3.1 Receivers

The switch chip contains eight identical receiver modules, one associated with each of the eight input ports. The receiver module performs five major functions: (1) administering the link flow-control protocol, (2) checking the link CRC codes, (3) buffering incoming data, (4) decoding packet routing information, and (5) deserializing incoming packets into 8-flit *chunks* when the packet is blocked. Buffering is accomplished with a 31-flit first-in-first-out (FIFO) queue. When an incoming packet encounters no contention for the selected output port, packets are immediately forwarded via the Switch chip's crossbar. When the packet is blocked, 8-flit packet chunks are sent to the central queue for temporary buffering.

3.2 Crossbar Routing

The switch chip incorporates an unbuffered logical cross-bar that allows packets to pass directly from the receivers to the transmitters. These flit-serial crossbar paths permit packets to pass through the chip with low latency whenever there is no contention for the output port. As soon as a receiver decodes the routing information carried by an incoming packet, it asserts a crossbar request to the appropriate transmitter. If the crossbar request is not granted by the time the entire first chunk of the packet has been received, the crossbar request is dropped (and hence the packet will go to the central queue). Each transmitter arbitrates crossbar requests on a least-recently-served basis. A transmitter will honor no crossbar request if it is already transmitting a packet or if it has packet chunks stored in the central queue.

3.3 Transmitters

There are eight transmitter modules, one corresponding to each output port. When the central queue contains packet chunks destined for a transmitter, that transmitter requests the next packet chunk. Transmitter modules are served by the central queue in a least-recently-served fashion. As long as data is available, one transmitter is served each clock cycle. The transmitter accepts packet chunks from the central queue, serializes them, buffers them in a 7-flit output FIFO, and transmits them to the link in accordance with the link flow-control protocol. The transmitter is also responsible for computing and transmitting the CRC codes.

3.4 Central Queue

The central queue implements a buffered, time-multiplexed 8-way router. It accepts packet chunks from the receivers, stores them, and eventually passes them to the appropriate transmitters. The Vulcan central buffering scheme is similar to the centrally-buffered, dynamically-allocated switch described by Tamir and Frazier [14], but the Vulcan chip allocates central queue space on a chunk basis instead of a packet basis.

The central queue stores packets until they can be transmitted. The storage, a 128 by 64-bit dual-port RAM, holds up to 128 eight-flit packet chunks. The queue does not reserve a fixed amount of space for each output port; storage is allocated dynamically according to demand.

As long as queue space is available, one receiver can be served every cycle. As long as data is available, one transmitter is also served every cycle. Thus the bandwidth through the central queue matches the bandwidth into and out of the switch chip: eight flits per cycle.

To prevent starvation, receivers requesting service from the central queue are divided into two groups: those offering *critical* chunks and those offering *non-critical* chunks. A critical chunk is one that is ready for immediate forwarding to its destination transmitter. One chunk of central queue storage is reserved for each transmitter. The remaining central queue storage is shared. Whenever a receiver has a non-critical chunk and space is available in the shared pool, the receiver requests service from the central queue. Whenever a receiver has a critical chunk, it requests service. Critical chunks are stored in the destination transmitter's reserved queue slot. Priority is assigned to the receivers on a least-recently-served basis.

3.5 Tuning

In the Vulcan prototype, all tuning is software-controlled, and utilizes delay logic at each Vulcan output port (VOP). In contrast, the Monarch parallel processor incorporated a hardwired method for tuning a serial link [2]. Two service commands enable the service software to synchronize data arrival for a channel. One command (tuning-control) adds an identical amount of delay to each of the VOP's Data and Tag outputs, and selects a separate delay adder for the VOP's Token input. A second command (cycle-counting) determines the number of cycles taken to: (a) send a pulse out of the VOP on a Data or Tag signal, (b) receive the pulse at the downstream input port, and *reflect* it back onto the Token signal, (c) receive the reflected pulse back on the VOP Token signal. "Reflection" at the input port is an action utilized specifically during tuning that: (1) clocks the 9 input signals into a register R, and (2) OR's the output of R into its Token output flip-flop. A number of cycle-counting experiments—for carefully chosen delays—are performed to determine the best delay adder for the VOP's output and input signals for synchronizing signal reception.

4 The Vulcan Prototype

The goal of the Vulcan prototype was to build the minimal set of hardware necessary to demonstrate all key components of Vulcan. To this target, the prototype comprised processor nodes, communication links, switch boards, I/O nodes, and a host node that provides a connection to a standard Micro Channel[1] slot. Full functionality of the system was achieved by early 1992. Figure 1 showed the components in a complete Vulcan system. Each component of the system was designed and simulated for 50 MHz operation. The Intel i860 was chosen as the processor because it was an off-the-shelf readily available single-chip