A Virtual Channel Router for On-chip Networks

Nikolay Kavaldjiev, Gerard J. M. Smit, Pierre G. Jansen

Department of EEMCS, University of Twente, the Netherlands
{nikolay, smit, jansen}@cs.utwente.nl

ABSTRACT

This paper proposes an architecture of a virtual channel router for an on-chip network[1]. The router has simple dynamic arbitration which is deterministic and fair. We show that the size of the proposed router is reduced by 49% and the speed increases 1.4 times compared to a conventional virtual channel router.

I. INTRODUCTION

It is expected that interconnection technology will become a limiting factor in future system-on-chip (SoC) designs [1]. A possible approach for coping with this problem is to use an on-chip interconnection network instead of ad-hoc global wiring [2]. Such a network provides an on-chip communication infrastructure for interconnecting the system components.

Several solutions for on-chip networks have been proposed [2][3][4][5]. While all of them are based on simple routers interconnected through network channels, (usually in a mesh topology), they differ in the techniques used for the router implementations.

In this paper we advocate a packet switching network with virtual channel flow control, an approach proposed in [2]. We believe it provides the performance, flexibility, area and energy efficiency needed in a dynamic SoC. We propose a router architecture that simplifies the dynamic arbitration compared with a conventional virtual channel router and makes it deterministic and fair.

Paper organization: Section II introduces the SoC where our network on-chip is used. Section III presents a traditional virtual channel router architecture. In Section IV we propose the new router architecture. Section V presents implementation results.

II. BACKGROUND

To put the proposed router architecture into perspective and to simplify its explanation, we first present the SoC where it will be used.

We target our network at a heterogeneous system-on chip, a platform for future wireless multimedia devices, depicted in *Fig. 1*. The system integrates computational units of different levels of granularity and configurability (e.g. GPP, DSP, FPGA, ASIC). They are interconnected through a packet-switched on-chip network that should provide a well-structured, flexible, and efficient communication infrastructure.

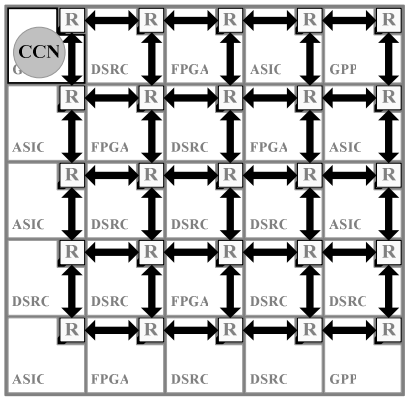

Fig. 1. A tiled heterogeneous reconfigurable SoC. GPP – General-Purpose Processor; DSP – Digital Signal Processor; ASIC – Application-Specific Integrated Circuit; FPGA – Field-Programmable Gate Array; DSRC – Domain-Specific Reconfigurable (e.g. [12])

We assume that the system is organized as a centralized system: one node (a GGP tile), called *Central Coordination Node* (CCN), performs system coordination functions. The main task of the CCN is to manage the system's resources. It performs mapping of the newly arrived tasks on suitable computation units and inter-task communications to network channels. It also tries to satisfy Quality of Service (QoS) requirements, to optimize the resources usage and to minimizing the energy consumption. The CCN does not perform scheduling of tasks and communications, but only mapping and allocation.

The centralised mapping of communications on network channels requires the use of *source routing*, where the route a packet takes in the network is predetermined and completely described in the packet's header.

For describing the network traffic in the system, we adopt the notation used in [3]. According to the type of services required, the following types of traffic can be distinguished in the network:

[1] This work is partially supported by the Dutch Organization for Scientific Research NWO and IST-FP6

- *GT (guaranteed throughput)* – this is the part of the traffic for which the network has to give real-time guarantees (i.e. guaranteed bandwidth, bounded latency).

- *BE (best effort)* – this is the part of the traffic for which the network guarantees only fairness but does not give any bandwidth and timing guarantees.

Furthermore we assume that the traffic in a multimedia terminal is often stream based. This means that for a long period of time subsequent data items of a stream follow the same route.

III. WORMHOLE ROUTING WITH VIRTUAL-CHANNEL FLOW CONTROL

Wormhole routing with virtual channel flow control is a well-known technique from the domain of multiprocessor networks [6]. It allows minimization of the size of the router's buffers - a significant source of area and energy overhead [3][7], while providing flexibility and good channel utilization.

A general structure of a wormhole router with virtual channel flow control is depicted in *Fig. 2*. In this example the router has 5 input/output ports: 4 for connection with the neighbour routers and one for connection with the local node. At each input port the virtual channels (VCs), 4 in this case, are demultiplexed and buffered in FIFOs. Status information is kept for each of them. After the FIFOs they are multiplexed again on a single channel which goes to a crossbar. The operation of the router is controlled by an arbitration unit (AU). It determines on a cycle-by-cycle basis, which virtual channels may advance.

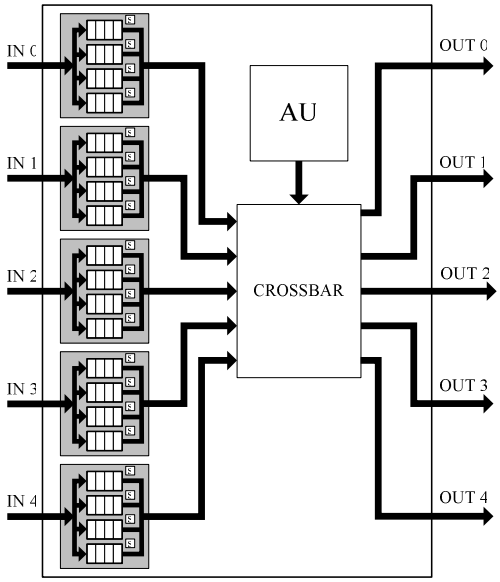

Fig. 2. General structure of a virtual channel router with 5 ports and 4 virtual channels.

During the operation of the router a VC can be in one of the following states: *idle* – the VC is not used at the moment; *busy* – there is a packet using the VC; *empty* – busy VC with empty FIFO; *ready* – busy VC with nonempty FIFO.

After initialization all VCs are in the *idle* state. When a new packet arrives on a certain VC, the state of this VC is changed to *busy*. In order to start forwarding of the newly arrived packet the router needs the following information:

output port (p) – number of the output port the packet has to go

output VC (v) – number of the VC of the *output port* the packet has to be sent on.

In the sequel we refer to these values as *p* and *v* respectively.

Since we use source routing (see Section II), the values for *p* and *v* are obtained from the packet's header. Each packet consists of a multi-flit header followed by data flits and terminated by a tail flit. For each router the packet passes there is one corresponding header flit. The header flit contains three fields: a value for *v*, a value for *p* and *id* (the function of the last one will be explained later). Every router examines the first header flit, extracts the values for *p*, *v* and *id* from it, stores them as state information for the corresponding input VC and then removes the flit from the packet. The values for *p*, *v* and *id* are the address information a tile in the system needs in order to send a message on the network. They are generated by the CCN during the task mapping stage and provided to the source tile at the tile configuration stage.

After the router has obtained values for *p* and *v* it can start forwarding the packet. The packet's VC starts competing for a crossbar connection to the output port *p*. The competition is governed by the AU. Each cycle the AU decides which of the *ready* input VCs may advance.

The main task of the arbitration unit is to solve the following possible conflicts between VCs:

- *conflicts at the inputs* – at each cycle only one VC can advance from an input port
- *conflicts at the outputs* – at each cycle an output port can accept data from one input VC only.

The arbitration has to be fair and give equal chances to all competing VCs. It also has to aim at maximal utilization of the output channels to obtain a high network throughput. These last two requirements are contradicting as for many traffic patterns maximal throughput can only be achieved if fairness is sacrificed. Moreover, to allow handling of the GT traffic the arbitration has to be deterministic.

In our first implementation of the virtual channel router we used the general structure presented in *Fig. 2* and a SLIP arbiter [10]. The SLIP arbiter is fair, has good performance, but is nondeterministic and so cannot handle the GT traffic.

In the next section we propose an architecture which overcomes this problem. It is smaller in size, has fair deterministic arbitration and high throughput.

IV. PROPOSED ROUTER ARCHITECTURE

The proposed architecture is shown in *Fig. 4*. It differs from the traditional one in that the VCs are not multiplexed after the FIFOs in each input block, but connected directly to the crossbar. The multiplexers for the request and acknowledge signals are also integrated in the crossbar.

Since there are no *conflicts at the inputs* anymore, the AU can be reduced to a small round robin arbiter (RRA) [11] for each output port. The arbitration is deterministic and fair. We can give an upper bound for the latency a packet experiences when passing through the router. Since the arbitration is based on round robin arbiters without any dependencies between them, each VC, if *ready*, is served every 4-th cycle and thus receives at least ¼ of the channel bandwidth (this in a case of 4 VCs per port). Because there are conflicts only at the out ports, this router can achieve a throughput of an output queued switch (100%) [13].

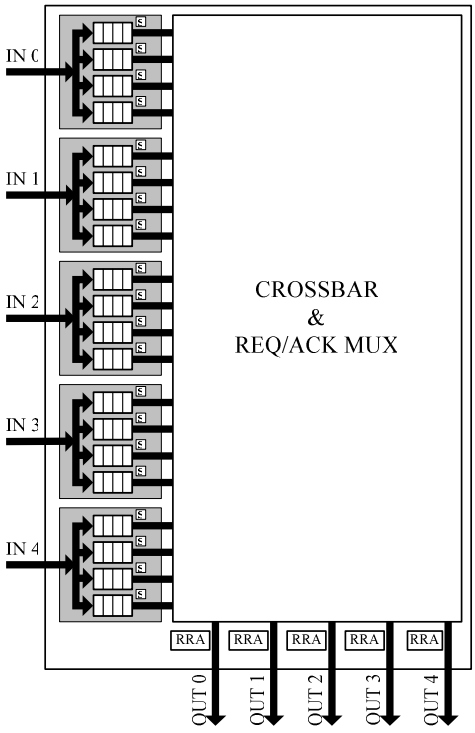

Fig. 4. A virtual channel router with simplified arbitration.

The crossbar is asymmetric and larger than before. If *N* is the number of router's ports and *V* is the number of VCs per port, then we need a crossbar of size *(V×N)×N*. We see that this solution is not efficient for routers with large number of ports (or VCs), but since on-chip networks will most probably use two-dimensional topologies we expect the routers to have N=5 ports.

The request/acknowledge multiplexers are uniformly distributed in the crossbar. Each cross point of the crossbar, together with the 3-state drivers, contains a simple decoding/multiplexing circuit.

A simplified cross point is shown in *Fig. 5*. The function of the circuits presented there is to multiplex the request/acknowledge signals between the input VC and the desired output VC. One comparator recognises if this is the cross point to the destination output port (*p=P*). If so, the arbitration request, *REQ*, is demultiplexed to the destination output request line. All the requests to an output VC are collected by an OR chain spanning along the crossbar column. Finally, the requests to all VCs (*Req_0, Req_1, Req_2, Req_3*) are sent to the arbiter (RRA) of that port.

The RRA arbitrates only between those output VCs for which a request is sent and for which there is free buffer space in the next router. Whether there is free buffer space for each VC is indicated by the signals *Rdy_0*, *Rdy_1*, *Rdy_2*, and *Rdy_3* coming from the next router. Every cycle the RRA issues the number of the granted output VC (*Vc_ack*) which spans back along the crossbar column. In the cross point a comparator recognizes whether the requested output VC is granted and if so, enables the 3-state driver and sends an acknowledge signal, *ACK*, to the input VC (again through an OR chain).

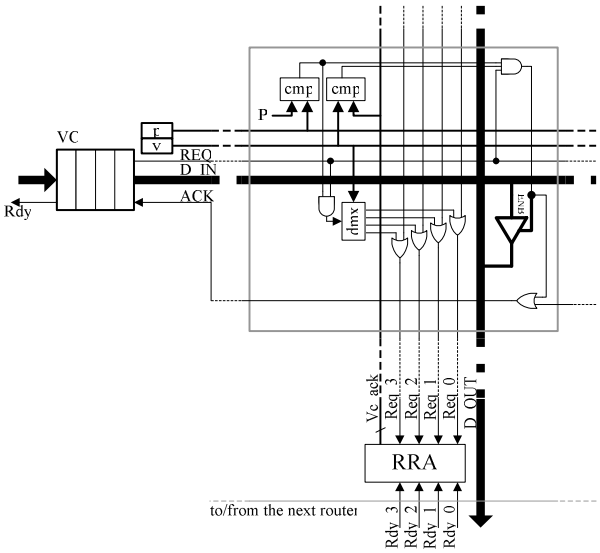

Fig. 5. Request/acknowledge multiplexing circuits in a crossbar's cross point.

The mechanisms described above works properly if only one input VC uses a certain output VC. The CCN can provide that condition. If the CCN guarantees for a network connection that it is the only user of the VCs it traverses, then the packets of this connection will always find the VCs free. Hence they can immediately attain them without experiencing any latency. Since the router arbitration is deterministic, after the packet has attained a VC we can give an upper bound for the latency. If *L* is the length of the packet in flits and *V* is the number of VCs per port, then the number of cycles, *T*, it takes for the packet to traverse the

router is: $L \le T \le V \times L$. Thus, we can handle the GT traffic

When the CCN allows a VC to be shared between several connections, a packet of such a connection may find the VC busy and experience nondeterministic latency before attain it (this latency can be estimated only statistically). Thus no guarantees can be given for such connection and they can only be used for BE traffic.

In a case of a shared VC we have to arbitrate between all the input VCs willing to use the same output VC at a time and to allow only one of them to attain it. The arbitration takes place when an input VC goes from *Idle* to *Busy* state, or in other words when a new packet arrives. From that moment the VC starts competing for the desired output VC until attains it.

For implementing this arbitration in each cross point we add the circuits shown in *Fig. 6*. For each input VC there are two new signals: *hold* and *vcb* (VC is busy). The *vcb* signal shows whether the desired output VC is busy or not. The input VC activates its *hold* signal when attains the desired output VC. In the selected cross point (*p=P*) the *hold* is demultiplexed to the desired VC and collected by an OR-chain. The OR-chains produce signals (*Busy_0, Busy_1, Busy_2, and Busy_3*) showing whether the corresponding output VC is busy or not. In the cross point the respective *Busy* signal is selected by a multiplexer and sent through an OR chain to the input VC as the *vcb* signal.

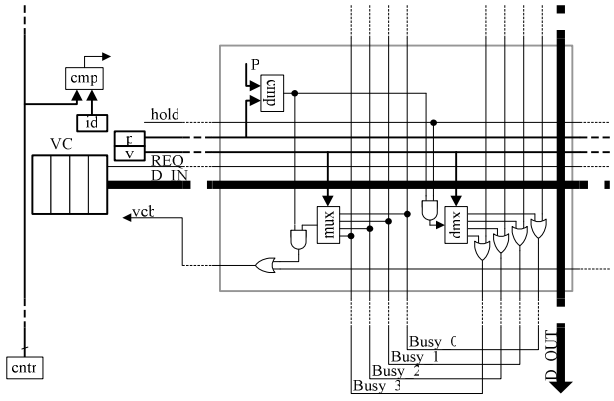

Fig. 6. Additional multiplexing circuits in a crossbar's cross point

In order to simplify the arbitration circuits we implemented the following solution. Each of the VCs competing for a certain output VC has a unique identifier, *id*, which is provided on a higher system level by the CCN. In the router there is a counter, *cntr*, that counts each clock cycle and which value is distributed to all input VCs. A competing VC attains the desired output VC only if the output VC is not busy (the signal *vcb* is not active) and the current counter value equals the VC's *id*. When attaining an output VC the input VC activates its *hold* signal and thus makes the output VC busy. When the packet finishes, the input VC goes in i*dle* state and deactivates its *hold* signal, which releases the output VC.

The uniqueness of the *id* guaranties that the arbitration is conflict free. Because the counter counts permanently and independently of the state of the VCs, at the moment of arbitration we find it generally in a random state. This randomness provides fairness of the arbitration.

The stream communications can be handled by representing the stream as a packet of infinite length. This network allows that as it does not put any restrictions on the packet length. To open a stream the source just sends a packet header. To close the stream a tail flit has to be sent. Since VCs used by the stream cannot be shared, the streams always belong to the GT part of the traffic.

V. RESULTS

Below we present the implementation results for two virtual channel routers – the first one with a traditional architecture (*Fig. 2*) and a SLIP arbiter, the second one with the architecture proposed in Section IV (we refer to these two architectures as "Reference architecture" and "Proposed architecture"). Both routers have the same parameters: 5 input ports, 16 bit channels, 4 virtual channels per port, and 4 word buffers depth.

To get an indication about the size and speed difference the routers were synthesized for Xilinx Vitrex-II FPGA and standard 0.5 μm ASIC technologies. For size indication we took the number of utilised CLB slices in the FPGA and the number of utilised gates in the ASIC.

We experimented with two implementations for the FPGA. In the first one OR-chains were used, as it was described in Section IV. In the second one we replaced the OR-chains with 3-state driven lines taking advantage of the fact that the 3-state buffers in Virtex technology are weakly pulled up. Thus we emulated a wired-OR line.

TABLE I. IMPLEMENTATION RESULTS

		Units	Reference architecture	Proposed architecture /OR chains/	Proposed architecture /3-states/
Virtex-II	Buffers	CLB slices	764	600	600
	Arbiter		733	53	53
	Crossbar		335	672	278
	Total		1832	1325	931
	F_{max}	MHz	66	86	97
ASIC 0.5um	Buffers	Gates	25467	24718	-
	Arbiter		21719	1043	-
	Crossbar		5782	15198	-
	Total		52968	40959	-
	F_{max}	MHz	68	96	-

The results are presented in Table I. It gives the total amount of CLB slices and gates used and also their distribution between the buffers, crossbar and arbitration. It also gives the maximal clock frequency reported by the synthesis tool (Leonardo Spectrum).

The area distribution for the FPGA and ASIC implementations is presented graphically in *Fig. 7* and *Fig. 8* respectively. There we see that in the reference architecture the arbitration unit area is much larger than the crossbar area, while the proposed architecture changes this proportion and also reduces the total router area. The area reduction is 23% for the ASIC, 26% for the FPGA using OR-chains, and 49% for the FPGA using wired-OR.

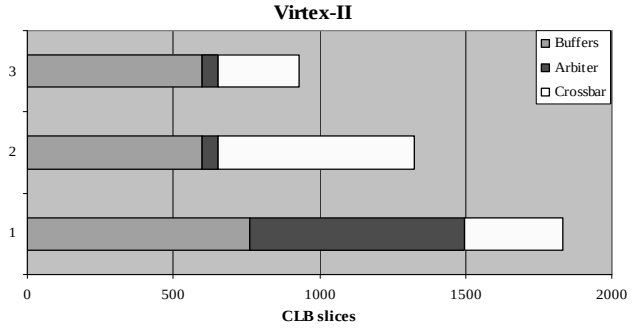

Fig. 7. Router area distribution for FPGA implementation. Bar 1 – Reference architecture. Bar 2 – Proposed architecture /OR-chains/. Bar 3 – Proposed architecture /3-states/

For the FPGA implementation we see that the buffer area is also reduced. This is because the reported area is in fact for the whole input block including the multiplexers and demultiplexers, routing logic and the VC state logic. In the proposed architecture we have removed the multiplexers after the FIFOs.

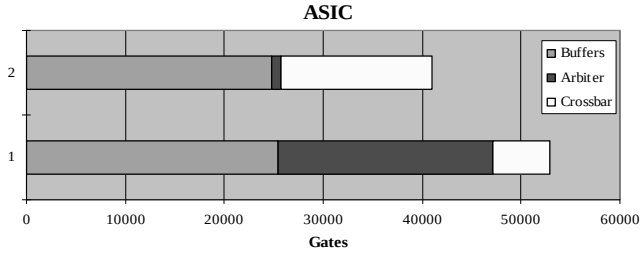

Fig. 8. Router area distribution for ASIC implementation. Bar 1 - Reference architecture. Bar 2 - Proposed architecture.

For the Virtex, when using 3-state drivers (wired-OR) instead of OR-chains we allow the synthesizer to utilize the unused 3-state drivers in already used CLB slices instead of allocating new CLB slices. That is the reason for the area reduction gained when using wired-OR.

VI. CONCLUSION

In this paper we presented an architecture for a 5-port virtual channel router with simplified dynamic arbitration. The simplification allows fair and deterministic arbitration, capable of handling guaranteed throughput traffic as well as best effort traffic. The architecture shows to be efficient in area and suitable for on-chip networks with two-dimensional topologies. We showed that the size of the proposed router is reduced by 49% for the FPGA implementation and 23% for the ASIC implementation and that the speed is improved 1.4 times compared to a conventional router. The router can achieve the maximal throughput of an output queued switch, 100%.

REFERENCES

[1] Luca Benini, Giovanni De Micheli, “Networks on Chips: A New SoC Paradigm.”, *IEEE Computer*, January 2002 (Vol. 35, No. 1), pp. 70-78.

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

[3] E. Rijpkema, K. G. W. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander, "Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip", *Proceedings of Design Automation and Test Conference in Europe*, March 2003.

[4] Michael Bedford Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffman, Jae-Wook Lee, Paul Johnson, Walter Lee, Albert Ma, Arvind Saraf, Mark Seneski, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe and Anant Agarwal, ‘The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs”, *IEEE Micro*, March/April 2002.

[5] J. Liang, S. Swaminathan, and R. Tessier, “aSOC: A Scalable, Single-Chip Communications Architecture.”, *In the Proceedings of the IEEE International Conference on Parallel Architectures and Compilation Techniques*, Philadelphia, PA. October 2000.

[6] W. J. Dally, “Virtual-channel flow control”, *IEEE Transactions on Parallel and Distributed systems*, vol. 3, no. 2, pp. 194-205, March, 1992.

[7] Hang-Sheng Wang, Li-Shiuan Peh and Sharad Malik, "A Power Model for Routers: Modeling Alpha 21364 and InfiniBand Routers.", *In IEEE Micro*, Vol. 23, No. 1, January/February 2003 (Selected articles from Hot Interconnects 10).

[8] McKeown N., Anderson T.E., "A quantitative comparison of scheduling algorithms for input-queued switches", *Computer Networks & ISDN Systems*, vol. 30, n. 24, Dec. 1998, pp. 2309-26.

[9] Li-Shiuan Peh, "Flow Control and Micro-Architectural Mechanisms for Extending the Performance of Interconnection Networks.", Ph.D. Thesis, Stanford University, August 2001

[10] Nick McKeown, "iSLIP: A Scheduling Algorithm for Input-Queued Switches" *IEEE Transactions on Networking,* Vol 7, No.2, April 1999.

[11] Pankaj Gupta and Nick McKeown, "Designing and Implementing of a Fast Crossbar Scheduler", *IEEE Micro Magazine, Jan-Feb 1999.*

[12] Heysters P.M., Smit G.J.M. & Molenkamp E.: “A Flexible and Energy-Efficient Coarse-Grained Reconfigurable Architecture for Mobile Systems”, The Journal of Supercomputing, volume 26, number 3, Kluwer Academic Publishers, Boston, U.S.A., November 2003, ISSN 0920-8542.

[13] M. Karol, M. Hluchyj, and S. Morgan, "Input versus output queueing on a space-division packet switch," *IEEE Transactions on Communications*, vol. 35, no. 12, pp. 1347--1356, December 1987.