PACKET TRANSFER BETWEEN ROUTERS WITH CUT-THROUGH FLOW CONTROL FOR MPP COMPUTERS CONNECTED IN A DOUBLE-LOOP HYPERCUBE INTERCONNECTION NETWORK

Milen Georgiev Angelov
Technical University – Varna, Chief Assistant Professor, PhD, Engineer

Abstract: *Data flow control on the routers connecting the nodes in multiprocessor MPP systems allocates their buffers and the bandwidth of their channels to the data units – packets and flits. The efficient implementation of data transfer is of essential importance for achieving high throughput and low latency. In this paper, a study of a variant of packet transfer between the output channel and the input buffer of two neighboring routers for a Double-loop Hypercube (DLH) interconnection network is presented. The transfer is implemented at a low (hardware) level over a bidirectional full-duplex channel using Virtual Cut-Through flow control. The functional units required for this purpose, their operating algorithms, and the control signals are presented. The software component developed on this basis implements the packet transfer simulation and verifies the correctness of the architectural solutions through the obtained results. The chosen approach achieves high speed in resource switching at the beginning and the end of the transmission, as well as a high degree of channel utilization and throughput.*

Keywords: *Router, Virtual Cut-Through (VCT), input buffer, FIFO queue, output channel, Round Robin Arbiter (RRA), Flow Control Digit (flit)*

I Introduction

The router is the primary functional unit of high-performance interconnection networks for parallel computers [8]. The design of a router for an MPP system includes the selection of network topology, routing algorithm, and flow control technique. Subsequently, the virtual channels, buffer organization, switches, and pipeline organization are determined. The main goal is achieving minimum latency in the transfer of data units – flits and packets.

The main principles and means used in creating the architectural platform of the router under consideration are:

1). Interconnection network: Static, with DLH topology, decentralized control, and asynchronous communication;

2). Routing: Minimal adaptive routing, allowing a packet to select one of several possible shortest paths [4, 5];

3). General architecture (Fig. 1): Input buffers with pools of queues; input buffers directly connected to the Crossbar switch [3]; arbitration at each output channel;

4). Switching technique: Data exchange at the packet level using VCT switching technique, with pipeline organization at the flit level for data receiving and sending;

5). Input buffers: Based on a pool of FIFO queues;

6). Packet sending: A packet consists of one header (Header) flit, a body of several flits (Body), and one tail flit (Tail). The information in the header flit determines its route. In this case, the packet may have variable length not exceeding 16 flits, with each flit being 32 bits wide;

7). Packet route: An ordered sequence of nodes in the network from sender to receiver. The resources used by the packet during transfer are queues, switching units (Crossbar), output registers, and links. After a packet's Header flit is received into a queue, the router determines the output channel based on resource state (occupied buffers and output channel status) using the minimal adaptive routing algorithm. Allocation of input queues to output channels is performed using the iSLIP algorithm with RRA arbiters at the outputs [1];

8). Pipelining: The three separate stages of data transfer at the physical level (Fig. 1) are: Writing a flit into a free and selected queue of the input buffer (Flit Write); Routing, arbitration, and path setting through the switch (Routing, Arbitration and Path Setting); Switch Traversal [1, 2, 3].

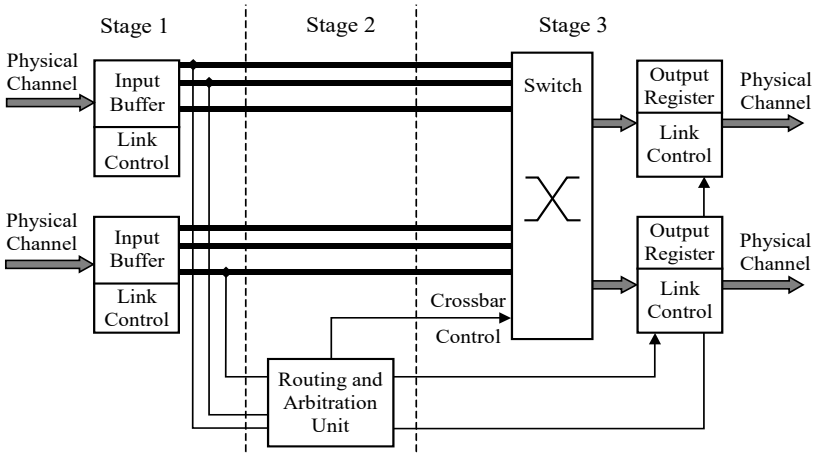

Fig. 1. Flit transfer pipeline organization in a router

II Architecture of the router's input buffer

Input buffers of the router are critical for achieving low latency and high throughput during packet transfer [11]. Fig. 2 shows the architecture of the input buffer. It implements the VCT switching technique, enabling packet-level transfer. The input buffer consists of a demultiplexer DEMUX, eight FIFO queues, a PRIORITY ENCODER, a B_FIFO_STATUS register for queue status, a DC decoder for FIFO queue selection, and control logic. The necessary control signals are defined and described in the program section below. In this architecture the main principles are:

1). Use of a pool of FIFO queues for each buffer;

2). Presence of a separate local arbiter ARB[k] (in this case kϵ[1..8]) for each queue;

3). The FIFO queues of all input channels are directly connected to the switch (Fig. 1), which allows more than one packet to be sent from one input buffer to the output channels in a given time interval.

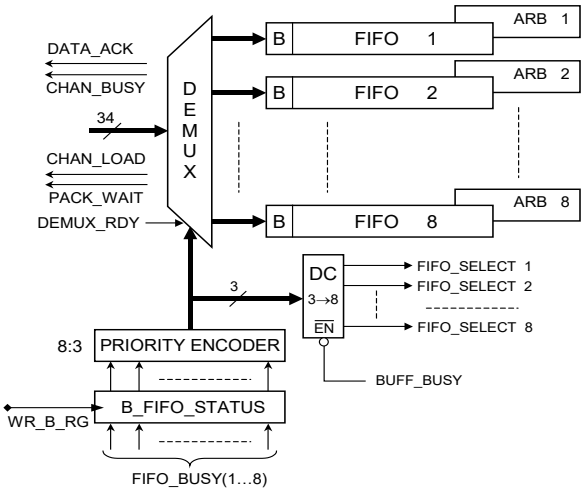

Fig. 2. Architecture of the router's input buffer and its control signals

III Architecture of the Allocator

Fig. 3 shows in general form the architecture of the router's two-stage allocator, which forms the basis of the routing and arbitration block (Fig. 1). For clarity, in addition to the

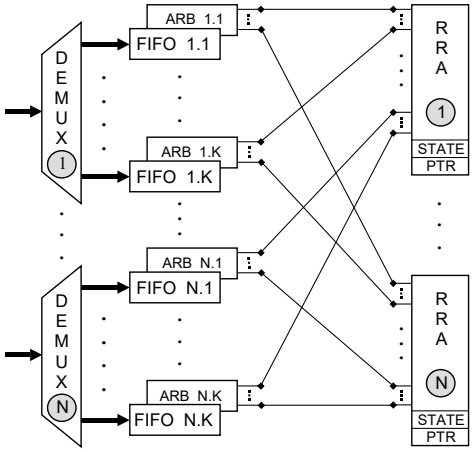

Fig. 3. Architecture of the router's two-stage allocator

arbiters (ARB), the demultiplexers (DEMUX) and the queues (FIFO) of the input channels are also shown. The allocator consists of two groups of arbiters: input arbiters (ARB) and output arbiters (RRA). The routing algorithm is implemented in hardware inside the router, with part of this implementation residing in the allocator itself. The following operating principle is used:

After a FIFO queue in an input buffer receives the Header flit of an arriving packet, it issues a request to its local arbiter ARB, which starts the routing and arbitration phase for that packet. This phase consists of three steps: Request-Grant-Accept.

At the beginning of each allocation time interval, every input-queue arbiter that is not currently involved in a concurrence may issue one or more requests (Request) to the output-channel RRAs. The number of requests depends on the possible routes the arriving packet can take to its destination and on the state of the output channels (free / busy / disabled). Each output RRA (if its channel is not masked, not busy, and has at least one request) selects the highest-priority request and returns a grant (Grant) to its sender. If an input arbiter receives one or more grants, it selects the highest-priority one through its priority logic according to local network conditions and replies (Accept) to the RRA whose grant it accepted. An input/output pair consisting of a FIFO queue and an output channel is formed. A data path is established through the router's crossbar switch (Fig. 1) and packet transfer from the selected queue to the output channel begins immediately. After the entire packet has been sent, the resources are released and can participate in subsequent allocations.

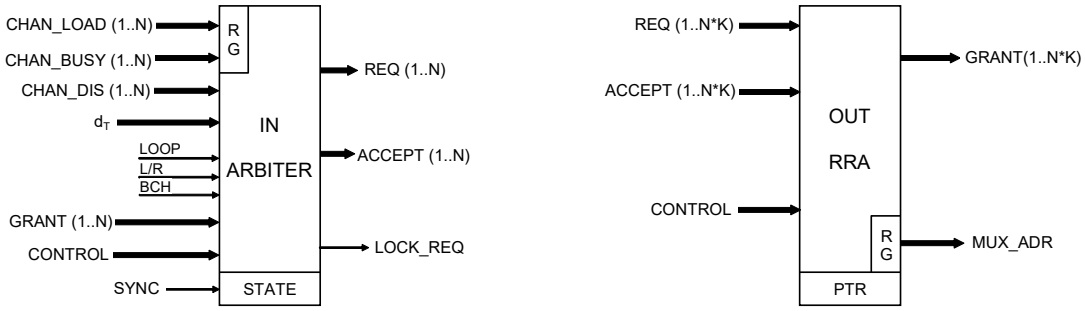

Fig. 4. General view of input arbiter, output arbiter and their signals

All arbiters in the allocator operate in parallel and synchronously. In one time interval the iSLIP algorithm performs a single iteration; depending on the state of the input queues, output channels and their RRAs, the allocator may realize 0, 1 or more matches. Conflicts and contention occur only at the output channels, which is an important factor for ensuring maximum throughput through the router [6, 7]. Fig. 4 generally shows the two types of arbiters in the allocator and the important signal groups (REQ, GRANT and ACCEPT) required for arbitration.

IV Architecture of the output RRA

The main goal in designing the RRA is to implement fair arbitration with maximum speed [9, 10]. Fig. 5 presents the architecture of the RRA for an output channel with **n** inputs

and **n** outputs, where **n = N×K**; in this case n=96. The input and output signals correspond to those shown in Fig. 4, while the control signals are described below. The main blocks are:

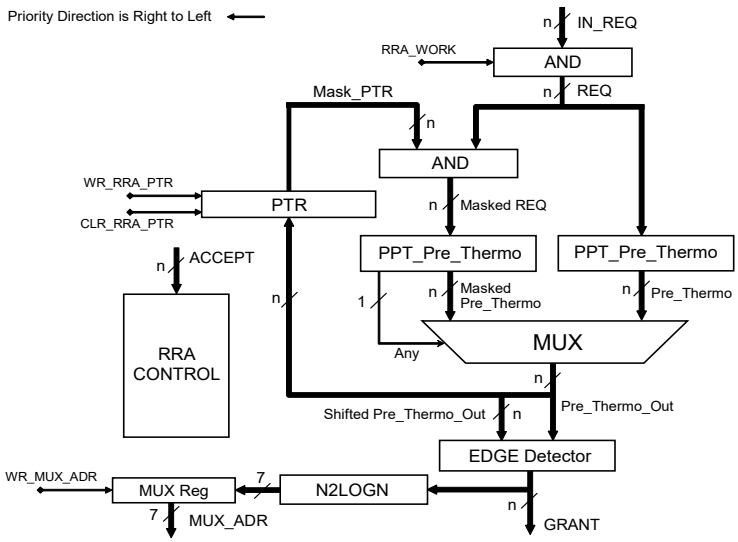

Fig. 5. Architecture of the Round-Robin Arbiter for an output channel

1). PPT_Pre_Thermo: A combinational logic circuit with **n** inputs and **n** outputs that performs thermometer encoding on an arbitrary input binary vector. In this case, two PPT_Pre_Thermo encoders are used, operating in parallel on the input vector and on the masked input request vector;

2). MUX: An **n**-bit 2→1 multiplexer for selecting one of the two PPT_Pre_Thermo outputs;

3). EDGE Detector: A single-level combinational logic circuit with **2n** inputs and **n** outputs that generates the output GRANT vector;

4). PTR: An **n**-bit register that stores the current round-robin priority pointer of the RRA;

5). AND: A logic block for bitwise masking of the input requests according to the content of PTR;

6). N2LOGN: An encoder that performs **n→log$_2$n** encoding, indexing in binary the position of the active bit in the GRANT vector. If this response is eventually accepted, the code will be written into MUX_Reg and will address the input queue selected by the arbiter;

7). RRA CONTROL: Control block, part of the output channel control.

V RRA management and the router's output channel

Fig. 6 shows the state graph of the finite state machine of one output channel, together with the control signals and possible transitions. The main components of the output channels are the output register, the header flit modification block, and the control logic (not shown here). The RRAs are part of the router's allocator but are controlled by the output channels.

Below are the states and the signals (defined and described in the program section below) that are asserted in each of them:

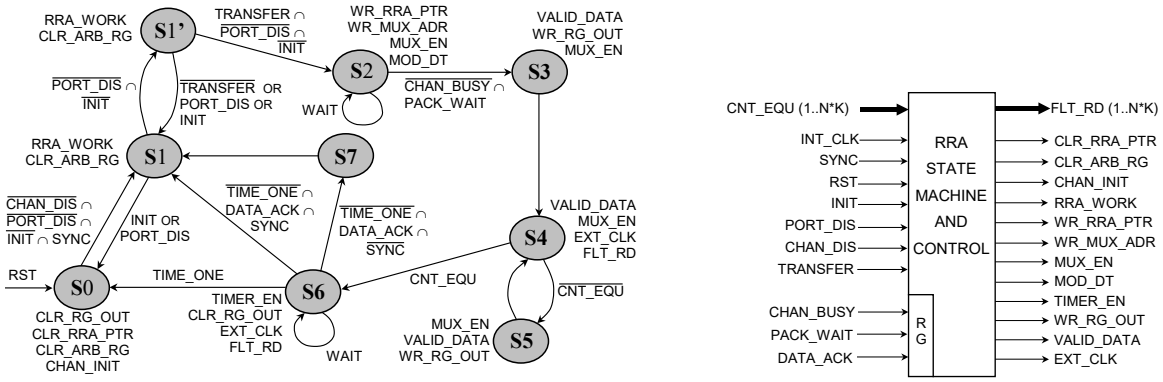

Fig. 6. State diagram of the finite state machine controlling one output channel and its signals

S0: Initial state *(Init)*. The signals CLR_RRA_PTR, CLR_ARB_RG, CLR_RG_OUT and CHAN_INIT are generated;

S1: First state of the arbitration phase *(Routing_and_Arbitration_1)*. The signals RRA_WORK and CLR_ARB_RG are generated;

S1': Second state of the arbitration phase *(Routing_and_Arbitration_2)*. The signals RRA_WORK and CLR_ARB_RG are generated;

S2: Path setting through the switch *(Path_Setting)*. Along this path the packet will move from the selected input queue to the output channel. The signals WR_RRA_PTR, WR_MUX_ADR, MUX_EN and MOD_DT are generated;

S3: Start of a packet transfer *(Transfer_Begin)*. The signals MUX_EN, WR_RG_OUT and VALID_DATA are generated, the latter indicating valid data between two neighboring routers. The modified HEADER flit is written into the output register;

S4: Transfer state 1 *(Transfer_1)*. The signals MUX_EN, VALID_DATA and EXT_CLK are generated; they perform the write of the currently transmitted flit into the input buffer of the next router in the chain. The signal FLT_RD increments the read counter and addresses the next flit in the selected FIFO queue;

S5: Transfer state 2 *(Transfer_2)*. The signals MUX_EN, VALID_DATA and WR_RG_OUT are generated;

S6: End of transfer *(Transfer_End)*. Awaits hand check of a successfully received packet from the downstream router via the DATA_ACK signal. The signals CLR_ARB_RG, CLR_RG_OUT, EXT_CLK, FLT_RD and TIMER_EN are generated;

S7: Synchronization state *(Synchronization)*. An intermediate state in which the routing and arbitration phase is synchronized via the SYNC signal.

VI Program section for controlling one output channel

In the program fragment presented below, the algorithm of the finite state machine for controlling one output channel is coded, written in Verilog (a hardware description language used for the design, simulation, and verification of electronic circuits).

```
// Design Name:          HPC_Router
// Module Name:          Router_ OUT_Channel_FSM
// Project Name:         Router_ OUT_Channel
//
// **   DECLARATIONS,  INITIALIZATION,  CORE LOGIC
`timescale 1ns/1ps              // Modeling Time, Step of Modeling
//
//         Constants
`define  DLY                    0.5             // Signals Delay on the Device
`define  TMR_bits               3               // Width of timeout Timer
`define  Timer_Start_Value      3'b 111         // Current Load Value of Timer
//
//         Main Program
// **-------------------------------------------------------------------------------------------**
module Router_OUT_Channel_FSM (
   input  RST,                  // Asynchronous RESET with active HIGH Level, from HOST
   input  INT_CLK,              // FSM Clock, from Local Node
   input  INIT,                 // Initialization by DMA with active HIGH Level
   input  SYNC,                 // Synchronization, Tsync = 2 x Tint_clk
   input  PORT_DIS,             // Disable the Output Channel during the Initialization Process
   input  CHAN_DIS,             // Disable the Output Channel until the Next Initialization
   input  TRANSFER,             // Success Arbitration in the Current Cycle
   input  CHAN_BUSY,            // The Correspondent IN Channel into other Neighboring Router is BUSY
   input  PACK_WAIT,            //The Correspondent IN Channel into other Neighboring Router wait Packet
   input  DATA_ACK,             // The Correspondent IN Channel successfully accepted a Packet
   input  CNT_EQU,              // End of Received Flit
   input  TIME_ONE,             // TimeOut, from Counter into FSM
//
   output FLT_RD,               // Increment RD Counter
   output CLR_RG_OUT,           // Clear the OUT Register into OUT Channel RRA, from FSM
   output CLR_RRA_PTR,          // Clear the Pointer into OUT Channel RRA, from FSM
   output CLR_ARB_RG,           // Clear the IN FSM Fix Register, from FSM
   output CHAN_INIT,            // The OUT Channel FSM is in the INIT State, from FSM
   output RRA_WORK,             // Enable the working RRA States, from FSM
   output WR_RRA_PTR,           // Fix Current Circle Pointer Value into Register, from FSM
   output WR_MUX_ADR,           // Fix the Address of Selected Queue into Register, from FSM
   output MUX_EN,               // Enable Multiplexers for Selected Data Path, from FSM
   output MOD_DT,               // Enable Modification of the Header Flit, from FSM
   output TIMER_EN,             // Enable Counter -> Timer, from FSM
   output WR_RG_OUT,            // Fix Current Flit into OUT Register, from FSM
   output VALID_DATA,           // Output Data Lines are Valid, from FSM
   output EXT_CLK,              // Write Data into Neighbor Router IN Buffer,       from FSM
   output Timer );              // State of Timer
```

```
//
        reg[14:0]  FSM_OUT_Channel_State;
//
// **   FSM_OUT_Channel_State have 9 States
// **-------------------------------------------------------------------------------------------------**
        parameter[14:0]
INIT_State                          = 15'b 001_1110_0000_0000,      // State S0
ROUTING_ARBITRATION_1_State         = 15'b 000_0101_0000_0000,      // State S1
ROUTING_ARBITRATION_2_State         = 15'b 100_0101_0000_0000,      // State S1'
PATH_SETTING_State                  = 15'b 000_0000_1111_0000,      // State S2
TRANSFER_BEGIN_State                = 15'b 000_0000_0010_0110,      // State S3
TRANSFER_1_State                    = 15'b 010_0000_0010_0011,      // State S4
TRANSFER_2_State                    = 15'b 100_0000_0010_0110,      // State S5
TRANSFER_END_State                  = 15'b 011_0000_0000_1001,      // State S6
SYNCHRONIZATION_State               = 15'b 000_0000_0000_0000;      // State S7
//
// **   FSM Outputs
// **-------------------------------------------------------------------------------------------------**
assign HLP_BIT          = FSM_OUT_Channel_State [14];
assign FLT_RD           = FSM_OUT_Channel_State [13];
assign CLR_RG_OUT   = FSM_OUT_Channel_State [12];
assign CLR_RRA_PTR = FSM_OUT_Channel_State [11];
assign CLR_ARB_RG   = FSM_OUT_Channel_State [10];
assign CHAN_INIT        = FSM_OUT_Channel_State [9];
assign RRA_WORK     = FSM_OUT_Channel_State [8];
assign WR_RRA_PTR   = FSM_OUT_Channel_State [7];
assign WR_MUX_ADR= FSM_OUT_Channel_State [6];
assign MUX_EN           = FSM_OUT_Channel_State [5];
assign MOD_DT           = FSM_OUT_Channel_State [4];
assign TIMER_EN         = FSM_OUT_Channel_State [3];
assign WR_RG_OUT    = FSM_OUT_Channel_State [2];
assign VALID_DATA   = FSM_OUT_Channel_State [1];
assign EXT_CLK          = FSM_OUT_Channel_State [0];
//
        reg[`TMR_bits-1:0]    Timer;                    // TimeOut Timer_1
//
// ---------------------------------------------------------------------------------------------------
initial                             // Use initial One Time in the Beginning
   begin
      FSM_OUT_Channel_State = INIT_State;
   end
//
//      All Conditions in one Body
// ------------------------------------------------------
always @(posedge INT_CLK or posedge RST) begin
        if (RST) begin                                          // First is the RESET
          FSM_OUT_Channel_State <= #`DLY INIT_State;
          Timer <= #`DLY `Timer_Start_Value;
        end
//
        else begin
//
//      Change the State of FSM, if they are Conditions
//------------------------------------------------------
        case (FSM_OUT_Channel_State)
```

```
//
            INIT_State: begin                                   // S0
               if (~CHAN_DIS & ~PORT_DIS & ~INIT & SYNC) begin
                  FSM_OUT_Channel_State <= #`DLY ROUTING_ARBITRATION_1_State;
               end
             end
//
            ROUTING_ARBITRATION_1_State: begin          // S1
               if (INIT | PORT_DIS) begin
                  FSM_OUT_Channel_State <= #`DLY INIT_State;
                end
                else begin
                  if (~INIT & ~PORT_DIS) begin
                     FSM_OUT_Channel_State <= #`DLY ROUTING_ARBITRATION_2_State;
                     Timer <= #`DLY `Timer_Start_Value;
                  end
               end
            end
//
            ROUTING_ARBITRATION_2_State: begin          // S1'
               if (~TRANSFER | PORT_DIS | INIT) begin
                  FSM_OUT_Channel_State <= #`DLY ROUTING_ARBITRATION_1_State;
               end
               else begin
                  if (TRANSFER & ~PORT_DIS &~INIT) begin
                     FSM_OUT_Channel_State <= #`DLY PATH_SETTING_State;
                  end
               end
            end
//
            PATH_SETTING_State: begin                   // S2
               if (~CHAN_BUSY & PACK_WAIT) begin
                  FSM_OUT_Channel_State <= #`DLY TRANSFER_BEGIN_State;
               end
            end
//
            TRANSFER_BEGIN_State: begin                 // S3
              FSM_OUT_Channel_State <= #`DLY TRANSFER_1_State;
            end
//
             TRANSFER_1_State: begin                    // S4
               if (~CNT_EQU) begin
                  FSM_OUT_Channel_State <= #`DLY TRANSFER_2_State;
               end
               else begin
                  if (CNT_EQU) begin
                     FSM_OUT_Channel_State <= #`DLY TRANSFER_END_State;
                  end
                end
             end
//
             TRANSFER_2_State: begin                    // S5
                FSM_OUT_Channel_State <= #`DLY TRANSFER_1_State;
             end
//
```

```
            TRANSFER_END_State: begin                    // S6
               if (Timer == 0) begin
                 FSM_OUT_Channel_State <= #`DLY INIT_State;
               end
               else begin
                 if ((Timer != 0) & DATA_ACK & SYNC) begin
                    FSM_OUT_Channel_State <= #`DLY ROUTING_ARBITRATION_1_State;
                 end
                 else begin
                    if ((Timer != 0) & DATA_ACK & ~SYNC) begin
                       FSM_OUT_Channel_State <= #`DLY SYNCHRONIZATION_State;
                    end
                    else Timer <= #`DLY Timer - 1;
                 end
               end
             end
//
            SYNCHRONIZATION_State: begin                 // S7
               FSM_OUT_Channel_State <= #`DLY ROUTING_ARBITRATION_1_State;
             end
          endcase
    end
endmodule
```

VII Experimental results

The experimental results of the study were obtained through simulations using the PlanAhead software product from Xilinx. The simulations examine and visualize, via timing diagrams, packet transmission from the output channel of the current router to the input channel of a neighboring router with a clock period of INT_CLK=5ns and DLY=0.5ns. The input signals to the finite state machine (excluding RST, INT_CLK and SYNC) are grouped into the Out_Vector vector for greater clarity. They are defined in the program using the *input* directive, while the output signals of the state machine are defined using *output*. The combinations within each vector and the sequence of vectors examine the behavior of the state machine: states, generated output signals, and transitions from one state to another depending on the input effects.

Fig. 7 shows the beginning of a packet transmission. At 160 ns the allocation and arbitration phase of the input queues with packets that are candidates for the current output channel begins. TRANSFER is activated at 167 ns, and at 170 ns the output channel state machine transitions to the *Path_Setting* state, establishes the packet path through the switch multiplexers (WR_MUX_ADR=1), reinitializes the RRA priority pointer (WR_RRA_PTR=1), and enables modification of the packet's header flit (MOD_DT=1). The readiness of the input channel of the neighboring router, declared by the condition (PACK_WAIT=1) ∩ (CHAN_BUSY=0), switches the state machine to the Transfer_Begin state at 175 ns, and at 180

ns it unconditionally transitions to *Transfer_1*. Actual transmission begins in *Transfer_Begin* with the activation of VALID_DATA and the writing of the first flit of the packet into the output channel register on the rising edge of WR_RG_OUT. The arbitration and switching time is two clock cycles relative to the SYNC signal.

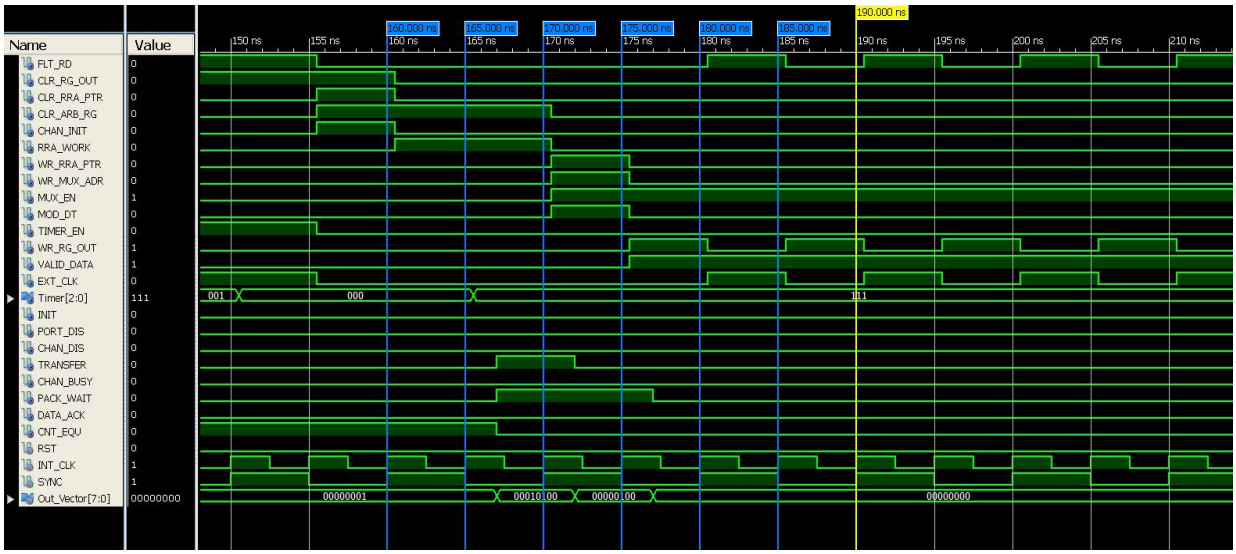

Fig. 7. Beginning of a transmission from the output channel to the input buffer of a neighboring router

Fig. 8 shows the end of a successful packet transfer. At 252 ns, CNT_EQU=1 is activated, which at 255 ns triggers the transition from state *Transfer_1→Transfer_End*. VALID_DATA=0 indicates the end of the packet, and the Timer is started (TIMER_EN=1) as a backup exit via TimeOut. At 257 ns the acknowledgment signal DATA_ACK for a successfully received packet from the neighboring router is activated, and at 260 ns, under the condition (CHAN_DIS=0) ∩ (PORT_DIS=0) ∩ (INIT=0) ∩ (SYNC=1), the finite state machine transitions to the *Routing_and_Arbitration_1* state for the next allocation. The time for completing the transmission and releasing the resources is two clock cycles relative to SYNC.

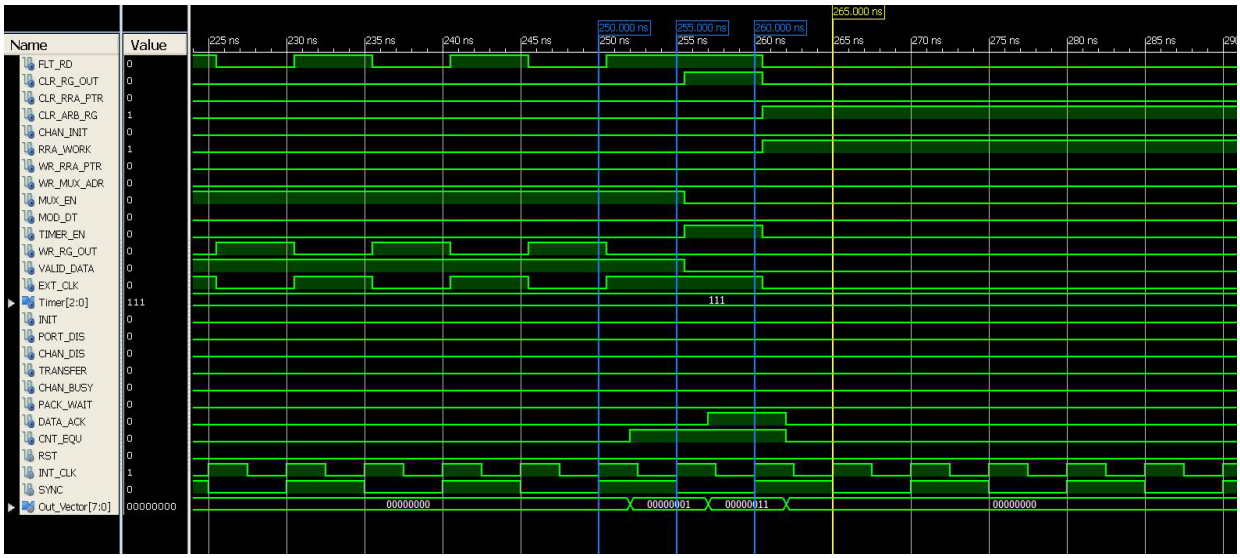

Fig. 8. End of transmission from the output channel to the input buffer of a neighboring router

Fig. 9 shows the complete transfer of the packet whose beginning and end of exchange are shown in Fig. 7 and Fig. 8 respectively. The packet has a length of eight flits and its transmission takes place under normal conditions for this purpose (free buffer in the neighboring router and successful arbitration for one clock cycle). The timing diagram shows the minimum latency of this packet in the current router. In this specific case it is two clock cycles (in the interval between 155 ns and 175 ns, given that the internal clock signal INT_CLK has a period of 5 ns) and includes the following actions: 1). Writing the header flit of a packet into a selected queue of the input channel; 2). Routing and arbitration (allocation of the packet to an output channel); 3). Formation of a path for the packet through the switch to the output channel.

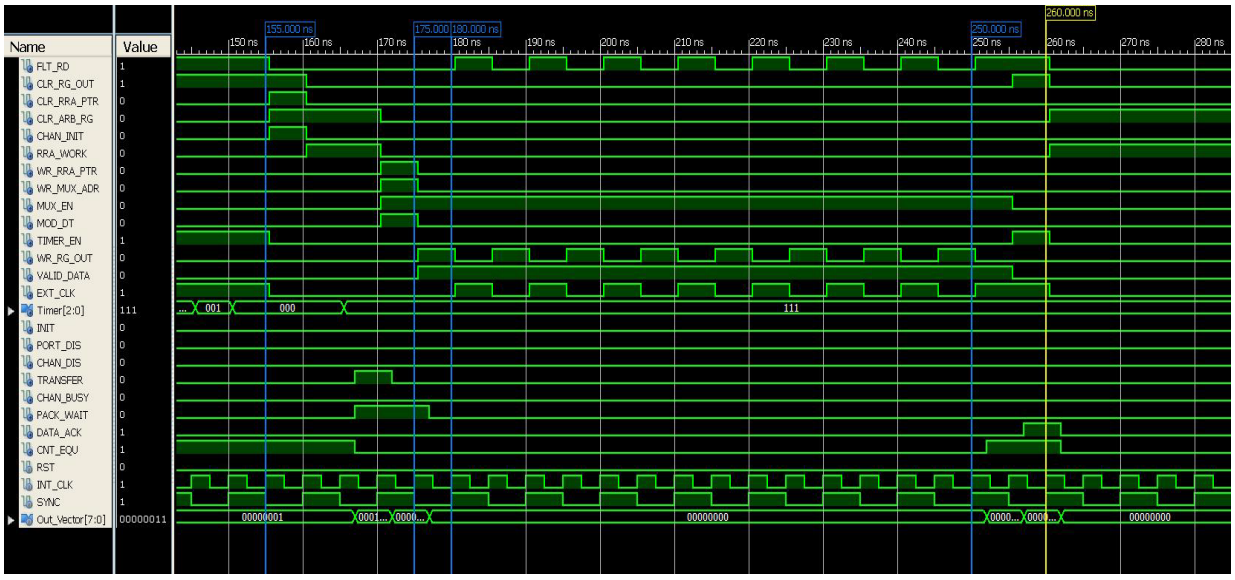

Fig. 9. Transfer of an eight-flit packet from the current router to the neighboring router

VIII Conclusion

In this paper, the input channel, the request allocator, the output channel control, and the RRA of a router for a DLH network with VCT flow control are presented. The experimental results from the studies conducted through simulations prove both the correct operation of the presented functional units in their interconnection and the achievement of minimum latency during packet transfer as well as the high degree of utilization of the interconnection channels. The timing diagrams visualize the fast switching when acquiring and releasing resources during packet transfer between two neighboring routers. Latency was measured in the number of clock cycles used for control (acquisition and release) of input and output channels at the beginning and end of packet transfer. The obtained results of two SYNC clock cycles for switching prove the minimum latency during packet transfer and the high degree of channel utilization in the router. It should be emphasized that latency does not depend on packet length and, as a direct consequence – the longer the packets transmitted over a link, the higher its usability.

REFERENCES

[1]. James Aweya, Switch/Router Architectures Systems with Crossbar Switch Fabrics, Published June 25, 2024 by CRC Press, ISBN 9781032654218

[2]. M. Rezaei-Zare, M. Fathy, A. Rezaei-Zare, Design of a high-performance router with distributed shared-buffer for load balancing for on-chip networks, Microelectronics Journal, Volume 132, February 2023, https:// doi.org/10.1016/j.mejo. 2022.105647

[3]. Dejun Shi, Xiaohu Han, Weijian Chen, et al. Overview of router architecture in high performance computing, Proc. SPIE 12717, 3rd International Conference on Artificial Intelligence, Automation, and High-Performance Computing (AIAHPC 2023), 127171R (21 Jul 2023); https://doi.org/10.1117/12.2686172

[4]. M. Besta, J. Domke et al., High-Performance Routing with Multipathing and Path Diversity in Ethernet and HPC Networks, IEEE Transactions on Parallel and Distributed Systems, Vol. 32, Issue: 4, 01 April 2021, Page(s): 943–959, DOI: 10.1109/ TPDS.2020.3035761

[5]. Ruoyao Xiao, Yiming Cui, Yuanyong Liang. Efficient Routing Algorithms for HPC Interconnect Networks, 2024, ⟨hal-04595791⟩

[6]. J. Jiao, Y. He, T. Cao, M. Kondo, Enabling circuit-switching in modern on-chip networks, Microprocessors and Microsystems, Vol. 95, November 2022, https://doi.org /10.1016/j.micpro.2022.104712

[7]. Monika Katta, T. K. Ramesh, Juha Plosila, SB-Router: A Swapped Buffer Activated Low Latency Network-on-Chip Router, September 2021, IEEE Access PP(99):1-1, DOI:10.1109/ACCESS. 2021. 3111294

[8]. Ping-Jing Lu, Ming-Che Lai, Jun-Sheng Chang, A Survey of High-Performance Interconnection Networks in High-Performance Computer Systems, Electronics 2022, 11, 1369, https://doi.org/10.3390/electronics11091369

[9]. N. L.Venkataraman et al, Design of matrix, distributive round robin, ping pong and enhanced ping lock arbiter for shared resources systems, Indonesian Journal of

Electrical Engineering and Computer Science Vol. 32, No.3, December 2023, ISSN: 2502-4752, DOI: 10.11591/ijeecs.v32.i3.pp1337-1345
[10]. Himashinii M, An Efficient Round-Robin Arbiter, IJSART - Volume 8, Issue 12, December 2022, ISSN [ONLINE]: 2395-1052
[11]. S. Salunke1, R. Kinagi, Design and Performance Evaluation of a High-Speed VLSI Router Using Efficient Buffering Techniques, IJRASET, Volume 14 Issue Jan 2026, ISSN: 2321-9653, ww.ijraset.com