A Family of Routing and Communication Chips Based on the Mosaic

Charles L. Seitz and Wen-King Su
Computer Science 256-80
California Institute of Technology
Pasadena, CA 91125

Abstract

This paper describes the specifications, internal design, and performance characteristics of the latest series of Caltech Mesh-Routing Chips (MRCs), slack-protocol-conversion chips, and router-interface chips. These high-performance, self-timed designs were derived from the corresponding parts of the message-passing system of the Mosaic (see the paper "The Design of the Caltech Mosaic C Multicomputer" in this volume), but are packaged as separate chips to allow their use by other projects engaged in the development of highly concurrent computers.

In addition to n data bits, an MRC channel includes a bit to mark the tail of the packet, a self-timed request signal, and an opposite-going self-timed acknowledge signal. MRC channels operate in a queue (FIFO) discipline, in which the request and acknowledge signals provide both timing and flow control. The timing and electrical characteristics of these signals are critical to reliable communication between MRCs. The timing of the internal channels of an MRC is based on a transition-signaling, ripple-through FIFO whose properties are exploited to simplify the internal design the MRC. The MRC implements oblivious, dimension-order, blocking-cut-through (wormhole) routing with an internal organization based on elementary routing automata.

Because the zero-slack protocol of an MRC channel limits the communication performance over long cables, Slack chips provide a conversion to a k-slack protocol. These Slack chips can be implemented with the same FIFO that is used in the MRCs. Dialog chips provide this same slack function, but within a protocol that allows a dialog of control symbols to be exchanged on a channel.

In order to help the system designer deal with these high-speed, asynchronous channels, we also developed router-interface chips to provide synchronization and data conversion between MRC channels and microprocessor buses.

1 Introduction

Our research group has for many years been providing routing chips to other projects, but, until recently, in only one configuration similar to the Mosaic router (Figure 1). These Caltech Mesh-Routing Chips (MRCs) provide packet communication and oblivious, dimension-order, cut-through routing in a 2D-mesh network that connects a set of computing nodes. The north-, east-, west-, and south-bound channels that form the mesh fabric are referred to as the *"news"* channels. The channels to and from the node are referred to as the *"p"* channels. Dimension-order routing assures freedom from deadlock provided that the *po* channel consumes the packets routed to it, and also permits this 2D router to be implemented, as indicated in Figure 1, as a cascade of two, identical, 1D routers. The Caltech MRCs are asynchronous, and their *news* and *p* channels operate according to a zero-slack, self-timed protocol on 8-bit-parallel flow-control units (*flits*) at a rate that is typically throttled to ≈70MB/s.

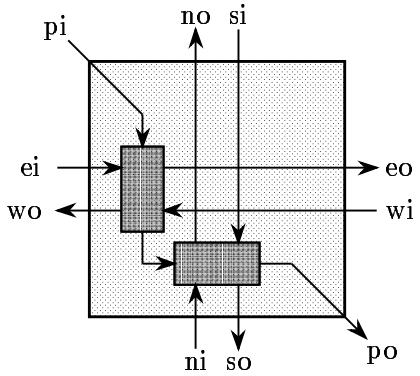


Figure 1: A Caltech Mesh-Routing Chip (MRC)

MRCs are typically used to construct the message-passing network of a multicomputer, or the switch between the processors and memories of a multiprocessor. For example, MRCs form the active, routing-mesh backplane for the Ametek S2010 multicomputer [5, 7] and the Intel Delta multicomputer. The *news* channels are connected on the backplane. The multicomputer nodes plug into the backplane, and connect to the *p* channels. Speculations on other applications of this routing technology can be found in [8].

The "Elko" router used in the production version of the Mosaic C is the latest in a series of pin-compatible but internally different MRC designs that started with the 1987, 30MB/s, "Ginsu" routers (GMRCs) used in the Ametek S2010 multicomputer and the MCC ES-kits. The 1989, 80MB/s, "Frontier" routers (FMRCs) were used in early Mosaic systems, the Intel Delta, and several other projects. In 1991–92, Intel developed their own version of the "Frontier" router, the iMRC, for the Intel Paragon

multicomputer. The iMRC employs the same internal, routing-automata scheme [3] as the FMRC, and many of the same circuits, but has 16-bit rather than 8-bit flits, and supports packet broadcast. Also, its external channels operate according to a streaming (also known as a transmission-line write-ahead) protocol similar to the slack protocol.

The 2D, 8-bit version of the "Elko" MRC, designated the EMRC-2D8, was designed by the authors at the end of 1991. In addition to providing better setup- and hold-time margins than earlier routers, the EMRC is more modular and more tolerant of process variations. It was, accordingly, easy to generate EMRC-2D9 (2D with 9-bit flits) and EMRC-1D16 (1D with 16-bit flits) variants of the basic EMRC-2D8 design.

The zero-slack, self-timed protocol used in MRC channels operates correctly even when the routers are connected through cables, but the cycle time increases with the delay of the cable. In order to maintain the channel bandwidth over long cables, a series of Slack chips was developed using the same FIFO circuits that are used internally in the EMRCs. These slack chips convert between the zero-slack protocol of the router channels and a k-slack protocol that permits the sender to get k requests ahead of the acknowledges from the receiver.

The *pi/po* channels presented to the node create a challenging design problem for building interface circuits from commodity parts. On the *pi/po* side, the interface is asynchronous, self-timed, one or two bytes wide, and capable of operating at rates in excess of typical clock frequencies. On the node side, the interface must typically be synchronous and word-wide. Elko-Router Interface Chips (ERICs), based in part on the Mosaic packet interface [9], provide highly reliable synchronization, and reduce the data rate by providing a 4- or 8-byte synchronous, microprocessor-bus interface to and from the node.

This family of routing, communication, and interface chips can be used together to build the message-passing network and interfaces for highly concurrent computers. The layout modules can also be combined to provide more highly integrated solutions on single chips.

2 Specifications for the Routing-Chip Channels

Packets traverse the routing network in a queue discipline. Each n-bit data item presented on a channel by the sender is acknowledged by the receiver, and is, accordingly, referred to as a *flow-control unit*, or *flit.* A packet is composed of a sequence of flits starting with a routing *header,* and ending with a flit that is tagged as the *tail.* The "wormhole" routing used in the EMRCs is a blocking form of cut-through routing. If the required output channel is available, the head of an incoming packet is advanced immediately into this channel, and the packet is spooled through this established path until the path is broken by the tail flit. If the required output channel is

already in use, the packet is blocked in place by the flow-control discipline until the required output channel becomes available.

Although routing is recognized as the function of the EMRC chips, reliable, high-bandwidth, packet communication between EMRC chips has required nearly as much design attention as the routing itself, and is the most appropriate starting point for this exposition.

2.1 Router-Channel Signals, Timing, and Performance

Each n-bit router channel is composed of $n + 3$ signals: a request (r) in the direction of the channel, an acknowledge signal (a) in opposite direction, a tail bit (t), and the n bits of data ($0, \ldots, n\text{-}1$). The names of these signals are formed as illustrated in Figure 2. The bidirectional channels generally used in multicomputer networks (see section 3 of [6]) are implemented simply as a pair of unidirectional channels.

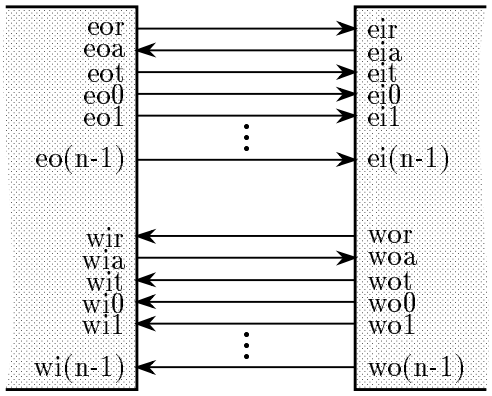


Figure 2: East- and west-bound channels connecting two routers.

The request and acknowledge signals of a channel conform to a pipeline form of 2-cycle (transition) signaling (see figures 7.16 and 7.24 in [4]), and provide both timing and flow control. The use of transition signaling assures that the rate of transitions on the request and acknowledge signals is no higher than the highest rate of transitions on the data signals. Each flit is conveyed by a transition of the request signal followed by a transition of the acknowledge signal. This is a *zero-slack* protocol: The channel output may not generate another request until the outstanding request has been acknowledged by the channel input.

At a channel output, data signals become valid at least t_{dr} before the transition of the request signal, and remain stable until after the corresponding transition of the acknowledge signal. At a channel input, data inputs are sampled coincident with the transition of the acknowledge signal ($t_{\mathrm{ad}} = 0$). The minimum values for the three parameters shown in Figure 3, $t_{\mathrm{dr}}, t_{\mathrm{ra}}$, and t_{ar}, can be observed directly on a channel that is operating at full rate. The unobservable parameter t_{ad} is, by design, determined by the

difference between two very small delays within the channel input circuits, and is so small in magnitude that it can be taken to be zero. The t_{ra} interval is determined by the receiving router, and may be larger than its minimum value if the flow is blocked downstream. The t_{ar} interval is determined by the sending router, and may be larger than its minimum value if the "upstream" packet is being supplied to the sending router more slowly than the output channel can operate.

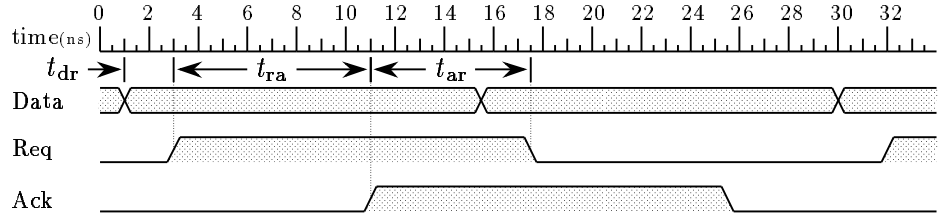


Figure 3: Router-channel timing between two routers that are physically close and operating at maximum speed.

Typical values for these key parameters at 25C and V_{dd}=5V, with EMRCs connected by 10cm PCB traces (approximately 20pF, consisting of 15pF wire and 5pF input-pin capacitances) are: $t_{\mathrm{dr}} = 2.0$ns, $t_{\mathrm{ar}} \geq 6.5$ns, $t_{\mathrm{ra}} \geq 8.0$ns, and $t_{\mathrm{ad}} = 0.0$ns. In terms of these parameters, the period is $(t_{\mathrm{ra}} + t_{\mathrm{ar}}) \geq 14.5$ns, corresponding to a bandwidth $\leq$ 69MB/s. The setup-time margin is $(t_{\mathrm{dr}} + t_{\mathrm{ra}}) \geq 10$ns. The hold-time margin is $(t_{\mathrm{ar}} - t_{\mathrm{dr}}) \geq 4.5$ns.

The 10ns setup-time margin was deliberately favored within the 14.5ns period. For routers that are close together, the 4.5ns hold-time margin is more than adequate. The expressions above can be augmented to allow for the additional delay, t_p, in the wires that connect two routers that are separated by a greater distance. (These expressions can also be augmented to allow for skew.) Additional delay due to increased capacitance and flight time increases the period by $2t_p$, and increases the hold-time margin by t_p, but does not change the setup-time margin. The routers were, accordingly, designed to have a larger setup-time margin than hold-time margin.

2.2 Electrical Characteristics

Although it would be possible and desirable to use techniques such as low-voltage drivers and receivers, or simultaneous bidirectional communication [2], the routing chips that we have developed to date have been intended principally for applications in which the routers are physically close, and in which the simplicity of conventional 5V CMOS signals outweighs the benefits of these other techniques. For these ordinary CMOS inputs and outputs, typical input-pin capacitance is 5pF, and input and output pins are protected against ESD and latchup to at least $\pm$100mA. The input switching threshold at V_{dd}=5V is 2.4V $\pm$ 0.1V.

EMRCs and Slack chips are ordinarily fabricated in the MOSIS 1.2μm

scalable CMOS process ($\lambda = 0.6\mu$m (n-well), the Hewlett-Packard CMOS-34 process).

All output pins are driven by a p-channel—n-channel MOSFET pair sized to produce nearly the same transition and delay times for positive-switching and negative-switching signals. The i–v characteristics plotted in Figure 4 were obtained from electrical measurements of the pins of a typical EMRC chip at 25C and $V_{\rm dd}$=5V. Although the static saturation current of the p-channel output driver to 0V is ≈35mA, whereas the static saturation current of the n-channel driver to $V_{\rm dd}$=5V is ≈31mA, the devices exhibit asymmetric nonlinearities due to differences in velocity saturation between the p-channel and n-channel transistors, and due to the n-well process.

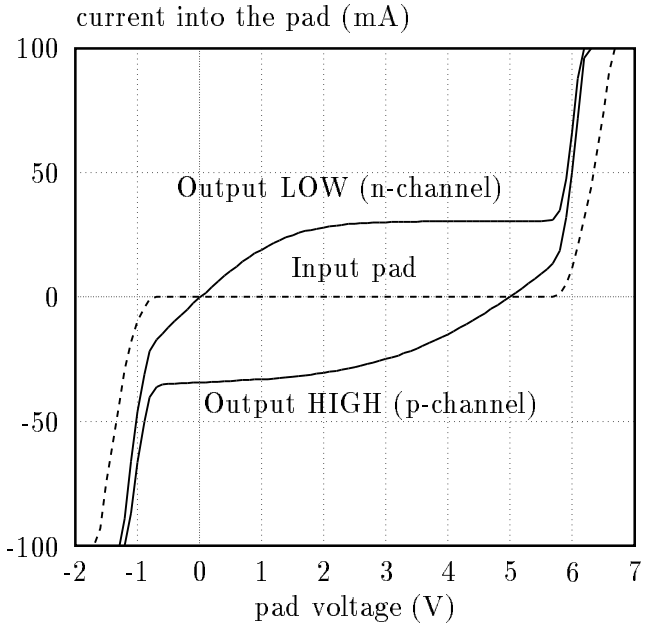


Figure 4: EMRC I/O-pad i–v characteristics.

The net result of the transistor sizing is to produce very similar (±0.2ns difference) switching and delay times into typical (20pF) lumped-capacitive loads. The source impedances are also closely matched. Load lines to the half-voltage points yield effective output impedances for source-terminated driving of transmission lines of ≈88Ω for the high output, and ≈84Ω for the low output.

Although these CMOS pad drivers are far from ideal, they are adequate as source-terminated transmission-line drivers for long PCB runs and even for cables in the 90Ω-impedance range. Channels may be connected through up to about 10m-long cables, typically shielded ribbon cables, or ordinary ribbon cables with alternate wires grounded. Because of the zero-slack protocol, performance is degraded by the flight time of the signals, but reflections from the far end of long cables return while the outputs are assured not to be switching.

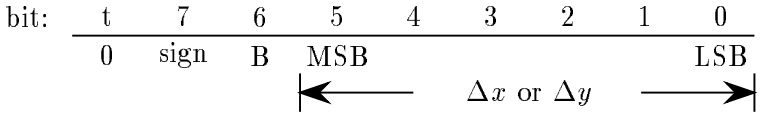
Power dissipation of the routing chips is dominated by the output drivers, and can be calculated based on the load capacitance and expected number of transitions/s. For example, the average number of transitions per flit for random data with an 8-bit-router channel is six (one request, one acknowledge, and four data transitions, ignoring the occasional tail), resulting in an average energy of 1.5nJ/flit if the signals drive 20pF loads ($\frac{1}{2} \cdot 6 \cdot 20\text{pF} \times (5\text{V})^2 = 1.5\text{nJ}$). Thus, the power required to maintain a throughput of 60Mflits/s on such a router channel is 90mW.

3 The Elko Mesh-Routing Chips

The three current versions of Elko Mesh-Routing Chips — the EMRC-2D8 (2D with 8-bit flits), the EMRC-2D9 (2D with 9-bit flits), and the EMRC-1D16 (1D with 16-bit flits) — are identical in timing and performance, and similar in packet format.

3.1 Packet Format

A packet may be of any length so long as it includes the appropriate header. A header flit of the EMRC-2D8 is represented as follows:



The sign bit is 0 for positive and 1 for negative; the Δx or Δy distance is represented in sign and magnitude, with the 6-bit magnitude represented in binary. B specifies packet broadcast, which is not implemented in the current EMRCs.

The EMRC-2D9 has a 9th data bit that can be used for parity checking or other purposes; this 9th bit is simply passed through, and does not influence the routing. The EMRC-1D16 has the same header format as the EMRC-2D8, except that the sign bit appears in bit 15 and the magnitude in bits 5–0; bits 14–6 are passed through but ignored. The Mosaic router has the same header format as the EMRC-2D8, except that there is no broadcast bit and a 7-bit magnitude appears in bits 6–0.

On the *pi, ei,* or *wi* channels of 2D routers, the header is composed of a Δx flit followed by a Δy flit. The sign of the Δx flit is significant only for the *pi* channel: if the magnitude is non-zero, the sign determines whether the packet is routed east (+) or west (−), and the packet will leave the router with the magnitude of the distance decremented. When a packet enters a router on *pi, ei,* or *wi* with the magnitude equal to zero, the Δx header flit is stripped off, and the packet is passed to the y router. For a 1D router such as the EMRC-1D16, there is no y router, and the packet is passed to the *po* channel. These EMRC-1D16 routers can, of course, be used in pairs to provide 2D routing.

For packets that enter the y router from the x router, or that enter the y router from the *ni* or *si* channels, the header is composed only of a Δy flit. The y router is identical to the x router, but connects to northbound (+) and southbound (−) rather than eastbound and westbound channels. When a packet enters the y router with the Δy magnitude equal to zero, the Δy header is stripped off, and the packet is routed to the *po* channel.

If $\Delta x = \Delta y = 0$ on the *pi* channel, the packet is routed as usual through the x and y routers back to the *po* channel. The packet is terminated and the path freed by any non-header flit in which the tail (t) bit is 1.

3.2 The EMRC Internal FIFOs

Caltech MRCs are optimized for throughput rather than latency. Throughput is maintained through the input, head-flit-decrementing and -testing, decision, merge, and output stages of the routing process by operating these stages in a pipeline. Instead of the clocked registers used in synchronous pipelines, the MRCs use asynchronous, ripple-through FIFOs (*cf* [4, 10]). The "Ginsu," "Frontier," and "Elko" routers have each used different FIFO designs that determined the internal timing and overall design style of the routing chip.

The EMRC uses the unusual, two-cycle FIFO shown in Figure 5. This FIFO is similar to the two-cycle FIFO that Ivan Sutherland advocates for micropipelines [10]. The output of a Muller C-element [4] becomes 1 if both inputs are 1, becomes 0 if both inputs are 0, and otherwise holds its value. The effect of this chain of C-elements is to control the flow of data in a FIFO discipline through the two rows of latches. A cell is "empty" if its C-element output is the same as the C-element output (or acknowledge signal) to its right. For example, the entire FIFO is empty if the input request and acknowledge, the output request and acknowledge, and all of the C-element outputs are all 0 or all 1. A cell is "full" if its C-element output differs from the C-element output (or acknowledge signal) on its right. Its data is latched in the upper row if its output is 0, and in the lower row if its output is 1. Whenever an empty cell appears to the right of a full cell, the empty cell's C-element will change state, latching the data from the full cell, and making the full cell empty. Fullness and data thus propagates to the right, and emptiness to the left.

Although the control part of this self-timed FIFO is speed-independent, correct operation requires that the propagation delay of the latches be less than the propagation delay of the C-elements and their output drivers. In fact, the propagation delay of the latches is 1ns, and of the C-elements and their driver amplifiers is 3ns, for transitions in either direction. The structure is so symmetric that it is easy to assure that fullness ripples forward at the same 3ns/cell rate that emptiness ripples backward, a property that is inessential but simplifying. The minimal period of the FIFO is thus 6ns, corresponding to a maximal throughput of 166MB/s. The FIFO is capable

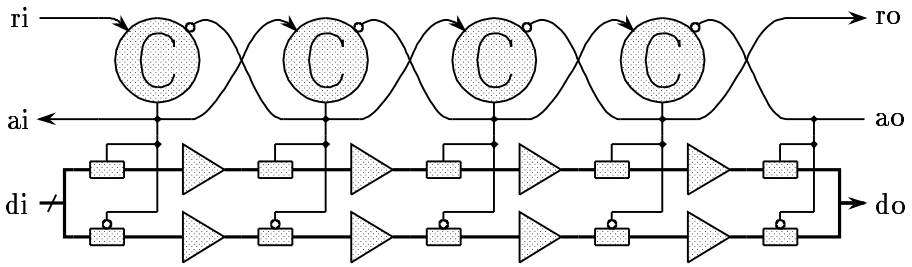


Figure 5: The EMRC, two-cycle, ripple-through FIFO.

of operating at about twice this speed, but extra delays are included in the C-element output driver for reasons described in the following paragraph. The external ≈70MB/s throughput rate is throttled by the pad-input and pad-output circuitry.

The difference between the 3ns C-element delay and the 1ns latch delay is responsible for a very useful property of this FIFO. On successive stages, the data will be available 2ns earlier than in the previous stage, relative to the request transition, up to a maximum of the 6ns period of the FIFO. Thus, after only three FIFO stages, the data is assured to lead the request by 6ns. The EMRC design takes advantage of this property by performing the head-decrementing operation (see Figure 6) in ordinary combinational logic (without the need for a completion signal) after a 3-stage input FIFO. The decrementer produces its result in somewhat less than 3ns, and two following FIFO stages restore the lead time of the data. A head bit is generated at the beginning of the three-stage input FIFO from reset and a unit delay from the tail bit of the previous flit, and is carried in an extra bit of the FIFO; thus, a packet is framed internally by both a head and a tail bit. The head bit is the carryin of the head decrementer. The carryout from the head decrementer indicates whether the head flit was zero, in which case the head flit will later be stripped. This zero bit is carried in still another extra bit of the FIFO.

One other useful property of this FIFO is that the lead time of the data over the request transition can be at least partially preserved when combining the data from the two rows of latches. The lead time of the data into the output stage, together with the acknowledge-switched output multiplexor, is what provides the 2ns t_{dr} interval (Figure 3) in the EMRC routers.

Although the basic cell is somewhat larger, the EMRC FIFO has numerous advantages over the FIFO used in the earlier FMRC. Correct operation of this FMRC FIFO, which is similar to the asynchronous FIFO illustrated in figure 7.24 of [4], depends on more precise determination of internal circuit delays. Also, whereas the EMRC FIFO-control output switches only once to transfer the data into the cell, the FMRC FIFO-control output switches twice to generate a narrow pulse. Only a limited number

of transitions and their timing margins can be accommodated within the period of a high-speed, ripple-through FIFO. The EMRC FIFO reduces the number of required transitions to the minimum of one, maximizes timing margins, and provides two useful properties that simplify the rest of the router design.

3.3 The Internal Design of the EMRC

The internal structure of the EMRC 1D router is shown in block-diagram form in Figure 6. Just as the 2D router is composed of two 1D routers, the 1D router is composed from more elementary automata that also operate on packet streams. For this reason, we refer to the elements within the block diagram of Figure 6 as *routing automata,* and to the schema of this router as its routing-automaton formulation.

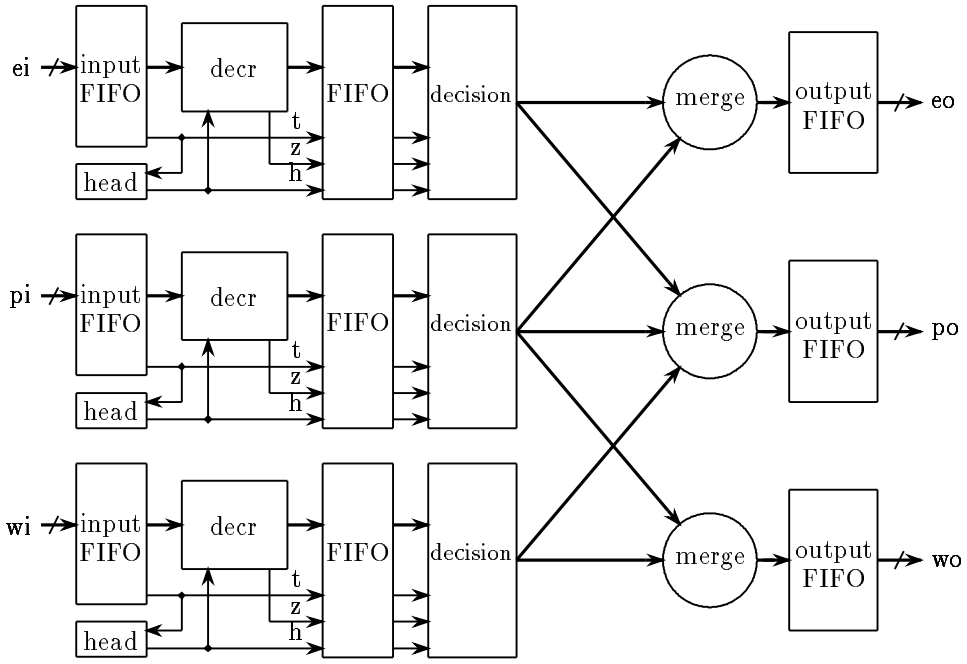


Figure 6: Internal routing-automata structure of a 1D EMRC router.

A "switching yard" follows the previously described preprocessing in the input FIFOs and decrementers. Based on the zero and sign bits of the packet header, the decision automata produce a packet-level request to the appropriate merge element. The merge elements arbitrate these packet-level requests, and set their internal multiplexors to accept flits from the source that receives the packet-level acknowledge. The packet is then spooled through this established path by flit-level requests and acknowledges until the tail flit is acknowledged, after which the decision element removes the packet-level request. The three-way merge also strips the header by acknowledging the first flit without generating a request to the output FIFO.

The merge operations are strictly fair. If a packet-level request to a two-way merge occurs when the other source is acknowledged, this request will be granted as soon as the other source removes its request. Thus, in heavy traffic, packets from the two merge sources alternate. The three-way merge is implemented with two, two-way arbitrations, such that *pi* can source one-half of the output packets, and *ei* and *wi* one-quarter each. This scheme gives *pi* some priority for injection into the network, and is also symmetrical. Giving priority to injection is based in part on the hypothesis that the multicomputer runtime system (see the paper "The Design of the Caltech Mosaic C Multicomputer" in this volume) may need to evacuate packets from a node at least as fast as they enter.

Due principally to the FIFOs in the internal routing paths, the typical *ei*→*eo*, *wi*→*wo*, *ni*→*no*, and *si*→*so* path-formation latency for the head of a packet is ≈30ns.

4 Slack and Dialog Chips

Slack and Dialog Chips contain a pair of protocol-conversion circuits that translate between:

- the zero-slack, self-timed signaling protocol used by the EMRCs or Mosaic channels, and
- a k-slack signaling protocol suitable for conveying packets through long cables.

4.1 Slack-chip Usage

Slack or Dialog chips are used as illustrated in Figure 7. The slack-channel CMOS outputs are not suitable for driving transmission lines because the k-slack protocol does not constrain the time at which reflections may return to the outputs. Slack or Dialog chips must, accordingly, be used together with transmission-line drivers and receivers appropriate to the type of cable. Slack and dialog chips are communication devices; they are generally oblivious to the format of packets. The nominal rate of the slack channels is 60MB/s.

Slack and dialog chips share a common pinout and signal-name convention, although not all signals are used by all chips. In all cases, the request and acknowledge signals employ transition signaling. The signals and timing specifications of the *ei* and *eo* (where *e* now stands for "Elko" rather than "east") channels are the same as those of the EMRCs, and include a 9th data bit. The signals of the slack input and output channels have 3-character names constructed as *si* or *so* (where *s* now stands for "Slack" rather than "south") followed by one of (*r a c t 0–8*), where:

r is the self-timed request (transition signaling).

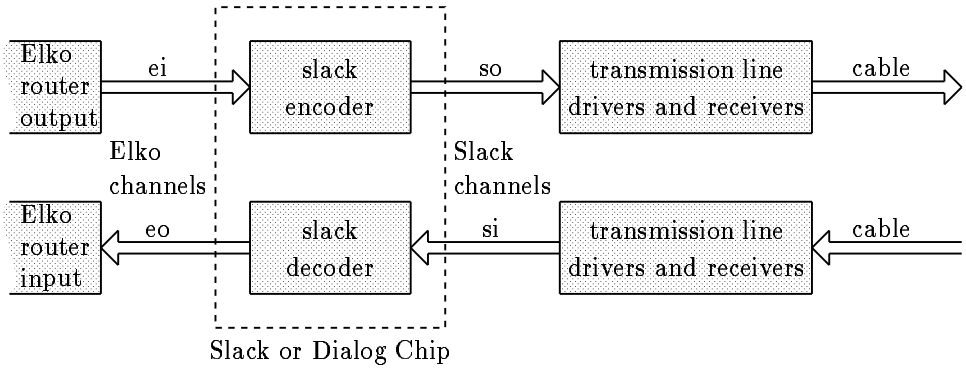


Figure 7: Slack- and dialog-chip usage.

a is the self-timed acknowledge (transition signaling). This signal is not used in Dialog chips.

c is a control-escape bit. This bit is not used in Slack chips.

t is the tail bit, which marks the end of a packet.

0-8 are data bits 0-8.

Slack-chip slack channels, which include the back-going acknowledge signal but not the control-escape bit, have exactly the same signals as the zero-slack channels. The difference between the zero-slack and slack protocols is in the timing and flow control encoded in the request and acknowledge.

Dialog-chip slack channels do not include the back-going acknowledge signal; all of the signals propagate in the direction of the channel. Flow control is accomplished by the Dialog chip inserting an occasional ACK control symbol into the stream conveyed by the opposite-going channel. A control symbol is distinguished by the control-escape bit being 1. Additional control symbols are used for other purposes.

4.2 Timing specifications of the slack channels

The request and acknowledge signals of the slack channels are a variation of the transition-signaling discipline used for the zero-slack channels. The slack protocol conforms to two essential rules:

Rule 1: Each *transition* of a request signal, whether positive or negative, conveys a flit. At the slack output (*so*), data are presented a half period before the request transition, and are held at least a half period after the request transition.

The time scale in Figure 8 assumes a nominal 16ns minimal period (62.5Mflits/s maximal rate); the actual period varies somewhat from chip to chip. The use of a minimum of a half period between data and request

transitions maximizes the tolerance to skew introduced by the cable between the data and request signals. At the slack input (*si*), the data are sampled at the *sir* transition ±0.2ns. Thus, the communication from an output channel to an input channel will tolerate skew between data and request signals up to 0.2ns less than the worst-case (minimum) half period.

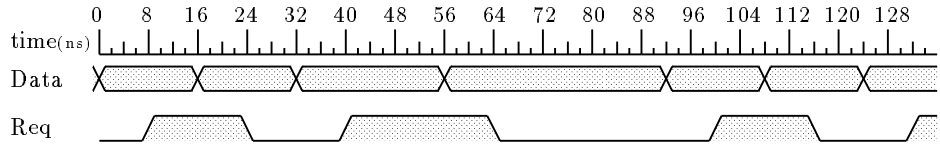


Figure 8: An example of the slack-channel protocol, as seen by the sender. The data are irrelevant a half period after a request transition, but are held at their previous value. Delay in addition to the minimum may be caused either by the data not yet being available to the encoder because of the rate at which the packet is provided to the encoder, or by the encoder not yet receiving an acknowledge indicating the freeing of slack buffers at the receiver.

Rule 2: For a slack k, corresponding to a slack decoder with k slack buffers, the encoder is permitted to present the data and request for k more flits than it has received acknowledges.

A small technicality: In the zero-slack (non-interference) protocol employed in the EMRCs, the time at which data is sampled is *referenced to the acknowledge* signal rather than to the request. The sender is, accordingly, permitted to present $k+1$ rather than k more requests than it has received acknowledges. For slack protocols in which the time at which data is sampled is *referenced to the request* signal, a slack of k corresponds to the receiver having at least k slack buffers, and the minimum value of k to support communication is one.

For Slack chips, acknowledges are also encoded as transitions. In order to limit the maximum frequency of the acknowledge signal, the receiver assures that the time between acknowledge transitions is at least the nominal period. For the Slack-20 chip, $k = 20$.

4.3 Slack-chip structure and performance

The easiest way to construct a slack chip, as illustrated in Figure 9, is to use a FIFO for the slack buffers in the slack decoder, and the FIFO control alone in the encoder as a model of the state of the decoder. The FIFO control is used here as a curious type of asynchronous, up–down, unary counter that is incremented each time a flit is sent, and decremented each time an acknowledge is received.

The slack decoder contains the k slack buffers, which are organized as a k-deep, ripple-through, asynchronous FIFO. The *sia* acknowledge is generated when a flit is passed from the FIFO to the *eo* output, and its frequency must

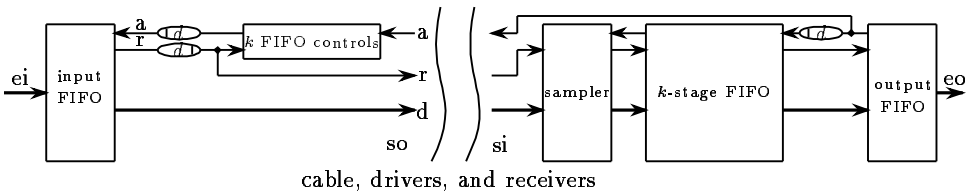


Figure 9: Structure of a slack encoder and decoder based on ripple-through FIFOs.

be limited by an internal delay. The *decoder latency,* the minimum time from the *sir* transition at the decoder input to *this flit* being presented at *eo* and being acknowledged by a transition at *sia* is ≈50ns for the Slack-20 chip.

The encoder maintains a count of the number of requests generated minus the number of acknowledges received. This count is initialized to 0 on reset, is incremented on each request generated, is decremented on each acknowledge received, and inhibits additional requests if the count is equal to k. The count is maintained implicitly in a replica of the decoder's FIFO control. It requires ≈35ns in the Slack-20 chip for the *soa* acknowledge transition to propagate through the FIFO and control circuitry to the input circuitry where *sor* and *eia* are generated. This *encoder latency* is an inevitable part of using ripple-through FIFO, and, as with the decoder latency, grows with k. The latency is as small as it is because the FIFO structures in the Slack-20 chip have been set to a higher speed than the FIFOs in the EMRC.

Starting from a condition in which all of the slack buffers are free, the encoder may generate a stream of k flits before receiving any acknowledges. For $k = 20$ and a period of 16ns, this stream lasts for 320ns. For this stream to continue without pause, the sum of the round-trip cable delay, encoder latency, and decoder latency must be not more than 320ns. The ≈85ns encoder and decoder overhead limits the round-trip cable delay for full-speed operation to ≈235ns. For cables with typical propagation velocities of $\frac{c}{1.5}$, $k = 20$, and 62.5MB/s operation, this limitation corresponds to a cable length of ≈23m. The slack protocol conveys flits correctly with larger cable delays, but not at the full bandwidth.

4.4 Dialog-Chip Protocols and Performance

We expect in the near future to replace the current use of Slack chips in Mosaic cables and in the ATOMIC LAN [1] with a more general scheme that we refer to as "dialog" protocols. For Dialog chips, acknowledges are encoded as control flits ($c = 1$) on the opposite-going channel. In order to minimize the use of channel bandwidth for acknowledges, a single ACK symbol is sent to acknowledge r flits. We expect to standardize on $k = 64$ and $r = 16$.

In an effort to reduce the magnitude and dependence on k of the

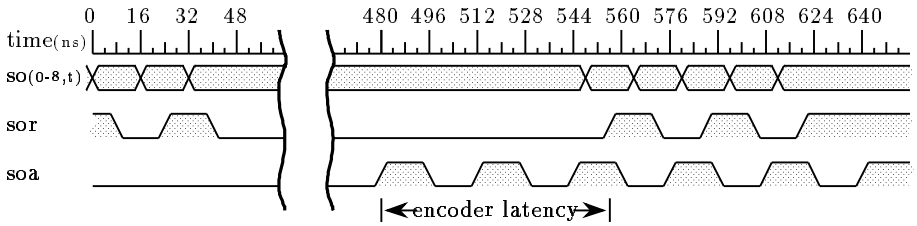


Figure 10: Example of the slack-channel protocol for an 8-flit packet with slack $k = 20$, as seen at *so,* and starting from a condition in which the receiver is blocked and all but 3 slack buffers on the receiving end are filled. The encoder may fill the 3 remaining slack buffers, but must then pause until it receives an acknowledge.

encoder and decoder latency, our design for a Dialog chip employs register-based rather than ripple-through FIFOs. The slack circuits are otherwise conceptually identical to those in the Slack chips: The decoder includes a k-deep, register-based FIFO, and the encoder includes only the FIFO control. The register-based FIFO, which includes a pair of cyclic, 1-of-k counters for the read and write pointers, also takes care of detecting each time the rth flit has left the decoder. Because r divides k, it is necessary only to detect when the read pointer advances past those FIFO cells that are multiples of r. On each such occurrence, an ACK control symbol is inserted into the opposite-going stream. On the receipt of an ACK, the dialog encoder advances its replica of the read pointer by r. Of course, the encoder's replica of the write pointer is advanced by one each time a flit is sent.

The only advantage of the dialog protocol over the ordinary slack protocol is that it is extensible. Control symbols do not appear on the *ei/eo* side of the Dialog chip, and may be inserted anywhere within a packet or between packets on the Dialog channels. Control symbols are not buffered; hence, they can be sent at any time, independent of the flow control. The minimal implementation of the Dialog chip employs only one control symbol for the flow-control acknowledge (ACK). However, between a pair of chips, there can be an arbitrary "dialog" of control symbols that are never blocked. Thus, dialog protocols allow, for example, continuous monitoring of the continuity of a bidirectional (full-duplex) channel, such that normal network blocking can be distinguished from such error conditions as a disconnected cable or an unpowered communication partner.

5 The Extended Family

There are several other members of this family currently being designed or fabricated, and that are described briefly here.

5.1 Elko-Router Interface Chips (ERICs)

The Elko-Router-Interface Chips (ERICs) provide an interface between the byte-wide, self-timed *p* channels of an EMRC – either an EMRC-2D8 or EMRC-2D9 – and either an 8-byte-wide synchronous bus (ERIC8 chip), or a 4-byte-wide synchronous bus (ERIC4 chip).

The bus interface can best be understood as a synchronous interface to a send FIFO and a receive FIFO, each (at least) 16 words long. Packets are sent by appending to the send FIFO from the bus, and are received by causing the receive FIFO to drive the bus and to advance. Between the bus interface and the self-timed channels, the ERIC provides FIFO buffering, highly reliable synchronization to the external clock, and word↔byte conversion (packing and serialization). The EMRC-2D8 and -2D9 routers strip the routing header off of each packet as it is routed. In order to maintain word alignment, the flits of an incoming packet are packed into words starting with the third byte. Thus, the packet received has the same word alignment as the packet sent, and the same word is marked with the tail bit. The ERIC chips also include a status register loaded from the third byte of each incoming packet. The status register outputs appear on pins, and may be used to control node functions.

5.2 EMRCs with broadcast

Wormhole-routing broadcast is deadlock-free only if broadcast is restricted to one direction – positive by convention – in each dimension the mesh. Broadcast can be implemented by a simple variant of the decision element of the EMRC (see Figure 6). A packet from *pi* or *ei* tagged for broadcast is sent both to *po* and *eo* if the header is positive and non-zero. In effect, a copy is dropped off at each node until the header is decremented to zero. If the header is zero, the packet is, as usual, routed only to *po.*

5.3 Bypass chips

As noted in the paper, "The Design of the Caltech Mosaic C Multicomputer," in this volume, the bisection of a large mesh network can be augmented economically by using *bypass channels.* For example, on a network of large radix, any center-directed packets can be diverted at the $\frac{1}{3}$ and $\frac{2}{3}$ points onto the bypass channels if their destinations are outside of the central $\frac{1}{3}$ of the nodes in this dimension. The $\frac{1}{3}$ and $\frac{2}{3}$ points are optimal for maximizing average throughput under random traffic; the $\frac{1}{4}$ and $\frac{3}{4}$ points are optimal for minimizing latency. In practice, any pair of points in these regions is nearly optimal. On a 128×128 Mosaic, for example, bypass chips can be inserted between 8×8 boards 5 boards (40 nodes) from each edge.

A bypass chip requires only a very elementary router, which can be assembled from EMRC parts with no new layout, and slack or dialog circuits to drive the cables.

5.4 A Router with Integrated Fault-Detecting and Interface Circuits

We recently developed a custom router for a commercial company that is developing a multicomputer aimed at high reliability and non-stop operation. This chip includes an EMRC-2D8 core; fault-detecting circuits on the requests and acknowedges of all of the *news* channels; synchronous, full-duplex, flit-wide channels to and from the node; and timeout and clock-detection circuits to assure consumption at *po* even when the node is unplugged or unpowered. This project has been a learning vehicle for us to understand how to achieve more nearly non-stop operation of routing networks.

Acknowledgments

The research described in this paper was sponsored by the Defense Advanced Research Projects Agency, and monitored by the Office of Naval Research. John Toole at DARPA started this effort by encouraging us to make this mature routing technology more widely available. In addition, let us acknowledge the help we have received from MOSIS in fabricating and packaging these chips, and from our operations manager, Arlene DesJardins, in packaging, testing, and distributing them.

References

[1] Cohen, D., Finn, G., Felderman, R., DeSchon, A. ATOMIC: A High-Speed, Low-Cost, Local Area Network. USC/ISI Technical Report ISI/RR-92-291, Sept-1992.

[2] Kevin Lam, Larry Dennison, and William Dally. Simultaneous Bidirectional Signaling for IC Systems. *Proceedings of the 1990 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 430–433, IEEE, 1990.

[3] Charles M. Flaig and Charles L. Seitz. Inter-Computer Message Routing System With Each Computer Having Separate Routing Automaton for Each Dimension of the Network. US Patent filed 2 June 1988, issued 14 April 1992, # 5,105,424.

[4] Charles L. Seitz. System Timing. Chapter 7 in *Introduction to VLSI Systems* by Carver A. Mead & Lynn A. Conway, Addison-Wesley, 1980.

[5] Charles L. Seitz, William C. Athas, Charles M. Flaig, Alain J. Martin, Jakov Seizovic, Craig S. Steele, Wen-King Su. The Architecture and Programming of the Ametek Series 2010 Multicomputer. *Proceedings of the 1988 Hypercube Conference,* ACM Press, New York, 1988.

[6] Charles L. Seitz. Multicomputers. Chapter five in *Developments in Concurrency and Communication,* edited by C. A. R. Hoare, Addison-Wesley, 1990.

[7] Charles L. Seitz. Concurrent Computation and Programming. Chapter one in *VLSI and Parallel Computation,* edited by Roberto Suaya and Graham Birtwistle, Morgan Kaufmann Publishers, 1990.

[8] Charles L. Seitz. Let's Route Packets Instead of Wires. *Advanced Research in VLSI: Proceedings of the 1990 MIT Conference,* MIT Press, 1990.

[9] Jakov Seizovic. The Architecture and Programming of a Fine-Grain Multicomputer. Caltech Computer Science PhD thesis in preparation; publication expected June 1993.

[10] Ivan E. Sutherland. Micropipelines. (Turing Award paper) *CACM* 32:6 (720–738), June 1989.