A Microprogrammable VLSI Routing Controller for HARTS*

J. W. Dolter, P. Ramanathan, and K. G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109–2122

Abstract

This paper presents the design and implementation of a VLSI routing controller for use in the Hexagonal Architecture for Real-Time Systems (HARTS) that is currently being built at the Real–Time Computing Laboratory at The University of Michigan, Ann Arbor. The routing controller is a microprogrammed unit designed to function as an intelligent front-end interface for the interconnection network. Unlike other routing controllers known to date, this routing controller allows for the flexibility to investigate several low-level routing algorithms by downloading the appropriate microcode into the routing controller.

The design was carried out using the combination of Seattle Silicon Technology's silicon compiler and Mentor Graphics' schematic capture and simulation tools. The implementation is targeted for a 64-pin package using a 1.2 μm CMOS process.

1 INTRODUCTION

This paper presents the VLSI design and implementation of a routing controller for a distributed computing system based on a hexagonal mesh architecture [1, 4, 5]. This effort is part of a larger research project to design and implement an experimental distributed real-time system called Hexagonal Architecture for Real-Time Systems (HARTS). The ongoing development of HARTS has influenced the architecture of the routing controller to a great extent.

The routing controller will serve as the front-end interface to the hexagonal mesh in each node of HARTS. It implements the data link layer and portions of the network layer of the OSI seven layer model for network communications. The two primary goals of the routing controller are: (i) to provide support for the timely delivery of messages in HARTS, and (ii) to permit various low-level routing algorithms to be investigated.

The routing controller provides support for routing algorithms that are based on either packet switching, circuit switching, or virtual circuit cut-through. The primary routing algorithm being considered for HARTS exploits the properties of virtual circuit cut-through. In adopting virtual circuit cut-through, messages do not always get buffered in the intermediate nodes. Instead, messages arriving at an intermediate node are forwarded to the next node in the route if a circuit can be established. This approach differs from conventional circuit switching schemes since messages do not wait for the entire circuit to the destination to be established before proceeding toward the destination. If a circuit cannot be established at an intermediate node, the messages are buffered at that node for later transmission. It can be shown that by using this approach the mean delivery times for messages can be considerably reduced.

Other notable work concerning routing controllers is presented in [2, 3, 5]. These efforts have focused on achieving high bandwidth communication for a specific topology and a specific routing algorithm. In [2, 3], Dally *et. al* described an an elegant implementation of a deadlock free routing algorithm for a k-ary n-cube. In [5], Stevens proposed a VLSI architecture for the inter-node communication in the FAIM-1 sys-

*The work presented here has been supported in part by the Office of Naval Research under Contracts N00014–85–K–0122 and N00014–85–K–0531, and Grant N00014–87–G–0086. Any opinions, findings and conclusions or recommendations expressed in this paper are those of the authors and do not reflect the view of the ONR.

tem that is based on a hexagonal mesh interconnection topology. But to the best of our knowledge, this architecture is yet to be implemented. Our work, due to the experimental nature of HARTS, emphasizes the ability to investigate multiple algorithms for routing and provide assistance in network testing and diagnosis. Different routing algorithms can be implemented by downloading the appropriate μcode into the routing controller. For example, the routing controller can support the primary routing algorithm in HARTS, source-directed routing, routing in k-ary n-cube for $n \leq 3$, and other mesh routing algorithms.

The routing controller is composed of six receivers and six transmitters interconnected through a time-slice bus. The receivers convert serial data coming from neighboring nodes into parallel form, make routing decisions and, if possible, relay the messages to the next node. These routing decisions are made under the control of a microprogram resident in each receiver.

The key features of the routing controller are:

- Provides front-end interface functions in a single chip solution.
- Supports packet switching, circuit switching, and virtual circuit cut-through.
- Designed using fault-tolerant state machines.
- Incorporates an integrated testing approach.

The implementation was targeted for a 1.2 μm CMOS process resulting in a 102,000 transistor die measuring 8.2 mm $\times$ 7.0 mm. The design was carried out using the schematic capture and simulation tools of Mentor Graphics®, DRACULA^{TM} of ECAD, and the silicon compiler, CONCORDE^{TM} , from Seattle Silicon.

The rest of this paper is organized as follows. Section 2 describes the environment for which the routing controller was designed. The internal architecture is introduced in Section 3. In Section 4 a description of the physical implementation is presented. The paper concludes with Section 5.

2 Operating Environment

HARTS is an experimental testbed for research in distributed real-time computing. The primary goal of HARTS is to investigate low-level architectural issues in the design of real-time systems such as message scheduling, routing, buffering, etc. The dimension of a hexagonal (H–) mesh is defined as the number of nodes on a peripheral edge of the H-mesh. The current version of HARTS is a 3-dimensional H–mesh and is comprised of 19 nodes interconnected in a C-wrapped H–mesh topology, which is formally defined as follows.

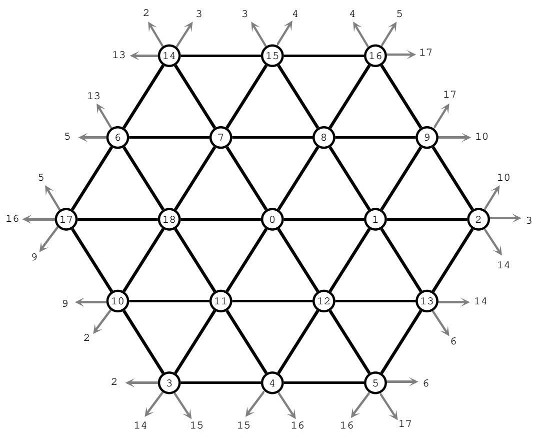

Figure 1: A hexagonal mesh of dimension 3.

Definition 1 A *C-wrapped hexagonal mesh* of dimension e is comprised of $3e(e-1)+1$ nodes, labeled from 0 to $3e(e-1)$, such that each node s has six neighbors $[s+1]_{3e^2-3e+1}$, $[s+3e-1]_{3e^2-3e+1}$, $[s+3e-2]_{3e^2-3e+1}$, $[s+3e(e-1)]_{3e^2-3e+1}$, $[s+3e^2-6e+2]_{3e^2-3e+1}$, and $[s+3e^2-6e+3]_{3e^2-3e+1}$, where $[a]_b$ denotes a mod b.

A C-type wrapping has several nice properties as shown in [1]. First, this wrapping results in a homogeneous network. Consequently, any node can view itself as the center (labeled as node 0) of the mesh. Second, the diameter of a H-mesh of dimension e is $e-1$. Third, there is a simple, transparent addressing scheme such that the shortest paths between any two nodes can be determined by a $\Theta(1)$ algorithm given the address of the two nodes. (At each node on a shortest path there are at most two different neighbors of the node to which the shortest path runs.) Fourth, based on this addressing scheme it is possible to devise a simple routing algorithm that can be efficiently implemented in hardware as

® Mentor Graphics is a registered trademark of Mentor Graphics Corporation

^{TM}DRACULA is a trademark of ECAD

^{TM}CONCORDE is a trademark of Seattle Silicon Technology Inc.

shown in this paper. Figure 1 illustrates an example of a C-wrapped H-mesh of dimension 3 in which the gray links on the periphery are connected to the nodes as indicated by the label of the link.

The C-wrapped H-mesh described above is isomorphic to the interconnection topology presented in [5]. However, the above formalism allows routing of messages between all pairs of nodes to be treated uniformly and does not require any special treatment of the "wrap lines" as was necessary in [5] when the "axial offset" was between e and $2(e-1)$.

The six neighbors of a node in a C-wrapped H-mesh can be thought of as being in directions d_0, d_1, $\ldots$, d_5. To send a message, the source node calculates the shortest paths to the destination and encodes this routing information into three integers denoted by m_0, m_1 and m_2. These three integers represent the number of hops from the source node to the destination node along the d_0, d_1 and d_2 directions, respectively. Before sending the packet to an appropriate neighbor, intermediate nodes update these values to indicate the remaining hops in each direction to the destination. Hence, $m_0 = m_1 = m_2 = 0$ indicates that the packet has reached its destination. This routing scheme and its associated algorithm are currently the primary routing algorithm being used in HARTS.

For example, the (m_0, m_1, m_2) triple calculated using the algorithm in [1] for routing a message from node 1 to node 10 is $(1,1,0)$ (See Figure 1). This triple encodes all the shortest paths from node 1 to node 10, i.e., node 10 can be reached by either going from node 1 to node 2 in the d_0-direction and then from node 2 to node 10 in the d_1-direction or by first going from node 1 to node 9 in the d_1-direction and then from node 9 to node 10 in the d_0-direction. Note that in routing this message from the source to the destination the "wrap" link was used transparently by all nodes in the message path.

In addition to communicating with the routing controllers of the six neighboring nodes, the routing controller interacts with the network processor's buffer management unit (BMU) and interface manager (IM) as shown in Figure 2.

The BMU serves as an interface to the buffer memory subsystem and provides the routing controller with six virtual inbound channels for incoming packets and four outbound channels for outgoing packets. (It was found through both

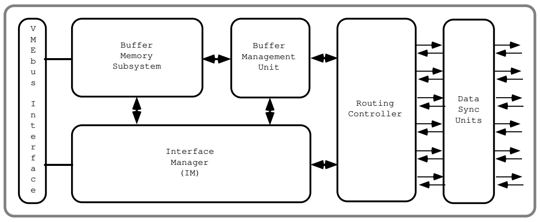

Figure 2: Routing controller environment.

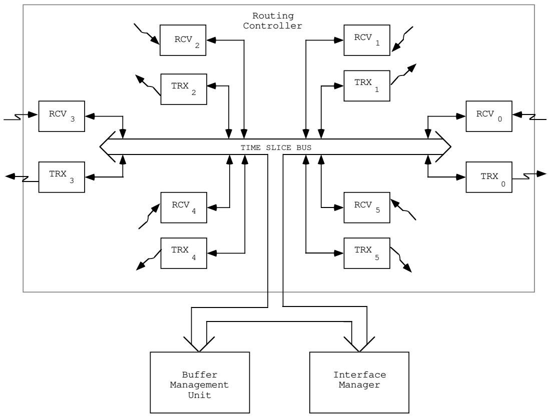

Figure 3: Routing controller.

analytical and simulation models that increasing the number of outbound channels beyond four provided only marginal performance improvement.) The IM interacts with the routing controller during system initialization and monitoring activities.

The communication packets used in HARTS, similar to HDLC, are framed between a *start of packet* (SOP) and *end of packet* (EOP) bytes. Following the SOP is a byte indicating the type of packet (e.g., Data, Control, Broadcast, etc.) and three bytes indicating routing information. The routing controller imposes no restrictions on the length of the packet.

3 Routing Controller Architecture

The routing controller consists of six microprogrammable receivers, six transmitters, and an interconnecting time-slice bus (Figure 3). Three primary goals influenced the architecture of the routing controller. First, the routing controller was to efficiently support virtual circuit cut-through with as little impact on the complexity and performance of the network processor as possible. This feature weighed heavily since the fast delivery of messages is extremely important in HARTS. Second, in order to preserve the experimental nature of HARTS the routing controller had to be flexible enough to allow multiple low level routing algorithms to be investigated. Last, all six transmitter–receiver pairs should eventually fit onto a single silicon die.

A distributed reservation scheme was used to efficiently implement virtual circuit cut-through. In this scheme the transmitters are reservable resources. The users of these resources are the receivers and the BMU of the network processor. To arbitrate for these resources the time-slice bus provides a mechanism for receivers and the BMU to reserve/unreserve the transmitters. Each transmitter independently maintains its own reservation status.

The receivers shift in serial data coming from the neighboring nodes and convert the data into parallel form. The receivers then identify the routing information and determine an appropriate route(s) for the packet. If the packet is to be relayed to a neighboring node, the receivers try to reserve the corresponding transmitter(s). In the case that the transmitter(s) cannot be reserved, the receivers pass the packet to the BMU. Each of the components in the routing controller will be detailed in Section 4.

There were several factors that contributed to the selection of the time-slice bus as the interconnecting structure between the internal components of the routing controller and the BMU. First, the number of components that need to communicate is fixed, and thus the lack of expandability of a time-slice protocol was not an issue. Second, the bandwidth of a parallel time-slice bus matched well with the number of devices and the incoming serial data to the receivers. Last, the time-slice bus was able to support a reservation scheme that allowed receivers and the BMU to check the availability and reserve transmitters, thus simplifying the implementation of virtual circuit cut-through.

The adoption of a microprogrammable architecture over a state machine approach was necessary in order to satisfy the requirement that the routing controller be able to support multiple routing algorithms. For example, the routing controller can support the primary routing algorithm in HARTS, source-directed routing, routing in k-ary n-cube for $n \leq 3$, and other mesh routing algorithms. Furthermore, the routing controller can operate in circuit-switched, packet-switched, and/or a virtual-circuit cut-through mode, simultaneously. The decision of which delivery mode to use can be made on a per-message basis by the source node.

At first, the cost of this flexibility appears quite high without providing any clear benefits. On the contrary, our experience has shown this first impression to be false. The microprogrammable nature of the routing controller has proved useful not only in supporting multiple routing algorithms but also in self-test and network testing roles.

4 IMPLEMENTATION

This section describes the internal architecture and implementation details of the transmitter and receiver modules and the time-slice bus protocol. We will restrict our discussion to a single transmitter-receiver pair (T_{ij}, R_{ji}), where T_{ij} represents the transmitter in node i communicating with node j and R_{ji} represents the receiver in node j communicating with node i. Data being transmitted from T_{ij} to R_{ji} is first filtered through a Data Encoding Unit (DEU) in node i and then a Data Recovery Unit (DRU) in node j before being sent to R_{ji}. This filtering was necessary to convey clocking information from T_{ij} to R_{ji} since nodes i and j do not operate synchronously under the same clock. For example, the sampling clock at R_{ji} can be constructed from the incoming data stream using a digital phase-locked loop.

Figure 4 shows the resulting layout from this implementation.

4.1 Time-slice Bus Protocol

The time-slice (TS) bus consists of a set of signal lines that can be divided into five functional groups: clocks(2), address(4), control(4), data(9), and acknowledgment(1).

The clock group, ϕ_1 and ϕ_2 provide a two phase non-overlapping time-base used in the generation of the remaining signals on the bus. We define a *minor cycle* to be from the rising edge of ϕ_1 to the next rising edge of ϕ_1. A minor cycle is an atomic action on the TS bus.

The address group can be further divided into bus master and device address lines. The bus master lines are used to identify and enable the current master of the TS bus. The device address lines are in turn used by the current bus master to select slave devices during each minor cycle. During normal operation the receivers and the outbound channels of the BMU function as bus masters in tandem with the transmitters and inbound channels of the BMU functioning as slave devices. The bus master lines cycle through the possible master devices in a round robin fashion giving the receivers and the outbound channels equal and deterministic access to the TS bus. This cycle repeats every twelve minor cycles and will be referred to as a *major cycle*. The lower three bits of the address lines identify the addressed slave while the most significant bit conceptually "tees" an incoming packet to both the BMU and a transmitter simultaneously. This provides for an efficient implementation of message broadcast in an H-mesh.

The control lines are used to define the action (command) to be taken during the current minor cycle. The commands can be divided into two fundamental modes of operation: *run* mode and *download* mode. During download mode, the TS bus is used to initialize the microsequencers in the receivers. During this operation the IM maintains control of the TS bus. During run mode, TS bus provides two types of data transfers and supports the reservation of the transmitters. The commands *reservation request* and *reservation release* provide for an easy and efficient implementation of virtual circuit cut-through.

4.2 Transmitter

The transmitter T_{ij} performs two major functions. First, T_{ij} maintains its own reservation status and responds appropriately to the reservation request and reservation release commands. Second, once T_{ij} is reserved, T_{ij} converts the parallel data on the TS bus into properly formatted serial data. These functions are implemented using a PLA for data padding, a decoder for command/address decoding, a shift register, and a fault-tolerant controlling state machine.

T_{ij} has two modes of operation: sync mode and packet mode. In sync mode, T_{ij} transmits a continuous stream of zeros. T_{ij} switches from sync mode to packet mode when a device has reserved T_{ij} and the first byte of the packet is transferred to T_{ij} from the TS bus. Once in packet mode, T_{ij} will transmit either a byte of data if that data was provided in time from the TS bus or insert a "null" byte into the data stream. The null byte will be ignored by the receiver R_{ji}. By supporting null byte transmission, the BMU is relieved of the responsibility of providing a continuous stream of data during peak loads.

When T_{ij} receives a reservation release command, T_{ij} returns to sync mode after a "backoff" period. This backoff time is to allow R_{ji} to finish processing the current packet. T_{ij} is not reservable during this period.

In addition to the normal reservation request and release commands, T_{ij} also responds to the hold and check commands from the IM. The hold command guarantees the IM a reservation of T_{ij} at the end of the current packet. The check command is used by the IM to inquire the status of a pending hold command. This allows the IM to obtain a guaranteed reservation that can be used to transmit time-critical messages.

4.3 Receiver

The receiver is comprised of a Data Detection Unit (DDU), a microsequencer, a data unit, an eight word FIFO unit, and a TS bus interface. The DDU shifts in the serial data from a neighboring node and provides the data to the microsequencer in a parallel form. The data unit consists of an ALU, an accumulator, and four registers. The microsequencer identifies the routing information in the incoming packet and determines the appropriate route for the packet. If the packet is to be relayed to a neighboring node, the microsequencer attempts to reserve the corresponding transmitter. In the case that the transmitter is not currently reserved by any other device, the microsequencer reserves the transmitter and relays the packet through the TS bus. However, if the transmitter has already been reserved, then the microsequencer attempts to reserve alternate transmitters that can be used to deliver the packet. (Note that multiple shortest paths may exist between a source node and a destination node.) If all the alternate transmitters are currently reserved, then the microsequencer relays the packet to the BMU.

The order in which the microsequencer attempts to reserve the transmitters is determined by the microcode. This allows the end user to easily to change the transmitter reservation policy. Transmitter reservation policies can have direct impact on the network traffic patterns.

4.3.1 Data Detection Unit

The operation of the DDU in receiver R_{ji} is tightly-coupled to the operation of T_{ij}. In addition to transforming the incoming serial data, the DDU also identifies the special bytes, SOP and

EOP. The DDU is comprised of a shift register, a depadding unit, a controlling state machine, and a bit FIFO.

The shift register transforms the incoming serial stream into words. After reconstructing a word, the depadding unit identifies and removes the padded zeros from the data as it is transferred to a buffer register in the data unit. The depadding unit also informs the controlling state machine of the receipt of SOP and EOP bytes.

The state machine is used to keep track of the mode of the DDU. The DDU has three primary modes of operation: sync mode, packet mode, and recovery mode. The DDU remains in sync mode when no real data is being received from T_{ij}. In sync mode all internal modules are passively waiting for the receipt of a SOP. On detecting a SOP, the DDU enters the packet mode. In packet mode, the DDU is continuously receiving a stream of data until an EOP is received. The EOP causes the DDU to return to the sync mode.

Sync mode and packet mode are sufficient for operation in the absence of data transmission errors. Errors in either the SOP or the EOP could put the DDU in an undesirable state. For example, an error in EOP would leave the DDU in packet mode even after T_{ij} has completed transmitting the entire packet. To ensure reliable operation in the presence of errors in either SOP or EOP, the DDU was supplemented with a recovery mode. In recovery mode, inter-message gaps are detected and used to reset the DDU into sync mode.

4.3.2 Microsequencer

The microsequencer is the source of intelligence of the receiver. It consists of a controlling PLA, a writable control store, a pipeline unit and a flag unit.

The writable control store provides the user with the flexibility of implementing different low-level routing algorithms. This feature was essential to preserve the experimental nature of HARTS. It is comprised of sixty-four 16-bit words of memory. It is loaded from the TS bus by using the download commands.

The instruction set provides a rich set of operations necessary to implement different low-level routing algorithms. The code size for representative routing algorithms were found to be sensitive to subtle changes in the instruction set implemented. For example, the introduction of asynchronous event instructions, such as the Wait instruction, resulted in a substantial reduction in the code size.

Figure 4: Routing controller layout.

The Wait instruction can be used to wait for a particular event. While waiting for an event the user has an option of enabling an exception handler. This option acts as a pseudo-interrupt to the microsequencer while waiting for an event. The Jump on Condition causes the microsequencer to jump to the address specified in the instruction when the condition being tested is true. It provides the user with a decision making capability in the receiver. The Jump instruction also provides the user with the option of a link capability. Along with the Return instruction this option can be used to implement single-level procedure calls.

The rest of the instructions are used to control the flow of data through the receiver. The ALU instruction can be used to manipulate the routing information, the Load Constant instruction can be used to load immediate data while the the Transfer instruction can be used to transfer data in the receiver. The Set Flags instruction provides the user with the capability of saving state information.

5 CONCLUSION

In this paper we have described a routing controller designed for use in HARTS. The end result is a single chip solution for interfacing the nodes of HARTS.

The most important feature of this design is that it supports the experimental flexiblilty needed to investigate differing low level strategies in network design. In particular, the routing controller directly supports packet switching, virtual circuit cut-through, and circuit switching techniques. It also has ability to investigate multiple low level routing algorithms. It incorporates an integrated testing approach and provides assistance in network testing and diagnosis. The above features could not have been achieved using the traditional hardwired controller approaches.

The design of the routing controller has been carried out in such a fashion that a single transmitter-receiver pair can be fabricated separately. This will allow functional verification at a much reduced cost. An implementation with a single transmitter-receiver pair has been fabricated and is currently under test.

Acknowledgements

We would like thank Richard B. Brown of The University of Michigan and Kendall Russell of Seattle Silicon Inc. for their assistance and support throughout the course of this ongoing project.

References

[1] M.-S. Chen, K. G. Shin, and D. D. Kandlur, "Addressing, routing and broadcasting in hexagonal mesh multiprocessors," *IEEE Trans. Computers*, vol. 39, no. 1, pp. 10–18, January 1990.

[2] W. J. Dally and C. L. Seitz, "The torus routing chip," *J. Distributed Systems*, vol. 1, no. 3, pp. 187–196, 1986.

[3] W. J. Dally and P. Song, "Design of a self-timed VLSI multicomputer communication controller," in *IEEE Int'l Conf. on Computer Design: VLSI in Computers*, pp. 230–234, 1987.

[4] A. J. Martin, "The torus: An exercise in constructing a processing surface," in *Proc. Caltech Conf. on VLSI*, pp. 527–537, 1981.

[5] K. S. Stevens, "The communication framework for a distributed ensemble architecture," AI Technical Report 47, Schlumberger Research Laboratory, February 1986.