LA-SUB--94-77

To: Steve Tenbrick, C-5, LANL
From: John Rasure, Associate Professor, University of New Mexico
Date: June 1, 1991
Subject: Final Report

Final Report For "Analysis of Crossbar Computer Network"

The attached document summarizes the results of the research and development effort.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

MASTER
cp

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

Design and Evaluation of a Distributed Asynchronous VLSI Crossbar Switch Controller for a Packet Switched Supercomputer Network

Andrew J. DuBois and John Rasure
Department of EECE, University of New Mexico

Abstract—A key component in a new high-speed prototype network, called the Multiple Crossbar Network (MCN) being developed at Los Alamos National Laboratory, is a crossbar switching core and its controller. This switching core allows for up to 32 (800 Mbit/second) connections to pass through it simultaneously. The motivation for this network is to allow visualization data to be sent at movie rate speeds to a users workstation from a supercomputer. This paper reviews the key concepts and components of the MCN, and then focuses on the behavioral level design of a distributed asynchronous VLSI controller chip set for the switching core. A unique design methodology is presented and then applied to the controller which uses an externally distributed chip set, and distributed asynchronous finite state machines internally.

I. Introduction

A. Multiple Crossbar Network Overview

The MCN is currently intended to be a high-speed (800 Mbit/second) packet switched point-to-point network in which crossbar switching nodes are used to interconnect the resources of the network [1, 2]. The primary motivation for developing this network is to allow visualization data to be sent at movie rate speeds from a supercomputer to a users workstation. Another motivation is to provide for higher bandwidth communications between supercomputers and other machines which will allow for large grain parallel distributed computing as well as general purpose networking.

There are four primary components which comprise this network. They are: the crossbar interfaces (CBIs) which are protocol processors that store and forward the packets on the network [5]; the high-performance parallel interfaces (HIPPIs) which are the simplex point-to-point channels [3]; the crossbar switches (CBSs) which handle the connecting of source HIPPIs to destination HIPPIs, and allow for up to 32 simultaneous connections [1, 2, 6], and lastly; the resources on the network such as supercomputers, workstations, and central file systems. Fig. 1 shows how the various network entities are connected in the MCN. A more detailed discussion of the MCN can be found in Hoebelheinrich [1, 2].

B. MCN Components

Crossbar Interface

The CBIs, Fig. 2, were developed by Digital Equipment Corporation's Southwest Engineering section, and perform the protocol processing for the MCN [5]. By using a separate protocol processor, the hosts are not burdened with packet processing which reduces the packet delay through the network. The CBIs consist of a specialized processor (AM29K RISC processor) and buffers (four register VRAMs) which handle the storing and forwarding of packets as they pass through the network. The CBIs handle data frame

protocols, intranet routing, network access protocols, network management, and have name server capabilities.

Fig. 1. Block diagram of the Multiple Crossbar Network.

Fig. 2. Block diagram of the Crossbar Interface.

High Performance Parallel Interface

The HIPPI, Fig. 3, was developed by the ANSI X3T9 committee. It is a simplex point-to-point channel which allows for data rates of 800 Mbits/second to 1.6 Gbits/second [3]. It uses a simple synchronous bus protocol to achieve this data rate using a 32-bit or 64-bit data width and a 40ns clock rate. To initiate a connection a simple hand-shaking scheme is used. The source HIPPI asserts its request line along with an I-field which is placed on the data lines. The I-field is an information field which can be used by an intermediate device (e.g. a crossbar switch) for selecting a destination channel. The connection is completed when the destination HIPPI asserts its connect line.

Data frame functionality is incorporated into the HIPPI by means of packet and burst control lines which delimit packet and burst boundaries. A packet can consist of one or more bursts. A normal burst consists of 256 words, and one short burst of less than 256 words is allowed for each packet transfer [4]. Flow control is provided by a ready control line which is asserted by the destination HIPPI whenever it can receive another burst. Error detection is also implemented using LLRC on each burst, and error detection/correction is provided using odd parity on each word of a burst.

Fig. 3. Block diagram of the High Performance Parallel Interface.

Crossbar Switch and Its Controller

The CBS, Fig. 4, is where the multiple paths through the MCN are made. Up to 32 source and destination HIPPIs can be connected to one CBS. The CBS is capable of supporting up to 32 simultaneous connections if all the sources are requesting a unique destination. The CBS prototype, developed by Los Alamos National Laboratory, currently consists of a stack of Fairchild CE2012 crossbar chips, where each crossbar chip handles two HIPPI signals from each input channel, and a controller which monitors the HIPPIs and makes the connection via the crossbar chips [1, 2].

Fig. 4. Block diagram of the Crossbar Switch and its controller.

In general, the CBS controller's role is to monitor the HIPPIs, and initiate and terminate connections through the crossbar. To accomplish this, it checks the source HIPPI request lines to see if they are initiating a connection. If one is, then it checks that channel's I-Field and determines which destination HIPPI it is requesting. The controller then checks to see if the destination is busy or free. If it is busy, the controller can put the requesting channel on a waiting queue, or reservation list, or just go on to another channel. If the destination is free, then the controller commands the crossbar system to make the appropriate connection. When a connection is terminated (i.e. both the request and connect lines are unasserted) the controller marks the destination as free.

The CBS controller design reviewed in this paper uses a set of identical, distributed VLSI chips where each chip uses distributed asynchronous finite state machines internally to implement the CBS control functions. The VLSI chip is referred to as a Distributed Asynchronous Crossbar Controller (DACC), and the complete system is referred to as the DACC system. The DACC system's modular design allows it to control from a 16 x 16 up to a 32 x 32 CBS system. It can make a single connection in approximately 80ns, using a 1.6 μ m CMOS implementation for the control chip, and using the Fairchild CE2012 crossbar chips. The DACC design has been specified and simulated at the behavioral level which allows for an implementation technology to be chosen for the DACCs which best suits the needs of a given CBS implementation (i.e. required controller speed, cost, etc).

II. Discussion and Evaluation of the DACC System

A. Design Decisions

The design decision first addressed was to determine if the CBS controller should be centralized and use a single chip, or be distributed and use multiple chips. The number of I/O pins required by a centralized controller for a 32 x 32 CBS was the main factor for not pursuing this approach. For each input channel a minimum of seven inputs are required (the inputs are the request, connect, and five I-Field lines) which means a total of 224 pins are needed just for the 32 input channels. In addition, various output, power and ground, and control pins to the crossbar are required which would make the total pin count very high for current low cost chip packaging technology. Given this conclusion, it was decided that a distributed controller would be the only practical solution. An additional benefit of using a distributed controller is that it reduces the individual chip complexity (i.e. fewer transistors per chip). Given the I/O pin requirements above, eight channels was the maximum that could be placed on a single chip, and this number was chosen for the final controller design.

The issue of a combined controller and crossbar chip was also studied. Again, the problem of too many I/O pins was encountered. It would be impractical at this time to build a single chip to do both the controller and crossbar functions using the current parallel implementation of the HIPPI. Each HIPPI has 44 signal lines, and any CBS system with more than four input and output channels would require more than 350 I/O pins to implement on a single chip. An alternative to a single chip implementation would be to use one chip per HIPPI signal. The major problem with this approach is that the request, connect, and I-Field lines are all required in making connections, and since this information would be divided among several chips, it would be very difficult to implement a scheme which would make connection decisions and provide this information to all the chips. If a serialized HIPPI is developed, then the I/O pin problem will be removed, but an additional burden will be placed on the processing speed required by a chip or chips to make connections involving multiple serialized 800 Mbit/second channels.

Another major design goal was to maximize the speed of the controller. An asynchronous approach was chosen for the finite state machines in the DACC. The primary reason is that it allows the state machines to run at a much higher speed than a synchronous approach. Along with the asynchronous approach, a distributed implementation for the

state machines was chosen, Fig. 5. This allows for a more modular approach to designing the chip control logic. It is the same concept used in writing modular software, and allows for partitioning specific control functions into separate finite state machines. This technique does require communication between the finite state machines using some kind of interlock mechanism (see section II.F).

Fig. 5. Block diagram of several of the DACC's distributed asynchronous finite state machines.

The last design issue was which crossbar chip to use with the DACC system. It was decided mainly out of convenience to use the Fairchild CE2012 crossbar chips being used for Los Alamos National Laboratory's prototype CBS.

B. Implementation

A block diagram of the DACC system in a 32 x 32 configuration is given in Fig. 6. Each DACC supports eight HIPPIs, so four DACCs are needed to control this CBS configuration. There are three principle areas of functionality for this controller design. The first is the internal functioning of the DACC. The second is the DACC-to-DACC communication, and the third is the interaction with the crossbar chips. Each of these is discussed below along with a discussion of the distributed asynchronous state machines used to implement the control functions.

C. Internal Functioning of the DACC

The basic function of the DACC is to check each of its eight HIPPI request lines one after another to see if any HIPPI is making a request. It also checks each channels connect line to see if a connection has been terminated. This process is performed by the DACC's main finite state machine (MFSM) shown in Fig's 9 and 11.

If a channel is making a request, the MFSM looks at its I-Field to find out which HIPPI output channel it is trying to connect to. The MFSM then checks to see if the destination is free or is in use. To accomplish this a 32-bit memory is used to store the current state of each destination. If the destination is busy, the MFSM tries to reserve the

Fig. 6. The DACC system in a 32 x 32 configuration.

requesting channel for that destination. A 32-bit memory is also used to set a reservation bit for the destination. In addition, a register is set for the requesting channel to identify that it has been reserved. If the destination is busy and it is already reserved, then the MFSM goes on to check its next input channel. If the destination is available, the MFSM checks if this destination has been reserved, and if it has, the MFSM checks if the requesting channel is the reserved channel. If it is not the reserved input channel, the MFSM continues to the next input channel, otherwise the MFSM clears the destination reserve memory bit and the input channel's reserved register. It then sets the destination's connected memory bit to

indicate that the destination is in use, and then signals the crossbar chips to make a new connection from the requesting HIPPI to the destination HIPPI.

The MFSM also checks each of the eight input channel's connect line states to see if any of the channels has terminated a connection. If one has, the MFSM clears its connected memory bit to indicate that this destination is available for another connection.

Since the input channels are considered in a round-robin fashion, the reservation scheme described above allows for the next channel in line, which is requesting a destination currently in use, to get that destination after the current connection is terminated. This prevents another input channel which is in between the currently connected input channel and the requesting channel from stealing the destination. If a more complex priority scheme is used to decide the order in which channels are connected, then it would be necessary to use a reservation list which can contain more than one reserved channel per destination. This information would also have to be shared across the DACCs, since it is a distributed system, which would increase the amount of data transferred between the DACCs (see section II.D).

Another important function performed by the DACC is the handling of the various request and connect line sequences. A normal connection sequence has the source HIPPI asserting its request line, and then at a later time the destination HIPPI asserting its connect line. The normal connection/termination sequence has the request line being unasserted, and then the connect line being unasserted [3].

Abnormal connections can occur and are handled by the DACC. One such abnormal connection sequence occurs when the source asserts its request line, and the destination never asserts its connect line. When this happens there is no way the DACC can tell if the connect line will or will not be asserted. To solve this, the connection is not terminated until the request line drops, and is reasserted again. Other possibilities are that the connect line may drop before the request line (i.e. an abnormal termination), or the request line may drop prematurely. The DACC handles these by terminating the connection only when the connect line drops.

A final function which is implemented by the DACC is the blocking and passing of the request signals through the CBS. The eight HIPPI request lines at each DACC are blocked (i.e. not asserted at the output of the chip) until a connection is made with the crossbar. This is necessary because the connection paths are active through the crossbar even after a connection is terminated by the request and connect sequence given above. If the request lines are not blocked, they will pass through the crossbar core to the previous destination which is an error.

D. DACC-to-DACC Communication

Since this is a distributed chip design, a means of coordinating the multiple DACCs and passing shared data is necessary. This is accomplished by the passing of a token signal. The DACCs are connected in a ring, and a token signal is sent from chip to chip. The token signal consists of two pulses. The pulses are used to strobe data on the common Connected/Reserved data bus into the next DACC. This 32-bit bus is used to transfer the connected and reserved memories one at a time.

The connected and reserved memories are checked and modified by each DACC as it services its eight channels. When a DACC has finished servicing its input channels, it transfers the connected and reserved memories to the next DACC by placing the data on the bus and then generating a token pulse. After the second token pulse is generated by a DACC, it is put into an idle state, and when the second token pulse is received by the next DACC, it is activated. With this scheme, control and shared data are allowed to pass from chip to chip with only one DACC being active at any given time.

E. DACC to Crossbar Interaction

Each DACC has 11 lines connected to the Fairchild CE2012 crossbar chips. These lines allow the DACCs to make new connections through the crossbar core. A write enable signal along with destination and source addresses are used to accomplish this. Since all of the DACCs have these same lines, the write enable and source and destination address lines from each DACC are tri-stated together.

F. Finite State Machine Discussion

The method used for the design of the asynchronous finite state machines in this controller is a hybrid technique developed by David DuBois at Los Alamos National Laboratory. This design method is more convenient and less prone to errors than the traditional asynchronous design techniques. A detailed discussion of the traditional techniques can be found in Fletcher [7]. Fig's 11 - 16 show an example of this hybrid design process.

Fig. 7. State diagram for the Main finite state machine.

Fig. 8. State transition map for the Main finite state machine.

Fig. 9. State variable K-maps and equations for the Main state machine.

The first step that is required to implement an asynchronous state machine using this hybrid technique is to develop a state diagram. This diagram shows the state transitions, and the signals which generate these transitions. Care must be taken to show explicitly what signal states cause a state transition, and what signal states cause the machine to stay in a particular state. This is very important when a state has more than one potential next state destination.

III. Conclusion

This paper has presented an overview of a new 800 Mbit/second super computer network, called the Multiple Crossbar Network, being developed at Los Alamos National Laboratory. One of the major components of this network is a crossbar switching core and its controller which allow for up to 32 (800 Mbit/second) connections to pass through it simultaneously.

The emphasis of this paper has been the discussion of a unique design methodology used in implementing a crossbar controller called the Distributed Asynchronous Crossbar Controller (DACC) system. This system consists of a set of identical, distributed VLSI chips (called DACC's) which use internally distributed asynchronous finite state machines. Using the design methodology resulted in simpler, faster, and more robust finite state machines which reduced the design and debugging time of the DACC system.

References

- [1] R. Hoebelheinrich and R. Thomsen, "Multiple Crossbar Network: Integrated Supercomputing Framework", in *Proceedings of Supercomputing '89*, 1989, pp. 713-720.
- [2] R. Hoebelheinrich and R. Thomsen, "Multiple Crossbar Network: A Switched High-Speed Local Network", in *Proceedings of the 14th Local Computer Network Conference*, 1989, pp. 285-292.
- [3] High-Performance Parallel Interface (HIPPI) Mechanical, Electrical, and Signalling Protocol Requirements, ANSI X3T9/88-127 Rev 7.1.
- [4] High-Performance Parallel Interface (HIPPI) Data Frame Control Requirements, ANSI X3T9/89-146, Rev 2.3.
- [5] W. F. Hedberg, "The Multiple Crossbar Network Interface", in *Proceedings of the 14th Local Computer Networks Conference*, 1989, pp. 299-306.
- [6] Los Alamos National Laboratory internal design documents.
- [7] W. I. Fletcher, *An Engineering Approach to Digital Design*, Englewood Cliffs, N.J.: Prentice-Hall, 1980.