μSPIDER CAD TOOL: CASE STUDY OF NOC IP GENERATION FOR FPGA

Samuel Evain, Rachid Dafali, Jean-Philippe Diguet, Yvan Eustache, Emmanuel Juin

μSPIDER CAD TOOL: CASE STUDY OF NOC IP GENERATION FOR FPGA

Samuel Evain

CEA LIST
CEA Saclay 91191 Gif sur Yvette
email: samuel.evain@cea.fr

R. Dafali, J-Ph.Diguet, Y. Eustache, E. Juin

Univ. Européenne de Bretagne / CNRS / LESTER lab
56100 Lorient, France
email: jean-philippe.diguet@univ-ubs.fr

ABSTRACT

This paper introduces the μSpider CAD tool for NoC design under latency and bandwidth constraints and describes the different steps of the associated design flow. We show how the tool can be used to automatically generate a NOC IP compliant with Xilinx EDK tool. We present synthesis results and a real implementation of a video application based on a multi-processor architecture. Finally we conclude about research to be done at application/OS levels above current work to achieve a complete and efficient implementation of a multi-processor embedded system.

1. INTRODUCTION

The concept of Network On Chip (NoC) has been recently introduced as an alternative to bus in order to solve the tedious issue of emerging system on chip (SoC) interconnect design. Different arguments are commonly forwarded to sustain the interest for NoCs [1] based on packet switching. First the NoC can provide new spatial and time parallelism capabilities to cope with expanding bandwidth requirements and the management of an increasing number of inter IPs communications. It also contributes to raise the abstraction level of design tools for flexibility and productivity improvements. This point is crucial, since the design of complex SoC, implementing tens of processors, IPs and memories, means error prone communication schemes that can be intractable for designers under time to market pressure.

Moreover a NoC is based on physical and logical control flows that offer new opportunities for controlling quality of service including real-time, power and reliability. Finally, NoC intrinsically holds the property of scalability, which is a key point for design reuse and SoC configurability.

Today first industrial solutions [2] are available and ongoing researchs investigate further aspects. One of the research issues in which we particularly focus on is design flow to handle complexity and provide designers with CAD tool. Some equivalent work is proposed in Æthereal [3] for ASIC design. This approach is based on an architecture model and a methodology, which is close to ours but differs at level of mutual exclusions considerations and heuristics choices for path and time slot allocation. There is no real work about a specific CAD tools for NoC IP generation targeting FPGA. In [4] FPGA are used as a proof of concept independently from any CAD tool. Other work are based on FPGA for emulation purpose in [5] and to speed-up simulation in [6].

To our point of view, the main issue is the productivity gain for implementing complex communication schemes. Our project was, from the beginning, driven by the objective of providing the SoC designer with an ad-hoc component offering services for communication management at application level. To get over this challenge we have designed a CAD tool and a software layer for a simplified access to the communication medium at application level. On the one hand, it is well known that flexibility has a cost. An area overhead mainly due to interface and router FIFOs and an increase of communication latencies due to the path length and packet routing are generally observed. However, the paradigm can take benefit from a rigorous formalization that enables the implementation of efficient automatic methods. We based our framework development on the guaranty of latency and bandwidth requirements at application level while minimizing the resulting NoC cost.

μSpider CAD tool for NoC design performs design space exploration and code generation. Design space exploration is implemented in an interactive way based on designer choices for arbiter, routing policies and topology selections. Then automatic procedures are available for time-consuming and error prone tasks such as Time Division Multiplexing (TDM), FIFO sizing and path allocation for guaranteed traffic management. Then based on designer and tool choices, code generation is automatically carried out and can specifically targets Xilinx EDK tool by producing the IP folder including data/IP.mpd and data/IP.pao and /hdl/vhdl/*.vhd files with the correct format. In this paper we present the μSpider design flow as IP generator for reconfigurable MPSoC implemented on Xilinx FPGA.

2. DESIGN FLOW OVERVIEW

Our NOC model enables the implementation of two kinds of communications: best effort (BE) and guaranteed traffic (GT) based on a TDM technique. Figure 1 gives an overview of

the NoC design Flow. An interactive GUI (Fig. 2 helps the designers to easily follow the μSpider design flow.

The first step enables the designer to rapidly specify the application and the NOC parameters. The application is specified with a set of characterized communication tasks, with application throughputs (Application Graph.xml) and if necessary with links between mutual exclusive communications (Mutual Exclusion Graph.xml). The NOC knobs are related to the NOC topology which can be ad hoc or automatically generated as 2D Mesh. NoC parameters are specified by the designer, the main ones are: Path bit width (e.g. 32 bits) Router parameters (ports, routing policy, arbiter policy, etc.), Network Interfaces (NI), Flow control policy (with or without End to End flow control) and wrapper (slave, master, bus standard) and the IP mapping namely the IP/NI association. If real-time constraints are required, then related communications are implemented with virtual channel (GT, BE, BE with priority).

The second step deals with derivation of local latency and bandwidth constraints for each unidirectional communication from application I/O throughputs (e.g. telecom chain or image processing). The objective is to extract local latency and bandwidth constraints for each communication task from global I/O application constraints. The important issue, which is usually omitted in NOC design flows, is in practice necessary for applying following steps 3 and 4, which requires constraints for each individual and unidirectional communication. This work is not trivial since firstly different local decisions are possible to meet global constraints and secondly latency, bandwidth and TDM table size are strongly dependent. Moreover read operations imply two types of heterogeneous communications: the read command (request) and the data response for which two distinct set of constraints must be defined. Thus, the second step first transforms communication tasks into unidirectional ones. This aspect is required for read operations that need a lightweight forward communication for sending a read command and a backward communication for receiving data. Then we produce, for each GT communication, the minimum bandwidth and a set of rules for latency/bandwidth checking. Due to space restriction the step is not detailed in this paper but can be found in [7].

The third step computes the minimum TDM table size required for implementing GT communications and a minimum bandwidth for all BE communications. TDM tables are based on integer bandwidth division, so compared to real constraints higher bandwidths are usually obtained, so we first try to solve the TDM size issue while considering absolute lower bounds. Basically our method is based on a heuristic that starts with a minimum size, which is increased until a solution is found. As a result we obtain minimum latencies taken into account within the next step.

The fourth step [8] automates the more tedious task which is the exploration of time (TDM slots)-space (NOC paths) space in order to allocate time slots to each GT communication. It provides next steps with a complete NOC specification. The exploration / allocation step is based on a two steps heuristic that, for each communication, first evaluates link usage probabilities and then selects a valid path with minimum impact on non allocated paths. The heuristic parameters (cost function, sorting criteria) have been selected for globally minimizing the FIFO costs.

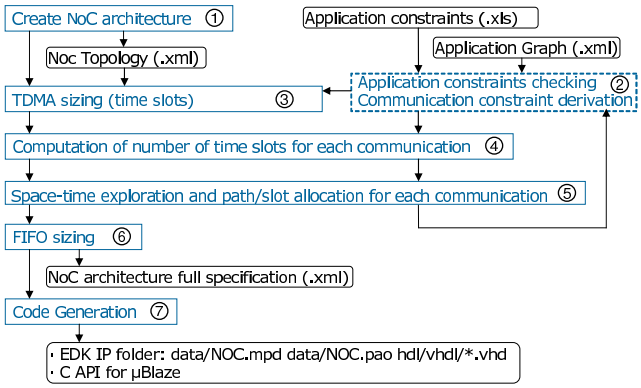


Fig. 1. μSpider design flow.

The fifth step is the VHDL code generator, some additional C API codes are also provided for interfacing NOC components with IPs which are compliant with the OPB bus standard.

3. ARCHITECTURE MODEL

The architecture model is a network of bus-based clusters connected through a NoC. In case of Xilinx targets, we use OPB-based clusters connected to a NoC component instantiated as a traditional IP in an EDK project. As depicted in Fig. 3, typically a cluster is composed of a Microblaze (MB) soft processor that can control one or more local RAM memories connected to its OPB bus. So, the NoC IP is in charge of inter-cluster communications, the interface cluster / NoC is built with two components : a wrapper that adapts OPB and NoC network interface protocols and the Network Interface, which manages NoC accesses and (un)packetise data. Each cluster can integrate masters able to initiate read or write communications and slaves that can respond to read requests. A cluster can communicate with other clusters through dedicated *channels*, each input or output channel is implemented as a FIFO. Note that the architecture model is based on first designer choices such as topology, routing policies and link bit width for instance.

In the following we present different aspects of the architecture model.

3.1. Communication model

Three kinds of communications are implemented: write, read and message passing. They are depicted in Fig. 4. Fig. 5

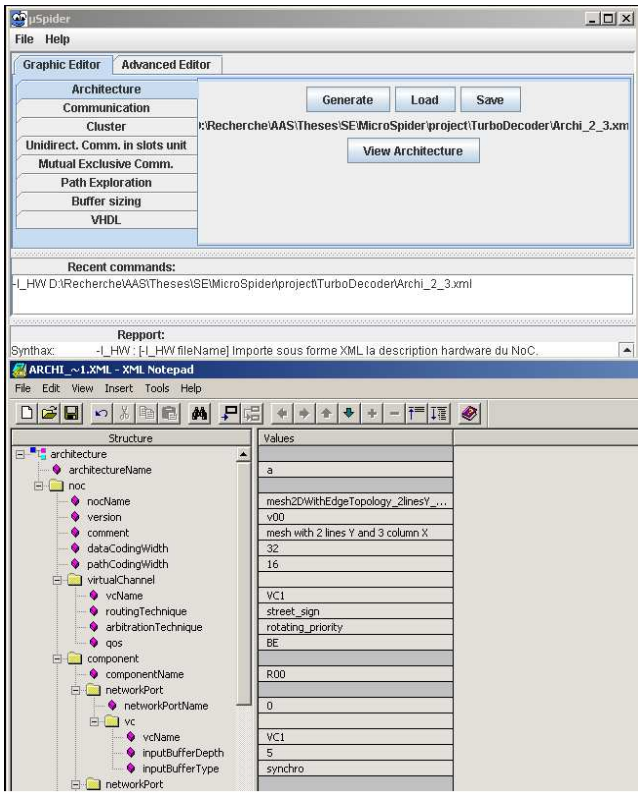


Fig. 2. μSpider CAD tool.

illustrates these different schemes, where the following commands are implemented between masters and wrappers:

- `Rc: Read`
- `S: Status register (In/Out FIFO not full, almost full, not empty, almost empty)`
- `Wc: Write`
- `D: Data FIFO`
- `rW: Write request`
- `rR: Read request`

The first scheme, described in (Fig. 5-a), is a `write` operation that initiates a master to write data in a remote slave RAM 2 after checking that enough space is available in input FIFO.

The second of one, depicted in (Fig. 5-b), is a `read` operation, which is quite tricky since a master cannot freeze the OPB bus while waiting for a slave to answer its request. So a real read operation is not appropriate and a write operation is used instead. In practice a master writes a read request in slave wrapper registers, an interrupt is emitted by the local wrapper to the master when requested data are available.

The third scheme, depicted in (Fig. 5-c), is a `message` passing between two masters, an interrupt is emitted to alert the remote master that a message is available.

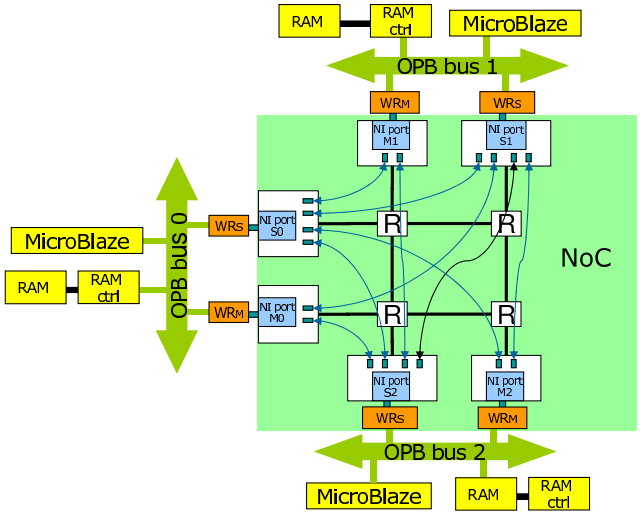


Fig. 3. Architecture model.

Note that an alternative solution, based on polling instead of interrupt, has been developed for wrappers / masters communication. Such a scheme is interesting when a single task is assigned to the master.

3.2. Software layer

From an application point of view, these communication mechanisms are available as basic C API, which provide MB with NoC services.

- `extern int WRAPPER_S_Read_Status(int Wrapper_Address)`: Returns the wrapper status for each channel.
- `void Wrapper_S_Read_(int Wrapper_Address, int channel, int Remote_Address,int *Data, int n)`:Read n data on channel (remote IP ID), at Remote_Adress (local address) and stores in *Data.
- `extern void Wrapper_S_Write_BRAM(int Wrapper_Address, int channel, int Remote_Address, int *Data, int n)`: Write n data on channel (remote IP ID), at Remote_Adress (local adress)
- `extern void Wrapper_S_Recvn(int Wrapper_Address, int channel, int Data, int n)`: Read data in FIFO channel.
- `extern void Wrapper_S_Sendn(int BaseAddress, int channel, int *Data, int n)`: Write data in FIFO channel.

A Hardware Abstraction Layer (HAL) has been developed to separate abstract and physical addresses, thus a single API is used above previous one:

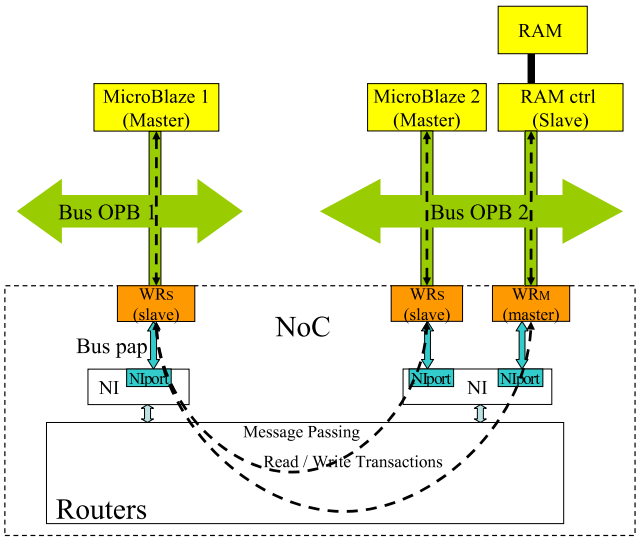


Fig. 4. Communication model.

```
Write(Logical_Address, N, RW,*p)
```

Where Logical_Address is then transformed into Wrapper_Address, channel and local address if necessary.

3.3. NI / OPB wrappers

Two kinds of wrappers are available to adapt OPB and NI protocols, a Slave wrapper depicted in Fig. 6-a) and a Master wrapper described in Fig. 6-b).

3.4. Network Interface

The Network Interface (NI) model, described in Fig. 7, is generic. A NI is specialized according to NoC specifications. The main configuration parameters are the number of channels, the TDM table. The number of output (resp. input) channels corresponds to the maximum number of receiver (resp. emitter) IPs. For cost reasons no more than two virtual channels are usually used, one for BE traffic and one for GT traffic, each channel can be associated to one virtual channel category. A TDM table is used to allocated contiguous time slots to output channel according to bandwidth and latency constraints as explained in section 2.

3.5. Router

The Router model, given in Fig. 8 is quite largely configurable. Main configuration knobs are number of routers network ports, routing technique (XY or Street sign) and arbiter policy for BE (round robin, first arrived - first served). Note that a new street-sign path coding model [9] has been designed in order to improve security, reconfigurability and header overhead. Consequently, based on a minimum number of coding bits according to each router arity individually, a path can be used as an identifier [10] and a simple mechanism is available to transform a forward path into a reverse path without knowledge about a possible new configuration (emitter placement, topology, router arity).

4. RESULTS AND EXPERIMENTS

4.1. Introduction

The μSpider design flow has been successfully applied for designing different NoCs in the domain of Telecom for instance [7], we also have recently proposed an architecture for security management in NoC-based reconfigurable SoC [10]. Implementing real MPSoC requires important engineering efforts especially if specific IPs are needed. So, in a first stage generated NOCs have been implemented for generic applications. The test architecture was based on 3 MB, 2 slave RAM and a 2x2 mesh NoC. The architecture is described in Fig. 3.

4.2. Test conditions

These experiments have been completed on a Xilinx Pro FF1152 BOARD based on a Virtex-II VP50-5 device, with Xilinx ISE 6.3 SP3 and EDK 6.3 SP2. In the EDK framework, the NOC is implemented as an usual IP (see Fig. 9).

Our experimental NOC has been parameterized as follows: bit width: 32; Routing: street sign; End to end flow control; Round robin arbiter; 2 routers with 3 ports; 2 routers with 4 ports; 3 NIs with 1 port and 2 Channels; 3 NIs with 1 port and 3 Channels; 3 master wrappers; 3 slave wrappers.

Different synthetic programs have been implemented on MBs in order to test a large set of configurations, each configuration has been validated in terms of data integrity and transfer timing. Tests results have been obtained through an UART connected to a MB. The following five virtual channel configurations have been successfully tested:

- BE: single BE channel with a 2 slots input FIFO;
- GT: single GT channel with a 16 slots TDMA table;
- GT-BE: two virtual channels, BE and GT;
- BE-BE: two virtual BE channels and two priorities;
- BE8: one single BE channel with a 8 slots input FIFO.

4.3. Results

Based on the XML NoC architectural description, VHDL code generation is performed with a delay lower than 2s on a standard PC. NOC specification roughly corresponds to 10.000 VHDL code lines. Moreover, as previously mentioned, a code is produced with the EDK format and can be used directly as any IP. Table 1 synthesizes results for each case study. NoC sizes don't include wrappers which have the same following features for all cases:

Table 1. NOC synthesis results.

		NoC	NI2Ch	NI4Ch	R3ports	R4ports
Case 1	Slices	7204	607	1167	430	645
BE	Freq.	94	120	118	136	136
Case 2	Slices	7226	624	1195	430	643
GT	Freq.	84	120	118	136	136
Case 3	Slices	9251	630	1181	851	1317
BE & GT	Freq.	85	93	90	118	118
Case 4	Slices	9205	621	1170	851	1317
BE & BE	Freq.	84	93	90	118	118
Case 5	Slices	7488	622	1203	443	662
BE8	Freq.	100	120	114	132	132

- Master wrapper: 1064 *slices*, max. frequency: 105MHz.
- Slave wrapper: 90 *slices*, max. frequency: 144MHz.

A complete NoC uses around 30-40% of the whole FPGA whereas a router needs about 1-5% and a NI represents 2-4%. NoC max. The frequency is greater than 80 MHz, it means that a theoretical link bandwidth around $32 * 80.10^6 = 2560.10^6 bits/s \approx 320 Mo/s$. For a given FIFO size, we observe that NIs require significantly more resources when both GT and BE virtual channels are implemented. Actually quality of service (GT or priority BE) has a cost which is mainly due to TDM tables and scheduling mechanisms in NIs.

4.4. Object tracking real-case study

4.4.1. Application Implementation

The aim of this implementation is to proof the ability of our platform to implement real and complex applications. The object tracking application has been initially developed by CEA-LIST in a very generic C code for multi-target prototyping purposes.

These experiments have been completed on the same VirtexII board used for previous tests. The target architecture is equivalent to Fig.9 except that the first MB is replaced by a Power PC. The reason of this choice reveals one the current practical locks in the domain MPSoC design, namely the question of interface standards. The NoC IP used is based on GT communications and correspond to case 2 in tab.1.

The application has been partitioned on the target as follows. The hard-mapped PPC processor runs four functions, Image loading & pre-processing (format adaptation), noise filtering based on an average of 2 frames, background subtraction and VGA control. When a new image is ready, a message is sent to the MB 0 that launches three tasks: Image loading from DDR SDRAM connected the PLB PPC bus, adaptive thresholding and load resulting data in its local RAM memory, finally MB0 sends a message to MB1 to inform it that it can load new data. MB1 runs functions Image loading from MB0 RAM, Dilatation, erosion, reconstruction, Gravity center computation, Labeling and Border drawing and storage in DDR SDRAM.

The implementation and the use of the IP NoC was quite plug and play, in that sense the experience was a success since we have implemented processing parts as software running on embedded (soft/hard) processors and communications through the NoC in a very short time. Moreover, the whole application functionality has been checked.

4.4.2. Observations

If the objective was not performances, results we obtained are quite poor since we finally get one image per second. The problem is basically not on the NoC side which is underused and able to provide expected bandwidth and latencies. Actually the causes of performance degradations, which can be solved by the way, are i) interface unavailability, ii) EDK limitations and iii) missing OS services.

Interface: The heterogeneity of standards and the availability of interfaces for various peripheral impose constraints to the designer, who, under time and economic constraints, implements possible options instead of his real choices. In our case we have wrappers for OPB bus standard, whereas a PLB bus is required for the PPC communications and the (free) VGA controller. It means that additional master and slaves OPB/PLB bridges have been implemented to cope with this point. This issue can be easily solved with the appropriate IP library, moreover PLB/OPB wrappers are very similar.

EDK: EDK tool is an efficient tool if the designer deals with the architecture model where memories are acceded through OPB or PLB buses. However if OPB interfaces are used for interfacing memories, a MB or PPC connected to the bus is required to initialize the address map. As a result all memories are implemented as slaves on OPB or PLB bus and introduce conflict accesses to shared memories, the consequence is that processor runs sequentially whereas a pipeline execution would be theoretically possible. Again this problem can be solved with ad hoc simple wrappers for memory interfacing.

OS: The last and much more complex issue is the question of synchronization at application level. Basically a NoC can prevent the system from transaction and transport level deadlocks with end to end credit-based and local handshakes flow controls, however the designer remains in charge of tuning the application control flow in such a way that no deadlocks happen at application level.

A solution to this tedious question is the implementation of a new class of OS services to manage NoC services while taking benefits from OS synchronization and mutex facilities. The last point is quite tricky since it has to be seen in relation with the kind of communications required by application in terms of burst sizes, periodic or sporadic, dynamic or static behaviors, data dependencies and so on. We currently work on this point to bring up NoC services as system level.

5. CONCLUSION

μSpider is a NoC CAD tool with two main parts, the first one performs architecture design and automatically runs tedious and error prone tasks such as path extraction and coding, TDM sizing, under QoS (latency, bandwidth) constraints while targeting resource minimization (FIFO sizes). Some features such as path minimization and multiplexing based on mutual exclusion properties are exploited to reduce final cost. The second part of the flow, based on the XML NOC architecture description, is a code generator. Moreover, NIs can be configured according to the space-time exploration step to respect communication constraints. VHDL code is automatically produced in such a way, that it can be used as a usual IP within the Xilinx EDK tool. Our approach and tool have been validated with real cases. These experiences have revealed weaknesses of current CAD framework for MPSoC design on FPGA. Interfacing problems will be fixed with CAD tool evolutions, however the definition and formalization of new OS/NoC services remain a real challenge and open promising perspectives to address the design of multiples processes with dynamic behaviors on future (reconfigurable) MPSoC. We currently work on this topic.

6. REFERENCES

[1] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *Design, automation and test in Europe conf. (DATE)*. ACM Press, 2000, pp. 250–256.

[2] "Arteris, the noc company," http://www.arteris.net, 2005.

[3] K. Goosens and al., "A design flow for application-specific networks on chips with guaranteed performance to accelerate soc design & verification," in *DATE*, Washington, USA, 2005.

[4] T. Marescaux and al., "Networks on chip as hardware components of an os for reconfigurable systems," in *13th Int. Conf. on Field Prog. Logic and Appli.*, Portugal, Sept. 2003.

[5] N. Genko, D. Atienza, G. Micheli, L. Benini, J. Mendias, R. Hermida, and F. Catthoor, "A complete network on chip emulation framework," in *DATE*, Washington, USA, 2005.

[6] P. H. P. Wolkotte and G. Smit, "Fast, accurate and detailed noc simulations," in *1st ACM/IEEE Int. Symp. on Networks-on-Chips (NoC07)*, Princeton, USA, may 2007.

[7] S. Evain, J.-P. Diguet, M. Khodary, and D. .Houzet, "Automated derivation of noc communication specifications from application constraints," in *IEEE Workshop on Signal Processing Systems (SiPS)*,, Banff, Canada, June 2006.

[8] S. Evain and J.-P. Diguet, "Efficient space-time noc path allocation based on mutual exclusion and pre-reservation," in *17th ACM Great Lakes Symposium on VLSI (GLSVLSI)*, Italy, mar 2007.

[9] J.-P. Diguet and S. Evain, "Patent fr 05/53280," oct 2005.

[10] J.-P. Diguet, S. Evain, R. Vaslin, G. Gogniat, and E. Juin, "Noc-centric security of reconfigurable soc," in *1st ACM/IEEE Int. Symp. on Networks-on-Chips (NoC07)*,, Princeton, USA, may 2007.

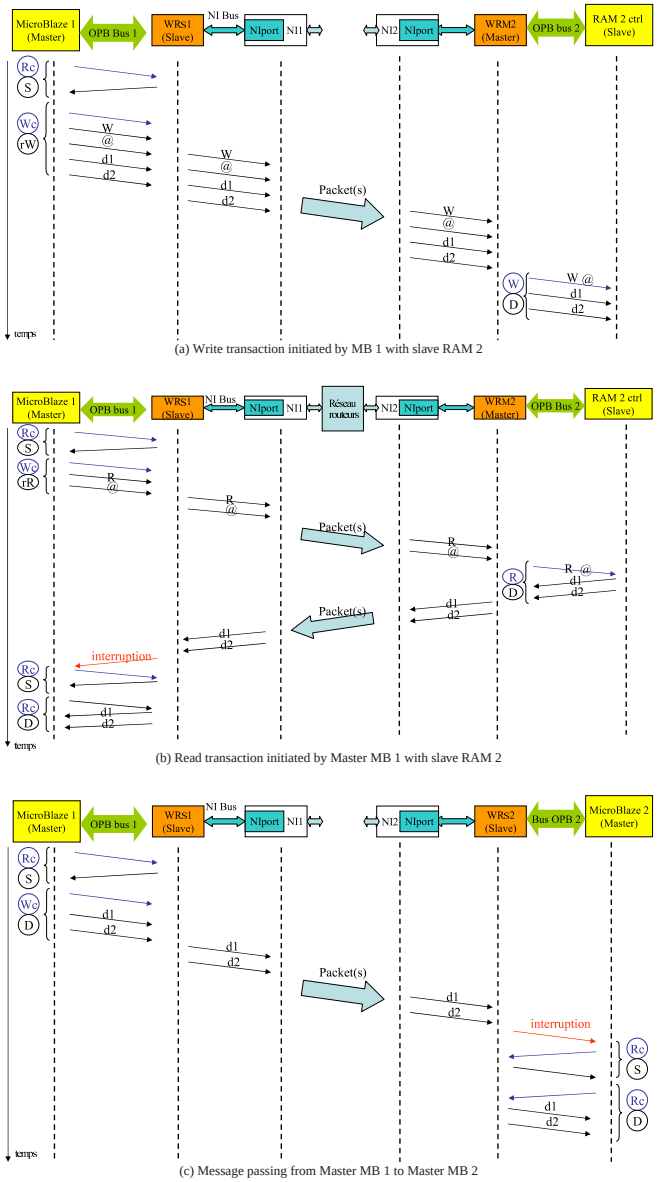
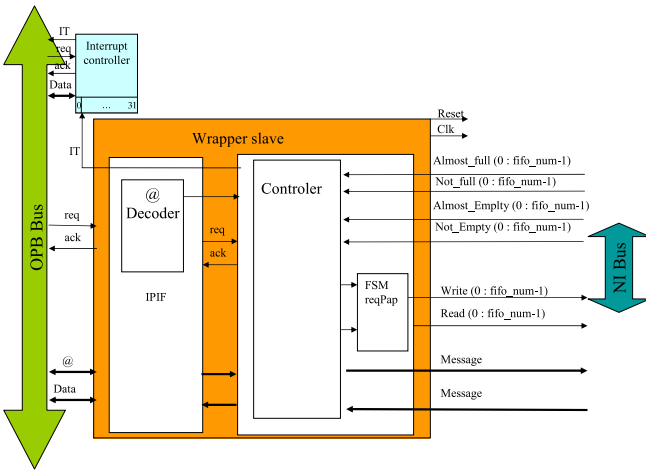
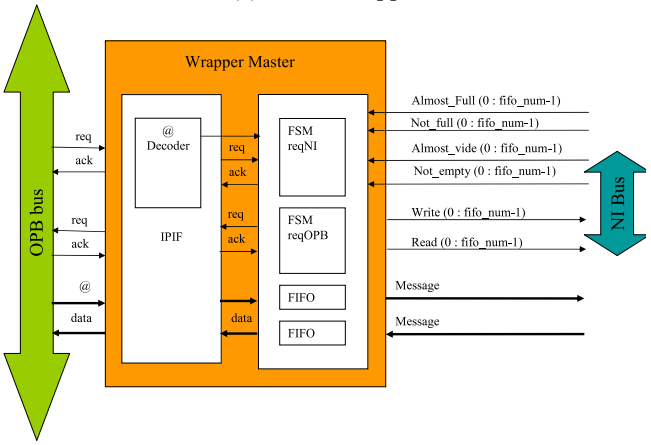


Fig. 5. Communication model scenario.



(a) Slave Wrapper



(b) Master Wrapper

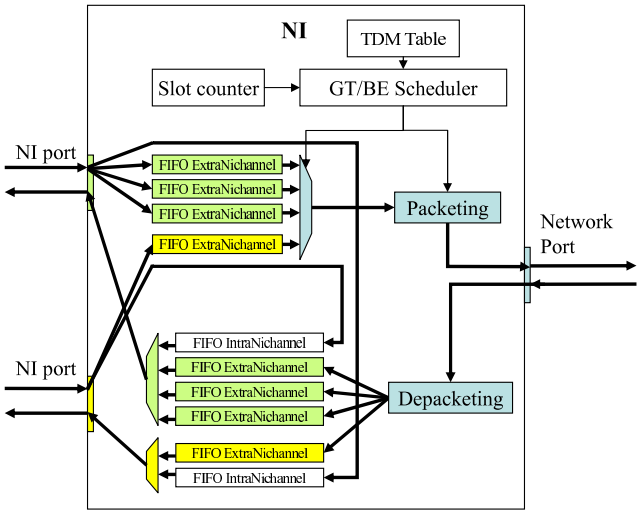


Fig. 7. Network Interface (NI) Model.

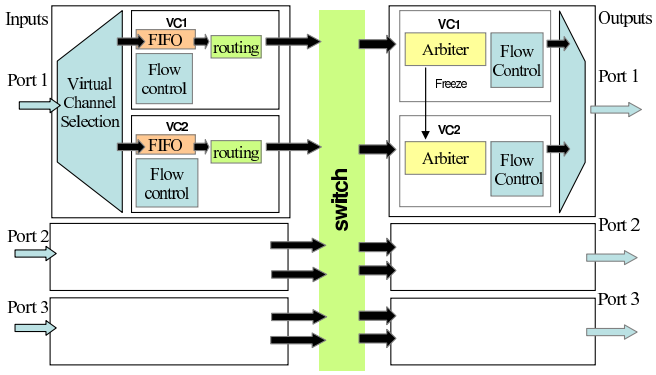


Fig. 8. Router Model.

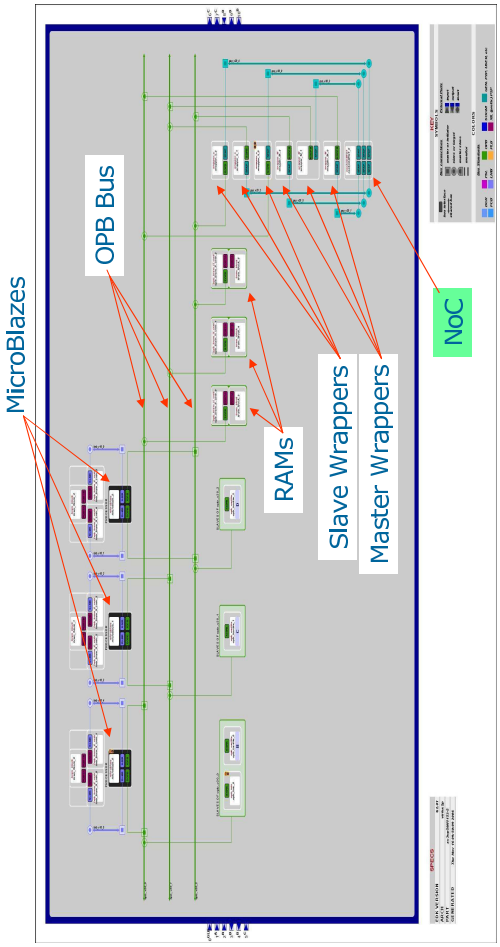


Fig. 9. IP NOC in EDK.