CROSSBUS: A PROGRAMMING MODEL AWARE NOC

Gabriel Oshiro Zardo[1,3] *and Dominique Houzet*[2,3]

1Universidade Federal do Rio Grande do Sul, Porto Alegre, BRAZIL {gabrieloshiro@gmail.com}
2Institut National Polytechnique de Grenoble, Grenoble, FRANCE {dominique.houzet@lis.inpg.fr}
3Grenoble Image Parole Signal Automation Laboratoire, Grenoble, FRANCE

ABSTRACT

Network on Chip (NoC) links can reduce the complexity of designing wires for predictable speed, power, noise, reliability, etc., thanks to their regular, well controlled structure. From a system design viewpoint, with the advent of multi-core processor systems, a network is a natural architectural choice. Besides of that, shorter time-to-market requires automation of the system synthesis from high-level specifications. This work presents a design flow to automate a Network on Chip generation targeting FPGA and dataflow applications. It also presents a hardware and software co-design flow for applications specified with the SystemC language as a programming model. We argue in this paper that efficient optimized NoC design is achievable through direct programming model primitives implementation in the NoC.

1. INTRODUCTION

Increasing transistor density, higher operating frequencies, short time-to-market and reduced product life cycle characterize today's semiconductor industry scenery. Under these conditions, designers are developing ICs (Integrated Circuit) that integrate complex heterogeneous functional elements into a single chip, known as a System on a Chip (SoC). Intellectual property cores, interconnection architectures and interfaces to peripheral devices compose a SoC. Traditional on-chip interconnection architectures, such as dedicated wires and shared busses, can be considered inefficient for future SoC. Dedicated wires present poor reusability and flexibility, while shared busses transmit only one word per clock cycle and offer limited scalability. According to ITRS estimation, in 2012, SoC will have hundreds of hardware blocks (called IP cores), operating at clock frequencies near 10 GHz [1]. In this context, a Network-on-Chip (NoC) appears as a possible solution for future on-chip interconnections due to the following features: (i) energy efficiency and reliability; (ii) scalability of bandwidth when compared to traditional bus architectures; (iii) reusability; (iv) distributed routing decisions. A NoC is an on-chip network composed of cores connected to routers, and routers interconnected by communication channels. Crossbus is the NoC solution developed by GIPSA Lab and it is a new approach for Network on Chip issues targeting FPGA devices.

Many networks on chip systems were developed based on different characteristics (interconnection type topology, commutation modes, routing algorithms, etc.) to present faster processing, good routing algorithms, quality of service, energy efficiency and reusability.

SPIN (Scalable Programmable Integrated Network) was develop by LIP6 in the year of 2000 [2]. This NoC is disposed in a topology of a fourth level tree, thus it has a dynamic routing algorithm. FAUST (Flexible Architecture of Unified System for Telecommunication) [3] was develop by CEA LETI, one of Grenoble's computer architecture laboratory. This NoC proposes a new architecture interconnection for fourth generation mobile phones. QNoC [4] has a 2D topology based on XY routing algorithm. Irregularities on its topology are allowed but it depends on the application. Priority levels can be attributed for packet routing depending on message type. STMicroeletronics developed Spidergon [5]. This NoC presents a particular topology, a ring where the transversal diagonals are linked, just like an octagon. Arteris is the first Network on Chip commercialized by Arteris, a French company created in 2003. It proposes exploration and implementation utilities [6]. Arteris accepts parameters for its creation. It is also compatible with a lot of academic and industrial standard interfaces. ×PIPES [7] can be generated by a model at RTL level using SystemC language. Application modeling is made by a task graph that shows the bandwidth between all communicating tasks. Ætheral [8] is a Network on Chip developed by Philips. It offers a design flow based on bandwidth and communication latency so IP blocks can automatically be placed around the NoC. µspider [9] is the result of a collaborative work between IETR and LESTER laboratory in Rennes and Lorient, France. It uses TDMA and virtual channels for quality of services, and the CoreConnect standard for communication interface.

All these NoC examples have two common properties: they were developed to be ASIC solutions and they were supposed to be general propose. With the growth of transistor density, now it is possible to have a NoC solution aiming FPGA devices. Besides of that, our Crossbus NoC is supposed to target dataflow applications like signal, image and video processing applications. SystemC is a good programming model for dataflow applications. We have selected SystemC as the programming model of the SoC based on Crossbus. So, based on target programming model and applications we can simplify and tune the architecture of the entire Network on Chip. Crossbus was deeply based on these two main ideas.

2. NOC CROSSBUS

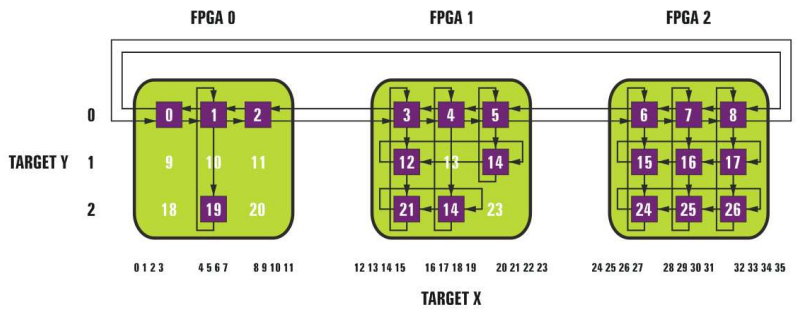

Figure 1 - Example of NoC Crossbus

An example of the Crossbus NoC is given in figure 1. Crossbus is composed by N FPGA devices, represented in green, where N is a natural non-null value. In this example we have N=3 because we have 3 FPGA devices linked as a ring. They are connected in a bidirectional ring topology. Each FPGA has a 2D torus router matrix of O x P routers, where O and P are two natural non-null values. In this example we have O=P=3, as we can see we have a square matrix of routers inside each FPGA device. Each router has Q IP blocks connections. In this case we have 4 connections for each router. 4 IP connections is the maximum allowed here because they will share a local bus. Also we can see in figure 1 that Crossbus uses fixed XY addresses for routing. It allows also irregularities, as shown by missing routers. Besides the fact that there are missing routers we notice that address numbering is always fixed, some addresses are thus lost. This regularity of routers arrangement and simplicity of router numbering let us automatically generate a Network on Chip from a XML description parsed from a SystemC application code. In this way it is possible to target automatically mutli-FPGA linked with high speed serial links. The serial links are incorporated in the NoC paradigm allowing a transparent implementation on multi-FPGA.

FPGA devices are linked as a ring by bidirectional high bandwidth links using Rocket IO protocol to communicate. Inside the FPGA device, routers communicate using Fast Simplex Links (FSL) standard, proposed by Xilinx. The routing algorithm used here is hardwired and deterministic. It is a kind of XY routing. The first row is called the main axis. Several restrictions must be followed:

- There is always a router on the main axis if there is at least one router in the same column.
- A router is fully connected if there are other routers in the same axis (row or column).

Thus if there is no direct link between two routers, it is always possible to route through the main axis. Two routers on the same column or row are routed directly. If two routers are not in the same FPGA, the routing is established through the main axis. If two routers are in the same FPGA but not on the same axis, the vertical routing is performed first. If we meet the same row before meeting the main axis, the rooting is performed directly on the row, else the rooting is performed through the main axis. Thus the main axis is more used than the others and then is doubled (bi-directional).

The torus topology is susceptible of deadlocks on router transfers. That is why a priority level is established between horizontal and vertical axis. The first row has the highest priority on transfers, the vertical axis have the second priority level, the horizontal axis have the third and local transfers have the least significant priority level. So every time that a low priority transfer (such as a local transfer) is being made and another transfer arrives by some higher priority link (such as vertical FSL), the local transfer must be stopped and the higher priority transfer will take place and then the less priority transfer can restart. This process avoids interlocking between data transfers. Also, this fully hardwired low overhead routing algorithm allows dynamic tasks creation and migration, for both software and hardware through partial dynamic reconfiguration of FPGA [10].

2. DESIGN FLOW

Designers increasingly rely on reusing of Intellectual Property and on raising the level of abstraction to respect SoC market characteristics. However, most hardware and embedded software codes are recoded manually from system level. This recoding step often results in new coding errors that must be identified and debugged. Thus, shorter time to market requires automation of the system synthesis from high level specifications. We propose a design flow intended to reduce the SoC design cost. This design flow unifies application hardware and software using a single high level language used as a programming model. It integrates hardware/software (HW/SW) generation tools and an automatic interface synthesis through a custom library of adapters.

(Step 1) First of all the application is modeled in SystemC [11]. The method presented in this paper focuses on SystemC models structurally described with sc_module, which model computation processes and either sc_signal, a primitive SystemC channel, or sc_fifo a basic SystemC channel acting as a First In First Out (FIFO) queue. The application is validated and profiled for exploration with ModelSim tool.

(Step 2) We manually separate the application in software and hardware parts. Our SysCellC tool generates the hardware IP blocks and the C or C ++ codes for embedded software.

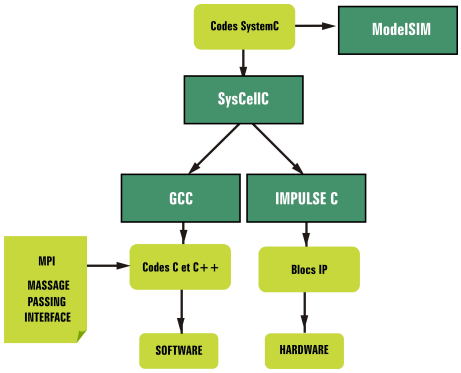

Figure 2 - Design flow (step 1 and step 2)

First SysCellC performs a parsing from synthesizable SystemC code to C code. Then the hardware blocks are generated by ImpulseC tool using synthesizable C codes. Software codes are compiled by Gcc for the embedded processors (MicroBlaze or PowerPC). Multitasking is available on the embedded processors with a lightweight static scheduler implementing the semantic of SystemC scheduler. For scheduling and communication library we use a subset of the Remote Memory Access MPI primitives: MPI_Init (initialization), MPI_Comm_Rank (ID recover), MPI_Barrier (synchronization), MPI_Put (DMA writing function) and MPI_Finalize (infinite loop). Up to now we have implemented only the sc_signal and sc_fifo communication channels. SystemC parsing is thus based on five basic primitives: Write_signal() and Write_fifo() that call MPI_Put to transfer data between two IP blocks; Read_signal() and Read_fifo() that copy a local memory buffer and Wait() that calls the lightweight scheduler. Also, as the SystemC programming model relies only on the RMA MPI library, the Network on Chip must only implement this subset. Thus for instance, no read (MPI_get) is implemented in the NoC. We illustrate the C generation process with a producer/consumer example as we can see in figure 3. The wait() calls are numbered with a parameter in order for the static scheduler to know where to come back in the thread.

```
class producer2 :
  public sc_module {

public:
  sc_out <int> a2;
  sc_in <bool> clk;

  SC_HAS_PROCESS(producer2);

  producer2(
    sc_module_name name) :
    sc_module(name) {

    SC_THREAD(main);
    sensitive_pos << clk;
  }

  void main() {
    int nb2;
    nb2=2;
    while(1) {

      if ((nb2%2)==0) {
        a2.write(nb2++);
        wait();
      }
      else {
        a2.write(nb2);
        nb2=nb2+3;
        wait();
      }
    }
  }
};
```

```
{
  int nb2;
  nb2=2;

  while(1) {
    if ((nb2%2)==0) {
      write_signal(
        P2,1,0,nb2++);
      wait(5);
    }
    else {
      write_signal(
        P2,1,0,nb2);
      nb2=nb2+3;
      wait(6);
    }
  }
}
```

SystemC Code | Generated C Code

Figure 3 - from SystemC code to C code

(Step 3) The NoC Crossbus must be generated according to a given topology. Routers, links between routers, local memories, memory controllers, hardware IP blocks and microprocessors (Microblazes) must be declared in a Microprocessor Hardware Specification (MHS) file [12]. When we synthesize the NoC Crossbus and the application IPs we generate a netlist (NGD) file.

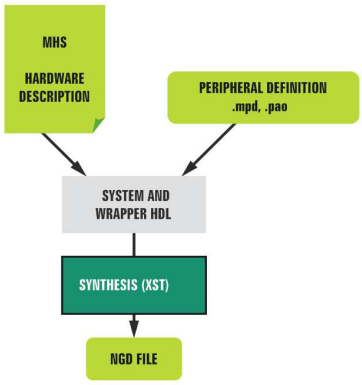

Figure 4 - Design flow (step 3)

(Step 4) The Microprocessor Software Specification (MSS) file has the entire system definitions concerning the compile settings and software drivers for all IP. All the application C codes must be compiled and linked with library files to produce an Executable Linked Format (ELF) file.

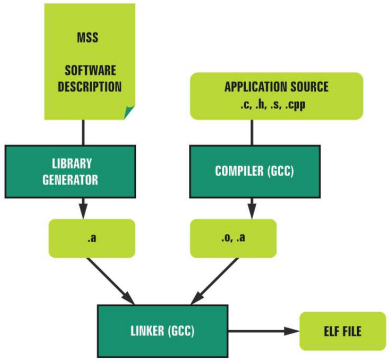

Figure 5 - Design flow (step 4)

(Step 5) Once we have the hardware and software files we can generate a bitstream file and download it to the FPGA device(s).
This fully automated flow allows the exploration in the loop through scripts. It is thus possible to explore different partitioning and topologies through direct execution on a multi-FPGA platform.

4. CASE STUDY

In order to evaluate the proposed design flow, we tested a producer-consumer application. We modeled a 16-threads application, with 8 producers and 8 consumers, a data flow application where every producer writes a packet of N 32-bit words in a channel connected with all the consumers. In this manner we can validate the entire system. The NoC Crossbus generated to support this application was a system with one FPGA, 4 routers,

each router connected to 2 Microblaze (MB) processors [13]. Each processor executed two threads, one producer and one consumer. Also a CDMA application with 7 threads and 7 processors was also used to validate the NoC Crossbus.

5. RESULTS

Table 1 (see below) shows the area cost for 32-bit routers with 1, 2 and 4 IP connections.The router cost is almost the cost of a MicroBlaze with the same frequency. We can thus build a large SoC with tens or hundreds of processors implemented on multi-FPGAs. The router cost presented here includes the network interface and IP wrapper adaptation. We are compatible with the Xilinx interface standards (FSL for FIFOs and LMB for memories). As a comparison, the Hermes NoC [14,15] which is a 8-bit lightweight NoC targeting FPGAs has a cost of 278 slices per router and 567 slices with the network interface. This can be compared to our 1494 slices for a 32-bit router with network interface which is slightly better. The formula below can be used to compute the total time T to deliver a set of packets:

$$T = (ST + (NF+2)*TF + NH)*NP + (NP-1)$$

ST = number of clock cycles to generate the header, 4 in the Crossbus NoC; NF = number of flits, 16 for instance + 2 for the header and end of packet; TF = 1, each flit spends one clock cycle to be transmitted to the next switch; NH = number of hops: each time a packet crosses a switch one clock cycle is needed; NP = number of packets. Table 2 shows the timing (in clock cycles) results for the simulation of the producer-consumer application for packets with 4, 8 and 16 words of 32 bits. As we can see below, the Crossbus NoC has an overhead to start to work (System initialization), but since it is working the other operations consume almost the same time.

Table 1 – Router area cost

BLOCK IP	I0s	CLB SLICES	FREQUENCY (MHz)
ROUTER with 1 MB	497	1494	173.3
ROUTER with 2 MB	582	1639	178.6
ROUTER with 4 MB	743	1994	203.3

Table 2 – Timing results for producer consumer example

ACTION	4 words	8 words	16 words
System initialization	1385	2281	4057
MPI_init()	426	426	426
MPI_Comm_Rank()	1	1	1
Scheduling	60	60	60
write_signal	86	133	175
MPI_Barrier()	1	1	1

System initialization depends on packet size, because it represents the first step of processing where all static variables are set to zero. Static variables represent SystemC communication channels linking components. MPI_Comm_Rank and MPI_Barrier take one clock cycle to execute because they can be directly translated to just one assembly instruction. Also the synchronization barrier is hardwired with combinatorial logic. MPI_Init() and Scheduling times are equal for all the examples because they don't depend on packet size, they depend only on the number of channels of communication. In all these examples we have 8 communication channels, one for each producer. Write_signal primitive represents the data transfer between processing units using sc_signal write() primitive. The SystemC application programming model is defined based exclusively on writes (RMA). Write_signal calls MPI_Put() function to write a memory block from a processing unit to another one. The results presented here show the efficient implementation of SystemC mechanisms thanks to a NoC design dedicated to SystemC programming model. This optimized NoC design allows FPGA targeting where area constraints are strong.

6. CONCLUSIONS

The Network on Chip became a reality and their importance in the market is growing, especially for embedded systems and telecommunications. Crossbus NoC proposes a new solution for Network on Chip issues. The results presented here show the efficient implementation of SystemC mechanisms thanks to a NoC design dedicated to SystemC programming model. This optimized NoC design allows FPGA targeting where area constraints are strong. The next step of this research project is to adapt a MPEG2 application on the NoC.

7. REFERENCES

[1] ITRS http://www.systemc.org/

[2] Guerrier P. and A. Greiner, « A Generic Architecture for on-chip Packet Switched Iterconnections ». In Design Automation and Test in Europe (DATE), Paris, Fance, pages 250–256, 2000.

[3] Lemaire R, Lattard D and Jerraya A, « Evaluation des performances de transferts de données sur un NoC régulé par un mécanisme de controle de flux ». JNRDM05, May 2005.

[4] Evgeny Bolotin, Israel Cidon, Ran Ginosar and Avinoam Kolodny, « QNoC : QoS architecture and design process for network on chip ». J. Syst. Archit., vol. 50, no2-3, pages 105–128, 2004.

[5] Faraydon Karim, Anh Nguyen and Sujit Dey, « An Interconnect Architecture for Networking Systems on Chips ». IEEE Micro, vol. 22, no5, pages 36–45, 2002.

[6] « ARTERIS announces STMicroeletronics use of NoC for next generation wireless infrastructure platform ». Mars 15, 2006.

[7] Matteo Dall'Osso, Gianluca Biccari, Luca Giovannini, Davide Bertozzi and Luca Benini, « xPIPES: a Latency Insensitive Parameterized Network-on-chip Architecture For Multi-Processor SoCs ». In ICCD '03: Proceedings of the 21st International Conference on Computer Design, (Washington, DC, USA), page 536, IEEE Computer Society, 2003.

[8] John Dielissen, Andrei Radulescu, Kees Goossens and Edwin Rijpkema, « Concepts and Implementation of the Philips Network-on-Chip ». IPSOC, 2003.

[9] S. Evain, J. P. Diguet and D. Houzet, « µSpider : A CAD Tool for Efficient NoC Design ». In IEEE NORCHIP 2004, 8-9 November 2004.

[10] F. Berthelot, F. Nouvel, D. Houzet, « A Flexible system level design methodology targeting run-time reconfigurable FPGAs », EURASIP Journal on Embedded Systems, vol. 2008.

[11] SystemC, http://www.systemc.org/

[12] Xilinx. « Platform Specification Format Reference Manual », June 2006. Software version 8.2i.

[13] Xilinx. « MicroBlaze Processor Reference Guide ». Juin 2006.

[14] Moraes, F. « Hermes: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip ». Integration the VLSI Journal, 38(1), Octobre, 2004, pp. 69-93.

[15] Mello, A. « Virtual Channels in Networks on Chip: Implementation and Evaluation on Hermes NoC ». 18th SBCCI, pp. 178-183.