Development of a Parallel Graphics Accelerator

- Router Design, Kernel System, and Monitor Program -

Tran So Cong

Table of Contents

1 Introduction 1

1.1 Research Background 1
1.2 Research Objectives 3

2 Message Passing Mechanism 6

2.1 The VoViAc System Overview 6
2.2 The Wormhole Routing 7
2.3 Pipelining and Latency Analysis 8
2.4 Flow Control and the Modified E-Cube Routing Algorithm 10
2.5 Broadcasting and Multicasting 11
2.6 Message Format and Packet Format 12

3 Router Design 15

3.1 Interconnection Network in the VoViAc 15
3.2 Router Structure 16
3.3 The Crossbar Switch 17
3.4 Input Buffer and Output Buffer 18
3.5 Crossbar Switch Controller 19
3.6 DSP Interface 20
3.7 Host Serial Communication 20

4 Kernel System 21

4.1 The Kernel Structure 21
4.2 Message Transmittal Mechanism 22
4.3 Node Setup Procedure 25
4.4 Router Device Driver (RDD) 26
4.5 Host Communication Device Driver (HDD) 27
4.6 Application Message Interface (AMI) 29
4.7 Memory Management 30

5 Monitor Programs, Command Interpreter, and Testing 32

5.1 Load and Run Scheme of Application Programs 32
5.2 Monitor Commands 33
5.3 A Test Application Analysis 35

6 Conclusions 37

Bibliography 38

List of Figures

Figure 2-1 System architecture of the VoViAc 6
Figure 2-2 Processing element structure 7
Figure 2-3 Distributed flits in flit buffer of successive routers 7
Figure 2-4 Asynchronous pipelining of flits between routers 8
Figure 2-5 Time-space diagram of a wormhole packet 9
Figure 2-6 Blocking flow control 10
Figure 2-7 Broadcast tree expansion 11
Figure 2-8 Message format 12
Figure 2-9 Packet format 14
Figure 3-1 VoViAc's interconnection network 15
Figure 3-2 Router structure 16
Figure 3-3 Crossbar switch 17
Figure 3-4 Input buffer state transition 18
Figure 3-5 Crossbar switch controller 19
Figure 4-1 Kernel structure 21
Figure 4-2 Message transmittal flow 22
Figure 4-3 The kernel's message processing layer 24
Figure 4-4 Node setup procedure flowchart 25
Figure 4-5 RDD flowchart 26
Figure 4-6 HDD flowchart 27
Figure 4-7 Memory page images 30
Figure 5-1 Modes of Monitor Programs 32
Figure 5-2 Test program flowchart 35

List of Table

Table 1 Monitor Commands 34

1 Introduction

1.1 Research Background

Swift advances in hardware, in particular faster, larger, and cheaper memories, have been transforming revolutionary approaches in computer graphics into reality. One typical example is currently shaping up in the field of volume graphics. This trend is rooted in the extensive research and development effort in scientific visualization in general and in volume visualization in particular [1].

Volume visualization is a method of extracting meaningful information from volumetric datasets through the use of interactive graphics and imaging, and is concerned with the representation, manipulation, and rendering of volumetric datasets [2]. Its objective is to provide mechanisms for seeing inside volumetric datasets and for probing into voluminous and complex structures and dynamics.

The source of volume data is sampled data of real objects or phenomena, computed data produced by a computer simulation, or modeled data generated from a geometric model. Examples of applications generating sampled data are medical imaging (e.g., computed tomography, magnetic resonance imaging, ultrasonography), biology (e.g., confocal microscopy), geo-science (e.g., seismic measurements), industry (i.e., industrial CT inspection), and molecular systems (e.g., electron density maps) [2]. Some examples of applications generating computed datasets by typically running a simulation on a supercomputer are meteorology (e.g., storm prediction), computational fluid dynamics (e.g., water flow), and computational chemistry (e.g., new materials).

Since the computational complexity for volume visualization is very high to be implemented on regular computers and workstation, other solutions must be considered. Although some algorithms can be implemented on general-purpose graphic systems, too many limitations on the algorithms are necessary to obtain high speed (especially in interactive

volume visualization). That is the reason why special purpose hardware is considered and the VoViAc (Volume Visualization Accelerator) [3,4] system is being developed.

A number of previous special purpose hardware approaches can be found in the literature such as Pixel-planes 5, the PixelFlow, the CUBE-3/4 architectures, VIRIM, the Verve architecture, and the PARCUM. The CUBE architecture was the first attempt to investigate the design issues of an interactive visualization architecture [5]. The key issue of this architecture, the skewed memory organization, is still in use in the CUBE-4 [6] architecture. The idea behind the skewed memory organization is the ability to read a beam of voxels in parallel. The CUBE-4 switches from ray parallelism (working on a ray in parallel in the rendering engine) to beam parallelism (working on a beam in parallel). The Pixel-Planes 5 [7] (PP5) architecture was developed at the University of North Carolina at Chapel Hill. The design is divided into the Graphic Processors (GPs), the Renderers, the Ring Network, the Frame Buffer (FB) and the Host Interface. The system provides both MIMD and SIMD parallelism. MIMD is provided by the Graphic Processors, each of which contains independent code and data stores. SIMD parallelism is provided by the renderer boards, each of which executes a single instruction stream in parallel for 128x128 one bit processing elements. Other graphic system being developed at the University of North Carolina at Chapel Hill is PixelFlow [8]. PixelFlow uses the technique of image composition: it distributes the rendering task over an array of identical renderers. Each renderer computes a full-screen image of a fraction of the primitives. The Verve (Voxel Engine for Real-time Visualization and Examination) architecture, currently called VOGUE, is the design from the University of Tubbingen in Germany [9]. The architecture is based upon the ray casting algorithm. Trilinear interpolation techniques are performed to estimate the voxel value at sample positions as well as the gradient at these positions. At the University of Mannheim, VIRIM, a massively parallel processing system for real-time volume visualization, is being assembled [10]. The hardware of VIRIM consists of several modules, each composed of a geometric unit for volume rotation, re-sampling, and gradient computation and a ray-casting unit for the final image generation. The GODPA system and the Voxel Processor, introduced in [11], have a basic assumption that in order to achieve significant speedup of a voxel based display algorithm, some form of parallelism has to be employed. The major difference between GODPA and the Voxel Processor is that GODPA is capable of displaying many independently controllable objects, whereas the Voxel Processor provides

only limited multiple object capabilities. The PARCUM (Processor Architecture based on a Cubic Memory) was first introduced by [12]. Its central feature is the cubic organization of its large voxel memory (512^3 binary voxels or 256^3 grey value voxels). Voxels are grouped in sets of 64 (4^3) so called macro volumes elements (MVEs). The memory is divided in 64 modules in such a way that the individual voxels in a MVE each lie in a different module. This allows the system to read or write 64 voxels from/to memory in parallel. Besides the above architectures, there are other architectures mentioned in the literature, like Farell's Colored-Range Method, the Pixel-Planes 4, the Insight system, 3DP4, the Kubota Denali, Hewlett Packard's Teramac, the Princeton Engine, and a PC-based 3D Imaging systems of the MIPG group at the University of Pennsylvania [13]. These will not be discussed, because they are not sufficient for performing high quality volume visualization or so complicated for the purpose of special purpose hardware for volume visualization.

Recently, performance of PCs and workstations is much more powered, but for volume graphics visualization, especially in interactive real-time application, these powers are still not sufficient. Research on volume graphics acceleration still attracts many researchers to invest their work to find out other better ways for volume visualization, both in speed and cost-performance factor. Our project, the VoViAc system, was designed in 1995 for this purpose. The VoViAc architecture bases on the 3D-torus topology. With the 3D torus topology, we should have the best fit for volume dataset division as the volume dataset is divided into smaller data for each processor in interleave fashion and then concurrently processed by processors. The VoViAc connects with host computer to receive volume dataset, processes it with ray casting or 3DDDA-algorithm and output image to display's raster memory.

A processing element (PE) is a set of a Processor, Local memory and a Router. For high speed in mathematical processing, DSP (digital signal processor) is used. PEs communicates with each others by Router over interconnection network. The router is designed to implement on FPGA (Field Programmable Gate Array). With FPGA, we can reduce development time and, if possible, can use the programmability of FPGA to add a sub-functional unit beside the processor for intensive calculations, so that increasing speed.

1.2 Research Objectives

The purpose of the VoViAc project is based on three key issues:

- Design a simple and compact architecture for volume graphics visualization,
- Develop of an executable parallel system, both in hardware and software,
- Research the applicability of FPGA.

The VoViAc is completed the design in architecture and electronic circuit around 1996 [3, 4]. We have been attempted to investigate on the development of the VoViAc to make the system executable and also consider usability of FPGA. Hence, the development of the VoViAc consists of the following works:

a) To design the Router: The VoViAc is a distributed memory parallel system. Thus, each processing element (PE) uses messages to communicate with the others in the system. To route message over the interconnection network, we create a device called Router in each PE. When a thread sends a message, the kernel packetizes it into packet(s) and then sends to the Router. The Router then automatically routes packets to destination.

b) To develop a kernel system: The VoViAc kernel is like an operating system for each node except it does not have a file system. During operations, the kernel communicates with a monitor program in host so that forms a pseudo-OS for the VoViAc. The kernel has to supply most basic device drivers, interrupts and functions for applications running on it such as message send and receive, memory management, synchronization, etc. At power up, the kernel is the software that the VoViAc runs to check itself and to communicate with the host computer thus a part of the kernel must be contained in ROM devices. With the small size of system and memory, the kernel must be as small and fast as possible.

c) To implement the Monitor program and Command interpreter that would allow users to program applications, load and run these programs on the VoViAc, and debug these applications. At startup, the Monitor program lets users load the Command interpreter to the VoViAc. After that, other operations will be enabled, such as load and run, debug, etc. The debug function allows users to investigate a running program.

The main purpose of the development process is to make an environment in which users can program, test and run application on the VoViAc system. This process needs a lot of effort since the VoViAc was developed from bare circuits with no previous tools or utility programs. The entire kernel was written in assembly language to optimize speed. The router was also designed in schematic to achieve the smallest size of gates to be fit in

the FPGA. The debug work for the kernel has been done using an electronic logic analyzer and other hardware instruments.

2 Message Passing Mechanism

2.1 The VoViAc System Overview

The architecture is shown in Figure 2-1 with 8 PEs (recent configuration). The PEs connect with each other's through an interconnection network in 3D-Torus topology. The Host and the VoViAc are coupled by a peripheral connection, which allows host communicates with the VoViAc. Processed data could be sent back to host or redirected to a 2D-raster memory of video hardware and then the processed images are projected on display. The 3D-torus connection has an advantage that the system is scalable. So that, we can extend the VoViAc with the smallest cost that we almost do not need to change the hardware and software design. The interconnection network is 8 bit wide, operates at 20 MHz. Thus, we should achieve maximum 160 MBPS throughput in a connection. In each PE (Figure 2-2 shows the PE structure), we have a microprocessor, which is a DSP for high speed in mathematical operations, local memory, and a Router. A bus system wires all devices to-

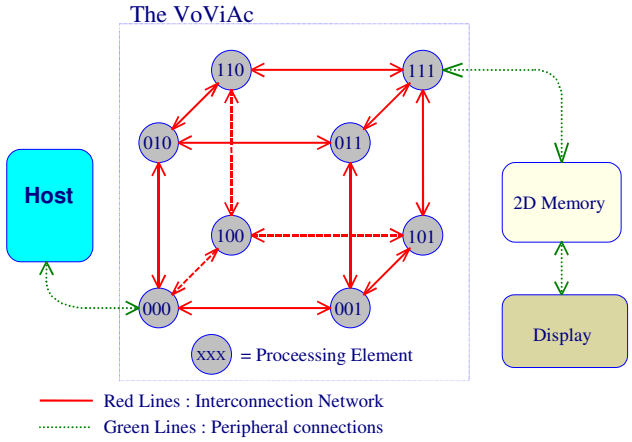


Figure 2-1 System architecture of the VoViAc

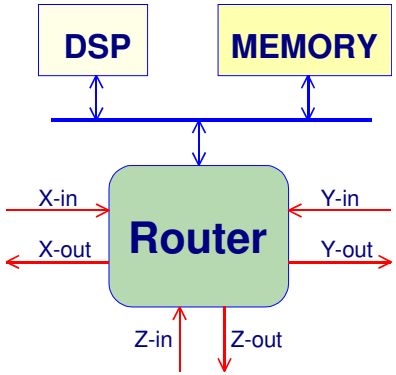


Figure 2-2 Processing element structure

gether. The DSP is Texas Instrument TMS320C31 operates at 40 MHz. Local memory capacity is 512 Kbytes. The Router is implemented on Xilinx FPGA XC4010D, which can host about 10,000 logic gates.

From the main purpose is to develop the VoViAc, the main task is to implement message passing mechanism and design the Router. The message passing mechanism set standards and specification for the design of the Router. It also affects the implementation of the Kernel system. In latter sections of this chapter, we will discuss the Message Passing Mechanism of the VoViAc.

2.2 The Wormhole Routing

There are various types of switching technique using in interconnection network systems such as the circuit switching, packet switching, virtual Cut-through, wormhole, mad post-man switching, etc [17]. We chose the wormhole switching to be implemented in the VoViAc because of the wormhole technique do not require to buffer complete packets

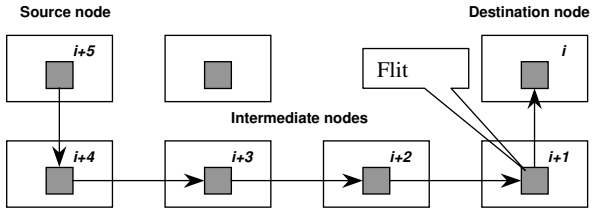


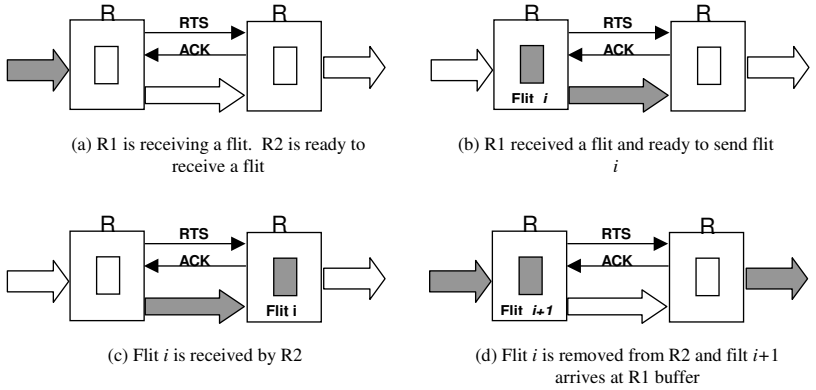
Figure 2-3 Distributed flits in flit buffer of successive routers

within router. Thus, we can construct a small, compact and fast router. Also, the wormhole routing offers the best latency over other types of routing technique.

In wormhole routing, message packets are pipelining through the network. A message packet is broken up into flits. A flit is the unit of message flow control. In the VoViAc's router, a flit is a byte. Several first flits contain routing information and form packet header; the others contain packet data. As a result, each incoming data flit of a packet is just simply forwarded along the same channel as the preceding flits. When a packet reaches a router, its header is examined immediately and then the router forwards the header to the output buffer as soon as the routing decision is made without waiting for the whole packet to be received into the router. At the same time, the channel from input buffer to output buffer inside the router is occupied until the whole packet is passed though the router. The same manner happens when the packet reaches the adjacent router in the interconnection network. Because the packet size is larger than router buffer size, a packet occupies buffer in several routers. For example, in Figure 2-3, when the i flit reaches the destination node, the i+1, i+2, i+3, i+4 flits is in intermediate nodes and the i+5 is in the source node. In the next cycle, the i+1 flit will reach the destination and the i+6 flit will be ready to be sent to the adjacent intermediate node in the transmittal pathway. It is why this routing scheme is called *wormhole* routing.

2.3 Pipelining and Latency Analysis

In physical channel, the flit can be constructed in terms of phits. A phit is the umber



(a) R1 is receiving a flit. R2 is ready to receive a flit

(b) R1 received a flit and ready to send flit *i*

(c) Flit *i* is received by R2

(d) Flit *i* is removed from R2 and filt *i*+1 arrives at R1 buffer

Figure 2-4 Asynchronous pipelining of flits between routers

of bit that can pass through physical channel at a time. For example, if the physical channel is 4-bit width, a phit is also 4-bit width. A flit can be one or more phits. In the VoViAc, a flit is also a phit and equal to a byte. Figure II-4 presents how a flit (also a phit) would be transferred (pipelined) between Router R1 and Router R2 in one direction. At first, flit *i* arrives at R1 out-buffer (a) and the router R1 assert the RTS (Request To Send) signal when flit *i* is to be transfer (b). When the flit *i* is received by R2, signal ACK (Acknowledged) is asserted. This leads to de-asserting the RTS by R1, which cause R2 to de-assert ACK (c) and the transmittal of the flit *i* is completed. In the next cycle, flit *i*+1 reaches the R1 and flit *i* is to be sent out from R2 to adjacent router in the pathway.

For latency analysis, the time-diagram of a wormhole routing packet is shown in Figure II-5. The clear rectangles illustrate the propagation t_w of flits across physical channel. The shaded rectangles illustrate the propagation of header flits across the physical channel. The router can make a routing decision in t_r second. Let t_s denotes the interrouter delay. The number of link between the source and the destination is assumed to be D links apart. Let L be the length of the message packet in bits, and W is the physical channel width.

The base latency of a wormhole routing packet can be computed as follows:

$$t_{wormhole} = D(t_r + t_s + t_w) + \max(t_s, t_w)\left\lceil \frac{L}{W} \right\rceil \qquad (2.1)$$

As the message packet becomes longer, the latency of wormhole routing becomes almost independent with the number of links D along the transmittal path. In contrary, in packet switching or circuit-switch routing, the latency is corresponding with the number of links D.

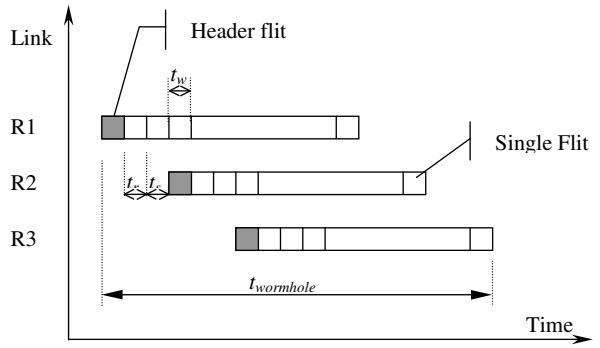


Figure 2-5 Time-space diagram of a wormhole packet

2.4 Flow Control and the Modified E-Cube Routing Algorithm

Flow control policy is how to control network traffic flow without causing congestion or deadlock. The wormhole routing uses a blocking policy incases of packet collision as illustrated in Figure 2.6. When packet 1 occupied the output channel, the second packet is being blocked from advancing. When the output channel is freed from packet 1, the packet 2 is then locating the channel and advancing. The blocking flow control is easy to be implemented but this may cause deadlock if we do not a suitable routing algorithm.

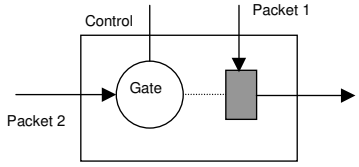


Figure 2-6 Blocking flow control

In some router, the message priority is also important. When two packets require the same a channel at the same time, a packet with higher priority will allocate the channel. Another packet will be blocked until the previous packet completely passes through the channel. Thus, a message with higher priority will be routed faster incase of high traffic appears in the interconnection network. This will support for multi-level priority task such as a kernel request communication for user interrupts will be more important than a normal application communication.

Routing algorithm determines communication path to guide a packet reaching destination over the interconnection network. The packet routing can be conducted out deterministically or adaptively. In deterministically routing, the complete communication path is computed out from source and destination addresses. In other words, the communication path is unique for a pair of source and destination. In contrary, adaptive routing depends on network conditions so that the communication path may be changed. In the VoViAc's Router, the deterministic routing is used to construct a compact and fast router. The modified E-Cube routing algorithm is an alternative of E-Cube routing algorithm [16], which is based on dimension-ordering routing. The modified E-Cube routing is used for 3D-mesh, 3D-torus topologies. This algorithm are described as follows:

The next node address $a\{x_a, y_a, z_a\}$ of a 3D-torus is calculated from present node address $p\{x_p, y_p, z_p\}$ and destination node address $d\{x_d, y_d, z_d\}$ as follows:

$$a\{x_a, y_a, z_a\} = p\{x_p+i, y_p+j, z_p+k\}$$

With

$i = 1$ if $(x_p \oplus x_d \neq 0)$

$j = 1$ if $(y_p \oplus y_d \neq 0$ and $j = 0)$

$k = 1$ if $(z_p \oplus z_d \neq 0$ and $j = 0$ and $k = 0)$

i, j, k = (0,1) and a, p, d are binary coded addresses

In this algorithm, a packet always routes in X dimension first, then Y dimension, and lastly Z dimension. When the packet reaches the same level of destination address in a dimension, it detours to the next dimension until it reaches the destination. Thus, a deadlock or live-lock situation will not occur.

However, this dimension-ordering routing do not offer minimal routes for the torus, but somehow, a deadlock-free routing that let packets traverse through longer path sometimes reduces some time can reduce network traffic or for other reasons.

2.5 Broadcasting and Multicasting

A broadcast pattern is the case of one-to-all communication while the multicast pattern corresponds to one-to-many communication. So, in some respect we can consider broadcast as a special case of multicast communication. Sending one message to each of the destinations can carry out by multicast communications. This method performs poorly because it requires a large amount of network resources and also it involves several software steps to prepare for messages. To further reduce the communication latency, the multicast operations need to be supported by hardware. The multiple destinations are encoded and appended to the message's packet header. Such multicast message is called multi-destination message. To perform better traffic and distance cost of multicast, the tree-base algorithm is used in the VoViAc system. The destination addresses are encoded in the packet header. When a multi-destination packet reached a destination, that destination is removed from packet header. Packet routing follows the dimension order algorithm described in the above section. For example, a broadcasting form node 0 in an 8 PEs system will have the following tree expansion (Figure 2-7):

At the node 0, the destination node 4 has the same level in X and Y dimension so a packet is forwarded to node 4 with only one destination address in packet header. The node 2 and node 6 have different Y dimension, so that, the packet is forwarded to node 2 with destinations are 2, and 6. After reaching node 2, the node 2 is removed from destination and then forwarded to the node 6. The same thing happens with node 1,3,5, and 7. With this scheme, we can reduce the traffic from 12 (1+1+1+2+2+2+3) to 7 while the maximal distance still remains at 3 and the start-up cost (for processing packet at sending time) is the same with a unicast.

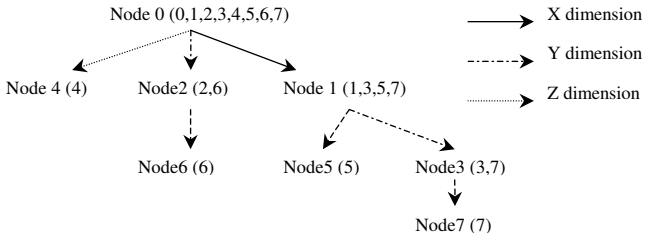


Figure 2-7 Broadcast tree expansion

When an application do a multicast communication, it places multiple destination to packet header and simply send the packet to router. The router then does the multicast automatically to destinations.

2.6 Message Format and Packet Format

Based on the message passing mechanism described above, we established the message format and packet format. A VoViAc's message will have unlimited (theoretically)

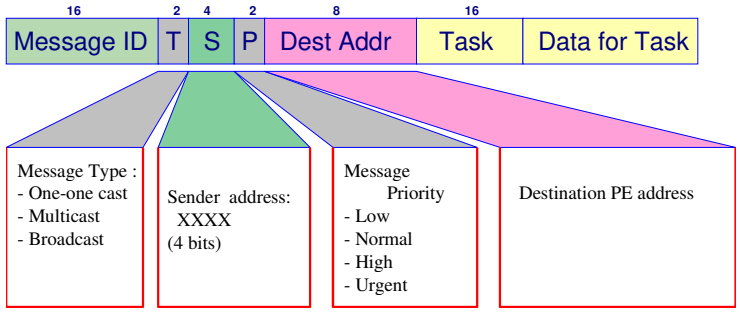


Figure 2-8 Message format

length. To be able to design and implement the router, we need to partitionize messages into one or more packets. The packet then will be assembled back to original message at the destination node. We can realize that messages are to be used with the kernel level while router (physical) level handles packets.

A message has the following format as illustrated in the Figure 2-8:

- Message ID is a 16-bit field used to identify the message.
- T field is the message's type that used to indicate the routing type of massage. Types are unicast (permutation), multicast (one to many) and broadcast (one to all).
- S filed is sender (source) address.
- Destination address field contains destinations: The VoViAc has 8 nodes, so herein we use 8 bit to encode destinations. The bit position is corresponding with binary address of destination. For example, bit 0 is corresponding with the node 000; bit 1 is node 001, and so on. The value of bits in this field indicates if the nodes are included in destination list or not. For example, with the value of 0001010, node 001 and 011 would be the destinations of the message.
- P field is the priority of the message. The purpose of this field is to make a right processing order when two or more messages reached destination at the same time, so that, message with higher priority would be first processed.
- Task field (T) holds the task which node must do in after receiving the message. In most cases, the task field also includes data length of the message.
- Data field carries data of the message.

With this message format, application can send and receive message to the other nodes easily, with unlimited length. The data type of data to be sent can be any sort of data such as array, integer, floating, etc.

As shown in Figure 2-9, each message is separated into parts, which then will be attached on packet to be routed by the router. Packets have fixed length at 31 byte. Several first byte to carry routing information as follows:

- In the first byte of package, we have T, S and P parameters, which was defined in massage structure section.
- The second byte is destination nodes; this field has the same structure in message's destination field.
- The third byte is always Package Serial Number (PSN). Package serial number is used to reassemble packages together to form original message in case a message required more than one package to transmit.
- The bytes from 3 to 30 are data (message data) of a package.

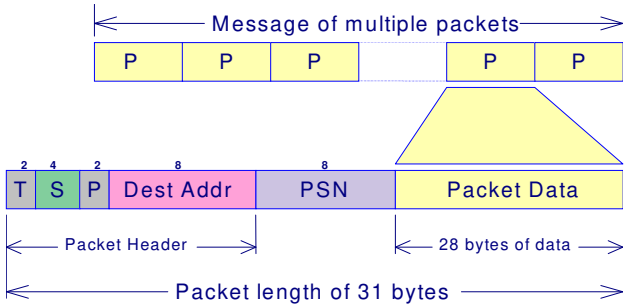


Figure 2-9 Packet format

3 Router Design

3.1 Interconnection Network in the VoViAc

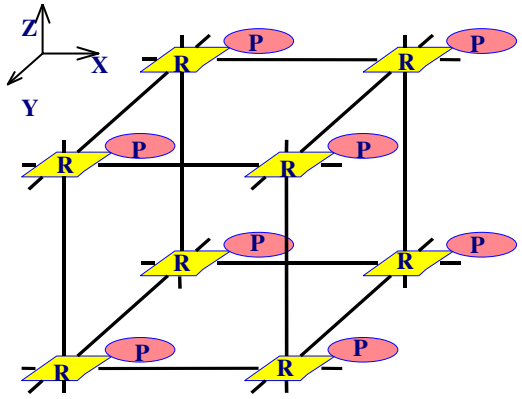


Figure 3-1 VoViAc's Interconnection network

The VoViAc's intercommunication network is a 3D torus as shown in Figure 3-1. PEs are connected to six neighbors, called north, south, east, west, up, and down. PEs on the edge of the mesh are connected to PEs on the opposite edge, creating the torus structure. The 3D-Torus is a scalable architecture.

Routers presence as valves at the junctions of connections to control the flow of packets in the interconnection network. In each dimension, the Router has an input port and an output port connecting to the previous and next node. A pair of injection and extraction channel conjoins router and local processor.

3.2 Router Structure

Figure 3-2 illustrates the router structure. The main part of this router is a Crossbar switch, which connects inputs to outputs performing packet routing. The Crossbar switch can tie up an input to ones or many an output at a time but can not connect multiple input to

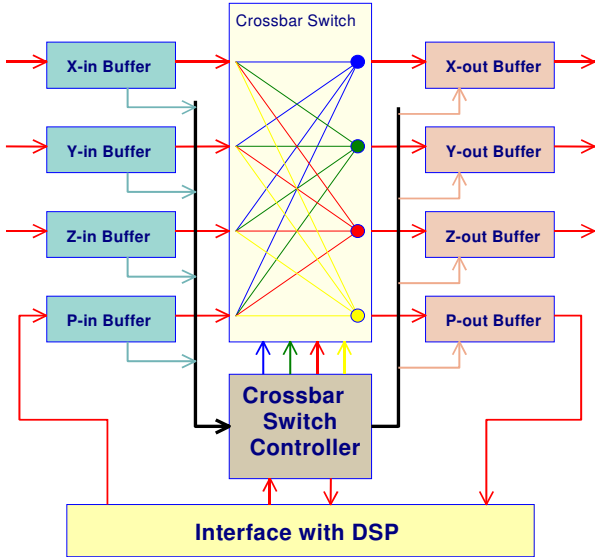


Figure 3-2 Router structure

an output channel. The one-one connecting performs unicast communication while the one-to-many connecting does multicast communication. The Input buffer controls the link with previous router and does the destination decoding. The Crossbar Switch Controller governs crossbar switch states; does scheduling for channel request; and manipulates the output buffer to change packet's header when needed. The P-input and P-output buffer is correspondingly injection and extraction channel for local processor. The Interface with DSP circuitry provides DSP control over the router.

When a packet reaches an input buffer, its header is firstly latch. Then, routing information, such as destination, priority and routing type are decoded to determine routing decision. After that, a crossbar channel request asks the Crossbar switch controller to provide channel. The Crossbar then queues up the request in a FIFO buffer. When the request is in its order to be solved, Crossbar switch controller manipulates the Crossbar switch to con-

nect input to output establishing the channel. After that, the packet passes through the router in pipeline fashion as described in section 2.3. If the packet is a multi-destination packet, header of output packet will be changed appropriately with output dimension as described in section 2.5. When the whole packet has passed over, the input buffer de-asserts its channel request; therefore, the crossbar switch controller orders the crossbar switch to disconnect the channel. The router finishes a routing cycle.

The Router can do many routing decisions concurrently. For example, maximal four unicast communications (X -> Y, Z -> X, Y-> P, P -> Z) can occur at the same time

3.3 The Crossbar Switch

In a typical crossbar switch, there will be a column of inputs on the left edge and a row of outputs on the bottom edge. The switch configures itself dynamically to connect an input to an output. As long as each input wants to communicate with a different output there will be no contention. If two or more inputs need to access the same output, one will be blocked until the switch reconfigures itself. A crossbar switch has a short diameter - information needs to pass through only one switching element on a path from one edge to another - but poor scalability. As the number of input and output increase, the complexity of one state crossbar growths rapidly. In this case, multistage switching network, which is compounded from many simple crossbar switches, wills be used to create uncomplicated network that is called bayan network. In VoViAc's Router, a simple 4 × 4 crossbar is used as shown in Figure 3-3.

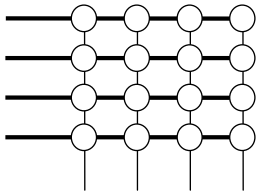


Figure 3-3 Crossbar switch

3.4 Input Buffer and Output Buffer

Input and output buffers manage the links between current node and adjacent nodes. The Input buffer contains packet header buffers, an address decoder, and a flow control circuit. Normally, the Input buffer waits for an incoming packet. When a packet reaches node, the input buffer latches its header. The input buffer shifts to Address Decode state. It may take several cycles for input buffer to decode the destination. Depend on the value of desti-

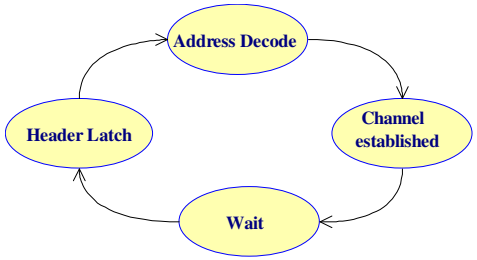


Figure 3-4 Input buffer state transition

nation field, a channel request will be forwarded to Crossbar switch controller along with packet information such as priority and routing type. The input buffer then waits for channel connected confirmation. When the channel has been established, the input buffer forwards its contents to output buffer while receives other data of the packet. After the entire packet has been passed through, the Input buffer returns to its Wait state and waits for another packet to be routed over. In multicast circumstance, multiple channel requests will be issued. In this case, the input buffer has to wait for all-channel approbation.

The physical propagation t_w for a flit to across the channel is 2 clock cycles. The maximal routing decision delay t_r is 12 cycles. The interrouter delay t_s is 6 cycles. So that, from equation 2.1, we have the latency to route a 31-byte packet from node 0 to node 7 is 246 cycle. With the clock frequency is 20 MHz, a packet needs approximately 12 microseconds to traverse along interconnection network.

The output buffer controls the links to next router in routing path. When a flit is received from input buffer, the flit is being sent to next buffer in the routing pathway as described in section 2.3. If the current node is in destination list, then an interrupt is issued and the packet is received into node's memory. Output buffer also gets the header modification information from the crossbar controller to change the header of packet in case of multicast as described in section 2.5.

All flit buffers in the input and output buffers operate in asynchronous pipelining fashion of wormhole routing. This fashion allows us to design router circuitry easily.

3.5 Crossbar Switch Controller

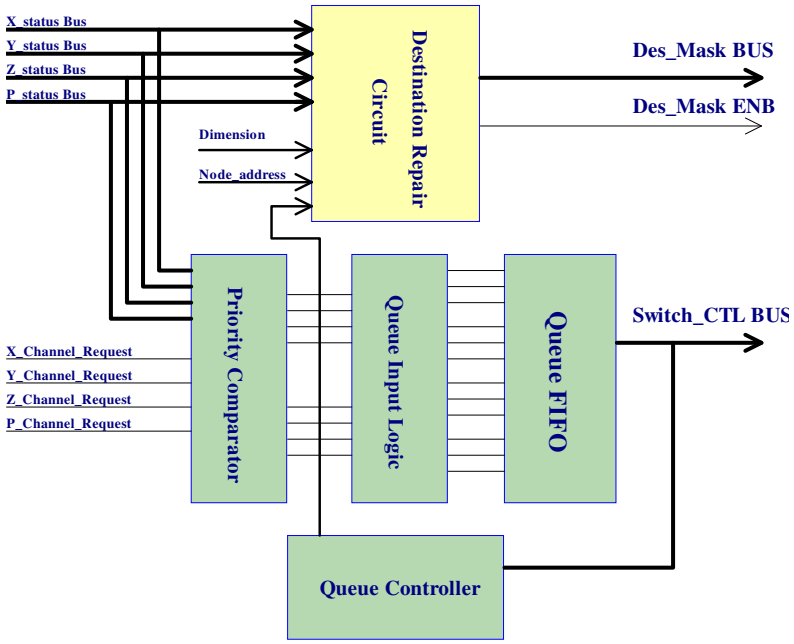


Figure 3-5 Crossbar switch controller

The Crossbar switch controller manages channel reservation (flow control) for the crossbar switch. The crossbar switch controller has 4 unique functional units for 4 output dimensions of the crossbar switch. Consequently, the router can manage 4 different channel requests at the same time. Each of functional units is illustrated in Figure 3-5. The Priority Comparator compares packet's priority when requests are asserted in to decide exactly what channel serving order. The requests will be stored in a FIFO waiting to be served. Depend on the being served request, the Destination Repair Circuit will try to repair the Destination field of the packet whenever necessary. At this time, the Des_Mask BUS contains the Des-

tination field mask for output buffer to delete unwanted destination in the destination field of the packet. Algorithm, which determines to be deleted destination, follows the multicast policy of message routing mechanism.

3.6 DSP Interface

The DSP interface circuitry provides the accessibility to the Router for DSP local processor. The interface has the following function:

- Provides the injection and extraction ports for the Router to read and write packet data to the router.
- Provides interrupt circuitry for the DSP. The interrupt scheme help the kernel software to detect when a packet is coming and/or when the Router buffer is ready to transfer data to.
- Supports router status monitoring. This function allows the kernel to control the operations of all functional units of the Router.
- Provides node's information for the router such as node address, kernel statuses.

All of these functions allow the Kernel cooperates with the Router smoothly.

3.7 Host Serial Communication

In the root node (node 0), an additional communication circuit is designed to help the VoViAc communicates with the Host computer. This circuitry is a normal serial communication circuit that follows the RS232-C standard. The serial communication bases on asynchronous frame of data. Each frame contains a start bit, a stop bit and 8 data bits. The host communication circuit can operate at 115200 bps. This circuit has been standardized; therefore, it won't be described hereby in details.

4 Kernel System

4.1 The Kernel Structure

The VoViAc kernel is like an operating system for each node except is does not have a file system. The kernel is the software that applications run on it [16]. (Figure 4-1) The kernel must be enough functionality to allow the VoViAc to communicate with the host computer. Also the kernel has to supply most basic device drivers and interrupts for appli-

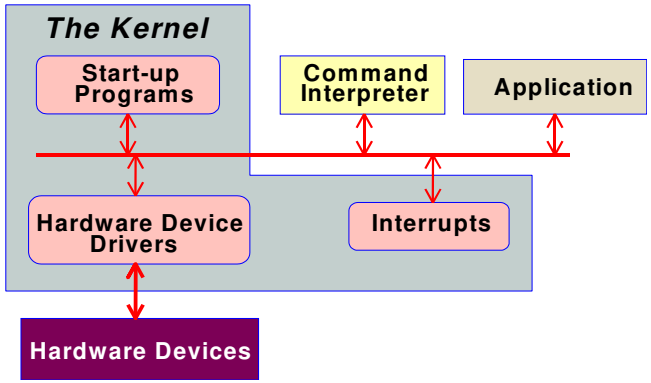


Figure 4-1 The Kernel stucture

cations running on the kernel. At start up, the kernel is the software, which the VoViAc run to check itself and to communicate with host computer. Therefore, a part of the kernel must be contained in EEPROM. The other part of the kernel will loaded from host computer.

The kernel is system software, which supports the following commissions:

- Setup the system hardware at power start-up.
- Communication program with host computer to load the other part of the kernel.
- Communication program with other nodes to check the overall system or to complete the kernel.

- Supply the most basic functions, interrupts and hardware device drivers of the system for applications that isolate the applications from system hardware and from system software environment.

4.2 Message Transmittal Mechanism

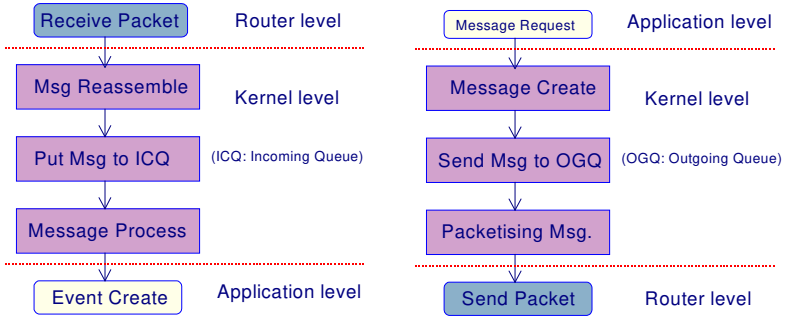
Figure 4-2 shows the basic of message transmittal mechanism.

In sending, an application sends a message request to the kernel. The kernel creates the message structure and sends the message to the bottom of outgoing queue. Meanwhile, the kernel packages the top message into packet(s) and orderly sends packets to the router. The router then sends packet(s) over the interconnection network to destination.

In receiving, the router receives packets and sends them to the kernel. The kernel reassembles the original message from packets with information in the header and the PSN and then sends the message to the bottom of incoming queue. Meanwhile, the kernel processes the top-level message and then creates an event to the application.

As in the Figure 4-2, The kernel has 3 levels of processing. The message layer processes the application request. The packet process layer packetizes messages into packet and reassembles packets back into message. The packet transmittal level moves packets from buffer to physical channel.

Figure 4-3 illustrates in detail how the kernel processes message. Hardware device drivers are Router Device Driver (RDD) and Host Communication device Driver (HDD). Hardware device driver should be programmed in interrupt driven method to achieve high speed and also to guarantee errorless data transmittal. But in the hardware implementation,



a) Message extraction b)Message injection

Figure 4-2 Message transmittal flow

due to the lack of interrupt request signal, these drivers have to share the same interrupt request signal. These affects directly to the software implementation due to the need of sharing interrupt. Due to the limit of memory, we have to limit message length to 1792 (256-packet size) word. The structure of message buffer also needs to be constructed carefully to get better performance.

In the hardware interface, transmit and receive procedures are stimulated by interrupts request; the Interrupt Distributor processes the request and then executes the corresponding procedure. These procedures get/put data (a packet data with router and a byte data with host) to devices when, respectively, from/or to local buffer. Because the above procedures are interrupt driven, these procedures must be as fast as possible to avoid data loss and to reduce overhead. After finish these procedure, the Interrupt distributor execute the Packet Processing procedure.

Packet Processing processes packets. It get packets from local receive buffer, sort them into time order and reassembles them back into message. In opposite direction, packet processing divides sending messages to packets and put them to local send buffer. Packet Processing also does filter job for packet to collect the packet that is intended to route to host or from host to other nodes. The packet filter then put these packets directly to router or host communication device without reassemble.

One important items in Packet Processing is the Receive Message Buffer (RMB) structure. The RMB can cause deadlock or greatly decrease the performance. Is the RMB have small size, another message must wait to be received while previous message is being processed or even deadlock can occur when the application wait for an incoming message but the buffer is full. It is easier to construct and to manage separated RMB which contains messages that came from the same node nut this method will dramatically reduces the size of each buffer due to the limit of memory space. So that only one big RMB is used to contains messages from all node. This method virtually increases the buffer size and allows more flexible buffer management. In the output direction, we can use the data buffer of application for sending message buffer. Application Message Interface (AMI) supplies function calls for application to send, receive, inquire messages, and to do barrier synchronization among nodes. When a message is to be sent, AMI call the Packet Processing procedure, then this procedure processes the message into packets and enable interrupt. The interrupt then stimulates hardware device drivers to transport packet to hardware [17].

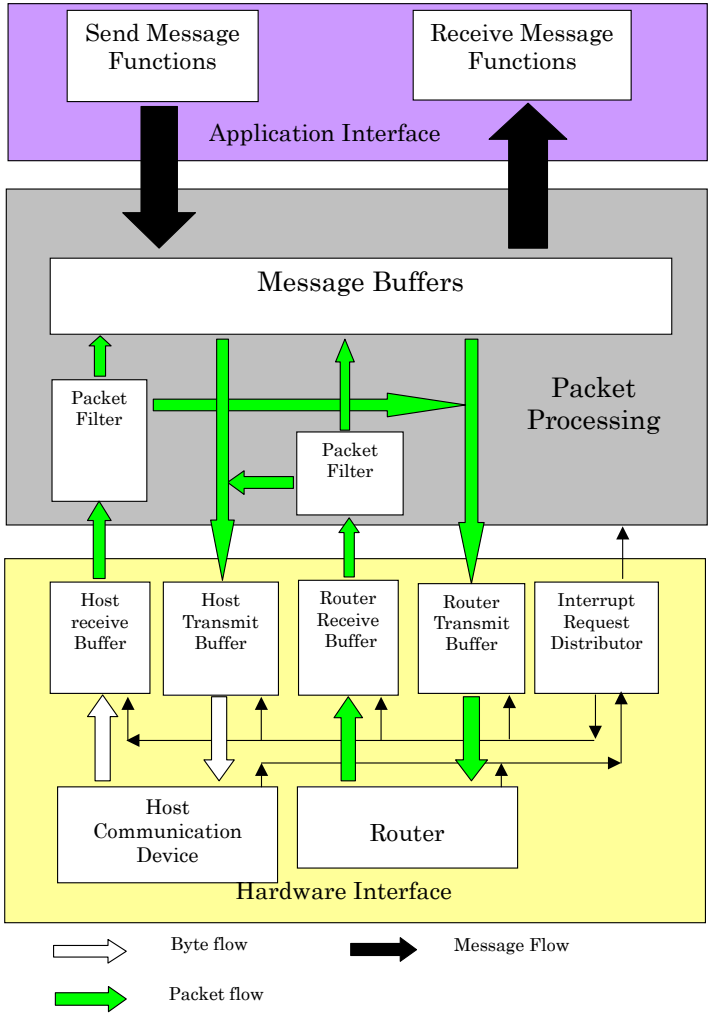


Figure 4-3 The kernel's message processing layer

4.3 Node Setup Procedure

The main purpose of the node setup procedure is to set-up processing element at power up. The flowchart of this program is shown in Figure 4-4. At first this procedure configures system hardware, programs the FPGA for Router functionality (and Host communication device incase of root node). For the checking device's functions, the program checks there devices (router, serial communication circuits) from inside point of view but not from outside so it just only checks registers reading, writing, devices initial status. The device's variables are about device's status and ability such as the device is available or not or others attributes of devices. There variables allow programmer to ignore about the hardware but still able to get there information when needed. After the setup procedure is finished, Command Interpreter is being loaded.

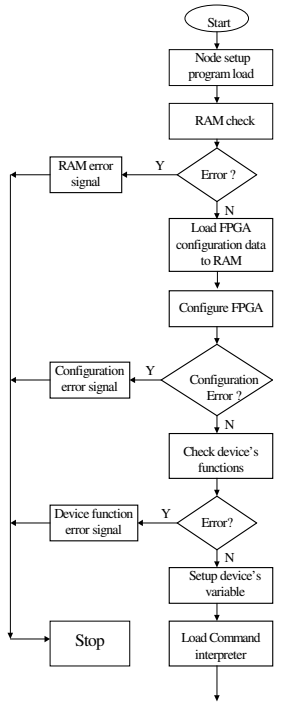
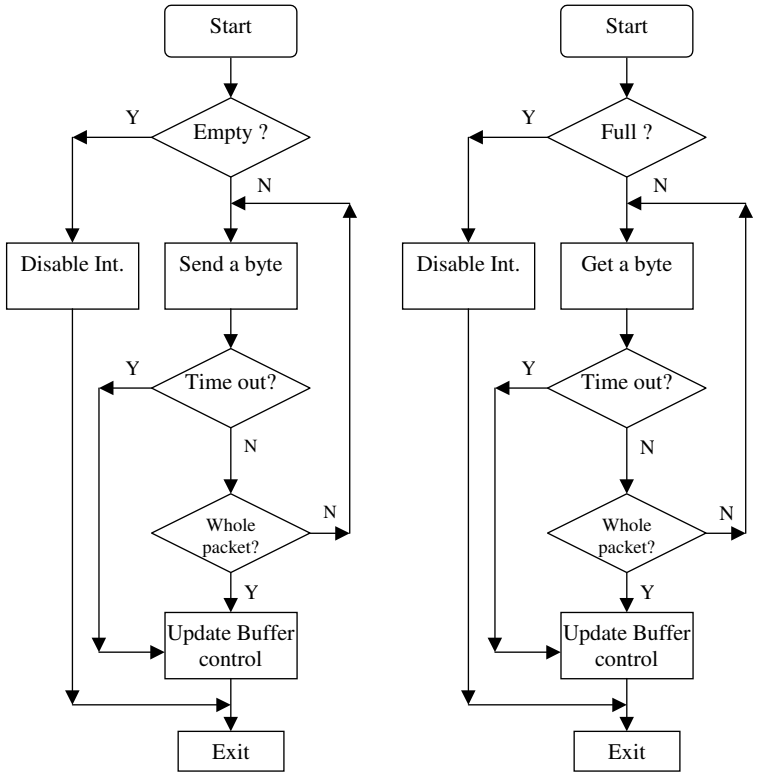


Figure 4-4 Node setup procedure flowchart

4.4 Router Device Driver (RDD)

The Router Device Driver carries packets of data between kernel's router packet buffer and the router. Because of the router has high speed connection, the software overhead must be reduce to achieve overall high-performance. So the RDD has been designed to receive data in packet mode. That means the router, once stimulated by interrupt, will wait in a loop to receive/send the whole packet instead of receive/send a single byte for each interrupt request. This mechanism reduces the interrupt processing cost dramatically.

The RDD gets the incoming packet in Router Receive Buffer and move the outgoing packet in Router Transmit Buffer to Router of destination and source node respectively.



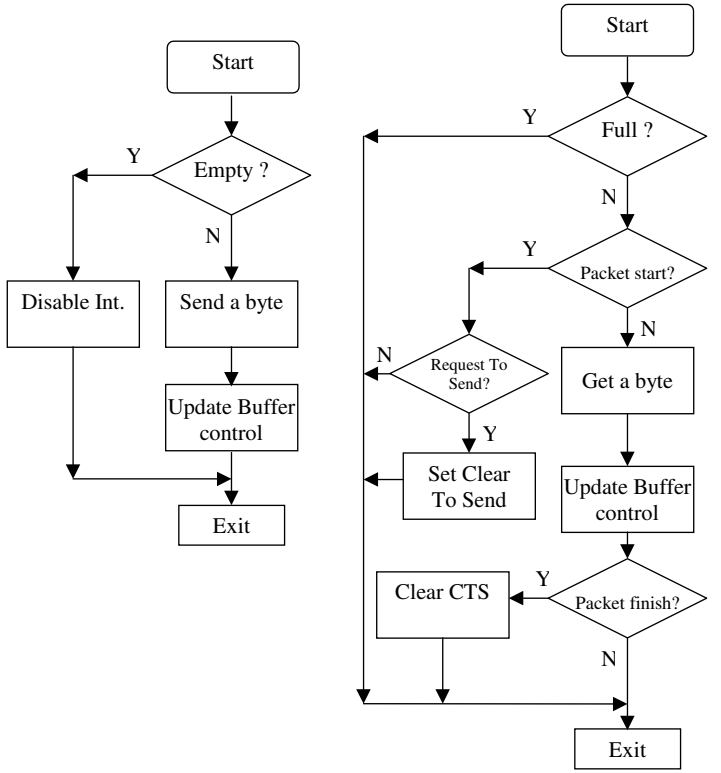
a) Packet sending to Router b) Packet receiving from Router

Figure 4-5 RDD flowchart

Each time a packet has been sent the RDD stimulates the Packet Processing Procedure to transport a new packet to Router transmit/receive buffer. If there is no other packet to be transmitted, the Router send interrupt will be disabled. Correspondingly, when a packet has received into Router Receive Buffer, the Packet Processing Procedure is also being stimulated to reassemble back the packet into the original message.

4.5 Host Communication Device Driver (HDD)

Host Communication Device Driver is used to carry packet data to the device that connects with Host computer. Because of the VoViAc and Host currently using the serial connection to communicate, the speed is slow. So that, in this device driver, we used the



a) Flowchart of a Byte sending to Host b) Flowchart of a Byte receiving from Host

Figure 4-6 HDD flowchart

byte transfer mode (instead of packet transfer mode in RDD). Each time an interrupt is risen, a byte is transferred to/from hardware device. After a packet is already delivered, this driver stimulates the Packet Processing Procedure to write or retrieve packets from Host Send Buffer and Host Receive Buffer.

A technique is used in the protocol of receiving a packet form Host Computer. Due to the serial communication is not secured (that means when a byte is not received timely, that byte could be lost and causes the data error), a handshaking protocol is used to ensure that a packet is going to be received correctly in the connection. This protocol uses a special byte to indicate that the host want to send a new packet, if the VoViAc is ready to receive it, a Clear To Send (CTS) signal is send back to host. The host, after has got the CTS, sends packet data to the VoViAc. When the VoViAc received a whole packet, the CTS signal is de-asserted to finish a packet receiving cycle from host computer. In the sending direction to host computer, the host computer could afford a big memory buffer so that the timing issue is not a critical problem anymore.

4.6 Application Message Interface (AMI)

Application Message Interface (AMI) supplies function calls for application to send, receive, inquire messages, and to do barrier synchronization among nodes. The functions are called through a software interrupt. The function to be called is depending on a parameter of the calling. The following is the brief description of the functions.

Function 00H: sends a blocking message. A blocking send requires the kernel to complete the transfer before return to application. This function's input parameters include address of data location to be sent, length of data, destination address(es), ID of message to be sent and routing type. The function returns error code if it could not send the message to the router.

Function 01: sends an unblocking message. An un-blocking send does not require the kernel to confirm that the message has been sent. So that, after the function is called, we can not assure that the data that was sent is available to be change or not. To confirm that, we uses the un-blocking send test function. Parameters of this function is the same as the above.

Function 02H: blocking receive a message. This function is scan input message buffer repeatedly to find out a required message. If the message was not found after a time-out, it returns an error code. Input parameters are physical address of the receive buffer of application, data length, ID of the message to be received and source address.

Function 03H: un-blocking receive a message. The function scans input message buffer for only once. It returns the data to application if message is found or error code in case of not found. . Input parameters are physical address of the receive buffer of application, data length, ID of the message to be received and source address.

Function 04H: does barrier synchronization. This function provides synchronization for processes of application that are running on PEs. When called, this function sends a special message to all nodes and then waits for reply. Once the replies from all nodes have been received, the function returns to application. If a timeout is reached in the waiting for replies, the function returns an error code. Additional parameter is not needed.

Function 05H: does un-blocking send test. After send a message by an un-blocking send function, we do not know whether the send data was already sent or not. Hence, the function provides a method to check this. The function checks the message ID in the output

message buffer. If the message has not been sent, then the message ID should appear in the buffer. The function returns code to indicate the status of the sending (finish or not finish).

User decides whether a unicast or a multicast is done when programs application by putting the destinations in destination field before calling send function.

4.7 Memory Management

Memory management procedures (MMP) are used to manage VoViAc's memory. In fact, due to the small size of VoViAc, these functions only control the memory by allocating and de-allocating memories for processes that are running in the VoViAc but not support for dynamic memory allocation or dynamic de-allocation. The main purpose of MMP is to help command interpreter controls process that are running, being loaded to or cleared from memory.

The MMP controls memory allocation and de- allocation through a memory table. This table have records that contain information for a memory unit. With the VoViAc memory space (128 Word), we divide it into 128 memory units of 1 Word length.

Record0	PID	Used length in unit
Record1	PID	Used length in unit
Record2	PID	Used length in unit
Record126	PID	Used length in unit
Record127	PID	Used length in unit

Figure 4-7 Memory page images

The MMP occupied the TRAP5 of software interrupts. Depend on the parameter of MMP when being called, the correspondent function will execute. (See the appendix A for the details of input parameter and returns).

The functions are the followings:

- Function 00H: This function allocates memory for a new process or allocates more memory for an already allocated process.
- Function 01H: This function de-allocates memory of a process.

- Function 02H: This function returns the PID of a memory location or of a process that called the function.

5 Monitor Programs, Command Interpreter, and Testing

5.1 Load and Run Scheme of Application Programs

The Monitor program provides control of VoViAc's operations. At startup, the Monitor loads Command Interpreter program to the VoViAc. And while co-operating with the Command Interpreter, the Monitor load, run and debug parallel program on the VoViAc. Thus, the Monitor Programs and the Command Interpreter must be coupled very tightly in operations.

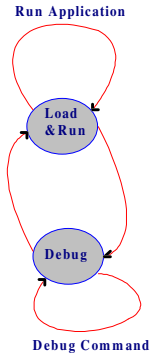


Figure 5-1 Modes of Monitor Programs

We can classify Monitor's operations into 2 modes (Figure 5-1): Load/Run mode and Debug Mode. The Load/Run mode just simply loads an application to memory and then executes it. This mode of execution support for data communicating between the host computer and the VoViAc. Application uses this mode to do processing. Debug mode can either load, run or do debug function with an application. But in this mode, the application can not communicate data with host computer.

The load and run scheme of an application have 3 phases:

- Loading phase: In this phase, the monitor program (MP) communicates with the Command Interpreter in nodes (CI) to load an application to memory of PEs. With the symmetrical characteristic of the CI, the kernel and SPMD computing paradigm of the VoViAc, the MP will send application's executable code to ALL nodes using broadcasting.

- Running phase: In this phase, the MP provides data to application and takes back application results. With kernel's feature, any node can communicate directly with the MP to get data and give back result. Because of application is changed time by time and the MP doesn't know what and when applications will exchange data, so that we have to classify messages in this phase into 3 type with specific message ID. Data requiring message is used incase of application require data form host. Data give back message is used for carry data to application. Result message is used to store back result in host.
- Stop phase: In this phase, application exit and return control to Monitor programs and Command Interpreter.

The application Load/Run scheme allows user to load programs to the VoViAc. For debug work, we use the debug function of the Monitor Programs. The following is the description of commands that user can use to manage the VoViAc system.

5.2 Monitor Commands

Table 1 lists the commands of the Monitor Programs. User runs those commands on host. The Command Interpreter co-operates with Monitor to accomplish those functions. In normal mode, the LR command loads and executes an application. The DEBUG command causes the Monitor Programs and CI change to debug mode. In this mode, we use the LOAD command to load applications to memory. The RUN Command in debug mode run the application until it reaches a break point, which is set by BREAK command. A break point, which was set by BREAK command, can be deleted by B_DEL command. The NEXT command causes applications run to its next program code step by step. Other commands such as memory read (MR), memory write (MW), register read (RR), register write (RW) are used to view and modify memory locations and registers. When user want to leave the debug mode, QUIT command is subjected to use.

Table1: Monitor Commands

Mode	Command	Description	Command name
Normal Mode	Load and Run	Loads an application to memory and executes it	LR [filename]
	Debug	Causes the CI to enter the Debug mode	DEBUG
Debug Mode	Memory Read	Reads a memory location and displays on the host	MR [address]
	Memory Write	Writes to a location in a node	MW [address] [value] [node]
	Register Read	Reads value of a Register in all node	RR [reg_name]
	Register Write	Writes to a register in a node	RW [reg_name] [value] [node]
	Load	Load an application to memory in debug mode.	LD [filename]
	RUN Command	Runs a loaded application from recent location until it reach a break point in debug mode	RUN
	NEXT Command	Causes an application run to the next code	NEXT
	BREAK command	Sets a break points	BREAK
	B_DEL command	Deletes a break point	B_DEL
	QUIT command	Cause the CI to leave debug mode	QUIT

5.3 A Test Application Analysis

The Monitor program has not finished yet. But recently, an application can be loaded and run on the VoViAc. This section describes a test program that was already running on the VoViAc system.

This testing application is a very simple program that uses parallel processing to calculate the summation of numbers from 1 to 30. The same programs are running in 3 node. The array of numbers is separated into 3 parts. Summation of each part will be carried out by a processing element. Then, the root node (node 0) collects results from other nodes and calculates the final result. The final result is then sent to the host computer. This testing program uses message interface of the kernel to communicate with each other. The program flowchart is illustrated in Figure 5-2.

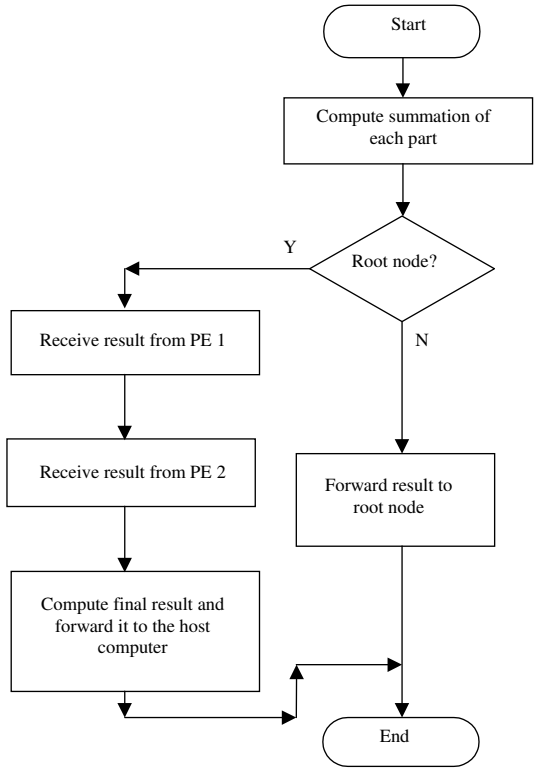


Figure 5-0 Test program flowchart

The command text and turn back results are as follows:

```
CILOAD              → Load the Command Interpreter to the VoViAc
304
  The Command Interpreter is loaded successfully

LR VTEST.OUT        → Load and run the VTEST program
  372
    The input file is loaded successfully

  0 0 0 65 0 0 0 1 0 0 0 ffd1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  → result
```

In the packet of data that was returned to the host computer, we can see the summation is 1D1H (hexadecimal) or 465 (decimal) and this result is correct. This proves that the Router, the kernel system, and Monitor programs are functioning.

6 Conclusions

The VoViAc system has been along several years of growth. The system was attempted to develop from nothing. Previous students had done much of work: designed the system organization, made printed circuit boards and several programs. Recent research is an endeavor of making the system executable both in hardware and software. Recently, some positive results have risen us up in the way of development: The Router was designed and implemented; host computer and the VoViAc can communicate together though serial hardware; the kernel system is established. And as a result, a simple test program, which is using message passing and running in 4 nodes, returns correct result. Although that results still very trivial, it means the system is now executable despite of difficulties for user to use it.

In near future, we will try to finish monitor programs, especially in debug functions, to let user can program on the VoViAc system much easier. Also, a specific C library for kernel functions must be created to allows user programs VoViAc's applications in high level language.

Also, for the main purposes, research about volume graphic algorithm on the VoViAc is the main purpose for us. This research will be difficult and take long time, but we would gain much knowledge from the research. The VoViAc would gain benefit in specific purpose system such as medical CT scanner (for high-speed image representation), in fly simulation system, etc.

Bibliography

[1] Arie Kaufman, Daniel Cohen, and Roni Yagel, “Volume Graphics,” IEEE Computer, Vol. 26, No. 7, pp. 51-64, 1993.

[2] A.Kaufman, “Volume Visualization,” IEEE Computer Society Press Tutorial, Los Alamitos, CA, 1990.

[3] 山崎恭啓, ボリューム 可視化用並列 アクセラレータ VoViAcの アーキテクチャの検討と実現,” 立命館大学理工学部情報システム学専攻修士論文，1996.

[4] 山崎恭啓, 山崎勝弘, ボリューム可視化用アクセラレータの検討，情報処理学会第 5 1 回全国大会論文集, 4S-11, Vol.2, pp. 337-338, 1995.

[5] A.Kaufman and R.Bakalash, “Memory and Processing Architecture for 3D Voxel-base Imagery,” IEEE Computer Graphics and Applications, Vol.8, No.6, pp.10-23, 1988.

[6] H.Pfister, A.kaufman, and F.Wessels, “Towards a Scalable Architecture for Real-Time Volume Rendering,” In Proceeding of the 10th Eurographics Workshop on Graphics Hardware, Volume EG95HW, EuroGraphics Workshop Proceeding Series, pp123-130, 1995.

[7] Pixel-Planes Home Page: http://www.cs.unc.edu/ pxpl/home.html.

[8] Eyles, John, Steven Molnar, John Poulton, Trey Greer, Anselmo Lastra, and Nick England, “PixelFlow: The Realization,” Proceedings of the Siggraph/Eurographics Workshop on Graphics Hardware, Los Angeles, CA, August 3-4, pp. 57-68, 1997.

[9] G.Knittel, “A Scalable Architecture for Volume Rendering,” Proceeding 9th EuroGraphics Workshop on Graphics Hardware, Volume EG94HW, EuroGraphics Technical Report Series, pp. 58-59, 1994.

[10] T.Gunther, C.Poliwoda, C.Reinhart, J.Hesser, R.manner, H.P.Meinzer, and H.J.Baur “VIRIM: A Massive Parallel Processor for Realtime Volume Visualization in Medicine,” Proceeding 9th EuroGraphics Workshop on Graphics Hardware, Volume EG94HW, EuroGraphics Technical Report Series, pp. 103-108, 1994.

[11] S.M.Goldwasser, “A generalized object display processor architecture,” IEEE Computer Graphics and Applications, Vol.4, No.10, pp. 43-55, 1984.

[12] D.Jackel and W.Strasser, “Reconstructing Solids from Tomographic Scans – The Parcum-II System,” In A.A.M. Kujik and W.Strasser, editors, Advances in Computer Graphics Hardware II, pp. 209-227, 1988.

[13] T.C.So, K.Nakajima, and K.Yamazaki, “The message routing mechanism and kernel system of a parallel graphics accelerator,” Proceeding 56th Information Processing Society of Japan Record, Vol.1, 1N-05, pp. 105-106, 1998.

[14] T.C.So, K.Nakajima, and K.Yamazaki, “Message in VoViAc system: Structure, routing and handle,” The Kansai-section Joint Convention of Institutes of Electrical Engineering Record, G304, 1997.

[15] 鷺島　敬之、西澤　貞次、浅原　重夫, 並列図形処理,コロナ社、1991.

[16] Kai Hwang, “Advanced Computer Architecture: Parallelism, Scalability, Programmability,” McGraw-Hill, 1993.

[17] Jose Duato and Sudhakar Yalamanchili, Lionel Ni, “Interconnection networks: An engineering approach,” IEEE Computer Society Press , 1997.

[18] Sivarama P. Dandamudi. E. Horwood, “Hierarchical hypercube multicomputer interconnection networks,” Ellis Horwood Press, Ellis Horwood series in computers and their applications, 1991.

[19] Stuart E. Madnick and John J. Donovan, “Operating systems,” McGraw-Hill, 1974.

[20] Texas Instruments, “TMS320C3x User’s Guide,” 1994.

[21] Texas Instrument, “TMS320C3x/C4x Assembly Language Tools User’ Guide,” 1997.

[22] Xilinx, “The Programmable Logic Data Book,” 1994/1995.

[23] Xilinx, “Mentor Graphics Interface/Tutorial Guide,” 1995.

[24] Xilinx, “XACT User Guide,” 1994.

[25] Xilinx: XACT Hardware & Peripherals Guide, 1994.