RASoC: A Router Soft-Core for Networks-on-Chip

Cesar Albenes Zeferino
UNIVALI – CTTMar
Rua Uruguai, 458
C.P. 360 – CEP 88302-202
Itajaí – SC – BRAZIL
zeferino@inf.univali.br

Márcio Eduardo Kreutz
UFRGS – II – PPGC
Av. Bento Gonçalves, 9500
C.P. 15064 – CEP 91501-970
Porto Alegre – RS – BRAZIL
kreutz@inf.ufrgs.br

Altamiro Amadeu Susin
UFRGS – II – PPGC
Av. Bento Gonçalves, 9500
C.P. 15064 – CEP 91501-970
Porto Alegre – RS – BRAZIL
susin@eletro.ufrgs.br

Abstract

The building block of a Network-on-Chip (NoCs) is its router. It is responsible to switch the channels which forward the messages exchanged by the cores attached to the NoC, and the costs and performance of the NoC strongly depends on the router architecture. In this paper, we present RASoC, a router architecture intended to be used in the building of low area overhead NoCs for embedded systems. The difference among RASoC and current routers relies on its implementation as a parameterized VHDL model, which improve the reuse of RASoC in the synthesis of NoCs with different sizes, and allows the tuning of the NoC parameters in order to meet the requirements of the target application. The paper presents details of RASoC architecture, the structure of the VHDL model and some experimental results which show the scalability of the soft-core and its costs.

Keywords: Systems-on-Chip. On-Chip Networks. FPGA.

1. Introduction

Networks-on-Chip (or NoCs) [1] promise to be the better approach, maybe the only one, that will meet the communication requirements of future Systems-on-Chip (SoCs). They are based on the concepts adopted on the building of interconnection networks for parallel computers. A NoC is composed by a set of routers and point-to-point channels interconnecting routers in a structured way. Each router has a set of ports which are used to connect routers with its neighbors and with the processing cores of the system (i.e. scalar processors, DSPs, controllers, memories, and others). The ports used to connect the processing cores are named local ports or terminals (in the network point-of-view).

A NoC can be described by its topology and by the strategies used for routing, flow control, switching, arbitration and buffering. The *network topology* is the arrangement of nodes and channels into a graph. *Routing* determines how a message chooses a path in this graph, while *flow control* deals with the allocation of channels and buffers to a message as it traverses this path. *Switching* is the mechanism that removes data from an input channel and places it on an output channel, while *arbitration* is responsible to schedule the use of channels and buffers by the messages. Finally, *buffering* defines the approach used to store messages while they cannot be scheduled by the router arbitration circuits.

In the point-of-view of a router, routing is the mechanism that chooses an output channel for a message arriving at an input channel. Flow control is the mechanism that regulates the traffic of incoming and outgoing messages at the channels. Switching defines how a message passes through the router, while arbitration decides when this switching can be done. Finally, buffering determines how and where the messages will stay while they are waiting to leave the router.

The router of a NoC can be implemented in a centralized or a distributed way. In the first one, the router is composed by: (i) a crossbar; (ii) a collection of FIFO buffers and link controllers for buffering and flow control; and (iii) centralized controllers for routing, arbitration and switching. In the distributed approach, both the crossbar and the controllers are split into modules distributed among the input and output channels of the ports at the router interface, as it is illustrated in Figure 1.

Typically, the distributed approach is the preferred in the building of routers for NoCs, and some examples include RSPIN [2], the router architecture presented by Dally and Towles [3], and the router of the NoC CLICHÉ [4].

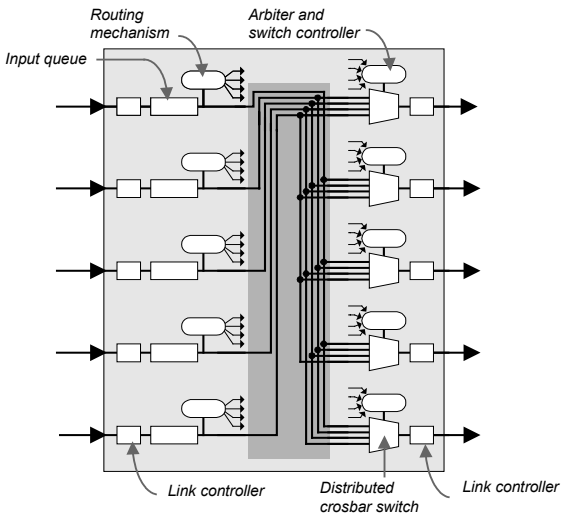

Figure 1. A distributed router architecture.

This paper presents the architecture of a new router named RASoC (*Router Architecture for SoC*), which was designed to be used in the synthesis of low area overhead NoCs for embedded SoCs. RASoC is built on a distributed way. Its architecture is based on the wormhole switching approach and it uses a deterministic source-based routing algorithm (the XY) [5]. Also, it applies the handshake protocol for link flow control, and uses round-robin arbitration and input buffering.

The difference among RASoC and current routers (eg. RSPIN [2]) relies on its implementation as a parameterized VHDL model, which improves the reuse of RASoC in the synthesis of NoCs with different sizes, and allows the tuning of the NoC parameters in order to meet the requirements of the target application. The paper presents details of RASoC architecture, the structure of the VHDL model and some synthesis results.

This paper is structured as follows. Section 2 gives a general overview about the RASoC's interface and organization based in two kinds of modules: input channel and output channel. Section 3 shows the structure of the VHDL model, and Section 4 presents some synthesis results. Concluding, we give some final remarks.

2. RASoC Architecture

Externally, RASoC is a routing switch with up to five bi-directional ports (Figure 2). These ports are named *L* (Local), *N* (North), *E* (East), *S* (South) and *W* (West). Depending on the position of a RASoC instance on the NoC, and on the network topology, one or two of them need not be implemented, reducing the network area.

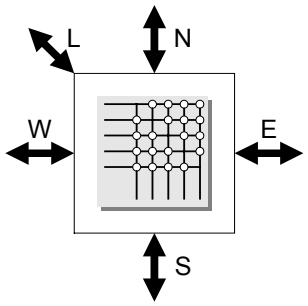

Figure 2. The interface of RASoC

RASoC ports include two unidirectional opposite channels (Figure 3), each one with its data, framing and flow control signals. Each channel includes *n* bits for data and two bits for packet framing: *bop* (begin-of-packet) is set only at the packet header, and *eop* (end-of-packet) is set just in the last payload word, which is the packet trailer. The *n* data bits can be extended to include *Higher Level Protocol* (HLP) signals, like the ones typically used for data integrity control (parity and error). Typical values for *n* are 8, 16 or 32 bits (not including HLP signals). Flow control bits are used to validate data at the channel (*val*) and to acknowledge the received data (*ack*).

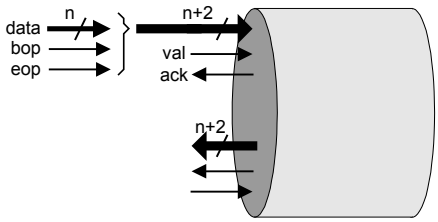

Figure 3. The channels of a port.

Internally, RASoC is implemented in a distributed way and it is composed by instances of two kinds of modules: input channel (*in*) and output channel (*out*). The following subsections describe the internal organization of these modules.

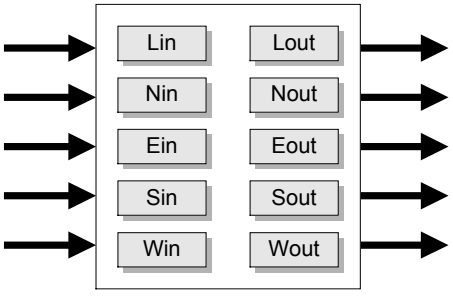

Figure 4. The internal organization of RASoC.

2.1. The Input Channel Module

The input channel module is represented in Figure 5. It is composed by four blocks named *Input Flow Controller* (IFC), *Input Buffer* (IB), *Input Controller* (IC) and *Input Read Switch* (IRS). In the RASoC terminology, the names of the signals at the interface of the router include the prefix "*in_*", for the input channel modules, and "*out_*", for the output ones. All the internal signals connecting the input and output channel modules use the prefix "*x_*".

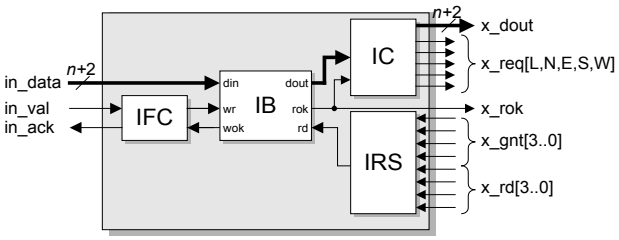

Figure 5. The input channel module.

The IFC block includes the logic that performs the translation between the handshake and the FIFO flow control protocols. It just implements an AND gate in order to set the output *in_ack* when both *in_val* and *wok* equal 1.

The IB block is a FIFO buffer and it is responsible to store *flits* (the *flow control units*) of the incoming packets while they cannot be forwarded to an output channel. In the wormhole switching networks, a flit is the small unit over which the flow control is performed. In RASoC, a flit equals the physical channel width, and each FIFO has p positions to store up to p (n+2)-bit wide flits.

The IC block performs the routing function. It detects the presence of a header at the IB block output, analyses the *Routing Information Bits* (RIB) included in the header, runs the routing algorithm to select an output channel, emits a request to the selected output channel, and, finally, updates the routing information in the header to take into account the performed routing. Its outputs include the updated header and the requests for the output channels. Since the input channel module is a generic model for all the instances in a network, it includes the requests for all the output channels, but not all of them can be actually requested by each input channel instance because it is not allowed to an input channel to request the output channel of its own port. For instance, *Lin* can request *Nout*, *Eout*, *Sout* or *Eout*, but never *Lout*.

The IRS block receives four pairs of *x_rd* - *x_gnt* signals from each output channel module, and connects the granted read command to the *rd* input of the IB block interface.

Basically, when a packet header arrives at an input channel, it is firstly stored in the IB block. At the moment it reaches the output of these buffer, the IC block runs the routing algorithm and emits a request to the selected output channel. While a grant is not received from the requested output channel, the remaining flits of the packet are stored in the buffer. However, if the packet length is greater than the buffer depth, the IFC block assures that no additional flit will be received while the buffer is full. The flits that can not be stored in the buffer are stalled in the previous buffers in the packet path, and the channels in this path remains reserved for such packet. When the grant for the blocked packet is received, the header is forwarded to the requested output channel, and the payload flits follow the header in a pipelined fashion.

2.2. The Output Channel Module

The internal organization of the output channel module is shown in Figure 6. It is composed by four blocks named *Output Controller* (OC), *Output Data Switch* (ODS), *Output Read Switch* (ORS) and *Output Flow Controller* (OFC).

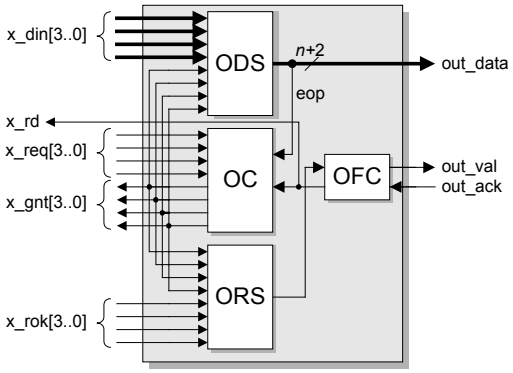

Figure 6. The output channel module.

The OC block, runs a round-robin algorithm to select one of the requests emitted by the input channels. After that, it sets the grant line to the selected request, commanding the ODS and ORS blocks to switch. These ones connect the *x_din* and *x_rok* signals of the selected input channel to the external output channel interface. However, before be connected to *out_val,* the *x_rok* signal goes through the OFC block. Since there is no functional difference between the handshake and the FIFO protocols at the sender side, the OFC block signal just implements wires connecting the selected *x_rok* to *out_val*, and *out_ack* to *x_rd*. If an another 2-wire based

flow control approach is used, this block can be easily replaced to implement the required logic (eg. an up/down counter in a credit-based strategy). The OC block also monitors *eop* and *x_rd* signals to determine when the last packet flit (the trailer) is delivered in order to cancel the established connection.

3. The RASoC Soft-Core

A soft-core for RASoC was implemented in VHDL using the hierarchy represented in Figure 7. The top-level entity, named *rasoc*, has three generic parameters, *n*, *m* and *p*, which define the data channel width, the width of the routing information in the header (RIB), and the FIFO depth, respectively. By tuning such parameters, one can synthesize routers with different cost and performance ratios. The lower-level entities receive from the higher-level ones the parameters they need to generate their architectures with the required dimensions. The acronyms in the names of the bottom level entities in Figure 7 represent the actual name of each entity (eg. IFC is implemented by the *input_flow_controller* entity).

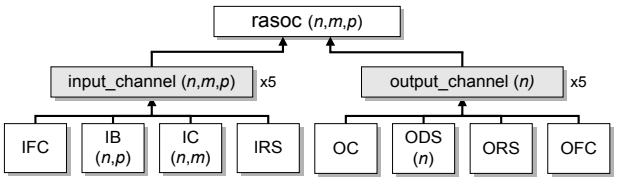

Figure 7. Hierarchy of entities in the model.

The architecture of the bottom level entities was defined in order to meet the constraints of the technology available for synthesis: Altera FPGAs. These devices have no internal tri-states and use LUTs (Look-Up Tables) to implement multiplexers (Figure 8). For the FIFOs, two approaches were considered. In the first one, they were described as *p*-deep, (*n*+2)-wide shift registers with an output multiplexer to selected the FIFO head (Figure 9). In the second one, it was used an Altera's parameterized megafunction that is automatically mapped onto the Embedded Array Blocks (EABs) available for synthesis of memories in the FPGAs. We consider that the first approach must be used when there is not enough embedded memory bits for both processing cores and routers, and the available ones must be reserved for the processing cores.

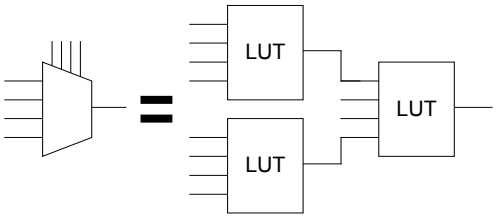

Figure 8. A LUT-based 4x1 multiplexer.

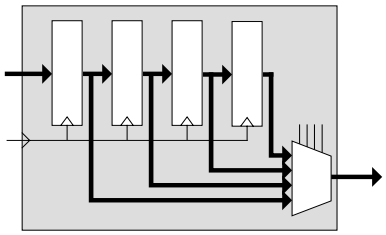

Figure 9. A 4-flit FF-based FIFO buffer.

4. Experimental Results

The RASoC VHDL model was synthesized in an FPGA of the family Altera FLEX 10KE by using Altera Quartus II 3.0. By changing only the parameters *n* (the data channel width) and *p* (the FIFO depth), and doing *m* (the width of the RIB field) fixed in 8 bits, we obtained the results presented below. The selected device was EPF10K200SFC672-1, a 200-Kgate FPGA with 9,984 LCs and 96 Kbits of RAM included in 24 EABs (each one capable to synthesize a 4-Kbit memory).

In Table 1, we compare the costs between the two approaches for FIFOs implementation. They are shown the number of logic cells (LC), flip-flops (Reg) and memory bits (Mem) consumed in each approach for $n = 8$, 16 and 32 bits, and for $p = 2$ and 4 flits. Each position in the buffer is n+2 bit wide).

Table 1. Costs of buffers.

		2 flits			4 flits		
		LC	**Reg**	**Mem**	**LC**	**Reg**	**Mem**
FF-based	**8-bit**	35	22	0	76	43	0
	16-bit	59	38	0	124	75	0
	32-bit	107	70	0	220	139	0
EAB-based	**8-bit**	13	5	20	19	8	40
	16-bit	13	5	36	19	8	72
	32-bit	13	5	68	19	8	136

As one can see, in the FF-based approach, the number of LCs increases with both the depth and the width of the FIFO due to the multiplexer needed to select the head of the FIFO (Figure 9). On the other hand, in the EAB-based approach, the numbers of LCs is smaller and increases only with the FIFO depth. Considering the registers, the first approach uses flip-flop to implement the memory elements, and the costs increases in the two directions. In the second approach, registers are used only for the pointers that select the positions to be read or write, and their costs are independent of the FIFO width. Finally, only the EAB-based approach uses the embedded RAM and the number of used memory bits used $(n+2)\times p$.

Table 2 shows results for the synthesis of 5-port routers based on the two approaches discussed above for the FIFO implementation. As one can see, the EAB-based approach spends less logic cells and flip-flops due to the use of embedded memory bits. Furthermore, the number of registers is fixed for a given FIFO depth and number of LCs grows mainly when the channels become larger due to the multiplexers related with switching in the OFC block.

Table 2. Costs of RASoC.

		2 flits			4 flits		
		LC	**Reg**	**Mem**	**LC**	**Reg**	**Mem**
FF-based	**8-bit**	570	160	0	795	265	0
	16-bit	770	240	0	1115	425	0
	32-bit	1173	400	0	1830	745	0
EAB-based	**8-bit**	460	75	100	486	90	200
	16-bit	540	75	180	566	90	360
	32-bit	700	75	340	726	90	680

It is important to highlight that the largest configuration in the EAB-based approach uses less than 0.7% of the memory bits available in the target FPGA.

The maximum operating frequency is about 56,7 MHz for the EAB-based approach (by taking the average of the performance measurements for the different configurations in Table 2). Considering the FF-based approach, the maximum operating frequency is about 64 MHz for 2-flit buffers, but decreases to 55,8 MHz due to the multiplexer at the outputs of the buffers.

In Table 4, we show the costs of the bottom-level entities in the synthesis of a 32-bit, 4-flit, EAB-based router. We highlight the more expensive ones. For instance, the five output controllers are responsible for 28% of the LCs consumed by the router.

Table 3. Costs of bottom-level entities.

Entities (5x)	**LC**	**Reg**	**Mem**
IRS - Input Read Switch	1%	0%	0%
IC - Input Controller	**8%**	0%	0%
IB - Input Buffer	**12%**	**44%**	**100%**
IFC - Input Flow Controller	1%	0%	0%
OFC - Output Flow Controller	0%	0%	0%
ORS - Output Rok Switch	1%	0%	0%
ODS - Output Data Switch	**49%**	0%	0%
OC - Output Controller	**28%**	**56%**	0%

By observing Table 3, we can see that the only blocks that could be optimized in order to reduce the router costs are the controllers, because there is no way to reduce the costs of the switches in the FPGA.

To get some idea of the costs of RASoC in comparison with some processing core, we show in Table 4 the costs (in number of LCs) of an 8/16-bit ASIP microcontroller (named FemtoJava) synthesized in an Altera FLEX 10K FPGA [6]. Comparing these costs with the ones shown in Table 2 (for 8- and 16-bit configurations), one can see that the costs of RASoC vary from 31% to 56% of the costs of FemtoJava.

Table 4. Number of LCs for FemtoJava.

Data width	**LC**
8 bits	1481
16 bits	1979

5. Conclusions

In this paper, we presented the architecture of RASoC, a router for the building of Networks-on-Chip. Results shown that the VHDL soft-core developed for RASoC allows the automatic building of instances with different sizes. Such architecture has been used in the building of networks-on-chip and in researches targeting different issues in the NoC domain: design methodologies and SoC test planning. Presently, we are working to develop cheaper versions for the router components in order to reduce RASoC costs. Also, we are modeling RASoC in CASS (Cycle-Accurate System Simulator) [7] in order to compare the performance of RASoC-based NoCs with the ones of SPIN [2] and PI-Bus [8], by using the methodology applied in [9].

6. Acknowledgments

This work was supported by CNPq.

7. References

[1] L. Benini and G. De Micheli, “Networks on Chips: a New SOC Paradigm”, *IEEE Computer*, Jan. 2002, pp.70-78.

[2] P. Guerrier and A. Greiner, “A Generic Architecture for on-Chip Packet-Switched Interconnections”, *DATE’2000*, IEEE CS Press, 2000. pp.250-256.

[3] W. J. Dally and B. Towles, “Route Packets, Not Wires: On-Chip Interconnection Networks”, *DAC’2001*, ACM Press, 2001. pp.684-689.

[4] S. Kumar et al., “A Network on Chip Architecture and Design Methodology”, *Annual Symposium on VLSI’2002*, IEEE CS Press, 2002. pp.105-112.

[5] J. Duato et al., *Interconnection Networks: an Engineering Approach*, IEEE CS Press, 1997. 515p.

[6] J. C. B. Mattos and L. Carro. “Efficient Architecture for FPGA-based Microcontrollers”. *ISCAS’2002*, IEEE CS Press, v.5, 2002. p.805-808.

[7] F. Petrot et al. “Cycle-Precise Core Based Hardware/ Software System Simulation with Predictable Event Propagation”, *EUROMICRO’1997*, IEEE CS Press, 1997. p.182-187.

[8] Siemens. “OMI 324: PI-Bus – Ver.0.3d”, Siemens AG, 1994. 35p.

[9] A. Andriahantenaina et al. “SPIN: A Scalable, Packet Switched On-Chip Micro-Network”. *DATE’2003*, IEEE CS Press, 2003. p.70-73.