Journal of Computer Science 1 (3): 429-436, 2005
ISSN 1549-3636

Asynchronous NoC Router Design

[1]Sami Badrouchi, [1]Abdelkrim Zitouni, [2]Kholdoun Torki and [1]Rached Tourki
[1]Laboratoire EμE, Faculté des Sciences de Monastir, 5019 Monastir, Tunisie
[2]Laboratoire TIMA, INPG, 46 Avenue Félix Viallet 38031, Grenoble Cédex, France

Abstract: The Quality of Service Network on Chip (QNoC) is the most perferment solution that provides low latency transfers and power efficient System on Chip (SoC) interconnect. This study presents an asynchronous NoC router, for use in 2-D mesh-connected networks. It comprises multiple interconnected input and output ports and dynamic arbitration mechanisms that resolve any output port conflicts based on the messages priorities. The proposed router protocol and its asynchronous modeling are based on the Speed Independent State Transition Graph (STG) model. The generated STG are transformed into VHDL data flow descriptions and the low level implementation is based onto a parameterized library. This library integrates the popular asynchronous SI modules such as C-element, Q-element, fairly arbiter, etc. The device is implemented in 0.35 μm CMOS technology and its performance is compared with a synchronous router of the same functionality. The asynchronous router enables a higher data rate and a comparable silicon area.

Keywords: System on Chip, Network on Chip, Asynchronous Router, Asynchronous Arbiter

INTRODUCTION

Since, data synchronization problems arise in multi-clock domain SoC and operating clocked interconnects becomes increasingly more difficult, large SoC are treated as Globally Asynchronous Locally Synchronous (GALS) systems, calling for suitable interconnects beyond conventional synchronous buses. GALS paradigm not only avoids the problem of clock skew but also leads to lower power consumption. NoC are advocated as a solution for the SoC interconnect problem [1].

Communication between system modules is done by handshaking, where signals are exchanged on the control path in order to arrange the exchange of data. The packet router may use a bundled data that are controlled and transmitted asynchronously [2]. The shared bus solution used for connecting the functional units by the commercial SoC [3-6] is not a suitable NoC interconnect since it can provide limited connectivity. The Networks on Chip (NoC) are a better alternative than the classic architectures based on busses. Every node is the point of passage of several plots coming of different claimants.

Several works has been focused on the synchronous NoC by using the 2-D mesh, torus, fat tree and hierarchical topologies [7-12]. The cut-trough and packet switching techniques have been used for the interconnect design. Synchronous NoC routers supporting virtual channels are described in [13, 14]. Other synchronous routers are discussed in [15]. A synchronous 5-port router provides for two service levels (best effort and guaranteed throughput) has been presented in [16].

Only a few works have been focused on the design of asynchronous NoC. Synchronous routers using round-robin arbitration and supporting asynchronous interconnect are presented in [17, 18], though synchronization issues are ignored. CHAIN [19, 20] has presented flexible asynchronous interconnection NoC structures by using a 1-of-4 encoding. A design and implementation of a fast, low-power and multi service levels NoC has been presented in [21]. NoC wrappers and synchronization issues are discussed in [22-27].

The main contribution of the proposed approach consists on synthesising QNoC router to be integrated into a 2D mesh NoC. The proposed router integrates a centralized Speed Independent dynamic arbiter based on messages priorities. STG are used as specification models and generalized C-elements are used for gate level implementation.

In section 2, we present the asynchronous library that is used as data base for the router specification. The C-element gates and the arbiter synthesis technique are outlined in this section. Section 3 describes the router design. In section 4, we present the experiment results and we compare the performance of the asynchronous router with a synchronous router of the same functionality. Section 5 concludes the study.

ASYNCHRONOUS LIBRARY

The first step in implementing the router was to construct a library of asynchronous components. Correct, hazard-free CMOS implementations were explored from a number of literature sources. To design the complexes asynchronous components such as arbitration modules, we have used the STG as input specification model and the C-element gate for the low level implementation. Starting from the specification step, we extract the C-elements production rules and transform them into dataflow synthesizable VHDL

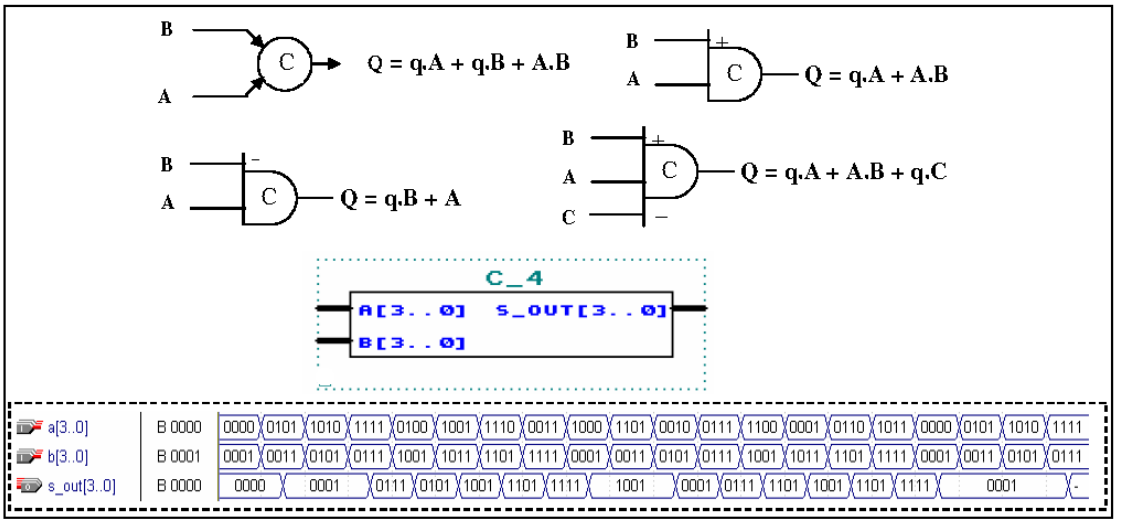


Fig. 1: Different forms of the Muller C-Elements and a Simulation Result of a 4x2 Inputs C-4 Module

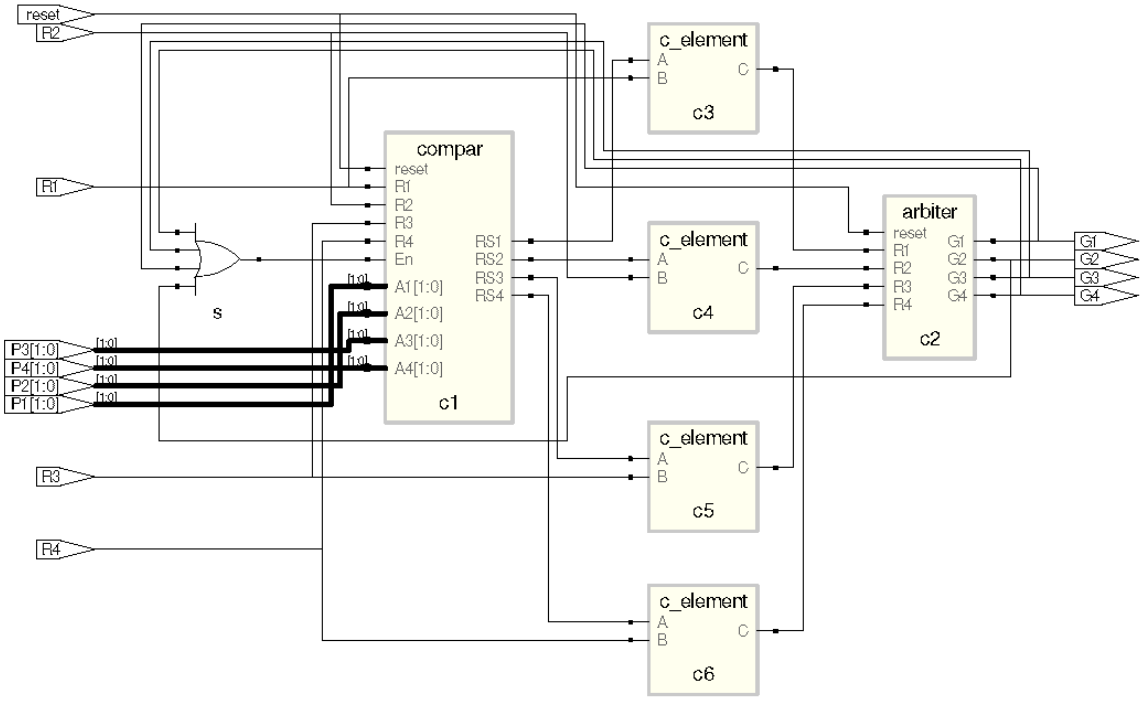


Fig. 2: A Dynamic Asynchronous Arbiter with Three Requesters

description. Many standard asynchronous components such as Fire arbiter, Q-element, Select and Decision wait are stored in this library. The components that we have developed to be used in the NoC router are described below.

Muller C-element: The symmetric Muller C-element is one of the main building blocks of SI asynchronous circuits. It performs the function of synchronisation. The device waits for an event on each of its inputs. When this occurs, an event is produced on the output. The device then waits for the next pair of input events. If two events arrive on the same input, without an event on the other input, then the events will cancel each other out and the circuit will again be waiting for an event on each input, as before. C-elements can be also constructed to respond to transitions in only one direction on particular inputs. These forms of C-element are known as generalized C-elements. Figure 1 presents the different forms of the Muller C-elements.

These symmetric C-elements have been regrouped by four, in "C-4" modules, for obvious reasons of convenience for reuse in the NoC router as explained

later. For every couple of entries we found the functionality of a two input C-element. The simulation of the quadruple C-element (Fig. 1), give results in conformity with those expected. The output remains at zero in the beginning of the simulation because the entries are either all two to zero, either different.

Dynamic Arbiter: The dynamic priority discipline is different form the static one as it uses dedicated priority ports to receive priority information from requesters. For each access, the requester that has the highest priority will gain the access. In [28], when there are many requesters that have identical priority orders, the requester that has the highest indices is conventionally granted the access. In our approach, when there are many requesters that could have the same priority orders, requesters are granted the access in a round-robin scheme. Each requester has a priority which is shifted in a circular way each time one of the requesters gets access to the resource. After being serviced, a requester receives the lowest priority. Such scheme is characterised by the absence of any absolute order in which requesters are being granted access to a resource.

Arbiter Architecture: The dynamic arbiter architecture is composed by two modules, a priority comparator and a round-robin arbiter interconnected by n C-elements (n represents the number of requesters) and an OR gate as shown by Fig. 2 in the case of an arbiter with four requesters. After receiving the request signals from the requesters, the priority comparator stores the priority values from the priority ports of the active requests and compares them. After the comparison step, the comparator sends out a set of internal signals that correspond to the requesters that have the highest priorities. After being combined with four C-elements, the output signals are transmitted to the arbiter module. Thus, only the requesters that have requested the access and that have been selected by the comparator module will be treated by the arbiter. If we have only one requester that has been selected by the comparator it will be granted the access automatically by the arbiter and the priority list is shifted in a circular way. When we have two or three or four requesters that have the same actual priorities orders, then the requester that has the last highest priority order will grant the access and the priority list is shifted in a circular way. We notice that the outputs of the C-elements that enter to the arbiter module will be activated only if the requesters have requested the access and they are selected by the comparator. We notice that if a requester has gained the access (its grant is activated) the comparator module will deactivate the generated internal signals. This scheme is implemented by an enable (En) signal that is generated by an OR gate with four inputs (G1, G2, G3, G4). The outputs of the C-elements that enter to the arbiter still high until the active requests become low. Also the comparator will be activated again only if the enable signal becomes active (the actual requester has released the bus).

Round Robin Arbiter Synthesis: The STG and the C-element based implementation of an asynchronous round-robin arbiter with two requesters are presented in Fig. 3. This graph is formed by two identical fixed priority sub-graphs where R1 have the highest priority in the first sub-graph (sub-graph (12)) and R2 have the highest priority in the second (sub-graph (21)). When the requester R1 has gained the access to the shared output port, the sub-graph (12) becomes inactive and the sub-graph (21) is activated. When the requester R2 has gained the access to the output port, the sub-graph (21) becomes inactive and the sub-graph (12) is activated.

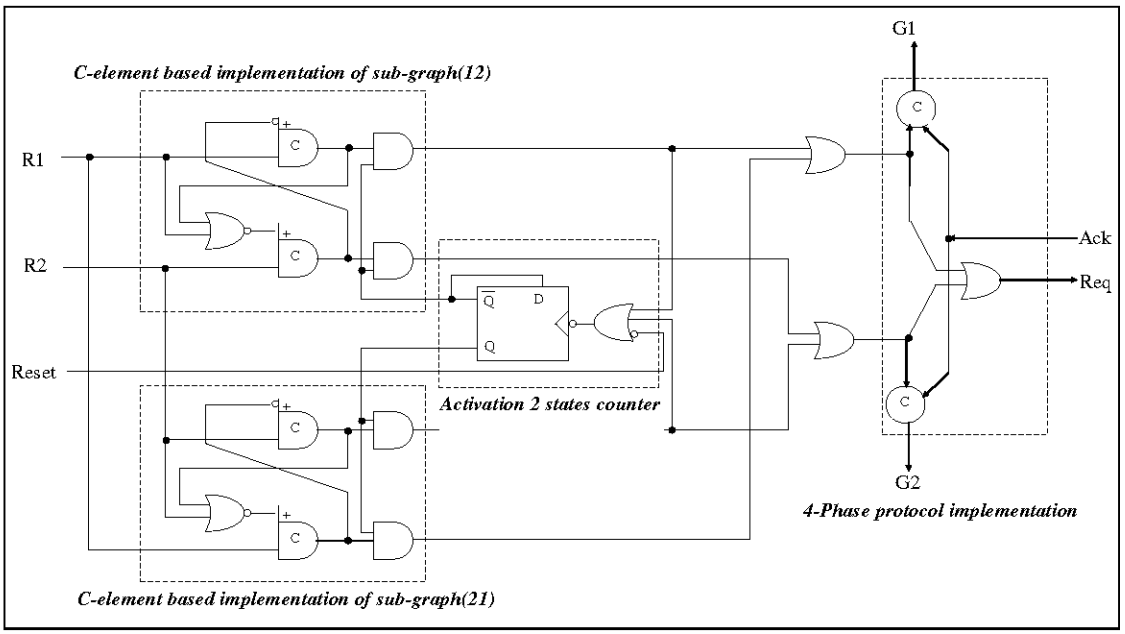
The implementation idea is to synthesise each sub-graph with a fixed priority arbiter with two requesters. If the arbiter receives two requests on both R1 and R2 concurrently, the generalized C-element based blocs selects the highest requester to be granted by asserting one of its two output wires. This in turn causes the resource to be requested by asserting Req. When the resource has completed its operation, it asserts Ack and acknowledgment is forwarded to the requesting module that was chosen depending on its priority order.

The activation and the deactivation of each sub-graph are performed by the outputs of a two states counter. The rising edge of the counter CLK is generated when R1 or R2 has terminated its resource access by detecting the falling edge of G1 or G2. At the beginning we assume that the requester R1 have the highest priority. This requester is activated by falling edge of the Reset signal. The proposed arbitration structure can be easily extended into asynchronous arbiter with n requesters.

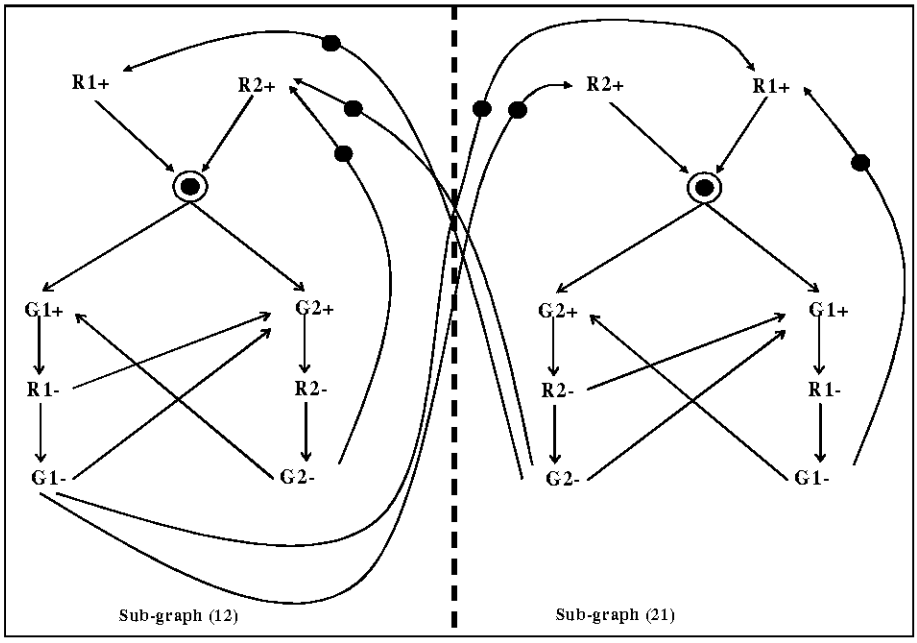
ROUTER DESIGN

We have adopted an asynchronous router with five input/output ports (North, East, Local, South and West), having each a bi-directional exchange bus suitable for 2D mesh NoC architecture. In such architecture each router is connected to four neighbours and a Processing Element (PE) as presented in Fig. 4. All inter-modules communications are carried out in packets. We assume that a packet coming through an input port is not returned to the output port of the same interface. Packets are partitioned into small flits, which are sent through the NoC using wormhole routing.

Header Fields: To support varying communication requirements, each packet carries priority information, related to data communication requirements. Each packet consists of three types of flits: a header flit, body flits and a tail flit, indicating End-of-Packet (EOP).



(a)



(b)

Fig 3: (a) C-element Based Implementation of an Asynchronous Round-Robin Arbiter with Two Requesters and (b) STG of Round-Robin Asynchronous Arbiter with Two Requesters

Every port sends a byte (header) that contains the address of its target port, the number of flits to be transmitted and the packet nature. Based on the packets nature, message priorities may be different. The header fields are presented in Table 1.

The highest two bits indicate the packet nature. The "00" code is associated to the signalling packet (Signalling[00]) such as urgent messages, short packets, interrupt and control signals that requires low transport latency and represents the highest priority packets.

Table 1: Header Fields

		Asked port address (3 bits)				
Packet nature	Flits number	North	East	Local	West	South
2 bits	3 bits	001	010	100	011	000

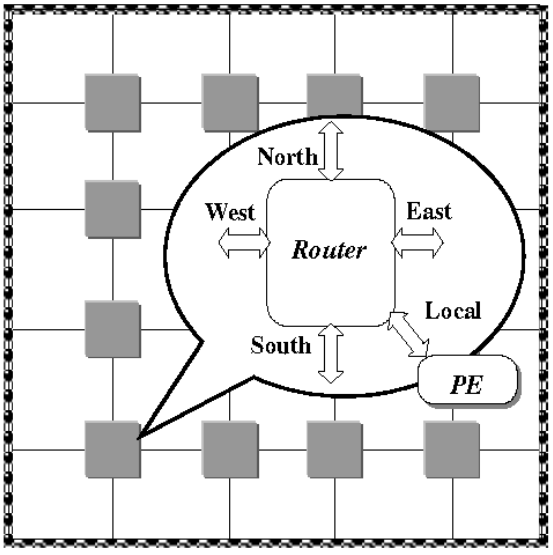


Fig. 4: NoC 2D Mesh Architecture

The "01" code is associated to the real-time application packets (Real-Time[01]). The "10" code is associated to the RD/RW packets (Read/Write[10]) such as short memory and register access. The "11" code is associated to the block transfer packets (Block-Transfer[11]) such as long messages and blocks of data that represents the lowest priority packets. We have adopted a source-specified routing and the lowest three header bits indicate the output port where the packet is to be transmitted. We assigned to every port an address as presented in Table 1. The "Adr_Ports" module generator contains the addresses of the different ports that it sends to the modules of comparison. It is activated as soon as a demand appears on one of the 5 ports. Each packet contains a list of output ports addresses that are extracted in each router and stored in a FIFO buffer.

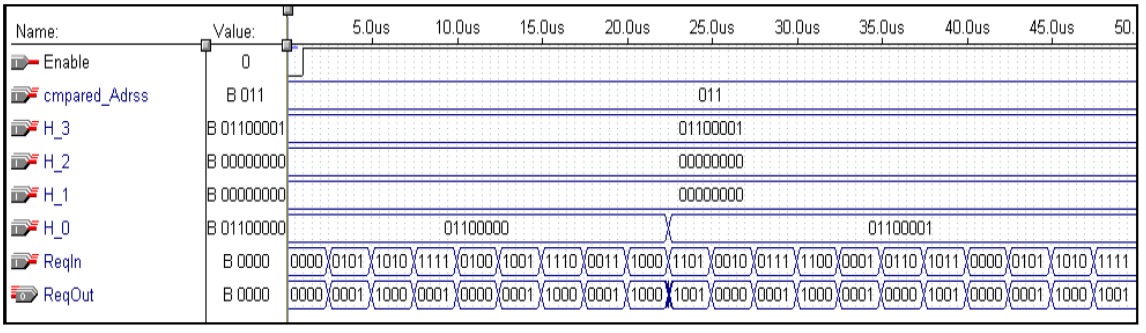


Fig. 5: Example of Simulation of the Comparator Module

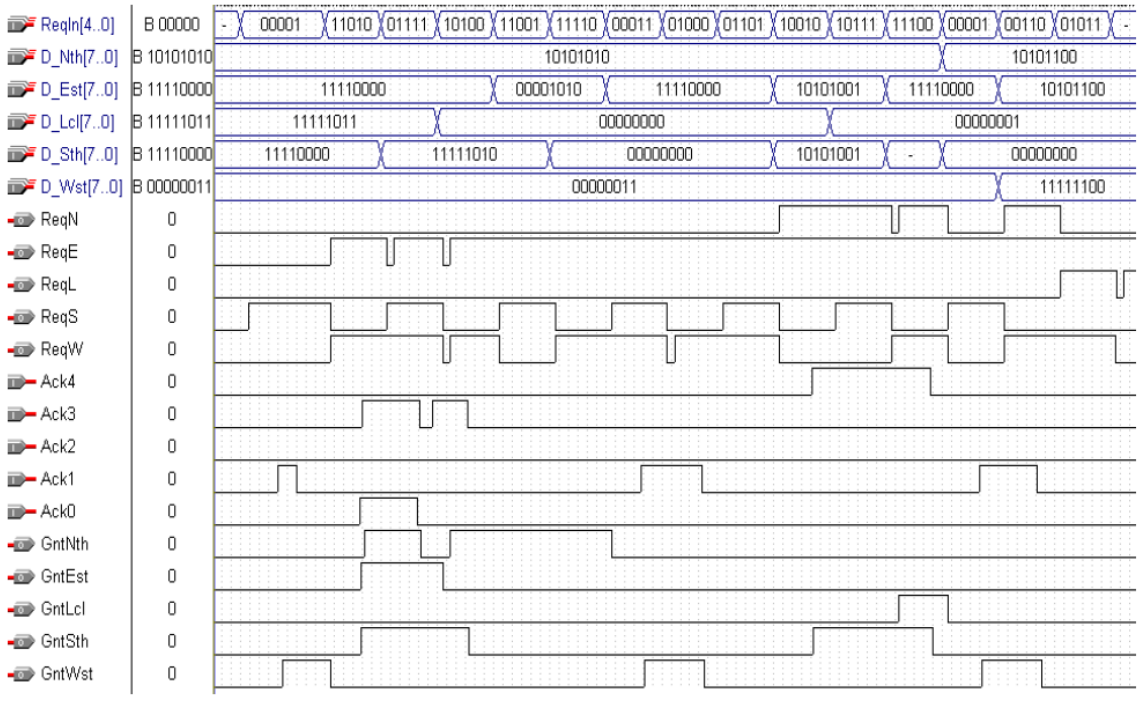


Fig. 6: Example of Simulation Results of the Mapped Architecture

The other header bits (Header(5 downto 3)) indicate the packet flits number. Thus, the packet where Header(7 downto 3) has the lowest value corresponds to the highest priority packet. This means that if a packet transfers large amounts of data in a single transfer over an output port, then that packet is assigned a lower priority, to prevent other packet from being denied access to the output port for a significant period of time while the data is being transferred.

Router Arbiter: Each NoC router integrates an arbiter to resolve access conflict to the output ports. Such arbitration can be performed by using either a tree arbiter, consisting of standard two-way arbiters, or a Mutual Exclusion (ME). Both architectures incur similar area but the ME seems to be slightly faster than the tree arbiter [26]. Since the ME does not always preserve the original order of the incoming requests it can grant the access to a requester that has a lowest priority that can performs preemptive routing according to packet priority.

By using the proposed dynamic scheme we have designed an asynchronous arbiter that allows the resolution of access conflict problems of each output port starting from the priority information of each incoming packet. The router architecture is constituted by five round-robin arbitration modules and five comparators interconnected by some C-elements and OR gates (Fig. 15(a)). Each comparator is affected to an output port. Each arbiter serves the 4 demands that are addressed to every port. The requests effectively allocated by the different comparators are treated by the corresponding referees.

The addresses of the different ports are sent to the comparators the moment the signal of activation passes to logic 1, as soon as a demand is received on the ReqIn bus of the requests port. Then, even though there is not an active demand anymore, the addresses remain to the rank of the outputs of the generating block of addresses.

Comparator Modules: The entries of the comparison module are the address to which is going to be compared, the addresses asked by the different ports, the enable signal (En), that permits to deactivate this module so much as a port had the access. Each comparator can be solicited only by the ports different from its corresponding port. For example, the North comparator could be solicited by demands that can come only from the ports East, Local, South and West. Every port sends a byte that contains the parameters of its target port, the number of flits to transmit and the order of priority of the packet.

The comparator module is disabled, as soon as a demand has been granted by the arbitration module, keeping the time to treat it. The data sent on the target port will not be interrupted.

The simulation of the comparison module gave the following results (Fig. 5): The H_n port represent the asked addresses and the bits of priority regrouped with the number of flits to exchange. As outputs, ReqOut, is a vector which bits are activated for the claimants, according to an order already established, when they ask for the port compared by this module and of course having the highest priority that first takes account of the priority of the packet defined its most elevated type (Signalling [00] (Highest), Real-Time [01], Read/Write [10], Block-Transfer [11](Lowest)). The number of flits is taken then in consideration in case of equality of priority between two claimants which one sends the less flits and it is kept.

The comparator module is functional as soon as the Enable signal is active. Before, the outputs are deactivated. The compared address is (011) that correspond to the South port. The different headers are examined then to see which the claimant ports are and which ones of them are asking for the South port. In Fig. 5 that corresponds to the simulation of the South comparator, for the period following the one where there is not any claimant, we observe two demands coming from the ports East and West (0101). If we see the ports for that they ask, we notice that the West port asks for the South port and the East port asks for the West port. Only the output that corresponds to the West port is activated.

EXPERIMENTAL RESULTS

Figure 6 presents a simulation result of the different modules grouped together to form the complete router. It is then immediate to see, according to the demands, the behaviour of the router. For example, the demands being to "01111" means that all ports are claimants except the North port. If we examine the asked ports, we notice that the input ports asks for the West and the Local ports. The West port asks for the South port. The last port asks for the East port. We notice also that ReqE, ReqS and ReqW are activated, since they are asked for. The demands pass then by the arbiters to adjust possible simultaneous demands (case of the demands Local and West toward the South port), it is sufficient to wait for an acknowledge to allow the access to every port.

The proposed asynchronous router is compared with a synchronous router of the same functionality. These routers were synthesized using Synopsys Design Compiler using a 0.35µm standard cell library. For the asynchronous router we have started from data flow and structural instantiated VHDL descriptions extracted from manual designed STG. For the synchronous router we have used VHDL/RTL instantiated descriptions. Since wormhole routing was used, some packets will be blocked for a period time in the network. The

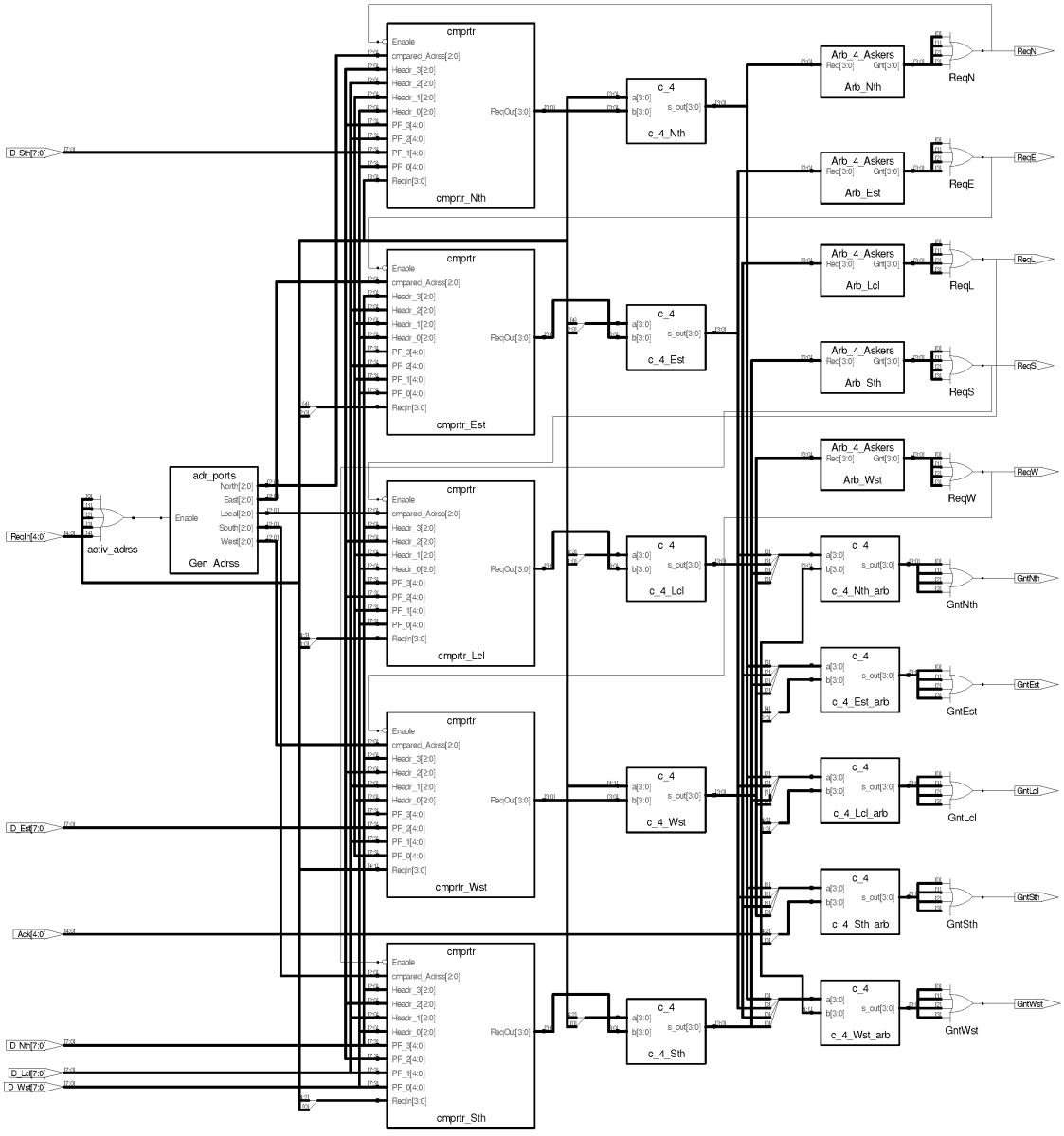


Fig. 7: Architectural View of the Proposed Router

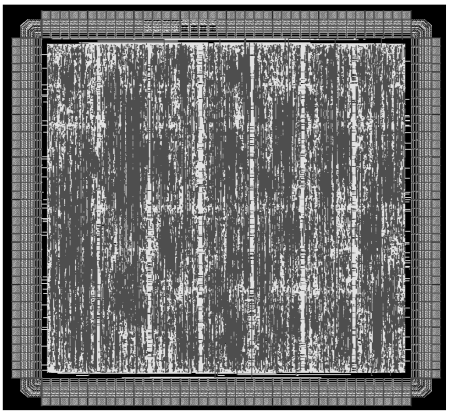


Fig. 8: Layout of the Final Asynchronous Router

synchronous router operates with 66 Mflits/s as data rate (~270 Mhz). After being integrated into a (5x5) 2D mesh NoC, the final devices occupy a smaller silicon area than the asynchronous one (4200 mm by 4200 mm). The same NoC based on asynchronous router occupy a 4881 mm by 4881 mm silicon area and the asynchronous router operates with 80 Mflits/s as a data rate. The synchronous router requires less area than the asynchronous router. The asynchronous router enables a data rate that is notably greater than the synchronous one. Figure 7 presents the architecture view of the asynchronous router controller. The input/output FIFO buffers and the header addresses FIFO buffer that form the data path are not shown. Figure 8 presents the layout result of the final asynchronous router.

CONCLUSIONS

In this study, we have presented an asynchronous router to be integrated into a 2D mesh NoC. The output ports arbitrate conflicting requests based into a dynamic arbiter according to packet priority. This scheme allows a preemption of lower priority transports.

We have used STG and an asynchronous library for the specification step. The synthesized router is SI and it is based on the generalized C-element gates and the 4-phase "valid"-"invalid" (Req/Ack) signals in order to avoid the inputs race problems. The routers have been designed by using a CMOS 0.35µm technology. The asynchronous router operates with 80 Mbytes/s as a data rate. A (5x5) 2D mesh NoC based on the asynchronous router occupies a 4881 mm by 4881 mm silicon area. Our design show that the asynchronous router enables a notable grater data rate than their synchronous counterparts (66 Mflits/s) but only a small difference less than the synchronous NoC in silicon area (4200 mm by 4200 mm). This is evident since QNoC for GALS SoC naturally benefit from asynchronous interconnects.

Future study will analyse the use of different links, such as serial versus parallel links. Packet and flit formats should also be questioned. The area and delay minimization based on a multiple service level router will be elaborated in order to reduce the interconnection problems. The entire NoC model should be analyzed when used for specific applications and different usage models and communication requirement mixes. The power consumption will also be studied in order to allow an efficient portability.

REFERENCES

1 Dally, W.J. and B. Towles, 2001. Route Packets, Not Wires: On-Chip Interconnection Networks. Proc. DAC.

2. Sutherland, I., 1989. Micropipelines. Comm. of ACM, Vol: 6.
3. Sonics, Incorporated, http://www.sonicsinc.com.
4. Peterson, W., 1999. Design Philosophy of the Wishbone SoC Architecture. In http://www.silicore.net/wishbone.htm.
5. Flynn, D., 1997. Amba: Enabling reusable on-chip design. Intl. J. IEEE Micro, pp: 20-27.
6. IBMCoreConnect Information, 2000. In http://www.chips.ibm.com/products/powerpc/cores.
7. Liang, J., S. Swaminathan and R. Tessier, 2000. A SOC: A Scalable, Single-Chip Communications Architecture. In the IEEE Intl. Conf. Parallel Architectures and Compilation Techniques, pp: 524-529.
8. Ho, H. and T.M. Pinkston, 2003. A Methodology for Designing Efficient On-Chip Interconnects on Well-Behaved Communication Patterns. In The 9th Intl. Symposium on High-Performance Computer Architecture (HPCA'03), pp: 377.
9. Kumar, S., A. Jantsch, J. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja and A. Hemani, 2002. A network on Chip Architecture and Design Methodology. In Proc. IEEE Computer Society Annual Symposium on VLSI, pp: 105-112.
10. Hu, J. and R. Marculescu, 2003. Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures. In Proc. Design, Automation and Test in Europe Conference.
11. Ye, T.T., L. Benini and G.D. Micheli, 2003. Packetized On-Chip Interconnect Communication Analysis for MPSoC. In Proc. Design Automation and Test in Europe, pp: 344-349.
12. Dally, W.J. and B. Towles, 2001. Route Packets, Not Wires: On-Chip Interconnection Networks. In Proc. the 38th Design Automation Conference.
13. Peh, L. and W.J. Dally, 2001. A Delay Model and Speculative Architecture for Pipelined Routers. 7th Intl. Symp. High-Performance Computer Architecture (HPCA).
14. Mullins, R., A. West and S. Moore, 2004. Low-Latency Virtual Channel Routers for On-Chip Network. Proc. 31st Intl. Symp. Computer Architecture.
15. Wang, H., L.S. Peh and S. Malik, 2003. Power Driven Design of Router Microarchitectures in On-Chip Networks. Proc. MICRO-36.
16. Rijpkema, E., K. Goossens *et al.*, 2003. Trade-offs in the Design of a Router with both Guaranteed and Best-effort Services for Networks on Chip. IEE Proc.-Comp. Digit. Tech., 150: 294-302.
17. Banerjee, N., P. Vellanki and K.S. Chatha, 2004. A Power and Performance Model for Network-on-Chip Architectures. Proc. DATE.
18. Vellanki, P., N. Banerjee and K.S. Chatha, 2004. Quality-of-Service and Error Control Techniques for Network-on-Chip Architectures. Proc. GLSVLSI'04, Boston, USA, pp: 45-50.
19. Bainbridge, J. and S. Furber, 2002. Chain: a Delay-Insensitive Chip Area Interconnect. IEEE Micro, 22: 16-23.
20. Bainbridge, W.J., L.A. Plana and S.B. Furber, 2004. The Design and Test of a Smartcard Chip Using a CHAIN Self-timed Network-on-Chip. Proc. DATE.
21. Bolotin, E., I. Cidon, R. Ginosar and A. Kolodny, 2004. Cost considerations in Network on Chip. Special issue on Networks on Chip. Integration-The VLSI J.
22. Dobkin, R., R. Ginosar and C.P. Sotiriou, 2004. Data Synchronization Issues in GALS SoCs. Proc. ASYNC.
23. Kessels, J., A. Peeters, P. Wielage and S.-J. Kim, 2002. Clock Synchronization through Handshake Signalling. Proc. ASYNC, pp: 59-68.
24. Villiger, T., H. Kaeslin, F.K. Gürkaynak, S. Oetiker and W. Fichtner, 2003. Self-Timed Ring for Globally-Asynchronous Locally-Synchronous Systems. Proc. ASYNC, pp: 141-150.
25. Muttersbach, J., T. Villiger and W. Fichtner, 2000. Practical Design of Globally-Asynchronous Locally-Synchronous Systems. Proc. ASYNC, pp: 52-61.
26. Moore, S., G. Taylor, R. Mullins and P. Robinson, 2002. Point to Point GALS Interconnect. Proc. ASYNC.
27 Moore, S.W., G.S. Taylor, P.A. Cunningham, R.D. Mullins and P. Robinson, 2000. Self-Calibrating Clocks for Globally Asynchronous Locally Synchronous Systems. Proc. Intl. Conf. Computer Design (ICCD).
28. Rigaud, J.B., 2002. Spécification de Bibliothèques pour la Synthèse de Circuits Asynchrones. Phd Thesis, National Institut of Polytechnique of Grenoble, France.