FPGA Prototyping and Design Evaluation of a NoC-Based MPSoC

Ridha SALEM, Yahia SALAH, Imed BENNOUR and Mohamed ATRI
Electronics and Microelectronics Laboratory, Faculty of Sciences
University of Monastir, 5000 – Monastir, Tunisia

***Abstract*—Chip communication architectures become an important element that is critical to control when designing a complex MultiProcessor System-on-Chip (MPSoC). This led to the emergence of new interconnection architectures, like Network-on-Chip (NoC). NoCs have been proven to be a promising solution to the concerns of MPSoCs in terms of data parallelism. Field-Programmable Gate Arrays (FPGA) has some perceived challenges. Overcoming those challenges with the right prototyping solutions is easy and cost-effective leading to much faster time-to-market. In this paper, we present an FPGA based on rapid prototyping in hardware/software co-design and design evaluation of a mixed HW/SW MPSoC using a NoC. A case study of two-dimensional mesh NoC-based MPSoC architecture is presented with a validation environment. The synthesis and implementation results of the NoC-based MPSoC on a Virtex 5 ML 507 enable a reasonable frequency (151.5 MHz) and a resource usage rate equals to 58% (6,586 out of 11,200 slices used).**

Keywords*—MultiProcessor System-on-Chip; Network-on-Chip; FPGA Field-Programmable Gate Arrays (FPGA) prototyping; design evaluation

I. INTRODUCTION

Technological advances in recent years on the programmable components, specifically FPGAs, have improved their capacity for integration and the connection between their different logic cells, thus making it possible to implement a complete MPSoC in a single FPGA device. These FPGA-based multiprocessors systems, with hard and soft cores, have become the standard for implementing heterogeneous embedded architectures [1]. They facilitate rapid prototyping and allow building scalable and modular applications. However, massive growth in size and complexity in recent years and future MPSoCs places on-chip interconnect at the system performance center. Traditionally on-chip communication has been conducted via dedicated point-to-point links or a shared media like a bus. Bus-based architectures are simple and completely widespread; use of these approaches do not scale very well when more intellectual property (IP) cores are integrated in a system and will not meet the requirements of the future MPSoCs because of their seriously limited scalability. Also, they quickly become the bottleneck of a system [2]-[4]. By using the interconnection network as the communication infrastructure between cores, Networks-on-Chip (NoCs) are emerging as an efficient and scalable alternative to existing on-chip interconnects which allow systems to be designed modularly. Different NoCs solutions are used in MPSoC platforms and commercialized by many companies such as SonicsGN [5] developed by Sonics, FlexNoC [6] by Arteris, Æthereal NoC [7] by Philips Research Laboratories and Teraflops Research Chip (also called Polaris) [8, 9] by Intel Corporation's Tera-Scale Computing Research Program. Other NoC-based multi-core system architectures are developed by teams from universities and research institutions such as SoCIN [10], OCCN [11], FAUST [12] and Ninesilica [13].

Some of the above-mentioned proposals and several many-core system designs still use simulation and mathematical analysis for the evaluation of their on-chip interconnects under various network configurations [14], [15]. However, it is important that prototyping must be considered to improve the evaluation accuracy by bringing the design closer to reality.

Unlike conventional hardware prototyping approaches, FPGA-based prototyping of mixed hardware/software MPSoC architecture became an extremely challenging task. It requires specific FPGA expertise hardware/software codesign flow and environments. Moreover, many competences are required such as the mastery of prototyping hardware platform (ML507), the software development flow (tools, drivers, RTOS, etc.) as well as the hardware development flow (specification, synthesis, placement, routing etc.). In addition, different interconnection solutions can be covered between the software and the hardware blocks. Also, the configuration and integration of IP blocks and the use of soft and hard processors were included.

This work focuses on the prototyping of a mixed hardware/software FPGA-based MPSoC using two-dimensional mesh NoC architecture. The basic performances of the investigated MPSoC are to be explored in a fast and efficient hardware-based way. The paper is divided into five sections. Starting with the presentation of a survey on existing FPGA prototyping approaches for MPSoC platforms (section II). Moving to Section III which shows the MPSoC design flow and describes the EDK Tools and Design Flow Integration. Then Section IV that gives the details of the NoC architecture and the Fast Simplex Link (FSL) bus interface as they are the basic elements of the MPSoC platform. Section V shows the FPGA prototyping of a NoC-based MPSoC. Section VI evaluates the hardware simulation and synthesis results. Last but not least, Section VII concludes the paper and highlights future work.

II. RELATED WORKS

MPSoC with NoC are strongly emerging as prime candidates for complex embedded applications. Also, the

interest in NoC prototyping is continuously growing, as many recent processing chips are multi-cores. On the one hand, prototyping such systems is a quite complicated task. In order to allow fast generation of these platforms in the development phase, a full design flow is required. On the other hand, modern FPGAs provide the possibility for fast and low-cost prototyping in HW/SW co-design, representing an efficient response to these needs. With the increase of available reprogrammable logic cells, many works have explored the possibility to implement an entire NoC-based MPSoC on FPGA [16], [17]. In [18], Lukovic et al. presented a framework, based on the Xilinx EDK design flow, for the generation of MPSoCs based on NoCs. This integrated design flow takes as an input a textual description of the system and produces as a final result a configuration bitstream file. In [19], Lokilo et al. proposed an array-based MPSoC architecture, matching requirements of applications where the data can be splitted into several subsets and processed in parallel, as is the case in numerous video processing algorithms. They have physically implemented a 2x2 Xtensa core system in a Virtex II Pro and tested it in a real time application. Van Langendonck et al. proposed an integrated framework MPSoCDK for rapid prototyping and validating NoC-based MPSoC project targeting FPGA devices [20]. Similarly to this design flow, MPSoCDK aims at speeding up the processes of designing, exploring and prototyping MPSoC projects. It also simplifies the process of designing complex projects through a Graphical User Interface (GUI), providing a hardware and software layer. However, the proposed flow produces pure synthesizable VHDL and does not create project files for tools such as Xilinx XPS or Altera SoPC. In [21], Geng et al. used the FPGA device to prototype the cluster-based MPSoC with 17 processing cores. Moreover, a suite of benchmarks, including several parallel applications with different characteristics of parallelism, workload and communication pattern, are designed and presented. It has been reported that a complete design methodology has been successfully used for the implementation of a NoC-based MPSoC, the NoCRay graphic accelerator. Noting that this design methodology tackles at once the aspects of system level modeling hardware architecture and programming model, the design which is based on 16 processors has been laid out in 90-nm technology after prototyping with FPGA. Post-layout results show very low power, area, and high frequency [22]. Wächter et al. presented an open source platform for MPSoC development named HeMPS Station which derived from the MPSoC HeMPS [23]. In its present state, it includes the platform (NoC, processors, DMA and NI), embedded software (microkernel and applications) and a dedicated Computer-Aided Design (CAD) tool to generate the required binaries and perform debugging. Experiments show the execution of a real application running in HeMPS Station.

The solution proposed in this work is based on a concept similar to [18]. However, its aim is to perform a low cost hardware realization in FPGA taking into account the integration in MPSoC environment. We have realized on a Xilinx Virtex5 FPGA, a system composed of MicroBlazes running without operating system (OS), shared memory blocks, and a NoC as an interconnecting medium among them.

III. Design Methodology and FPGA-Based System Prototyping of MPSoCs

A. MPSoC Design Flow and Verification Approach

The development of complex systems is increasingly involved with specific software and hardware components. The co-design provides solutions for this type of development. It is based on a set of steps that allow as to synthesize a SoC integrating software and hardware components that respect the imposed design constraints (e.g. time and surface). A standard design flow is typically composed by four main steps: specification, partitioning, synthesis and HW/SW verification (see Fig. 1). These steps can be summarized as follows:

1) The system modeling allows describing its functional behavior without taking into account the architecture. At this level, the interest is to obtain relevant results in terms of performance and timing.

2) The SW/HW partitioning is the step following the system modeling. At this level, the architectural details of communication are integrated with the scheduling of all operations.

This step appears to split the system into three major parts:

- A hardware part implemented as a hardware circuits and generally used for performance. This part can be considered as an IP obtained from a library or a hardware accelerator that is made especially for a specific task.
- A software part implemented as an executable program on processor and generally used for features and flexibility. This processor can be a General Purpose Processor (GPP) or a reconfigurable processor (configured according of the application needs).
- A communication interface between these two parts.

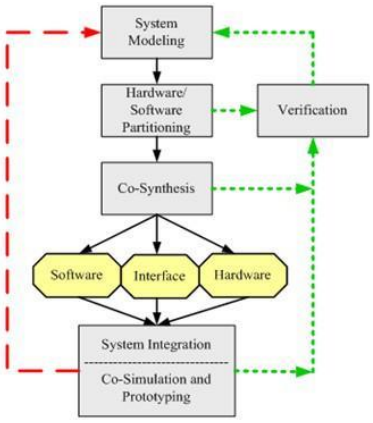

Fig. 1. MPSoC design flow [24].

In fact, these obtained parts must be verified and validated before the synthesis and implementation phases. If the partitions obtained are not satisfied, a feedback is needed to return to the partitioning stage in order to refine the weights that are associated with constraints for each part differently. Then, several simulations will be made to choose the best distribution between the software and the hardware parts.

3) The synthesis step also called implementation. In this step, the Register Transfer Level (RTL) description for the hardware part and the source code for the software part of the system are obtained. Obviously, verification and validation of the functionality of the generated model should be done. At this stage of design, the analysis is concerned with the performance of the architecture at the cycle level and at the bit level through co-simulations.

4) The last step of the design flow consists first of the logical synthesis of the RTL part of the system. Then the logical functions that have been synthesized which will be placed and routed on the chip. This process is accomplished by the use of commercial synthesis tools such as: Simplify [25], Xilinx Synthesis Technology (XST) [26], Leonardo Spectrum [27], etc. The software part of the system will be compiled to generate a hexadecimal image. Finally after obtaining the performance such as area and energy consumption in the logical synthesis, the co-simulation will be established. Once the architecture is validated, various real tests through the FPGA prototyping platform [28] are to be made.

During each step in the design flow of an MPSoC, the verification should be performed by designers. Consequently, it is ensured that the new components or the new implementation details providing a proper functionality. Verification can take up to 70% of the device design time [29], [30]. This step has a major cost in terms of time as well as financial. There are several techniques of verification: formal verification, simulation, co-simulation, emulation, co-emulation and prototyping. In this work the focus is mainly on the prototyping stage.

Prototyping is a solution that reduces the time of design, development, verification and validation of SoCs [31]. Although, FPGA has some perceived challenges, overcoming those challenges with the right prototyping solutions is easy and cost-effective leading to the fastest time-to-market.

The software components made by programs executed through one or more processors. However, the hardware components of the application are made with FPGA programmable blocks. It uses configurable components to implement physical blocks and connections. To achieve this type of prototyping, it is sufficient to just have a description of RTL or gate of all components and reconfigurable prototyping platform.

Several tools and companies have adopted this rapid prototyping in HW/SW co-design approach regarding their simplicity of synthesis and integration of new components. Among these development tools is EDK proposed by Xilinx [32].

B. EDK Tools and Design Flow Integration

Xilinx provides various software which enable to create embedded SoCs, among these softs the ISE and EDK. The ISE tool is used especially to produce hardware IP projects from a Hardware Development Language code (HDL) [33].

However, the EDK tool allows us to establish a direct link between the hardware and the software parts of a system. It includes a system generator for processor and Xilinx Platform Studio (XPS) [34]. Thus, all design flows are grouped in XPS environment [31].

The standard design flow of Xilinx consists of two main steps (Fig. 2): the first one consists of the conception and the synthesis of the design. The second step consists of the design implementation and verification. Moreover, the design of an embedded system typically includes four phases (creation and verification of the hardware and the same for the software platform).

For the EDK tool, the hardware platform is defined by the MHS (Microprocessor Hardware Specification) file. The verification platform allows the user to define the simulation model for each system component (processor and peripherals). If the software application is executable, then it can be used to initialize the memories. The software platform is defined by the MSS (Microprocessor Software Specification) file. The creation and verification of a software application involves several steps: To start with the writing of the code in C, C ++ or assembler language that will be executed through the software and the hardware platforms. After that this code is compiled and linked using the GNU tool (other tools can also be used) to generate the executable file in ELF (Executable and Link Format) format. Then, Xilinx Microprocessor Debugger (XMD) and the GNU debugger (GDB) are used to debug the application for the target processor [26].

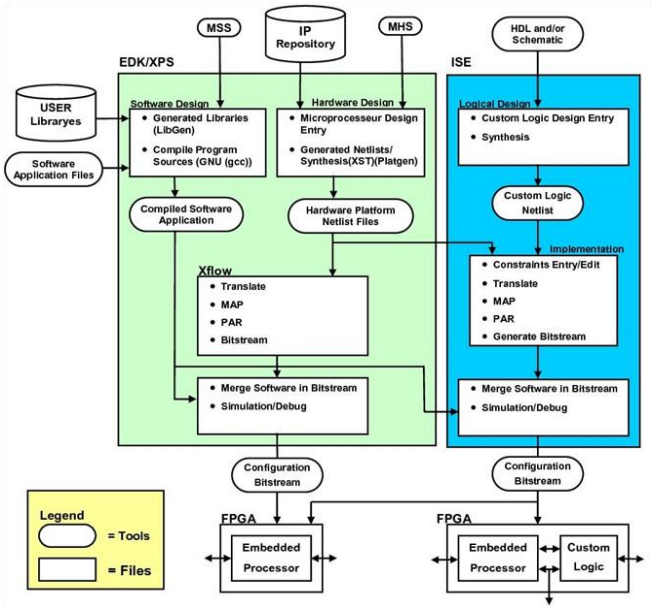

Fig. 2. Detailed design flow XPS/ISE for Xilinx FPGAs [24], [32].

Synthesis and simulation are the two main steps in the Xilinx design flow. The design tasks allow switching from one description to another to arrive at the bitstream configuration file. Indeed, a logical synthesis makes it possible to pass from an RTL description of the architecture to a description at the logical gate level (Netlist). The description of logical elements is optimized according to the speed, area or consumption constraints imposed by the designer. The XST synthesis tool

replaces the generic logical elements with the FPGA. Placement and routing convert the hardware description into a configuration file. It generates a file, which is used to configure the interconnection matrices of the FPGA circuit. At each stage of the design, the CAD enables us to perform simulations in order to validate each step of the implementation: Functional simulation at the RTL level, Post-synthesis simulation at the logic gate level and Post-layout simulation at the physical level.

IV. NoC-Based MPSoC Platform

The multi-processor platform template is shown in Fig. 3. The architecture platform consists of multiple tiles connected with each other by a NoC. Each tile contains a local memory (M), and a network interface (NI), that is accessed both by the local IP core inside the tile and by the NoC. The IPs are responsible for the computation of the desired functions and may be hardwired or programmable processors. The NoC connects all tiles together via its routers (R) and links (L).

Two different types of tiles are distinguished based on the functionality of the IP inside the tile and the size of the memory (M). The first type, called processing tile, contains an IP as a processor which executes the code of the applications running on the platform. The application code and some of the data structures needed when executing it are stored in the local memory of the tile. The second type called memory tiles contains a part of the memory sub-system that can be accessed from the processing tiles. From the memory's perspective, only the NI and IP processor try to access this memory. In this work, we are interested in the processing tile.

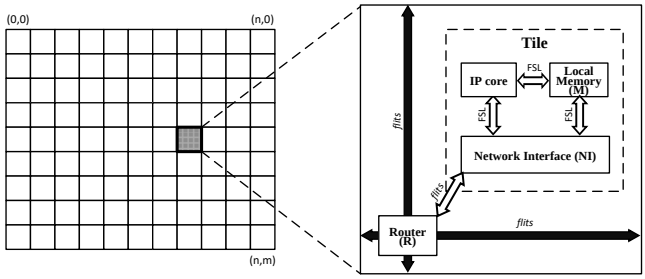

Fig. 3. NoC-based MPSoC architecture overview.

A. NoC Architecture

The on-chip communication structure between the tiles in the platform template should offer unidirectional point-to-point connections between pairs of NIs. The connections must preserve the ordering of the communicated data. To evaluate the MPSoC design, a system interconnection model is needed. The NoC model of Yang et al. [35] has been used.

The architecture platform consists of a set of routers which are connected to each other in an arbitrary topology. Regular topology is a popular NoC architecture due to its predictability and ease of design. The used NoC has a 2D-Mesh topology, where each router is connected with its neighbor and its own NI by bidirectional communication channels [36]. The size of a physical channel is 8 bits. A router has a routing unit, a control block and a number of generic input-output ports. This number depends on the used topology. In this case, there are five communication ports which are indexed as follows: East (index 0), West (index 1), North (index 2), South (index 3) and Local (index 4). The Local port provides communication between the router and its NI component. The other ports are connected to neighboring routers. In order to avoid deadlock, the XY routing algorithm is used where message or packet will always be routed firstly in X (horizontal) direction, and then into Y (vertical) direction. The serialization and the deserialization steps must be done at the NI interface in order to transfer the data to the heart of IP to the router.

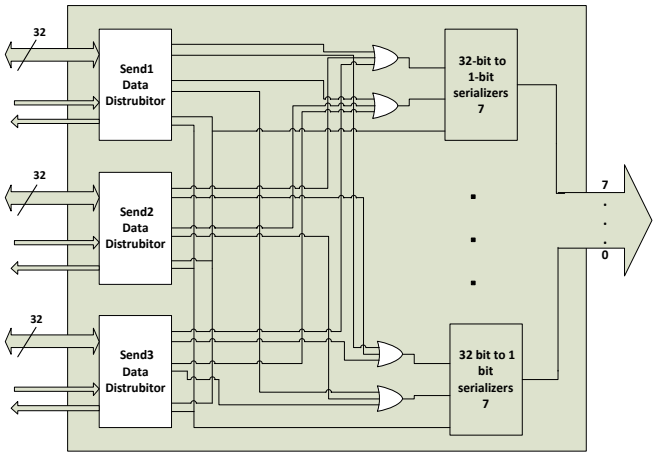

Fig. 4. Block diagram of the NI component.

Fig. 4 shows the NI architecture proposed by the work of [36]. On the one hand, it is connected to the router via eight wires. On the other hand, it is connected to the IP via three communication channels with 32-bit data width for each one. The NI architecture consists of a serializers number from 32-bit to 1-bit. This number depends on the number of wire per port connected to the router (eight wires in this case).

The architecture also consists of three data distributors where each one is connected with an output message queues via a communication channel of 32 bits data width. It is responsible for transmitting data received through this channel to the appropriate serializer. The serializer inputs consist of two OR gates. The first one allows all distributors to forward their data to each serializer. The second and the 1-bit output are handshaking signals between the data distributor and the serializer. The network can operate in normal mode or control mode. The last mode is used to program the routers. The data are used to program the NoC will be transmitted to the control network through the node (0, 0).

B. FSL Bus Interface

Two types of connections are possible to connect a MicroBlaze to the NI: the PLB (Processor Local Bus) or FSL buses [37]. The FSLs are used because they adapt well to the NoC. Indeed, one FSL bus allows fast access (two clock cycles) devices to the MicroBlaze and vice versa (8 FSL connections by MicroBlaze). The FSL bus width is 32 bits and the C-functions were used to read or write data into the FIFO of the bus. So, it is quite simple to create an adapter since it is enough to read or write words in a FIFO with checking their status. Fig. 5 shows the interface of a FSL bus. The IP master of the FSL connection is the MicroBlaze and the IP slave is the NI. Table 1 illustrates an overview of the FSL-related

predefined C-functions available in EDK tools and used in the software applications (swappx).

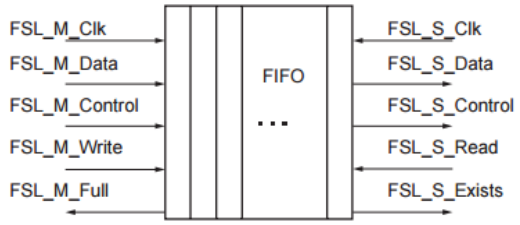

Fig. 5. Block diagram of the FSL bus.

TABLE I. SUMMARY DESCRIPTION OF THE C-FUNCTIONS USED IN SOFTWARE APPLICATIONS

C-Function Name	Description	Parameters	
		Argument	*Type*
microblaze_bread_datafsl microblaze_bwrite_datafsl	Blocking Data Read and Write to FSL Local Link	DeviceId Data	Xuint16 Xuint32
microblaze_nbread_datafsl microblaze_nbwrite_datafsl	Non-blocking Data Read and Write to FSL Local Link	DeviceId Data	Xuint16 Xuint32
microblaze_bread_cntlfsl microblaze_bwrite_cntlfsl	Blocking Control Read and Write to FSL Local Link	DeviceId Data	Xuint16 Xuint32
microblaze_nbread_cntlfsl microblaze_nbwrite_cntlfsl	Non-blocking Control Read and Write to FSL Local Link	DeviceId Data	Xuint16 Xuint32

V. 2D-MESH NOC-BASED MPSOC PROTOTYPING ON FPGA

The target system is an MPSoC composed of four MicroBlazes processors interconnected through a NoC mesh 2×2. Fig. 6 shows the system architecture. The four MicroBlazes processors are connected to the NoC via point-to-point links. A laptop connected via Universal Asynchronous Receiver/Transmitter (UART) at MicroBlaze A0; enables debug data to be sent/received in order to verify the NoC functioning.

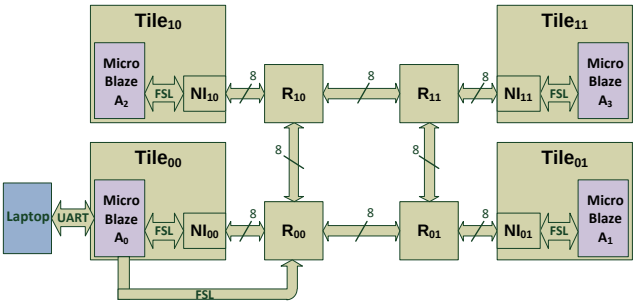

Fig. 6. System architecture of a 2x2 2D-mesh NoC-based MPSoC.

In this work, the Virtex5 FPGA Xilinx XC5VFX70 device is targeted to implement the MPSoC system prototype in order to provide area overhead, power dissipation and operating frequency. The investigated system (see Fig. 7) is composed mainly of Xilinx MicroBlaze processors, memory blocks and a NoC.

- The MicroBlaze is an embedded soft core provided by Xilinx [38]. Since processing tile was chosen in Section IV, processing nodes includes data and instruction memories connected to the MicroBlaze processor through the dedicated Local Memory Bus (LMB). We connect MicroBlazes to the rest of the system through their interface to the FSL
- Shared memory blocks are implemented using part of the Block RAM (BRAM) available on-chip in Xilinx boards. Memory cores are synchronous and three write mode options were supported: Read-Before-Write, Read-After-Write and No-Read-On-Write. A LMB BRAM controller is associated to the BRAM component in the aim to manage data transfer from and to the adopted bus system.
- NoC is basically composed of two elements: NIs and routers, as described in the previous section.

Fig. 7 shows the block diagram of the entire design of MPSoC based on a NoC 2D-Mesh 2x2. The standard peripherals that are connected to the MicroBlazes through the PLB bus were also presented. The different IPs that make up the MPSoC design prototyped in the Xilinx Virtex 5 target device are summarized in Table 2.

TABLE II. DESCRIPTION OF ALL IPS IMPLEMENTED IN THE MPSOC DESIGN

IPs used in MPSoC		*Version*	*Quantity*	*Description*
NoC	Router	--	4	K-way router with XY routing algorithm
	NI	--	4	NI component serialize and the de-serialize in order to transfer the data to the heart of IP to the router
Microblaze		8.00.a	4	for each processor is associated a frequency of 100 MHz and a separate software code
BRAM		1.00.a	4	Local memory blocks, one block for instruction and one for data, 16 KB for each block
LMB		1.00.a	8	Buses on which must be connected eight local memory controllers
LMB BRAM controler		2.10.b	8	Four controllers for instructions and data
PLB		4.6	1	Bus which connects the four processors and other IPs
FSL		2.11.c	9	Busses that connect the NoC to MicroBlazes processors
UART Lite RS232		1.01.a	1	To connect the Laptop to MicroBlaze A0, to enable debug data to be sent/received and to verify the NoC functioning.
MDM		2.00.a	1	For debugging MicroBlaze processor A0
Clock Generator		4.00.a	1	Takes in common clock requirement and generates architecture-specific clocking circuitry.
PSRM		3.00.a	1	Reset Module of the system
JTAG		--	1	To program the FPGA

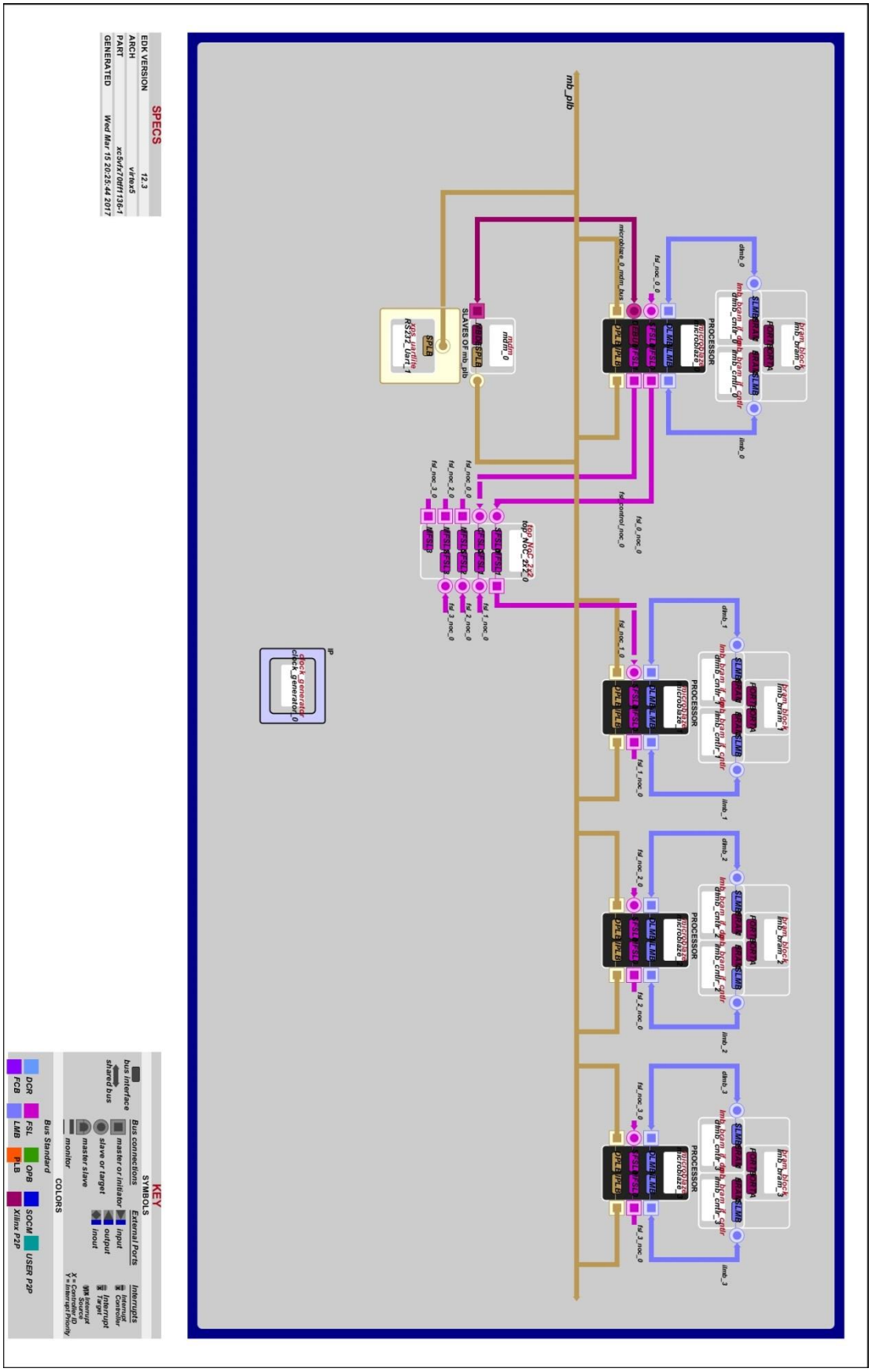

Fig. 7. Blocks Diagram of MPSoC design prototyped in the Xilinx Virtex-5 FPGA.

VI. Evaluation Results

In this section, simulation and synthesis results are presented to demonstrate the performance of this 2D-NoC-based MPSoC architecture.

A. Environments and Parameters

The Xilinx ISE environment is used for both design and implementation. VHDL behavioral simulations are typically performed with the ModelSim tool. For the MPSoC creation, several criteria are necessary for the choice of the used tools. Among these criteria is the type of the used materials (Xilinx prototyping platform in our case) where each supplier offers these own tools. Another criterion is about the nature and constraints of the MPSoC.

In the running case, the aim is to implement a 2D mesh NoC-based MPSoC at RTL level in a reconfigurable platform FPGA type. As a result, the use of ISE 12.3 tool for the design and implementation of Hardware accelerators, ModelSim for architectural simulation and verification.

Ultimately, EDK is used for the integration of different hardware accelerators into a complete MicroBlaze processor-based system. The necessary parameters used in the hardware accelerators (the 2D-NoC and the NI interface) are shown in Table 3.

TABLE III. Configuration Setups of Hardware Accelerators

IPs	NoC		NI	
Experiment Parameters	***Topology***	2x2 2D Mesh	***Resource allocation***	SDM
	Router	K-way		
	Number of router ports	5	***Data width***	32 bits
	Routing algorithm	XY		
	Flow-control Protocol	Handshaking	***Front-end protocol***	FSL
	Physical link width	8 bits		
	Data width	32 bits	***Back-end protocol***	Handshaking

B. Hardware Simulation Results

A test bench file is employed to replace the original IPs modules placed in their corresponding NIs and routers for testing the efficiency of NoC. The test bench module could generate a set of packets. The NoC and NI hardware accelerators are modeled in VHDL language, using the RTL description. Hardware accelerators were simulated with ISE Simulator. This is a very important step to verify the system function and to calculate system performance such as latency and throughput.

Fig. 8 illustrates the packets transmission from the three masters NI00, NI01 and NI10 (sources) to the same slave NI11 (destination).

C. Hardware Synthesis Results

In this section, the synthesis results are presented and disscussed. The MPSoC performance will be evaluated in terms of area, power consumption and clock frequency.

The synthesis begins when the system is fully integrated. The make file created in previous phase leads to the execution of HW/SW synthesis tools of the EDK design flow. Hardware flow is run first. After Netlist creation of the target system, Xilinx implementation flow is executed.

Then, the bitstream file is generated and the software flow takes place. This phase consists of three steps: As a first step, software applications (swappx) are added for the four MicroBlazes as cited: swapp0 for MicroBlaze 0, swapp1 for MicroBlaze 1, swapp2 for MicroBlaze 2 and swapp3 for MicroBlaze 3. In this swappx, the C-functions (Table 1) is used in order to send and receive data between the four MicroBlazes. OS is not needed because this system is not oriented neither real-time nor multitasking. The second step consists of the custom libraries generation which is followed by a compilation and linking of source code as a third step. Once both hardware and software flows are executed, the bitstream file is initialized with BRAM data (for initialization of data instruction memories attached to processing units). The final result of the automation engine is a configurable bitstream file which is directly downloaded to the attached Xilinx ML507 Virtex-5-XC5vfx70 platform using the prototyping flow.

The synthesis results of the MPSoC system on Xilinx Virtex 5 target device are summarized in Fig. 9.

The synthesis of the target design enables a moderate operating frequency around 151.5 MHz. The FPGA resource usage rate is about 58% (6,586 out of 11,200 slices used).

The synthesis result of NoC (routers + NIs) is given in Fig. 10. The resource utilization of the NoC is 31% of the device area and the maximum frequency is 264.6 MHz with a critical path delay of 3.386 ns.

It is clearly observed that the maximum frequency of the MPSoC system (152 MHz) is remarkably lower than the IP NoC (265 MHz). Note that the time is inversely proportional to the frequency, the time of the shortest path is higher in the system MPSoC. As a result, the minimal period in this system is higher than the IP NoC.

The resource utilization of the rest of blocks is given in Table 4. The IP that takes low slices is the FSL bus. However, the NI component and MicroBlaze take the higher area cost.

Table 5 illustrates a comparison between this evaluated NoC-based MPSoC design and the design proposed in [18]. The area of this MPSoC design is greater than the area of Homogeneous System presented in [18]. This is due to many reasons. First of all, there is a difference between the composition of the system composed of four MicroBlazes and NoC 2×2, and the other one with three MicroBlazes and NoC 2×1. Second, a five-port router was used while a three-port router was used in [18]. Finally, it is important to note that the NI is reliable and more efficient. Indeed, it gives many services such as the number of used serializers and deserializers. For that, it consumes 863 slices as compared to the NI reported in [18] that consumes 85 slices. Nevertheless, this MPSoC system achieves a higher frequency (151.5 MHz) for an attractive data rate.

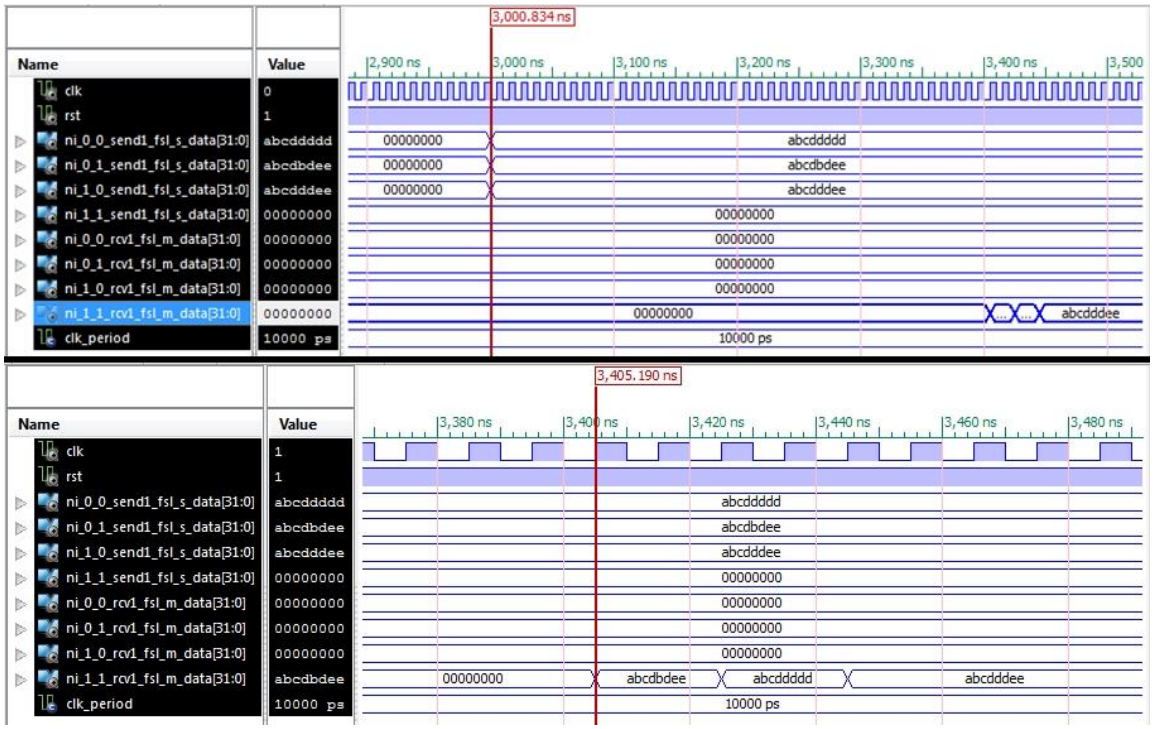

Fig. 8. Simulation waveform during packet transmission in 2×2 2D-mesh topology.

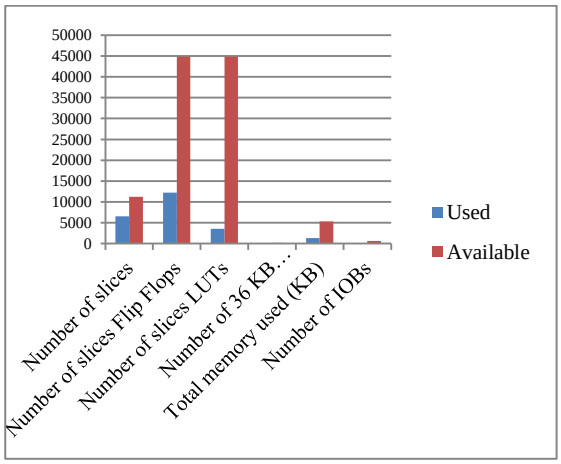

Fig. 9. Synthesis results of MPSoC system based Virtex5-XC5VFX70.

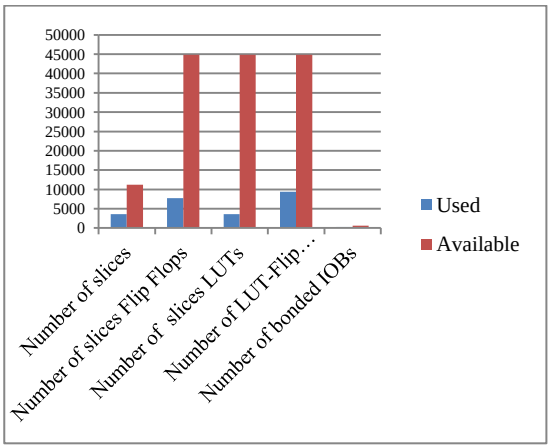

Fig. 10. Synthesis results of NoC-based Virtex5-XC5VFX70.

TABLE IV. SUMMARY TABLE OF AREA COST BY MPSOC SYSTEM COMPONENTS

Component	Area (slices)	Number
Router	78	4
NI	863	4
FSL	22	9
MicroBlaze	1,221	4
MPSoC	6,586	

TABLE V. THE COMPARISON BETWEEN THE EVALUATED MPSOC DESIGN AND OTHER

NoC-based MPSoC	FPGA	Composit-ion of the system	Performance Analysis			
			Area			*Max Frequency (MHz)*
			LUTs	*Flip Flop*	*Slices*	
Homogen-eous System of [18]	Xilinx Virtex-II Pro	3 Micro-Blazes and NoC 2×1	5,891	2,510	3,527	--
The evaluted MPSoC design	Xilinx Virtex 5 XC5-VFX70	4 Micro-Blazes and NoC 2×2	9,627	12,216	6,586	151.5

VII. CONCLUSION AND OUTLOOK

In this paper, an FPGA-based rapid prototyping in HW/SW co-design and design evaluation of a mixed HW/SW MPSoC using a network-on-chip (NoC) was described. Xilinx Virtex-5 FPGA installed in ML507 prototyping hardware platform with Xilinx EDK and ISE software was used to perform the prototyping of the system. The system consists of four MicroBlaze processors interconnected through a network-on-chip mesh 2×2. The design evaluation of a NoC-based

MPSoC, that is found, gives a reasonable frequency of about 151.5 MHz and FPGA resource usage rate of 58% corresponding to 6,586 out of 11,200 slices. The OS component Xilkernel has not been used and the system ,which is developed, was not oriented neither real-time nor multitasking. As a next work, the focus will be on investigating the prototyping multitasking real-time systems on multiprocessor architectures with OS using advanced prototyping platform.

REFERENCES

[1] Dammak B, Baklouti M, Benmansour R, Niar S, and Abid M. Hardware resource utilization optimization in FPGA-based Heterogeneous MPSoC architectures. Microprocessor and Microsystems, 2015, 39(8), 1108-1118.

[2] Bafumba-Lokilo D, Savaria Y, and David J.P. Generic crossbar network on chip for FPGA MPSoCs. In Proceedings of the Joint IEEE North-East Workshop on Circuits and Systems and TAISA Conference (NEWCAS-TAISA '08), 2008, 269–272.

[3] Hur J.Y, Stefanov T, Wong S and Goossens K. Customisation of on-chip network interconnects and experiments in field-programmable gate arrays. IET Comput. Digit. Tech., 2012, 6 (1), 59–68.

[4] Arjomand M, Boroumand A, Sarbazi-Azad H. A generic FPGA prototype for on-chip systems with network-on-chip communication infrastructure. Comput. Electr. Eng. 2014, 40(1), 158–67.

[5] http://sonicsinc.com/products/on-chip-networks/sonicsgn

[6] http://www.arteris.com/flexnoc

[7] Steenhof F, Duque H, Nilsson B, Goossens K and Llopis R.P. Networks on Chips for High-End Consumer-Electronics TV System Architectures. DATE Designers' Forum 2006, 148-153.

[8] Vangal S, Howard J, Ruhl G, Dighe S, Wilson H, Tschanz J, Finan D, Singh A, Jacob T, Jain S, Roberts C, Hoskote Y, Borkar N, and Borkar S. An 80-Tile Sub-100 W TeraFLOPS Processor in 65-nm CMOS. IEEE Journal of Solid-State Circuits. 2008, 43 (1), 29–41.

[9] http://www.intel.com/pressroom/kits/teraflops/

[10] Zeferino C.A, and Susin A. SoCIN: A Parametric and Scalable Network-on-Chip. Symposium on Integrated Circuits and Systems Design, 2003, pp. 121-126.

[11] Coppola M, Curaba S, Grammatikakis M.D, Maruccia G, and Papariello F. OCCN: a network-on-chip modeling and simulation framework. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, 2004, 3, 174–179.

[12] Vivet P, Lattard D, Clermidy F, Beigne E, Bernard C, Durand Y, Durupt J, and Varreau D. FAUST, an Asynchronous Network-on-Chip based Architecture for Telecom Applications. Proc. 2007 Design, Automation and Test in Europe (DATE07), 2007.

[13] Airoldi R, Ahonen T, Garzia F, Milojevic D, and Nurmi J. Implementation of W-CDMA Cell Search on a Highly Parallel and Scalable MPSoC. Journal of Signal Processing Systems, 2011, 64 (1), 137-148.

[14] Kiasari A.E, Jantsch A, and Lu Z. Mathematical Formalisms for Performance Evaluation of Networks-on-chip. ACM Comput. Surv., 2013, 45 (3), pp. 38:1–38:41.

[15] Hamid N, Walters R and Wills G. Simulation and Mathematical Analysis of Multi-core Cluster Architecture. The 17th UKSIM-AMSS International Conference on Modelling and Simulation, 2015, pp. 476-481.

[16] Hecht R, Kubisch S, Herrholtz A, and Timmermann D. Dynamic Reconfiguration with hardwired Networks-on-Chip on future FPGAs. International Conference on Field Programmable Logic and Applications, IEEE, 2005.

[17] Balal A, Erdogan A.T, and Khawam S. Architecture of a dynamically reconfigurable NoC for adaptive reconfigurable MPSoC. First NASA/ESA Conference on Adaptive Hardware and Systems (AHS'06). IEEE, 2006.

[18] Lukovic S, and Fiorin L. An automated design flow for NoC-based MPSoCs on FPGA. The 19th IEEE/IFIP International Symposium on Rapid System Prototyping. IEEE, 2008.

[19] Bafumba-Lokilo D, Savaria Y, and David J.P. Generic array-based MPSoC architecture. Microsystems and Nanoelectronics Research Conference, IEEE, 2009.

[20] Van Langendonck R, Kuti Lusala A, and Legat J.D. MPSoCDK: A framework for prototyping and validating MPSoC projects on FPGAs. Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 7th International Workshop on IEEE, 2012.

[21] Geng L.F, Zhang D.L, and Gao M.L. Performance evaluation of cluster-based homogeneous multiprocessor system-on-chip using fpga device. Fourth International Conference on Embedded and Multimedia Computing, IEEE, 2009.

[22] V.Tota S, R.Casu M, Ruo Roch M, Macchiarulo L, and Zamboni M. A case study for NoC-based homogeneous MPSoC architectures. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2009, 17 (3), 384-388.

[23] Wächter E.W, Biazi A, and G. Moraes F. HeMPS-S: A homogeneous NoC-based MPSoCs framework prototyped in FPGAs. Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 6th International Workshop on IEEE, 2011.

[24] Samahi A, and Boukadoum M. Improved MPSoC co-design methodology for stream oriented processing applications." Electronics, Circuits, and Systems (ICECS), 17th IEEE International Conference on IEEE, 2010.

[25] Synplicity Inc., "Synopsys FPGA Synthesis Synplify Pro for Microsemi Edition", User Guide, February 2013.

[26] Xilinx. XST User Guide for Virtex-4, Virtex-5, Spartan-3, and Newer CPLD Devices, UG627 (v 12.4), 14 Dec. 2010.

[27] Mentor Graphics Company, "Leonardo Spectrum for Altera User's Manual Software", Version v2001.1 July 2001.

[28] Sasongko A, Baghdadi A, Rousseau F and Jerraya A.A. Prototyping of Embedded Applications to Configurable Platform Driven by Communication Constraints. Proceedings of the 14th IEEE International Workshop on Rapid System Prototyping (RSP 2003), San Diego, USA, 2003.

[29] Jóźwiak L, Nedjah N, and Figueroa M. Modern development methods and tools for embedded reconfigurable systems: A survey." Integration, the VLSI Journal 43.1, 2010, 1-33.

[30] Zargaryan G.Y. Verification Environments for USB Controller. In Mathematical Problems of Computer Science, 2013, 39, pp. 72–80.

[31] Nikolov H, Stefanov T, and Deprettere E, Efficient Automated Synthesis, Programming, and Implementation of Multiprocessor Platforms on FPGA Chips. Field Programmable Logic and Application, 2006, pp. 1-4.

[32] Xilinx. EDK Concepts, Tools, and Techniques, UG683 EDK 12.2.

[33] Xilinx. Xilinx ISE Software Manuals, 2005.

[34] Xilinx. Xilinx Platform Studio User guide, UG113, 15 Feb. 2005.

[35] Yang Z.J, Kumar A and Ha Y. An Area-efficient Dynamically Reconfigurable Spatial Division Multiplexing Network-on-Chip with Static Throughput Guarantee. In Proceedings of the International Conference on Field-Programmable Technology, 2010, pp. 389 –392.

[36] Josephan Y.Z. Area-efficient dynamically reconfigurable Spatial Division Multiplexing Network-on-Chip with static throughput guarantee. Thesis of engineering at the National University of Singapore, 2010.

[37] Xilinx. LogiCORE IP Fast Simplex Link (FSL) V20 Bus (v2.11c), DS449, 2010.

[38] Rosinger H.P. Xilinx Connecting Customized IP to the MicroBlaze Soft Processor Using the Fast Simplex Link (FSL) Channel XAPP529 (v1.1) Dec. 19, 2003