Design of a New Pipelined Router for NoC

Shih-Hsun Hsu
Department of Electrical Engineering
National Cheng Kung University
e-mail: sishin@j92a21.ee.ncku.edu.tw

Jer-Min Jou
Department of Electrical Engineering
National Cheng Kung University
e-mail: jou@j92a21.ee.ncku.edu.tw

Chien-Ming Sun
Department of Electrical Engineering
National Cheng Kung University
e-mail: scm@j92a21.ee.ncku.edu.tw

Ming-Chao Lee
Department of Electrical Engineering
National Cheng Kung University
e-mail: chao@j92a21.ee.ncku.edu.tw

Abstract

Networks-on-a-chip (NoC) is a new architectural template, which helps to meet many of challenges of designing a complex system-on-a-chip (SoC). In the paper, we introduce the on-chip network and propose the pipelined router for a NoC template called adaptive NoC (aNoC). In the network, we focus on the topology and the switching technique which provide scalability and low latency as the system expands, and they are key issues for the NoC; in the router design, we propose the generalized routing algorithm: routing and arbitration mechanisms to solve contentions. The proposed routing and arbitration units in the router can be easily modularized when we adopt different routing and arbitration algorithms. Furthermore, we accomplish the hardware design of a router containing the mechanism of pipelining to accomplish the transmission of a packet. The proposed router was developed as an IP of NoC which can be easily modularized for users adopting different number of ports, virtual channels, channel width, buffer size and different routing (including dynamic and static routing) and arbitration algorithms. The benchmark router with 5 ports, each port constants 4 32-bit virtual channels, can operate at 200MHz and the data rate can be up to 1.6Gbps. The performance of the router is enough for an HDTV application on NoC.

Keywords: NoC, router, wormhole, virtual channel and pipeline.

1. Introduction

Recently, a single chip may contain up to one billion transistors; with such massive resources, SoC designers face many challenges, including component-level issues such as performance and power, and system-level issues: reusability, adaptability, and scalability.

An efficient solution to these problems is to treat SoCs as *micronetworks*, and that is what our *adaptive Network-on-Chip* (aNoC) [2] does. An example of aNoC in Figure 1 has a regular structure and it provides network interfaces for easier component reuse and plug-n-play. The aNoC template decouples the computation (each processing elements, i.e. IPs, including all kinds of processor cores, memory, DSPs or FPGAs) and the communication (the communication between IPs via networks and routers) so that the design and synthesis of each part is simpler and can be done separately.

2. Networks of aNoC

2.1 Network Topology

From the perspective of network, we regard the composition of a computation component, a router and a network interface which binds the above 2 components as a *node* or *point* of the network. The topology of the network in aNoC is *2D mesh* topology, which is a kind of *direct network (point-to-point network)*. Comparing with other topologies of the interconnection network, the 2D mesh has a regular structure and is easier to map into a 2D layout than other topology. For the scalability of nodes and reducing the complexity of the connection between 2 nodes, we don't adopt the topology of *bus network* (*shared-medium network)* and *indirect network* such as crossbar, tree and multistage networks.

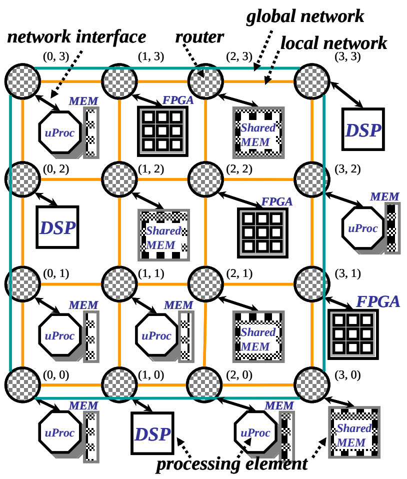


Figure 1: aNoC topology: 2D mesh

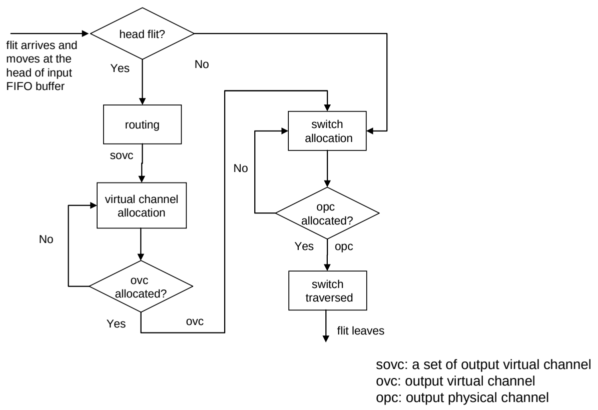
There are 2 hierarchical networks in the aNoC template, which are called *local network* and the *global network* from the lower to higher hierarchy respectively. The local network connects all adjacent nodes in the mesh and is used for the data transmission in the local area. For the scalability of the network and for reducing the latency of the transmission between 2 distant nodes, we add the extra global network between non-adjacent nodes [1]. The hop counts of the global network can be determined according to the size and the latency requirement of the system. Both in the local and the global network, there is a set of bidirectional full-duplex channel to connect 2 nodes. Take Figure 1 as an example, this is a 16-node 2D mesh network and the pair of (x, y) represents the address of nodes in the network. The hop count of the global network is 4.

2.2 Switching Technique

The switching technique of the aNoC network is the *wormhole routing*. In the regular packet switching, we divide a *message* into several *packets*. To decrease the latency of packet transmitting between the source and the destination node, we further divide each packet into several smaller flow control digits called *flits*, and only the *head flit* (i.e. the first flit of a packet) records the destination address of the packet for routing. Other flits in the same packet follow the path established by the head flit and these flits are transmitted in a pipelined-fashion. Besides, to increase the utilization of channels and to avoid *deadlocks* of the wormhole routing, we adopt a technique called *virtual channel* [5] which divides a single unidirectional *physical channel* into several *virtual channels*.

3. Proposed Routing and Arbitration Mechanism

Before introducing the mechanism we adopt, we first introduce the flow of a packet transmitting through our virtual channel router. Based on [4], the transmission of a packet through a virtual channel router can be divided into 4 steps shown in Figure 2: *routing*, *virtual channel allocation*, *switch allocation* and *switch traverse*.


head flit?
flit arrives and moves at the head of input FIFO buffer
Yes
No
routing
sovc
virtual channel allocation
No
ovc allocated?
Yes
ovc
switch allocation
No
opc allocated?
Yes
opc
switch traversed
flit leaves
sovc: a set of output virtual channel
ovc: output virtual channel
opc: output physical channel
Figure 2: The flow of a packet transmitting through a virtual channel router

3.1 Routing Mechanism

According to the destination address recorded in the head flit of a packet, we will get all possible output paths to the next router by the routing algorithm. The routing algorithm can be deterministic, adaptive, minimal path or non-minimal path. In the deterministic routing, we will get the fixed output path by the position of the destination node; however, in the adaptive routing, we will get the output path further by the workload of a channel. Minimal routing algorithm can route a packet along the shortest path, and a non-minimal routing algorithm can pass away the hot spots or broken nodes of the network. Besides the *dynamic routing* approach above, table lookup is a strategy of *static routing* for a known application mapped in the NoC system. In any case, the routing module in our proposed router only has to fit in with the following criterion: generating possible output virtual channels (i.e., a set of possible output virtual channels called *sovc* generated by the routing module) to the next router. In 2D mesh application of aNoC, we adopt the XY routing which is proven deadlock-free in 2D mesh topology and the routing module will generate all output virtual channels of a single output port as the *sovc* signal.

3.2 Proposed Virtual Channel Allocation Mechanism

When there are several individual virtual channels requesting the same output virtual channel for their individual packet transmission, we need a mechanism to solve the contention and the result of the allocation keeps during the transmission of an entire packet.

The virtual channel allocation can be regarded as a matching problem and it can be divided into 3 phases: *request*, *grant* and *accept*. For easier explanation of our mechanism and our definition of the priority on the arbiters, we consider the following example: 3 of 4 input virtual channels request their individual output virtual channels shown in Figure 3.

For easier identification of these input and output virtual channels, each input and output virtual channel has its own virtual channel number from 1 to 4. There is a pair of arbiters called *accept arbiter* and *grant arbiter* at each input virtual channel and output virtual channel respectively to accomplish matching. Each *grant arbiter* at each output virtual channel maintains the priority of all input virtual channels that want to request this output virtual channel. The initial priority of a *grant arbiter* in Figure 3 is determined by the order of the input virtual channel number. Each *accept arbiter* maintains the priority of all output virtual channels that this input virtual channel can request. The initial priority of an *accept arbiter* in Figure 3 is determined by the order of the output virtual channel number.

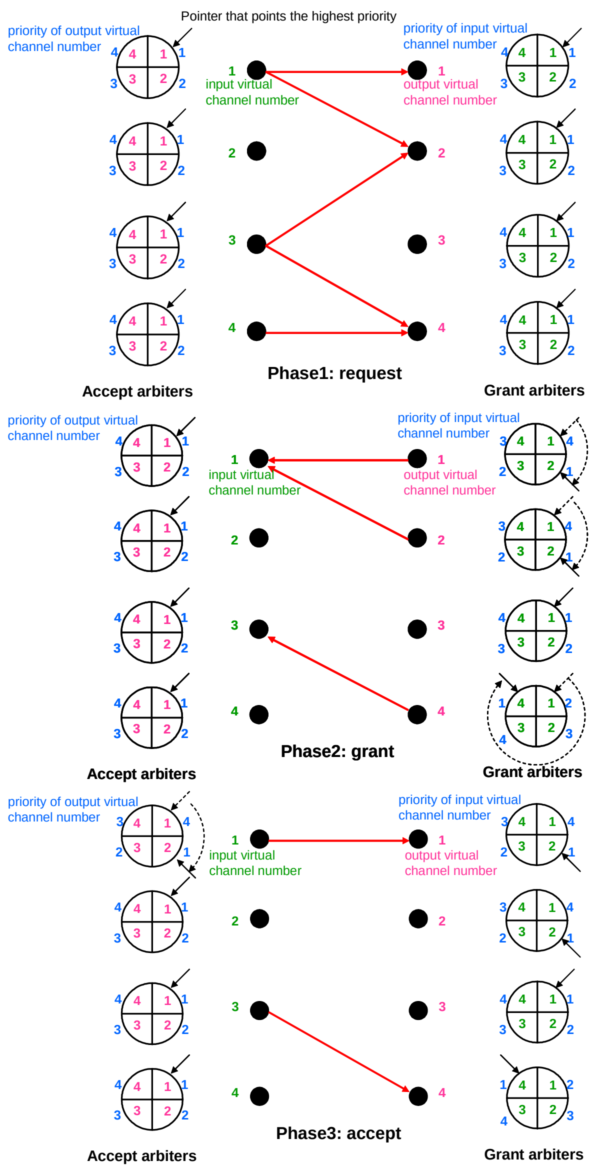

Pointer that points the highest priority
priority of output virtual channel number
priority of input virtual channel number
input virtual channel number
output virtual channel number
Phase1: request
Accept arbiters
Grant arbiters
Phase2: grant
Phase3: accept

Figure 3: Concept of virtual channel allocation

The arbitration algorithm used in both of the grant arbiter and the accept arbiter in Figure 3 is determined by a round-robin rule (which is a kind of fair scheme) for fairness. In order to avoid the *starvation* situation, we can adopt all kinds of *fair* arbitration algorithm in grant arbiters to find a maximal matching. In the example of Figure 3 we assume that priority of output virtual channels in each *accept arbiter* is 1>2>3>4, and the initial priority of input virtual channels in each *grant arbiter* is 1>2>3>4. 3 head flits in input virtual channel 1, 3 and 4 have gotten possible output virtual channels from the corresponding routing modules.

In the request phase, input virtual channel 1 requests for output virtual channel 1 and 2, input virtual channel 3 requests for output virtual channel 2 and 4, and input virtual channel 4 requests for output virtual channel 4 according to the output of the corresponding routing modules respectively. In the grant phase, output virtual channel 1 grants the request of input virtual channel 1. In Figure 3, there are 2 input virtual channels requesting for output virtual channel 2 and 4 respectively, and the *grant arbiter*s of these 2 output virtual channel only grant 1 request by the priority. Because the priority of input virtual channel 1 is grater than 3 in the *grant arbiter* of output virtual channel 2, and the priority of input virtual channel 3 is grater than 4 in the *grant arbiter* of output virtual channel 4, that's why the result of the grant phase. Please note that the change of priority in *grant arbiters* is round-robin for fairness. In the accept phase, input virtual channel 3 accepts the grant of output virtual channel 4.

Because there are 2 output virtual channels granting the same input virtual channel 1, the *accept arbiter* of input virtual channel 1 only accepts 1 request by the priority. Because the priority of output virtual channel 1 is grater than 2 in the *accept arbiter* of input virtual channel 1, that's why the result of the accept phase. With the 3 phase, we accomplish the virtual channel allocation of the first iteration. Those input virtual channels which fail to allocate any output virtual channels will re-allocate in the next iteration until they succeed.

3.3 Proposed Switch Allocation Mechanism

When there are several allocated input-output virtual channel pairs belonging to the same input physical channel or the same output physical channel, which are also the same input port or the same output port of a crossbar switch for flit transmission, we need a mechanism to solve the contention and the result of the allocation keeps during the transmission of each flit.

For easier explaining our mechanism and definition of the priority on arbiters for switch allocation, we consider the following example shown in Figure 4: 10 of 20 virtual channels (5 physical channels, Ei/Eo, Wi/Wo, Si/So, Ni/No, and fromIP/toIP, 4 virtual channels of each) requesting the usage of the crossbar switch.

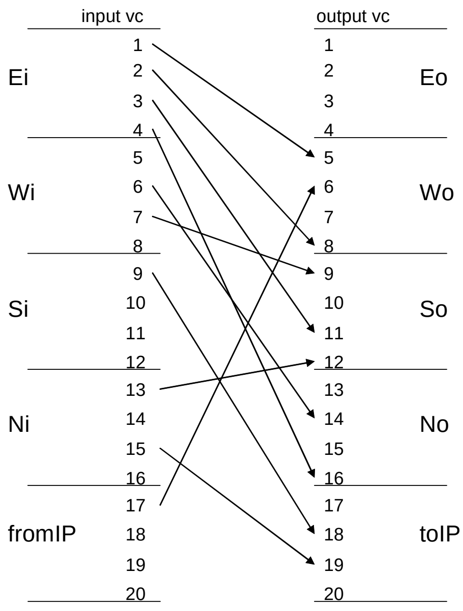


Figure 4: Example of switch allocation (5 physical channels, 4 virtual channels of each and 10 allocated input-output virtual channel pairs)

The switch allocation also can be regarded as a matching problem divided into 3 phases shown in Figure 5: *request*, *grant* and *accept*.

There is *n* pairs of accept arbiter and grant arbiter in a switch allocater, where *n* is the number of input/output port of a router. Each accept and grant arbiter maintains the priority of each input virtual channel and output virtual channel on the same physical channel respectively.

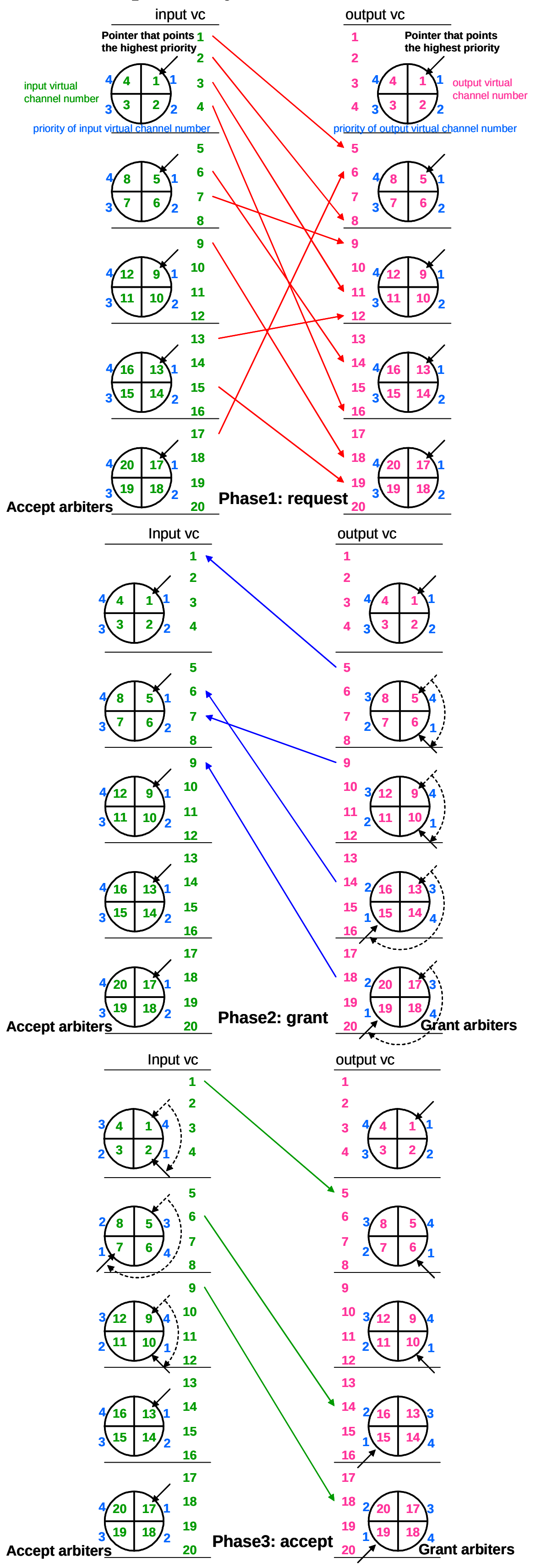

input vc
output vc
Pointer that points the highest priority
input virtual channel number
output virtual channel number
priority of input virtual channel number
priority of output virtual channel number
Accept arbiters
Phase1: request
Phase2: grant
Phase3: accept
Grant arbiters

Figure 5: Concept of switch allocation of the example of Figure 4

In order to avoid the *starvation* situation, we can adopt all kinds of *fair* arbitration algorithm in these grant and accept arbiters to find a maximal matching. The priority of these arbiters is determined by a round-robin rule. In the example of the Figure 5 we assume that the initial priority of input virtual channel in each accept arbiter is 1>2>3>4, 5>6>7>8…, and 17>18>19>20 respectively, and so are the initial priority of output virtual channels in each grant arbiter. In the request phase, input virtual channel 1, 2, 3 and 4 of input physical channel Ei request for output virtual 5, 9, 11 and 16 of output physical channel Wo, So and No respectively, and so do the input virtual channel 6, 7, 9, 13, 15, and 17 of input physical channel Wi, Si, Ni and fromIP respectively. In the grant phase, because there are 3 input virtual channels requesting for output virtual channel 5, 6 and 8 which are in the same output physical channel Wo, by the priority of the grant arbiter of output physical channel Wo, output virtual channel 5 grants the request from input virtual channel 1, and so do the output virtual channel 5, 9, 14 and 18 of output physical channel Wo, So, No and toIP respectively. In the accept phase, input virtual channel 1 of input physical channel Ei accepts the grant of output virtual channel 5. On the other hand, there are 2 output virtual channels granting input virtual channel 6 and 7 which are in the same input physical channel Wi, but input physical channel Wi only accepts 1 grant by the priority of the accept arbiter. Because the priority of input virtual channel 6 is grater than 7 in the accept arbiter of input physical channel Wi, that’s why the result of the accept phase.

4. Hardware Router Design

From the requirements of the switching technique, we proposed an input-buffered router and it provides the modularized routing module and arbitration units. Besides, the number of input/output ports (physical channels), the number of virtual channels per physical channel and the buffer size of each virtual channel can be parameterized in the design time. For the complex modules for virtual channel and switch allocation, we depict the dataflow of these modules to explain our design. The scheduling consideration is also mentioned.

4.1 Router Architecture

The general router architecture of *p* input/output ports, *v* virtual channels per port (physical channel) is shown in Figure 6. In the application of 2D mesh topology, *p* = 5.

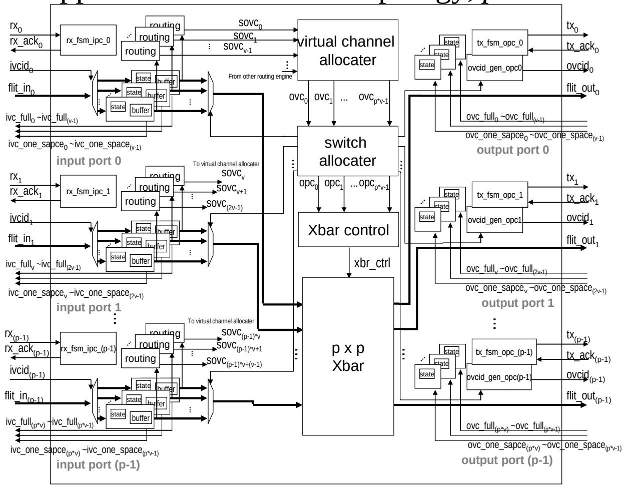


Figure 6: Router architecture

In each input port and output port there are modules dealing with the receiving and transmitting handshaking with other routers in the network. Each input/output virtual channel keeps its state. The state machine of each output virtual channel dominates the success of the virtual channel allocation and the switch allocation of each output virtual channel by maintaining the buffer usage of other routers via the *ovc_full* signal. The state machine of the input virtual channel dominates the operation of router and is mentioned latter. The input buffer of the router is separated into *v* individual lanes and supports an *ivc_full* signal to indicate other routers that the input buffer is full or not. Which lane the flits coming from other routers should be stored is decided by the *ivcid* (**I**nput **V**irtual **C**hannel **ID**entify) signal. There is a routing module belonging to each input virtual channel, which implements the routing algorithm of the router. The output of the routing module, *sovc* (**S**et of **O**utput **V**irtual **C**hannels) signal, sends to the virtual channel allocater for the virtual channel allocation to allocate 1 *ovc* (**O**utput **V**irtual **C**hannel) among 1 *sovc*. The switch allocater handles the switch allocation by the outcome of virtual channel allocation to decide which input virtual channel can send a flit through the crossbar switch to the next router. The number of input/output port of the crossbar switch equals the number of the physical channel, so that the complexity of crossbar will not increase with the increasing of virtual channels. In each output port, there is a module generating the *ovcid* (**O**utput **V**irtual **C**hannel **ID**entify) signal to indicate which lane the flit sent to other routers would enter by the switch allocation in this round.

4.2 Operation of Proposed Pipelined Router

Based on the theories of the *wormhole* and the *virtual channel* [4], we build a finite state machine for each input virtual channel shown in Figure 7 to control its proper operation of transmitting a flit.

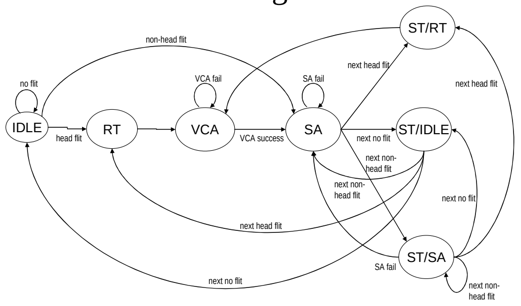


Figure 7: Finite state machine for each input virtual channel

There are 7 main states: IDLE, RT (*routing*), VCA (*virtual channel allocation*), SA (*switch allocation*), ST/IDLE (*switch traverse*), ST/RT (*switch traverse* and *routing* operating simultaneously), and ST/SA (*switch traverse* and *switch allocation* operating simultaneously). The initial state is IDLE. The ST/RT and ST/SA process 2 different flits in the same time and implement the pipeline operation.

When there is no flit in the input virtual channel, the input virtual channel keeps IDLE until there is a flit coming in. If a head flit comes, it will enter the RT state and activate the routing module to proceed routing. Otherwise, if a non-head flit comes, it will enter the SA state and activate the switch allocater to execute the switch allocation. After routing, the input virtual channel enters the VCA state and activates the virtual channel allocater to execute the virtual channel allocation. If the input virtual channel fails to get available *ovc* in the virtual channel allocation, it will keep in the VCA state. The input virtual channel getting available *ovc* enters the SA state to activate the switch allocater to allocate the usage of the crossbar switch. If the input virtual channel fails to allocate the usage of the crossbar switch in the switch allocation, it will keep in the SA state.

For the input virtual channel getting the usage of the crossbar switch to transmit a flit, there are 3 different state transitions depending on the next flit this input virtual channel want to process. If there is no next flit, the input virtual channel will enter the ST/IDLE state to set the control signal of the crossbar switch and the corresponding output virtual channel will activate the handshaking to accomplish the transmission of the flit. If the next flit is a head flit, the input virtual channel will enter the ST/RT state. In addition to the operation above, the input virtual channel also activate the routing module simultaneously. If the next flit is not a head flit, the input virtual channel will enter the ST/SA state. In addition to the operation of ST/IDLE, the input virtual channel also activate the switch allocation of the next flit simultaneously.

For the ST/IDLE state, there are also 3 different state transitions depending on the next flit. If there is no next flit, the input virtual channel will enter the IDLE state. If the next flit is a head flit, the input virtual channel will enter the RT state; otherwise it will enter the SA state. For the ST/RT state, after the routing module accomplishing routing the next flit and the present flit is transmitted, the input virtual channel will enter the VCA state to process the virtual channel allocation of the next flit.

For the ST/SA state, there are also 4 different state transitions depending on the next flit, and we have to consider 2 situations: the switch allocation of the ST/SA state is successful or not. If the switch allocation fails, the input virtual channel will enter the SA state. If the switch allocation succeeds, we have to consider another 3 situations. If the there is no next flit, the input virtual channel will enter the ST/IDLE state. If the next flit is a head flit, the input virtual channel will enter the ST/RT state; otherwise it will enter the ST/SA state again.

4.3 Dataflow of Proposed Virtual Channel Allocation Mechanism

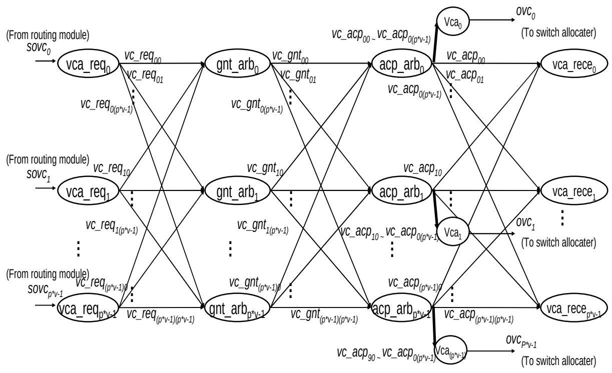


Figure 8: Dataflow of the virtual channel allocater for a router with *p*n* virtual channels

The dataflow of a virtual channel allocater for *p*n* virtual channels is shown in Figure 8. From left to right, it contains *p*n* modules generating the request signal for output virtual channels, *p*n* grant arbiters for the 1st arbitration of conflicted request signals for the same output virtual channel, *p*n* accept arbiters for the 2nd arbitration of the conflicted grant signals form output virtual channels for the same input virtual channel, *p*n* modules generating the *ovc* signal, which means the allocated output virtual channel of each input virtual channel, to the switch allocater and *p*n* modules for each output virtual channel to receive the final matched input virtual channel of the allocation.

Each grant arbiter maintains the priority of all input virtual channels that want to request this output virtual channel. Each accept arbiter maintains the priority of all output virtual channels that this input virtual channel can request. In order to avoid the *starvation* situation, we can adopt all kinds of *fair* arbitration algorithm in grant arbiters and accept arbiters to find a maximal matching in the virtual channel allocation.

4.4 Dataflow of Proposed Switch Allocation Mechanism

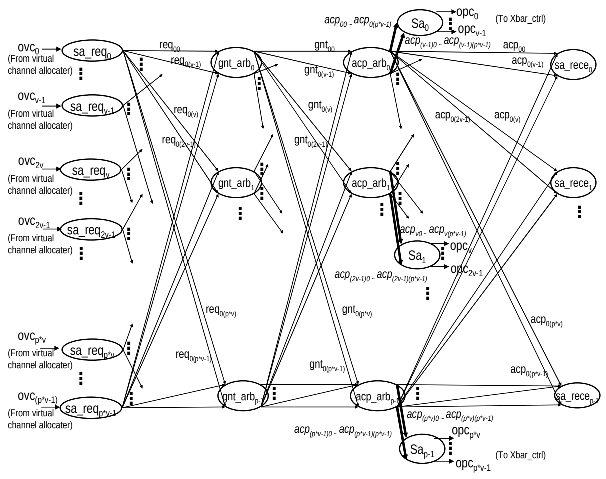


Figure 9: Dataflow of the switch allocater for *p* port router, each port with *n* virtual channels

The dataflow of a switch allocater for *p* port router, each port *n* virtual channels is shown in Figure 9. From left to right, it contains *p*n* modules generating the request signal for output virtual channels, *p* grant arbiters for the 1st arbitration of conflicted request signals for the same output port, *p* accept arbiters for the 2nd arbitration of the conflicted grant signals form output virtual channels for the same input port, *p* modules generating the *opc* signal, which means the input/output virtual channel pair that can transmit a flit, and *p* modules for each output virtual channel to receive the final matched input virtual channel of the allocation.

Each accept and grant arbiter maintains the priority of each input virtual channel and output virtual channel on the same physical channel respectively. In order to avoid the *starvation* situation, we can adopt all kinds of *fair* arbitration algorithm in these grant arbiters and accept arbiters to find a maximal matching in the switch allocation.

4.5 Scheduling of Proposed Router

The operation of the proposed router is scheduled as Figure 10.

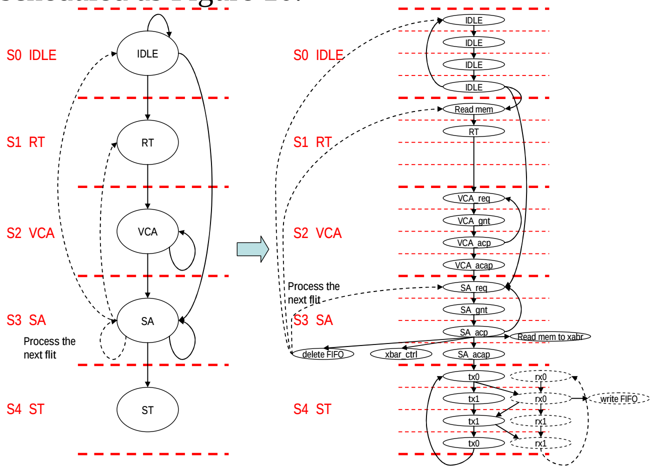


Figure 10: Scheduling of the proposed pipelined router

In the left of Figure 10 is the scheduling of the finite state machine depicted in Figure 7 and we schedule the router as 1 time unit to accomplish IDLE, RT, VCA, SA, and ST respectively. The dotted lines between states mean the router will process the next flit at the same time which accomplishes the pipelining.

From the dataflow analysis of virtual channel allocation and switch allocation, we discover that they are accomplished in 4 dependent operations:

1. generating *request*
2. 1st arbitration and generating *grant*
3. 2nd arbitration and generating *accept*
4. receiving *accept*

Thus we can let VCA and SA finished in 4 smaller time units depicted in the right of Figure 10. On the other hand, the ST (*switch traverse*) is implemented by 4-phase handshaking with other adjacent routers to accomplish the flit transmission, so it can also be finished in 4 smaller time units. Because our router is pipelined, the other operations such as IDLE, RT must be accomplished in 4 time units. If users adopt other complex routing algorithms in RT, they will have more 2 time units as the slack without decreasing the overall performance of the router. We make 1 time unit at the right of Figure 10 as 1 clock cycle, thus the pipeline latency of transmitting a flit is 4 cycles. The detailed finite state machine for each input virtual channel is shown in Figure 11 and it is also the lower hierarchy of the finite state machine depicted in Figure 7.

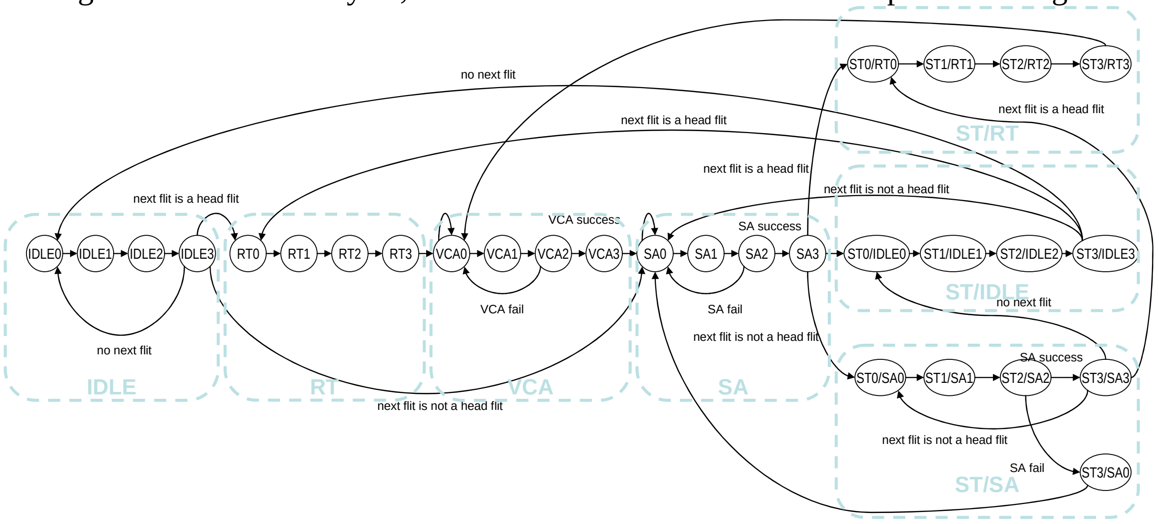


Figure 11: Detailed scheduling of the proposed pipelined router (lower hierarchy of Figure 7)

5. Verification and Performance

We completed the router design form the high level 3000-line C behavioral model, through the 6000-line synthesizable Verilog RTL code, to the low level physical layout. We employ the XY routing algorithm in the routing module for the 2D mesh routing, and the fair round-robin arbitration in both of the virtual channel allocation and switch allocation as the benchmark to verify the correctness of the router. The output of the router under test is connected to the “pseudo input of the next router” to verify the correctness of flit transmitting with pattern of 9 flits in each input port. The parameter of the benchmark router is 5 ports, each port with 4 virtual channels, and the size of input buffer of each lane contains four 32-bit flits. Each packet contains 4 flits in our test pattern.

To balance the time consuming among the virtual channel allocation, the switch allocation and the 4-phase handshaking in the switch traverse, each operation of the router takes 4 cycles. For a packet which doesn’t face blocking, a head flit takes 16 clocks to transmit, and other flits of the same packet take 8 clocks to transmit. With the pipeline design, the router takes 4 clocks to send a non head flit. After verifying the pre-synthesis code, we synthesized our router based on UMC Artisan 0.18 μm cell-library by the Synopsys Design Analyzer. After verifying the correctness of the post-synthesis code, we also accomplished the physical layout of the router by Cadence SoC Encounter shown in Figure 12.

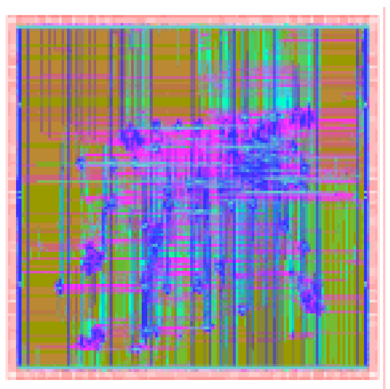


Figure 12: Physical layout of the NoC router

In the corner of typical operation, with rise/fall time and skew value of the clock signal of 0.1 nano second, the overall router chip spec. and the maximal delay of each stage are shown in the Table I and Table II respectively. The bandwidth of each port of the benchmark router is calculated below:

Bandwidth of each port
= [(200MHz / 4)*32 bit]
= 1600Mbps
= 1.6Gbps

The bandwidth is sufficient for the HDTV application mapped to an on-chip network mentioned in [3].

Table I: Pipelined NoC Router Spec.

Clock Frequency	200MHz
Data rate (32-bit channel)	1.6Gbps
Area (core)	1073945.125000 μm^2
Operating Voltage	1.8V
Power consumption	932.5797 mW (@200MHz)
Pipeline latency	4 cycles
Port Number	5
Virtual Channel Number / Port	4
Buffer Size / Input Virtual Channel	4 flits

Table II: Max. Delay

Operation	Time (ns)
RT	2.07
VCA	3.13
SA	3.19
ST/RT, ST/SA	3.19
ST/IDLE	1.93

In the NoC router project, we also develop an ASIP for the NoC router [6]. The performance running the same pattern between the hard-ware pipelined router and soft-ware ASIP router is shown in Table III. The result shows the speed advantage of the hardware implementation.

Table III: Speed advantage of the benchmark NoC Router implemented in ASIC over which implemented in ASIP

	Time (cycles)	Speedup ratio
H.W. pipelined router	60	9057
S.W. ASIP router	543445	1

6. Conclusion and Future Work

In this paper, we presented the chip design of a pipelined NoC router. The router architecture, the transmission mechanism of a packet, the virtual channel allocation and switch allocation are introduced explicitly. The user can adopt the parameters, routing algorithm for routing module and fair arbitration algorithm for virtual channel and switch allocation in our IP of router to suit the application they need. With these components, our router let the aNoC provide highly scalability and adaptivity. We are going to develop the high speed arbiter to overcome the bottleneck of the virtual channel and switch allocation and increase the overall operation speed of the router. The power analysis and low-power design is another research topic of the NoC router.

Acknowledgements

The research is partial supported by National Science Council under contract no. NSC-93-2213-E-006-110.

7. References

[1]Hangsheng Wang et al. ”Power-driven Design of Router Microarchitectures in On-chip Networks”, IEEE/ACM on Microarchitecture, 2003.

[2]Jer-Min Jou et al. “Adaptive Network-on-a-Chip Architecture System Design”, the 46th IEEE International Midwest Symposium on Circuits and Systems, 2003.

[3]Kangmin Lee, Se-Joong Lee, and Hoi-Jun Yoo, “A Distributed Crossbar Switch Scheduler for On-Chip Networks”, CICC 2003.

[4]Li-Shiuan Peh et al. ”A delay model and speculative architecture for pipelined routers”, International Symposium on High-Performance Computer Architecture, January, 2001.

[5]William J. Dally, ”Virtual Channel Flow Control”, IEEE Transactions on Parallel and Distributed Systems, vol. 3, no. 2, pp.194-205, March, 1992.

[6]Zi-lun Wang, et al. “Design of a Low Power ASIP for NoC Routers”, submitted to the WCEsp, 2005.