A Distributed Crossbar Switch Scheduler for On-Chip Networks

Kangmin Lee, Se-Joong Lee, and Hoi-Jun Yoo
Semiconductor System Laboratory, Dept. of Electrical Engineering, KAIST, Daejeon, Korea
kangmin@mail.kaist.ac.kr

Abstract

A scheduling algorithm is proposed for lightweight on-chip crossbar switch in on-chip networks. The proposed NA-MOO algorithm distributes the arbitration computing over all of the crossbar fabric nodes. Its implementation shows that it can reduce > 60% area and > 20% computation delay compared to conventional round robin based SLIP algorithm. Its feasibility is analyzed by using a SoC for HDTV as an example. The proposed techniques are area-efficient and show higher performance for the on-chip interconnection networks.

1. INTRODUCTION

System-on-chip (SoC) provides integrated solutions to challenging design problems in telecommunications, multimedia, and consumer electronics domains. One of the bottlenecks for achieving the operational goal of SoCs is the on-chip physical interconnection that will present a limiting factor for performance and, possibly, energy consumption [1]. Recently, a general-purpose on-chip interconnection network became of interest as a replacement for design-specific global on-chip wiring [1-2]. Using a network, which exploits the methods and tools used for general computer networks, is known to achieve efficient communication on SoCs [1]. An essential component of SoC networks and the computer networks is a switch fabric circuit, by which all network traffic is routed from ingress ports to egress ports. In particular, a nonblocking crossbar switch is widely used for a high-speed switching since it is free of *interconnect contention* and needs less buffer bandwidth. While most attention is focused on speed and capacity issues of switch fabrics in legacy computer networks, the silicon area of the switch fabric, including the crossbar core, input/output buffers and a crossbar scheduler, is becoming of a concern for SoC networks [2].

The success of on-chip network architecture depends on the ability to keep the overall area overhead to its minimum. The major components of a crossbar switch are in/out buffers, a crossbar core and an on-chip scheduler.
To reduce the area of a switch fabric core the link serialization is efficient [2]. This 4:1 serialization reduces the size of the switch core to 1/16, but the size of in/out buffers and a scheduler remain unchanged.

The area of the on-chip switch is heavily dominated by the space occupied by the on-chip buffers. This space limitation of the on-chip buffer comes in deep contrast with real data networks where there is ample room for very large buffers. Under these circumstances, the virtual output queueing (VOQ) scheme for alleviating head of line (HOL) blocking is difficult to be adopted since the VOQ needs non-shared independent input buffers of N^2. Moreover, VOQ is not so efficient because the traffic patterns of SoC networks are not as random as traffic patterns of data networks. To make matters worse, adopting VOQ increases the crossbar scheduler complexity twofold [4]. For SoC networks with a star topology, HOL blocking is very rare because a switch port is dedicated to a computing node. As a result, VOQ is inappropriate for on-chip networks so that we assume the input buffer as FIFO queueing not VOQ.

In this paper, we propose a method to reduce the area of OCN: a NA-MOO algorithm for the reduction of the area of an on-chip scheduler. By using this method we reduce the area of the scheduler by 60%. In addition, the speed of the scheduler is 20% faster than that of a conventional scheduler.

This paper is organized as follows: in section 2 the conventional crossbar's scheduler circuits are explained; in section 3 we propose a new crossbar scheduling algorithm which is compared with a conventional round robin algorithm from a viewpoint of an area, a computing speed and a switching performance especially for HDTV applications in section 4. Finally conclusion will be made in section 5.

2. BACKGROUND

A block diagram of a general crossbar scheduler with FIFO queueing is shown in figure 2.1. It consists of three blocks: input FIFO buffer, arbiters for each output port and a crossbar fabric core. Each input buffer generates a request to a destined output port's arbiter. Each arbiter selects a request among input ports to use the output link. The arbiter then generates proper control signals for the crossbar core to set up a path from the granted input port and gives a grant signal to the granted input port. The granted input queue transfers the packet through the crossbar core.

2.1. Crossbar Scheduler

An intelligent centralized scheduler is needed to guarantee the fairness among input ports and to use the switch fabric efficiently in the network switches and routers. Most crossbar

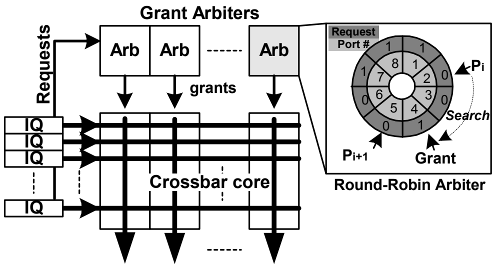


Figure 1. Crossbar switch with FIFO Queueing and round-robin arbiters [4]

schedulers are based on a round robin [4]. (See figure 1.) The round robin has a rotating priority denoted by *P*. The round robin arbiter guarantees that none of the input ports are starved, and that all are treated fairly. However, hardware for the round-robin arbiter is too complicated to integrate within a SoC [4].

State-of-the-art implementation of a round robin is shown in figure 2. A round-robin arbiter consists of a programmable priority encoder (PPE), registers to memorize the priority pointer, an incrementer and a binary encoder. The PPE consists of two simple priority encoders (smpl_PE), a thermal encoder and bitwise AND/OR gates. The fastest and smallest smpl_PE that use a multi-level look-ahead and folding scheme was proposed in [5]. The centralized priority pointer makes the round-robin scheduler complicated.

Therefore, we propose a new crossbar-scheduling algorithm called NA-MOO, which uses neither a centralized priority nor the area-consuming round-robin arbiter. Furthermore, since the NA-MOO scheduler can be implanted into a crossbar fabric core, the wiring area between the scheduler and the crossbar core can be removed. A description and a performance analysis are presented in detail in section 3.

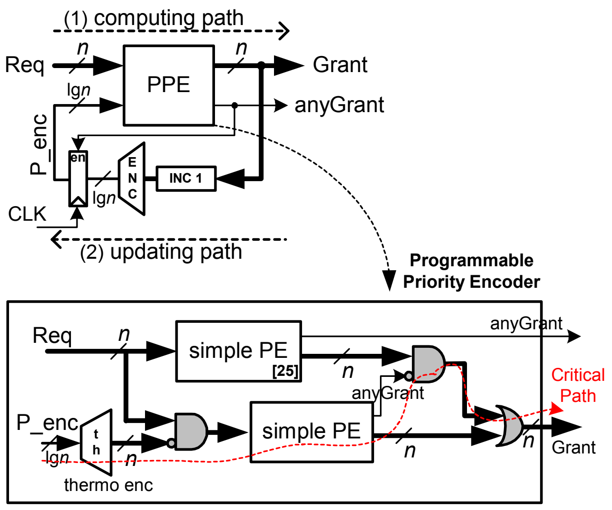


Figure 2. Implementation of a round-robin arbiter [4-5]

3. Proposed NA-MOO algorithm

3.1 Description of the NA-MOO algorithm

The proposed NA-MOO algorithm's scheduler embedded in crossbar core is shown in figure 3. This is mixture of the crossbar scheduler and Mux-Tree-based crossbar fabric core for an output port.

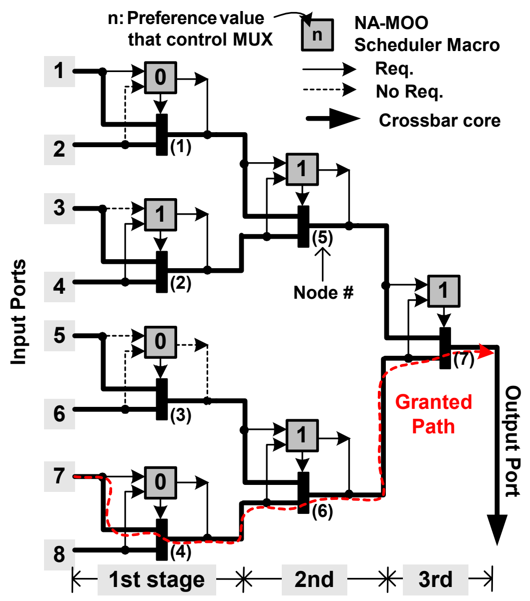


Figure 3. Proposed NA-MOO algorithm scheduler mixed with crossbar core

The arbitration computing is centralized and the priority rotates in a fixed round-robin fashion at round robin based SLIP algorithm [4]. On the contrary, at NA-MOO algorithm, the computing is distributed over crossbar core nodes and the priority is determined by each node's state. We call the state *preference value*. Due to the distributed computing rather than centralized computing, the NA-MOO scheduler complexity is reduced.

Each node from (1) to (7) in figure 3 has a 1-bit preference value in its NA-MOO scheduler macro by which each node's multiplexer selects an input to be transfer through the node. Actually the preference value is referred only when both inputs bring requests *i.e. output conflict*. (See figure 4)

By a propagation of (a request, packet data) through the multiplexer at each stage, only one packet data reaches the output port as shown as a dot line in figure 3. This is the so-called crossbar arbitration.

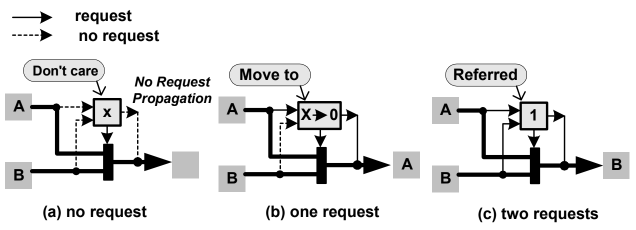


Figure 4. Operation at a node case-by-case

After a packet is transferred through a granted path from the granted input port to the output port, the preference values are toggled along the granted path to give the granted input port the lowest priority. This updating mechanism is similar to SLIP algorithm, which leads to a desynchronization of the output arbiters [4]. For example, in figure 3, if input #7 is granted; the preference values of (4), (6) and (7) nodes are toggled while other multiplexers don't change their preference values. Then, the path from the input #7 to the output is totally blocked because each node prefers the opposite path by altering preference values. Therefore, the input port #7 gets the lowest priority after granted.

The most distinctive feature NA-MOO scheduler is that it distributes the computing of the priority over all of the nodes. As a result, the computing time as well as the complexity of the scheduler is reduced. The complexity of NA-MOO is exactly O($\log_2$N), while that of the round robin is O(N). Moreover, since the NA-MOO scheduler can be implanted into the crossbar core, the overall area of the scheduler is further reduced. The wiring delay of the control signals, from scheduler to crossbar core, is also reduced. The NA-MOO scheduler's advantage of higher computing speed and smaller area will be discussed more in section 3.3.

3.2 The NA-MOO algorithm's unfairness and no-starvation

A disadvantage of the NA-MOO is that it cannot guarantee fairness as seen in figure 5. The NA-MOO algorithm gives more grants to the input ports which are located away from other active ports.

However, although unlikely in wide-area computer networks, fairness requirement is not so strict in on-chip networks. In actual on-chip communication, for instance, the vast and bustling traffic does not collide at the same time and same destination on purpose, otherwise the system lastly cease for a moment until the destined component serves the aggregated work-load with its limited service capacity. Therefore, output conflict on an on-chip switch does not often occur unlike a switch of wide-area computer network, which means unfairness does not results a big problem. The detail about the unfairness will be more discussed in section 5.

In NA-MOO algorithm, input ports never starve. Every input port can get a grant after at least *N* cell-time slots as long as there is a cell to be served on the input port, where *N* is the number of input ports. This starvation property is the same as in the round robin algorithm [4].

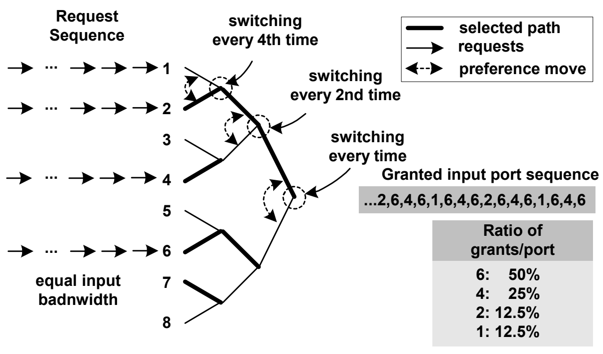


Figure 5. An example showing the unfairness of NA-MOO

3.3 Advantage of NA-MOO: reduction of area and computing delay

The hardware of the switch adopting NA-MOO algorithm's scheduler has high modularity, as shown in figure 3. The hardware unit, including the multiplexer and the NA-MOO scheduler macro, is repeated at every node. Figure 6 shows the hardware implementation of a node, and figure 7 shows the preference-updating (or toggling) hardware circuits. (See also Figure 2 for comparison with the round-robin scheduler)

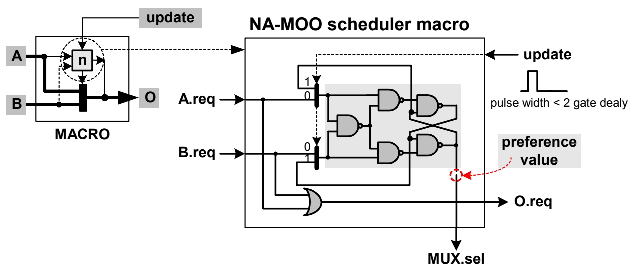


Figure 6. Implementation of a NA-MOO macro

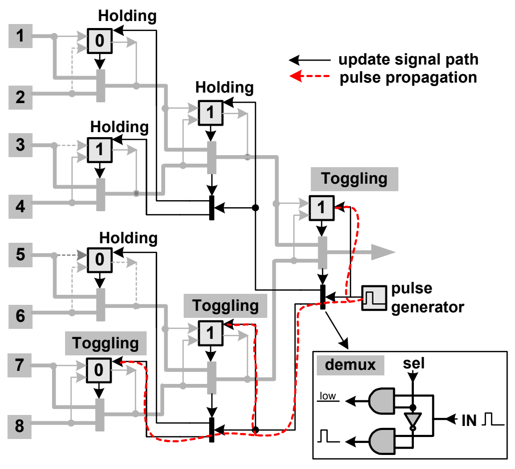


Figure 7. Preference updating circuits

Both the NAMOO arbiter and the round-robin arbiter have two paths: a granting path and a priority-updating path. Table 1 and 2 show the comparison of the gate counts and the critical path delay, respectively for an 8x8 switch. The NA-MOO arbiter consumes only 40% hardware resources than the round-robin arbiter in terms of the gate counts. Moreover, the circuits of simple priority encoder and the incrementer [5] in round-robin scheduler need complex wiring and larger gate width transistors.

The critical path delay of the NA-MOO arbiter is 80% of that of the round-robin arbiter. Actually, in the case of the round-robin arbiter, the grant signals should run across the whole crossbar core vertically and this long traveling signal increases the scheduler's latency (see figure 1). However, for the NA-MOO arbiter, the grant signal is generated at each macro to control the multiplexer (see figure 3). In this case there is no need for the grant signal to travel along the long wire.

Table 1. Required hardware resources

Hardware elements		NA-MOO scheduler	Round-Robin scheduler
NAND/NOR gate		48	44
INVERTER		3	12
2:1 MUX		14 (=2(N-1))	0
D-F/F		0	3 (=log$_2$N)
smpl PE		0	2
INCREMENTER		0	1
# of Tr.	NMOS	127	359
	PMOS	127	305
Comparison		~40%	100%

Table 2 Critical delay

Critical Delay	NA-MOO scheduler	Round-Robin scheduler [14][25]
Granting Path	(*tMUX*+2**tNAND*)*log*N*	5**tNAND* + 3**tGate of sPE*
Updating Path	log*N***tMUX*+2**tNAND*	6**tGate*+2**tNAND*+*tDFF*
Summation	14 (=9+5) * tINV_EQ	18 (=8+10) * tINV_EQ
Comparison	~ 80%	100%

4. EVALUATION IN HDTV SOC

In this section we apply the proposed NA-MOO algorithm to the SoC for High Definition Television (HDTV) based on the video processor with simultaneous decoding of two MPEG2 MP@HL streams and capable of 30frames/s reverse playback [3]. This SoC contains numerous components compatible with broadcasting receivers and home multimedia servers, including a transport stream (TS) decoding engine, and double speed MPEG2 decoding engine, and I-picture encoding engine of half HD with 30frames/s, two SDRAM controllers and two dedicated DSPs. The block diagram of the conventional HDTV video processor is shown in figure 8. There are two 32bit-wide 135MHz memory buses dedicated to access SDRAM controllers, and a 121.5MHz DSP bus. This communication architecture is quite application specific. If there are alterations on the memories configuration, the communication architecture should be reconstructed. In addition, because components can use the memory buses only for the memory access, there should be additional point-to-point links for other traffics among components.

For using communication resources efficiently, we replace the bus-based communication architecture with star-connected network architecture for the HDTV system as shown in figure 9. There are three crossbar switches: *SW1* constructs a forward network from master components to slave SDRAM controllers. *SW2* forms a reverse network from the slave SDRAM controllers to master components such as MC, DSP1, and Display Engines. Finally SW3 is for the communications between DSPs and MPEG2 Pipeline. The occasional traffics from MPEG2 Pipeline are aggregated into a *HUB* to avoid complicated links. Each link is 32bit-wide at the speed of 135MHz frequency.

The traffic characteristics between functional blocks are also shown in the figure 9. Thick arrows in the figure mean major traffics of bandwidth of giga-bit/s. We only analyze the *SW1* (the forward network) on which output conflicts occur among traffics from *DSP1*, *RC*, *MC*, and *Buffer Manager* to *SDRAM Controllers* for forward 2 channel play operation.

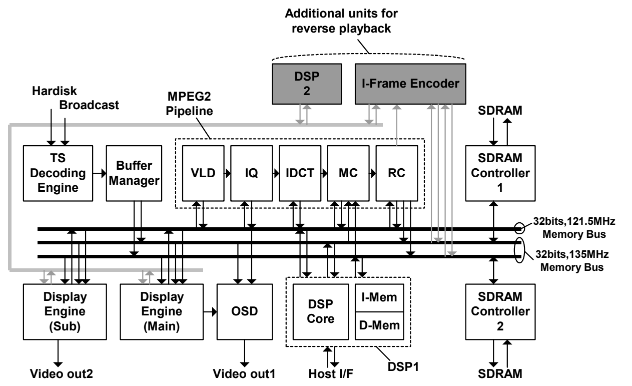
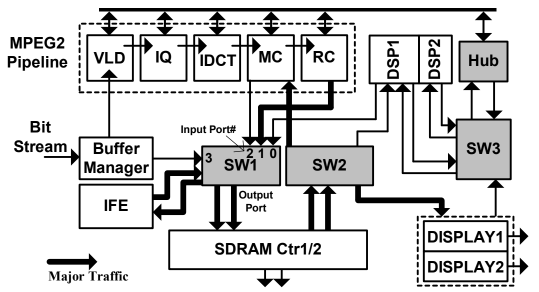


Figure 8. Conventional HDTV video processor with on-chip bus communication architecture [3]

The input bandwidth of port #0 is negligible. The bandwidth of other ports from 1 to 3 is 1.5Gbps, 240Mbps, and 160Mbps, respectively. The size of a packet is 32bits, and all traffics are bursty with burst length of 9 packets where the first packet has a memory address for writing to SDRAM. We adopt the proposed NA-MOO algorithm in the switch to analyze the performance such as packet latency or required buffer length. We also used a round robin based SLIP algorithm for comparison under same condition.



Source	Destination	Traffic Characteristic	
		Forward 2-ch play	Reverse play
BM	SDRAMC1/2	160Mbps	80Mbps
SDRAMC1/2	DSP1	160Mbps	80Mbps
SDRAMC1/2	MC	3Gbps	1.5Gbps
RC	SDRAMC1/2	1.5Gbps	0.75Gbps
SDRAMC1/2	DISPLAY(M/S)	3Gbps/3Gbps	1.5Gbps/1.5Gbps
IFE	SDRAMC1/2	-	1.5Gbps
RC	IFE	-	0.75Gbps
DSP1	VLD/IQ/IDCT/	RANDOM	RANDOM
	MC/RC		
DSP1	DISPLAY(1/2)	RANDOM	RANDOM
DSP2	IFE	RANDOM	RANDOM
IFE	DSP2	-	RANDOM
DSP2	DISPLAY(1/2)	RANDOM	RANDOM

Figure 9. On-chip networks for HDTV applications

In the NA-MOO scheduler, by allocating the negligible traffic (=port #0) beside the major traffic (=port #1), the port #1 of major traffic gets more grants than the other ports of #2, and #3. This is actually unfair among input ports as observed in section 3.2.

As a result shown in figure 10, the port #1 is served with shorter latency in NA-MOO scheduler than that in SLIP scheduler. At the same time, the minor traffic ports #2 and #3 are served with longer latency in NA-MOO than that in SLIP. The average packet latency over the whole packets on the switch is the same as that of SLIP scheduler. The required input buffer size of NA-MOO scheduler is almost the same as that of SLIP scheduler. By observing the plots, the characteristic on the ports seems to be more regulated in NA-MOO than that in SLIP.

Figure 11 shows packet latency distribution of whole packet on the switch for two kinds of schedulers. Due to the shorter latency of the major traffic (=port #1), less deviation on the packet latency distribution is obtained in the NA-MOO scheduler than that in the SLIP scheduler. There are two principal peaks in SLIP while one peak in NA-MOO.

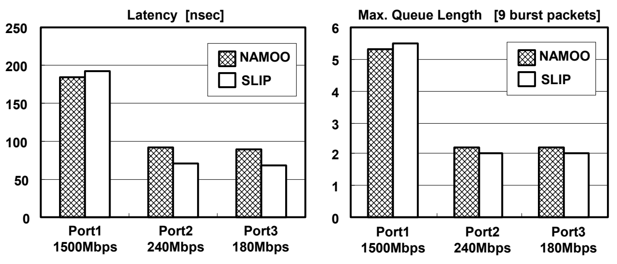


Figure 10. Packet latency and required input buffer size per port

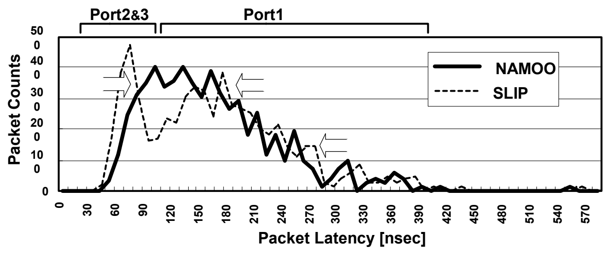


Figure 11. Packet latency distribution

By the simulation results for the HDTV SoC, the NA-MOO algorithm serves more *fairly* over input ports in the situation where the bandwidth of the input ports is uneven among the ports. Actually, in the SoC networks unlike the computer networks, the traffic patterns can be known in design stage, and those are not evenly distributed among links as shown in HDTV applications. This means that there are major traffics and minor traffics together on a switch.

From the analysis results of HDTV SoC, the NA-MOO scheduler is adequate for SoC networks in the aspect of silicon area, computing speed, and also the switching performance.

5. CONCLUSION

We have presented a lightweight crossbar switch scheduling algorithm especially for on-chip interconnection networks. The scheduler implementation of the proposed algorithm shows more than 60% area reduction and 20% computation delay reduction than conventional one with round robin based SLIP algorithm. Although the proposed algorithm does not always guarantee the fairness among input ports, once the traffic characteristics on the SoC networks are known, the algorithm serves more fairly among input ports.

The proposed algorithm demonstrates area-efficient and higher performance for the on-chip interconnection networks through an application example of HDTV SoC.

6. REFERENCES

[1] L. Benini and G. Micheli , "Networks on Chips: A New SoC Paradigm," in Computer Magazine, Vol.35 Issue:1, Jan. 2002, pp. 70-78

[2] Se-Joong Lee et al., "An 800MHz Star-Connected On-chip Network for Application to System on a chip," IEEE ISSCC Dig. Tech. Papers, Feb. 2003, pp. 468-469

[3] H. Yamauchi et al., "A 0.8W HDTV Video Processor with Simultaneous Decoding of Two MPEG2 MA@HL Streams and Capable of 30frames/s Reverse Playback," IEEE ISSCC Dig. Tech. Papers, Feb. 2002, pp. 372-373

[4] P. Gupta and N. McKeown, "Designing and Implementing a Fast Crossbar Scheduler," IEEE Micro, Vol. 19 Issue: 1,1999 Pp.20-28

[5] C. Huang et al., "Design of High-Performance CMOS Priority Encoders and Incrementer/Decrementers Using Multilevel Lookahead and Multilevel Folding Techniques," Solid State Circuits, IEEE Journal of, Vol.37 Issue:1, Jan. 2002 Pp.63-76