The H-Bus: A Media Acquisition Bus Optimized for Multiple Streams

Jeffrey J. Wong, John A. Watlington, V. Michael Bove, Jr.

{jjwong, wad, vmb}@media.mit.edu
MIT Media Laboratory, 20 Ames St., Room E15-324, Cambridge MA 02139 USA
(617) 253-0334, fax (617) 258-6264 (correspondence to third author, please)

ABSTRACT

The H-Bus is a dedicated input bus designed for transmitting digital media streams from multiple sources to one acquiring host device. Unlike previous solutions to this problem which utilized a star topology to connect each source to a bank of analog-to-digital converters in the host device, the H-Bus places the A/D converters at each of the sources and uses a linear topology to connect the devices together in one contiguous chain. In this paper we explain the engineering requirements which motivated our design, describe the electrical and physical characteristics of, and applications for, the H-Bus.

Keywords: multimedia capture, buses, interfaces

1 INTRODUCTION

Consider the problem of controllably acquiring and digitizing multiple media streams, such as might occur in a digital television studio, a machine vision laboratory, or a computer-processed surveillance application. The H-Bus, developed by the MIT Media Laboratory, is a daisy-chained media acquisition system in which the analog/digital (A/D) converters are placed at the sources, and a shared connection is made to a single digital port on the computer or other destination device. The distributed, modular nature of this system means that system cost scales with the number of sources to be acquired, while placing the A/D converters at the sources enables use in an electrically noisy environment. With the incorporation of simple subsampling hardware along with the A/D, several sources may be monitored with additional bandwidth/resolution allocated to sources of greater interest.

Through limiting the expected bus traffic to media streams, much of the overhead necessary for the flexibility of general purpose buses may be removed. For example, the bus arbitration mechanism, which typically favors the more common random bus traffic case to the detriment of real-time data transmission, may be designed to meet the demands of transporting multiple media streams. The H-Bus is designed to allow the maximum possible number of uncorrupted data streams to be sent to the bus master, even if the user attempts simultaneously to enable more devices on the bus than possible due to bandwidth constraints. A prioritized arbitration scheme ensures that bandwidth is concentrated on the higher priority streams and prevents excess packets from stealing needed bus resources.

A set of design parameters were developed to specify the requirements of this stream-oriented bus, including:

- High throughput capacity - 640 Mbit/s - sufficient to carry two medium resolution color video channels along with associated audio and sideband information.
- Capable of handling up to eight external, physically separated slave devices per bus.
- Capable of handling multiple virtual channels of data from each slave unit. Each virtual channel is an independent media stream, such as a single audio channel, or a single component video signal.
- Graceful degradation under heavy load
- Low cost
- Non-centralized bus arbiter, in order to allow a serial cabling topology.
- Easy to implement via conventional construction techniques - no ASICs or extensive surface mount work needed for slave prototype development.

2 CURRENT STANDARDS

One of the fundamental design problems faced by digital multimedia system architects is the transfer of large amounts of time-critical data between separate components in a system. A medium resolution digital video data stream may easily require bandwidths of 216 Mbit/s [1] or more. Prior to the recent growth of multimedia applications in personal computers, non-backplane peripheral buses with sufficient bandwidth for digital video were specialized, expensive, and designed for high performance computing, such as HiPPI. As the processing capabilities of personal computers increased, existing bus standards such as the Small Computer System Interface (SCSI) were upgraded and a new generation of higher bandwidth bus standards were developed.

While no currently available industry standard bus met all the requirements of our application, several were close enough to deserve discussion. In particular, limiting ourselves to electrical (vs. optical) interconnect, these are the High Performance Serial Bus (IEEE P1394), the Scalable Coherent Interface (IEEE P1596), and the latest extension to the SCSI standard.

High Performance Serial Bus The IEEE P1394-1995 Standard for a High Performance Serial Bus [2] had primary design goals of low cost, high bandwidth, ease of use, and support of both backplane and external cable variants. Each bus has a tree topology supporting up to 63 slaves in a non-cyclic point-to-point network constrained both in length and tree depth, with up to 1024 buses supported in a system. To reduce connector and cabling costs, the serial interconnect uses six conductors, two differential signal pairs and one power pair. Data transfer rates of 98.3 Mbit/s and 196.6 Mbit/s are currently available, with 393.2 Mbit/s and 1.2 Gbit/s forthcoming.[4] P1394 was designed to simultaneously handle both asynchronous and isochronous bus transfers. In the earliest stages of design, the bus architects realized that a normal arbitration scheme with no fairness mechanisms would allow the closest node to the root to dominate the bus. As a result, they introduced fair and isochronous arbitration mechanisms.[7]

Although IEEE P1394 meets most of our application requirements, the maximum throughput provided by the current implementations of P1394 are insufficient bandwidth for our application. While it is touted as a bus for digital video,[7] in order to carry multiple channels on a single bus the video must be subsampled and compressed. Second, a lack of P1394 interface devices at the time would have delayed implementations of H-Bus devices.

[1] The data rate of a 720x480 raster at 30 fps, with the chroma channels subsampled by 2, or CCIR 601/656.

[2] Apple Computer's trademarked FireWire is a proprietary (but widely licensed) implementation of IEEE P1394, basically the one considered here.

Scalable Coherent Interface The IEEE P1596-1992 Standard for a Scalable Coherent Interface (SCI) is a very high speed interconnect protocol that uses point-to-point links to achieve throughputs of 8 Gbits/s in local configurations. Designed to be an interface suitable for connecting thousand processor shared memory supercomputers, SCI was designed to be a fast, flexible interface which blurs the distinctions between I/O buses and high speed memory buses. The physical interconnect (wire version) used between nodes is unidirectional and consists of eighteen wires. Nodes may be connected using a variety of topologies, with up to 64K nodes in a system. Four priority levels are used to differentiate bus traffic, in order to minimize the latency of time-critical data. The highest priority is granted 90% of the usable bandwidth, and the remaining bandwidth is dedicated to the lower priority transfers. Fairness and latency reduction techniques such as priority inheritance and queue-entry reservation mechanisms ensure that the network stays fast and fair for high priority traffic.[8]

This priority scheme has a weakness when applied to "real-time" data transfers, which require a guaranteed minimum bandwidth and maximum latency.[2] As a result, multiple conflicting proposals on how to modify SCI have arisen and are still evolving, including SCI/RT-1995 and the P1596.6 SCI Specification for RealTime Applications. This, along with the need for an integrated (and complex) interface per node, prevents its use at present time. Nonetheless, with its emphasis on high throughput low latency flexible networks, SCI remains an attractive future solution for our application.

UltraSCSI The latest incarnation of SCSI using electrical interconnect is Ultra SCSI. Ultra SCSI (formally known as Fast-20 SCSI) is a part of the SCSI-3 standard drafted by the American National Standard XT310 Technical Committee. It defines electrical and mechanical improvements which double the maximum throughput of SCSI-2 buses while maintaining backwards compatibility with existing equipment. It has a parallel topology with up to eight devices in a single bus. Distributed arbitration is accomplished through bus contention logic to simplify wiring, and a rich command set allows the bus master to initiate a whole range of actions on a slave or obtain configuration and status information from a slave device[6].[5]

Several characteristics prevent Ultra SCSI from being an optimum solution for a dedicated real-time input bus. In order to maintin compatibility with existing SCSI devices, a parallel bus topology is used, resulting in lower signal speed and integrity when compared with serial topology buses. In addition, the timing of bus transactions is kept longer that what is necessary with current technology; arbitration periods, for example, must last 2.4 μs. It also retains the protocol overhead of a general-purpose bus, unnecessary for our application. Finally, a significant portion (eight wires) of the available wiring resources in the bus are used for control signals. Although this allows the bus state to be unambiguously determined from these lines, this approach is wasteful of wires.

3 ARCHITECTURE

Our proposed design, the H-Bus, meets the above requirements. It consists of between one and eight data-producing, physically separate slave units linked together in a linear topology, with a single bus master at one end of the chain. Data transfers are unidirectional, running downstream from the active slave (the slave which is actively transferring data) to the bus master, and grouped into small (4 Kbit) packets for flow control.

A separate command bus runs in parallel with the data bus, supporting lower bandwidth bidirectional communications between the devices on the bus. It is used by the bus master to configure or query slave device parameters (such as resolution, bit depth, component sampling structure, frame rate) and to issue direct commands to a slave (turn on, turn off, pause, etc.). To minimize cost, the H-Bus uses common 50-pin SCSI cables and connectors. The data bus uses 16 data lines, necessitating a data rate of 40 Mbit/s in order to provide the desired aggregate throughput, a data clock, and two arbitration lines. The command bus (described separately in Section 3.2) is utilizes four additional serial communication lines.

Table 1: H-Bus Signals

Name	Description	Pin	Name	Description	Pin
D15	Data signal	46	D3	Data Signal	29
D14	Data Signal	45	D2	Data Signal	28
D13	Data Signal	44	D1	Data Signal	27
D12	Data Signal	43	D0	Data Signal	26
D11	Data Signal	42	HV_CLK	Data clock	37
D10	Data Signal	41	ACK	Arbitration ACK	34
D9	Data Signal	40	REQ	Arbitration REQ	35
D8	Data Signal	39	HV_TxD+	Serial+ from Master	47
D7	Data Signal	33	HV_TxD-	Serial- from Master	48
D6	Data Signal	32	HV_RxD+	Serial+ from Slaves	49
D5	Data Signal	31	HV_RxD-	Serial- from Slaves	50
D4	Data Signal	30	GND	Ground	13, 14, 15, 16, 17
GND	Ground	1, 2, 3, 4, 5, 6, 7			18, 19, 20, 21, 22
		8, 9, 10, 11, 12			23, 24, 25, 36

3.1 Data Bus

This section describes the transfer of data between slave and master devices on the bus: the electrical signaling layer, the packet definition, and the arbitration protocol.

3.1.1 Electrical signaling layer

Unlike parallel topology buses such as SCSI-2 or PCI, data transfers on the H-Bus are made through point-to-point links between slave nodes. Each slave actively retimes and retransmits incoming data to allow a longer aggregate bus length and higher data transfer rates. Each link is actively terminated at the receiving node. The bus signals for the H-Bus are defined in Table 1. Pin number assignments refer the standard pin numbering for Centronics SCSI-I and SCSI-2 connectors as defined by ANSI.[6] Note that to save on wiring costs, many cables use less wires for ground than pins.

Backplane Transceiver Logic (BTL) [3] was selected as the electrical signaling protocol for the data bus, having several features which made it attractive. First, it was designed as bus logic (for IEEE 896 - FutureBus), using small amplitude (1 v.), edge-rate controlled (6 nS) signals to reduce crosstalk problems along transmission lines. Second, its receivers apply filtering to reject spurious glitches and transitions which do not correspond to properly driven waveforms. Although BTL is a single-ended signaling scheme, the cost of differential signaling with its improved noise immunity is the near doubling of clock rate necessary to obtain an equal data rate over the same number of wires.

Data values being transmitted along the data lines are synchronized using `HV_CLK` - an NRZ clock signal (whose rising and falling edges both correspond to valid clock edges). After gaining control of the bus through arbitration (which also indicates that the bus master has buffers available for incoming data), the active slave generates data and a dual-edged data clock signal (up to 40 MHz) indicating valid data words in a packet, and sends them downstream. Its downstream neighbor receives the data signals, filters and thresholds them to TTL levels, then latches them using a recovered clock edge. The output of this latch is translated back into bus logic levels and retransmitted, along with an appropriate clock, onto the downstream link to the next slave unit (or

[3] BTL is a trademark of National Semiconductor Corp.

the bus master.)

In asynchronous bus transfers, a handshaking protocol is used which, although allowing nodes of differing speeds and capabilities to communicate, can limit the data transmission speed for long communication channels. For example, during an asynchronous transfer along an eight meter SCSI chain, the target node must wait at least 84 ns before it can change the data lines from their previous value due to the round trip time of the `REQ/ACK` signals. Synchronous transfers, on the other hand, take advantage of knowledge about the data receiver [4] to transfer data as rapidly as possible, using a (delayed) handshake to ensure that buffers at the receiver aren't overrun. In order to maximize bus throughput, the H-Bus uses synchronous transfer of data within a "packet". Each slave in the chain not sourcing data immediately forwards received data to the downstream node. The four-phase asynchronous handshaking scheme used for bus arbitration is also used to control the overall flow of packets, preventing bus master buffer overflow.

3.1.2 Packet definition layer

A data packet consists of a sixteen bit header and 4 Kbits of payload, arranged as one word of header, followed by 256 words of data. The packet header identifies which virtual channel the data belongs to, as well as providing packet synchronization information (to detect the loss of packets in a data stream.) It is composed of the following bit fields:

Bits [15 ... 8] are the channel ID. Each virtual channel of data has a unique ID; thus, a slave node with three video outputs and two audio outputs would use five separate unique channel IDs. Up to 256 independent virtual channels are supported.

Bits [3 ... 1] are used as a sixteen value packet counter to identify the current data packet's position relative to other packets in the stream. Thus, if the slave node fails to transmit a small number of packets, the bus master will be able to detect the error and place the next transmitted packet at the correct location in the memory store.

Bits [5 ... 4] are similarly used as a four value frame ID to enable bus master to identify the frame which corresponds to the data packet. Thus, if long-term overflow condition causes a large number of packets to be lost, the bus master can place the next incoming data packets into the correct frame in memory, restoring frame sync.

The packet size chosen for the H-Bus is a compromise of several constraints: bus arbitration overhead, master address generator throughput, and the likelihood of buffer overrun in slaves blocked from bus access. Small packet sizes have several disadvantages: Dividing a constant data rate stream into smaller packets increases packet interarrival rate. As a result, the master interface has less time to process (generate an address and store) each packet. Eventually, as packets are made too small, the master interface would limit bus throughput. In addition, since arbitration delay is independent of packet size, smaller packet sizes lead to higher proportions of total bus time devoted to arbitration. Although larger packet sizes are more efficient, they result in competing slave units obtaining bus access less frequently. The likelihood of slave buffer overflow is increased unless the amount of slave buffer memory is increased accordingly. The packet size chosen, 4 Kbits, allows small packet buffers in slaves to hold several packets.

[4] In a more complicated bus protocol, SCSI for example, these synchronous transfer parameters may be negotiated between devices as part of each transfer. In the H-Bus, these parameters are fixed, simplifying the controller and minimizing the transfer overhead.

Figure 1: H-Bus Arbitration and Packet Flow Control

3.1.3 Arbitration protocol

In a bus characterized by random traffic, such as a computer backplane, short messages are sent between devices on the bus at pseudo-random intervals. If multiple devices request the bus simultaneously, a typical arbitration scheme will try to grant access to the devices in an unbiased fashion to minimize the latency seen by any one device. This approach is acceptable because messages are not time critical.

In a stream-based system, however, latency is an extremely important issue. Each source generates packets at fairly regular time intervals, and unlike the above system, the buffer space on the source may not be able to store more than one or two packets. Therefore, if an arbiter attempts to resolve conflicts with an unbiased algorithm, all streams will suffer degradation due to lost packets if traffic exceeds a certain fraction of the theoretical capacity of the bus. Furthermore, since even a single packet of lost data will cause objectionable artifacts in a multimedia stream, the fair arbiter may destroy all useful inputs to the master if the bus is overloaded for a short segment of time.

The H-bus arbitration scheme assigns priority to slaves based on their position in the bus topology. Highest priority is given to the slave closest to the master, and subsequent lower priorities are assigned to slaves further upstream. When two slaves attempt to request the bus simultaneously, the slave with the highest priority gets to make the request. This arbitration scheme is easily implemented in the physical structure of the H-Bus by using a serial arbitration scheme i.e., running the request lines downstream through each of the slaves and running the acknowledge line back upstream through the slaves. Thus, if a slave of lower priority attempts to request the bus, its request must pass through all the slaves of higher priority. If any of the higher priority slaves needs the bus, it can block the incoming request signal and replace it with its own. Similarly, if an acknowledge packet granting bus control is being sent upstream to a lower priority slave, a higher-priority slave can block the transfer of the acknowledge signal and utilize the bus to transfer its own data. This handshaking scheme is shown in Figure 1.

Overflow errors in a stream are guaranteed to be the loss of an integral number of packets, and are recoverable through synchronization information contained in the data packet headers. Bus designers have the option of implementing flow control mechanisms in software which enable the bus master to automatically reduce the throughput of the lower priority slave media sources if overflow errors occur.

Several bus arbitration protocols are still under performance evaluation. There is evidence that a bookkeeping scheme may help ensure that higher priority slaves do not unnecessarily block lower priority streams,[3] but the cost of implementing a complex arbiter may offset the benefits of such a system.

3.2 Command Bus

The expected command traffic patterns are bidirectional, small (on the order of 16 to 128-bit) and variably sized data transfers. In addition, the total thoughput is small and the latency requirements minimal.

Instead of multiplexing both the command and the data streams over the same physical bus, as is done in the majority of buses, we provide a separate interconnects. This simplifies the bus interface, allowing the data bus to be optimized for the transfer of data packets of a fixed size, whereas the command bus minimizes the number of physical wires used.

3.2.1 Electrical layer

The command bus is implemented as a four wire multidrop network consisting of one differential pair of wires connecting the bus master transmitter to all the slave receivers and another differential pair which connects (in parallel) all slave transmitters to the master receiver. Unlike the data bus, each slave merely passes the command bus through to the next device in the bus, without retiming or reshaping the signal. Since the communications bandwidth of the command bus is low (it is currently operating at 38.4 Kbits/s), this simpler topology is acceptable for the aggregate bus length specified.

The Electronics Industry Association RS-485 standard was chosen as the electrical and signaling protocol for the command bus. While a mature standard, it provides adequate throughput using a minimum of interconnect wires and supports multiple nodes connected in parallel. In addition, it uses a differential signal representation, minimizing the effects of crosstalk from the data bus. Although it is possible to use a single differential signal pair bidirectionally, the four wire multidrop configuration had several advantages. First, by isolating the master to slave bus and slave to master bus, we can eliminate packet ambiguity (slave receivers don't have to parse other slave to master packets.) Furthermore, a command protocol supporting transaction pipelining or split phase transactions is much more difficult to implement with a common serial channel.

3.2.2 Message layer

The command bus receivers on the slave devices monitor the `HV_TxD` signal pair for the start of a message from the master's transmitter. Upon detecting a new message, all receivers process the first byte of the message to determine if the message applies to them. Those not addressed by the message ignore the remainder, waiting for another message to be sent out. Unambiguous identification of the start of a message is difficult to guarantee. After considering several alternatives, we chose to dedicate one bit of every command word (eight bits) sent to this task. This method is supported by hardware in the Motorola MC68HC11 MCU typically used in the slave devices, and does not rely on slaves correctly parsing every packet sent over the bus, or the use of explicit message synchronization commands. Since most commands are only two to four words long[9] the 12.5% overhead is acceptable (a unique packet start word would require 25% to 50% overhead for most commands.) The command protocol currently supports single phase, unpipelined bus command transactions, which are initiated by the master and followed by a reply from the addressed slave unit.

3.3 Bus slave design

An H-Bus slave unit may be abstracted into four separate functional blocks, interconnected as shown in Figure 2 : the data producer, the packet buffers, the packet consumer, and the bus interface unit (BIU). The data producer is the media data generating mechanism on the slave unit. It may be a device with one channel

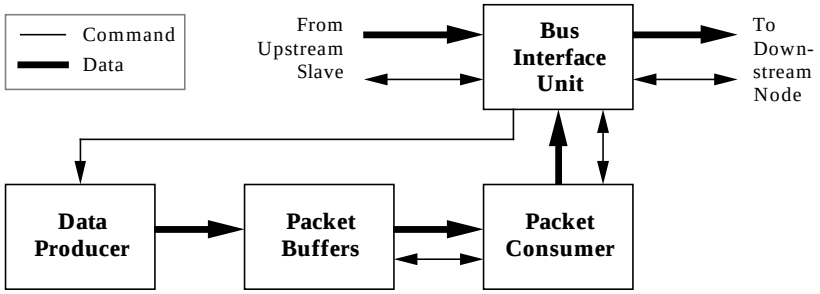

Figure 2: Block Diagram of H-Bus Slave

of output, such as a monochrome video digitizer, or a multichannel device such as an RGB component digitizer with stereo audio input. The producer also contains the slave control unit (typically a small microcontroller) that interfaces to the command bus and interprets the commands. It is assumed that the producer interfaces to the packet buffer using independent write clocks, allowing data production rates which are asynchronous relative to H-Bus interface.

The packet buffers act as a repository for data from the producer between the time when the producer initially generates data and the time when the consumer has successfully arbitrated for the bus and may transfer the packet of data to the bus master. As data is fed into the packet buffers by the producer, the packet consumer is provided with status flags indicating the fullness of the buffer. If an overflow condition occurs (the consumer fails to transfers packets as fast a data is being produced,) the packet buffers are responsible for recovering in a manner which results in the loss of an integral number of packets, preserving the synchronization of the remainder of the stream.

The packet consumer has the task of requesting that the BIU arbiter for the bus whenever a packet of data is available. If multiple virtual channels of data are supported by a slave, independent packets buffers are maintained for each channel, and the consumer monitors all of them for available packets. When the BIU signals that arbitration for the bus was succesful, the consumer generates the header word, then transfers it and the packet data to the BIU along with a clock signal.

The bus interface unit is composed of several functional blocks which together serve three purposes. First, it arbitrates for the bus whenever the consumer determines that a packet should be sent to the bus master. Second, when the slave unit is depositing data onto the H-Bus, the BIU converts the data signals from TTL levels used for internal logic into Bus Transceiver Logic (BTL) levels used on the cable segments between the slave nodes. Third, it interfaces to an upstream link. When an upstream slave device transfers data to the bus master, the BIU reshapes and retimes the data, then transmits it on its downstream link.

3.4 Bus master design

The bus master interfaces the H-Bus to a large shared data store via packet buffers, arbitrating internally for access to the memory with other bus controllers. It extracts the header from each packet received, and uses it to calculate the appropriate buffer location in the data store for the packet. The bus master consists of several components: the bus interface unit (BIU), the packet buffers, the address generator, and the command interface.

The bus interface unit is similar to the upstream bus interface of a slave unit. It receives data, and `HV_CLK` from the closest slave unit. All necessary input signal reshaping and retiming is performed, before the packet header and data are passed to the packet buffers. The interface generates a `REQ` signal, in response to an `ACK`

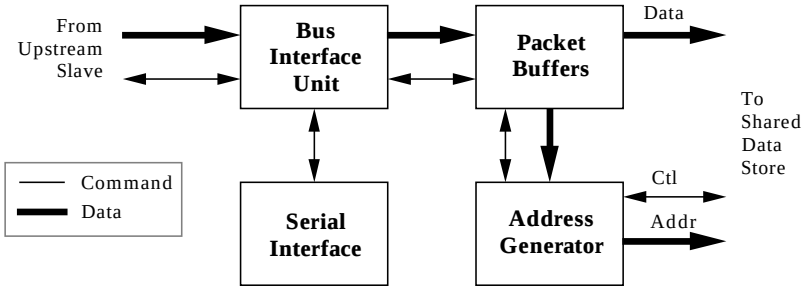

Figure 3: Block Diagram of H-Bus Master

signal passed to it by the nearest slave, when there is room available in the packet buffers for another packet.

An error checking state machine runs a counter tallying the number of payload words received for a packet, verifying that it is indeed 256. If the correct number of words has not been detected when the slave deasserts `REQ`, an error signal is asserted, allowing the address generator to clear the corrupted packet from the packet buffers.

The packet buffer, which is shared by all virtual channels in a system, should be large enough to store several (three or more) packets to effectively decouple the H-Bus data transfers from the shared data store access latency. The packet buffer typically transforms the 16-bit H-Bus data words into a larger word width for local transfer. The data generated by the BIU contains both packet header and data. The 16-bit header information is typically queued separately, where it can be accessed by the address generator.

The address generator examines the header information of a packet, and generates an appropriate destination address in the shared data store. Using the bus parameters specified above, a new packet address may be needed every 6.8 μs. The capabilities of the address generator may vary, but they allow the storage of packet data pertaining to each virtual channel to be stored in a separate (and possibly circular) buffer in the shared data store. A means of informing application software of the availability of new data should be provided, usually upon the arrival of a new "frame" of packets.

The command bus is driven by a general purpose asynchronous interface (UART) under the control of a software driver on the host providing an interface for applications.

4 IMPLEMENTATION

An implementation of the H-Bus has been built, and is being tested. The Cheops image processing system[1] has as one of its component modules a large (up to 32 Gbit) solid-state shared data store (M1). In addition to the Cheops system interconnects (allowing up to 2 Gbit/s of data I/O,) M1 supports up to 3.2 Gbit/s of I/O through a daughtercard interface. A video digitizer card (I1) for this daughtercard interface includes two H-Bus masters, which allow up to sixteen additional input units to shared the high-speed M1 daughtercard port. To date the only H-Bus slaves which have been constructed are test units.

4.1 I1 H-Bus master

The two H-Bus interfaces on a Cheops I1 Video input card each have a separate bus interface unit and packet buffers, but share a command bus interface and the address generator. The packet buffers generate the 128-bit

wide words required by the M1 shared data store interface. Enough memory is provided in the packet buffers (built of 64x18 FIFOs) for three data packets. Separate registers are provided for storing the packet headers.

The address generator is shared with the onboard digitizer as well as both H-Bus interfaces, requiring it to provide up to one destination address per 2 μs, and support over 500 virtual channels. While a custom design was contemplated, a programmable solution was determined to be cheaper and quicker to implement, due to the complexity of the buffer addressing algorithms desired.

An Intel i960CA 32-bit RISC microprocessor was dedicated to the task of constantly polling for newly arrived packets, parsing the packet headers and generating a destination address for each packet. It uses 1 Mbit of SRAM for both code and channel state storage. A message passing mechanism (using the shared SRAM) is used for communication with application software running on the Cheops system. The address generator supports fragmented circular buffers for storage of virtual channel data, queueing of digitization request and automatic temporal decimation of video data.

4.2 Bus slave

A test slave has been designed and is being built in order to allow testing the bus under controlled conditions. The test slave uses a small amount (2 Mbits) of pattern memory instead of a video or audio A/D to provide data for bus transfer. The pattern memory may be loaded with a particular data sequence by a local microcontroller in response to commands transmitted over the H-Bus. A single test slave is capable of saturating the data bus throughput, as well as sourcing all virtual channels.

5 CONCLUSIONS

Building systems capable of simultaneously acquiring and manipulating multiple media streams is still a difficult problem. While several industry bus standards are beginning to address the problems of real-time media distribution, they either do not provide the throughput required for acquisition of multiple sources using a single bus, or require expensive, complex bus interfaces. We present the H-Bus as an example of the functionality which may be provided using common components and an alternative approach specifically targeting the needs of multimedia acquisition.

6 ACKNOWLEDGMENTS

Components of this research have been sponsored by the Television of Tomorrow research consortium of the Media Laboratory, MIT.

7 REFERENCES

[1] V. M. Bove, Jr., J. A. Watlington. "Cheops: A Reconfigurable Data-Flow System for Video Processing," to be published in IEEE Trans. on Circuits and Systems for Video Technology, 1995. Available at `http://wad.www.media.mit.edu/people/wad/cheops_CSVT/cheops.html`

[2] David James, David Gustavson. "Draft Proposals for Real-Time Transactions on SCI." SCI/RT-1995.

[3] Saied Hosseini-Khayat, Andreas D. Bovopoulos, "A simple and efficient bus management scheme that supports continuous streams." ACM Transactions on Computer Systems, Vol.13, No. 2, May 1995, pp. 122-140

[4] Adam J. Kunzman, Alan T. Wetzel. "1394 High Performance Serial Bus: The Digital Interface for ATV," *IEEE Trans. on Consumer Electronics*, August 1995, Vol. 14, No. 13, pp. 893-900.

[5] Mark Nossokoff and Gene Freeman. "SCSI R.I.P. - NOT! The Case for FAST-20 SCSI." Computer Technology Review, April 1995. Also available at `http://www.symbios.com/articles/ctr_495.htm`

[6] Small Computer System Interface - 2, X3.131-199x, ANSI.

[7] Michael Teener. "New Technology in the IEEE P1394 Serial Bus - Making it Fast, Cheap, and Easy to Use," presented at the Hot Interconnects Symposium '93, Stanford University, Aug. 6, 1993.

[8] Ivan Tving. "Multiprocessor interconnection using SCI." Masters Thesis, Technical University of Denmark, 1994. Available at `ftp://ftp.SCIzzL.com/u/SCIzzL/sci/latest_pdf/TvingThesis.pdf`

[9] Jeffrey J. Wong. "Hoover Bus: An Input Bus Optimized for Multiple Real-Time Data Streams." MEng Thesis, Massachusetts Institute of Technology, Sept. 1996.