} a! \in Q_1 \text{and} a?? \in S_1 \tag{12.7}$$
$$\text{if} \#a! \in Q_1 \text{and} \#a?? \in S_1 \tag{12.8}$$
$$\text{then}(q_1, s_1) \xrightarrow{Q_1, S_1} (q_2, s_1) \in A \parallel B \tag{12.9}$$
$$\text{if} a! \in S_1 \text{and} a?? \in Q_1 \tag{12.10}$$
$$\text{if} \#a! \in S_1 \text{and} \#a?? \in Q_1 \tag{12.11}$$
$$\text{then}(q_1, s_1) \xrightarrow{Q_1, S_1} (q_1, s_2) \in A \parallel B \tag{12.12}$$
$$\tag{12.13}$$

Suspended/Delayed write(self-loop)

$Rel_c(Q_1, S_1)$ is true, when one action is self-loop with non-blocking outgoing transtion and other is blocking.

$$\text{When} q_1 \xrightarrow{Q_1} q_1 \tag{12.14}$$
$$q_1 \xrightarrow{Q_2} q_2 \tag{12.15}$$
$$s_1 \xrightarrow{S_1} s_2 \tag{12.16}$$
$$\text{if} a_{suspend} \in Q_1, a! \in Q_2 \text{and} a? \in S_1 \tag{12.17}$$
$$\text{then}(q_1, s_1) \xrightarrow{Q_1} (q_1, s_1) \in A \parallel B \tag{12.18}$$
$$\text{then}(q_1, s_1) \xrightarrow{Q_2, S_1} (q_2, s_2) \in A \parallel B \tag{12.19}$$
$$\tag{12.20}$$

When one action is self-loop with non-blocking outgoing transtion and other is non-blocking.

$$\text{if} a_{suspend} \in Q_1, a! \in Q_2 \text{and} b! \in S_1 \tag{12.21}$$

$$\text{then}(q_1, s_1) \xrightarrow{Q_1} (q_1, s_1) \in A \parallel B \tag{12.22}$$

$$\text{then}(q_1, s_1) \xrightarrow{Q_1, S_1} (q_1, s_2) \in A \parallel B \tag{12.23}$$

$$\text{then}(q_1, s_2) \xrightarrow{Q_1} (q_1, s_2) \in A \parallel B \tag{12.24}$$

Instantaneous write and Instantaneous write

$Rel_d(Q_1, S_1)$ is true for sets of actions on control channels of Q_1, S_1, when both actions are non-blocking.

$$\text{if} a! \in Q_1 \text{then} b! \in S_1 \text{then}(q_1, s_1) \xrightarrow{Q_1, S_1} (q_2, s_2) \in A \parallel B \tag{12.25}$$

Instantaneous read and Instantaneous read

$Rel_e(Q_1, S_1)$ is false for sets of actions on control channels of Q_1, S_1,when both actions are blocking.

$$\text{if} a? \in Q_1 \text{then} b? \in S_1 \tag{12.26}$$

$$\text{then}(q_1, s_1) \xrightarrow{Q_1, S_1} (q_2, s_2) \notin A \parallel B \tag{12.27}$$

Delayed read and Delayed read

$Rel_f(Q_1, S_1)$ is true for sets of actions on control channels of Q_1, S_1,when both actions are blocking.

$$\text{if} a?? \in Q_1 \text{then} b?? \in S_1 \tag{12.28}$$

$$\text{then}(q_1, s_1) \xrightarrow{Q_1, S_1} (q_1, s_2) \in A \parallel B \tag{12.29}$$

$$\text{then}(q_1, s_1) \xrightarrow{Q_1, S_1} (q_2, s_1) \in A \parallel B \tag{12.30}$$

$$\text{then}(q_1, s_1) \xrightarrow{Q_1, S_1} (q_2, s_2) \in A \parallel B \tag{12.31}$$

Read Previous

$Rel_g(Q_1, S_1)$ is true for sets of actions on control channels of Q_1, S_1,when both actions are blocking.

$$\text{if}\$a \in Q_1 \text{then} a! \in S_1 \tag{12.32}$$

$$\text{then}(q_1, s_1) \xrightarrow{Q_1, S_1} (q_2, s_2) \in A \parallel B \tag{12.33}$$

$$\text{if}\$a \in Q_1 \text{then} \#a! \in S_1 \tag{12.34}$$

$$\text{then}(q_1, s_1) \xrightarrow{Q_1, S_1} (q_2, s_2) \in A \parallel B \tag{12.35}$$

$$\text{if}\$a \in Q_1 \text{then} a?? \in S_1 \tag{12.36}$$

$$\text{then}(q_1, s_1) \xrightarrow{Q_1, S_1} (q_2, s_1) \in A \parallel B \tag{12.37}$$

Using rules for message passing between automaton, the correct communication automaton can be obtained from the complete product. The rules are defined in the next subsection which can be used to extract the desired communication paths.

Algorithm 2 CompleteSynchronousProduct(A,B)

1: Input:Two FSMs.This is used if both FSMs are synchronous FSM
2: Output: $C = A \parallel B$, complete parallel composition of two synchronous FSMS with all the states and transitions from initial state.
3: $P_c = (q_i, s_j)$ // list of state in the complete product starting from initial states of A and B
4: **for all** states $(q_1, s_2) \in P_c$ **do**
5: **for all** transitions $q_1 \xrightarrow{Q_1} q_2 \in A$ **do**
6: **for all** transitions $s_1 \xrightarrow{S_1} s_2 \in B$ **do**
7: **if** $Rel_a(Q_1, S_1)$ **then**
8: CheckAdd$(P_c, (q_2, s_2))$
9: **else if** $Rel_b(Q_1, S_1)$ **then**
10: CheckAdd$(P_c, (q_2, s_1) \vee$ CheckAdd$(P_c, (q_1, s_2))$
11: **else if** $Rel_c(Q_1, Q_2, S_1)$ **then**
12: [CheckAdd$(P_c, (q_1, s_1) \wedge$ CheckAdd$(P_c, (q_2, s_2)] \bigvee$ [CheckAdd$(P_c, (q_1, s_2) \wedge$ CheckAdd$(P_c, (q_1, s_2)])$
13: **else if** $Rel_d(Q_1, S_1)$ **then**
14: CheckAdd$(P_c, (q_2, s_2))$
15: **else if** $Rel_e(Q_1, S_1)$ **then**
16: CheckRemove$(P_c, (q_2, s_2))$
17: **else if** $Rel_f(Q_1, S_1)$ **then**
18: CheckAdd$(P_c, (q_1, s_1) \wedge$ CheckAdd$(P_c, (q_1, s_2) \wedge$ CheckAdd$(P_c, (q_1, s_2))$
19: **end if**
20: **end for**
21: **end for**
22: **end for**

Algorithm 3 CheckAdd(P,s)

1: Input:FSM P, state s
2: Output: Add state s in P, if not present already
3: **if** $s \notin P$ **then**
4: Add (s) to P
5: **end if**

The above algorithms are employed on complete synchronous product and gross product to obtain desired communication subset.

12.1 Model of GALs interface

The following section shows the example of communication between devices operation on different clocks, that communicate through an asynchronous FIFO.

Algorithm 4 CheckRemove(P,s)

Input:FSM P, state s
Output: Remove state s from P, if present already
if $s \in P$ **then**
    Remove (s) to P
end if

Algorithm 5 $Rel_a(Q_1, S_1)$

Input:Q_1, S_1 actions on transition
Output: True or False
if $a! \in Q_1 \wedge a? \in S_1$ **then**
    CheckAdd$(P_c, (q_2, s_2))$
    return True
else if $\#a! \in Q_1 \wedge \#a? \in S_1$ **then**
    CheckAdd$(P_c, (q_2, s_2))$
    return True
else if $a! \in S_1 \wedge a? \in Q_1$ **then**
    CheckAdd$(P_c, (q_2, s_2))$
    return True
else if $\#a! \in S_1 \wedge \#a? \in Q_1$ **then**
    CheckAdd$(P_c, (q_2, s_2))$
    return True
else
    return False
end if

Algorithm 6 $Rel_b(Q_1, S_1)$

Input:Q_1, S_1 actions on transition
Output: True or False
if $a! \in Q_1 \wedge a?? \in S_1$ **then**
    CheckAdd$(P_c, (q_2, s_1))$
    return True
else if $\#a! \in Q_1 \wedge \#a?? \in S_1$ **then**
    CheckAdd$(P_c, (q_2, s_1))$
    return True
else if $a! \in S_1 \wedge a?? \in Q_1$ **then**
    CheckAdd$(P_c, (q_1, s_2))$
    return True
else if $\#a! \in S_1 \wedge \#a?? \in Q_1$ **then**
    CheckAdd$(P_c, (q_1, s_2))$
    return True
else
    return False
end if

Algorithm 7 $Rel_c(Q_1, Q_2, S_1)$

Input:Q_1, Q_2, S_1 actions on transition
Output: True or False
if $a_{suspend} \in Q_1 \wedge a! \in Q_2 \wedge a? \in S_1$ **then**
    CheckAdd$(P_c, (q_1, s_1))$
    CheckAdd$(P_c, (q_2, s_2))$
    return True
else if $a_{suspend} \in Q_1 \wedge a! \in Q_2 \wedge b! \in S_1$ **then**
    CheckAdd$(P_c, (q_1, s_1))$
    CheckAdd$(P_c, (q_1, s_2))$
    CheckAdd$(P_c, (q_2, s_1))$
    return True
else
    return False
end if

Algorithm 8 $Rel_d(Q_1, S_1)$

Input:Q_1, S_1 actions on transition
Output: True or False
if $a! \in Q_1 \wedge b! \in S_1$ **then**
    CheckAdd$(P_c, (q_2, s_2))$
    return True
else
    return False
end if

Algorithm 9 $Rel_e(Q_1, S_1)$

Input:Q_1, S_1 actions on transition
Output: True or False
if $a? \in Q_1 \wedge b? \in S_1$ **then**
    CheckRemove$(P_c, (q_2, s_2))$
    return True
else
    return False
end if

Algorithm 10 $Rel_f(Q_1, S_1)$

Input:Q_1, S_1 actions on transition
Output: True or False
if $a?? \in Q_1 \wedge b?? \in S_1$ **then**
    CheckAdd$(P_c, (q_1, s_2))$
    CheckAdd$(P_c, (q_2, s_1))$
    CheckAdd$(P_c, (q_2, s_2))$
    return True
else
    return False
end if

Algorithm 11 $Rel_g(Q_1, S_1)$

Input:Q_1, S_1 actions on transition
Output: True or False
if $\$a \in Q_1 \wedge a! \in S_1$ **then**
  CheckAdd$(P_c, (q_2, s_2))$
  return True
else if $\$a \in Q_1 \wedge \#a! \in S_1$ **then**
  CheckAdd$(P_c, (q_2, s_2))$
  return True
else if $\$a \in Q_1 \wedge a?? \in S_1$ **then**
  CheckAdd$(P_c, (q_2, s_1))$
  return True
else
  return False
end if

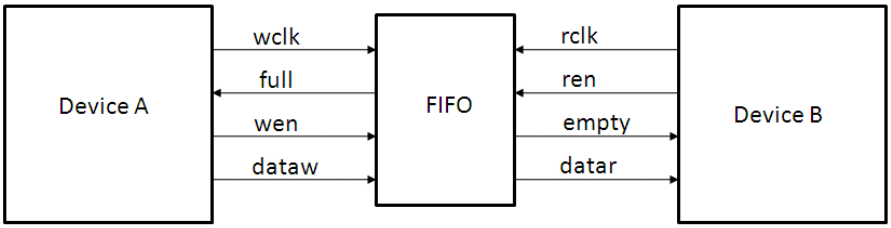
Algorithm 12 Instantaneous R/W rule

Input:P_c FSM
Method:Traverse all the paths, keep the transitions where $s!$ and $s?$ occur together. Removes ones with $s?$ without $s!$.
Output:FSM preserving states that follow the rule and eliminate states violating the ruke
if $Paths(P_c, q_0, q_f) \neq \phi$ **then**
  for all $(\pi \in Paths(P_c, q_0, q_f))$ **do**
    for all $s \in C_i$ **do**
      Writes(π, C_i) = Writepresences$(\pi, s! \in C_i) = i_1 < i_2 \cdots i_n$
      Reads(π, C_i) = Readpresences$(\pi, s? \in C_i) = j_1 < j_2 \cdots j_m$
      for all $len = 1 :: len \leq Reads[\pi, C_i]$ **do**
        if $Reads[len] = Writes[len]$ **then**
          Keep the transition
        else
          Remove transitions with standalone read instantaneous
          Keep transitions with standalone write instantaneous
        end if
      end for
    end for
  end for
end if

Algorithm 13 Suspended/Delayed Write rule

1: Input:P_c FSM
2: Method:Traverse all the paths, keep the transitions where $s_{suspend}$ and and preserve the self-loops when the outgoing actions are non-blocking writes to preserve delay.
3: Output:FSM preserving states that follow the rule and eliminate states violating the ruke
4: **if** $Paths(P_c, q_0, q_f) \neq \phi$ **then**
5: **for all** $(\pi \in Paths(P_c, q_0, q_f))$ **do**
6: **for all** $s \in C_i$ **do**
7: WriteSuspends(π, C_i) = Writepresences$(\pi, s_{suspend} \in C_i) = i_1 < i_2 \cdots i_n$
8: Writes(π, C_i) = Writepresences$(\pi, s! \in C_i) = j_1 < j_2 \cdots j_m$
9: **for all** $len = 1 :: len \leq Writes[\pi, C_i]$ **do**
10: **if** $WriteSuspends[len] = Writes[len] - 1$ **then**
11: Keep the self-loop
12: Keep the standalone non-blocking write
13: **end if**
14: **end for**
15: **end for**
16: **end for**
17: **end if**

Figure 12.1: GALS interface



13 Example

In this paper we consider modeling a GALS NOC called Asynchronous FIFO based NoC called ANOC proposed by F.Clermidy etal[31]. The ANOC interfaces to resources through a GALS interface. The GALS interface takes care of clock domain cross over using an asynchronous FIFO that can be written to and read with separate clocks.The clockless router interfaces to GALS interface on handshake lines. There are a number of handshake protocols proposed to ensure delay insensitivity on the communication line. The routers perform routing and scheduling of incoming messages using input and output controllers. We assume routers to contain a static routing table generated from a routing algorithm. The NOC performs priority based scheduling and the priority is encoded in the packets.

Algorithm 14 Delayed Read rule

1: Input:P_c FSM
2: Method:Traverse all the paths, check if there is a past or future write, with option for postponed read.
3: Output:FSM preserving states that follow the rule and eliminate states violating the ruke
4: **if** $Paths(P_c, q_0, q_f) \neq \phi$ **then**
5: **for all** $(\pi \in Paths(P_c, q_0, q_f))$ **do**
6: **for all** $s \in C_i$ **do**
7: Writepresent(π, C_i) = Writepresences$(\pi, s! \in C_i) = i_1 < i_2 \cdots i_n$
8: ReadDelayed(π, C_i) = Writepresences$(\pi, s?? \in C_i) = k_1 < k_2 \cdots k_l$
9: Writeabsent(π, C_i) = Writeabsences$(\pi, \#s! \in C_i) = j_1 < j_2 \cdots j_m$
10: **for all** $len = 1 :: len \leq ReadDelayed[\pi, C_i]$ **do**
11: **if** $(ReadDelayed[len] \geq Writepresent[len - 1]) \bigvee ((ReadDelayed[len] \geq Writeabsent[len-1]) \wedge (Writepresent[len-1] \geq Writeabsent[len-1]))$ **then**
12: Keep the transition
13: **else if** $(ReadDelayed[len] \geq Writepresent[len + 1])$ **then**
14: Keep the transition
15: **else if** postpone read **then**
16: Keep successive postpone read states of gross product, decides to make a transition
17: **else**
18: Remove transitions with delayed read
19: **end if**
20: **end for**
21: **end for**
22: **end for**
23: **end if**

Algorithm 15 Read Previous rule

1: Input:P_c FSM
2: Method:Traverse all the paths, update read previous register, keep the transitions where $\$s$ if read previous is true.
3: Output:FSM preserving states that follow the rule and eliminate states violating the ruke
4: **if** $Paths(P_c, q_0, q_f) \neq \phi$ **then**
5: **for all** $(\pi \in Paths(P_c, q_0, q_f))$ **do**
6: **for all** $s \in C_i$ **do**
7: Length of path $l =\mid \pi \mid$
8: **for all** $(r = 0; r \leq l)$ **do**
9: updatepresences$(\pi, C_i) = 1$ if present and 0 if absent
10: **end for**
11: Readpreviouses(π, C_i) = Readpresences$(\pi, \$s \in C_i) = j_1 < j_2 \cdots j_m$
12: **for all** $len = 1 :: len \leq Readpreviouses[\pi, C_i]$ **do**
13: **if** $updatepresences[Readpreviouses(len) - 1] = 1$ **then**
14: Keep the transition
15: **else**
16: Remove transitions with read previous
17: **end if**
18: **end for**
19: **end for**
20: **end for**
21: **end if**

13.1 Synchronous handshake Protocol

The synchronous handshake protocol specified for Asynchronous NoC [31] is based on virtual channel multiplexing. The handshake protocol with message sequence is shown in figure1.

The conditions for the sender to transmit new data on virtual channel are: presence of accept signal in the previous clock cycle and by asserting the send signal. Atmost one virtual channel can use the communication channel at a given time.

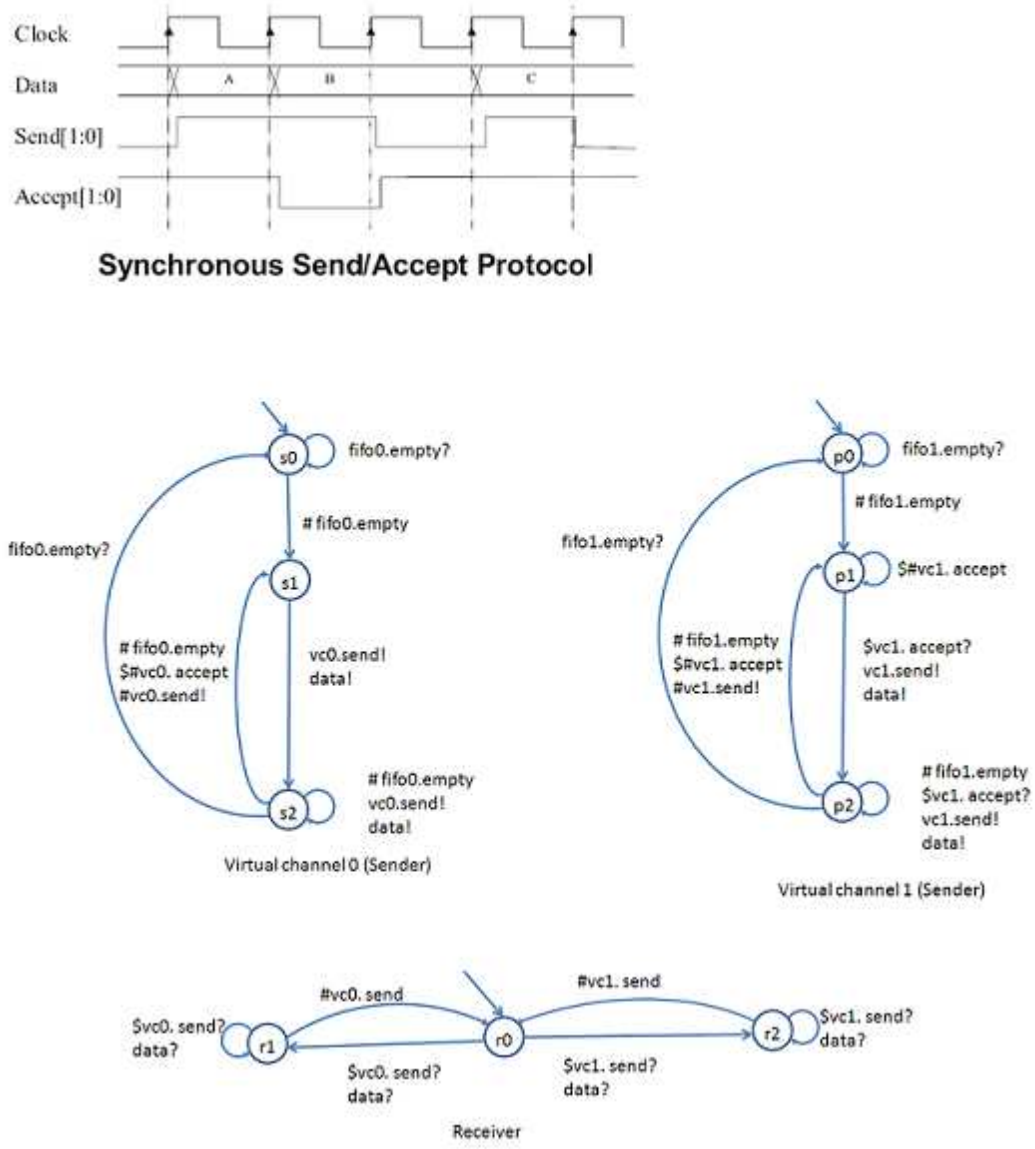
The parallel composition according to the given synchronous parallel composition rules are in Appendix 1.

13.2 Asynchronous handshake Protocol

The handshake communication protocol between the devices is shown. The data / send / accept signal is therefore implemented as an asynchronous handshake channel. The handshakes on the send and accept signals are sufficient to perform synchronization between the devices. The channel is idle when the accept signal is high. Since this is a 4-phase handshake protocol, the signal level of the handshake signals send and accept alternate 4 times before data is exchanged. The data transmitted on the asynchronous pipeline is modeled as communication action with send as pre-guard and accept as post-guard.

The automata of asynchronous protocols at the sender, receiver and the parallel composition using asynchronous parallel composition rules are given in

Figure 13.1: Synchronous Handshake Protocol



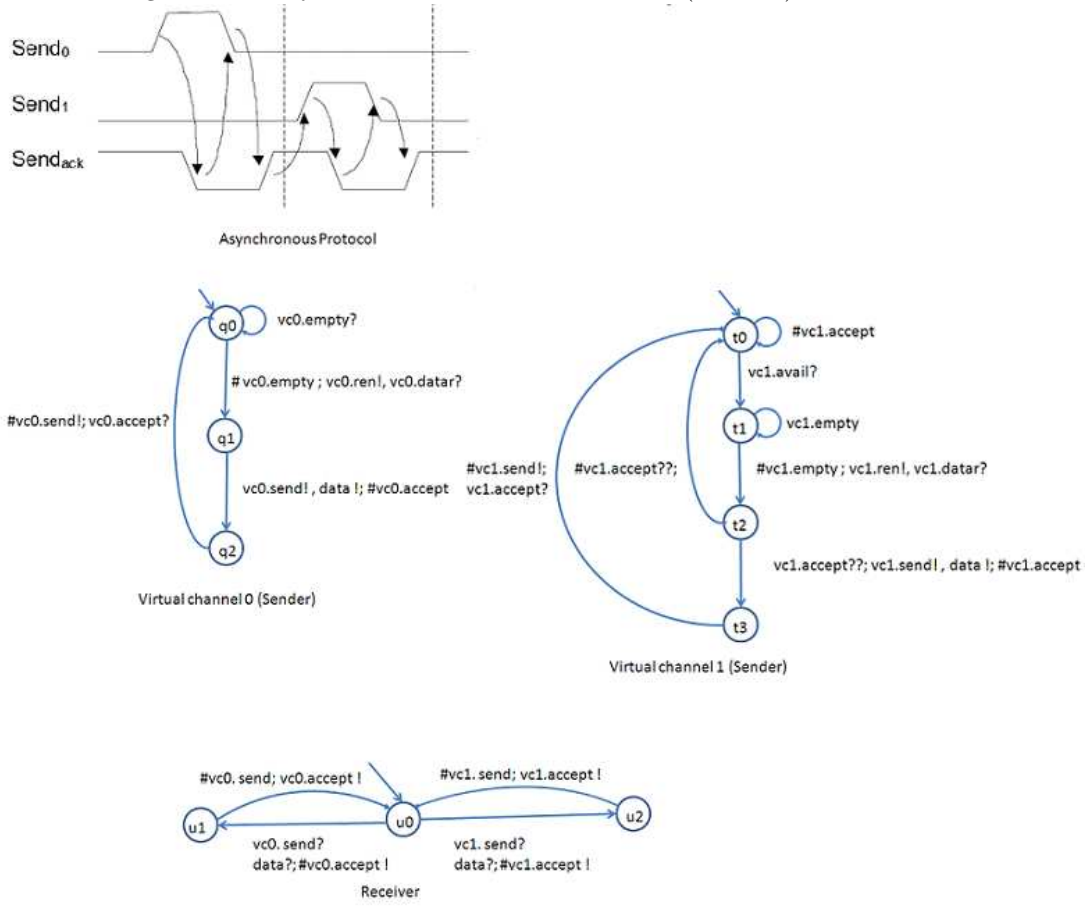
Appendix 2.

14 Verification

In the parallel composition of synchronous protocol between sender and receiver we perform verification of properties such as:

- Dead lock: Absence of states without next state

Figure 13.2: Asynchronous Handshake Protocol (4-Phase)



- Priority: high priority channels are not blocked by lower priority channels at a given time

The properties can be presented using temporal logic.

15 Road Map

- Tighten may_s and may_a rules
- Do more examples for async parellel composition as clocked with unclocked and asynchronous clocks
- Propose parallel composition rules for counters
- Verify properties on smaller models using existing model checking tool or develop model checking algorithms if the properties cannot be verified using existing tools

- Model and verify complete NoC including Switch

16 Conclusions

The interface can be interchangeably modeled using any type of communication methodology. Each modeling methodology denotes different kinds of formal methods. But using different formal methods to model requires methods of integrating these methods. But we have proved here that same formalism can have different formal semantics to model different communication interfaces.

Bibliography

[1] L. Benini and G. De Micheli. Networks on chips: a new soc paradigm. *Computer*, 35(1):70–78, Jan 2002.

[2] P. Wielage and K. Goossens. Networks on silicon: blessing or nightmare? *Digital System Design, 2002. Proceedings. Euromicro Symposium on*, pages 196–200, 2002.

[3] Hannu (Eds.) Jantsch, Axel; Tenhunen. *Networks on Chip*. Springer, 2003, VIII, 303 p., Hardcover.

[4] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, M.Millberg, and D. Lindqvist. Network on chip: An architecture for billion transistor era. *Proceeding of the IEEE NorChip Conference*, November 2000.

[5] Jörg Henkel, Wayne Wolf, and Srimat Chakradhar. On-chip networks: A scalable, communication-centric embedded system design paradigm. *IEEE Computer Society,VLSID '04: Proceedings of the 17th International Conference on VLSI Design*, page 845, 2004.

[6] William J. Dally and Brian Towles. Route packets, not wires: on-chip interconnection networks. *ACM,DAC '01: Proceedings of the 38th conference on Design automation*, pages 684–689, 2001.

[7] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. *ACM, DATE '00: Proceedings of the conference on Design, automation and test in Europe*, pages 250–256, 2000.

[8] K. Goossens J. Dielissen, A. Radulescu and E. Rijpkema. Concepts and implementation of the phillips network-on-chip. *In Proceedings of the IP based SOC (IPSOC)*, Nov 2003.

[9] L. Benini D. Bertozzi, G. Biccari, M. Dallosso, and L. Giovannini. xpipes: a latency insensitive parameterized network-on-chip architecture for multi-processor socs. *IEEE, ICCD 03: Proceedings of the 21st International Conference on Computer Design*, 2003.

[10] R. Thid M. Millberg, E. Nilsson and A. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip. *IEEE, DATE 04: Proceedings of the conference on Design, automation and test in Europe*, 2004.

[11] F. Moraes A. Mello, N. Calazans, L. Moller, and L. Ost. Hermes: an infrastructure for low area overhead packet-switching networks on chip. *VLSi Journal*, 38(1), 2004.

[12] J. Sparso T. Bjerregaard. A router architecture for connection-oriented service guarantees in the mango clockless network-on-chip. *IEEE, DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, 2005.

[13] F. Clermidy A. Clouard, E. Beigne, P. Vivet, and M. Renaudin. An asynchronous noc architecture providing low latency service and its multi-level design framework. *IEEE, ASYNC 05: Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 54–63, 2005.

[14] E. Friedman D. Rostislav, V. Vishnyakov and R. Ginosar. An asynchronous router for multiple service levels networks on chip. *IEEE, ASYNC '05: Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 44–53, 2005.

[15] J. Pontes, M. Moreira, R. Soares, and N. Calazans. Hermes-glp: A gals network on chip router with power control techniques. *Symposium on VLSI, 2008. ISVLSI '08. IEEE Computer Society Annual*, pages 347–352, April 2008.

[16] D.M. Chapiro. Globally asynchronous locally synchronous systems. *PhD Thesis,Stanford University*, Oct 1984.

[17] J. Sparsø. Future networks-on-chip; will they be synchronous or asynchronous? (invited talk). *SSoCC'04 (Swedish System on Chip Conference, Båstad)*, 2004.

[18] K Goossens. Formal methods for networks on chips. *IEEE, Application of Concurrency to System Design, ACSD 2005*, pages 188–189, 2005.

[19] M.D. Grammatikakis M. Coppola, S. Curaba, R. Locatelli, G. Maruccia, and F. Papariello. Occn: a noc modeling framework for design exploration. *Elsevier, J. Syst. Archit.*, 50(2-3):129–163, 2004.

[20] J. Joven D. Castells-Rufas and J. Carrabina. A validation and performance evaluation tool for protonoc. *System-on-Chip, 2006. International Symposium on*, (1-4), 2006.

[21] F.W. Vaandrager B. Gebremichael and M. Zhang. Formal models of guaranteed and best-effort services for networks on chip. *Technical Report,ICIS-R05016,Radboud University Nijmegen*, March 2005.

[22] M. Zhang B. Gebremichael, F. Vaandrager, K. Goossens, E. Rijpkema, and A. Radulescu. *Deadlock Prevention in the thereal Protocol.* Correct Hardware Design and Verification Methods,Springer, Book Chapter, 2005.

[23] J. Schmaltz G.A. Sammane and D. Borrione. Formal design and verification of on chip networking. *IEEE, Information and Communication Technologies: From Theory to Applications*, pages 657–658, March 2004.

[24] J. Schmaltz and D.Borrione. A generic network on chip model. *LNCS, Book TPHOLs*, 3603:310–325, 2005.

[25] L.Pierre D.Borrione, A.Helmy. Acl2-based verification of the communications in the hermes network on chip. *Proc. International Workshop on Symbolic Methods and Applications to Circuit Design (SMACD'06)*, 2006.

[26] R. Shankar A. Agarwal. Modeling concurrency in noc for embedded systems. *Proc. High Performance Embedded Computing,Massachusetts Institute of Technology*, 2006.

[27] S. Kumar R. Holsmark, M. Hgberg. Modeling and evaluation of a network on chip architecture using sdl. *LNCS,11th International SDL Forum, Stuttgart, Germany*, 2708, July 2003.

[28] O. Bringmann A. Siebenborn and W. Rosenstiel. Communication analysis for network-on-chip design. *IEEE,PARELEC '04: Proceedings of the international conference on Parallel Computing in Electrical Engineering*, pages 315–320, 2004.

[29] L. Tsiopoulos and M. Waldn. Formal development of noc systems in b. *Nordic Journal of Computing*, 13(1-2):127 – 145, 2006.

[30] Y. Thonnart C. Koch-Hofer, M. Renaudin and P. Vivet. Asc, a systemc extension for modeling asynchronous systems, and its application to an asynchronous noc. *Networks-on-Chip, 2007. NOCS 2007. First International Symposium on*, pages 295–306, May 2007.

[31] Y. Thonnart G. Salaun, W. Serwe and P. Vivet. Formal verification of chp specifications with cadp illustration on an asynchronous network-on-chip. *IEEE,ASYNC '07: Proceedings of the 13th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 73–82, 2007.

[32] I. Sander Z. Lu and A. Jantsch. Towards performance-oriented pattern-based refinement of synchronous models onto noc communication. *IEEE,DSD '06: Proceedings of the 9th EUROMICRO Conference on Digital System Design*, pages 37–44, 2006.

[33] K. Goossens A. Rădulescu. Communication services for networks on chip. *In Domain-Specific Processors: Systems, Architectures, Modeling, and Simulation, Book Chapter*, pages 193–231, 2004.

17 Appendix 1

18 Appendix 2

Figure 17.1: Synchronous Parallel Composition

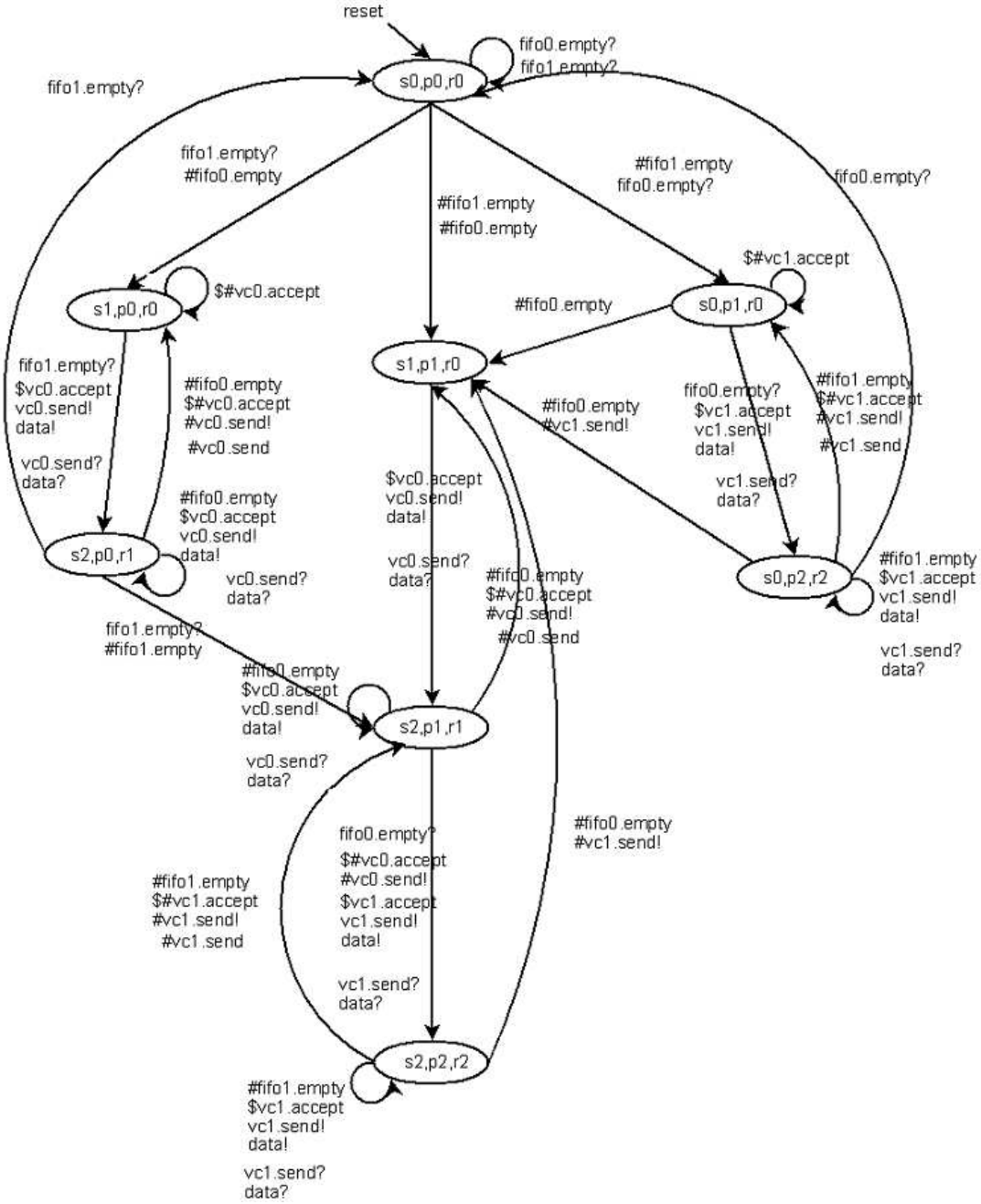


Figure 18.1: Asynchronous Parallel Composition