LOTOS Code Generation for Model Checking of STBus Based SoC : the STBus interconnect

Pierre Wodey, Geoffrey Camarroque, Fabrice Baray
ISIMA - LIMOS
BP 10125
F-63173 AUBIERE Cedex - FRANCE
{Pierre.Wodey, Geoffrey.Camarroque,Fabrice.Baray}@isima.fr

Richard Hersemeule, Jean-Philippe Cousin
Advance Systems Technology-AST
STMicrolectronics
12, Rue Jules Horowitz
F-38000 GRENOBLE - FRANCE
{Richard.Hersemeule,Jean-Philippe.Cousin}@st.com

Abstract

In the design process of SoC (System on Chip), validation is one of the most critical and costly activity. The main problem for industrial companies like STMicroelectronics, stands in validation at the complete system level. At this level, the properties to verify concern the well behaviour composed of the different processes interconnected around the system bus. In our work we consider the deadlock-free property. In this paper we present an approach for deadlock detection consisting in generating automatically a LOTOS description of the system. Then, by using CADP toolbox developed at INRIA by the VASY team, the LOTOS description can then be used for the evaluation of temporal logic formulæ, either on-the-fly or after the generation of a Labelled Transition System (LTS). The automatic LOTOS code generation is decomposed in two parts, the code generation of the processes behaviour (work under progress) and the code generation for the interconnection of processes on a given SoC bus. This paper presents the principles of interconnect abstraction showing that deadlock detection has to take into account properties of the implemented communication channel, avoiding the possibility to build a general deadlock detection tool. The resulting principles are then applied on the STMicroelectronics proprietary SoC bus, the STBus, leading in the development of the LOTOS code generation software.

1. Introduction

This paper presents the work conduced by the SEMANTSYS (for SYStems SEMANTics) team at the LIMOS laboratory, in collaboration with STMicroelectronics. The aim is to be able to check automatically some properties of SoC at the global system level. Let us first introduce the industrial SoC design and validation problems and then our approach for validation, presented in this paper.

1.1 SoC design and validation in industry

For several years the trend in microelectronics area is to integrate more and more functionalities. This larger integration is mainly due to technology improvement. With a $0.13\mu m$ technology we may foresee 100 million transistors and hundreds of thousands of code lines embedded on the same chip.

If the design paradigms have changed for some years, the architecture of new systems on chip (SoC) evolves also significantly.

Most system architectures are now built around a system bus, several MCU and DSP and complex IP (Intellectual Property) blocks providing a high level of communication and programmability. Consequently the number of transactions between blocks is growing.

EDA tools are efficient enough to provide sufficient productivity in term of gates and software. Nevertheless, the main emerging problem stands in the lack of efficient verification methods. We estimate that 60% of the time is dedicated to functional verification of IP and the whole system level where the most events are asynchronous. Whereas there exist satisfying verification solutions for IP blocks (synchronous), at the present time, there do not exist acceptable solutions for the whole system. In order to deal with larger complexity, the verification strategy consists, rather than undertaking a verification process very late in the design (RTL level), in the introduction of verification at the same time we start the design (HW and SW). The complexity of the communication engine on a SoC is probably the area where most bugs occur.

That is the reason why we need to turn our attention to protocols and deadlock-free communications validation.

Verification and validation also include an economic aspect. We have to take into account that a mask on a $0.13\mu m$ costs around 500 K$. Thus, it is necessary to reduce potential errors, using even more sophisticated technologies, as, for instance, the automatic model checking of the whole system whose first step study is presented in this paper.

1.2 Our approach for SoC model checking

Since 1997, SEMANTSYS team studies the problems of introducing the verification by model checking in Codesign methodologies at the system level [11, 2, 1]. These studies have been lead on general Codesign approaches consisting in architecture exploration and communication synthesis [6].

The main problem pointed out stands in the communication architecture and protocol, whose characteristics highly influence the global system behaviour. The aim of our work consists in generating automatically the verification model of the system and also the temporal logic formulæ to validate the deadlock-free property.

For properties verification, we use the CADP toolbox based on OPEN-CÆSAR [4], developed at INRIA Rhône-Alpes, by the VASY team. This toolbox is based on the notion of Labelled Transition System (LTS) [8], describing the behaviour of the system by means of states and transitions, a transition being labelled by the name of the action performed by the system.

In order to generate the LTS of a system, we will generate at first a LOTOS [7] description of the system. This language being one of the entries of the CADP toolbox and its semantic being clearly and well defined by means of an LTS.

The LOTOS description is generated from the design descriptions of the system, the aim being to use LOTOS as an intermediate format not seen by the designer.

The global structure of the LOTOS code generator is decomposed in two parts :

- the code generator at the top level, instantiating processes and synchronizing them or not, according to a given STBus architecture. This is the part described in this paper ;

- the code generation for the processes behaviour, from a given high level description (existing in the design environment) such as a functional model or, better, a transactional level model (TLM) as, for instance, defined in SystemC [5]. The study of this part is under way and some corresponding perspectives described in the conclusion.

Considering the LOTOS code generation of an interconnect of processes, it is necessary to take into account some properties of the implemented communication channel : process blocking or synchronization conditions, atomicity of bus transactions, etc... This is illustrated in section 2 on a simple example independent of STBus. This section presents different ways of modeling manually a system and gives the main properties to be preserved in a LOTOS model. One result of this study is that deadlock detection is dependent from the implemented communication channel, and thus avoids to intend to build a more general deadlock detection tool. These principles have then to be applied automatically on an STBus interconnect. Section 2 contains also a short introduction to LOTOS.

Section 3 presents a brief overview of STBus features.

Section 4 and 5 describe the LOTOS code generation for an STBus interconnect, with respect to the principle of section 2. Section 4 presents how to abstract the bus arbitration by considering a single node. Then general STBus functionalities are taken into account in section 5.

2. System level abstraction for model checking

In order to tackle the complexity at the system level, a maximal abstraction of the behaviour is needed. Generally, abstraction is made by the use of a specification language which can be a Formal Description Technique (FDT) such as SDL [10] or LOTOS. But, according to the aim of deadlock detection, the abstract model of the system has to preserve the interesting properties of the real system behaviour. These properties concern :

- the communication channels properties : blocking conditions, atomicity of communication operations and memorization in the channel and access policy of memorized values.

- the asynchronism and non-determinism among processes or tasks due to : the scheduling strategy of operating systems managing software tasks on a given microprocessor and non deterministic interrupt arrivals from the outside world of the system, non constant computing time of processes when data dependant.

In this section we show a simple example of three processes communicating and the problems of abstract modeling. At first, we present the example and the semantic of the implemented system where a deadlock is present. Then we show a SDL and a LOTOS specification of the system and their associated LTS demonstrating that these descriptions do not preserve the right properties of the system to detect the deadlock. Finally, we present a Transaction Level LOTOS Model of the system whose associated LTS is equivalent to the implemented system one.

2.1 Implemented system description

We consider a simple system composed of three processes named A, B and C. Each process has an entry queue with infinite capacity, these are named Q_A, Q_B and Q_C. These processes exchange messages composed by the identification of the sender process id_A, id_B, id_C. The communication operations are : `write(queue,id)`, sends a message with identification id to the queue ; `read(id)`, the process reads from its queue the first message if the sender was the process with identification id.

The properties of the communication channel are :

- non blocking write,
- blocking read when the entry queue is empty,
- blocking read if the head of the queue has not the right identification (this will lead to deadlock),
- a single communication operation, `read` or `write`, is atomic, which is not the case of any succession of operations.

The behaviour of the processes is the following :

```
Process A :           Process B :          Process C :
 while true {          while true{          while true{
  write(QB, idA);       read(idA);           read(idB);
  read(idC);            write(QC,idB);       write(QA,idC);
 }                      read(idC);           write(QB,idC);
end Process            }                    }
                      end Process          end Process
```

The behaviour of this system is as follows :

- sequentially : A writes to the queue of B, then B reads from its queue, then B writes to the queue of C, then C reads from its queue, then C writes in the queue of A.
- then a non deterministic behaviour : either C writes into the queue of B before A reads and writes into the queue of B, the system continues on step (1), or A reads from its queue and write into the queue of B before C writes into the queue of B, the system falling in the deadlock state.

The LTS of the implemented system is shown in figure 1 in which the deadlock state is state 11 (state without outgoing transition).

2.2 Abstract SDL model

When considering an abstract model of this system, one can choose to build up a model in SDL. This model is given by figure 2.

According to the SDL semantic, a transition between two states, which corresponds to the reception, computing and

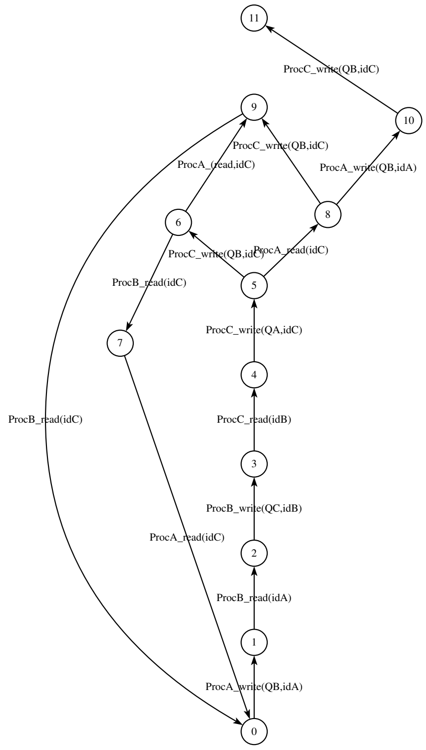


Figure 1. LTS for the implemented system

sending of messages, is atomic in the global system behaviour. This means that, for instance, in process C, the three operations (one read and the two writes) constitute an unique atomic action in the system behaviour. Thus, as shown in the corresponding LTS (see figure 3), there is no deadlock in the system with SDL semantic. This model shows the inadequation of an SDL model due to the non respect of atomicity properties in the channels.

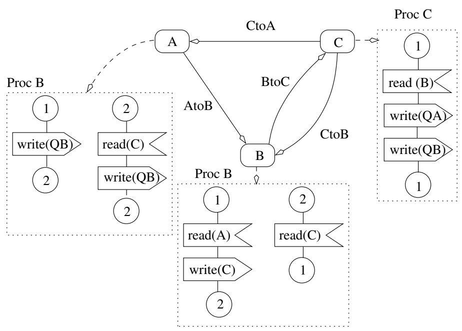


Figure 2. Processes SDL behaviour

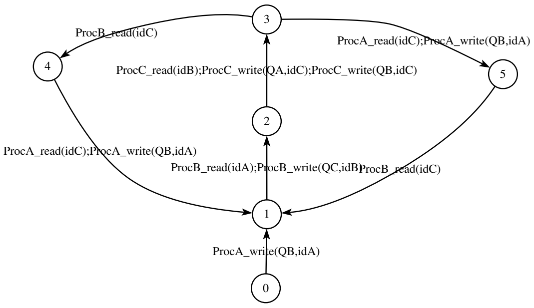


Figure 3. System LTS with SDL semantic

2.3 Abstract LOTOS model with rendez-vous

Let us illustrate another problem when considering an abstract LOTOS description in which the queues are no more described and the communication directly modeled by rendez-vous. The LOTOS model is described in figure 4, at the top level three gates are used to synchronize the processes in pair, and the interconnect is shown in figure 5.

Let us shortly introduce LOTOS operators on behaviours `B1 op B2`: `||` the two behaviours are synchronized on all common gates, `|||` the interleaving operator (no synchronizations), `|[G1,..,Gn]|`, the two behaviour are synchronized on the listed gates and `[]` is the nondeterministic choice between `B1` and `B2`. A gate operation can be completed by an offer which is a list of value either forced by `!value` or free `? variable:type`. The synchronization between two behaviours on a gate is performed if the forced values are the same on the two offers or values and free offer type are compatible. The offers are used to transfert data but can also be used to route the information by mean of source and destination identifiers defined by forced value offers. A LOTOS expression can also instantiate (call) a process which has some gates and some parameters. In a process, a recursive terminal call (not followed by any action) consists in iterating the process from the start with eventually new value parameters.

```
Process A [GAB,GCA]:noexit :=
   GAB!idA;GCA!idC;A[GAB,GCA]
endproc

Process B [GAB,GBC]:noexit :=
   GAB!idA;GBC!idB;GBC!idC;B[GAB,GBC]
endproc

Process C [GCA,GBC]:noexit :=
   GBC!idB;GCA!idC;GBC!idC;B[GCA,GBC]
endproc
```

Figure 4. Processes abstract LOTOS model

The semantic of LOTOS rendez-vous between processes induces that :

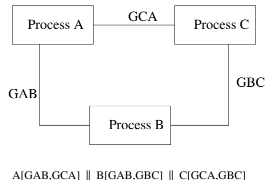


Figure 5. LOTOS processes interconnection

- a communication is done in one atomic action (the write and corresponding read are done atomically);
- the queue QB is replaced by two distinct gates GAB and GBC, so process B when refusing the synchronization with A is still able to accept the synchronization with C.

This induces that the global system becomes sequential and any interleaving of actions is avoided. The result is also that no deadlock is detectable with this semantic, as shown in figure 6.

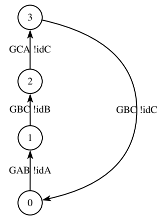


Figure 6. Abstract LOTOS model LTS

2.4 TLM LOTOS model

The LOTOS model presented in the previous section does not respect the communication semantic of the implemented system. The aim of this section is build out a LOTOS model with the same communication semantic, this corresponds to a TLM LOTOS model of the system.

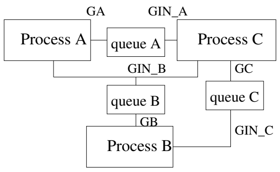
We consider thus a LOTOS model where the queues are instantiated and described by LOTOS processes. When writing into a queue a process perfoms a rendez-vous with FIFO process. Symetrically a process reads from its queue by means of a rendez-vous.

The code of a queue process is shown in figure 7, the data type being ommitted (`consend`, `car`, `cdr` and `nil` having a common meaning as in LISP for instance). The process has one input gate, one output gate and the memorized messages implemented in a parameter named file (built up as a list), initialized to nil.

The LOTOS specification is shown graphically in figure 8, also including the LOTOS expression. In this version, the processes C has three gates.

```
Process queue [GIN,GOUT](file:list):noexit :=
   (GIN?id:proc_id;queue[GIN,GOUT](consend(id,file)))
  []
   [not (list eq nil)]->
      (GOUT!car(list);queue[GIN,GOUT](cdr(list)))
endproc
```

Figure 7. Queue LOTOS model



(A[GIN_B,GA] ||| B[GB,GIN_C] ||| C[GC,GIN_A,GIN_B])
||
(queue[GIN_B,GA](nil) ||| queue[GIN_B,GB](nil) ||| queue[GIN_C,GC](nil))

Figure 8. LOTOS specification with queues

The resulting LTS is not shown as it is the same as the LTS of the implemented system shown in figure 1, modulo the labels names.

2.5 Conclusion on abstraction

The different exposed models of the system illustrate that, for deadlock detection, it is necessary to preserve the right channel properties and processes asynchronism. This illustrates also that LOTOS can be used to model a system interconnect with the right channel properties. The right LOTOS model is a Transaction Level Model of the system.

3 STBus features overview

This section presents an overview of STBus features which is illustrated by an example of Set Top Box SoC. These features have to be taken into account in the LOTOS code generator in order to build up a LOTOS interconnect generator including the well defined abstractions, in the same way as they where described in section 2.

STBus is an STMicroelectronics proprietary On Chip Bus protocol. It is dedicated to high bandwidth systems on chip for applications such as audio/video processing. The STBus interfaces and protocols are closely related to the industry standard developed by the open VCI (Virtual Component Interface) body defined by VSIA. A STBus interconnects components which can be either an initiator (when initiating transactions on the bus by sending requests) or a target (when responding to the requests). The bus architecture is decomposed in nodes (sub-bus in which initiators and targets can directly communicate), internode communications being performed through buffers (FIFOs).

3.1 Protocols

“Peripheral” protocol (STBus Type 1) is equivalent to the Peripheral VCI protocol. The peripheral or Type 1 STBus interface is the simplest of the STBus family and is targeted at modules which require a low complexity medium data rate communications path with the rest of the system. This typically includes standalone modules such as general-purpose input/output or modules which require independent control interfaces in addition to their main memory interface.

”Basic” protocol (STBus Type 2) is equivalent to the Basic VCI protocol. The Type 2 interface increases the performance and functionality of the STBus port, it supports all type 1 functionalities and adds split transactions and the ability to support all transactions including compound operations, source labeling and some priority and transaction labeling. It is aimed at devices which need high performances but do not require the additional system efficiency associated with shaped request/response packets or the ability to re-order outstanding operations to improve performance.

”Advanced” protocol (STBus Type 3) is equivalent to the Advanced VCI protocol. The Type 3 supports all type 1 and 2 functionalities, while enabling increasing significantly system/interface performances. This is enabled through support of a more efficient packet protocol (the number of cells might differ between request and response), and allowing the system to reorder operations. It allows performance improvements either by allowing more operations to occur concurrently, or by rescheduling the operations more efficiently.

3.2 Components

Associated with those protocols, hardware components have been designed in order to build complete reconfigurable interconnections between Initiators and Targets. A toolkit has been developed around this STBus (graphical interface) to generate automatically Top level backbone, Cycle accurate High level model generation, way to implementation, bus analysis (latencies, bandwidth), bus verification (protocol and behavior).

An STBus system includes three generic components as skeletons :

- Switch or node : it arbitrates and routes the requests and optionally, the responses. It can be crossbar, to full cross-bar, implement filtering features, initiator/target priority management...
- Converter or bridge domain : it converts the request from a protocol to another (”basic” to ”advanced”). Each component is configurable to implement specific features : bus size, ordering support...

- Size Converter: it is used between two buses of same type but of different widths. It includes buffering capability.

3.3 STBus interconnection example

Definition of STBus interconnect must take non trivial criteria such as performance constraints, initiators process constraints, type and size constraints, floorplan constraints... Figure 9 is an example of a complete System On Chip designed for Set Top Box application. It is composed of different nodes and buses interconnected by means of FIFOs or type converters.

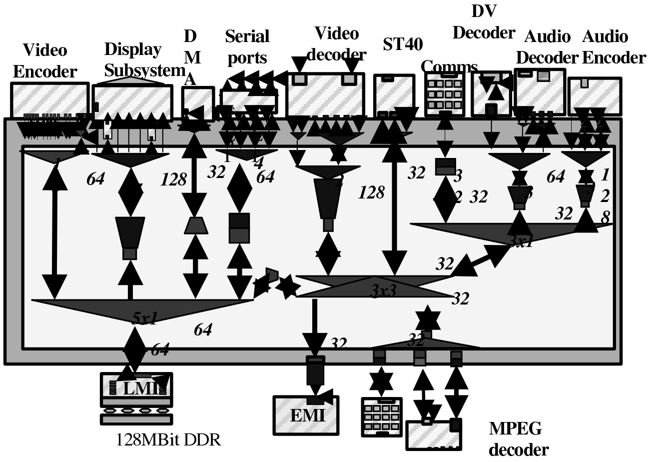


Figure 9. Set Top Box STBus based SoC

Interconnect clearly appears to be one of the critical part of the SoC, this bottleneck will certainly increase in next systems on-chip generation. Many efforts have been done to improve on chip bus architectural exploration, design process and verification around those fields. Verification challenges for complex On Chip Bus such as VCI based STBus will require more efficient methodologies than simulation. This is why in next few years formal methods should be used intensively.

4 TLM LOTOS model for single node system

Let us at first explain that for the LOTOS model of a STBus interconnect, we concentrate, at this point of our study, on the synchronizations and routing of messages. We will thus not consider data messages or operations, but just source (initiator) identifier and destination (target identifier). Thus all communications on the STBus are gate action with offer composed of four field `GATE !sN!I!dN!T` being respectively the source node number, the initiator number, the destination node number and the target number. One can notice that in type 3 protocol, there is one additional field named `tid`, the packet identification number.

We consider a single node system constituted by a n_I initiators and n_T targets, all of type 2 protocol. We describe the different abstractions made on the arbitration strategies and on the communication phases (request/grant).

4.1 Arbitration strategy abstraction

The STBus can implement various strategies of arbitration and allows to change them dynamically. Arbitration is used when multiple initiators (resp. targets) simultaneously request (resp. response request) access on the bus in order to choose one of them by granting it.

One can notice that when a single process requests access on the bus, it is granted independently from the arbitration strategy.

When considering the asynchronism among processes, they may, non-deterministically request acces on the bus either simultaneously (case 1), or in any given order (case 2). When requesting simultaneously the arbitration strategy will determine one order, which is included in the orders of case 2.

Consequently, the generated LOTOS model will not implement the arbitration strategy, it just implements the non deterministic choice of granting one of the requesting processes. The corresponding LTS will thus cover all the orders of request arrivals and also be arbitration-strategy independent for the simultaneous requests.

4.2 Single node communications

In a node, a communication between one initiator and a target is performed in several steps :

1. a request/grant step between the initiator and the node, corresponding to an atomic rendez-vous operation of the system,

2. the transfer by the node of the request to the target,

3. a response-request/grant between the target and the node,

4. the transfer by the node of the response-request to the initiator.

To generate the LOTOS model we have studied two possibilities, the first named channel-oriented in which a channel process is modeled and the second named system-oriented. We compare these two approaches by means of an example.

4.3 Channel-oriented model

In this model the four communication steps are generated as four atomic operations between the initiators/targets and two processes named respectively `ARB` (for abstract ARBiter) for the requests and `r_ARB` for the response-requests.

Figure 10 shows the structure of the LOTOS specification, where processes are interconnected by means of four gates, each dedicated to a given communication step.

All initiators and targets are interleaved, there is no direct synchronization, but each is synchronized with the two abstract arbiters (interleaved) on their common gates. The LOTOS description is given by the following LOTOS expression :

```
(    INI_1[I_req,I_r_req] ||| INI_2[I_req,I_r_req]
 ||| INI_3[I_req,I_r_req] ||| TAR_1[I_req,T_r_req]
 ||| TAR_2[T_req,T_r_req] ||| TAR_3[T_req,T_r_req]
)
||
(ARB[I_req, T_req] ||| r_ARB[I_r_req,T_r_req])
```

The behaviour of an abstract arbiter, a request arbiter for instance, consists in accepting a request from a given initiator and transferring it to the corresponding target before accepting another request.

We have generated the LTS in which any initiator can comunicate with any target, the internal behaviour consisting in waiting for the response before sending another request. The associated LTS, which is not shown, is composed of 151 states and 328 transitions.

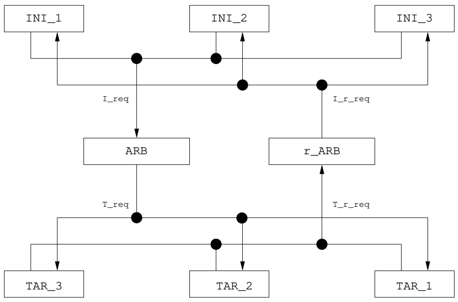


Figure 10. Channel-oriented model of example node

4.4 System-oriented model

In the system oriented model we consider direct rendezvous between a given initiator and a target. As an arbiter (for request or response) transfers sequentially the information before accepting another request, the two steps are atomic in the arbiter point of view. It is also the same when considering an initiator or a target.

In this point of view, the abstract arbiter processes are omitted, as shown in figure 11.

The initiators are interleaved and this behaviour synchronized with the interleaved targets, as described by the following LOTOS expression :

```
(    INI_1[I_req,I_r_req] ||| INI_2[I_req,I_r_req]
 ||| INI_3[I_req,I_r_req])
 ||
(    TAR_1[I_req,T_r_req]  ||| TAR_2[T_req,T_r_req]
 ||| TAR_3[T_req,T_r_req]
)
```

With the same initiator and target processes as for the channel oriented model, the corresponding LTS is composed of 15 states and 46 transitions.

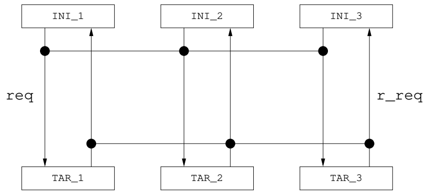


Figure 11. System oriented model of example node

4.5 Conclusion on the single node model

The results of the two approaches indicate that the LTS obtained by the system-oriented model is about 10 times smaller than that of the channel-oriented model.

The question is about the semantic of the two models regarding the implementation of the system. The difference between these two versions stands in the fact that a request and a response can both transit on the bus. As simultaneous actions are interpreted as non deterministic sequential actions in the LOTOS semantic, we have the two following behaviours : in the *channel-oriented* model, there is an interleaving between two sequences of two actions (thus three states), producing a part of LTS composed of 9 states ; in the *system-oriented* model, the interleaving is between two sequences of one action (thus two states), producing a part of LTS composed of 4 states.

In the system oriented model, the bus internal actions are omitted, and the four states describe the four possible states of the interconnected processes. In the channel oriented model, some states correspond to internal bus states, real processes states are those following two request steps or two response steps. In the set of 9 states, only four states have this property.

So when considering the system validation, rather than bus validation, we have to consider the system oriented model which describes all the interesting processes states and system executions.

5 LOTOS TLM model for general STBus

Let us consider a STBus interconnect, composed of :

1. n nodes, named $N_1, \cdots, N_n$,
2. each node N_i is composed of nI_i initiators, named $I_{i,1}, \cdots, I_{i,nI_i}$, and nT_i targets, named $T_{i,1}, \cdots, T_{i,nT_i}$,
3. inter-node communication links, represented by the set of pairs $FT \subseteq \{1, \cdots, n\}^2$, such that $(i,j) \in FT$ means there is a communication from node N_i to node N_j
4. for each inter-node communication $ft = (i,j) \in FT$, let $s(ft)$ be the size of the internode FIFO,
5. for each node N_i, we consider the destination nodes set $DN(i) = \{j \in \{1, \cdots, n\} / (i,j) \in FT\}$ and source nodes set $SN(i) = \{j \in \{1, \cdots, n\} / (j,i) \in FT\}$

At first we consider the basic STBus interconnect. The size converters and protocol type converters will be considered later.

5.1 Basic STBus interconnect

The LOTOS model of a given STBus interconnect instantiates processes corresponding to initiators, targets and internode FIFOs. The LOTOS expression describes the synchronizations on gates between these processes.

In a given node, the initiators are interleaved together, and are synchronized with the targets (interleaved on their side). We name `REQ_i` the LOTOS request gate of node N_i, and `R_REQ_i` the response request gate of node N_i.

The LOTOS behaviour of the internode FIFO is always the same, only the size of the FIFO is parameterized at the instantiation. An internode FIFO from node N_i to node N_j is synchronized with the initiators of node N_i and the targets of node N_j for the requests and for the responses. It is seen as a target for node N_i and an initiator for node N_j. An internode FIFO is composed of two internal FIFOs of same size, one for the requests and one for the responses.

As the FIFOs are synchronized separately on the two nodes, the global topology is, in the general case, not linear and cannot be expressed in LOTOS by using the same request and response gates as those used between initiators and targets in a node (i.e. `REQ_i` and `R_REQ_j`...). Thus it is necessary to define a specific gate for each destination and source node. We name `FIFO_i_j`, the FIFO for the internode communication from node N_i to node N_j. Such a FIFO has four gates :

- `S_REQ` (request) and `S_R_REQ` (response) for the synchronizations with the initiators of node source (N_i in the example),
- `D_REQ` (request) and `D_R_REQ` (response) for the synchronizations with the targets of destination node (N_j in the example).

Each node N_i has following LOTOS gates :

- `REQ_i_i_j` and `R_REQ_i_i_j` for each $j \in DN(i)$, being the request and response gates in node N_i (connected to the initiators of N_i) for the communications from node N_i to N_j and to `S_REQ` and `S_R_REQ` of the buffer `FIFO_i_j`,
- `REQ_i_j_i` and `R_REQ_i_j_i` for each $j \in SN(i)$, being the request and response gates in N_i (connected to the targets of node N_i) for the communications from node N_j to N_i and to `D_REQ` and `D_R_REQ` of the buffer `FIFO_j_i`.

Figure 12 illustrates the case of a system composed of two nodes N_1 and N_2, each constituted by two initiators and two targets, and internode communication from N_1 to N_2 and from N_2 to N_1.

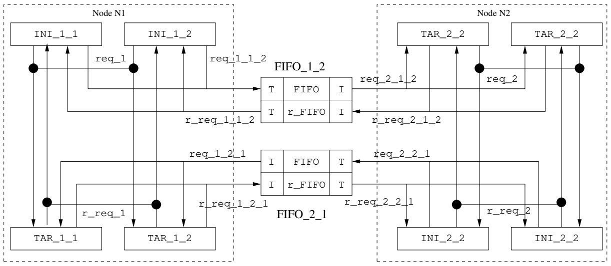


Figure 12. Two nodes LOTOS model

The interface of the processes of node N_i are :

- the initiators have gates `REQ_i`, `R_REQ_i` and $\forall j \in DN(i)$, the gates `REQ_i_i_j` and `R_REQ_i_i_j`, this list of gates is called `GI(Ni)`, for Initiator gates of node N_i,
- the targets have the gates `REQ_i`, `R_REQ_i` and $\forall j \in SN(i)$, the gate `REQ_i_j_i` and `R_REQ_i_j_i`, this list of gates is called `GT(Ni)`, for target gates of node N_i.

Figure 13 shows the LOTOS description of a general STBus interconnect.

5.2 Protocol types and bus sizes

In the previous section we did not consider the protocol types and bus sizes. The protocol types of processes and the

```
/* nodes : initiators and targets */
(
  /* Node N1 */
  ( (I_1_1[GI(N1)] ||| ... ||| I_1_nI1[GI(N1)])
   |[REQ_1, R_REQ_1]|
    (T_1_1[GT(N1)] ||| ... ||| T_1_nT1[GT(N1)]))
  ||| ...  |||
  /* Node Ni */
  ( (I_i_1[GI(Ni)] ||| ... ||| I_i_nIi[GI(Ni)])
   |[REQ_i, R_REQ_i]|
    (T_i_1[GT(Ni)] ||| ... ||| T_i_nTi[GT(Ni)]))
  ||| ...|||
  /* Node Nn */
  ( (I_n_1[GI(Nn)] ||| ... ||| I_n_nIi[GI(Nn)])
   |[REQ_n, R_REQ_n]|
    (T_n_1[GT(Nn)] ||| ... ||| T_i_nTn[GT(Nn)]) )
)
/* synchonized on all common gates with internode FIFOs */
||
(/* for all (i,j) in TF = {(i1,j1), ..., (ik,jk)} */
  FIFO_i1_j1[REQ_i1_i1_j1, R_REQ_i1_i1_j1,
        REQ_j1_i1_j1, R_REQ_j1_i1_j1](s(i1,j1),nil, nil)
||| ...|||
  FIFO_ik_jk[REQ_ik_ik_jk, R_REQ_ik_ik_jk,
        REQ_jk_ik_jk, R_REQ_jk_ik_jk](s(ik,jk), nil nil)
)
```

Figure 13. General LOTOS expression

converters (size or protocol type) may have an influence on the global system behaviour.

At first let us consider the size converters. When a given initiator communicates with a target with different bus size, a converter is introduced in order to split packets or merge packets. But the overall operation is still atomic on the bus, and especially for these two processes. Thus, the size converters are not generated in the LOTOS abstract model, but rather all the processes communicating directly with the greatest bus size on single rendez-vous.

Secondly, the protocol type 2 and 3 have different behaviour :

- type 2 protocol preserves the order of requests and responses. One constraint is that, when communicating with a given target, an initiator cannot send a request to a new target until it has received all the responses from the current target. The unresponded requests are called “pending requests”, and this constraint is controlled by a pending request controller ;
- a given type 2 target is assumed to send the responses in the same order as the requests arrival order ;
- in type 3 protocol, the order of responses may not be guaranteed, and an initiator can communicate with any target, even if it has not received all the responses from a previous one ;

Two nodes may be of different protocol types, type 2 and type 3 for instance, and then a type converter is used :

- a converter from type 2 to type 3 has to transform the type 2 request packets by adding a “tid”, and reorder (by memorization) the type 3 responses to generate the responses for the type 2 initiator ;
- a type 3 to type 2 converter consists in deleting the `tid` (ie transaction identifier) of type 3 requests and regenerating the `tid` on the type 2 responses (coming in order) for the type 3 initiator.

In order to model the behaviour of the system with respect to these properties, the generated LOTOS model has the following additional properties :

- a type 2 initiator process is completed with a pending request controller (PRC) LOTOS process, synchronized on all its gates. Thus in the node, the requests and responses are performed on a rendez-vous between 3 processes (initiator/target/pending request controller). The PRC process just counts the requests and responses for a given target, and allow or not the initiator to communicate with a new target ;
- in a given N_i type 2 node, the source internode buffers (for incoming requests from node N_j), are also considered as initiators and, thus, synchronized with their own PRC ;
- in the same way, a type 3 to type 2 converter is considered as an initiator in the type 2 node and, thus, synchronized with its own PRC ;
- we have modeled a type 2 to type 3 converter, parameterized by the size of the FIFOs, and it replaces an internode FIFO presented in the general STBus case ;
- a type 3 to type 2 converter is modeled by a specific process.

5.3 Pending request controller for type 2 protocol

Let us consider a node N_i with nI_i initiators, and $DN(i)$ the destination nodes and $DS(i)$ the source nodes. Each initiator $I_{(i,k)}$ is synchronized on all its gates with its own PRC process. An initiator PRC of node N_i is named `PRC_I_i`, the gates of the process depending on the node. Each incoming FIFO, `FIFO_j_i` from node N_j to the current node N_i, is synchronized on its gates `REQ_D` and `R_REQ_D` with its own PRC. A FIFO PRC is named `PRC_FIFO`.

The profile of the `PRC_I_i` process of node N_i is defined by :

- the gates `REQ_i`, `R_REQ_i` and the gates `REQ_i_i_j` and `R_REQ_i_i_j`, $\forall j \in DN(i)$;
- and the parameters `init`, initiator number in the node, `node` the node number, `last_target` and `last_node` the identification of the last or current target with which the initiator communicates and

`nb_pending`, the number of pending requests with the current target.

The LOTOS model of the interconnection including these features is the one of figure 13 in which some FIFOs are replaced by protocole type converters and :

- the type 2 protocol initiator instances are replaced by the expression
  `(I_i_k[GATES] ||PRC_I_i[GATES](k,i,0,00))`
- all the buffers (or converters) instances, from a node N_i to a node N_j of type 2, are replaced by the following expression :
  `PRC_FIFO[REQ_j_i_j, R_REQ_j_i_j](0,0)`

5.4 LOTOS code generation tool

The input of the generator, is a structural description of the interconnect consisting in : the nodes identification, for each node, it protocole type, the identification of the initiators and targets, for each initiator or target, and all possible internode communications (from/to), the size of the associated buffer. The generator builds the LOTOS expression instantiating initiator and target processes. Buffers, converters and PRC are instances of generic predefined LOTOS processes.

6 Conclusion and perspectives

The LOTOS code generator has been implemented in C++. The generator as well as the different LOTOS models (internode BUFFER, type converters, pending request controllers) have been validated by analyzing (by mean of simulation and temporal logic formulæ evaluation) the LTS generated for different kinds of systems (STBus interconnect) and different arbitrary LOTOS processes for the initiators and targets. This part has to be completed by taking into account the type 1 protocol in which request and response are performed atomically on the bus. As LOTOS semantic does not allow to build a response depending on a request in one single rendez-vous, one solution consists in perfoming 2 successive actions non interleaved with any other action by means of an non-interleaving process [1].

In the same way this work treats SoC bus, it is necessary to take into account the RT-OS properties (scheduling strategy, inter-task communication mechanisms) for the processes implemented in software on a same microprocessor. One can refer to SystemC Version 3 for instance. The described approach can also easily be applied on other SoC buses such as AMBA Bus for instance.

In order to generate automatically the LOTOS model of a complete system future work will be conduced on the processes behaviour. At first, defining a behavioural translator from a design description language (SystemC) into a LOTOS process. Then, in order to avoid the state space explosion, we will need to apply abstraction on data computing and on control domains. For this, we start a study consisting in merging static code analysis on SystemC (by means of abstract interpretation, for instance) with the system temporal model checker. On one hand, the results of static analysis should conduce to automatically abstract data and control domains. This is an important step as we will consider compositional LTS generation in which, for a given process, the entry domain has to be limited by means of interface process [3]. Furthermore, the state space can also be reduced by limiting the interleaving of actions from independent subsystems or processes. This has been studied and defined as partial ordering [9].

A first step has been made in the aim the generate automatically LOTOS code, temporal formulæ for a given SoC. By completing this work it would be possible to check automatically a given SoC by a design engineering, with much concepts being masked to him.

References

[1] F. Baray. *Contribution to the Integration of Model Checking in the Codesign Process of Systems (in french)*. PhD thesis, University Blaise Pascal, Clermont-Ferrand II, FRANCE, July 2001.

[2] F. Baray and P. Wodey. Verification in the Codesign Process by Means of LOTOS based Model-Checking. In *5th International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, April 2000.

[3] H. Garavel and F. Lang. Svl: a scripting language for compositional verification. In M. Kim, B. Chin, S. Kang, and D. Lee, editors, *Proceedings of the 21st IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems FORTE'2001 (Cheju Island, Korea)*, pages 377–392. IFIP, Kluwer Academic Publishers, Aug. 2001. Full version available as INRIA Research Report RR-4223.

[4] H. Garavel, F. Lang, and R. Mateescu. An overview of cadp 2001. Rapport technique RT 254, INRIA, Dec. 2001.

[5] T. Grötker, S. Lian, G. Martin, and S. Swan. *System Design with SystemC*. Kluver Academic Publisher, 2002.

[6] T. Ismail and A. Jerraya. Synthesis Steps and Design Models for Codesign. *IEEE Computer*, February 1995.

[7] ISO-8807. LOTOS*, A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*. 1988.

[8] R. Keller. Formal verification of parallel programs. *Communications of the ACM*, 19, 1976.

[9] D. Peled. Combining partial order reduction with on-the-fly model-checking. *In Dill*, 19:371–384, 1997.

[10] SDL. *CCITT. Recommendation Z.100: Specification and Description Language, volume X.1-X.5*, 1988.

[11] P. Wodey and F. Baray. Linking Codesign and verification by means of E-LOTOS FDT. In B. Werber, editor, *Euromicro, 99 Digital Systems Design*. IEEE Computer Society, September 1999.